
spatial tree
algorithms

learning objectives

algorithms

your software

system software

hardware

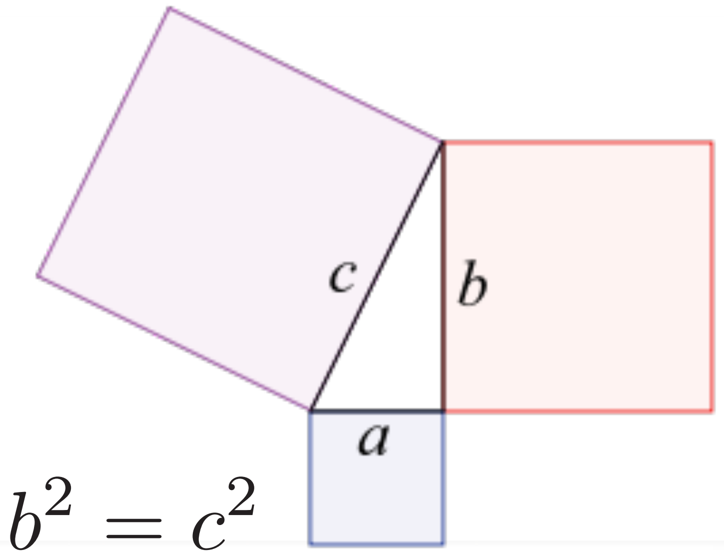
- learn the characteristics of spatial data
- learn several spatial indexing data structures
- learn basic algorithms for using such structures

computational geometry

a branch of computer science focusing on **data structures & algorithms** for solving **geometric problems**

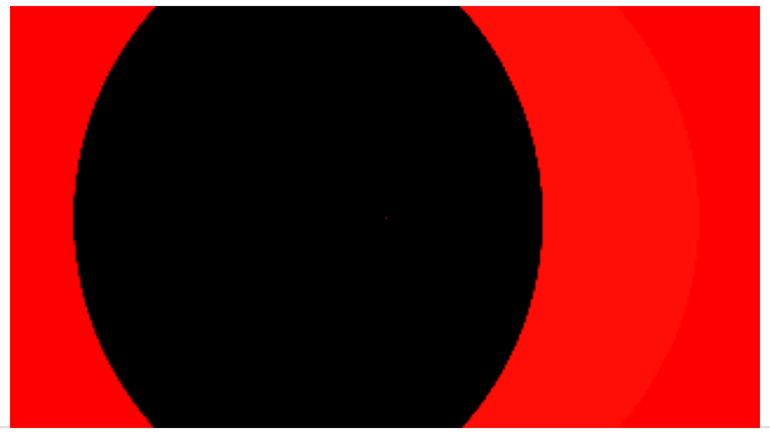
development made possible by **exponential progress in computer graphics**, with multiple applications

mathematical visualization, e.g., proofwithout words, mandelbrot sets, etc.



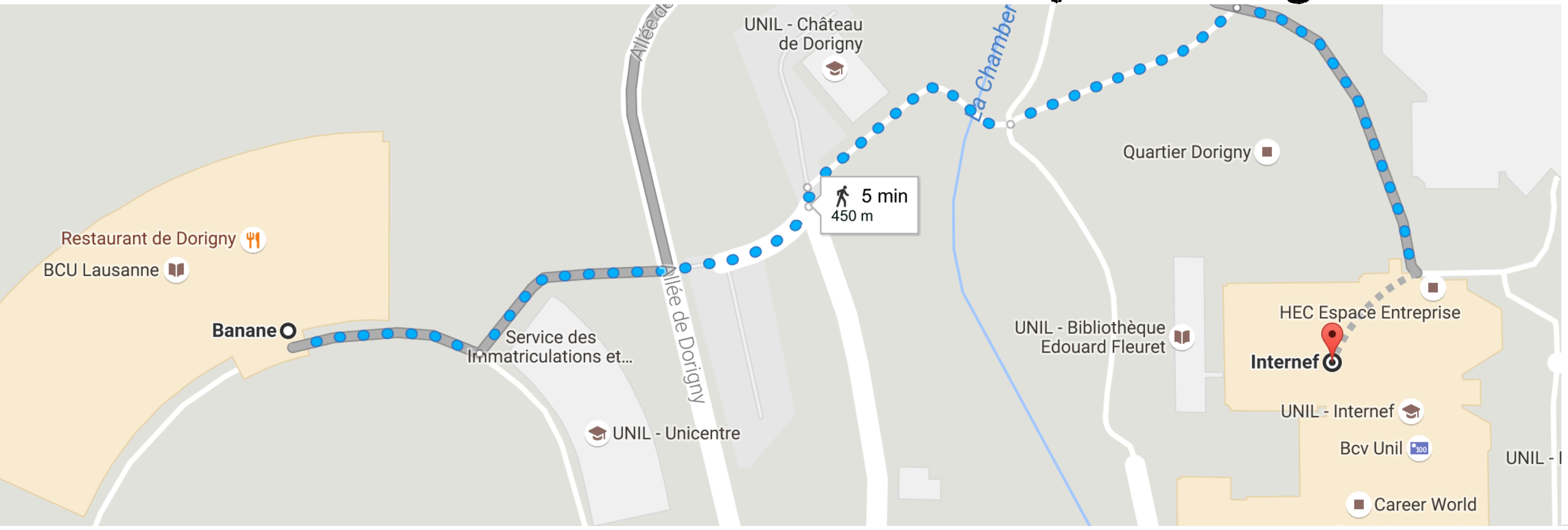
$a^2 + b^2 = c^2$

$z \mapsto z^d + c$



computer vision e.g., 3D graphics in games

geographic information systems, e.g., location search & route planning



computer-aided engineering, e.g., mechanical design

computational geometry

what's specific to spatial data?

with 1-dimensional data, natural ordering implicitly partitions the data, e.g., binary tree

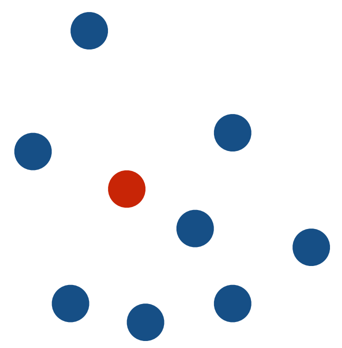
spatial data is intrinsically multidimensional, so there is no natural ordering of data (e.g., of points)

with 1-dimensional data, the static case is rather simple and solved by sorting the data

with multidimensional data, the static case is far from simple and solved by several partitioning techniques

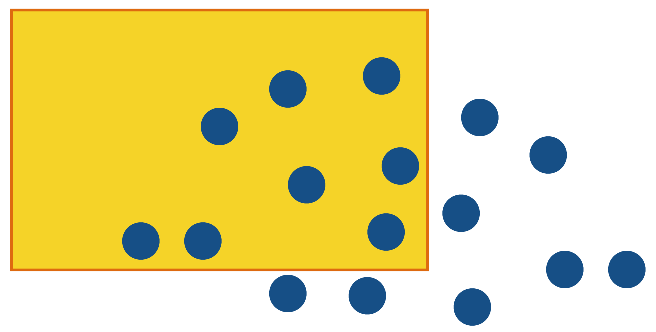
computational geometry

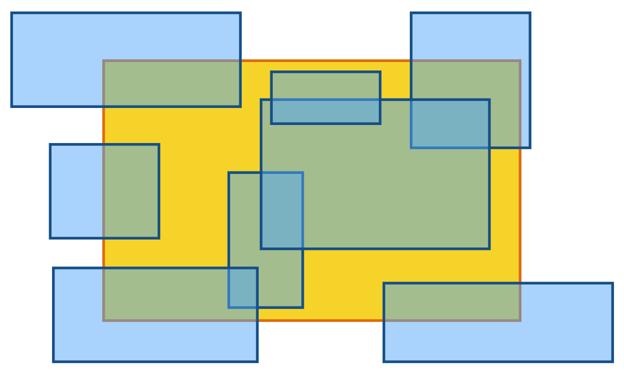
typical problems



nearest neighbor: given a set of points P, find which one is closest to a target point p_t

