Acceleration Structures

Computer Graphics
Instructor: Sungkil Lee

Overview

Ray Tracing Cost

- **Where does it spend most of the computation in ray tracer?**

```
color trace( ray i, int step )
{
    if( step > max ) return background_color;

    status s = intersect(i,q); // q: output ray
    if(s==light_source)    return light_source_color;
    if(s==no_intersection) return background_color;

    vec3 n = get_face_normal(q); // do not use a vertex normal
    vec3 r = reflect(q,n);
    vec3 t = refract(q,n);

    color local = phong_shade(q,n,r);
    color reflected = trace(ray(q,r), step+1);
    color refracted = trace(ray(q,t), step+1);

    return local + reflected + refracted;
}
```

Ray Tracing Cost

- **For each ray, the cost is linear in the number of objects in the scene**
  - Complexity $O(n)$ per ray
  - Total cost = objects*rays

- **Example**
  - at 1024x1024 resolution, trace 1000 triangles
  - 10^9 ray-triangle intersections (only for primary-ray intersection) !!!

Acceleration Structures: Overview

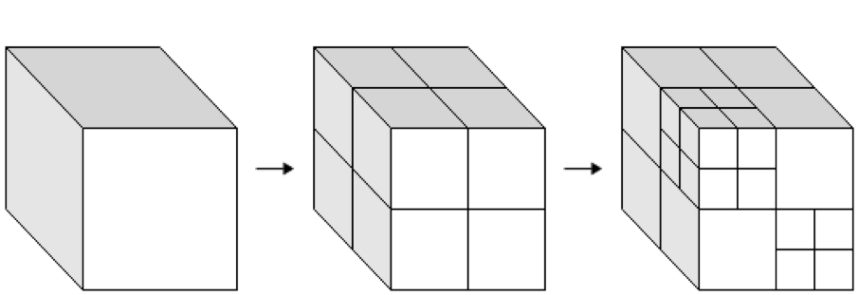
- **Goal: sub-linear complexity**
  - Don't touch every single object
  - Build a hierarchical tree for sublinear performance
  - A binary tree is one of the good natural candidates.

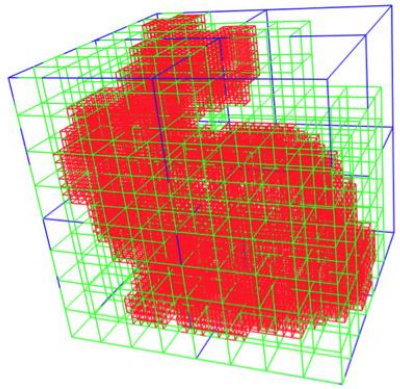
- **Two fundamental approaches**
  - Object subdivision
  - Spatial subdivision

Two fundamental approaches

- **Object subdivision:**
  - hierarchies of groups of objects
  - e.g., Bounding Volume Hierarchy (BVH)

- **Spatial subdivision:**
  - Regular spatial partitioning: quadtree (2D), octree (3D)

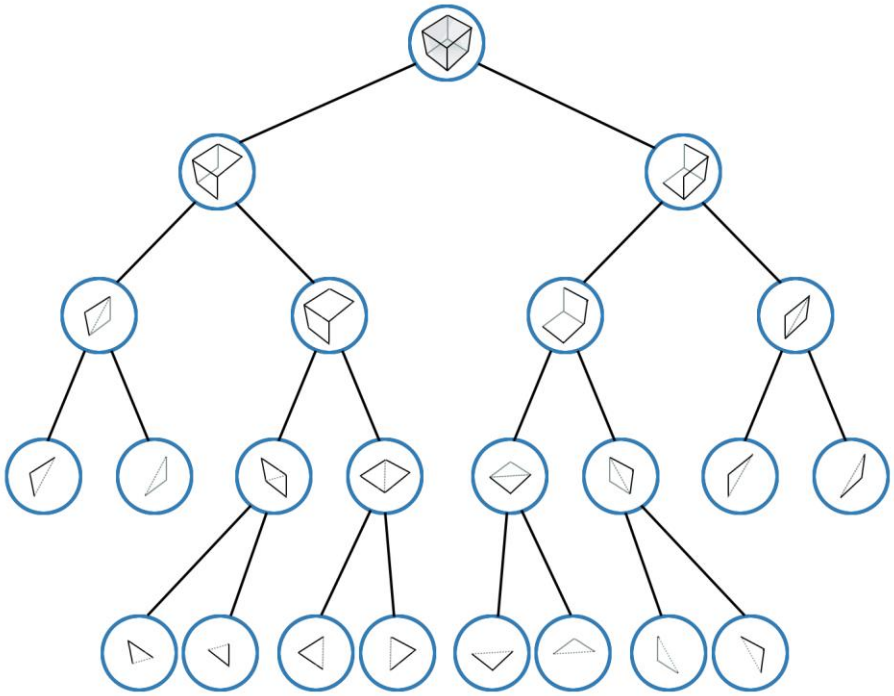




Example of an octree

Two fundamental approaches

- **Spatial subdivision:**
  - Irregular spatial partitioning:
    - Examples: k-D tree, Binary Space Partitioning (BSP) tree
    - BSP tree is common in the intersection test with terrain in games.
    - k-D tree is a special case of BSP tree, which aligns the split along axes.



Conceptual illustration of BSP tree

Bounding Volume Hierarchy (BVH)

Bounding Volume Hierarchy

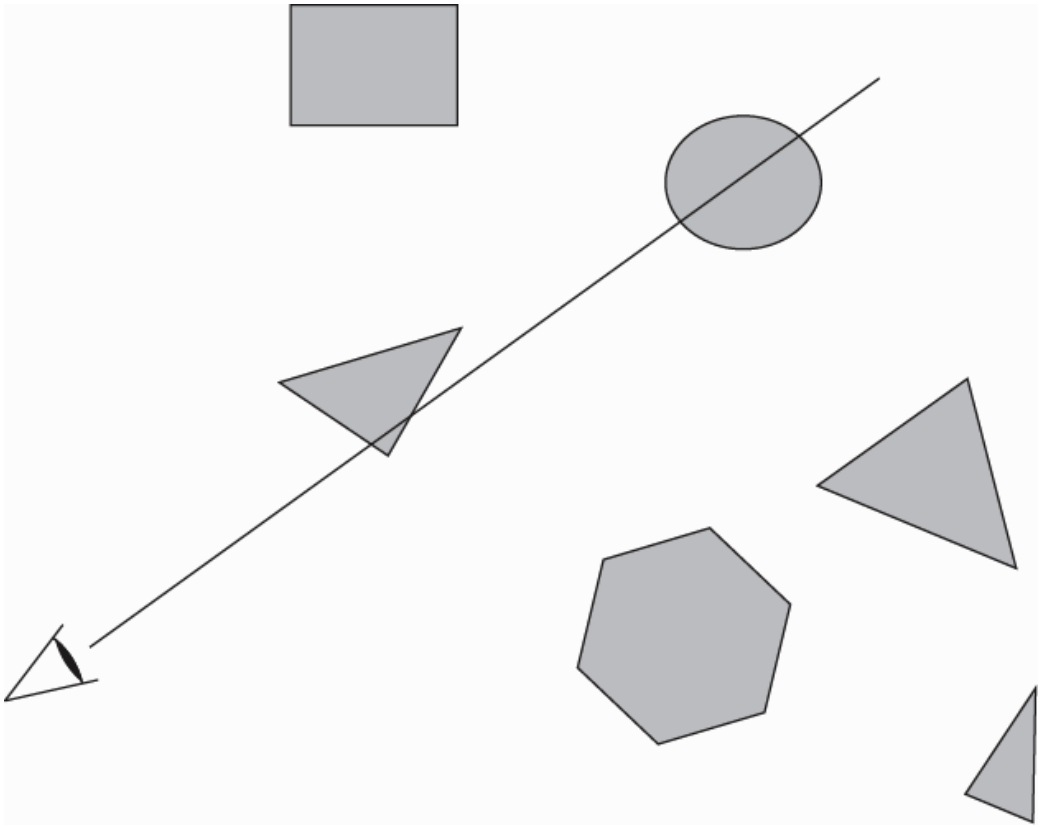
- **Hierarchies of groups of objects**
  - Groups are represented by aggregate objects with bounding volumes

- **Logarithmic complexity:** $O(\log n)$
  - BVH is a binary tree.

- **Bounding volumes**
  - Bounding boxes, spheres, anything (?)

