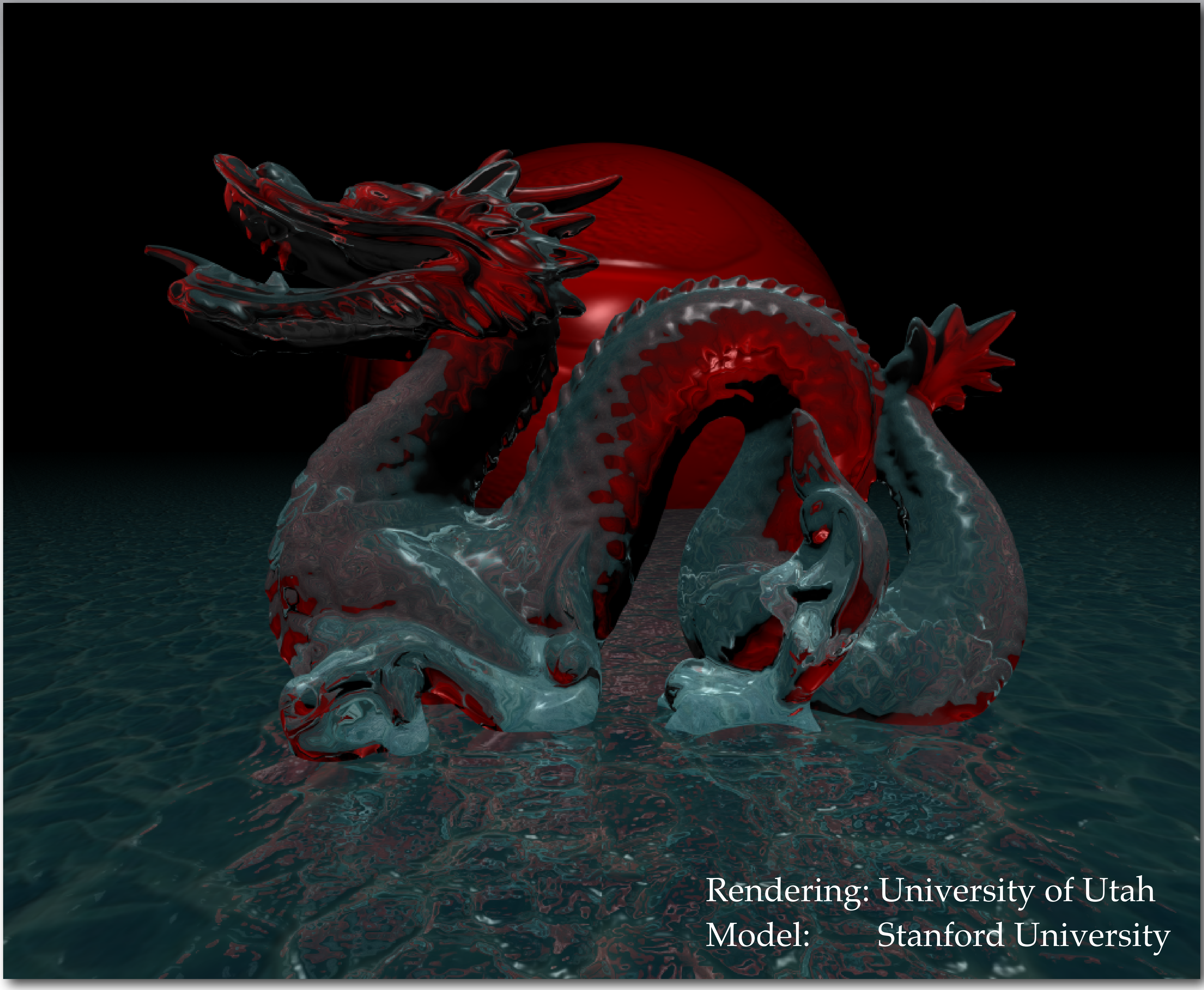
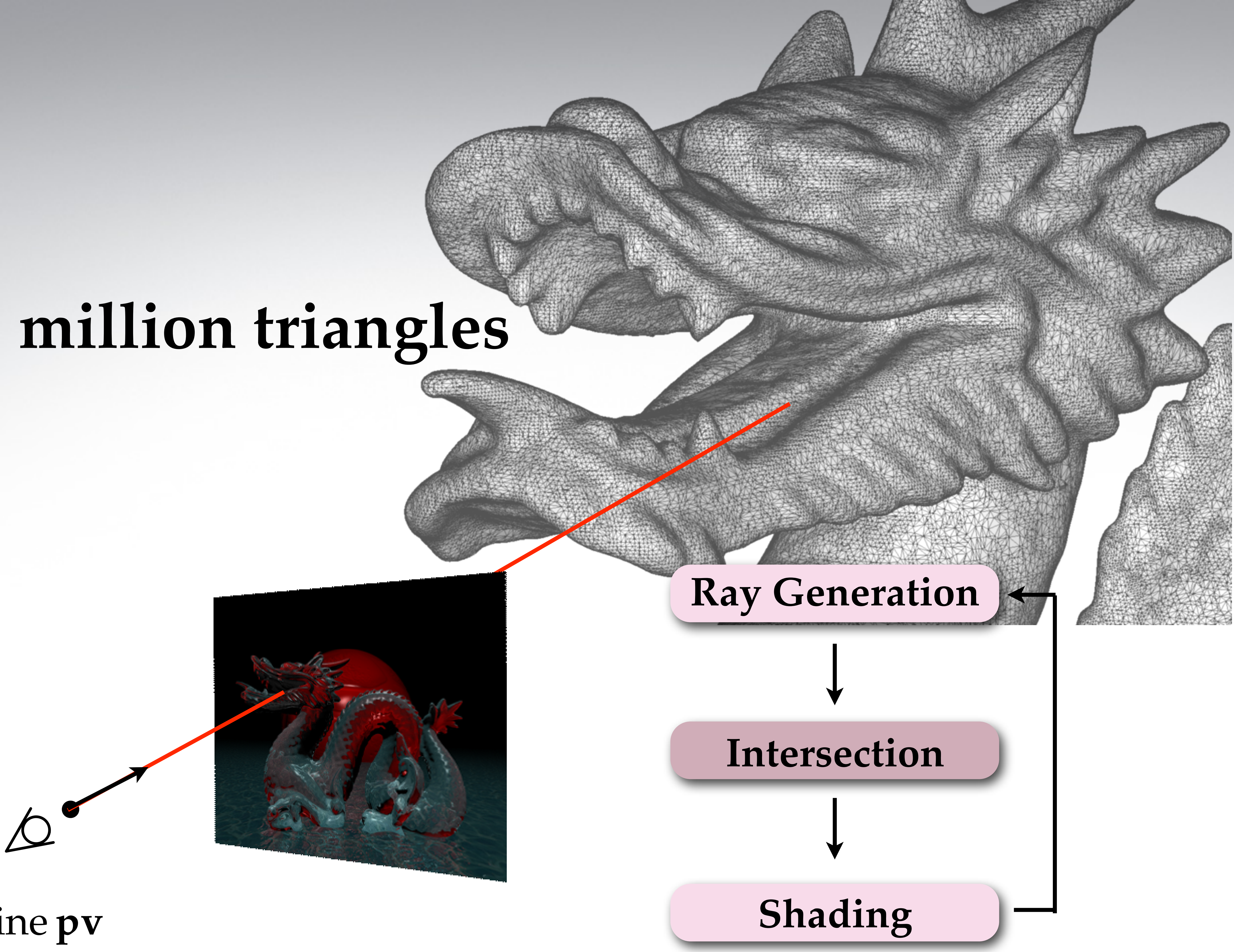
Scans: XYZRGB

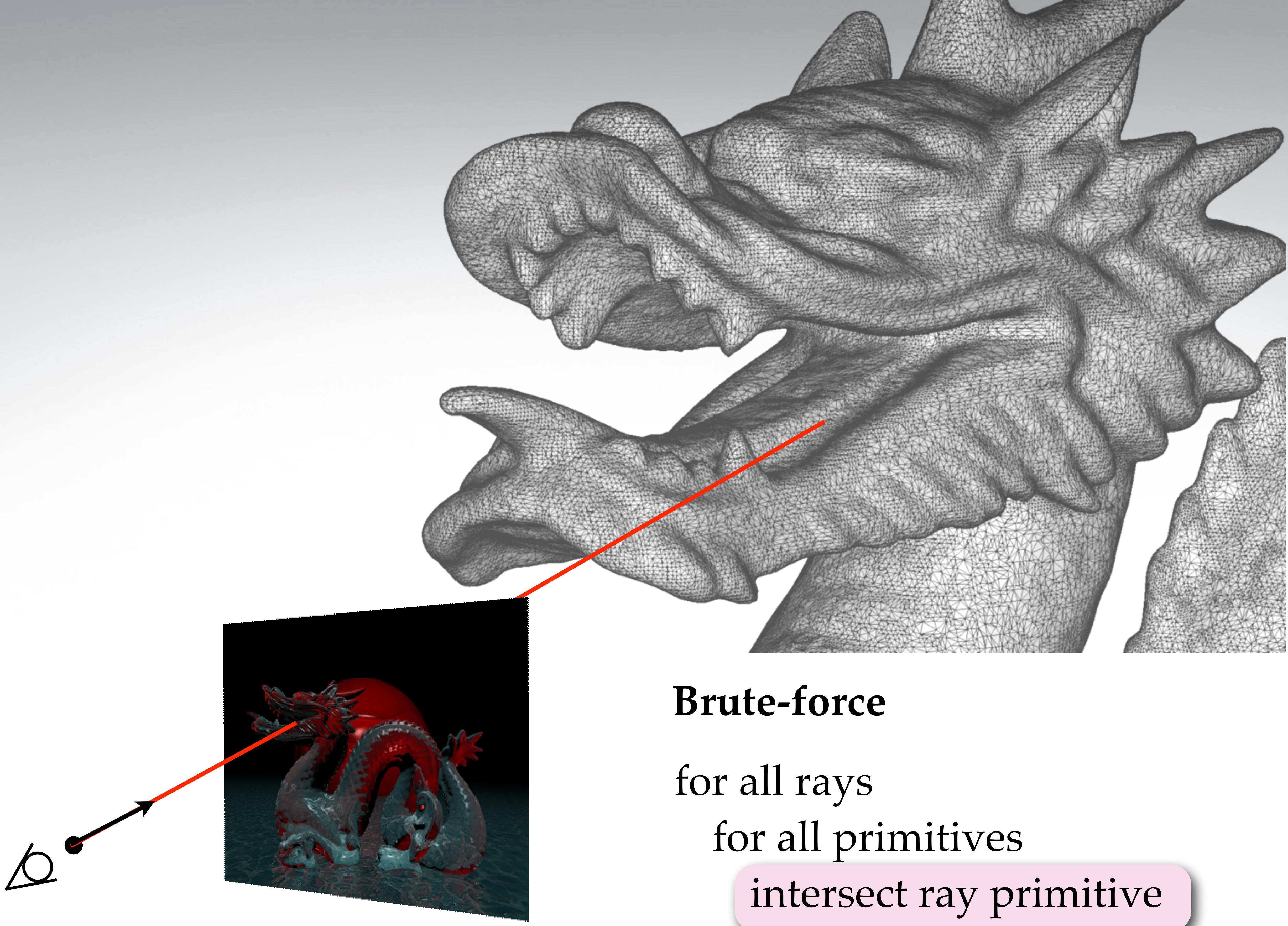


Rendering: University of Utah
Model: Stanford University

5.5 million triangles



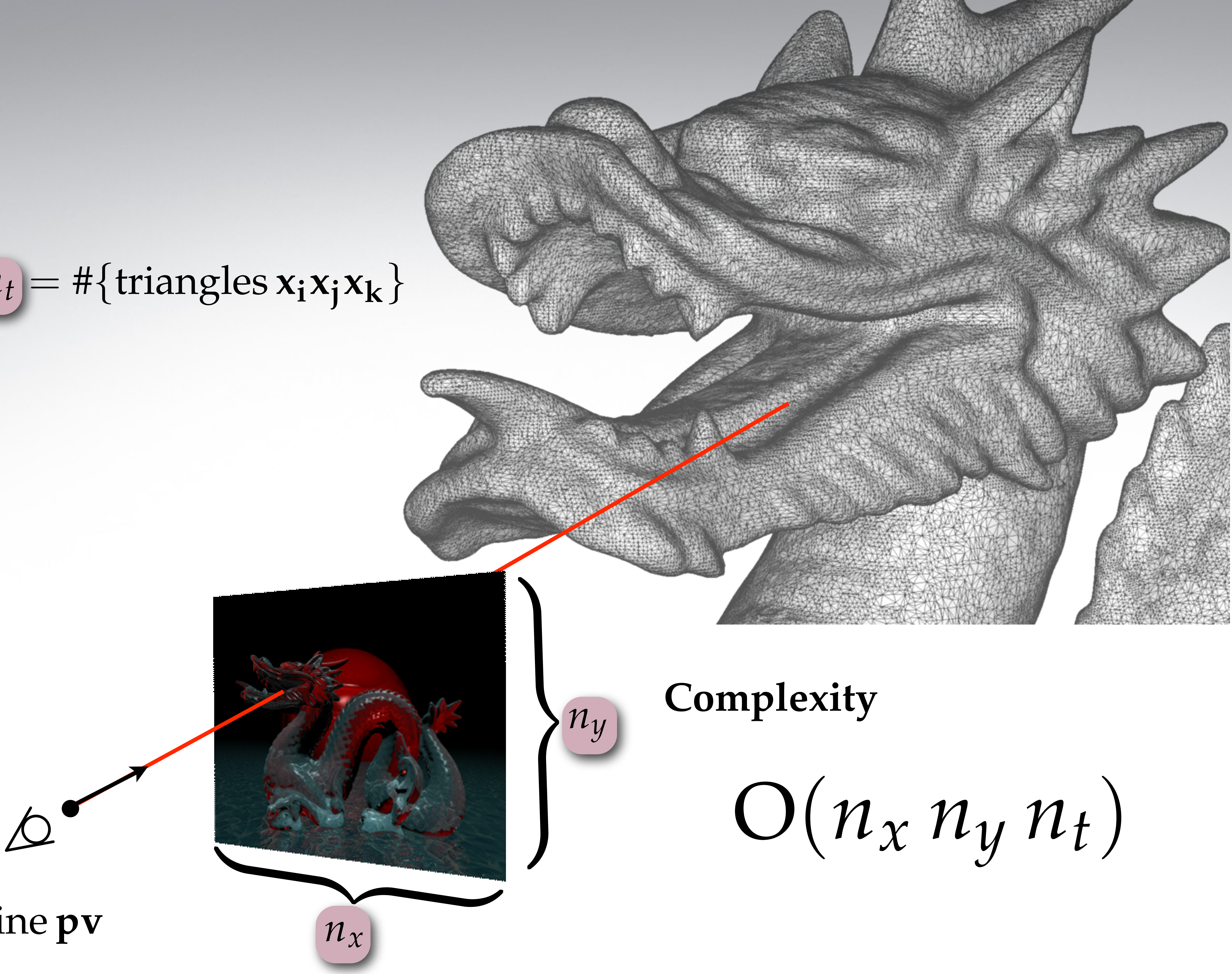
half-line **pv**



Brute-force

```
for all rays
    for all primitives
        intersect ray primitive
    return closest intersection
```

$n_t = \#\{\text{triangles } \mathbf{x_i x_j x_k}\}$



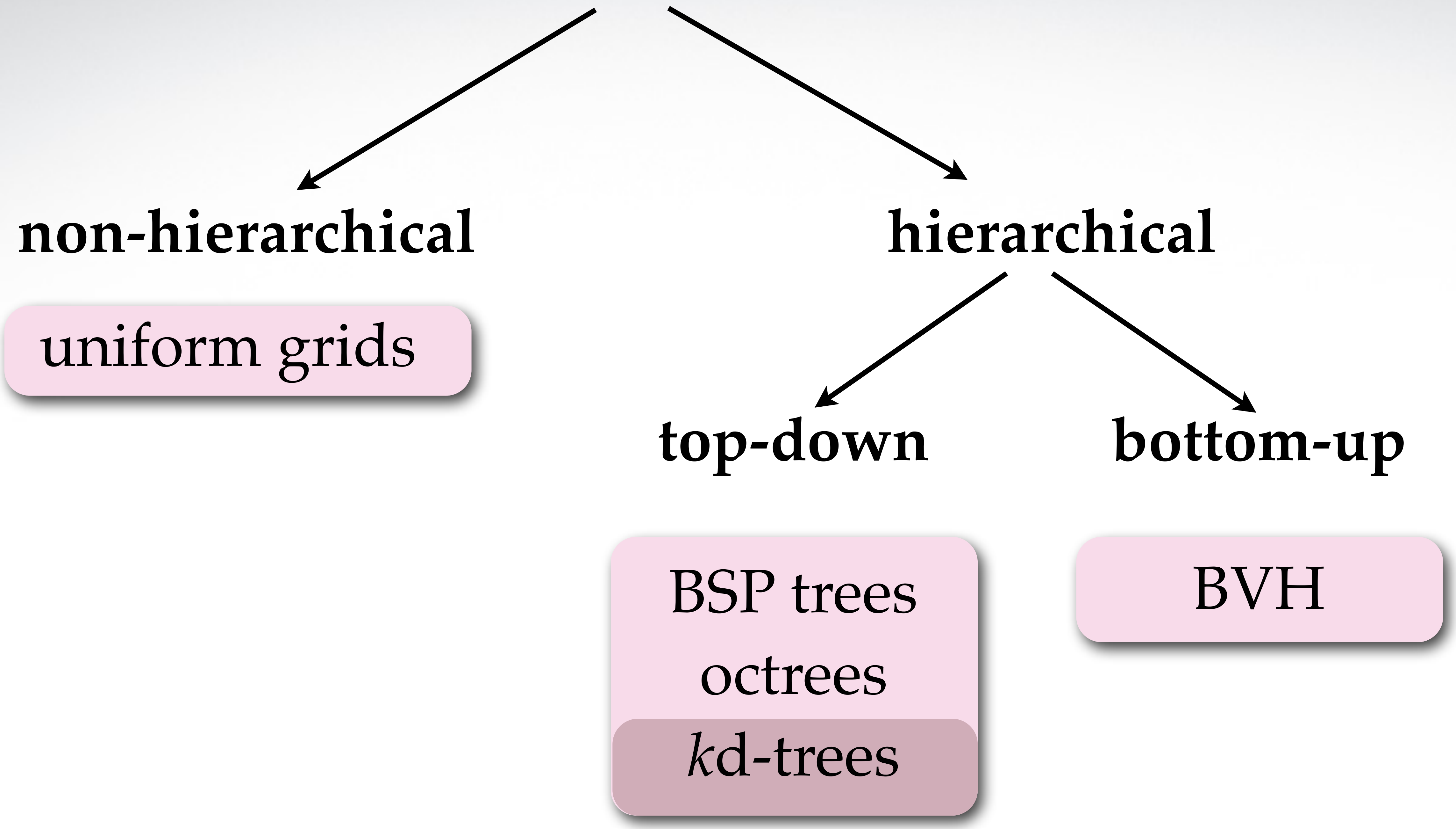
Complexity

$$O(n_x\, n_y\, n_t)$$

“95 % computing time is spent on intersection computations”