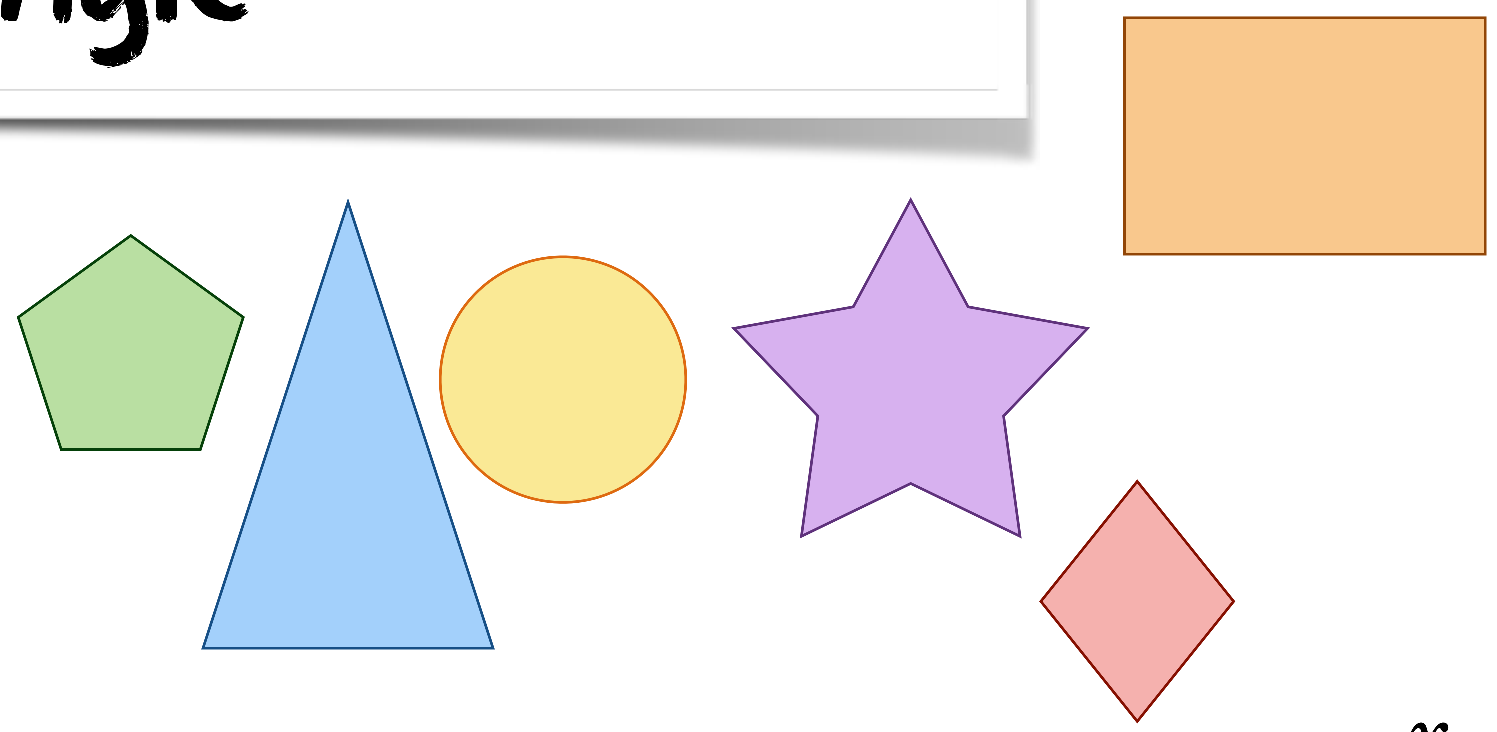
range queries: given a set of points P, find the points contained within a given rectangle



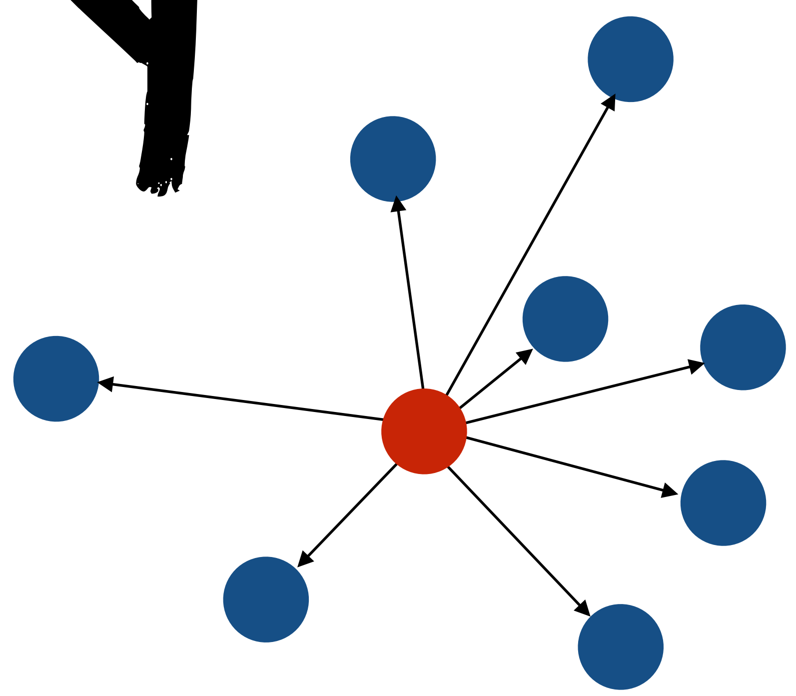


intersection queries: given a set of rectangles R, find which rectangles intersect a target rectangle

collision detection: given a set of shapes S, find the intersections between all these shapes



computational geometry
typical approaches



brute-force algorithm

nearest neighbor: given a set of points P, find which one is closest to a target point p_t

Complexity: $O(n)$, with $n = |P|$

$\textsc{Nearest–neighbor}\ (P, p_t)$

$p \leftarrow \text{NIL}$

$min \leftarrow \infty$

for each $p_i \in P$

if $distance(p_i, p_t) < min$

$min \leftarrow distance(p_i, p_t)$

$p \leftarrow p_i$

return (p, min)

---

spatial tree structures

they index spatial objects

Complexity: $O(\log n)$, with $n = |P|$

R-trees

quad-trees

kd-trees

R-tree

A. Guttman. ***R-trees: A dynamic index structure for spatial searching***. In Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, pages 47–57, New York, NY, USA, 1984. ACM.

a recursive tree, where each node has between M and $m = \left\lfloor \frac{M}{2} \right\rfloor$ children, except for the root which has at least two

only leaf nodes contain actual spatial object entries, each consisting of the spatial object itself and a minimum bounding region (mbr) containing that object, i.e., $object = (shape, mbr)$

internal nodes contain children entries, each consisting of a link to the child node and an mbr covering all children nodes of that child, i.e., $node = (child, mbr)$