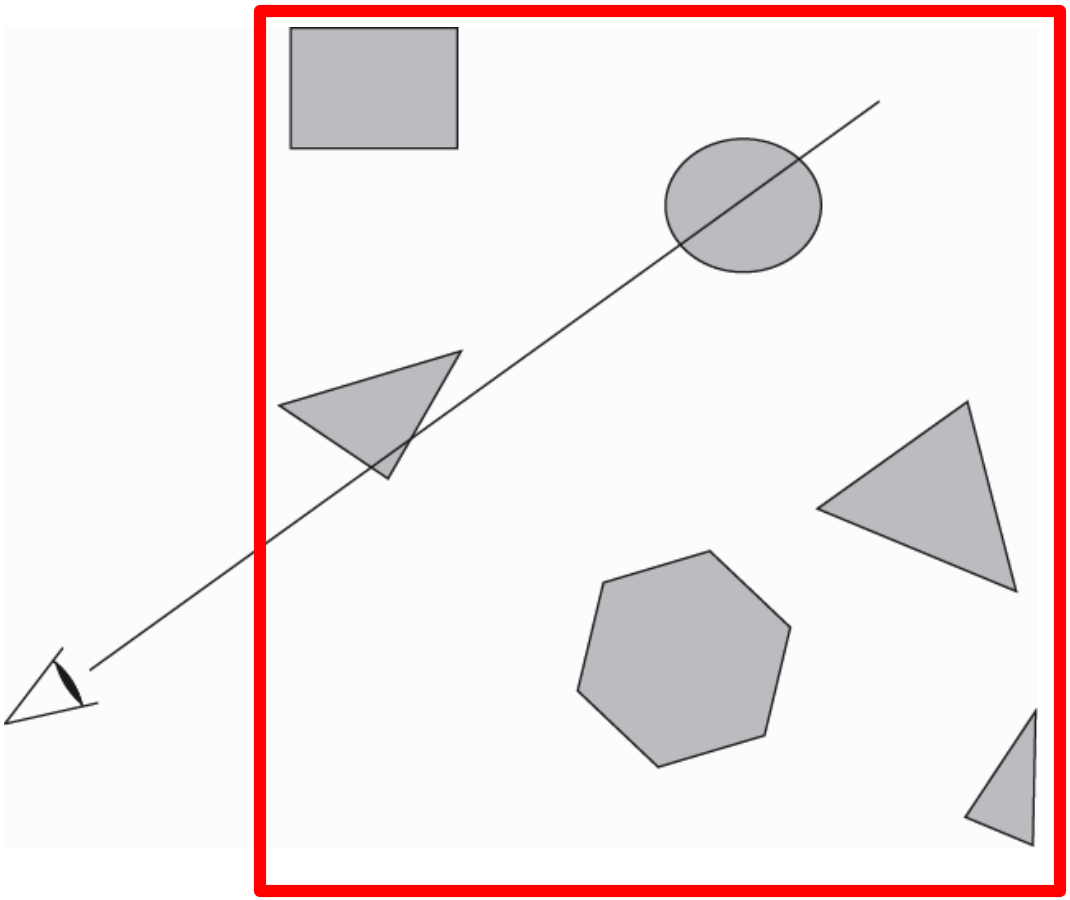
Bounding Volume Hierarchy

- **Hierarchies of groups of objects**



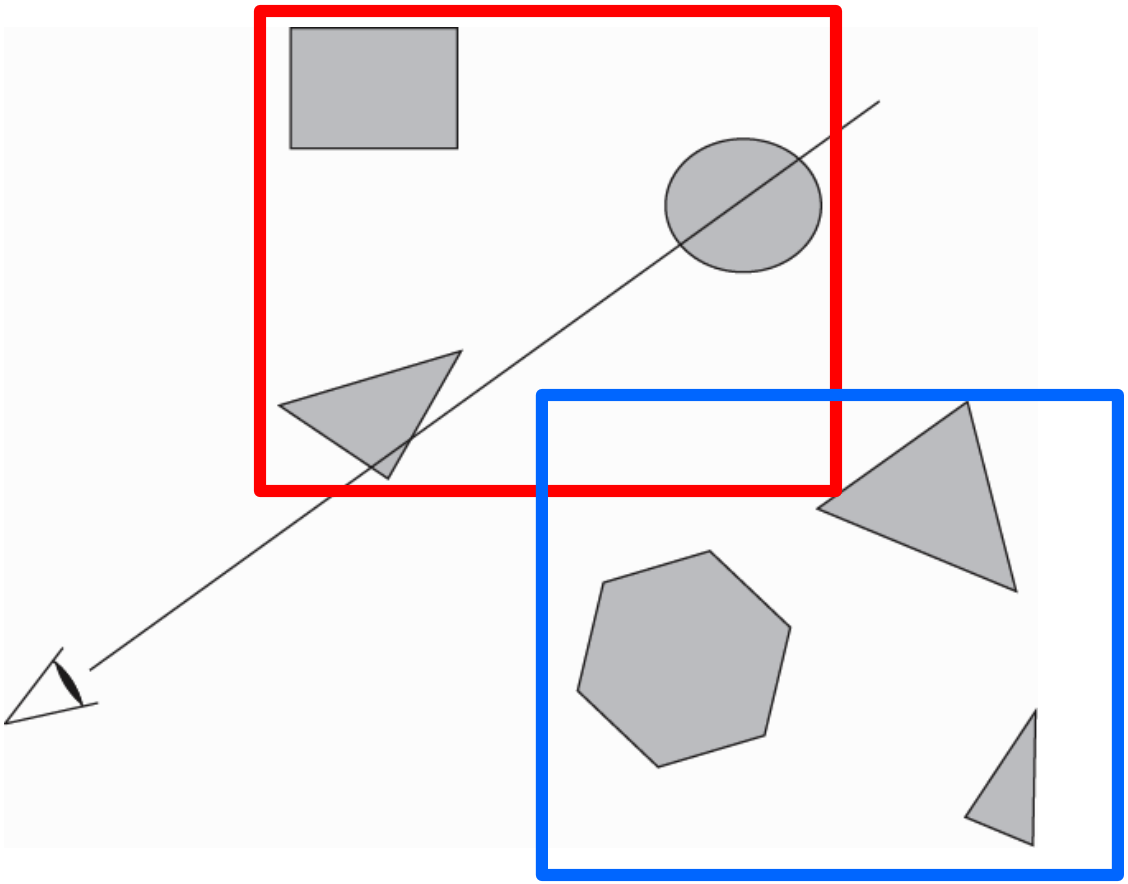
Bounding Volume Hierarchy

- **Hierarchies of groups of objects**



Bounding Volume Hierarchy

- **Hierarchies of groups of objects**



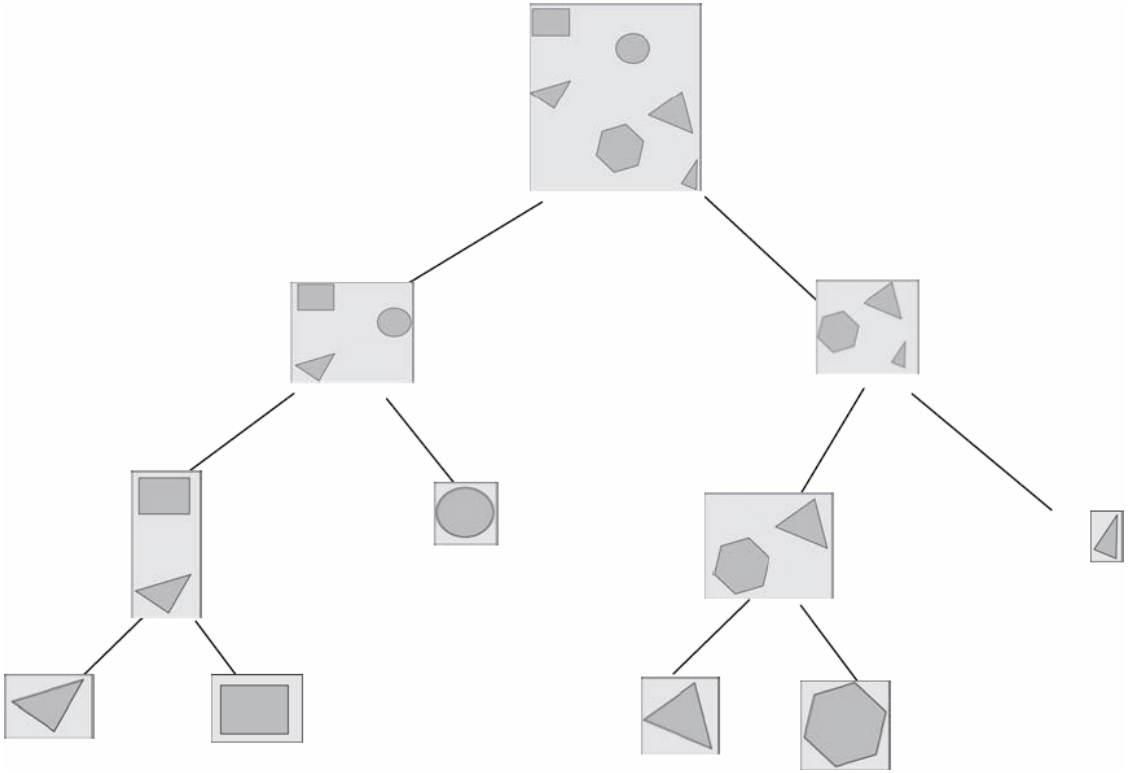
Bounding Volume Hierarchy

- **All objects in a subtree are within the bounds of its root**
  - Not all objects within the bounding volume of a node need to be in its subtree
  - Subtrees can overlap spatially, and are not ordered in any way.
    - c.f., in space subdivision, subtrees can be ordered.

- **Intersection handling**
  - If a node is not intersected by a ray, none of the objects in its subtree are intersected.
  - If a node is intersected, all children have to be tested for intersections

BVH Construction

- **Partitioning objects along coordinate axes**
- **Binary tree**



BVH Construction: How to Split

- **Where to place the split plane?**
  - Locally minimize the cost function:

$$t_t + P_B \sum_i t_i(b_i) + P_A \sum_j t_j(a_j)\,,$$

    - t_x: cost to traverse the interior node
    - P_B, P_A: probabilities to hit children below, above split

Surface-Area Heuristic (SAH)

- **SAH**
  - A simple yet powerful greedy optimization strategy
    - Automatic creation of object hierarchies for ray tracing (Goldsmith, 1987)
  - Probability to hit child

$$p_A = P(A|root) = \frac{S_A}{S_{root}}$$

    - S_A, S_{root}: the surface areas of the child and root nodes

  - Among multiple choices, we can choose the split having minimum cost with the probability above.

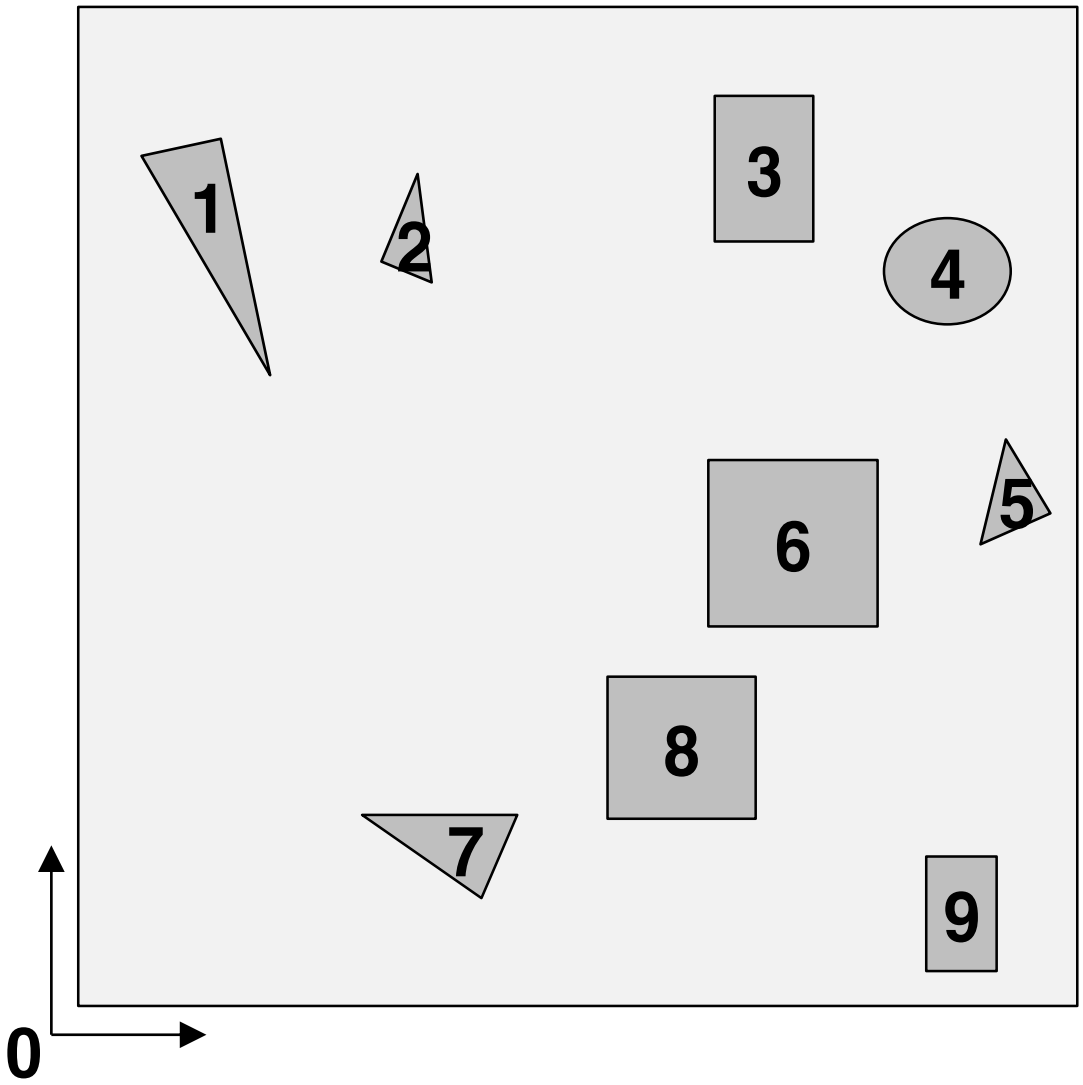
- **The SAH can be used for other acceleration structures**
  - E.g., K-D trees

K-D Trees

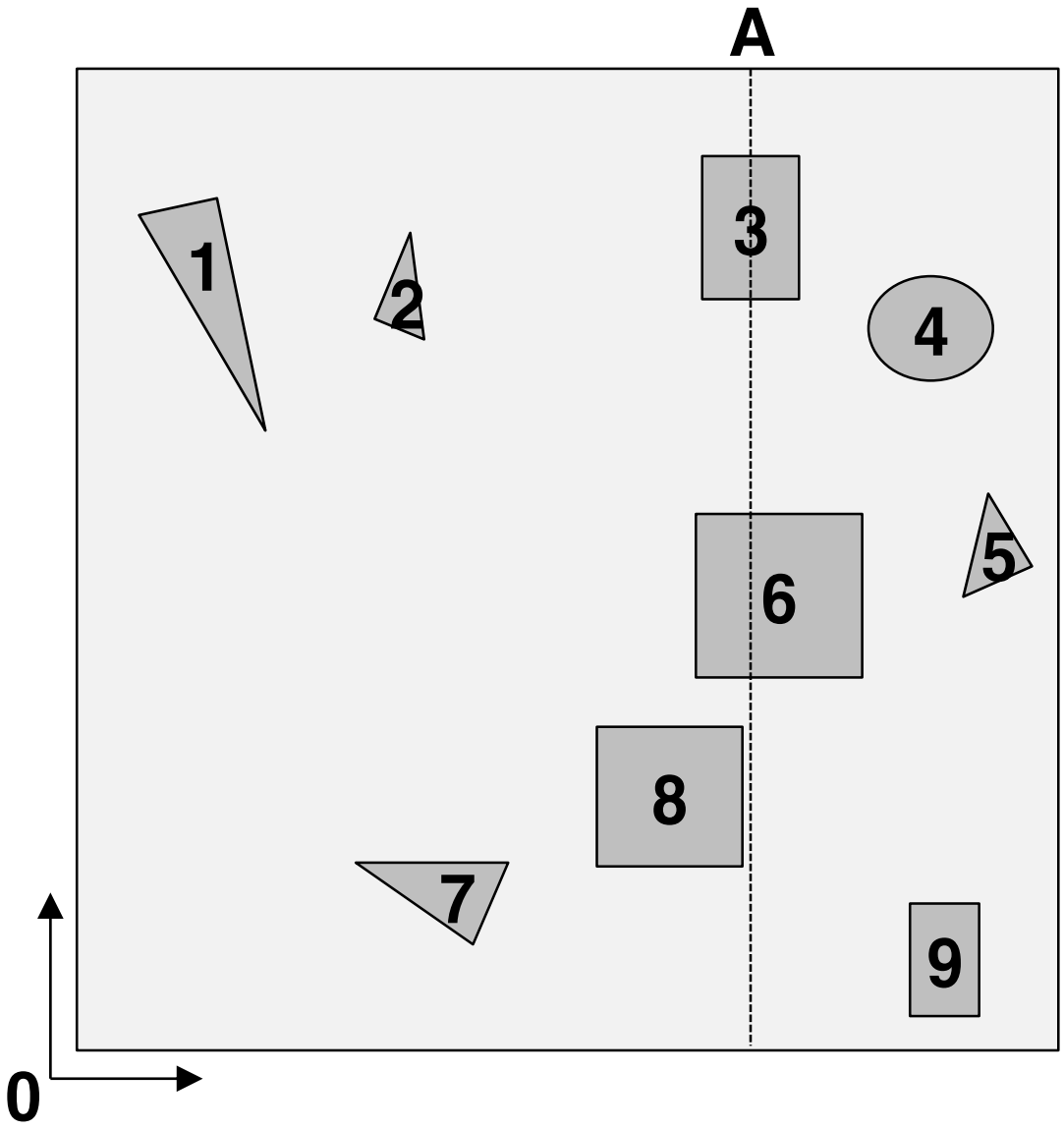
K-D Trees

- **Binary Space Partitioning (BSP) trees:**
  - Recursively divide space into two parts
- **K-D trees**
  - a special case of BSP trees whose dividing planes are axis-aligned.
- **Example:**
  - Subdivide until fewer than 3 objects in node
  - Left child below split plane
  - Right child above split plane