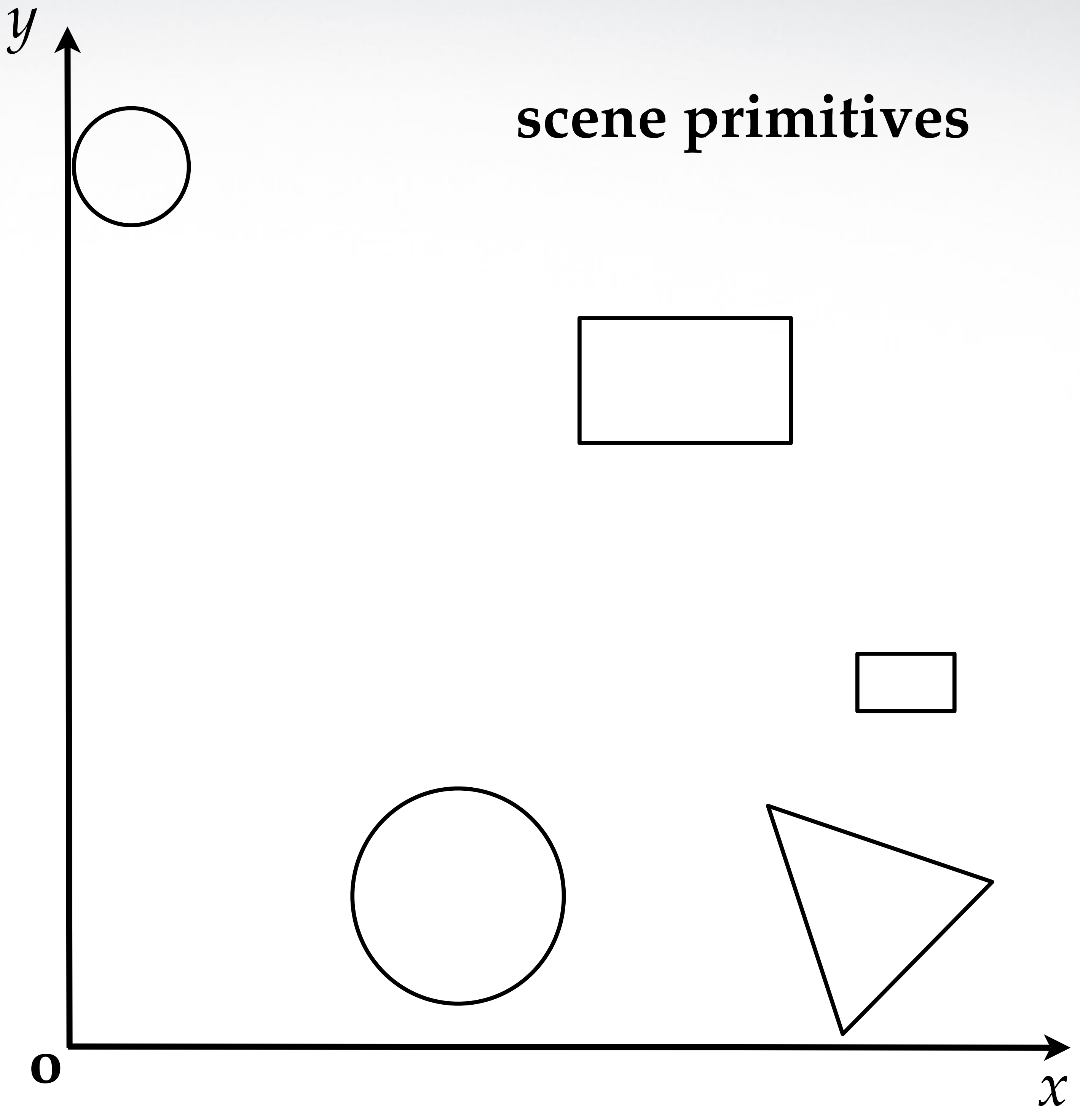
Whitted, 1980

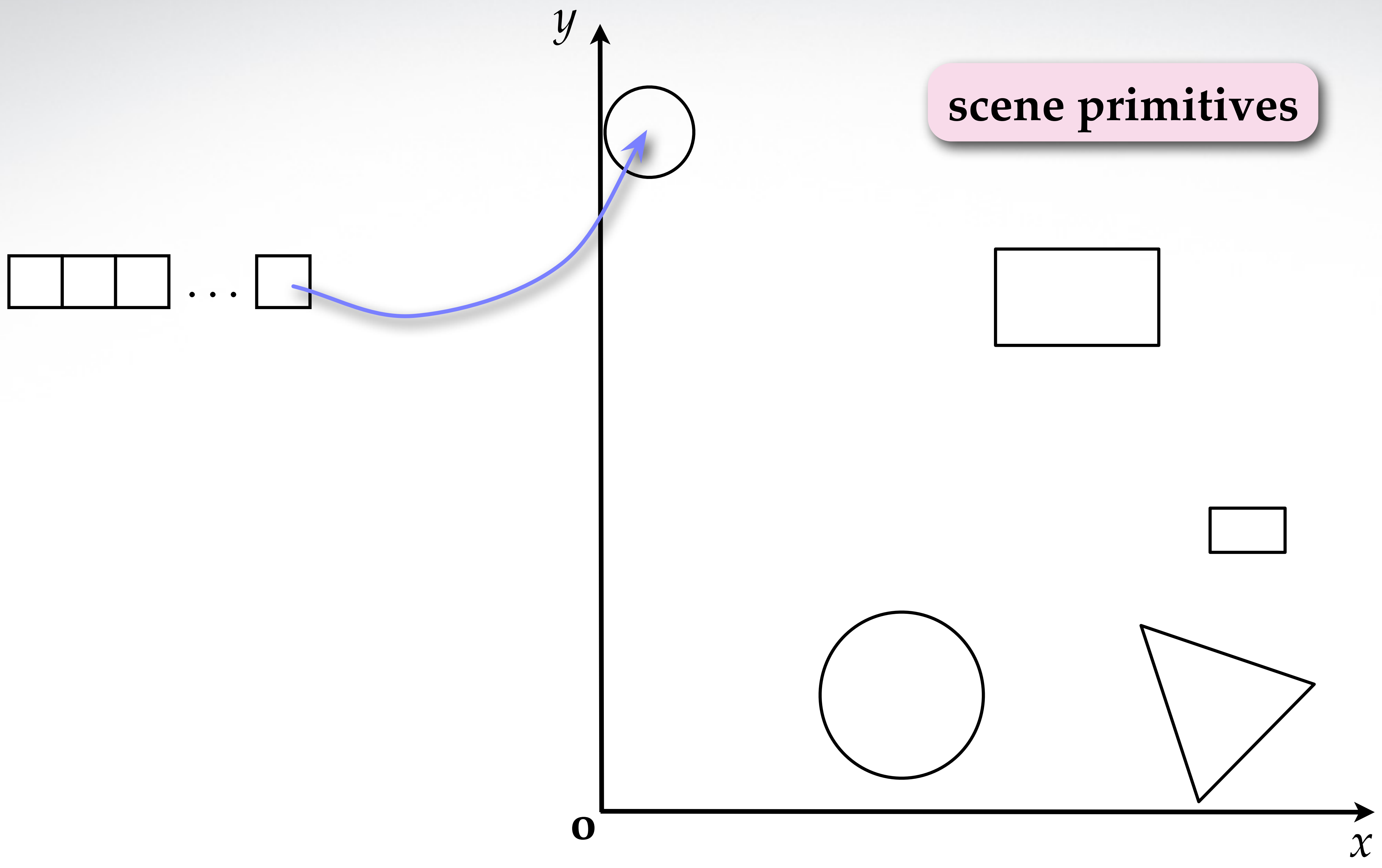
Acceleration methods for ray-surface intersections



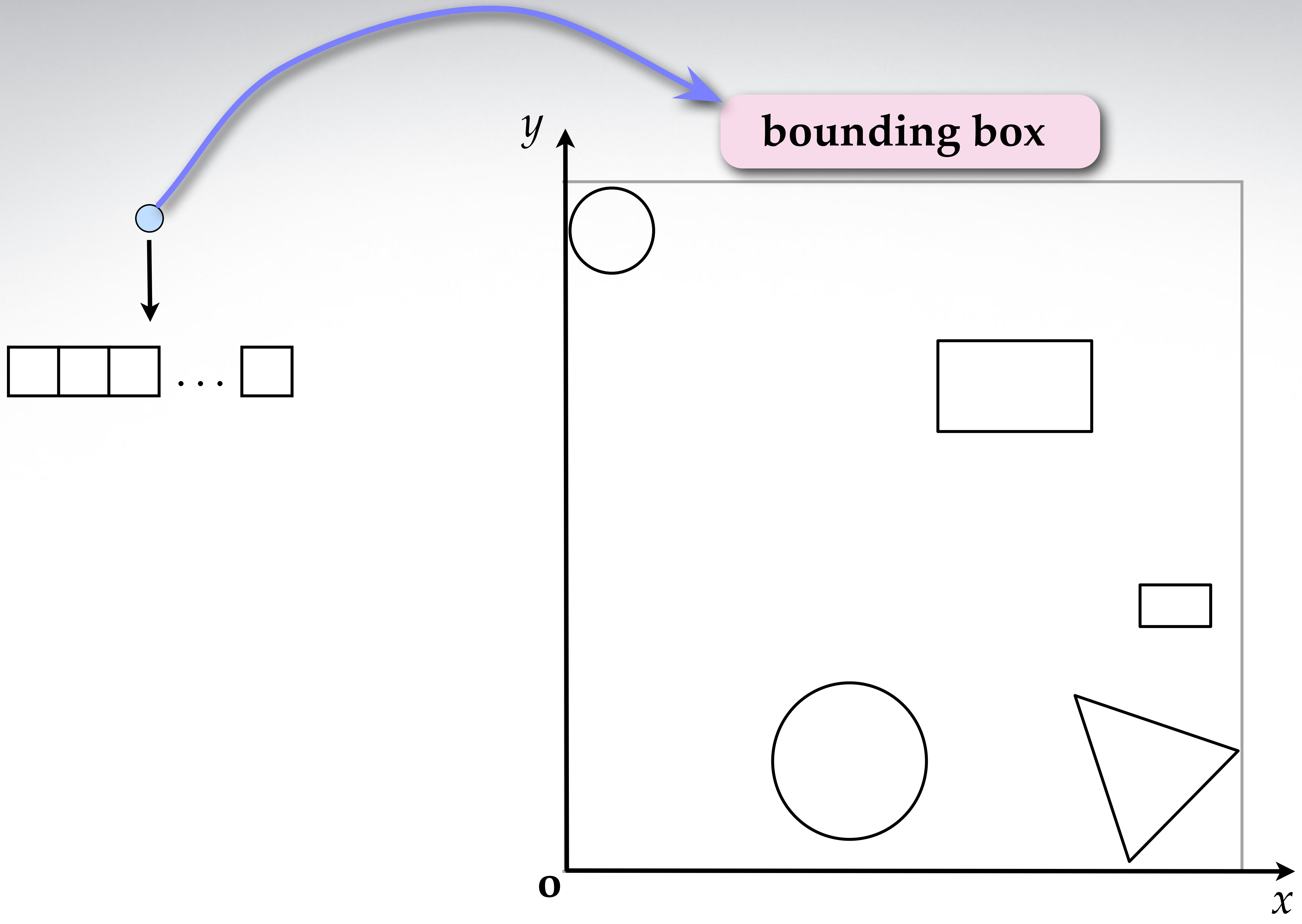
The *k*d-tree (k=2)

scene primitives

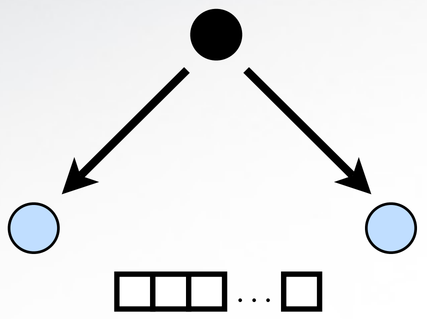


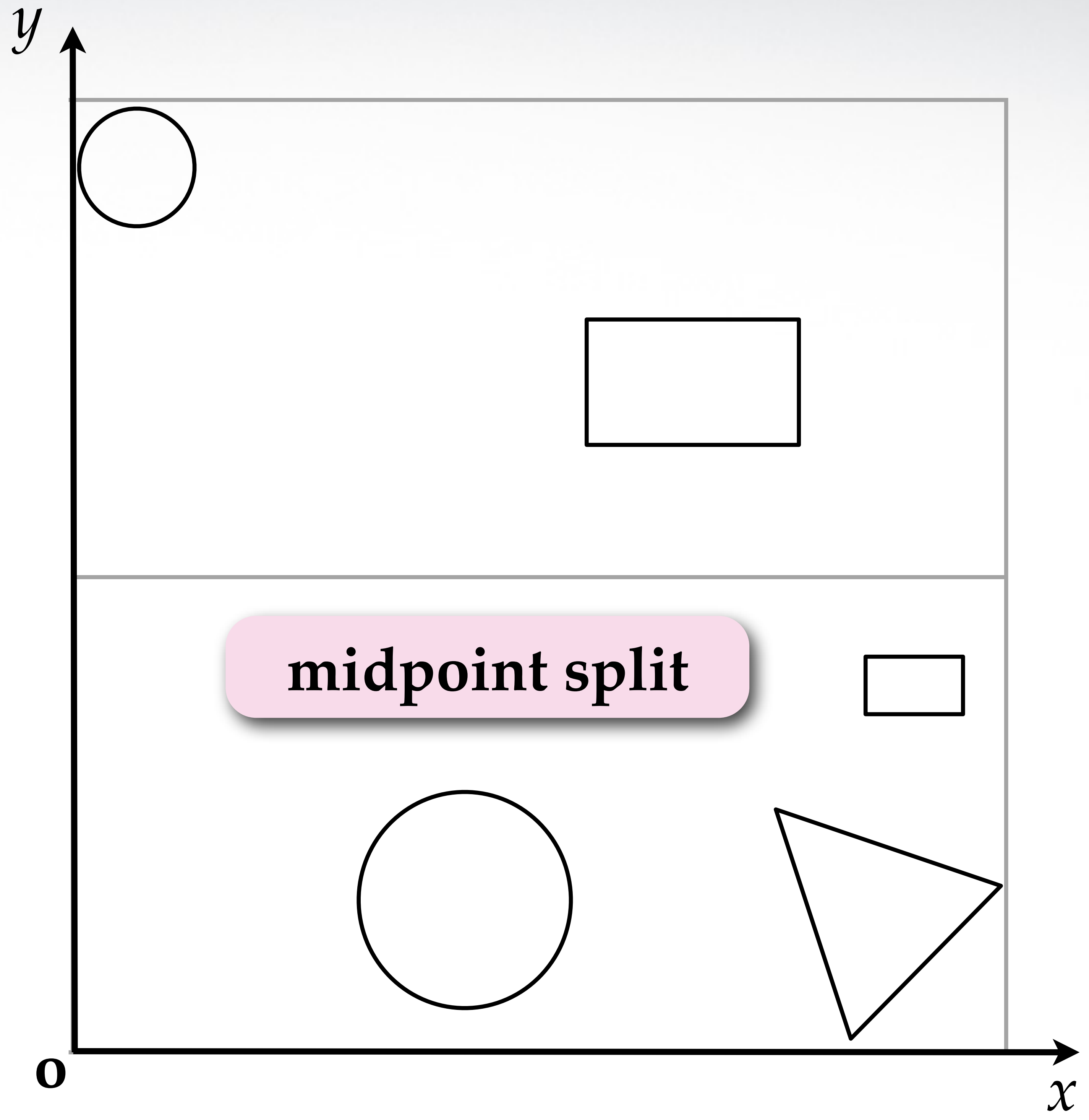


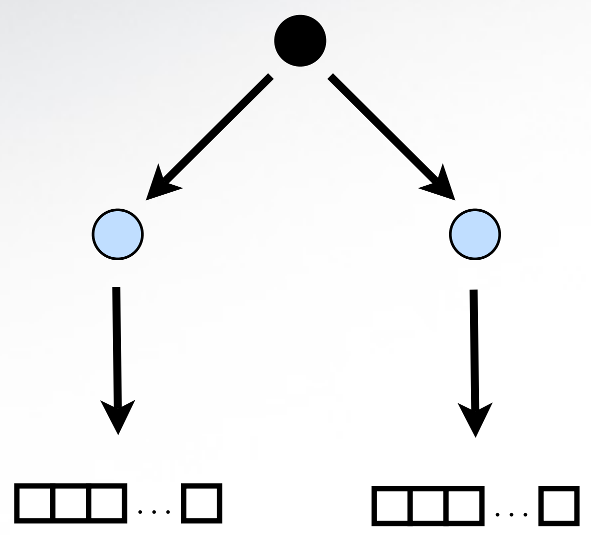
y
scene primitives
$\ldots$
$\mathbf{o}$
x

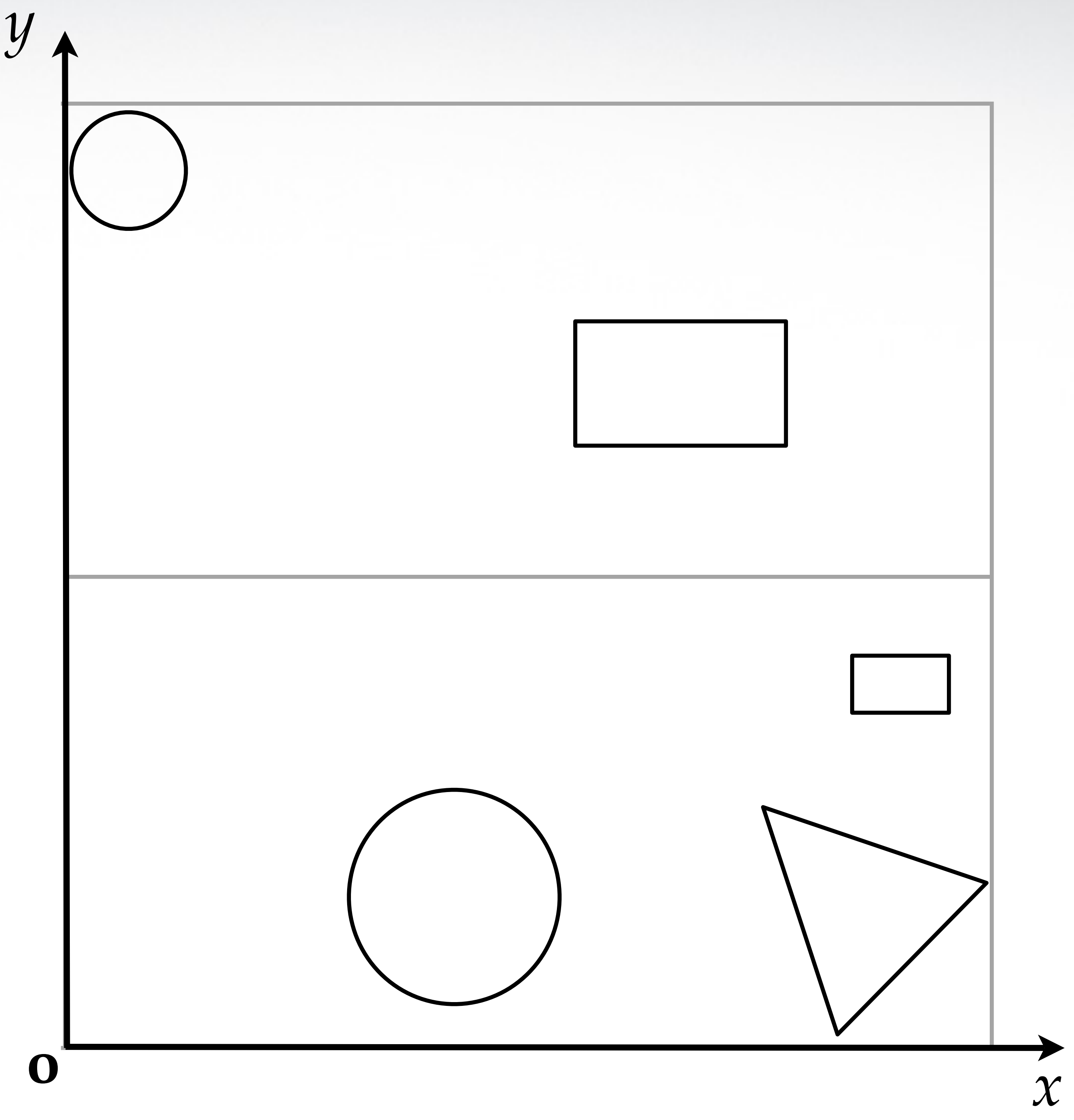

bounding box
y
x
o
. . .