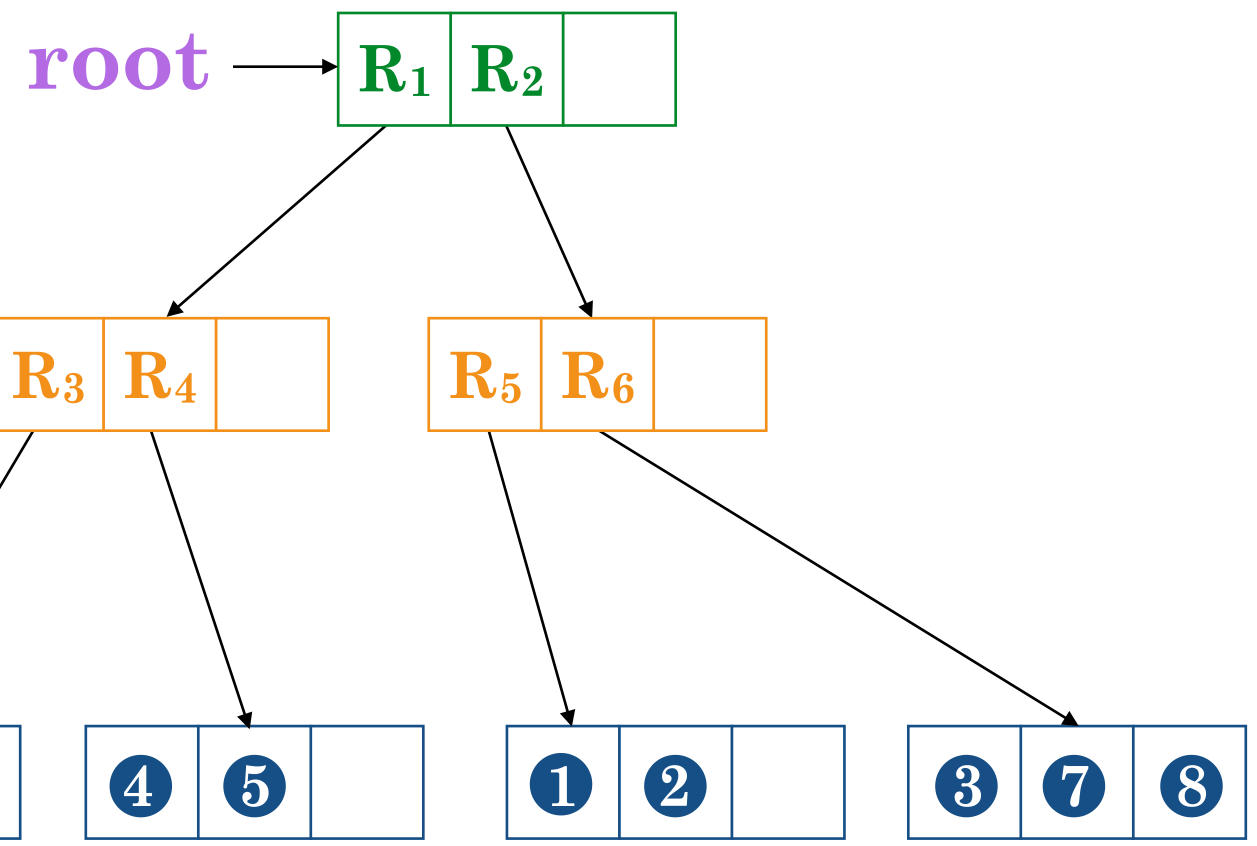
an minimum bounding region is typically of the form $mbr = (x_{min}, y_{min}, x_{max}, y_{max})$

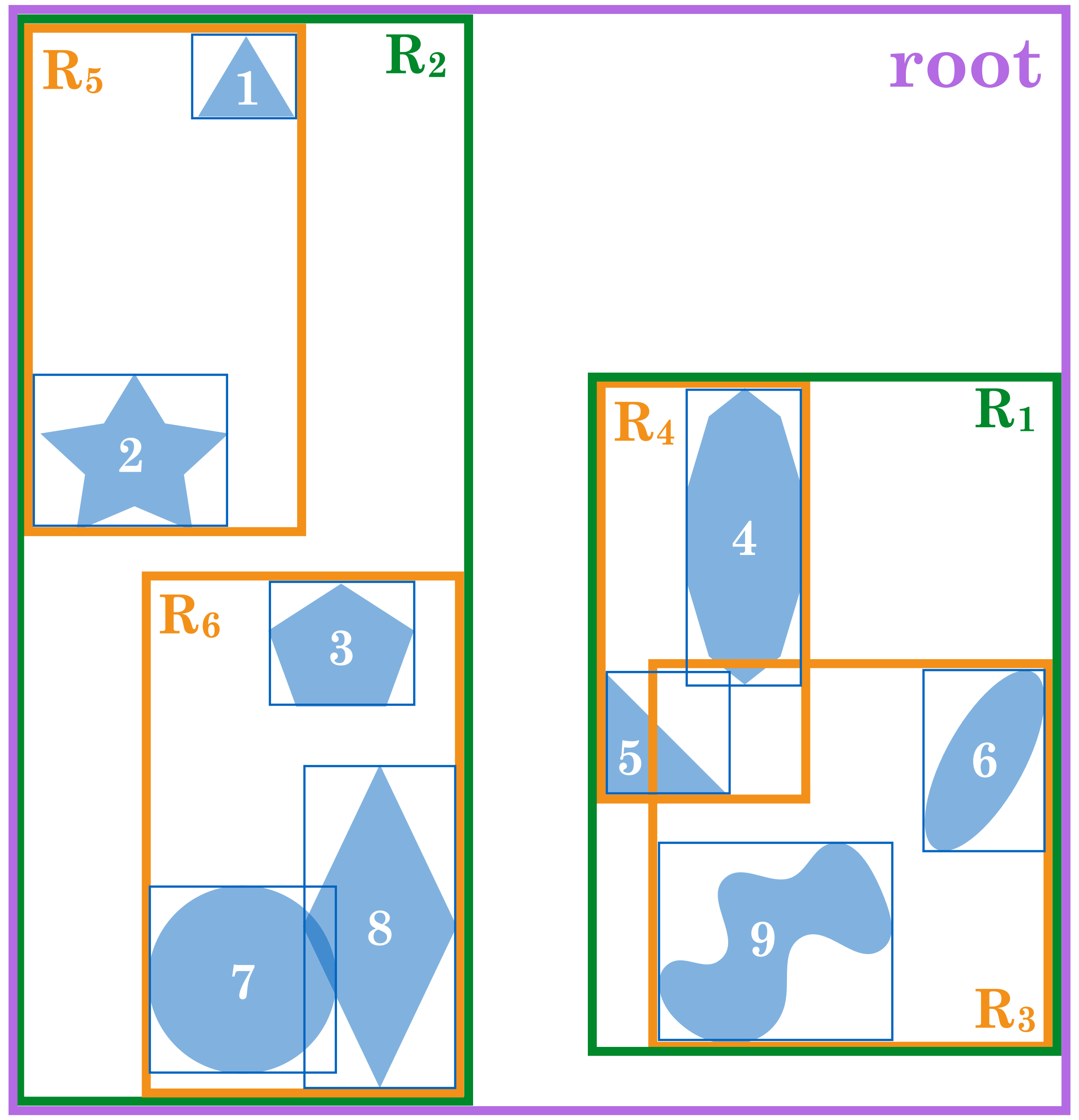
all leaves are at the same level, i.e., the tree is height balanced

R-tree

only leaf nodes contain actual spatial object entries, each consisting of the spatial object itself and a minimum bounding region (mbr) containing that object, i.e., $object = (shape, mbr)$

internal nodes contain children entries, each consisting of a link to the child node and an mbr covering all children nodes of that child, i.e., $node = (child, mbr)$