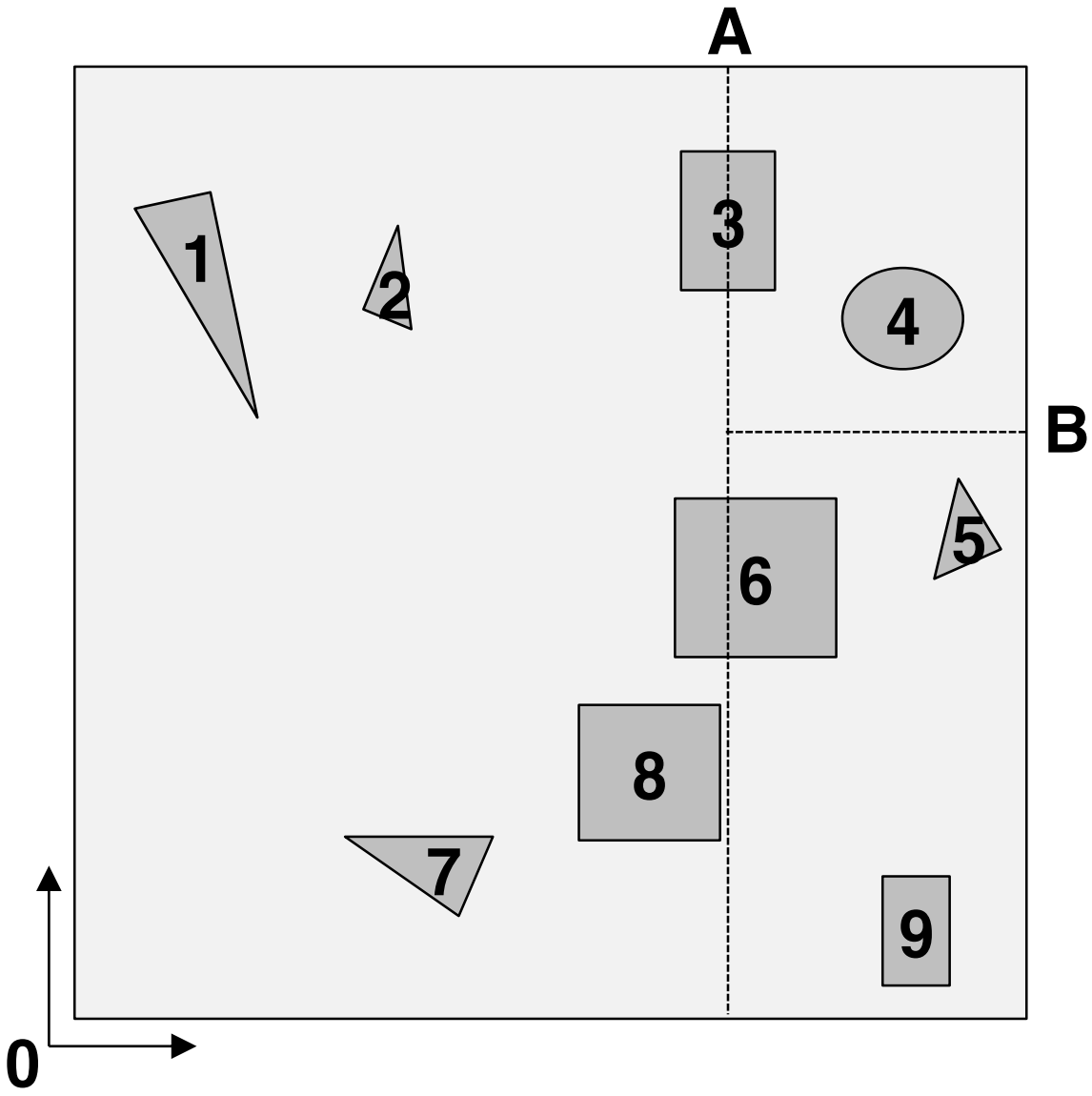
K-D Tree Construction

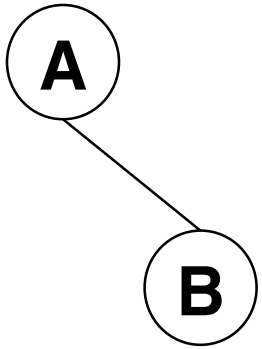


K-D Tree Construction

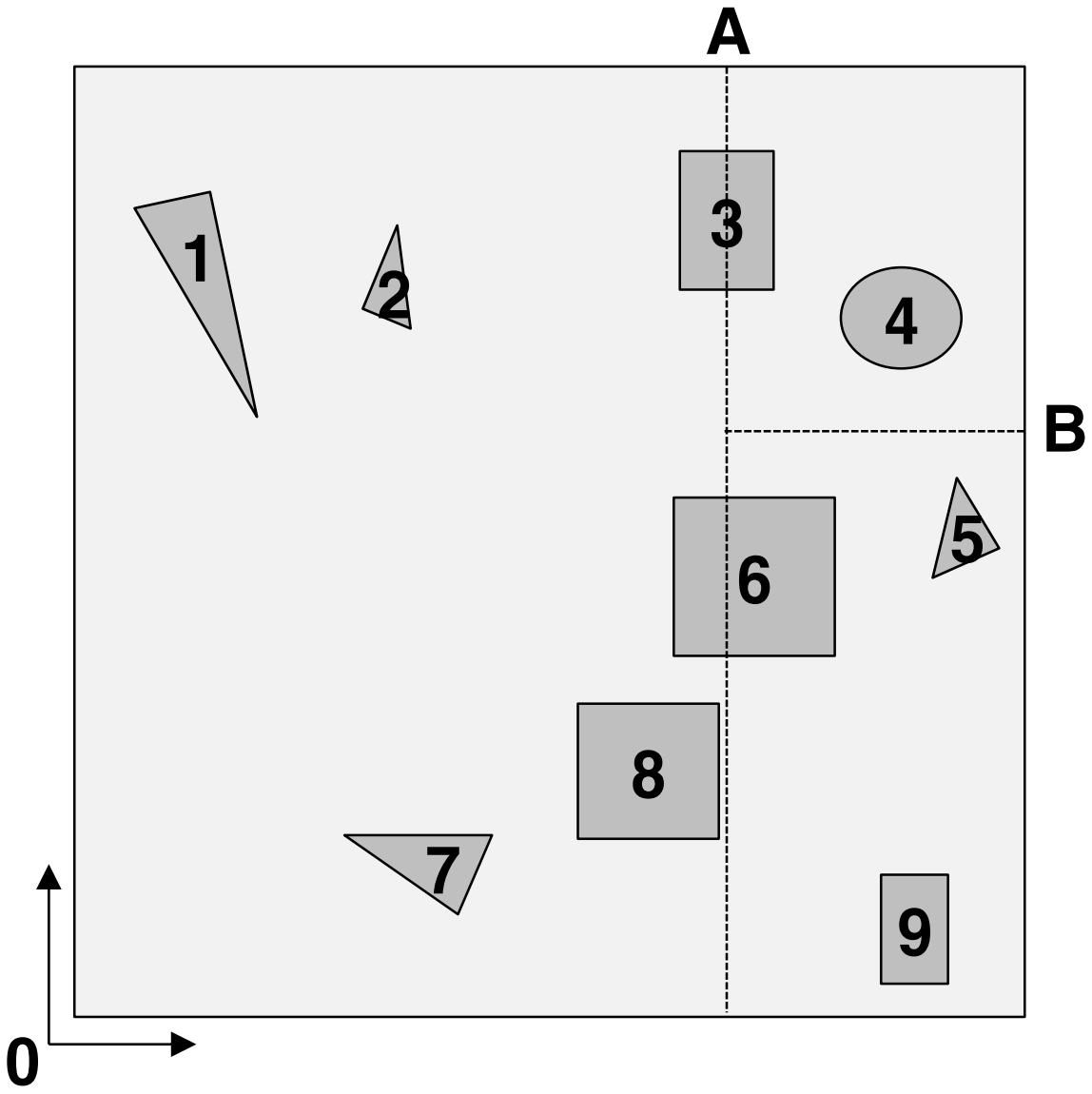


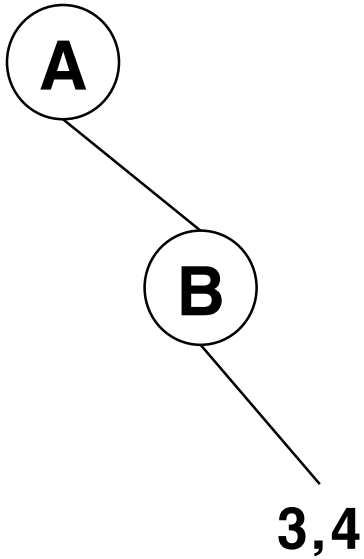
K-D Tree Construction



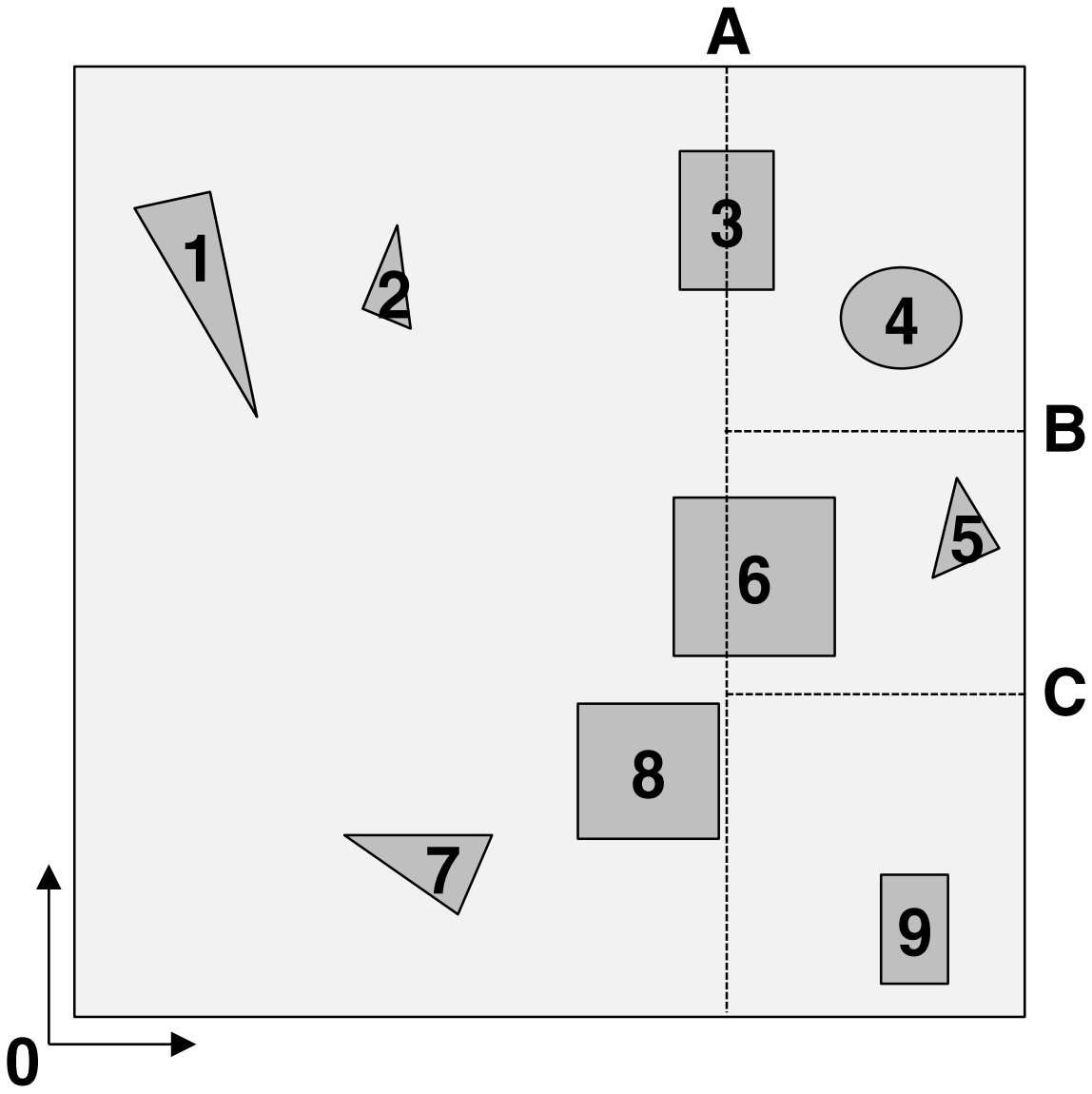


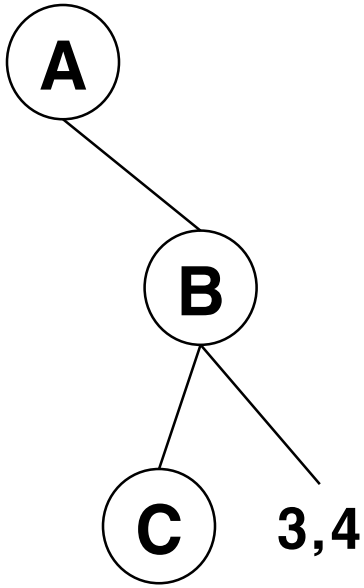
K-D Tree Construction



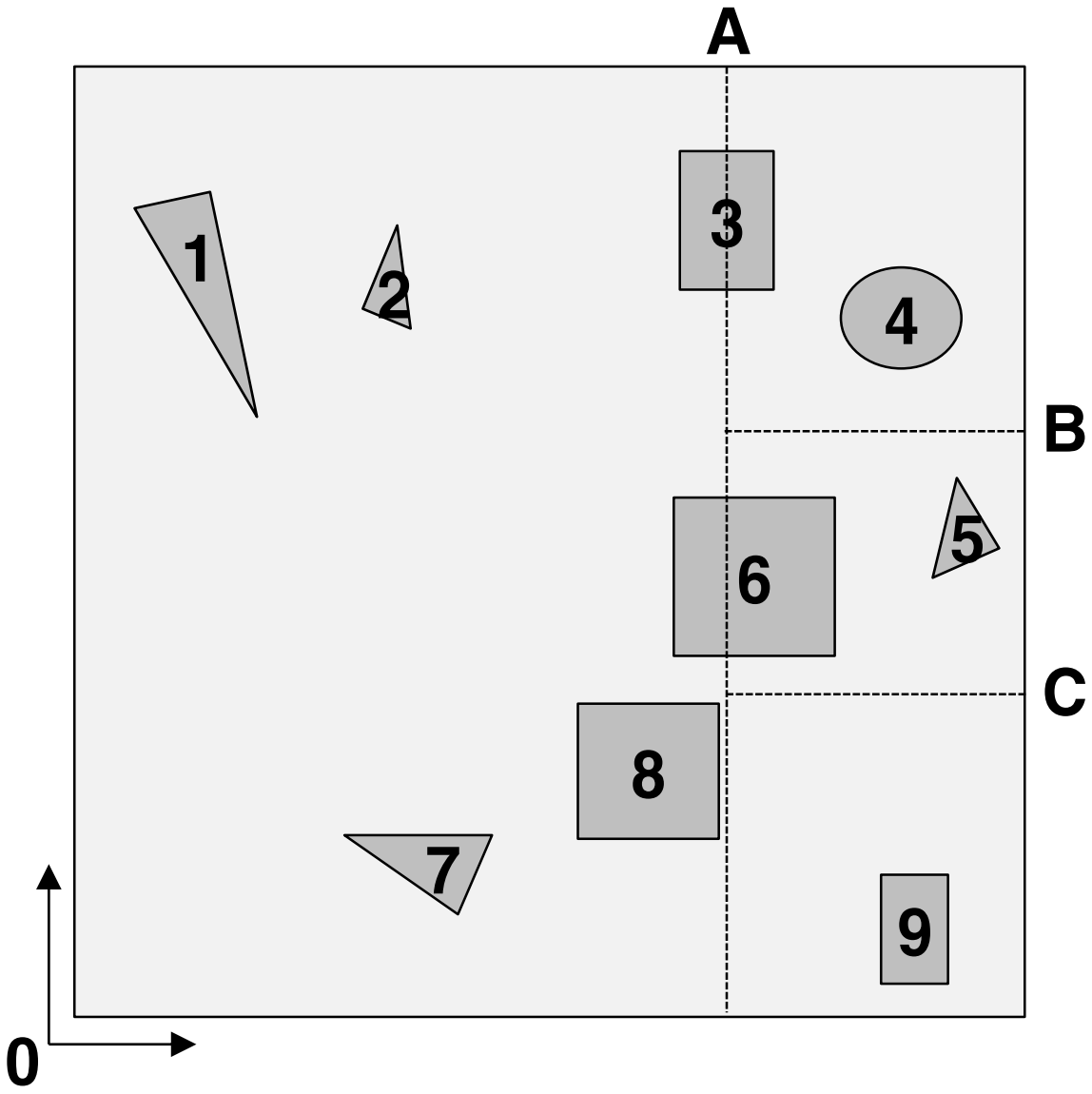


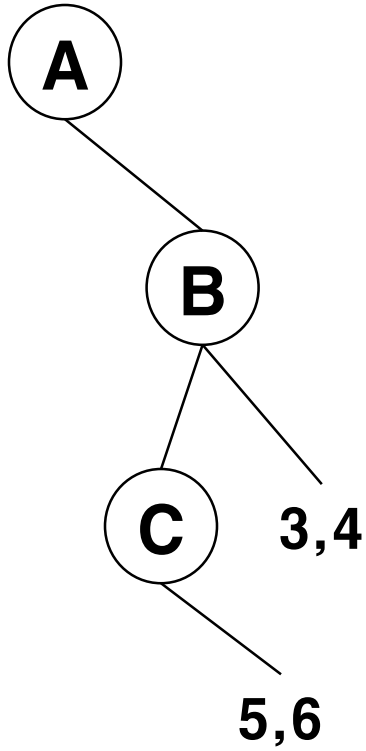
K-D Tree Construction



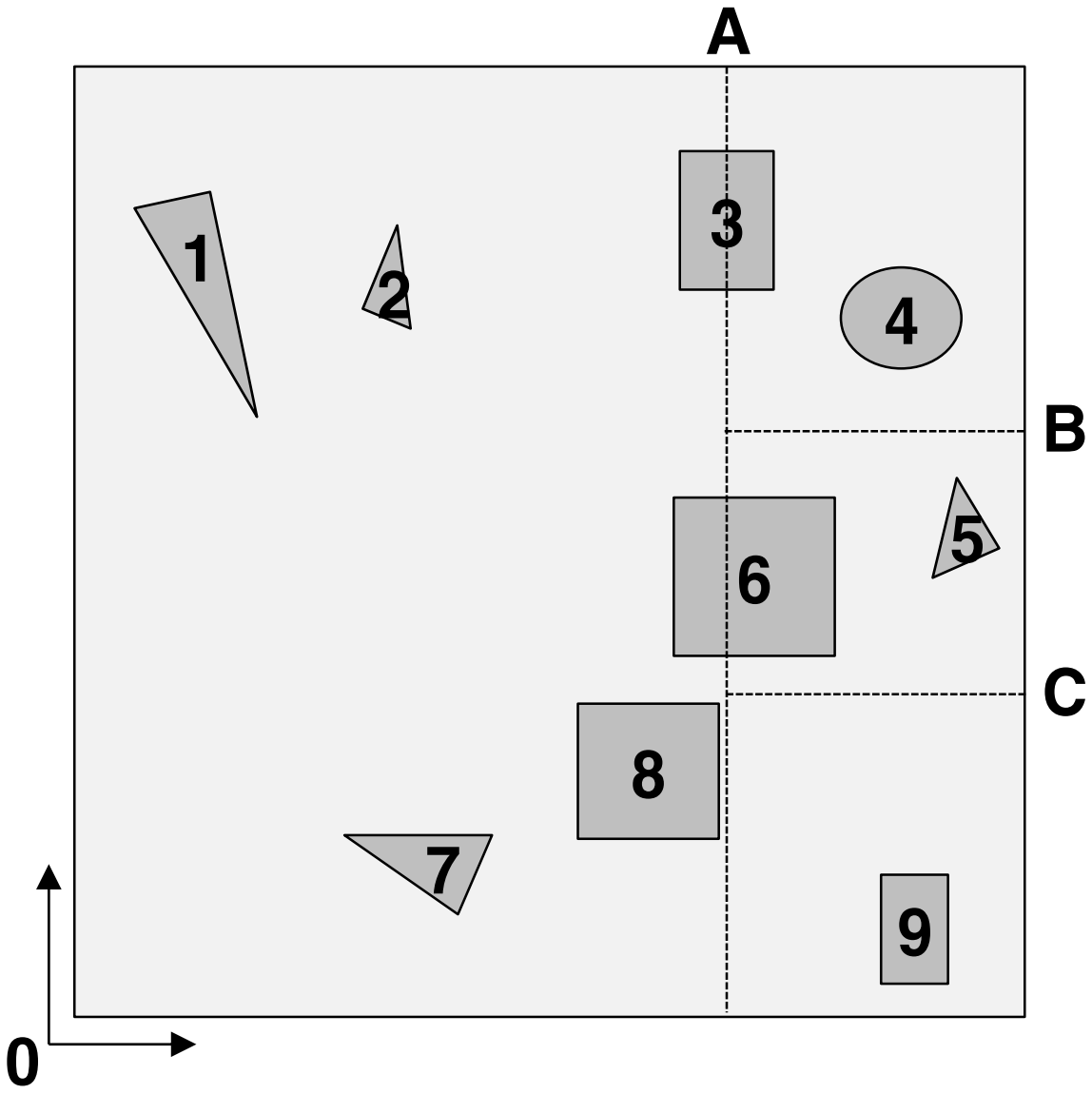