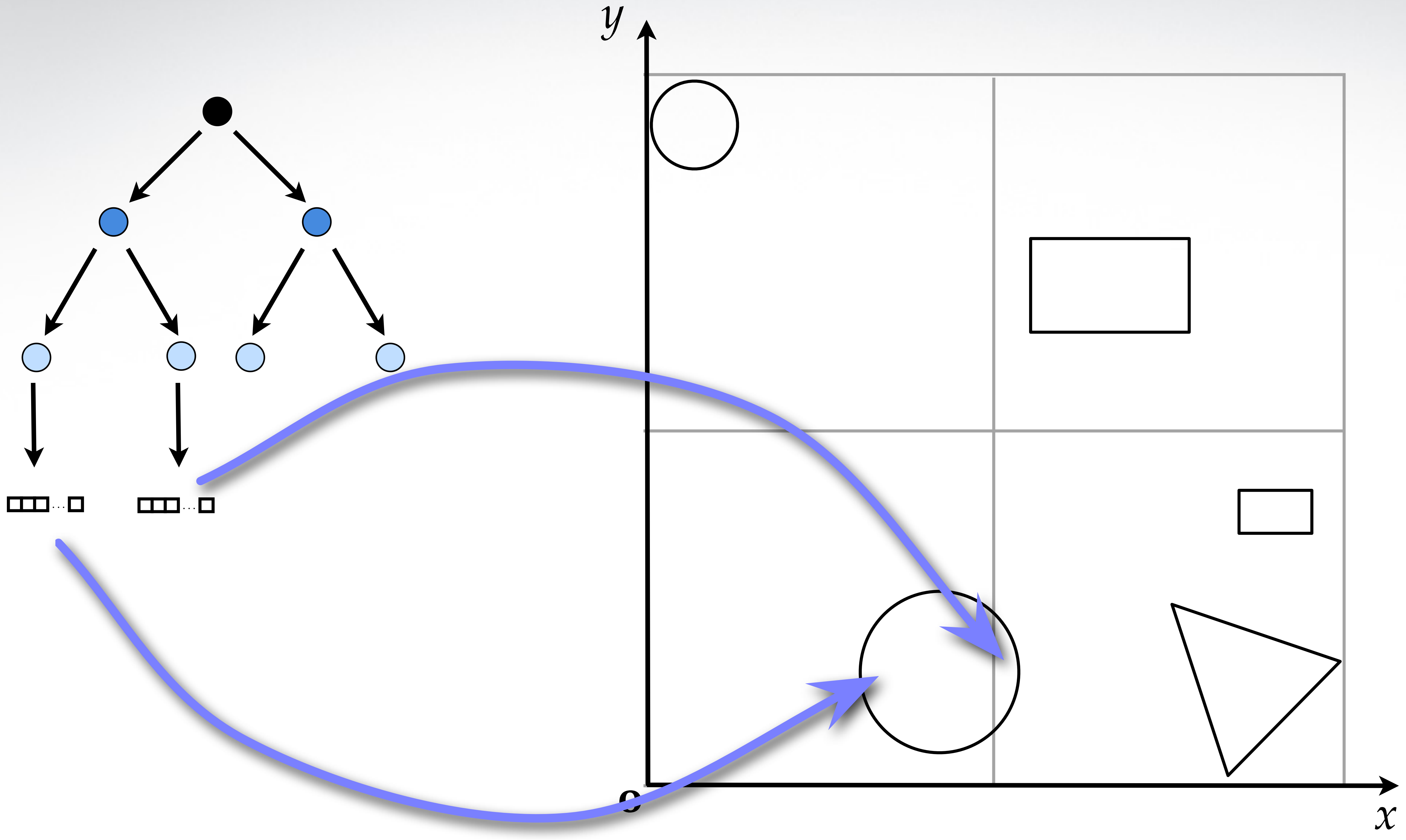
Goal: Reduce number of intersections

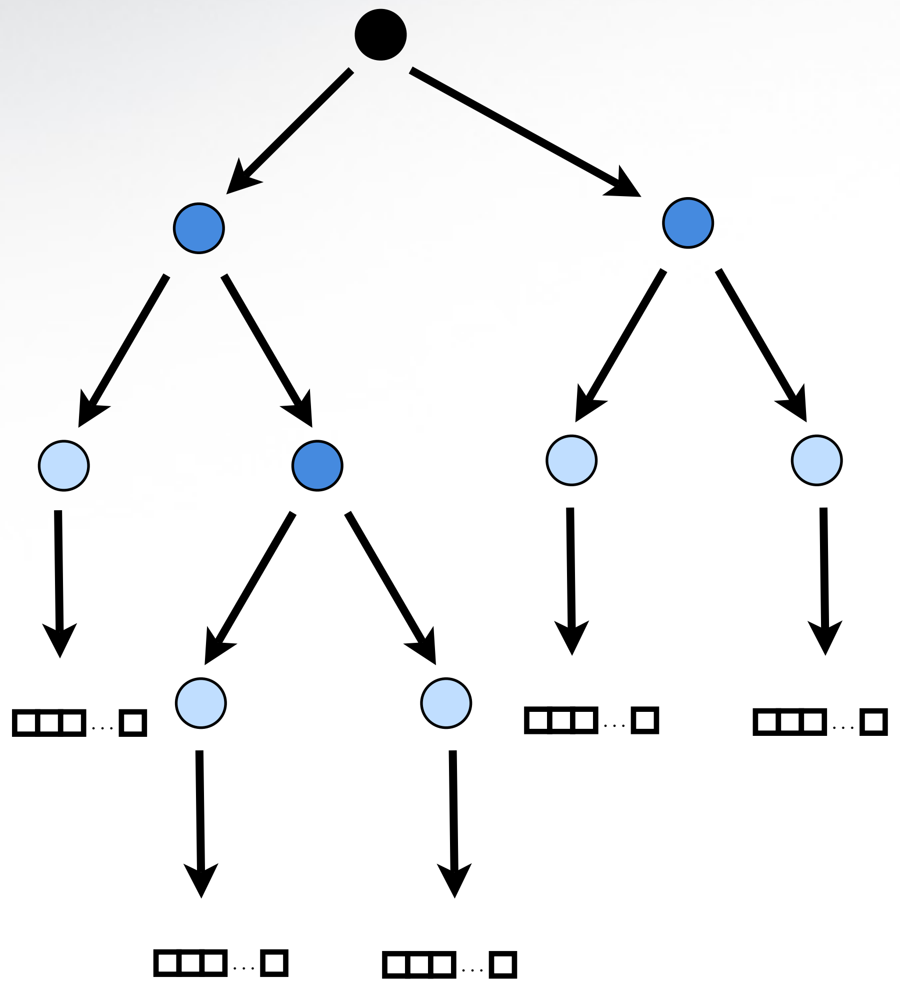


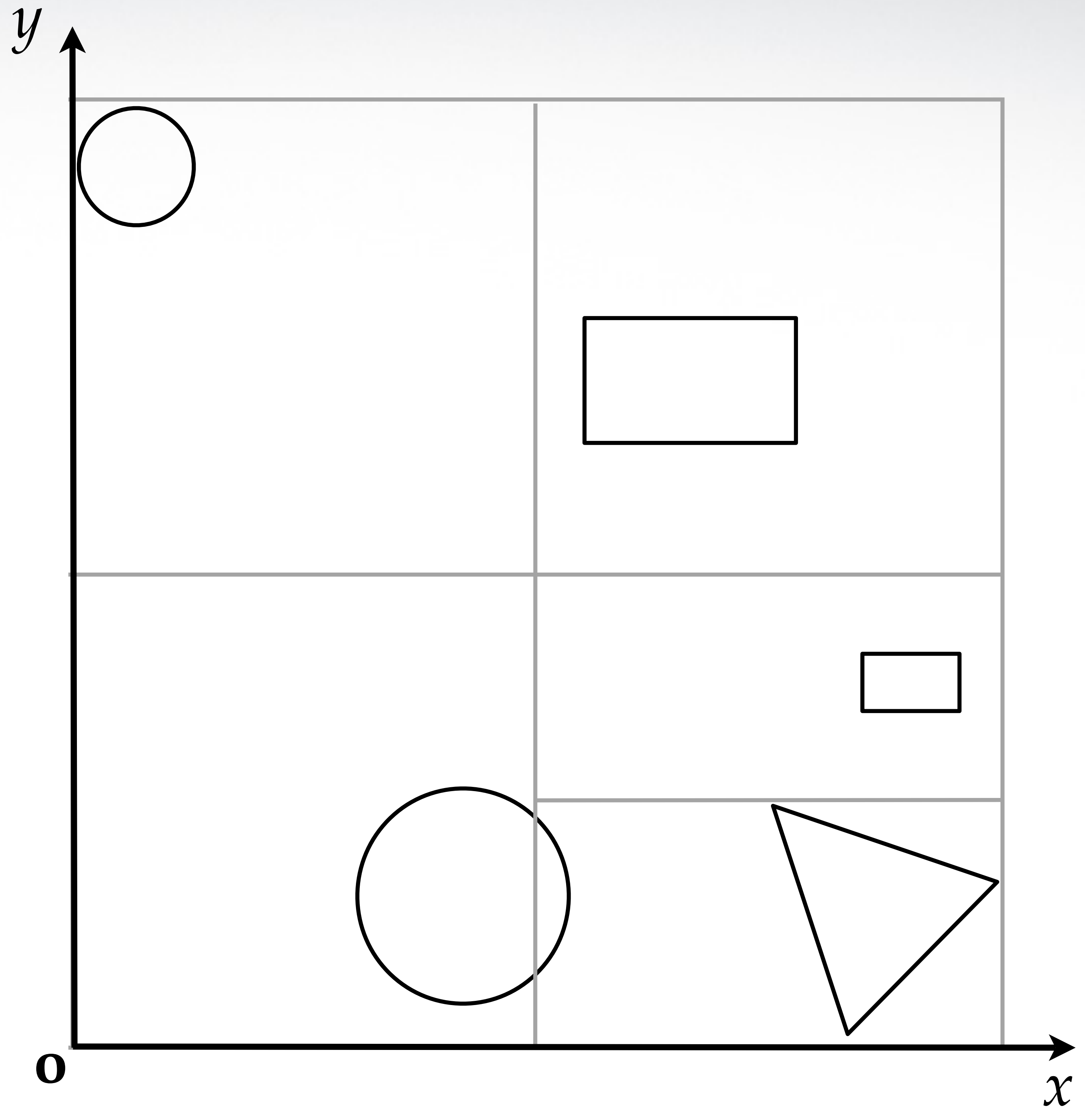


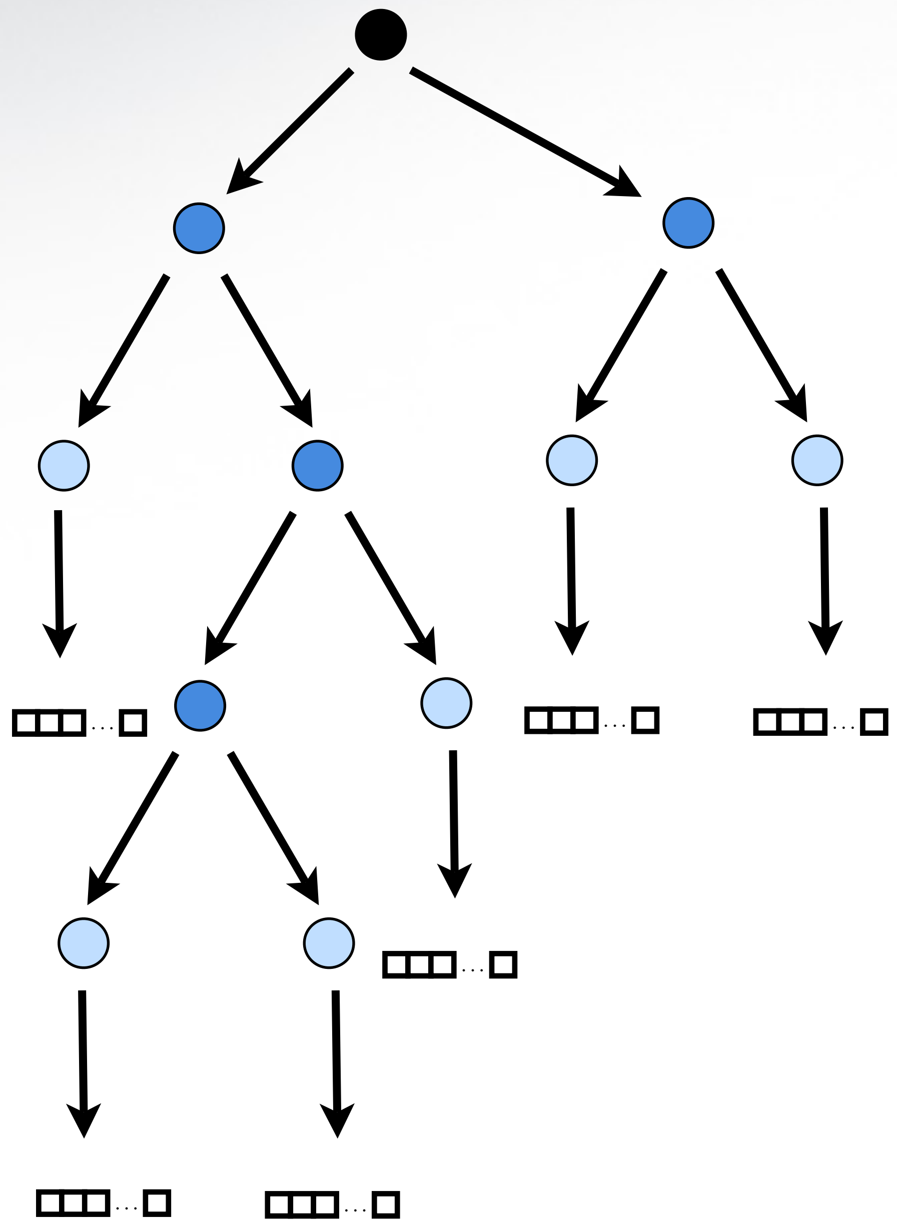


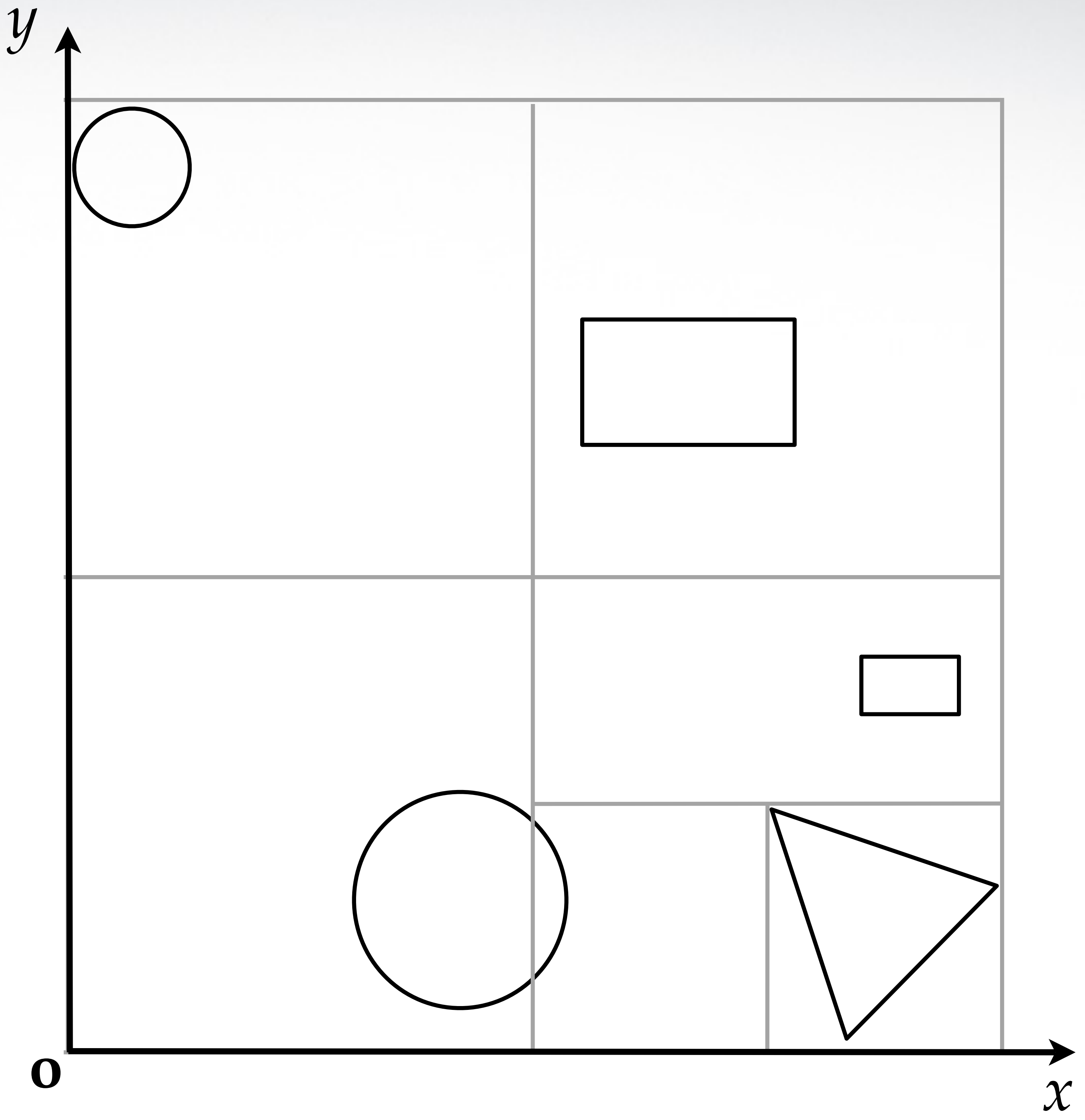

y
x
O


y
x
0

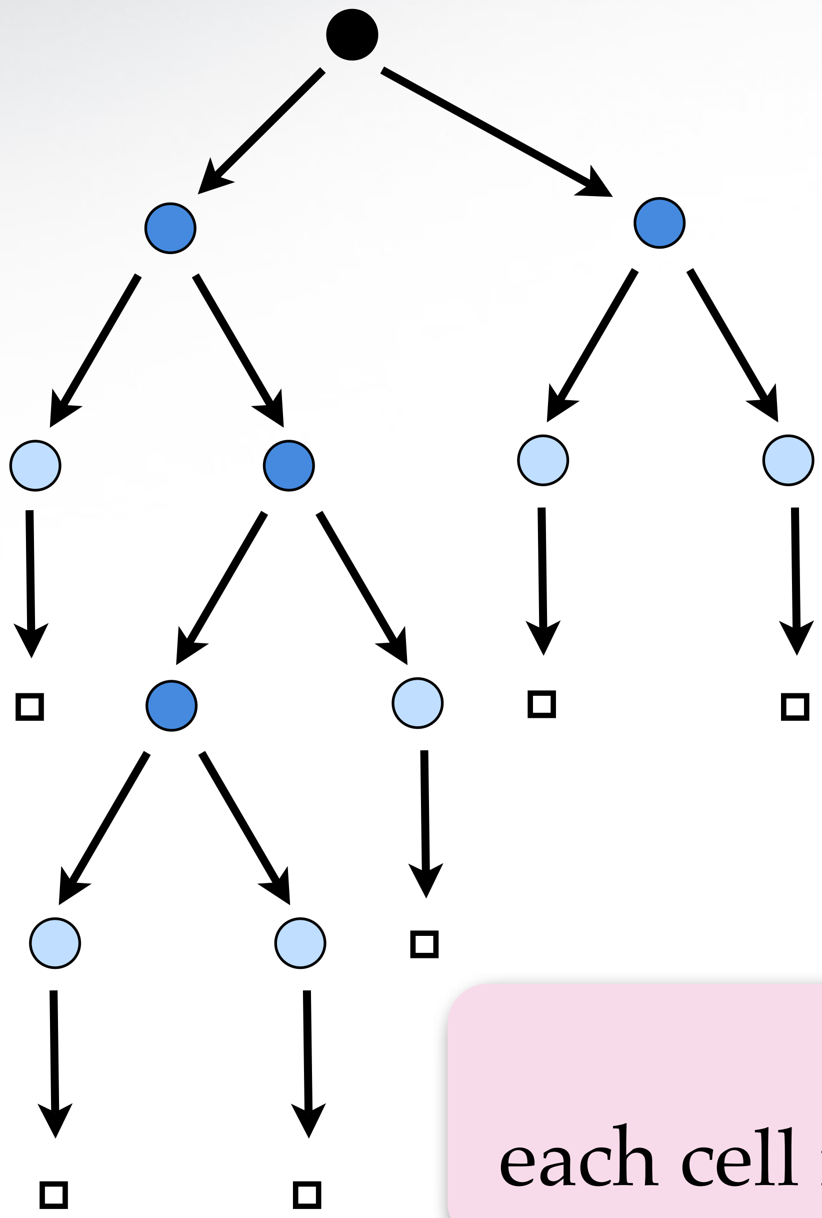


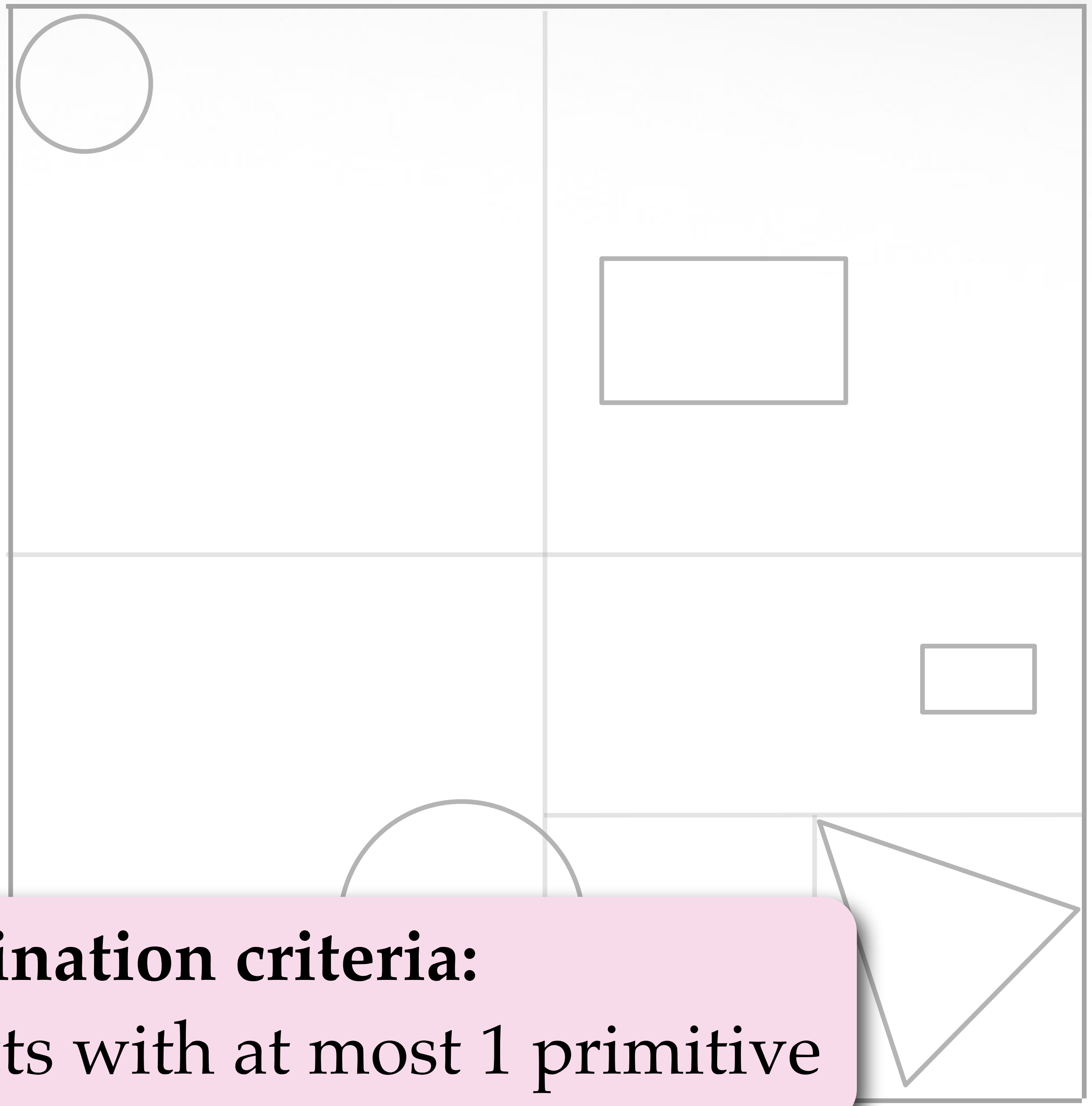

y
0
x



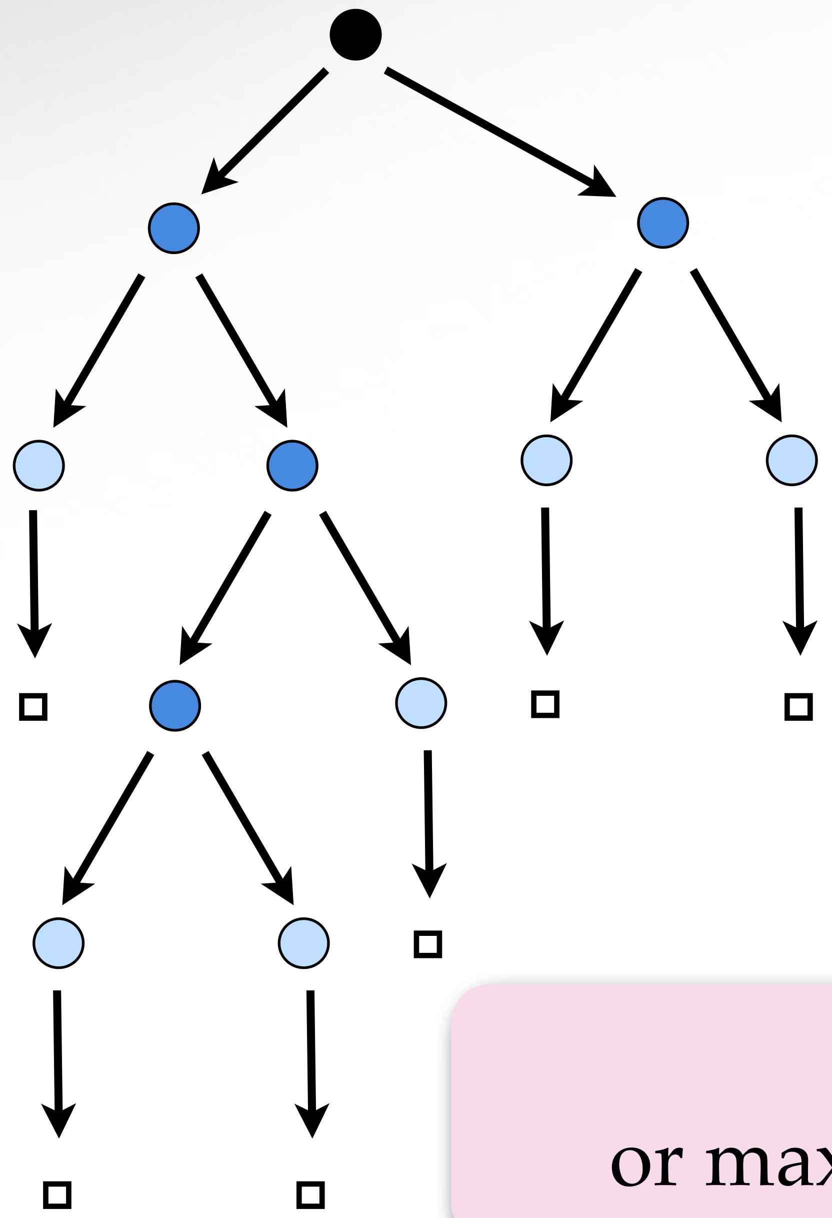


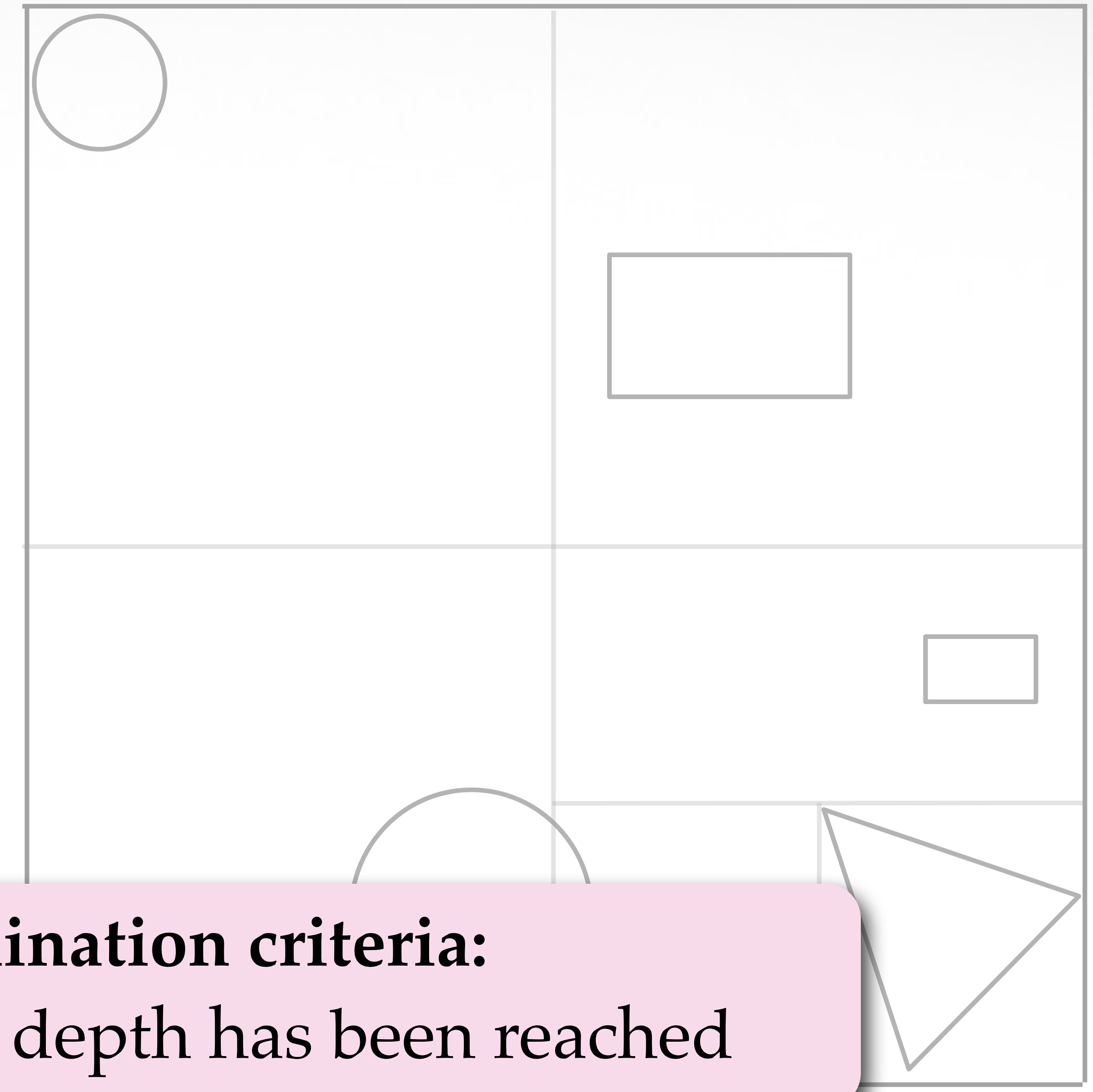
y
0
x





Termination criteria:
each cell intersects with at most 1 primitive

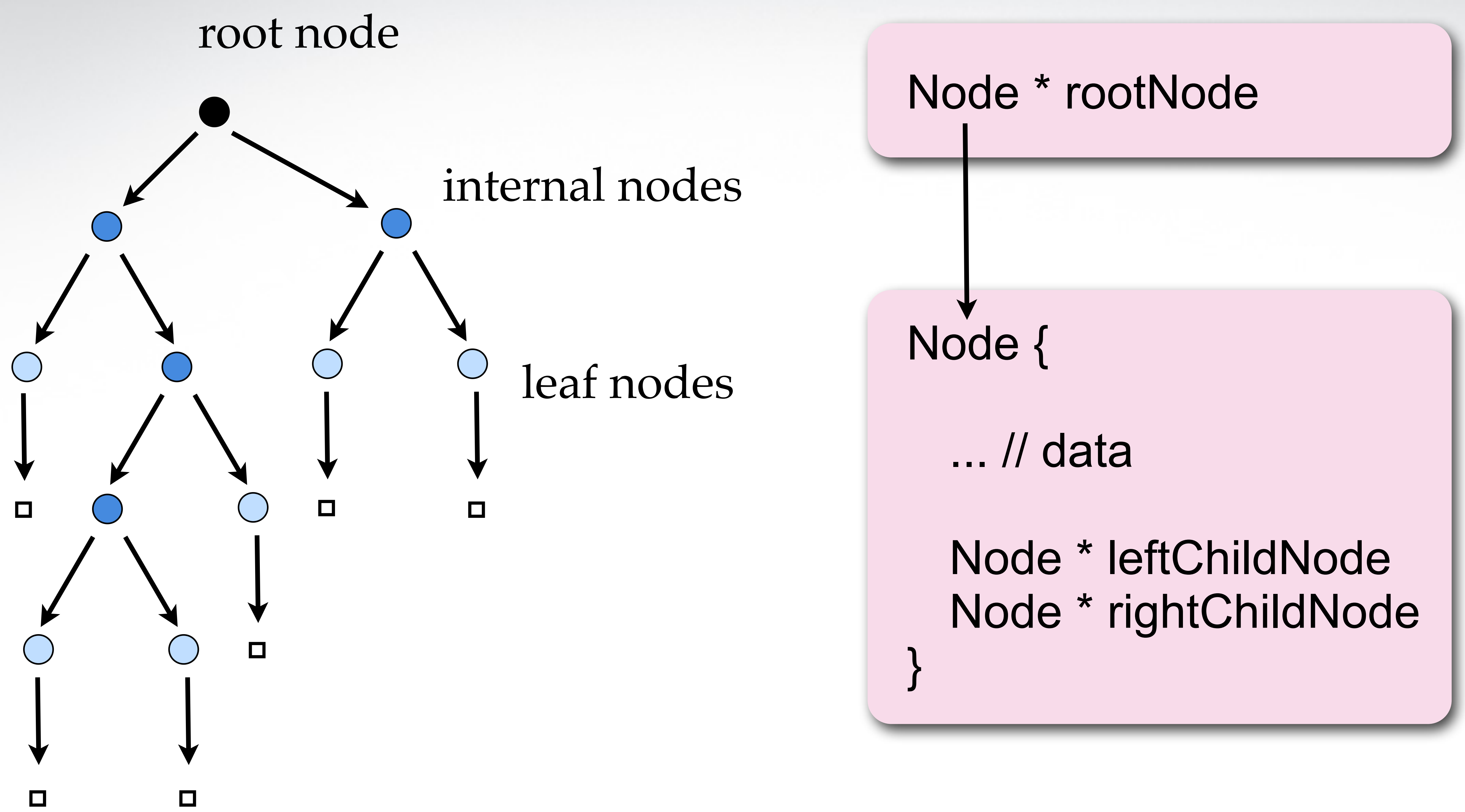




Termination criteria:

or maximum depth has been reached

binary tree



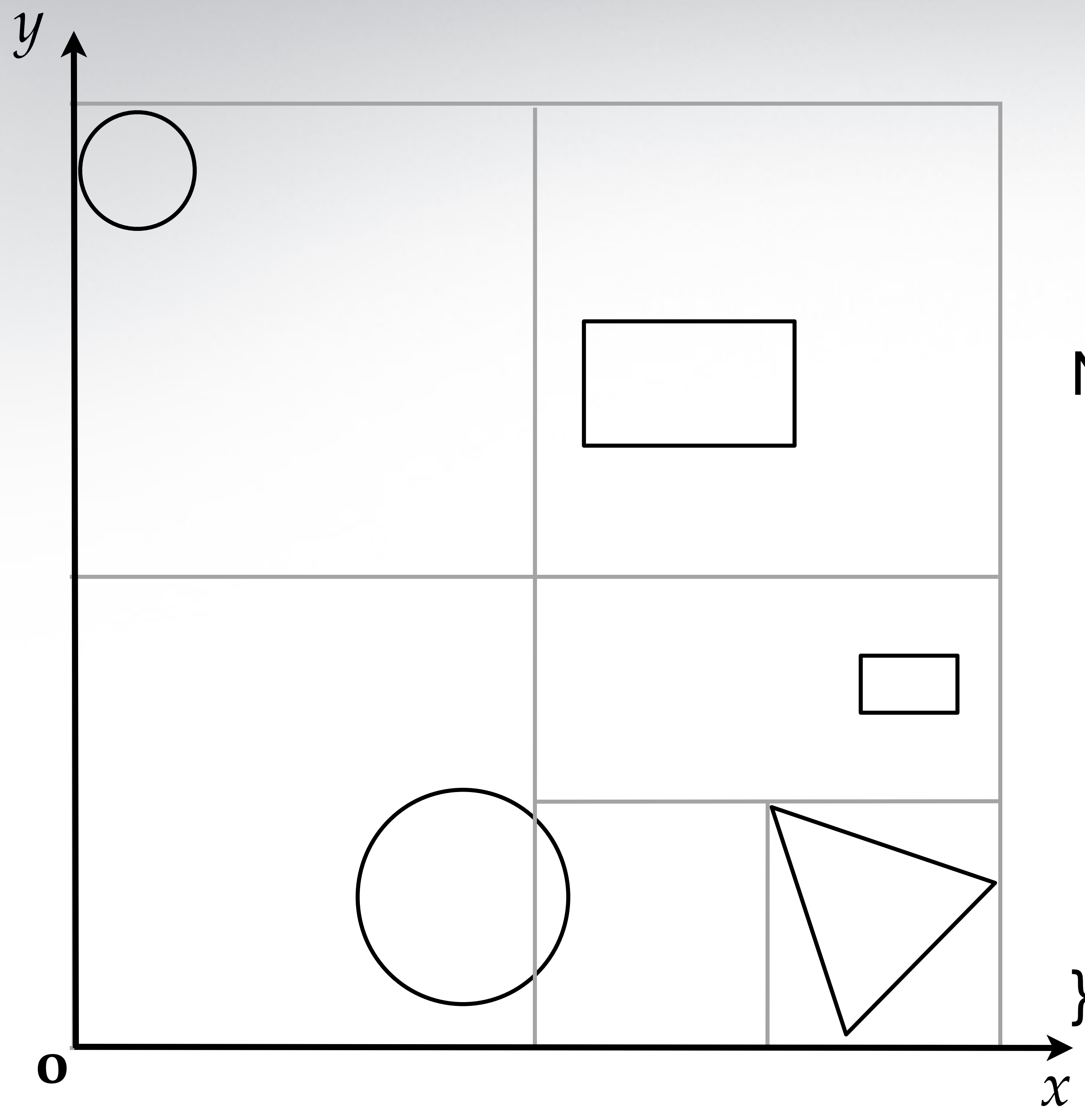
```
Node * rootNode
```

```
Node {

    ... // data

    Node * leftChildNode
    Node * rightChildNode
}
```

All the data we need:



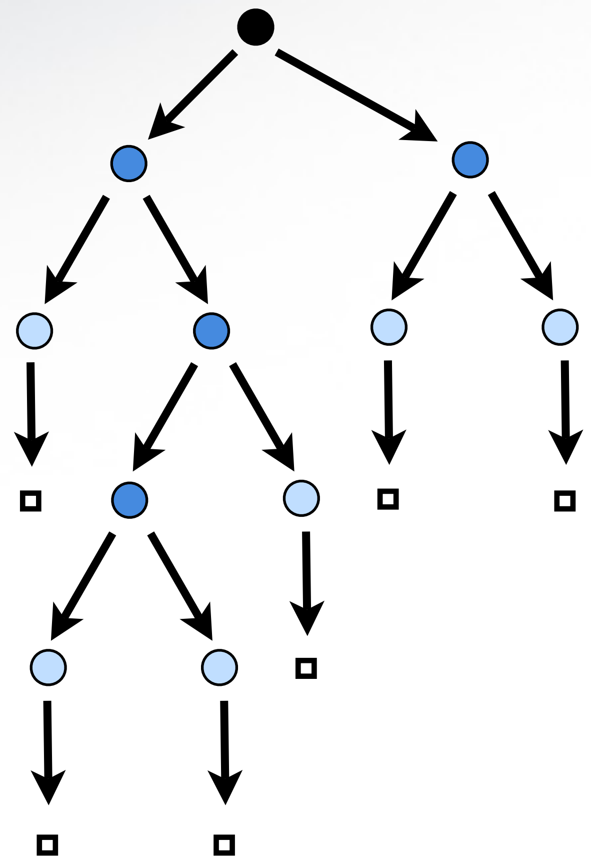
Node * rootNode

Node {

AABB boundingBox
axis splittingAxis
double splittingCoordinate
list pointersToPrimitives(if leaf)

Node * leftChildNode
Node * rightChildNode
}

How do we build it?