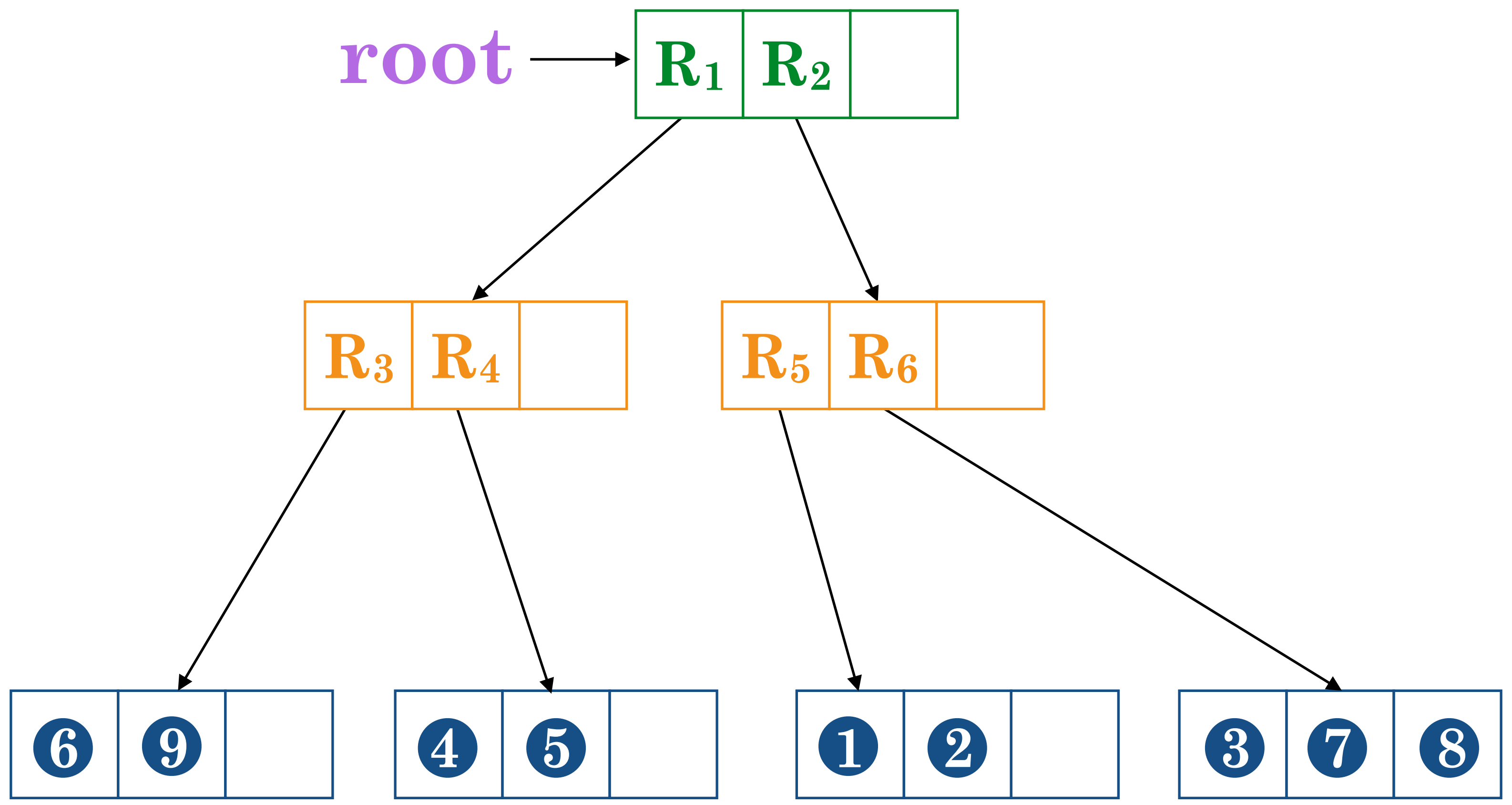




important: the root also contains a minimum bounding box

R-tree

INTERSECT (*node, region*)
  if $node.mbr \subset region$
    return $\{\, object \mid object \in$ REACHABLE-LEAVES($node$) $\}$
  if *node is a leaf*
    return $\{\, object \in node \mid object.mbr \cap region \neq \varnothing \,\}$
  $result \longleftarrow \varnothing$
  for each $kid \in node.children$
    if $kid.mbr \cap \text{region} \neq \varnothing$
      $result = result \cup$ INTERSECT ($kid.child$, region)
  return *result*



SEARCH (*node, shape*)
  if *node is a leaf*
    if $\exists\, object \in node : object.shape = shape$
      return *object*
    return NIL
  for each $kid \in node.children$
    if $shape.mbr \subseteq kid.mbr$
      return SEARCH($kid.child$, $shape$)
  return NIL

important: the root also contains a minimum bounding box

quad-tree

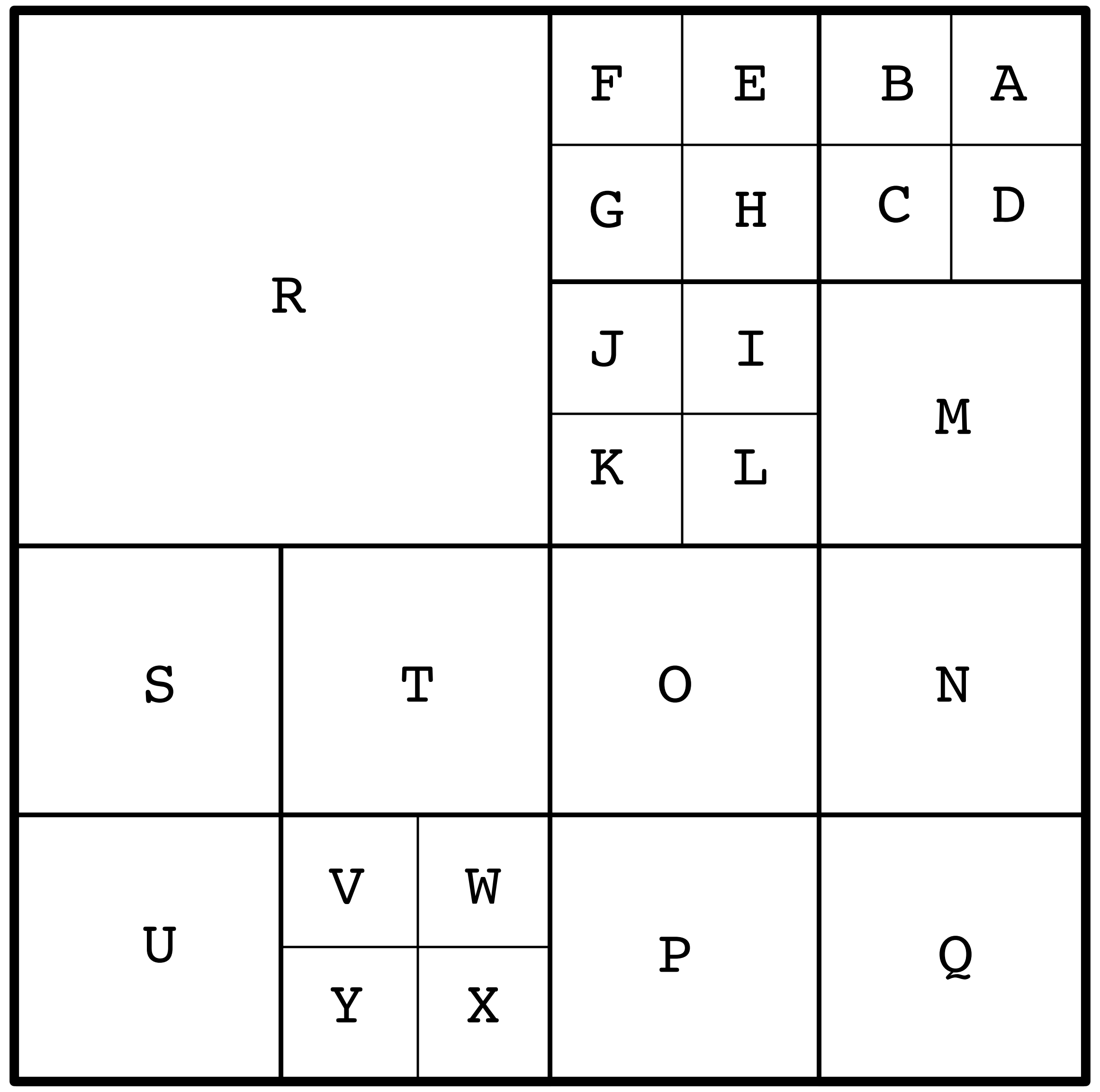
R. A. Finkel and J. L. Bentley. ***Quad trees a data structure for retrieval on composite keys.*** Acta Informatica, 4(1):1–9, 1974.