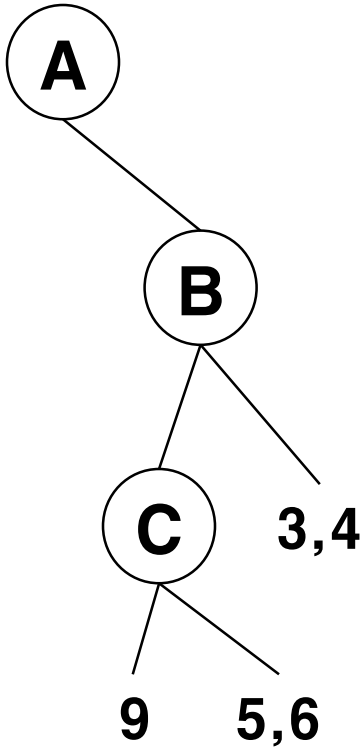
K-D Tree Construction



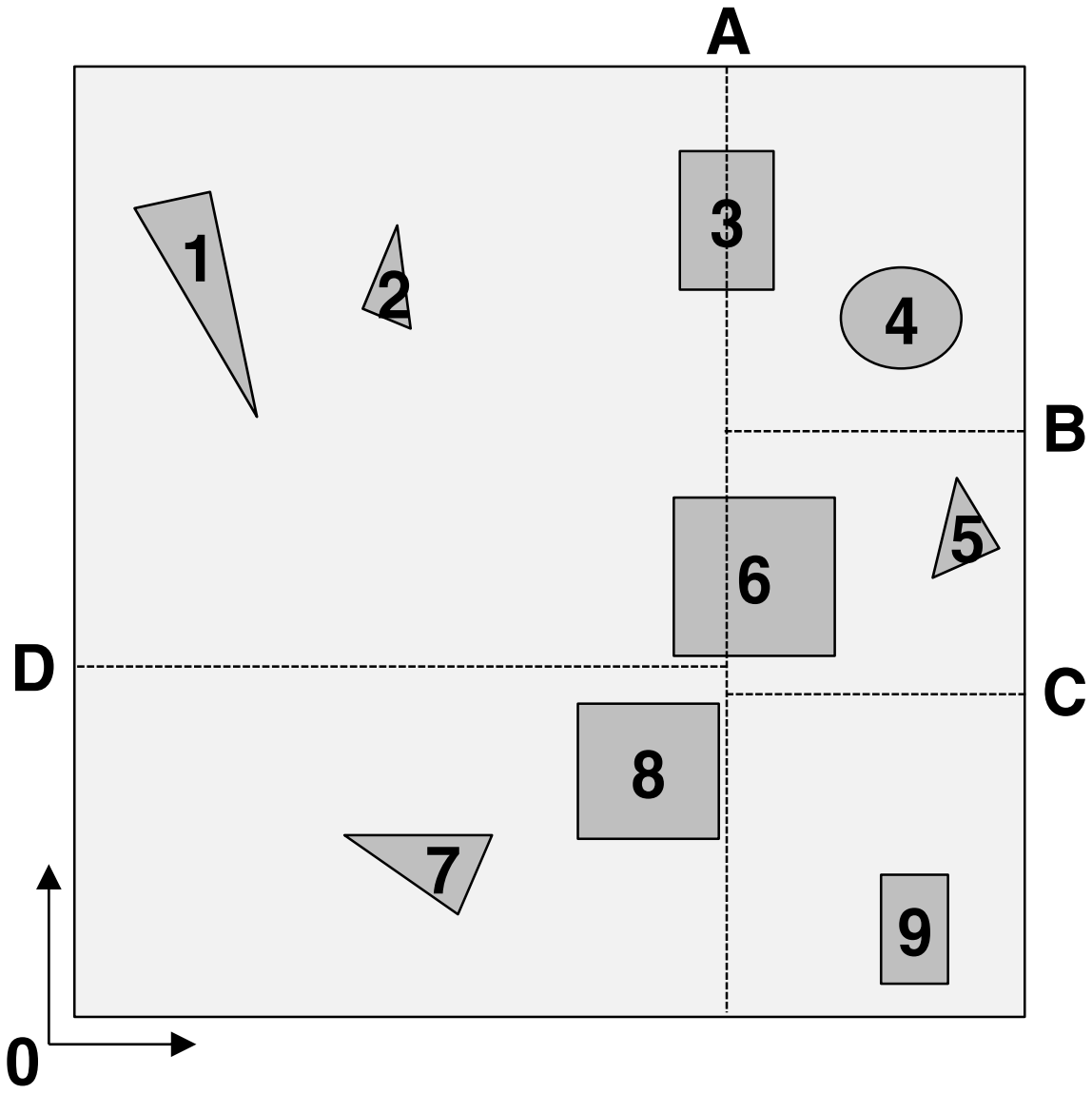


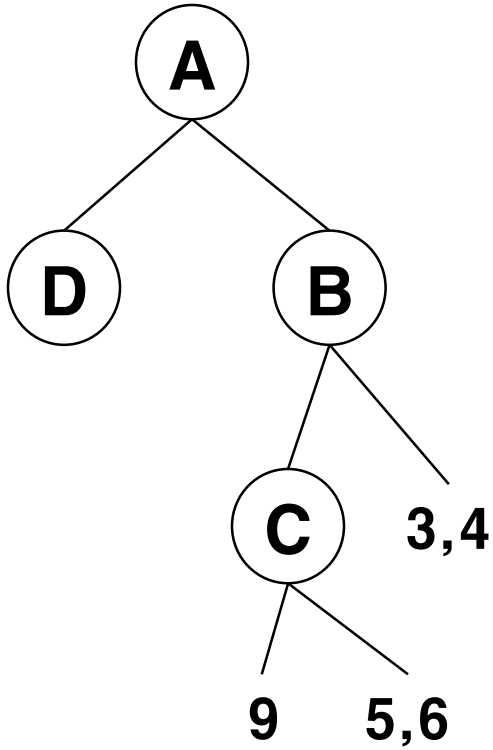
K-D Tree Construction



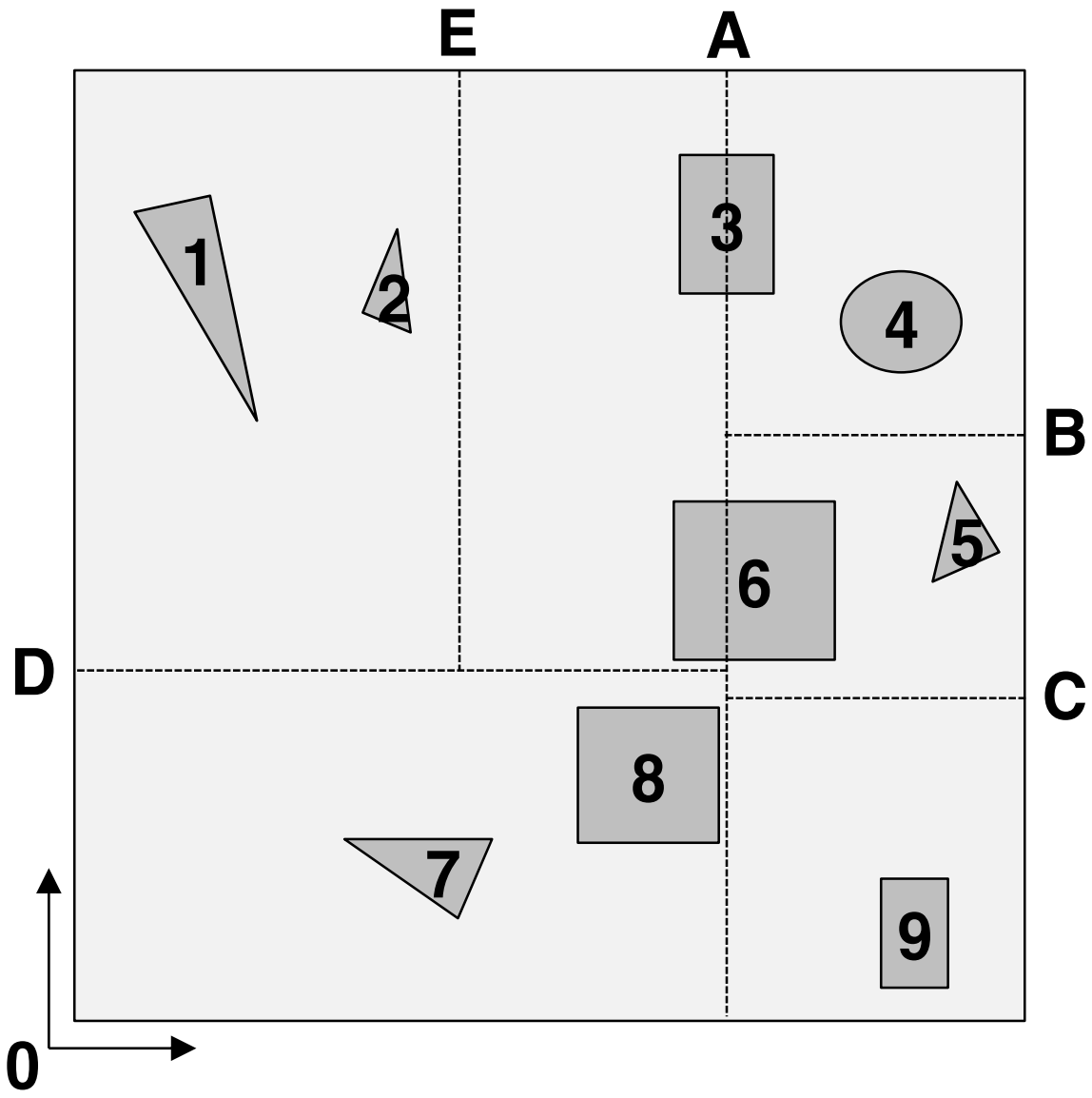


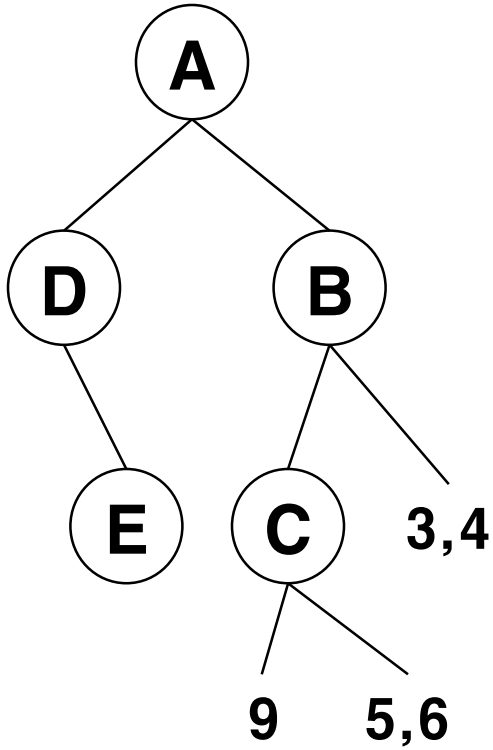
K-D Tree Construction



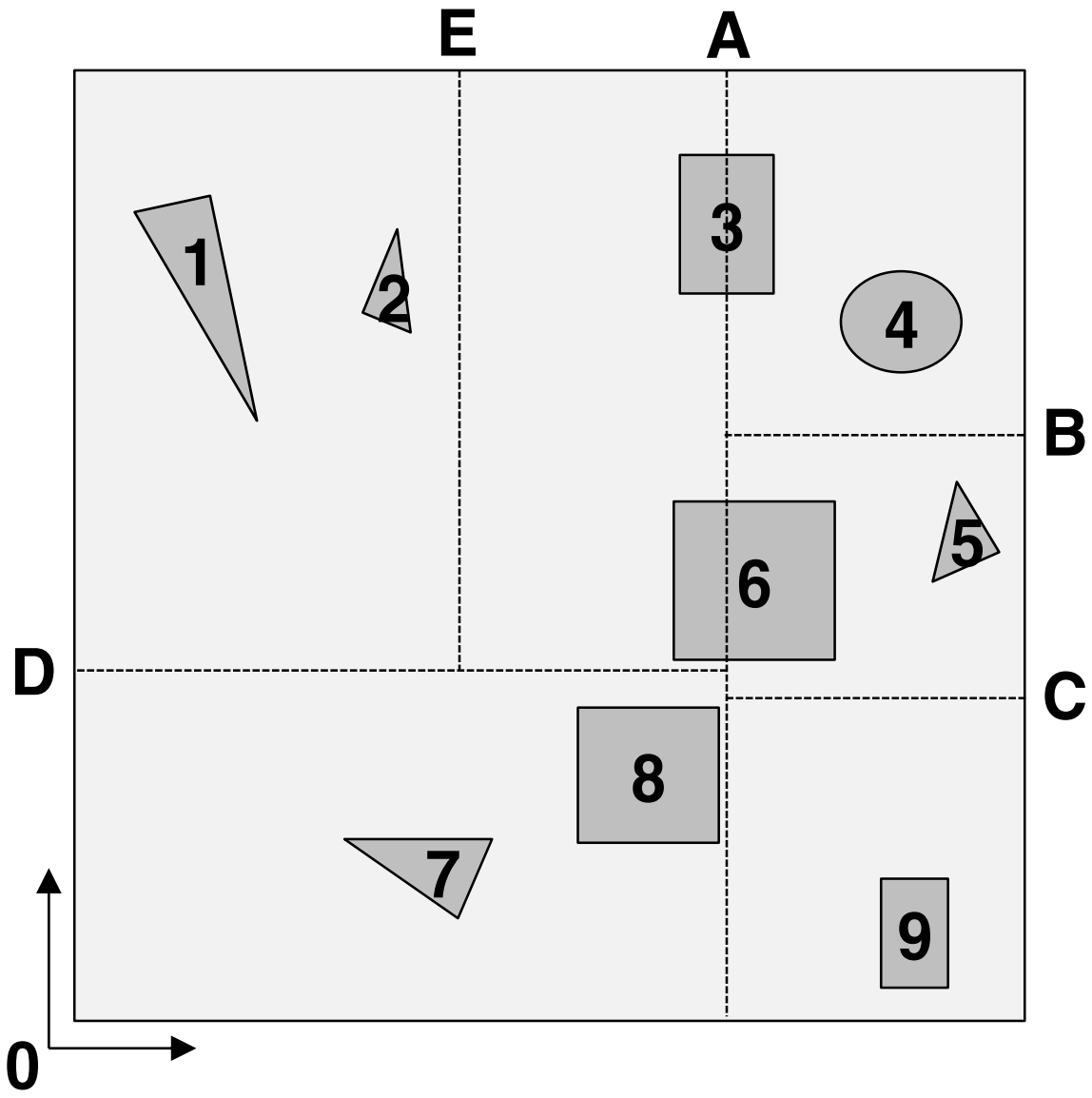


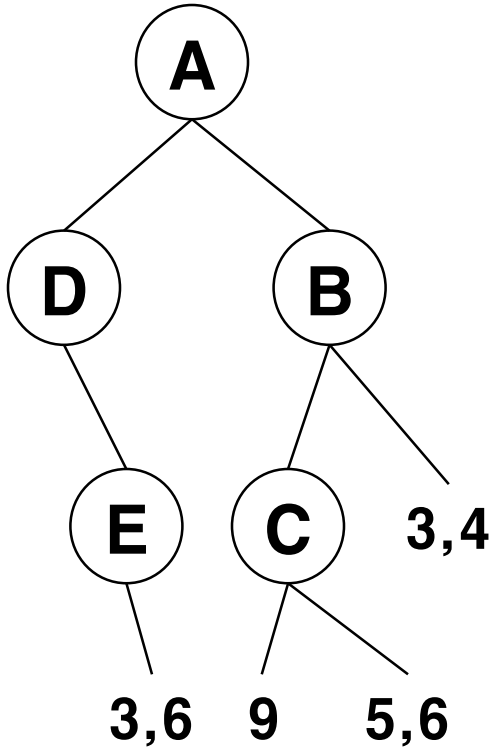
K-D Tree Construction



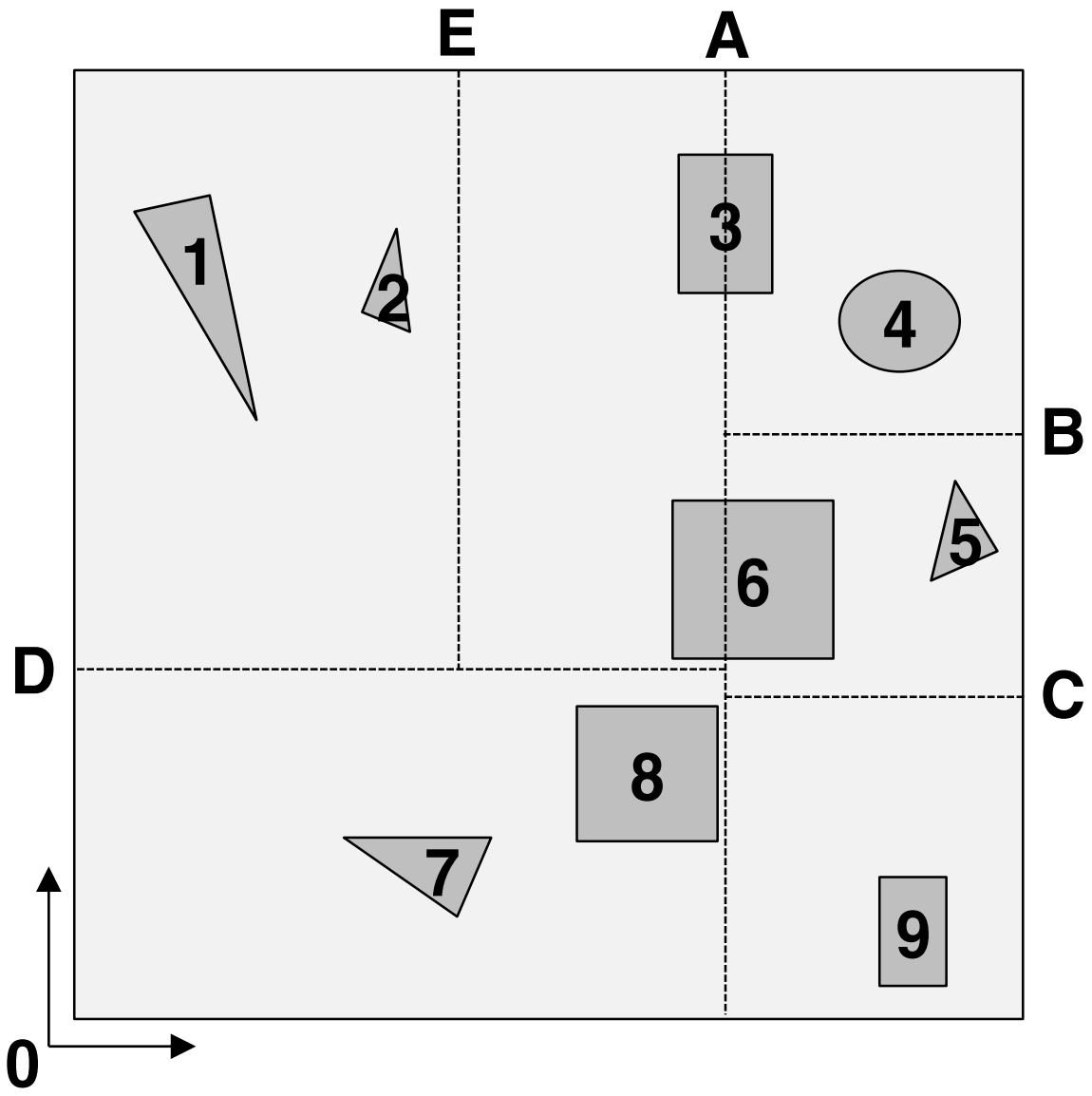


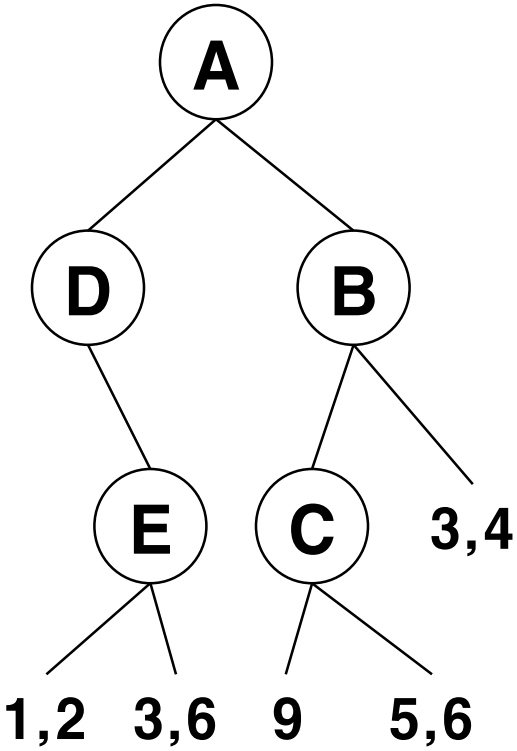