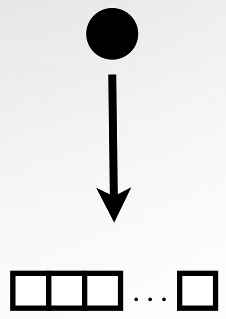


<u>procedure buildKDTree(Scene, Node, ...)</u>

```
initScene(scene, rootNode)

subdivideCell(scene, node, ...) //recursive
```

Initialization

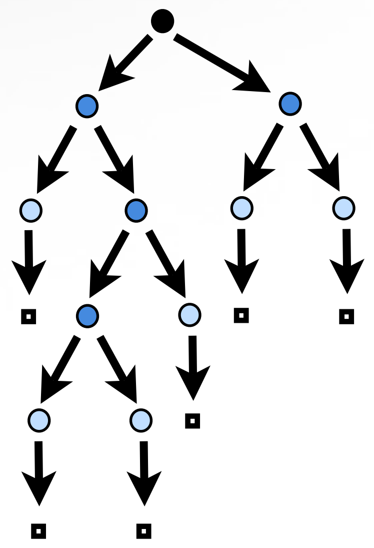


procedure initKDTree(Scene, Node)

```
rootNode→ boundingBox = computeBB(...)
          splittingAxis = X
          splittingCoordinate = midPoint(...)
          leftChildNode = NULL
          rightChildNode = NULL

for each primitive P in scene
   rootNode→ pointersToPrimitives.add(P)
```

Recursion



```
procedure subdivideCell(Scene, Node, ...)

if(!terminateConstruction(node, ...))

    currentDepth++

    computeSplittingCoordinate(node)

    ... // create children

    ... // initialize children
            node→leftChildNode.splittingAxis=nextAxis(node)

    ... // move primitives to children

    subdivideCell(scene, node→leftChildNode, ...)
    subdivideCell(scene, node→rightChildNode, ...)
```

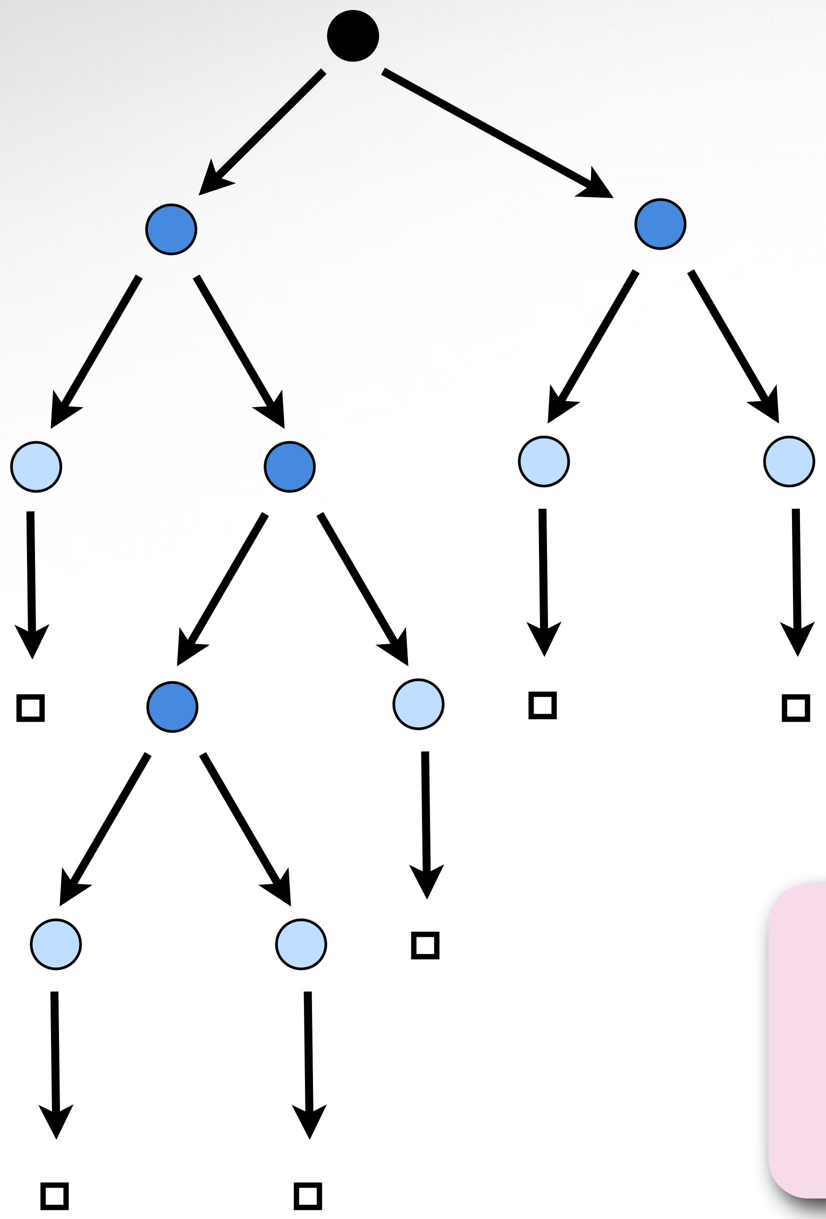
Termination

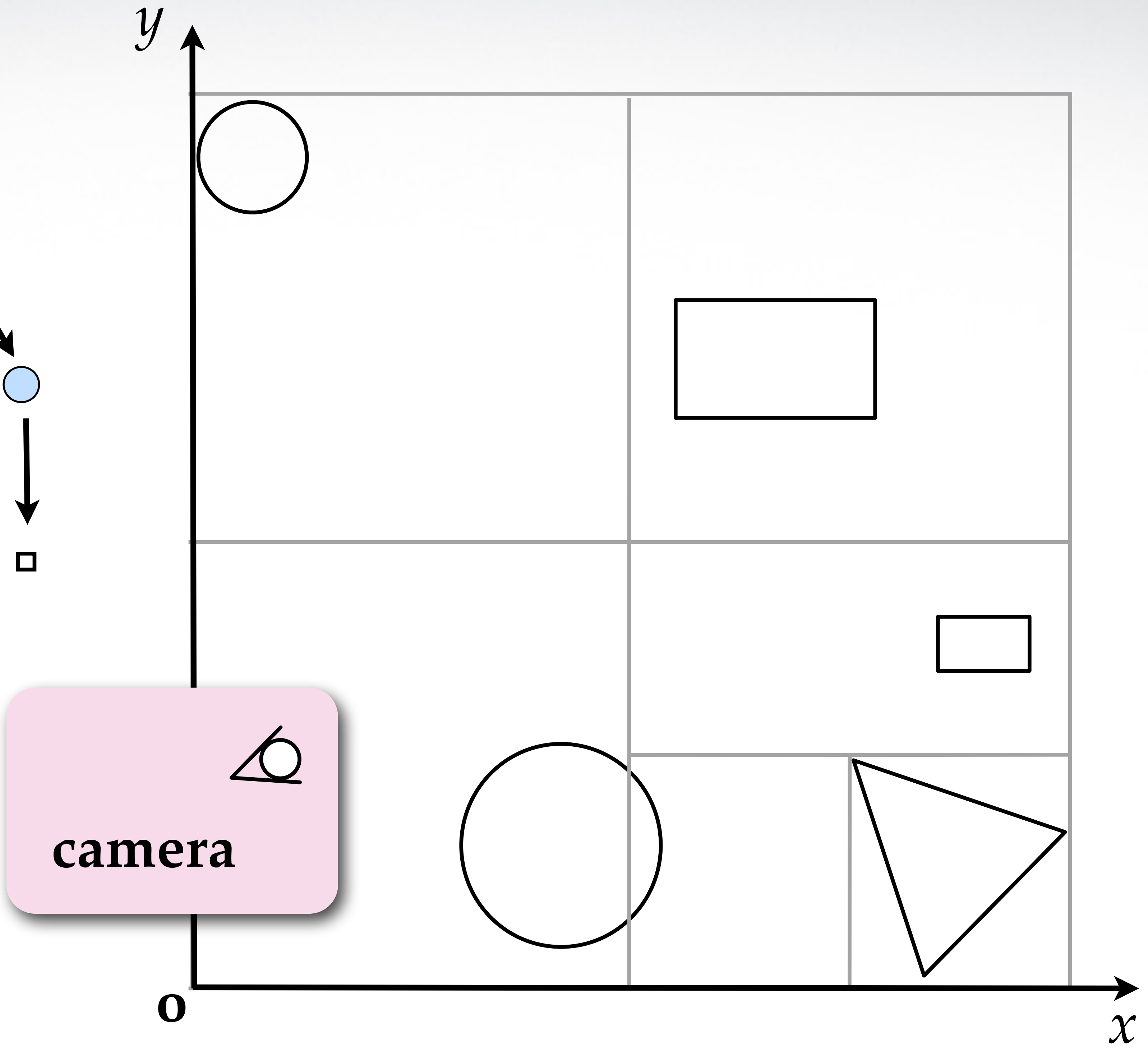
<u>function terminateConstruction(node, ...)</u>