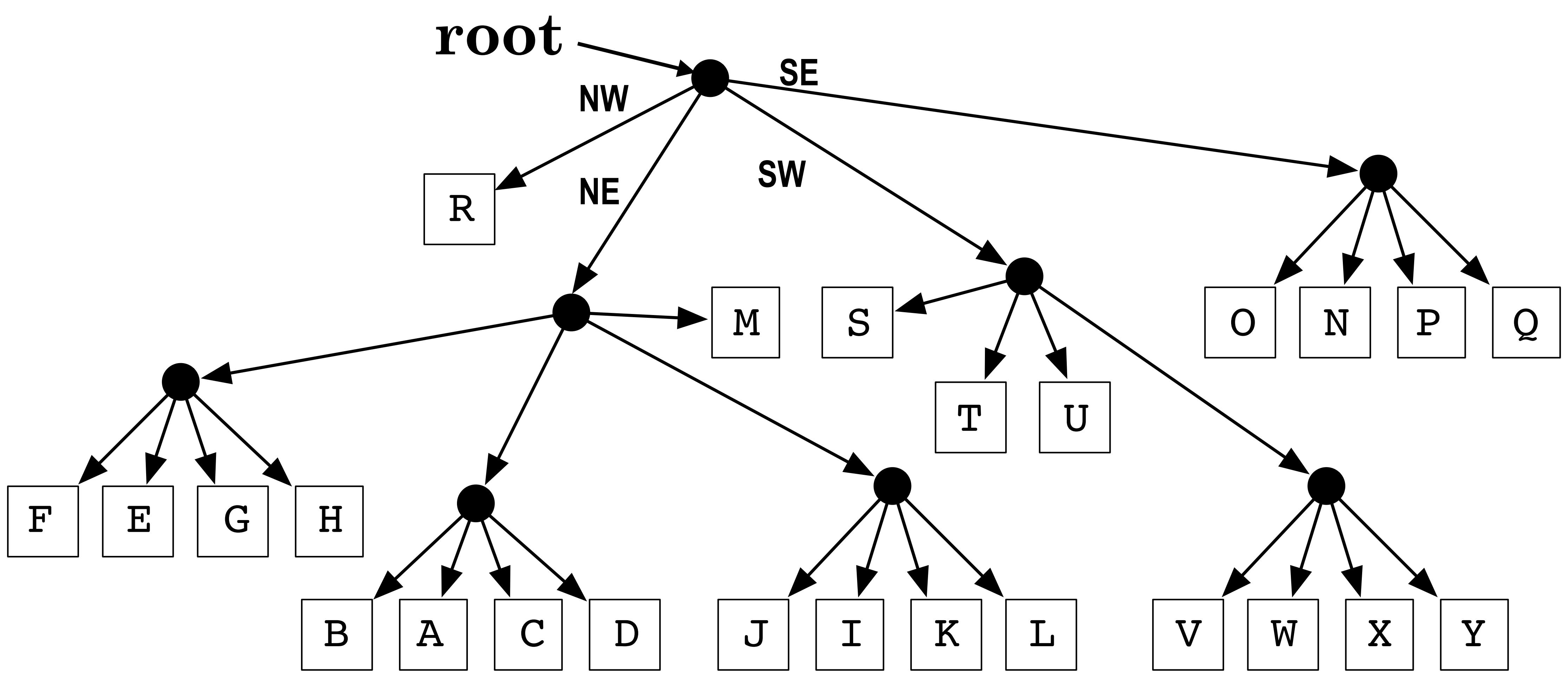
a recursive tree where each internal node has four children

each node represents a cell in the geometrical space, with its children partitioning that cell into an equally sized subcell

predefined partitioning with subcells (quadrants) named as North West (NW), North-East (NE), South-West (SW) and South-East (SE)

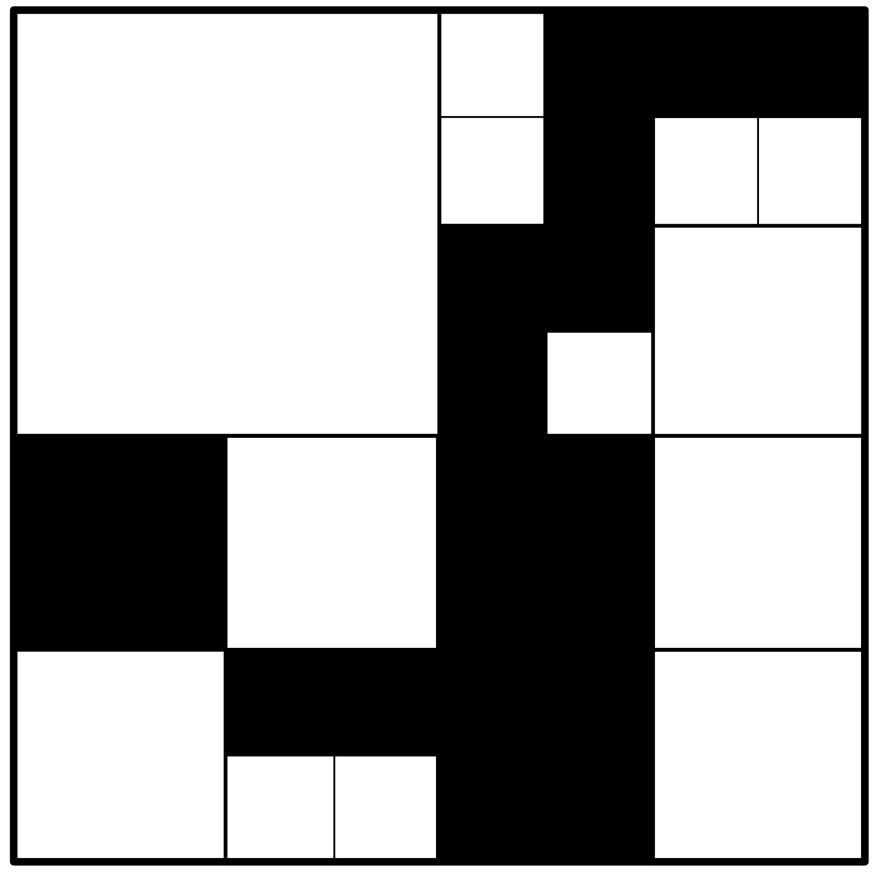
like R-trees, only leaf nodes store actual geometrical objects