K-D Tree Construction



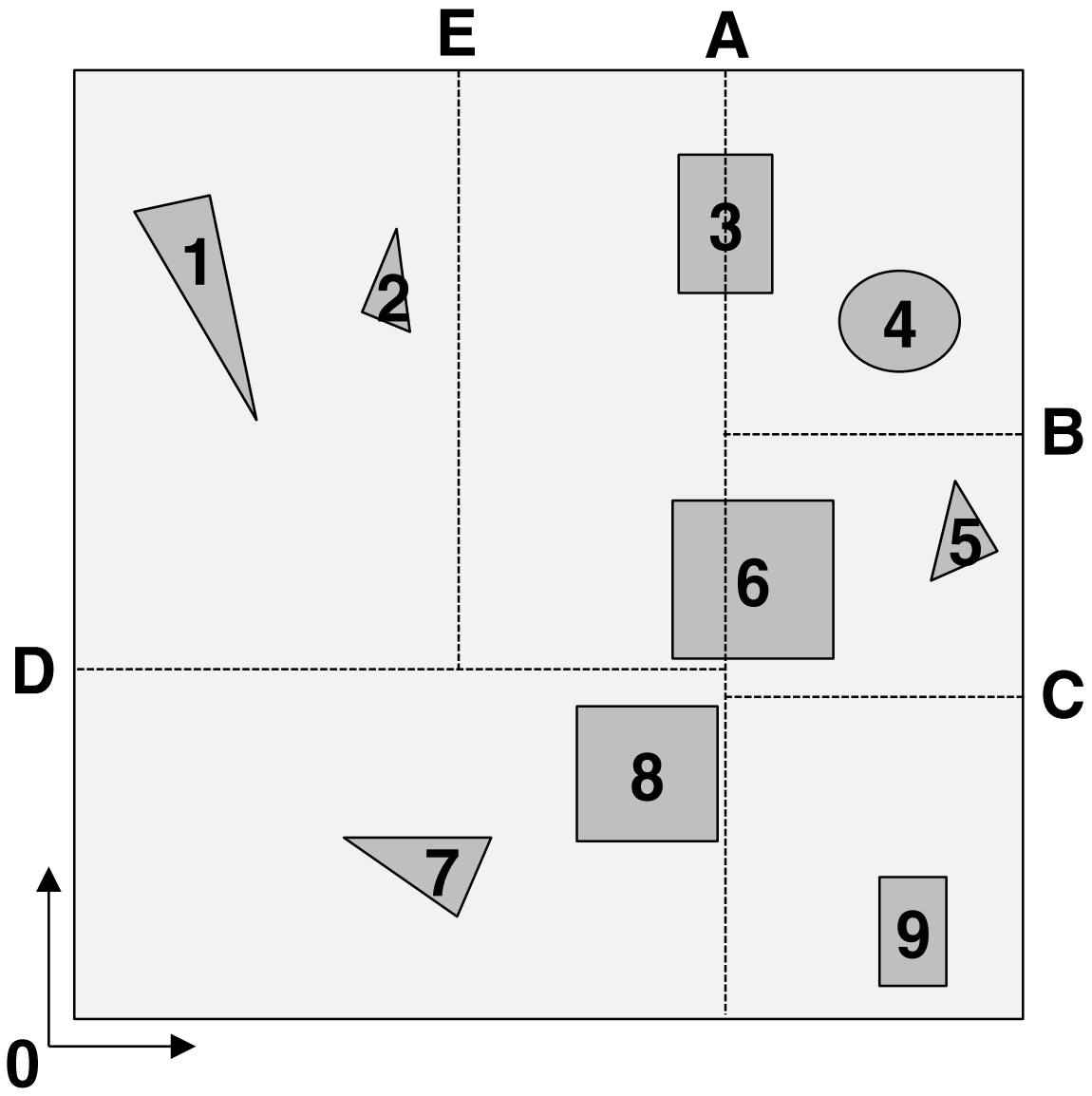


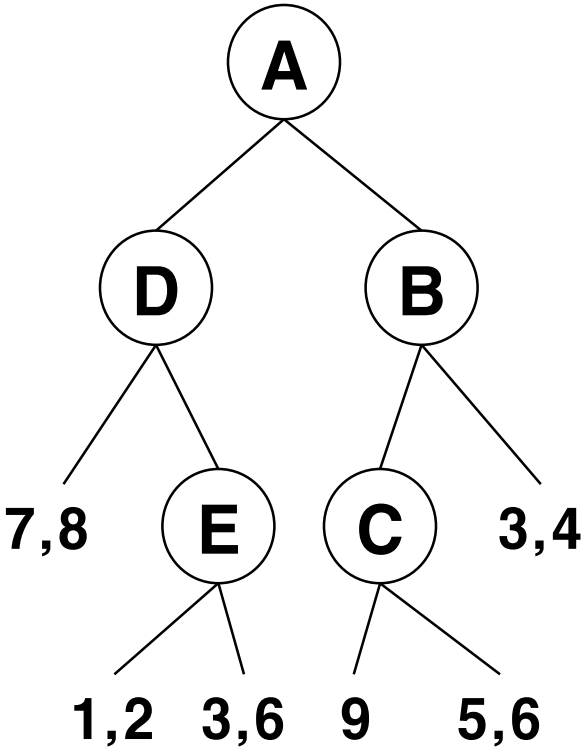
K-D Tree Construction





K-D Tree Construction

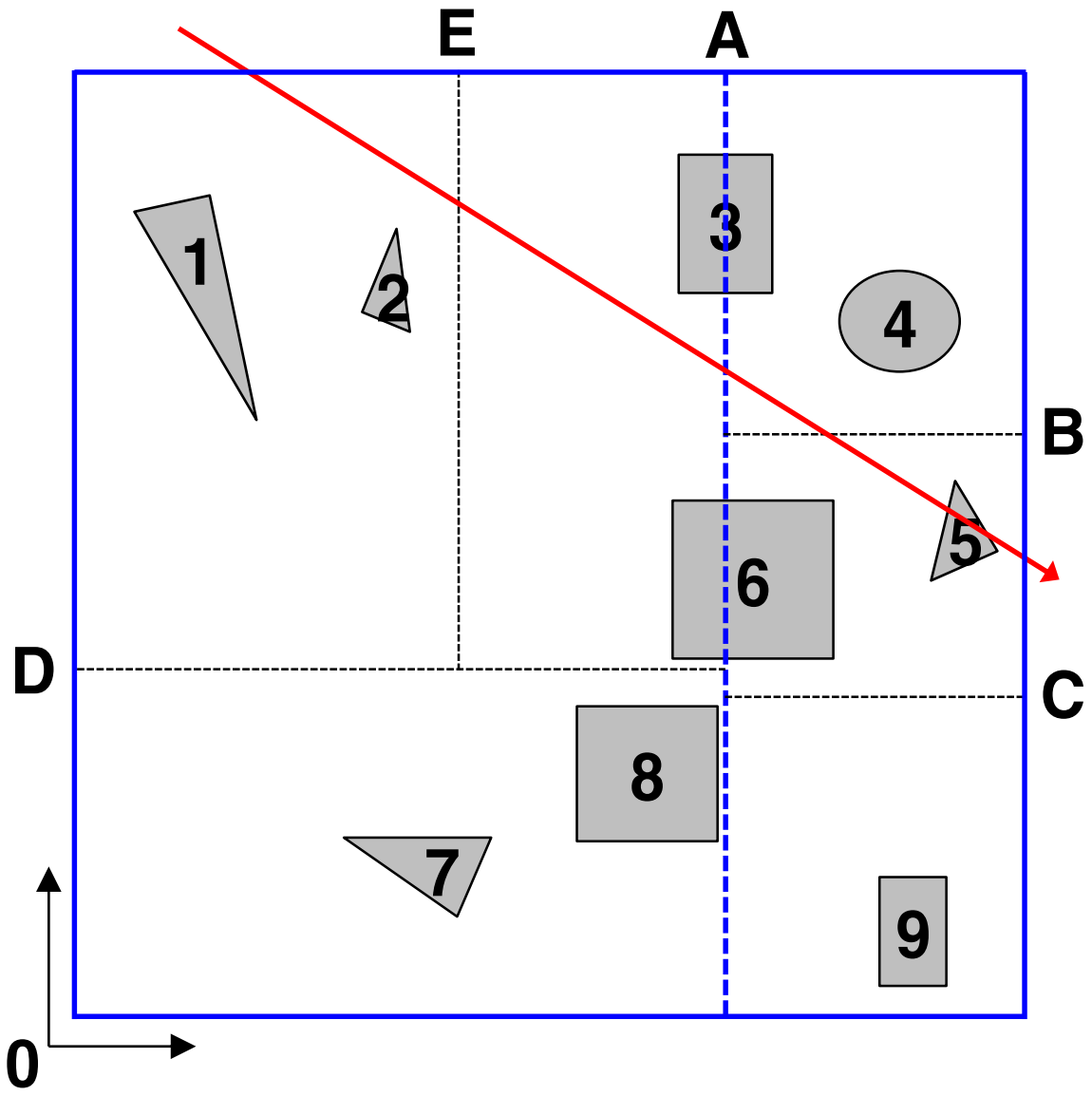


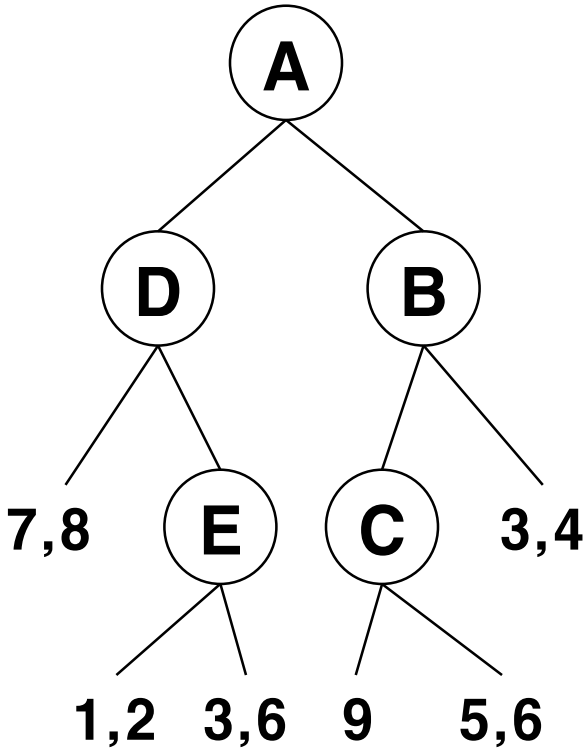


K-D Tree Traversal

- **Front-to-back traversal**
- **Traverse child nodes in order along rays**
- **Stop traversing as soon as surface intersection is found**
- **Maintain a stack of subtrees to traverse**
  - More efficient than recursive function calls