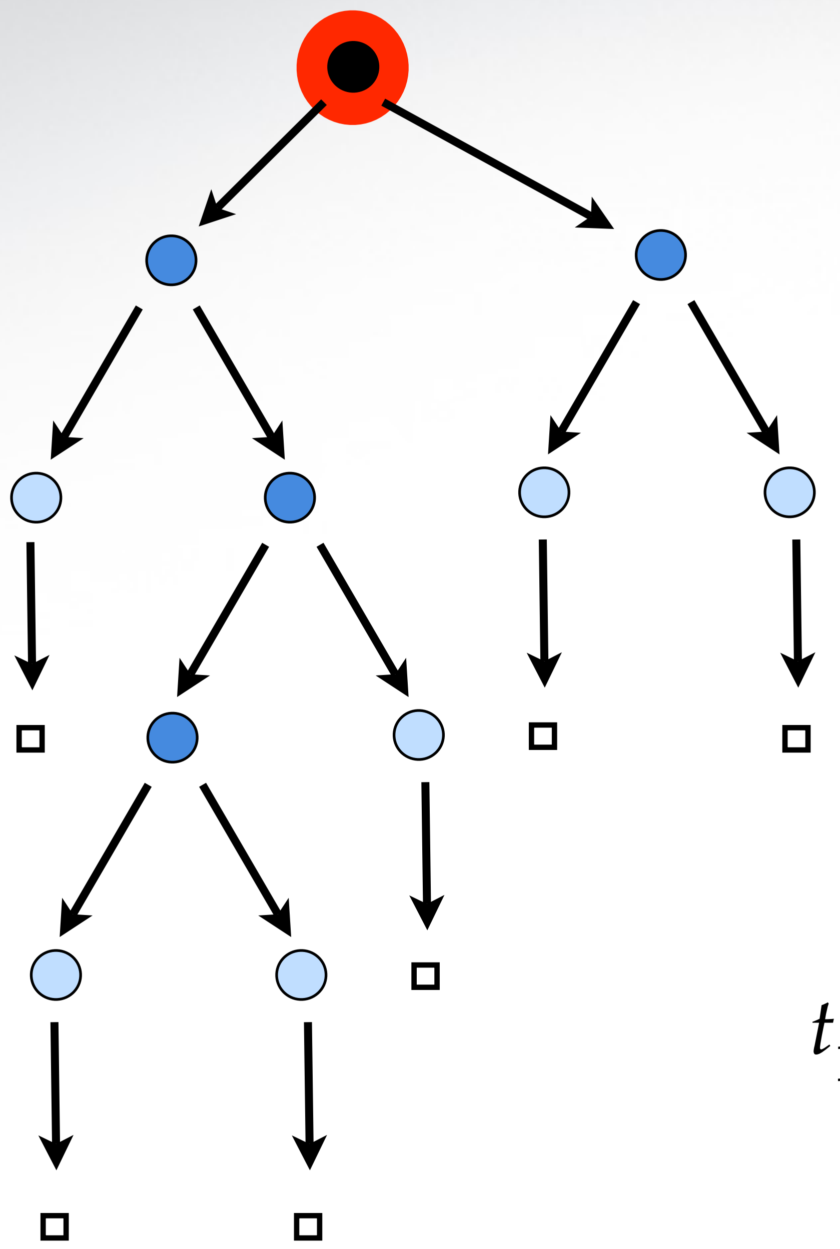
```
if (currentDepth == maxDepth) or
   ... // too few primitives in node
   return true
else
   return false
```

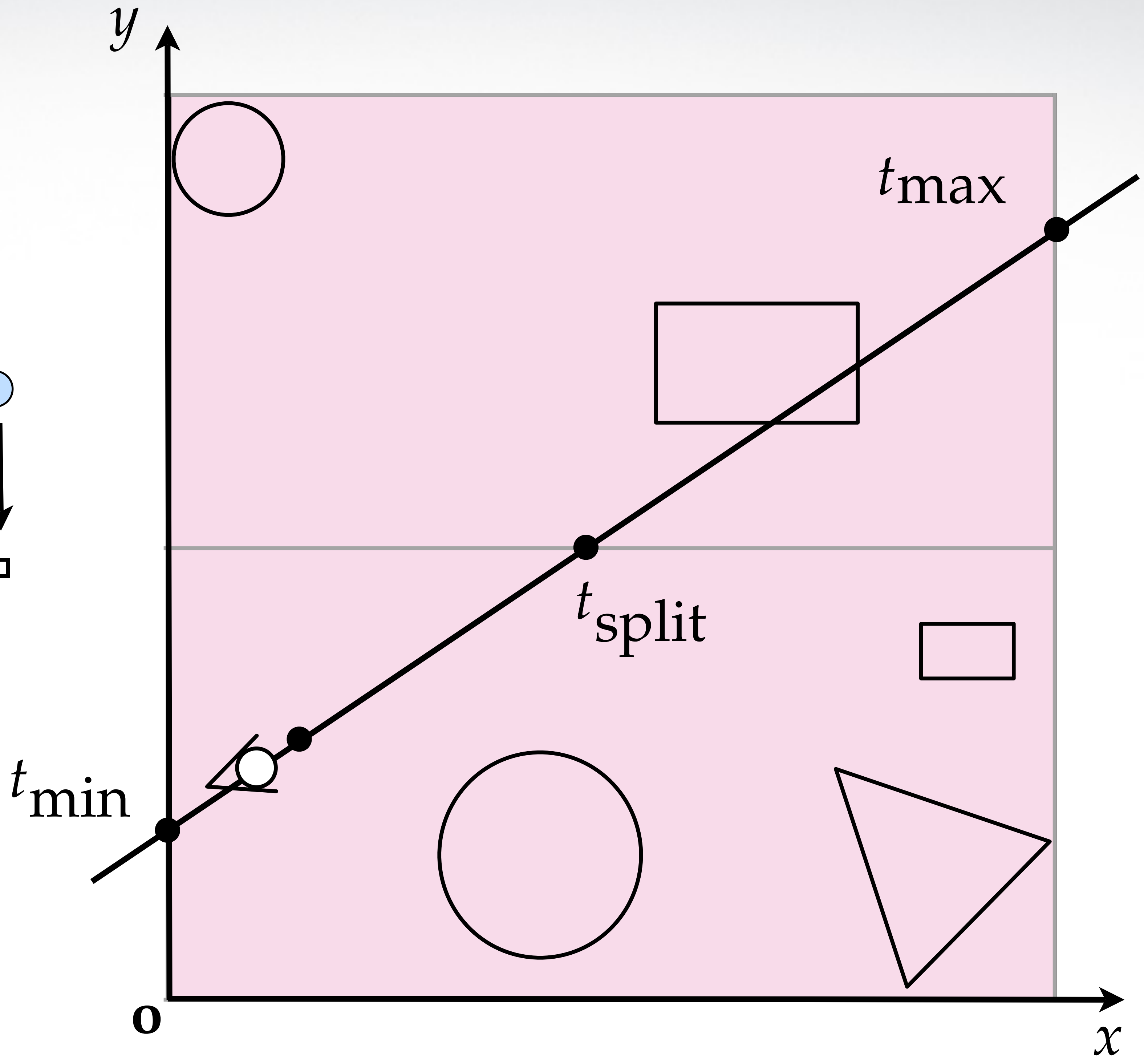
The cells of the *k*d-tree localize the **relevant primitives** for the intersections

Traversing the *k*d-tree is fast...