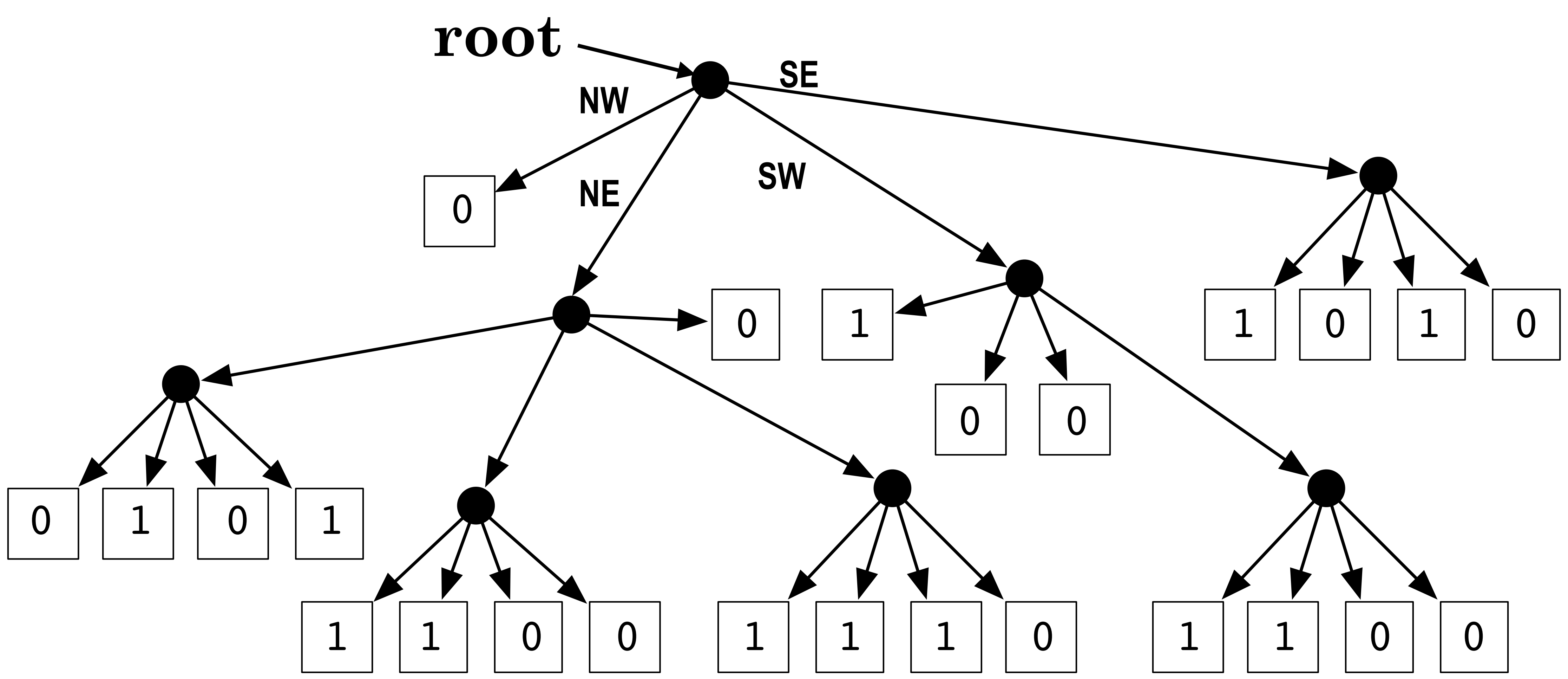




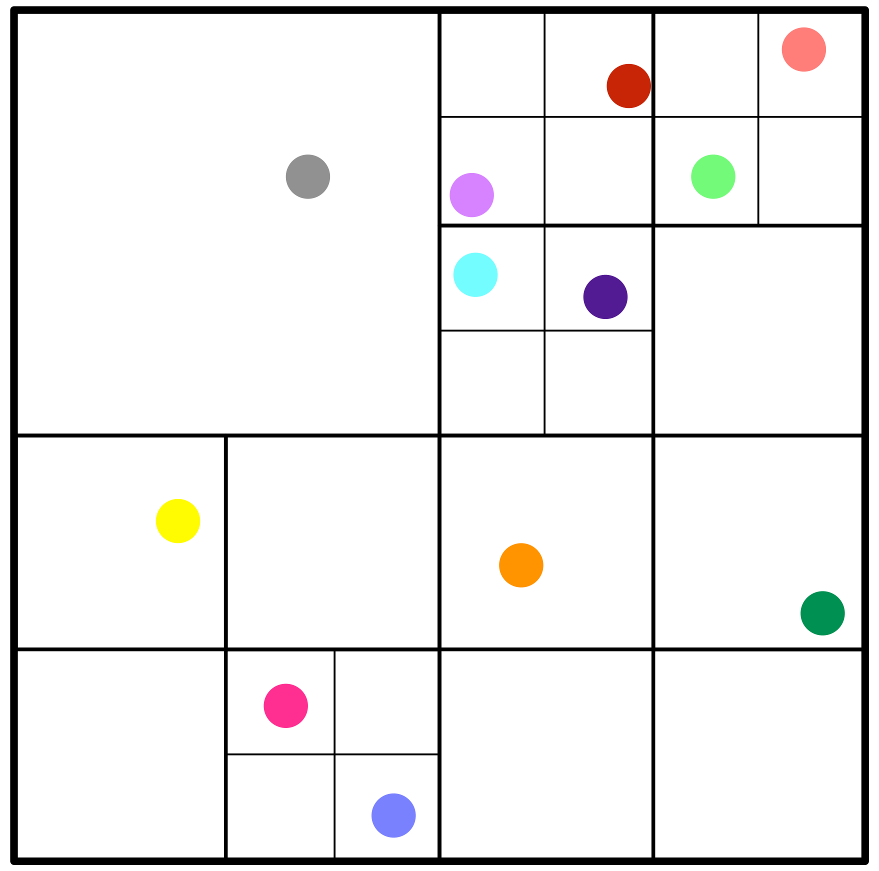
quad-tree

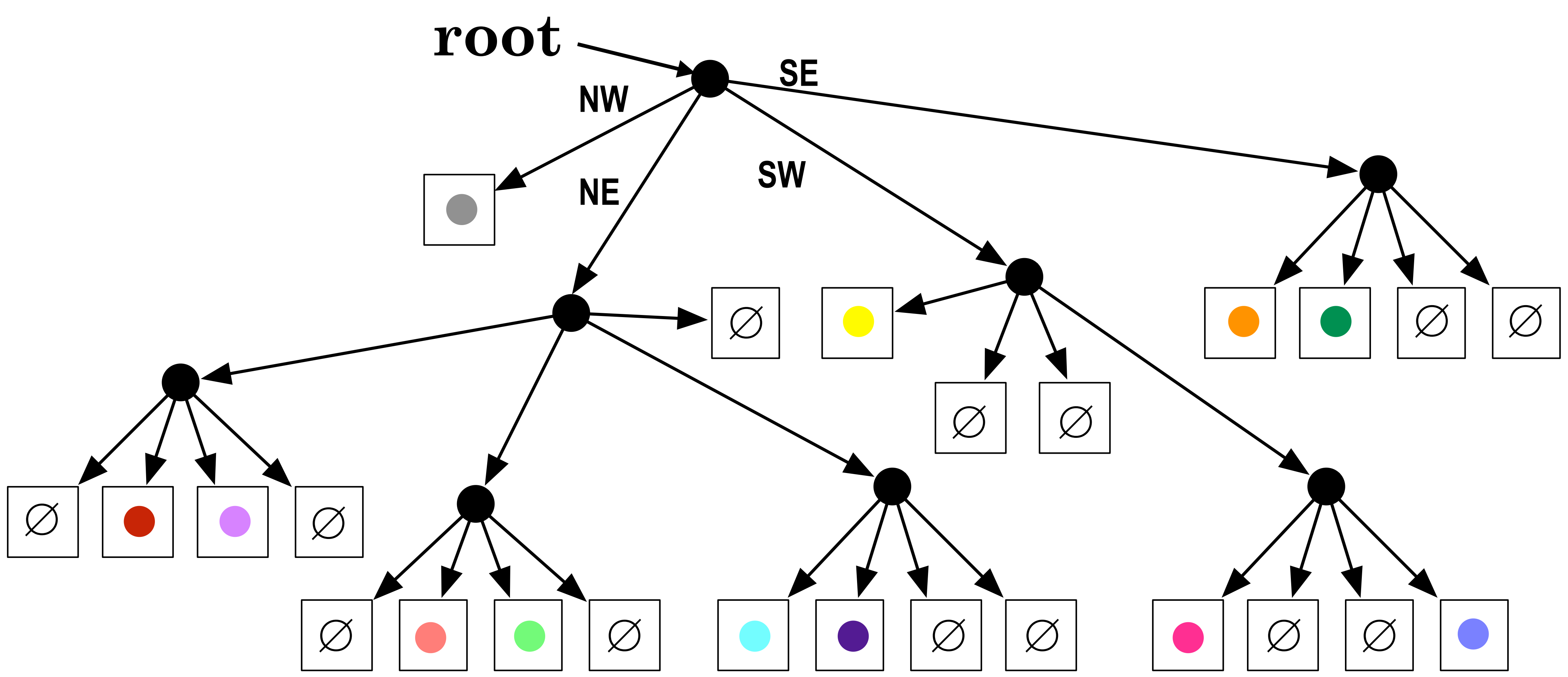
region quad-tree





point-region quad-tree





quad-tree

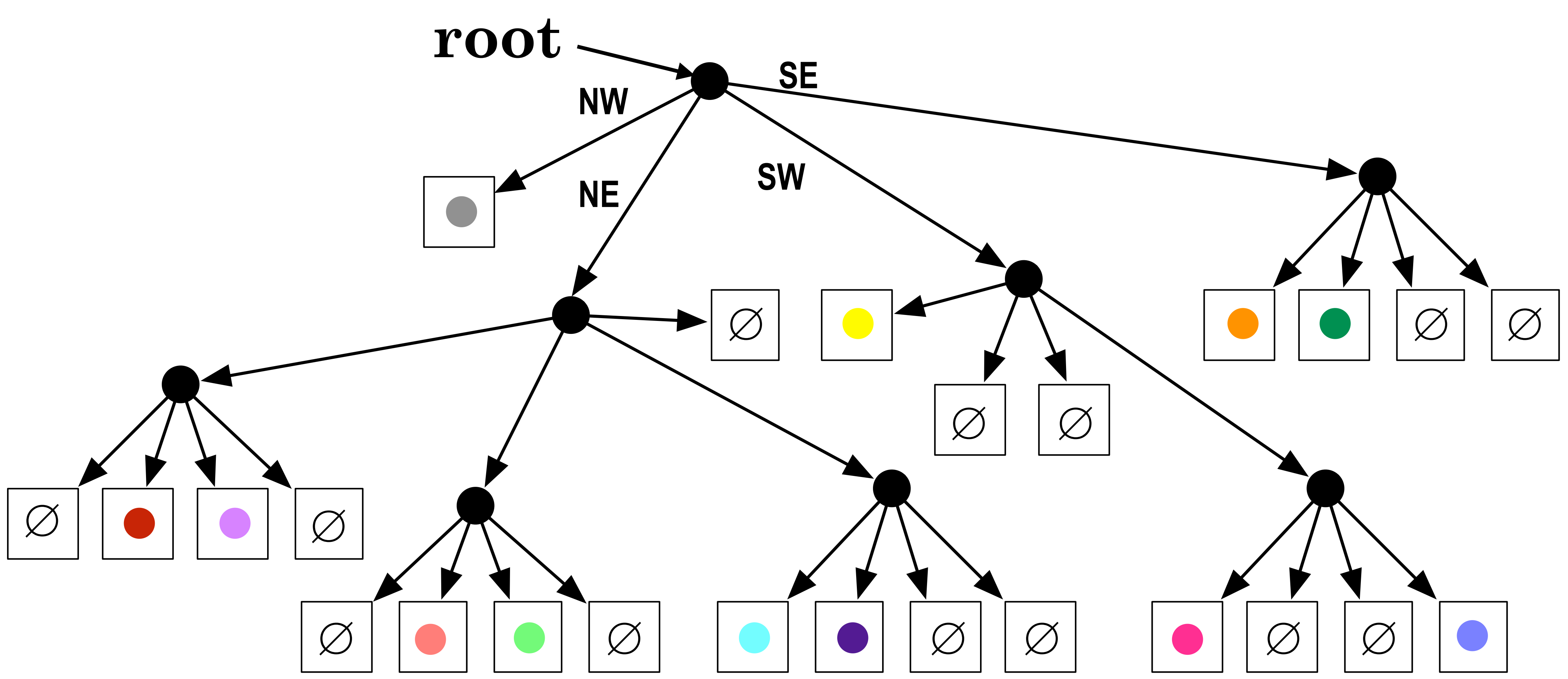
```
ADD (node, point)
    if point ∉ node.cell
        return FALSE
    if node is a leaf
        if node.point = point
            return FALSE
        if node.point = NIL
            node.point ⟵ point
            return TRUE

    quadrant ⟵ FIND-QUADRANT(node, point)
    if node is a leaf
        SUBDIVIDE(node)
    return ADD (node[quadrant], point)
```

```
INTERSECT (node, region)
    if node is a leaf
        if node.point ∈ region  return { node.point }
        return ∅

    if node.cell ⊂ region
        return { node.point | node ∈ REACHABLE-LEAVES(node) }

    result ⟵ ∅
    for each quadrant ∈ { NW, NE, SW, SE }
        if node[quadrant].cell ∩ region ≠ ∅
            result = result ∪ INTERSECT (node[quadrant], region)
    return result
```



kd-tree

J.L. Bentley. ***Multidimensional binary search trees used for associative searching***. Commun. ACM, 18(9):509–517, September 1975.