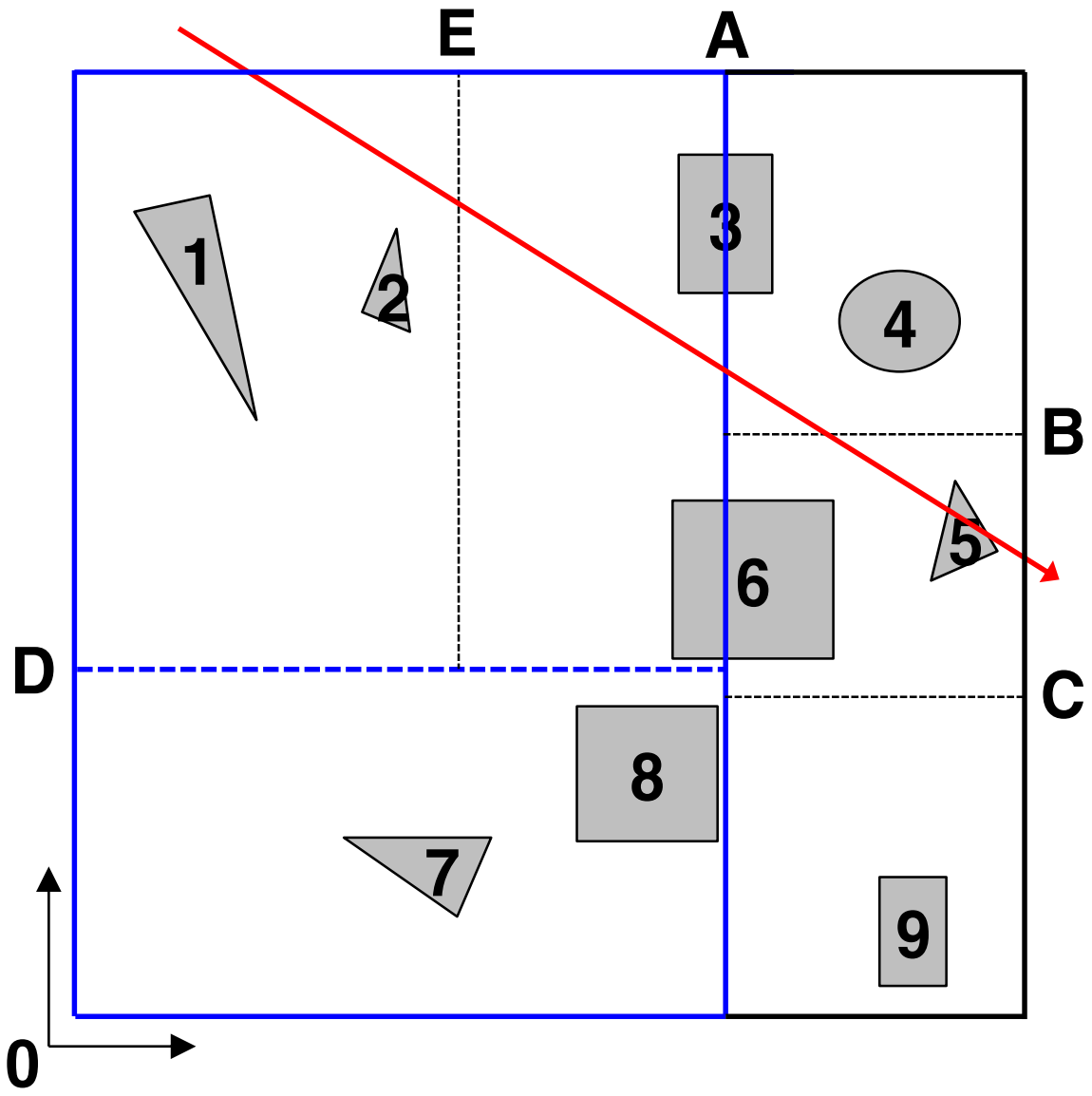
K-D Tree Traversal

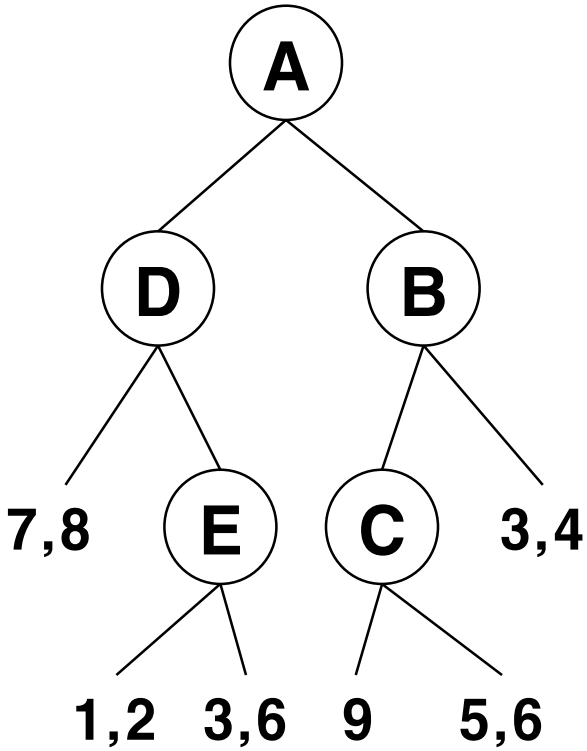




Process	Stack
A	

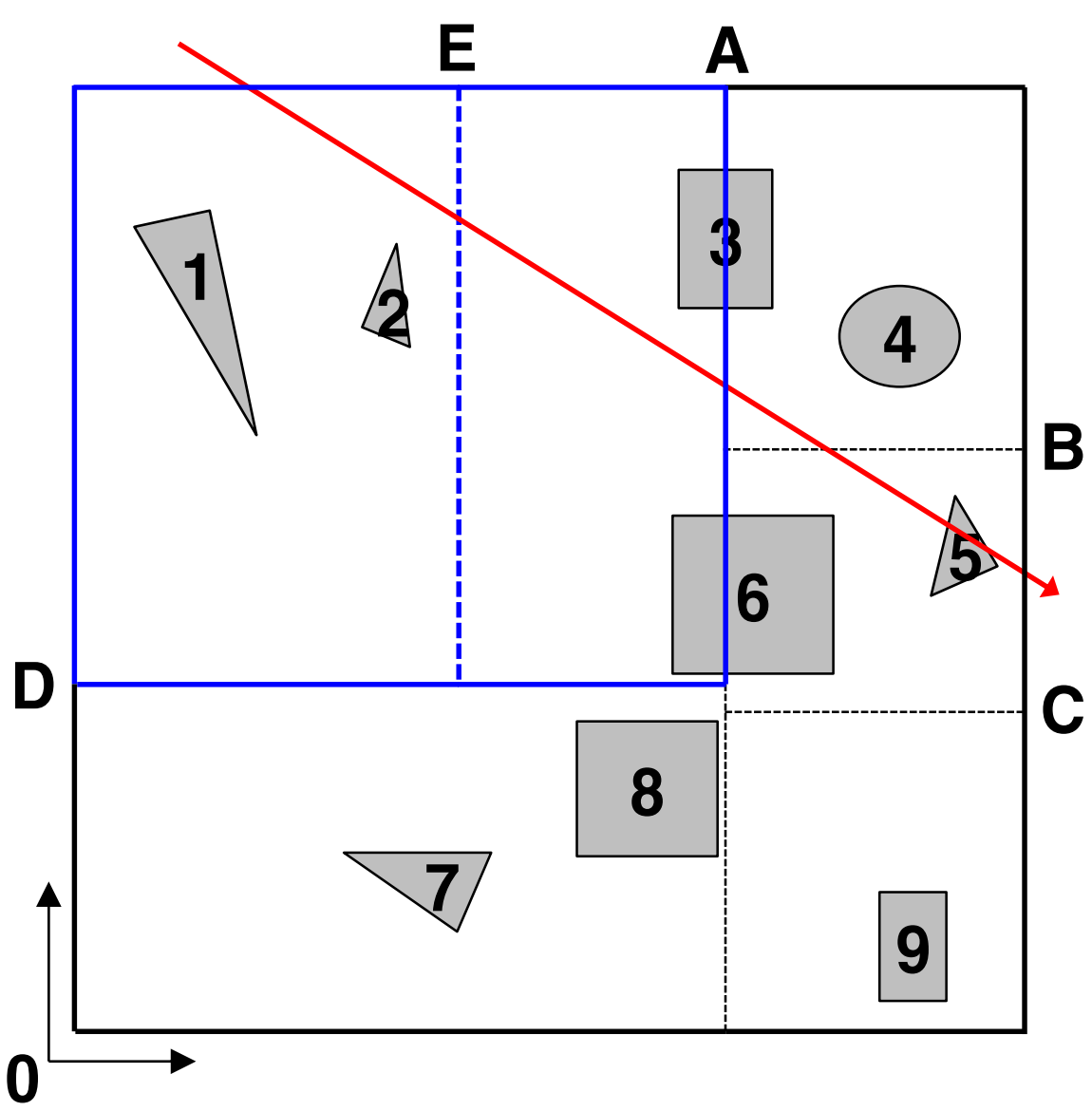
K-D Tree Traversal

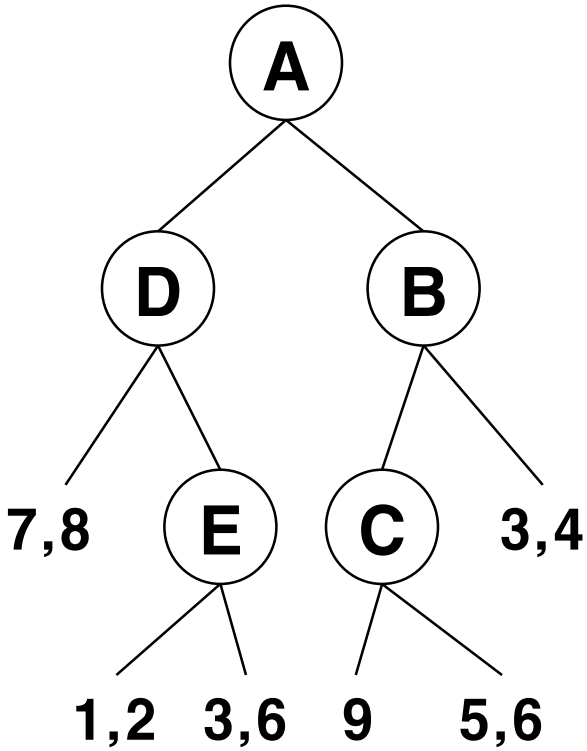




Process	Stack
D	B

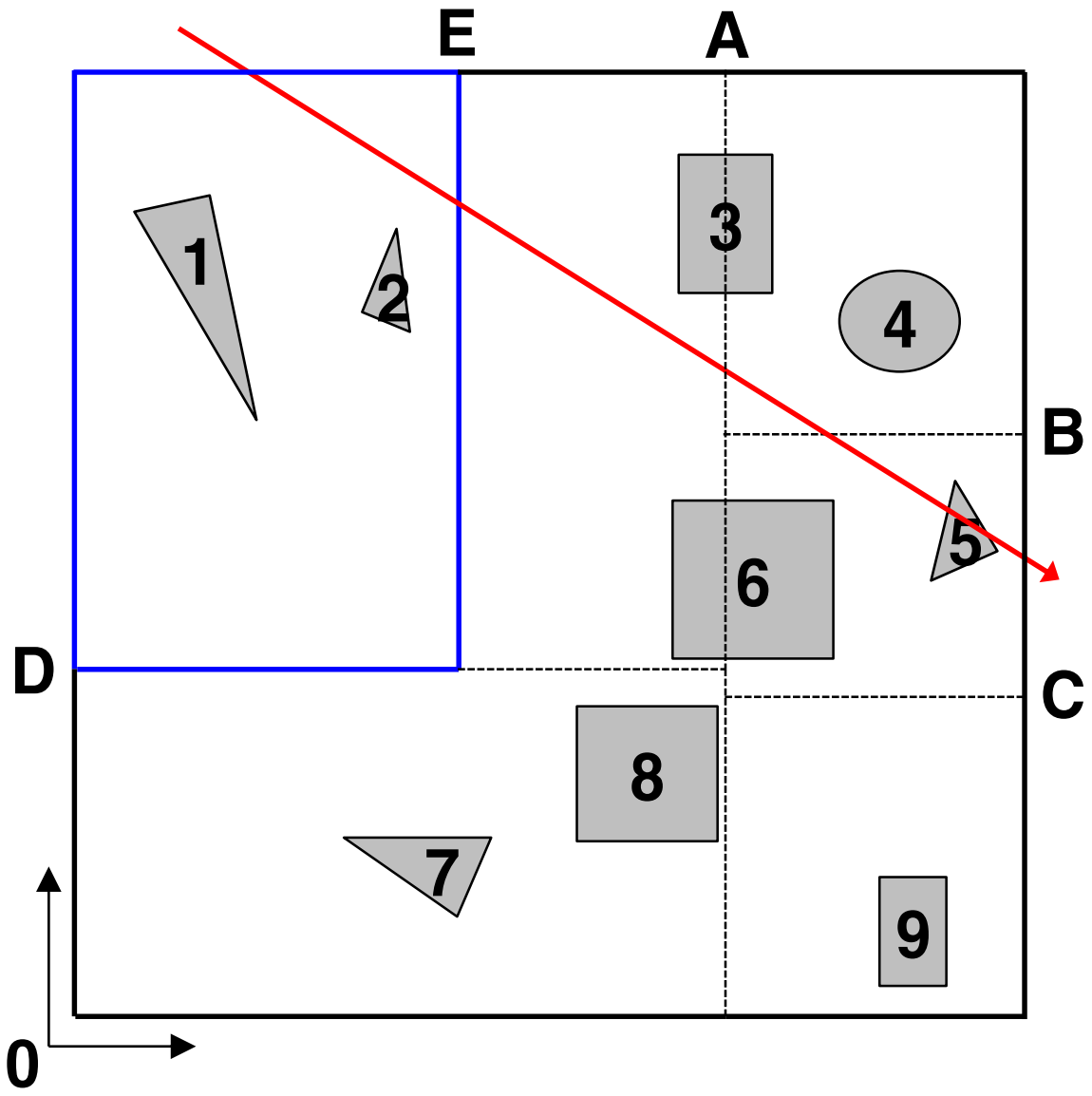
K-D Tree Traversal

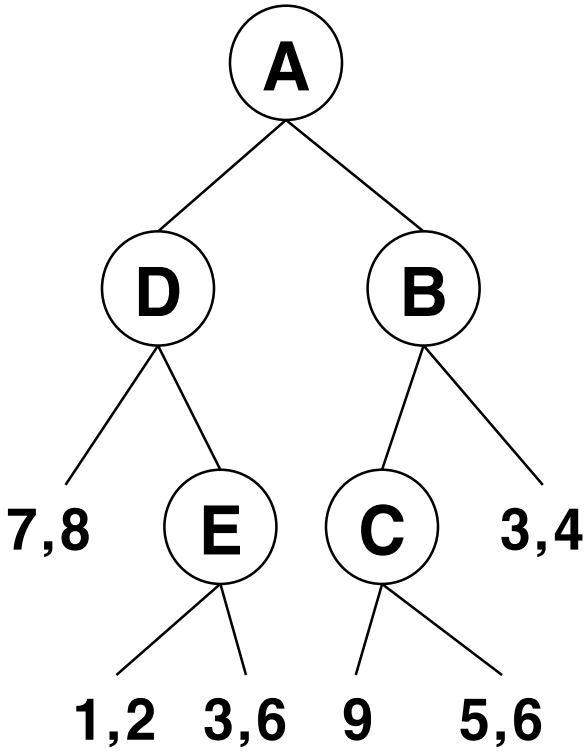