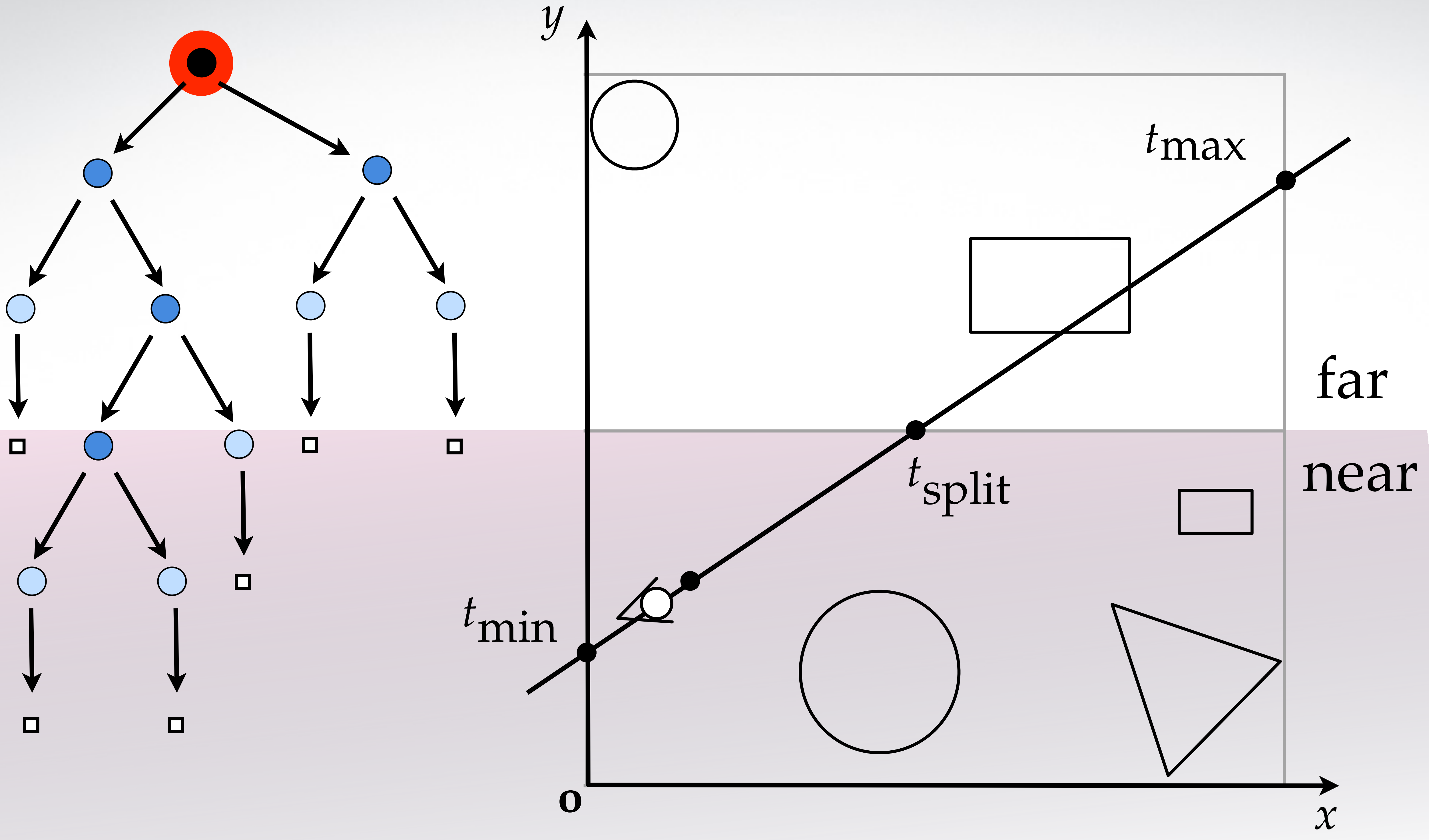



y
camera
o
x

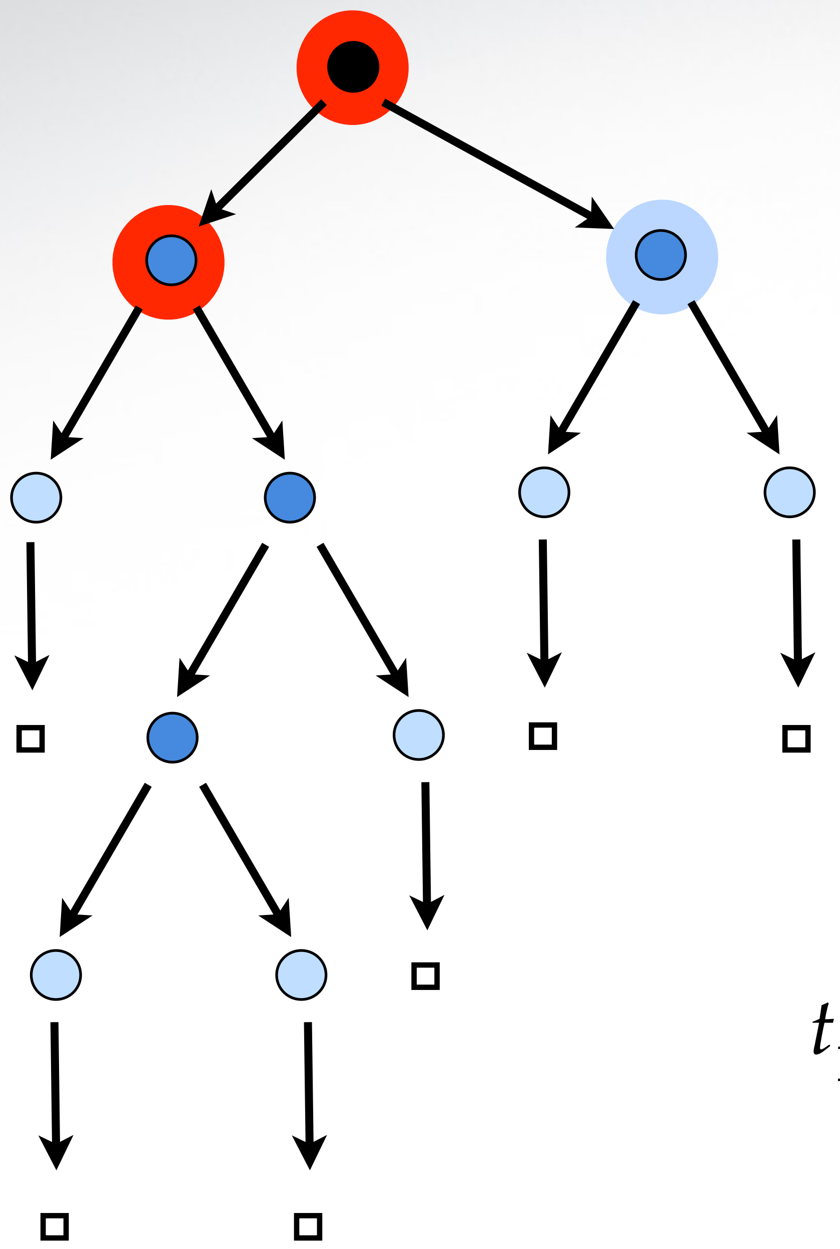


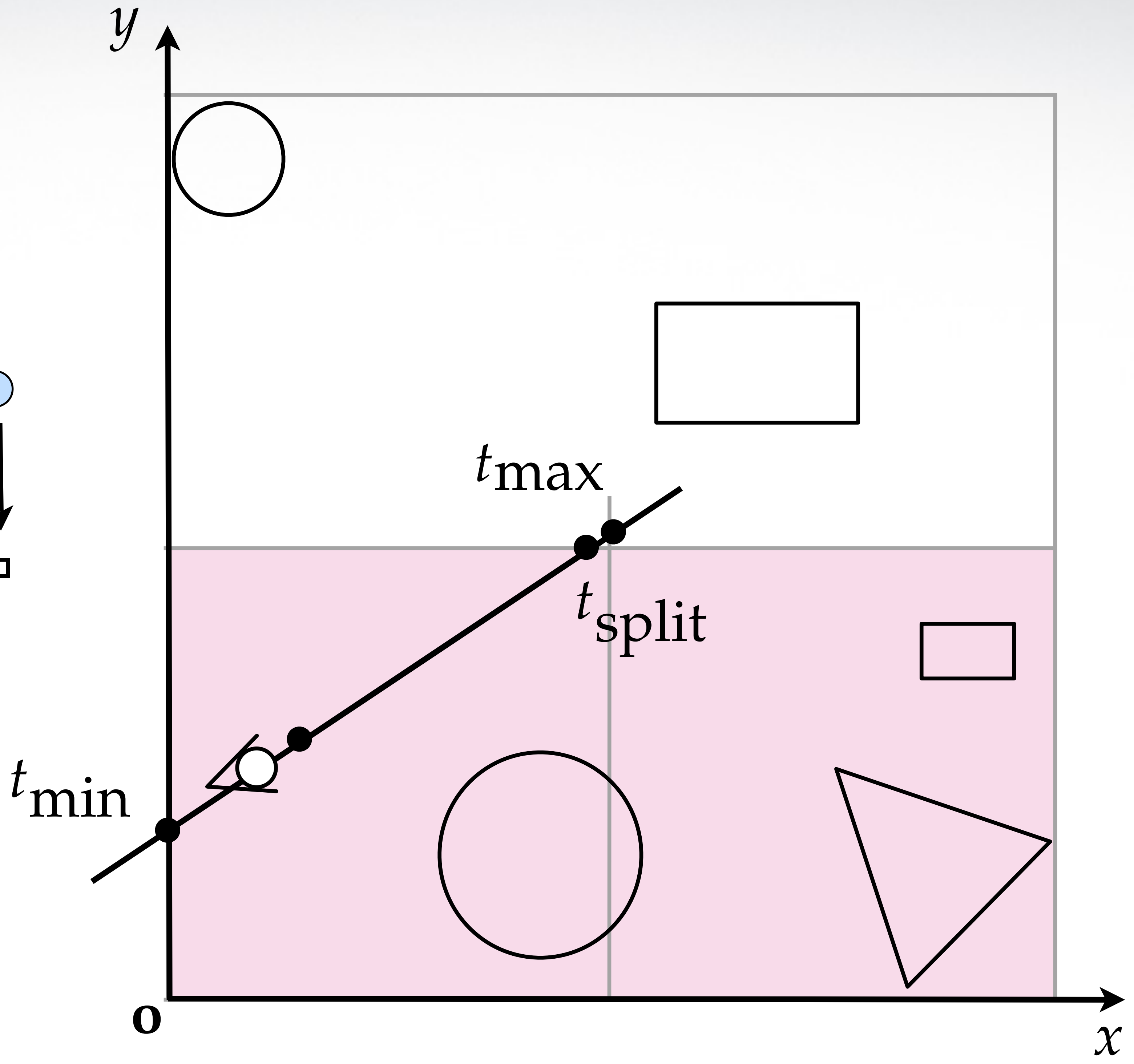


y
t_{max}
t_{split}
t_{min}
$\mathbf{o}$
x

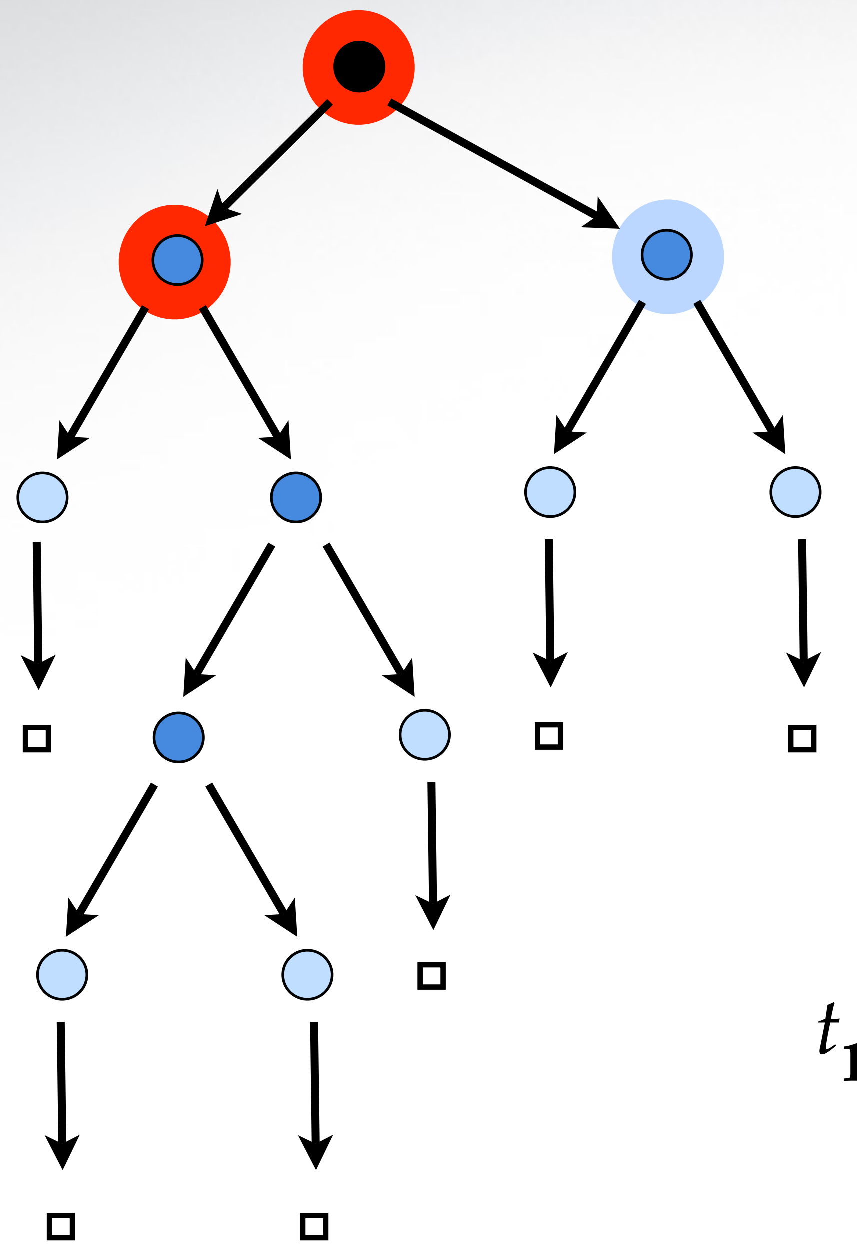


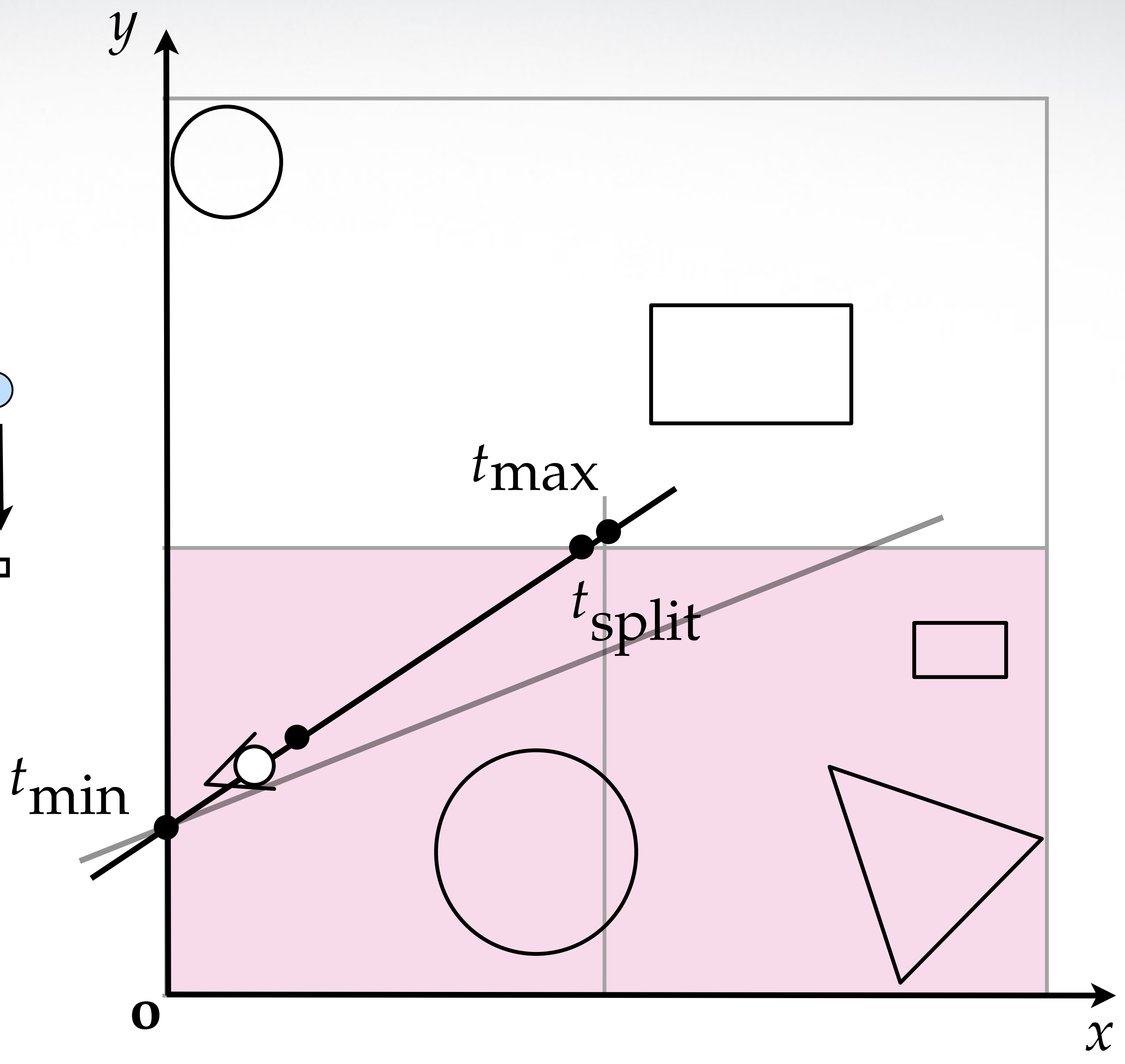
y
$t_{\max}$
far
near
t_{split}
$t_{\min}$
$\mathbf{o}$
x



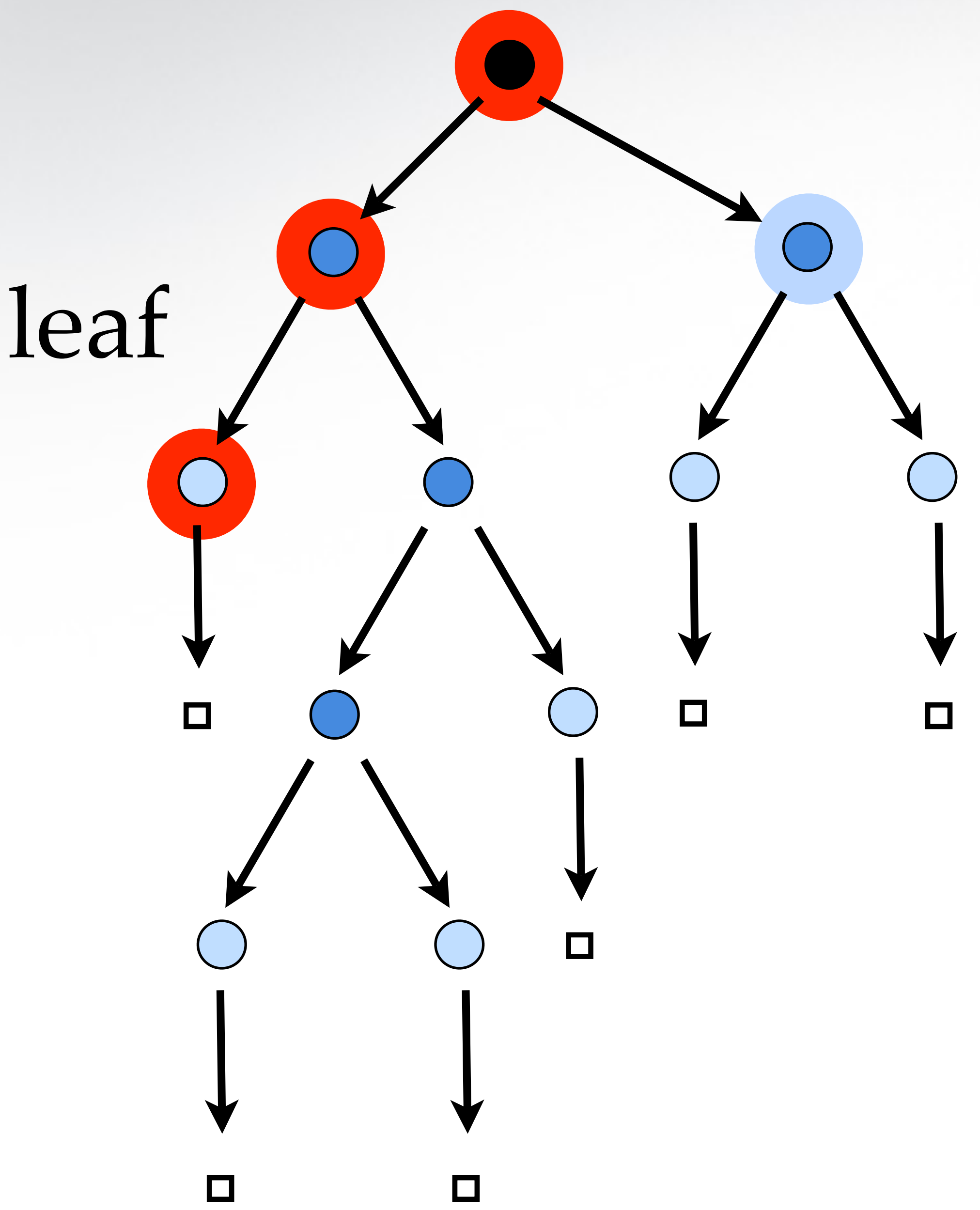


y
t_{max}
t_{split}
t_{min}
$\mathbf{o}$
x

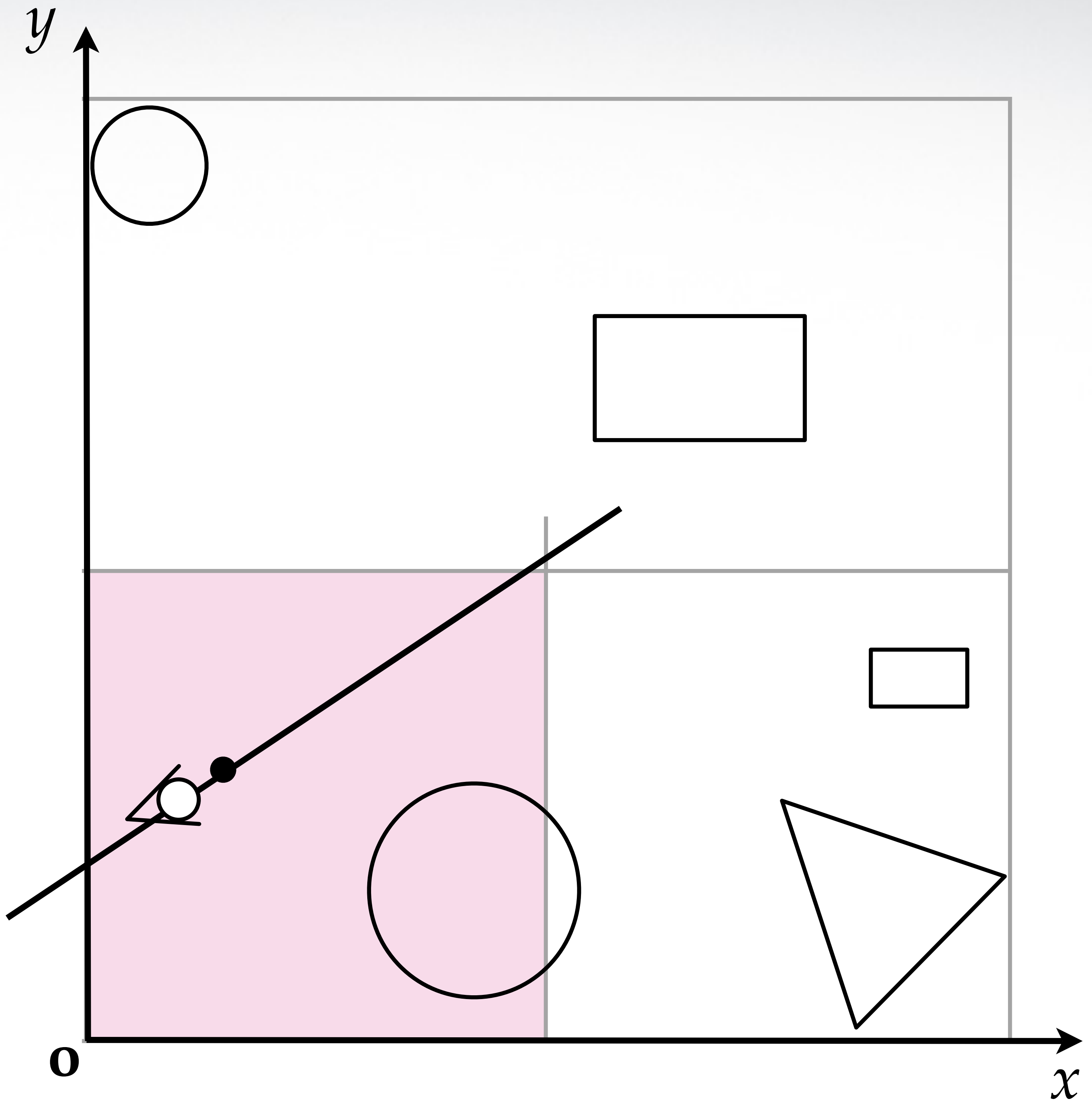




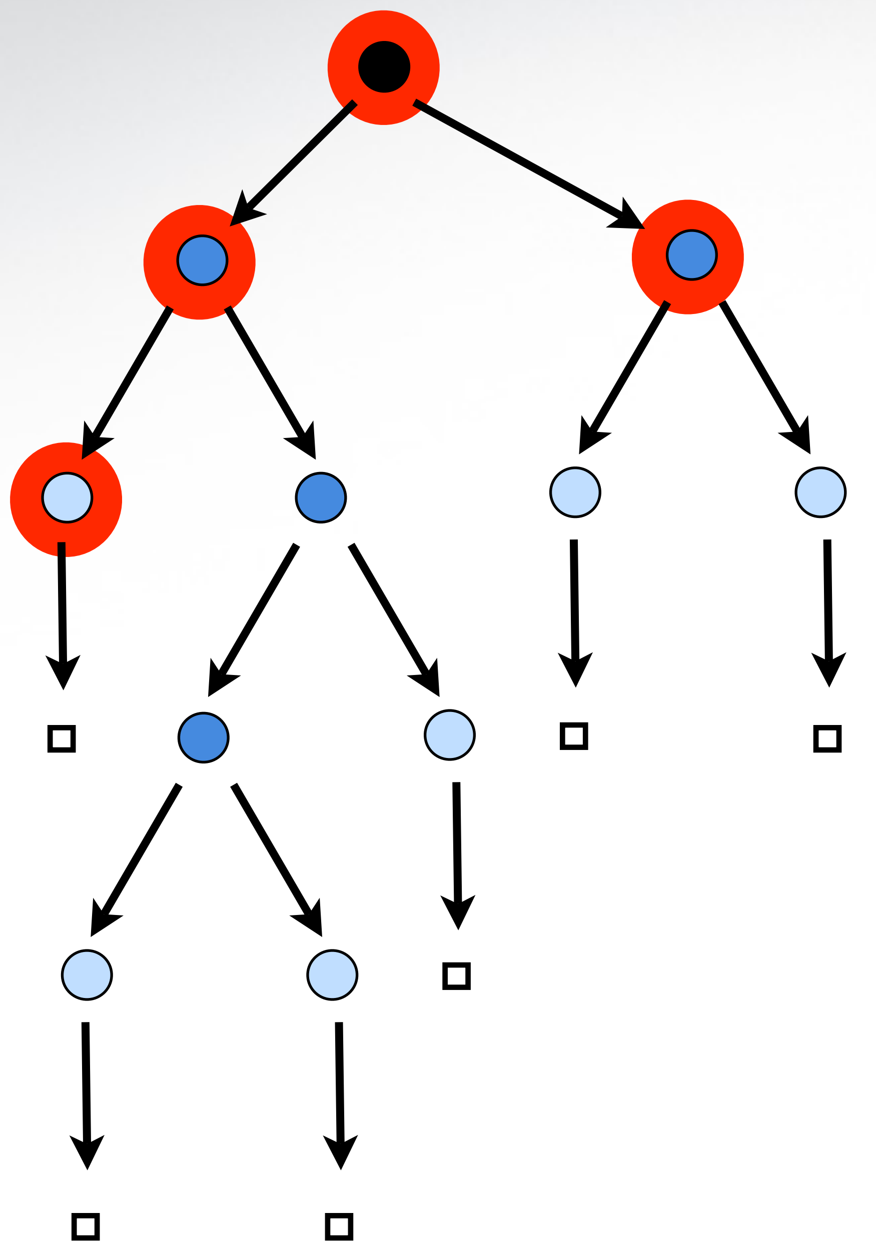
y
$t_{\max}$
t_{split}
$t_{\min}$
$\mathbf{o}$
x