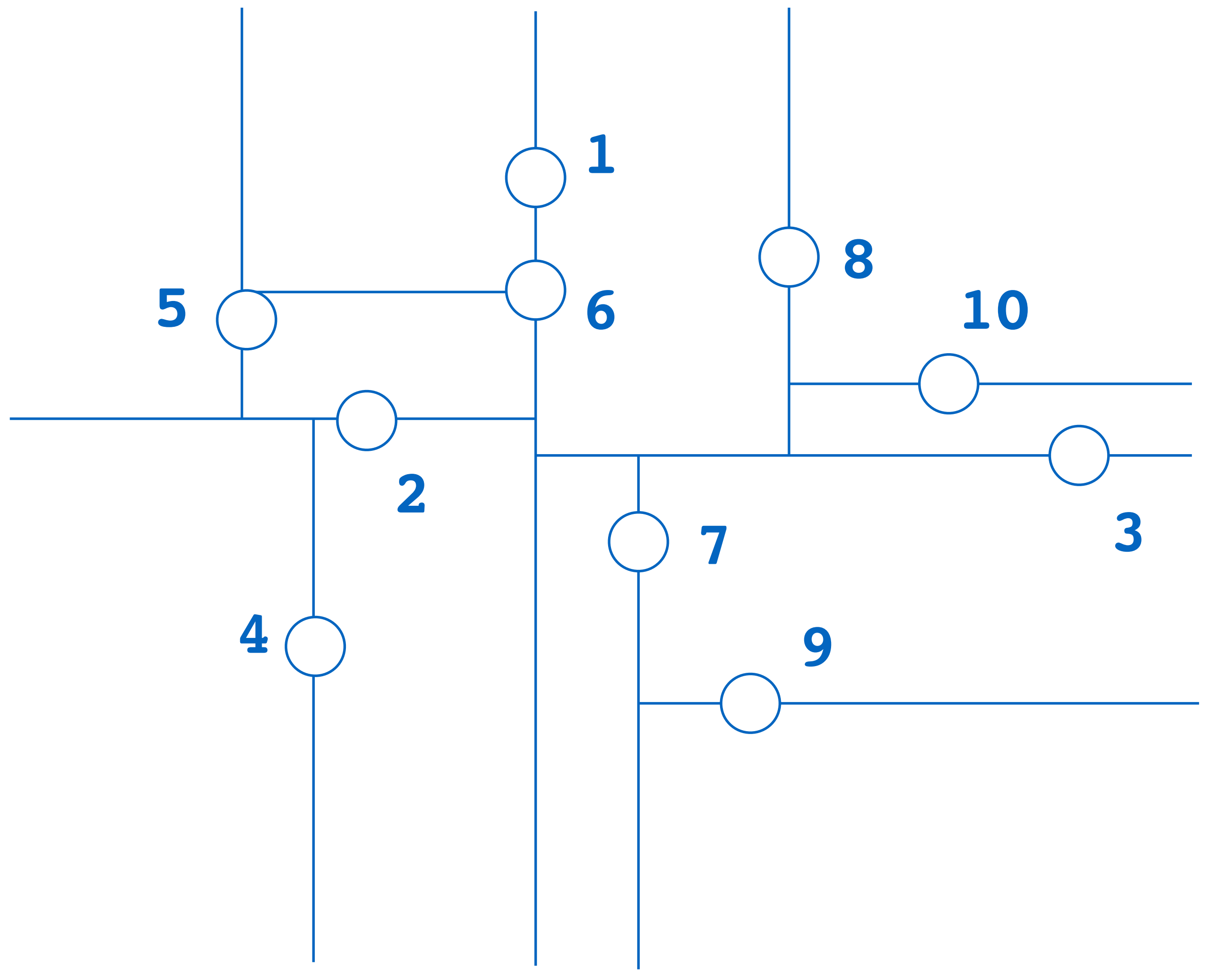
a kd-tree (short for k-dimensional tree) is a binary tree in which every node is a k-dimensional point

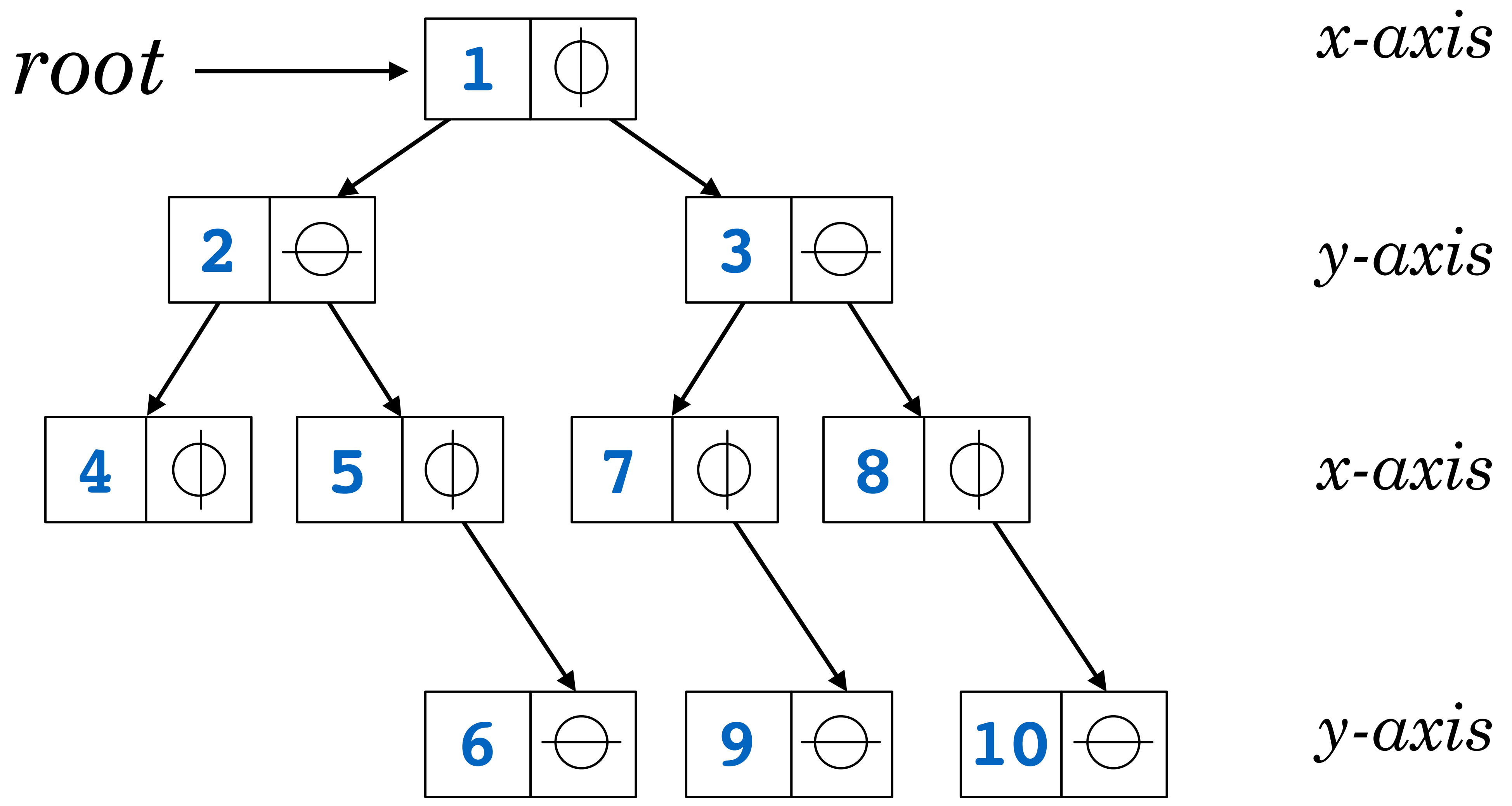
in addition, each internal node divides the k-dimensional space into two parts known as half-spaces

all points in one half space are contained in the left subtree of the node and all points in the other half space contained in the right subtree

all nodes at the same level (height) divide the k-dimensional space according to the same cutting dimension (axis)

k-d-tree





Remarks

- *Points are stored as k-dimensional arrays*
- *Each axis corresponds to an index:*
  - *x-axis corresponds to index* 0
  - *y-axis corresponds to index* 1
  - etc…
- *So assuming point* $p_i = (x_i,y_i) = (3,7)$*, we have that* $p_i = [3,7]$, $x_i = p[0] = 7$ *and* $y_i = p[1] = 7$
- In this example, initially *root* = **NIL** and points are inserted as follows:
  - ADD($root$,p_1, 0)
  - ADD($root$,p_2, 0)
  - ADD($root$,p_3, 0)
  - etc…

```
ADD (node, point, cutaxis)
    if node = NIL
        node ← CREATE-NODE
        node.point = point
        return node
    if point[cutaxis] ≤ node.point[cutaxis]
        node.left = ADD(node.left, point, (cutaxis + 1) mod k)
    else
        node.right = ADD(node.right, point, (cutaxis + 1) mod k)
    return node
```