Process	Stack
E	B

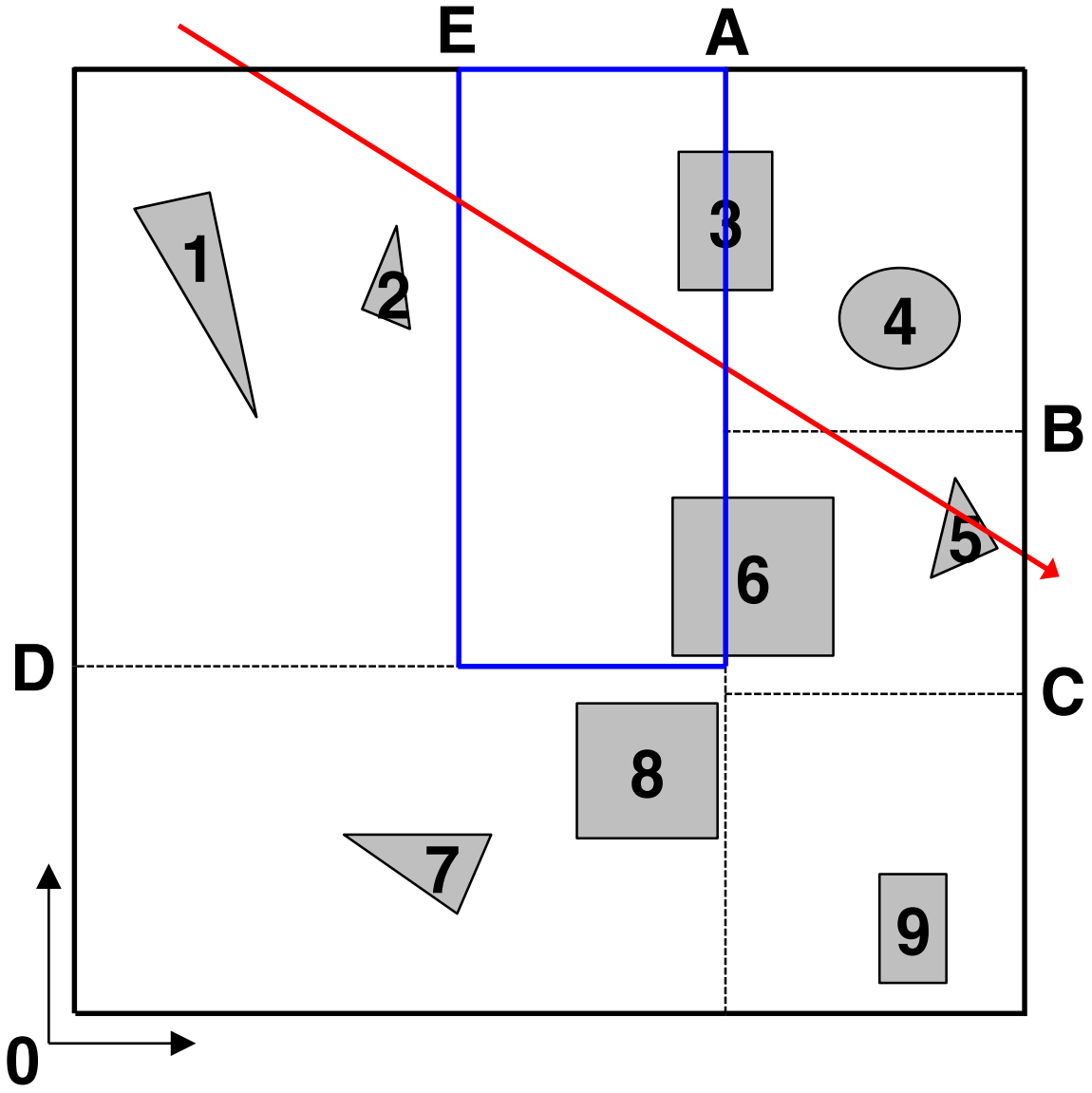
K-D Tree Traversal

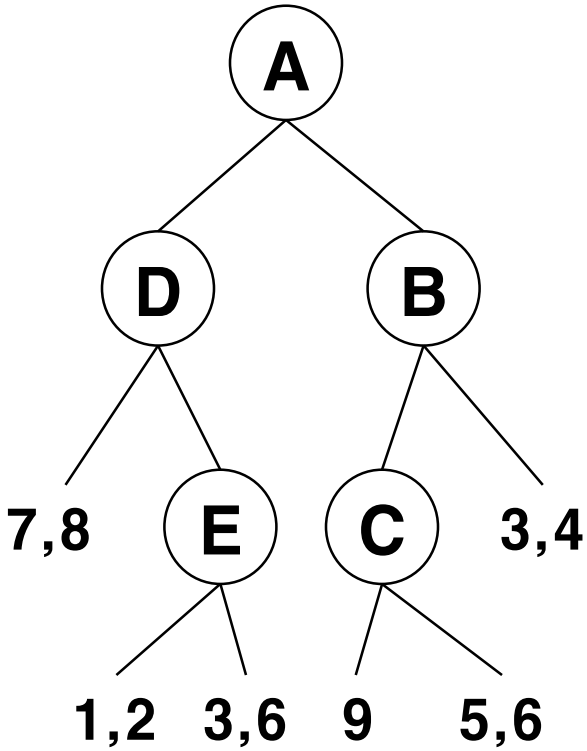




Process	Stack
1,2	B
	3,6

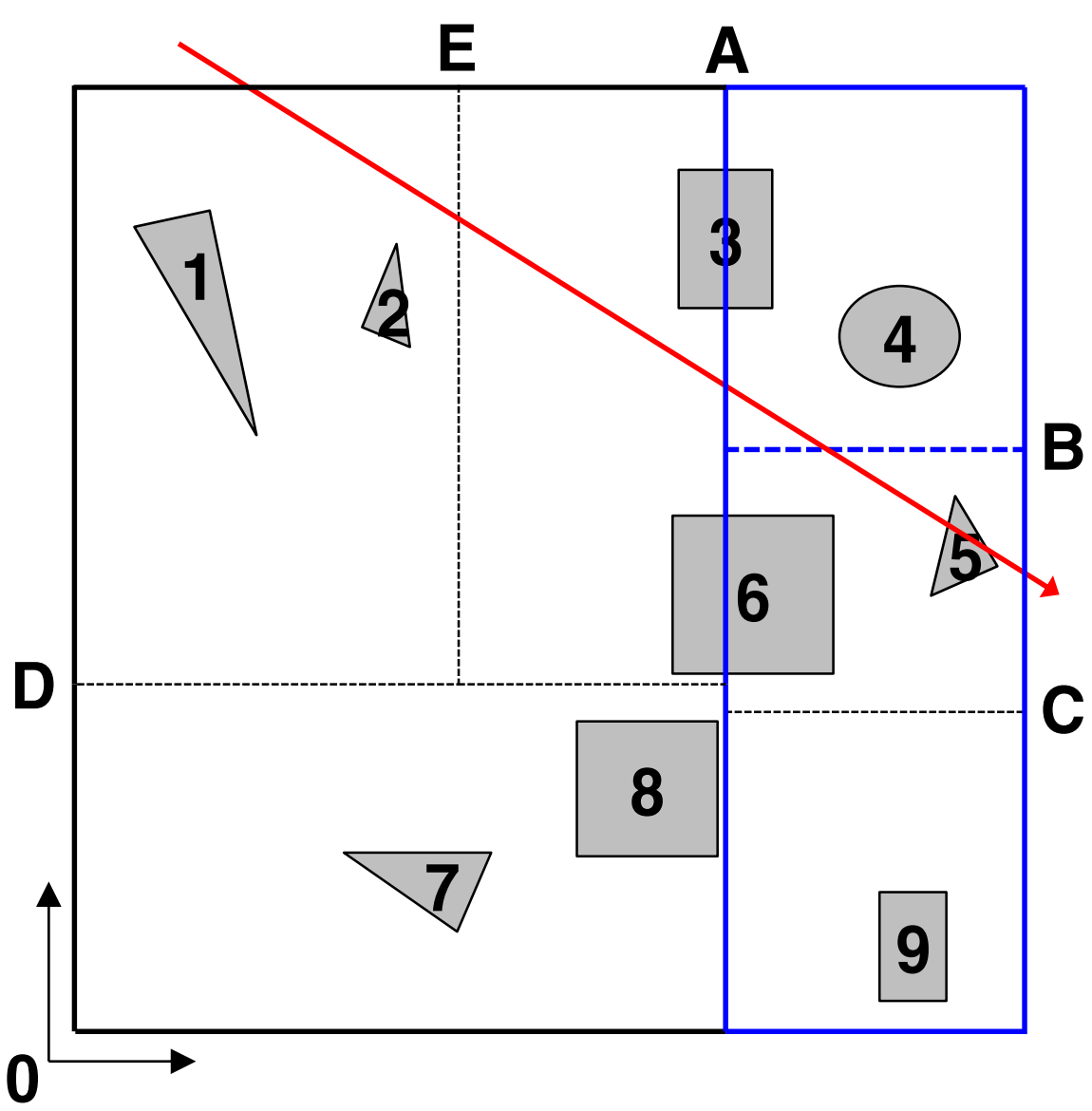
K-D Tree Traversal

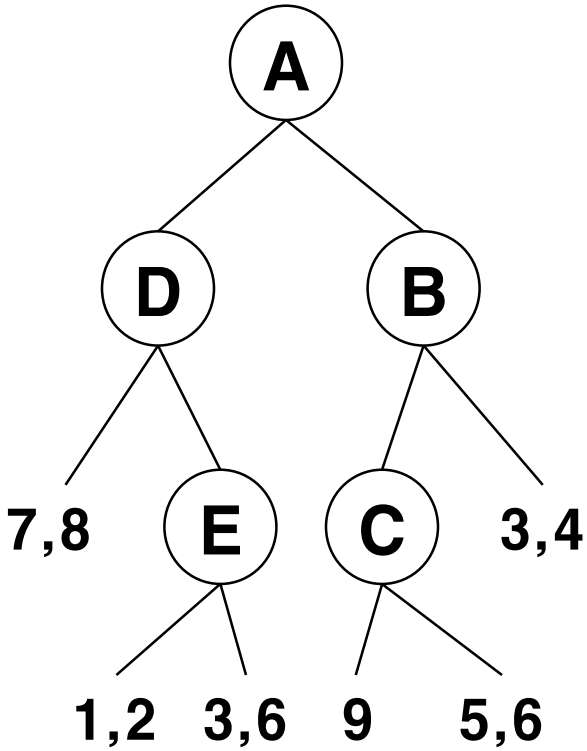




Process	Stack
3,6	B

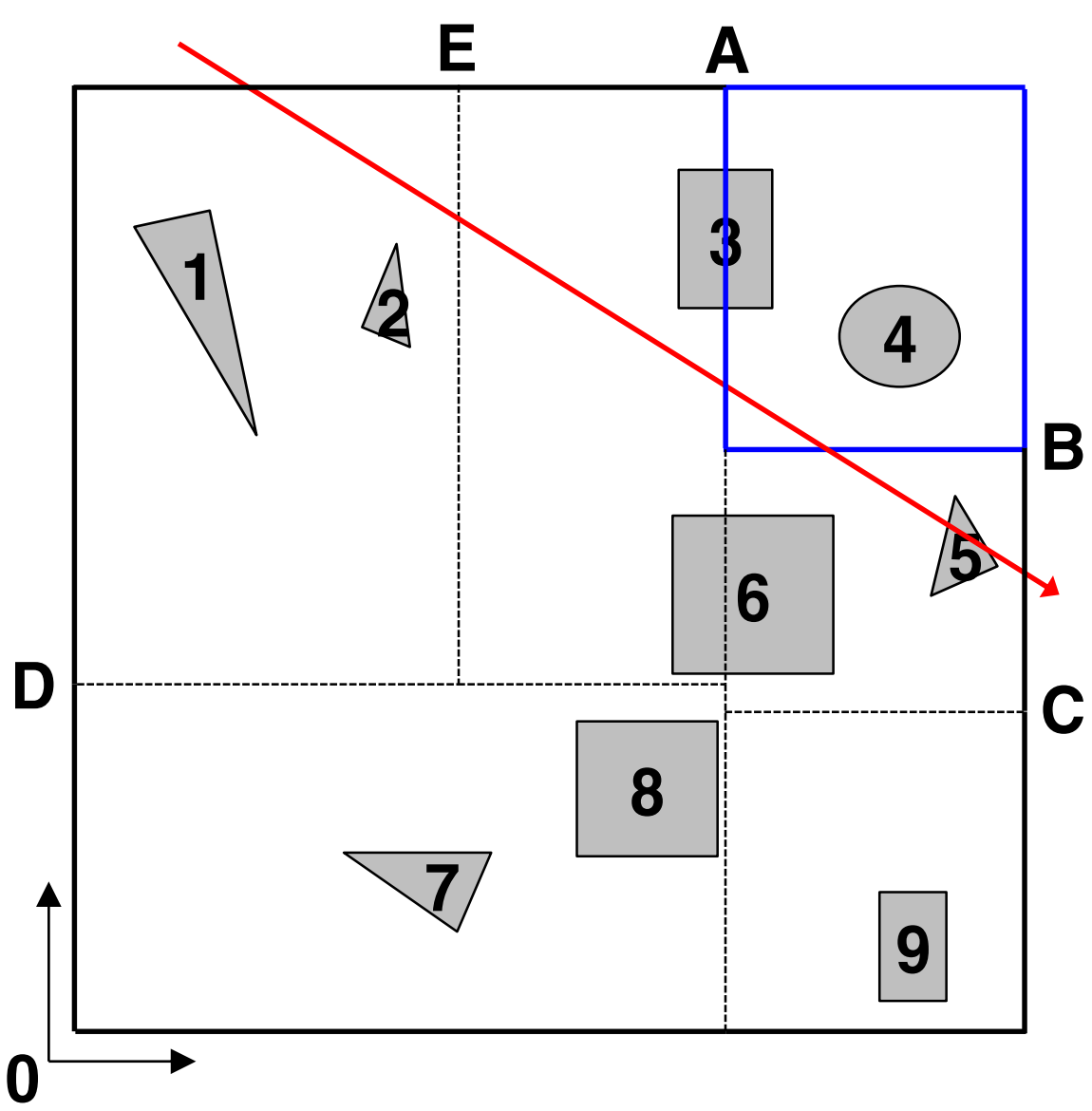