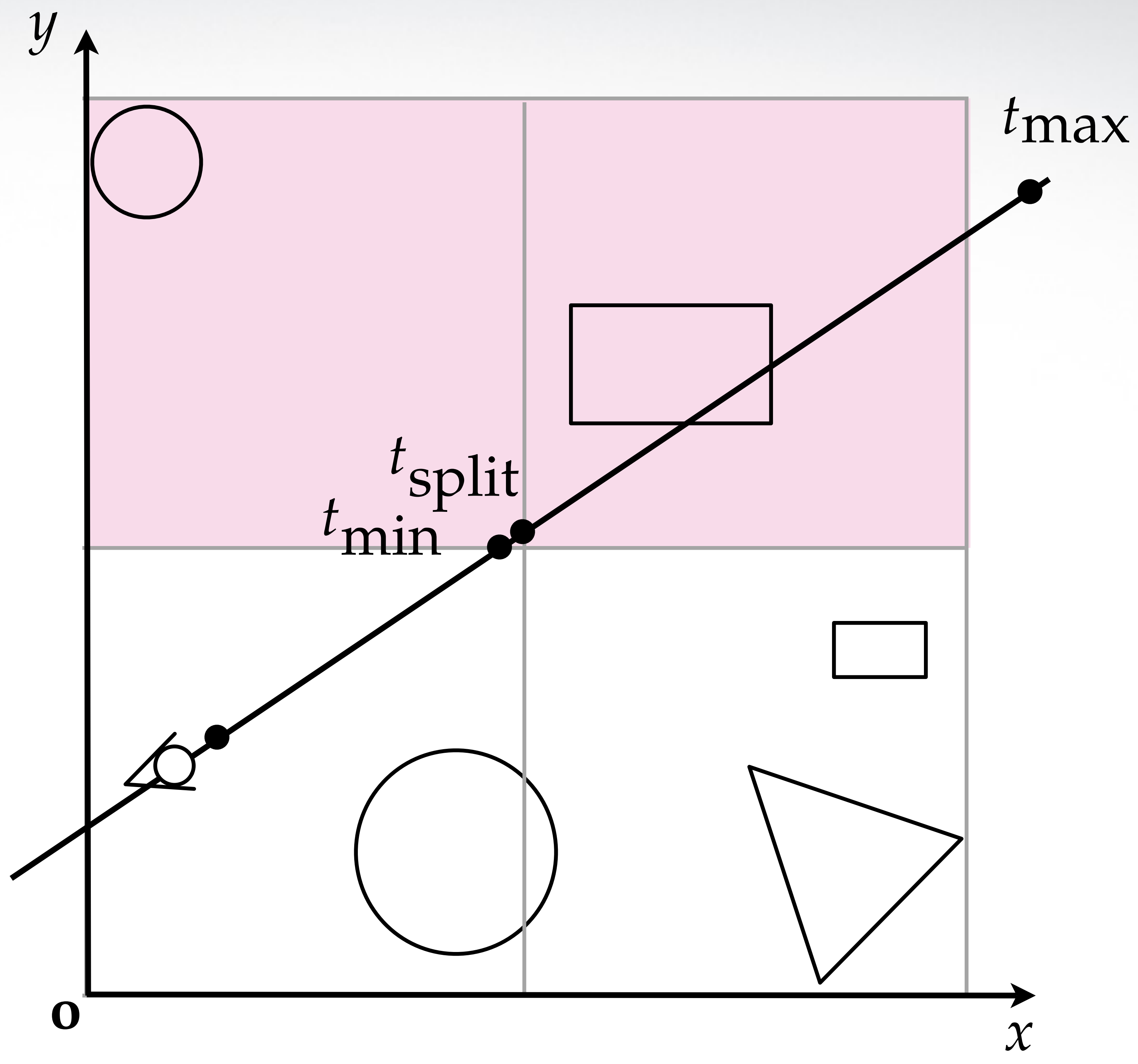


leaf

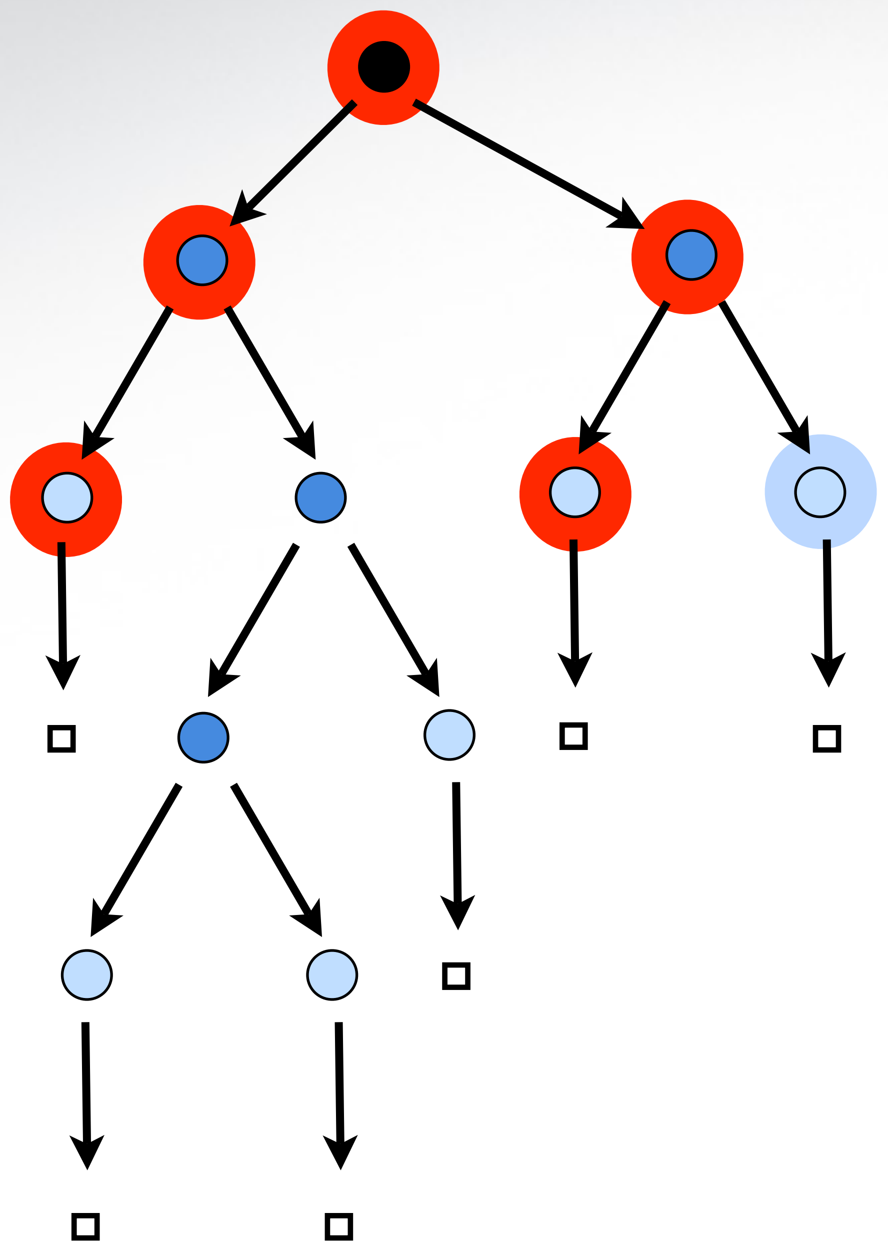


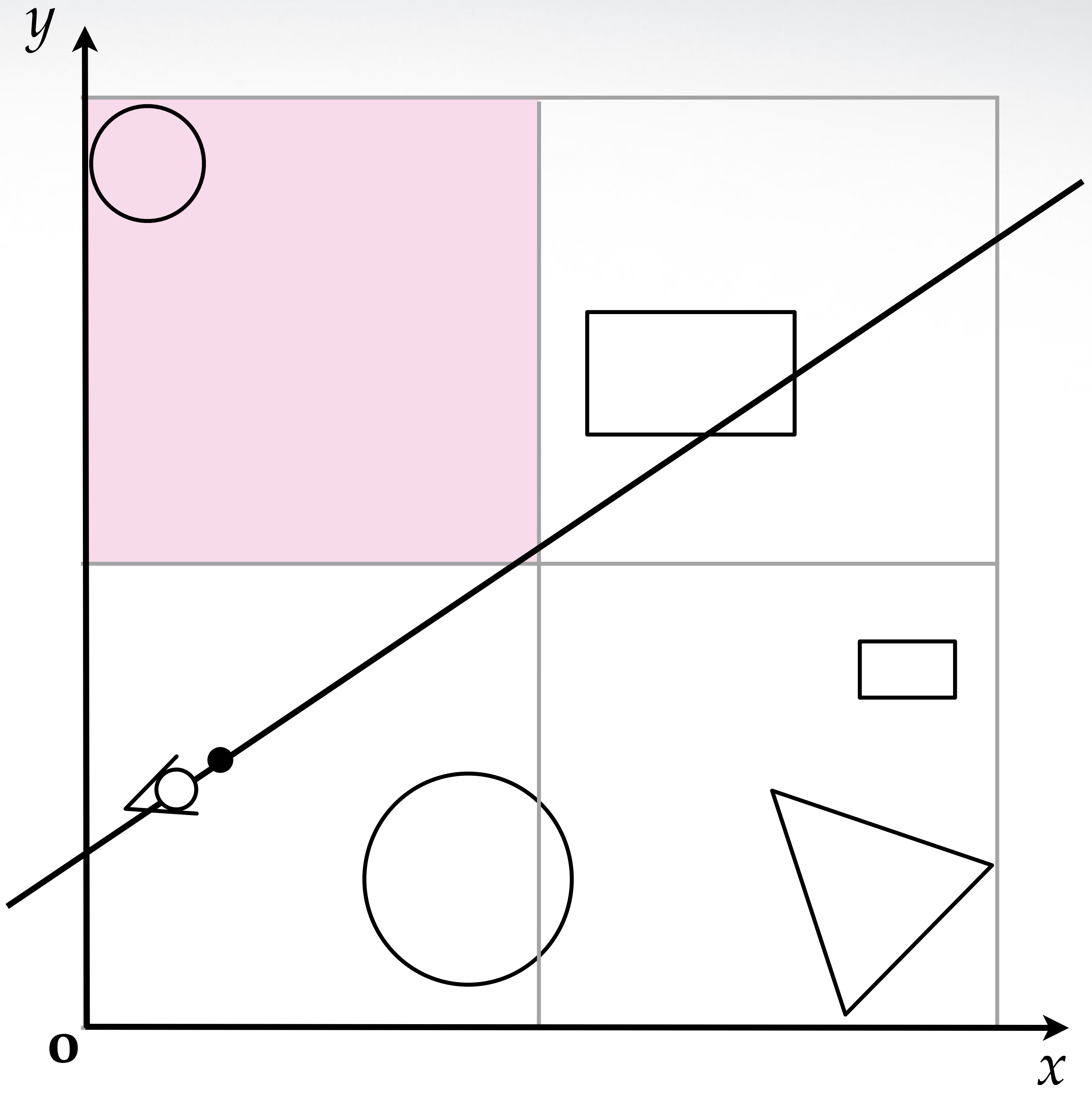
y
o
x



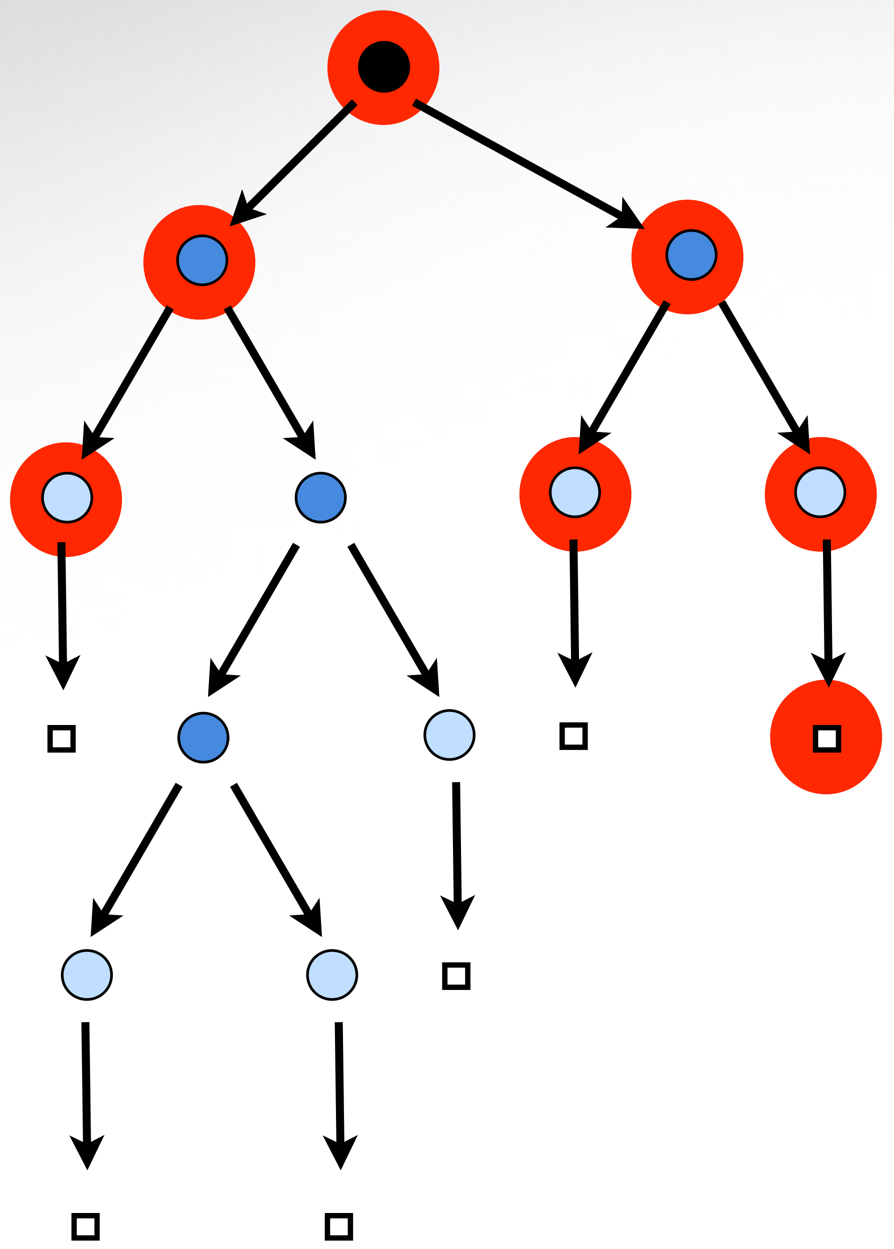


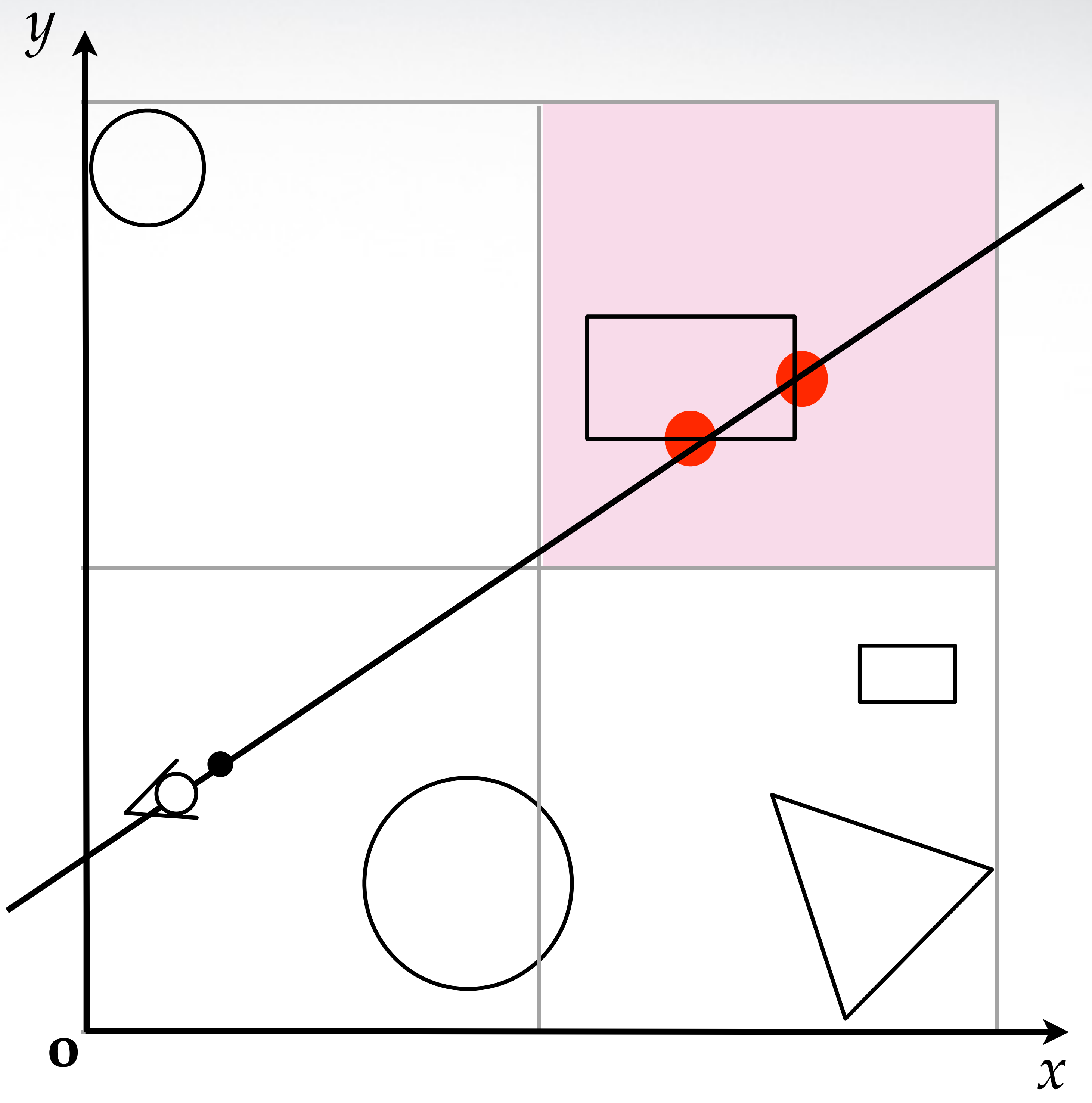
y
t_{max}
t_{split}
t_{min}
$\mathbf{o}$
x




y
o
x

Intersections found!



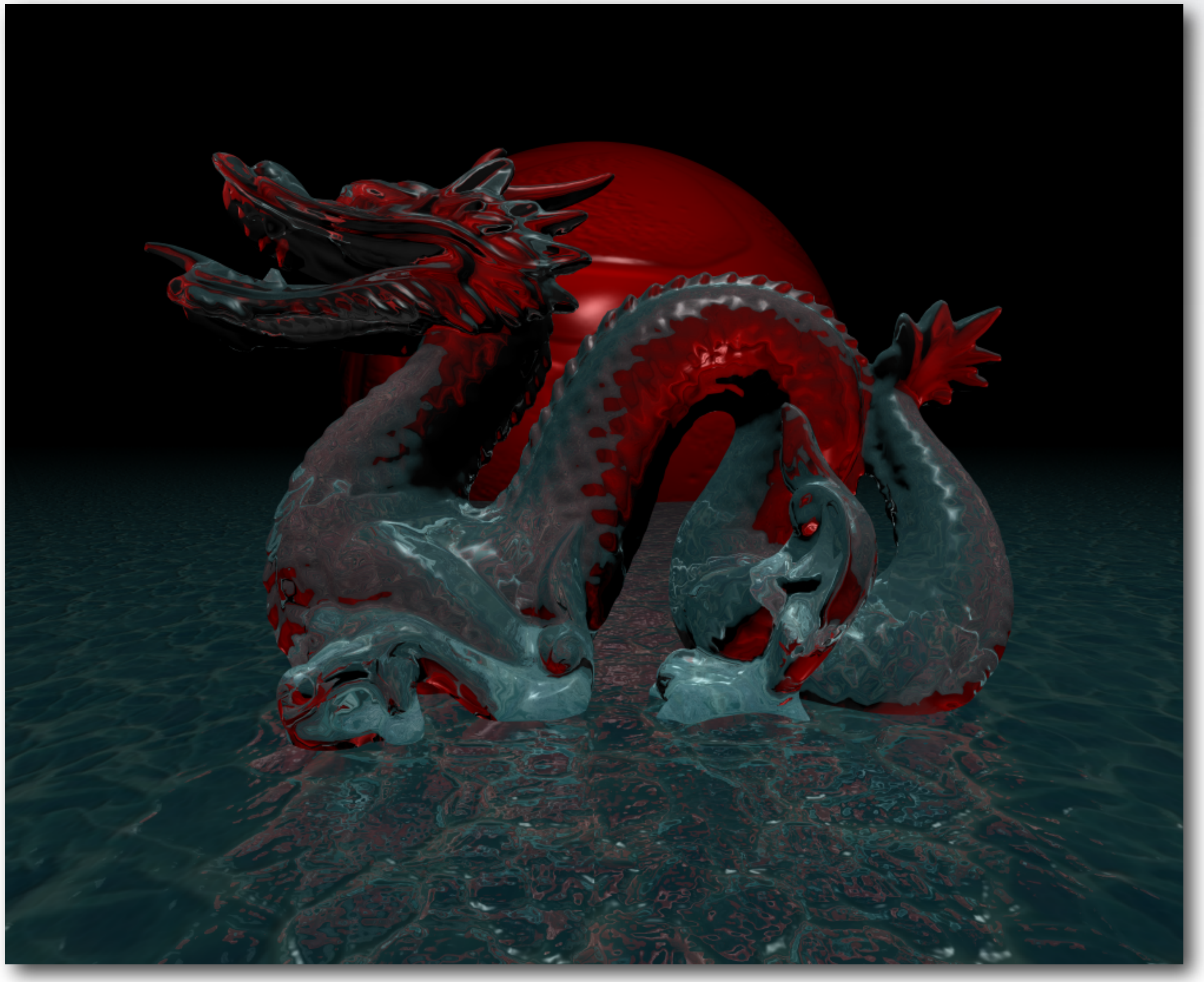


Average Speed-up

$O(n_x\, n_y\, n_t)$

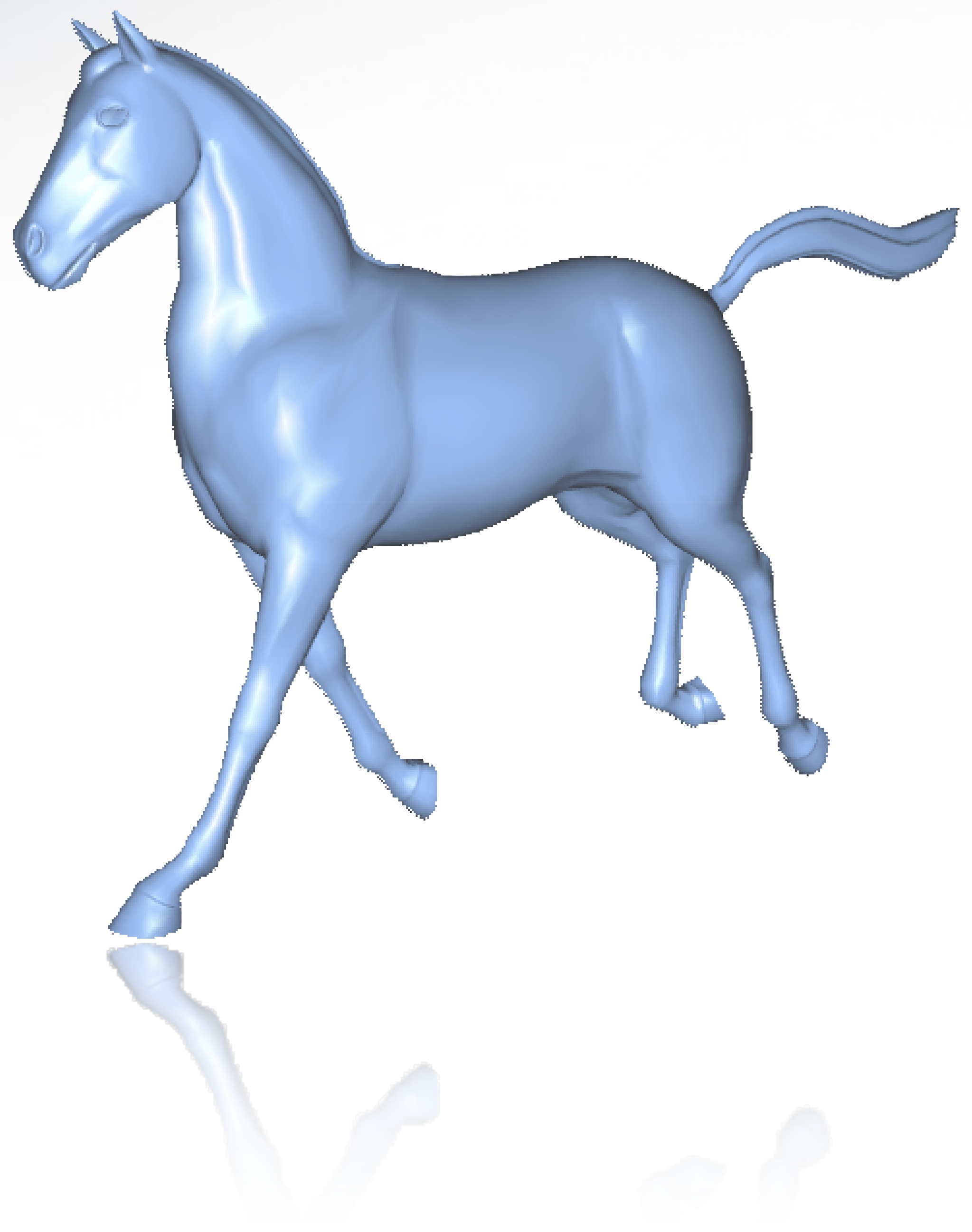
$\Downarrow$

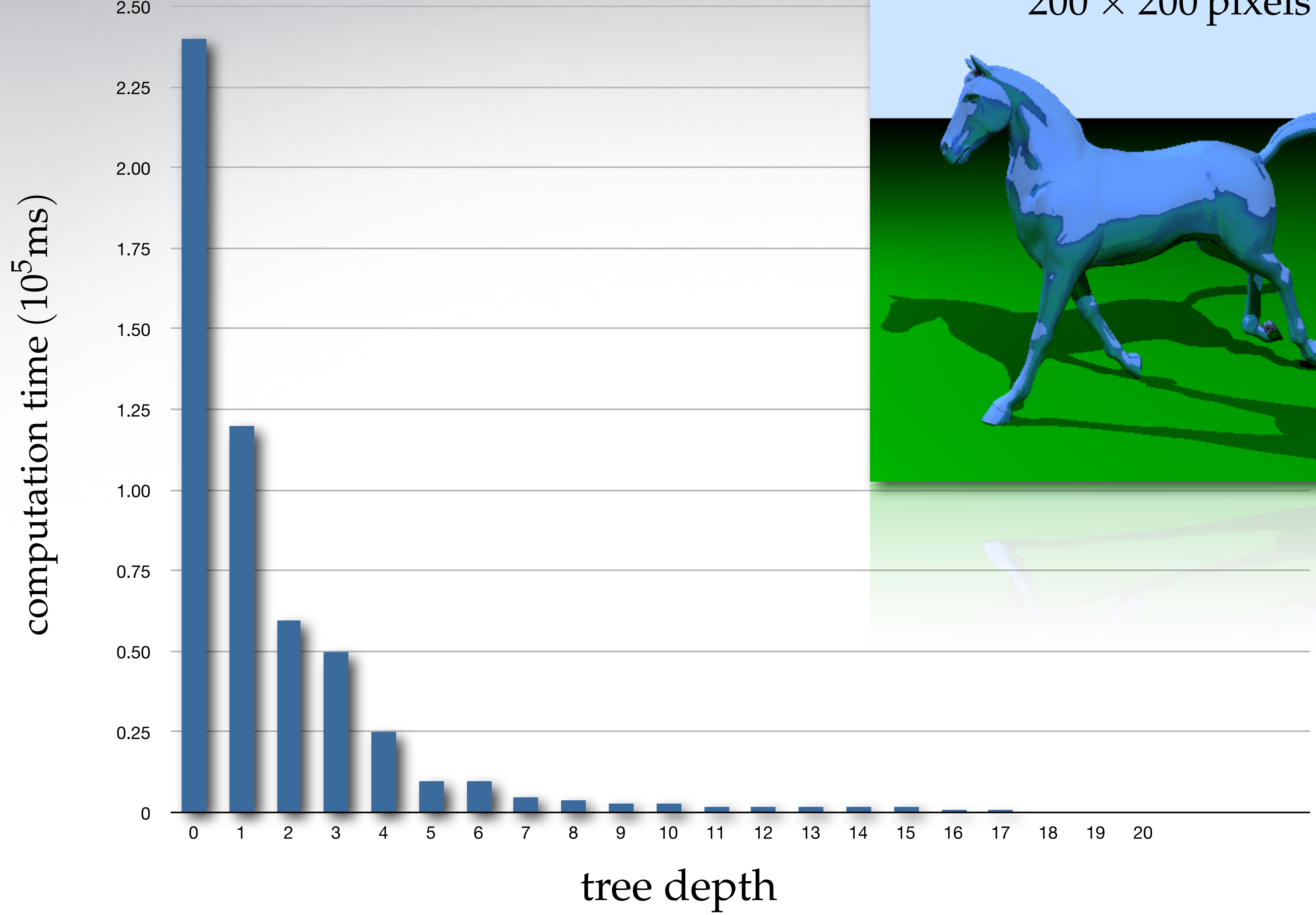
$O(n_x\, n_y \log(n_t))$



in other words...

16843 faces




computation time (10^5ms)
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
tree depth

200 × 200 pixels

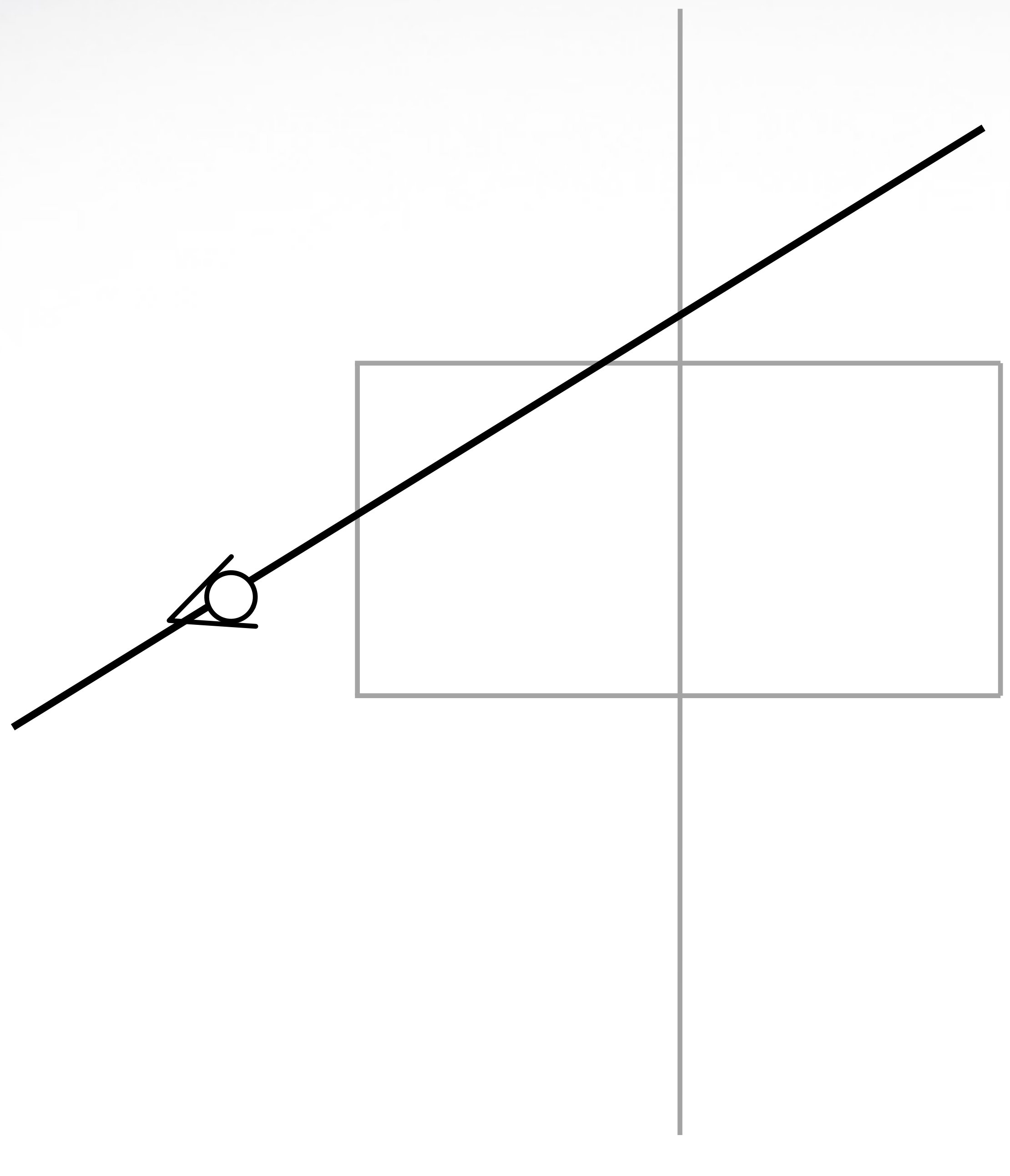
one more thing...

Pseudo code for traversal [Havran 01]

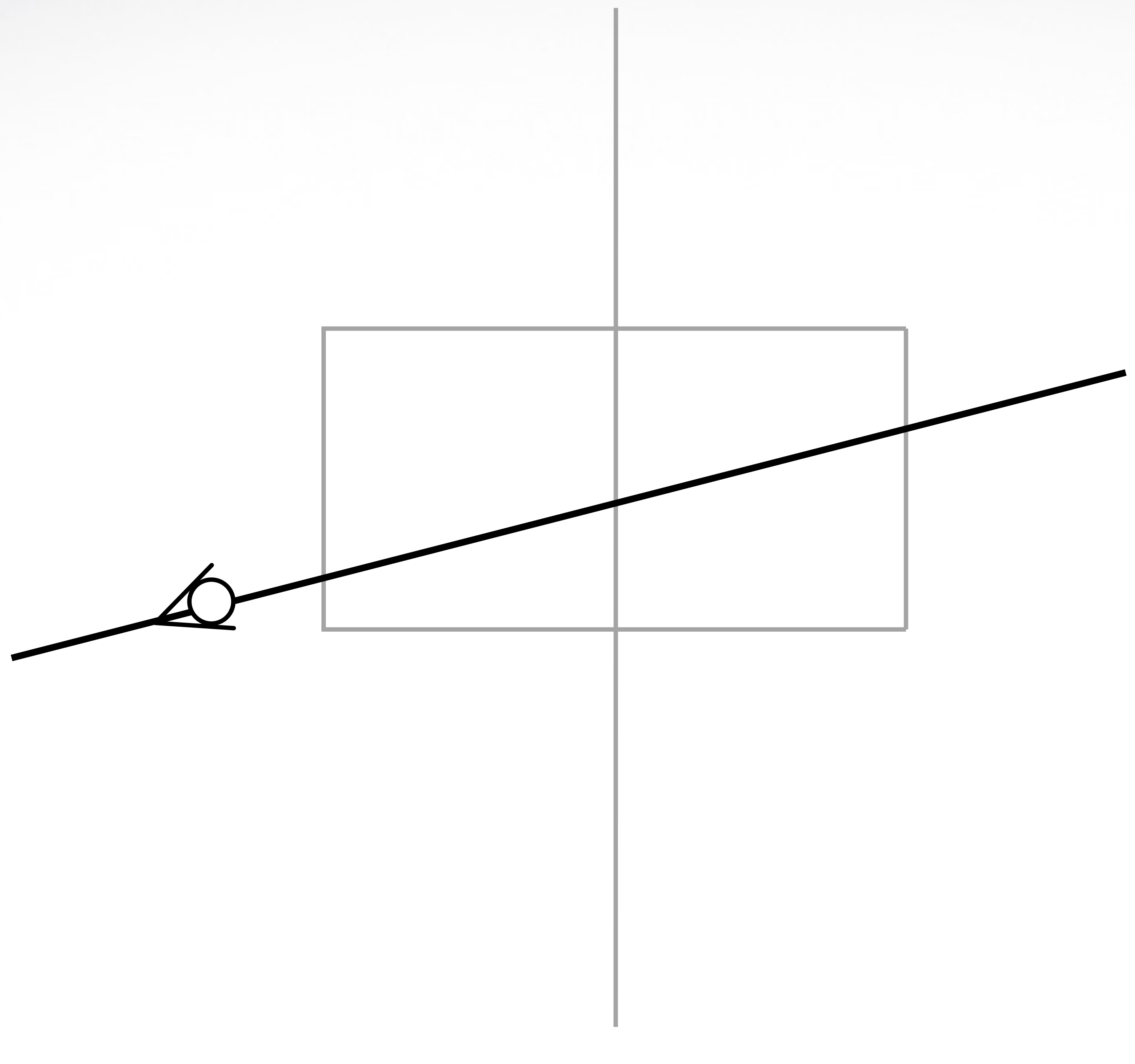
```
function rayTreeIntersection(Ray ray, Node node, double tMin, double tMax)

if node is empty
    return NULL
else
    if node is leaf
        intersect ray with all primitives in node
        return closest primitive
    else
        compute tSplit
        nearNode is child of node of near side
        farNode is child of node of far side
        if (tSplit > tMax)
            return rayTreeIntersection(ray, nearNode, tMin, tMax)
        else if (tSplit < tMin)
            return rayTreeIntersection(ray, node containing tMin and tMax, tMin, tMax)
        else
            intersectedPrimitive = rayTreeIntersection(ray, nearNode, tMin, tSplit)
                if (intersectedPrimitive not NULL)
                    return intersectedPrimitive
                else
                    return rayTreeIntersection(ray, farNode, tSplit, tMax)
```

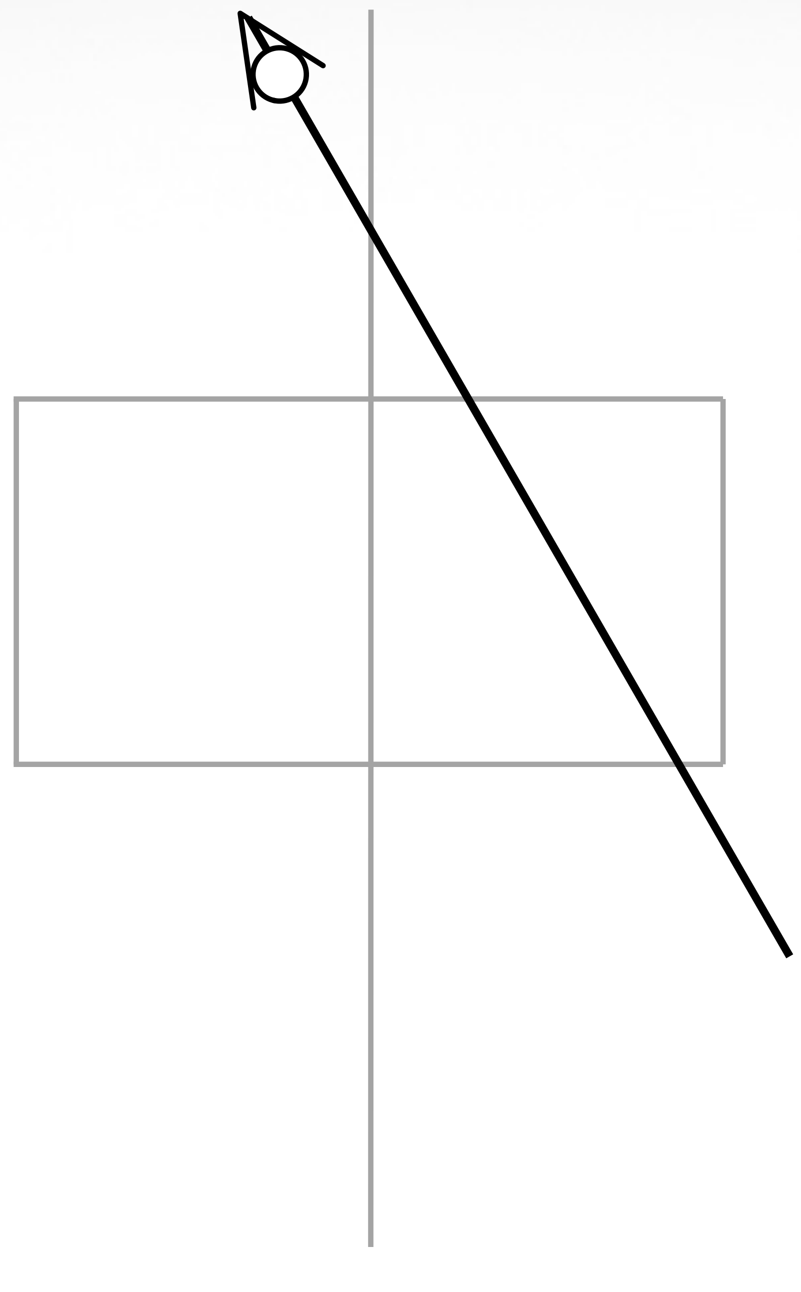
Hint: case distinctions



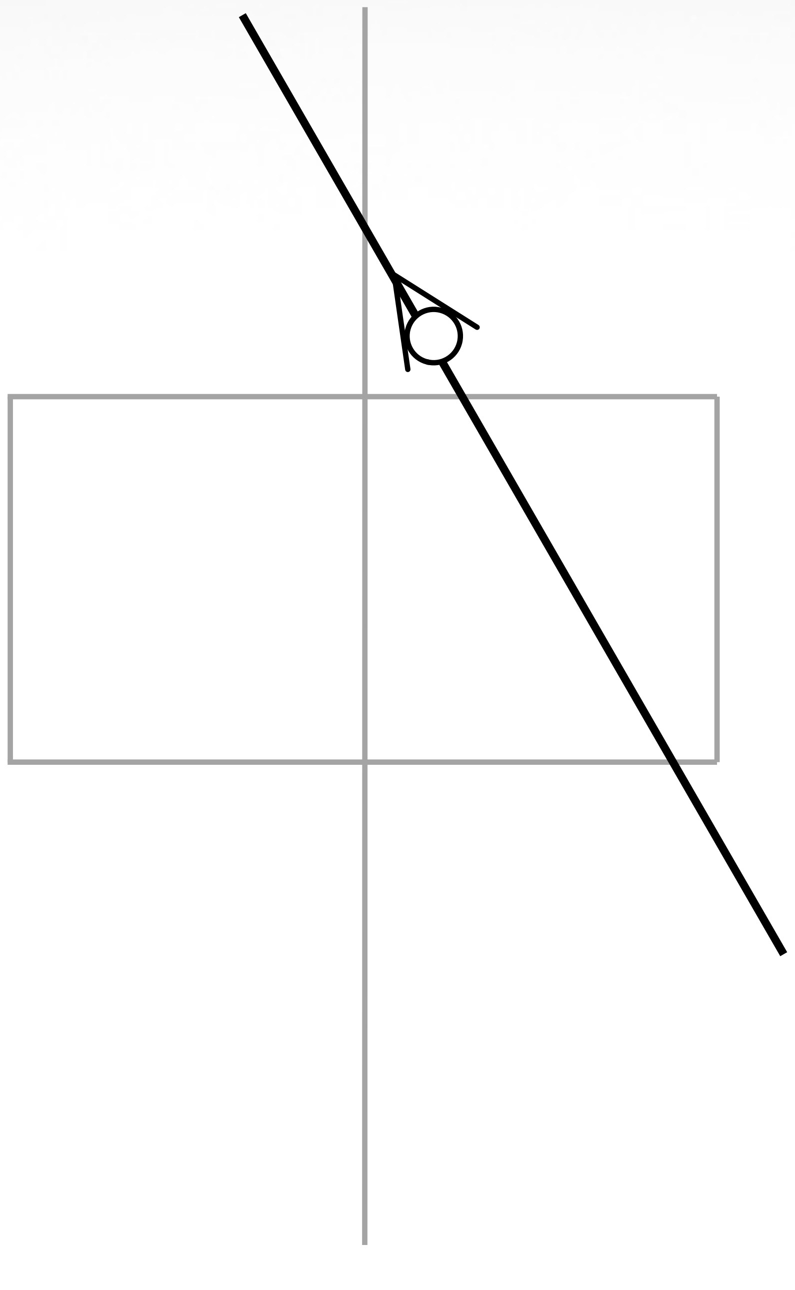
Hint: case distinctions



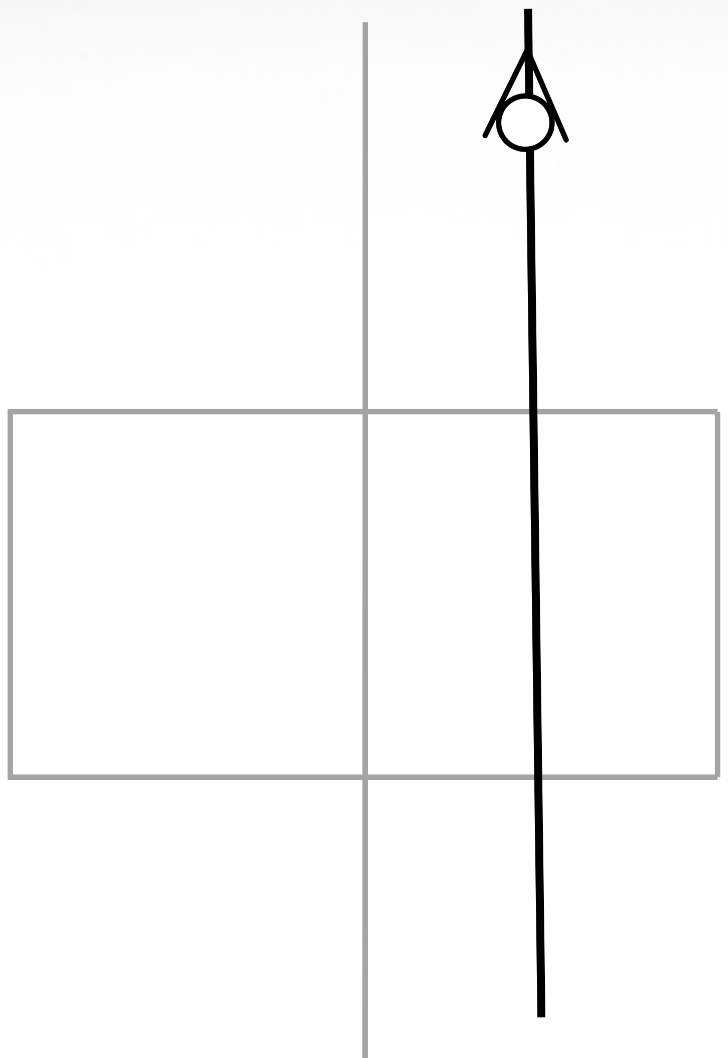
Hint: case distinctions



Hint: case distinctions



Hint: case distinctions



Further Readings

- Heuristic Ray Shooting Algorithms [Havran 2001]
- Realtime Ray Tracing [Ingo Wald 2004]
- Multidimensional Binary Search Trees Used for Associative Searching [Bentley 1975]

?

hao@inf.ethz.ch