K-D Tree Traversal

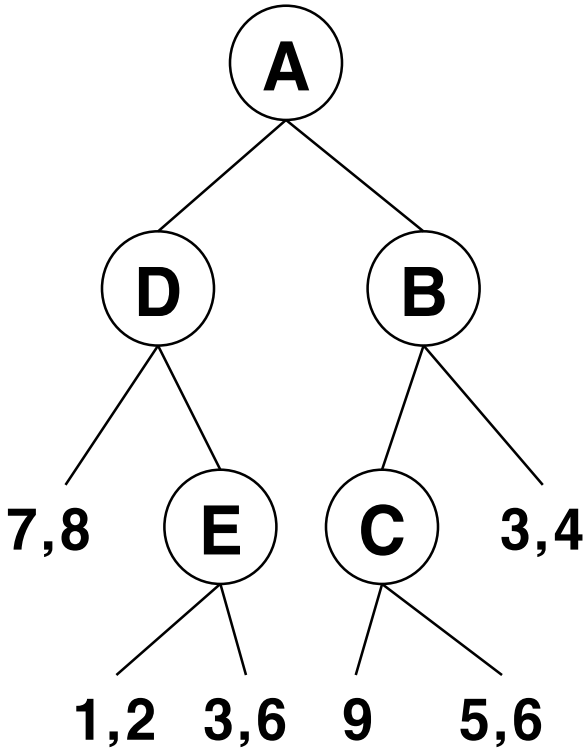




Process	Stack
B	

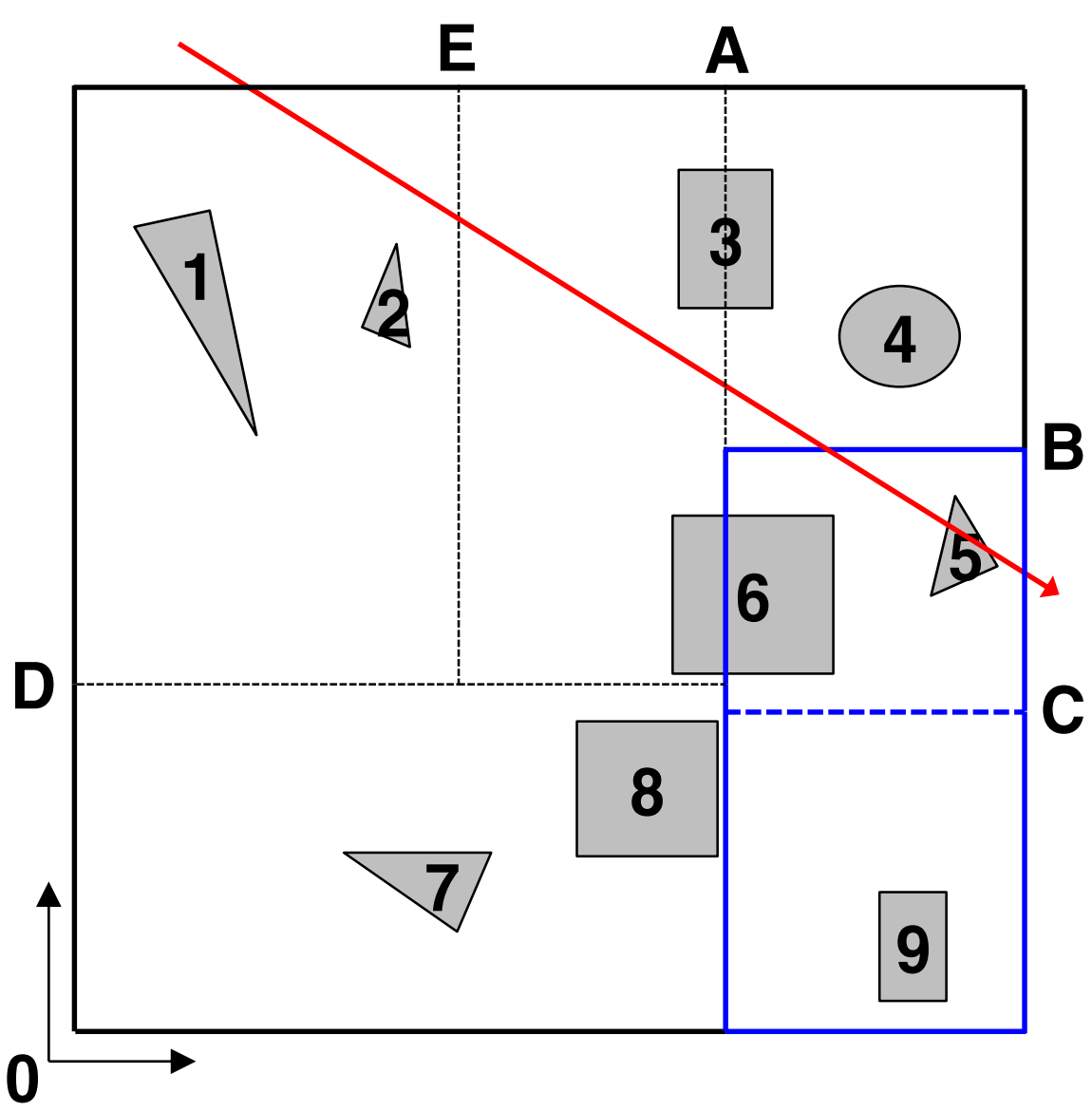
K-D Tree Traversal

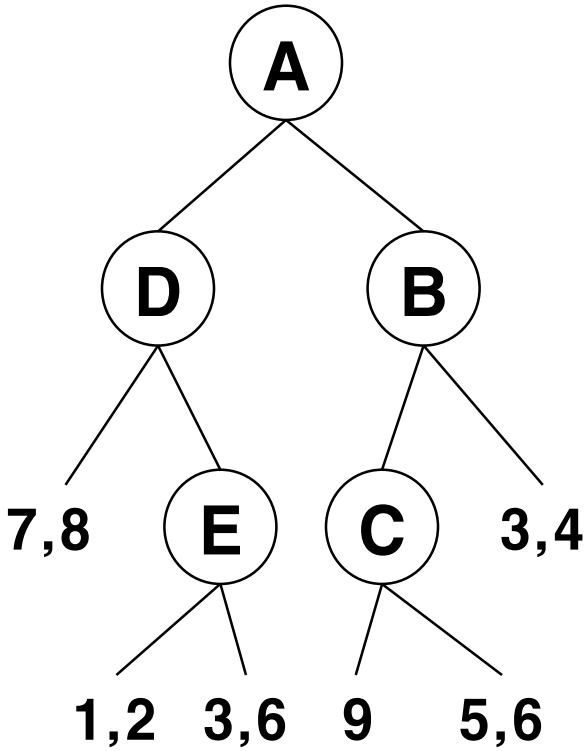




Process	Stack
3,4	C

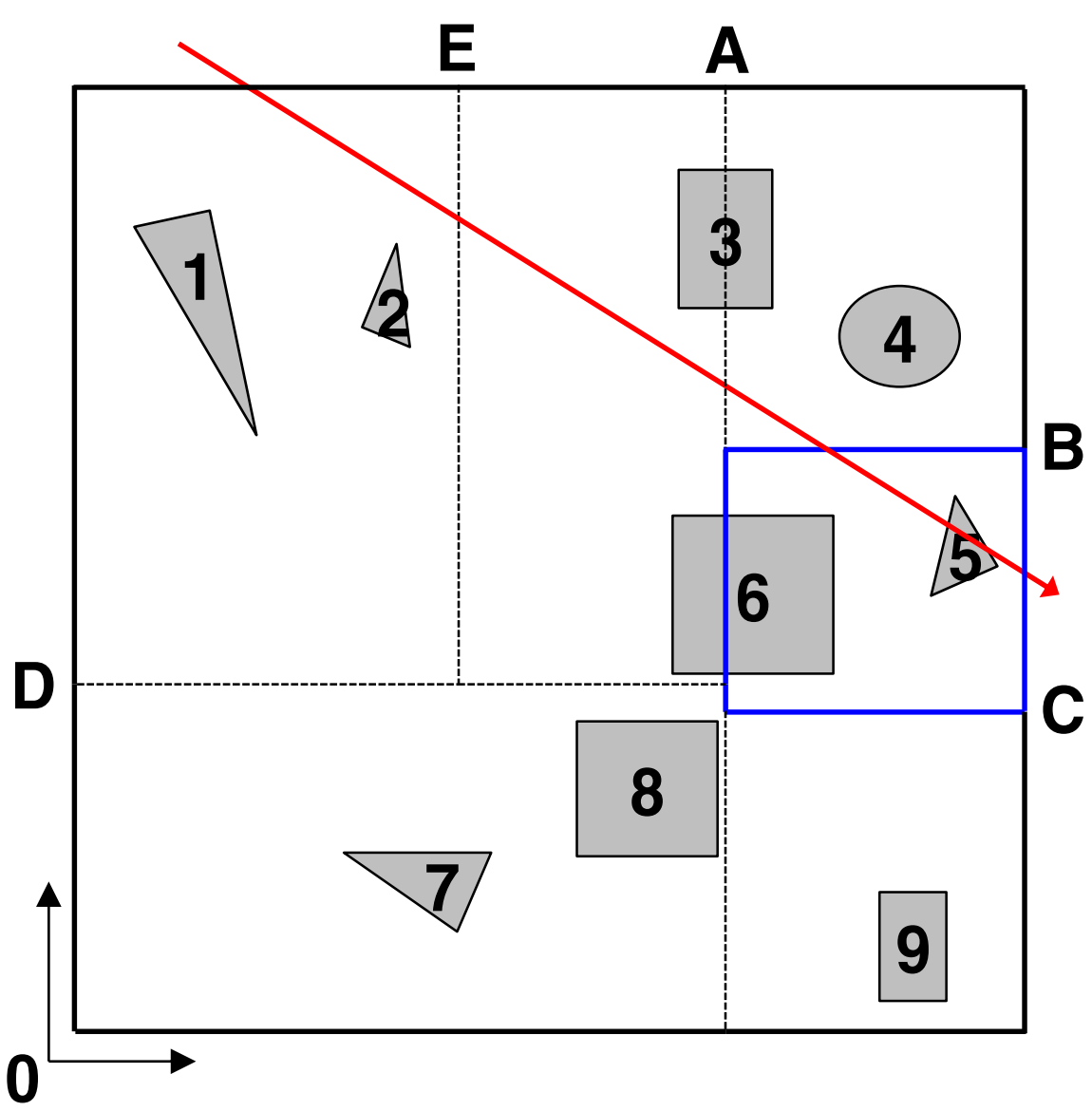
K-D Tree Traversal

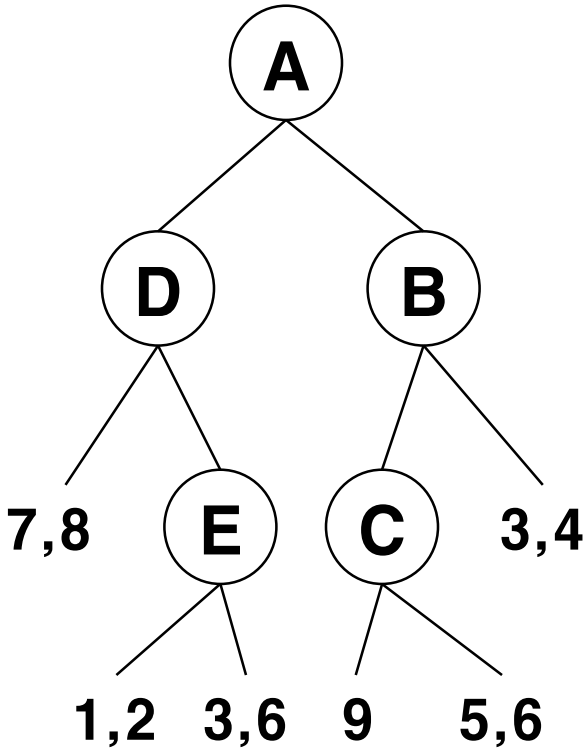




Process	Stack
C	

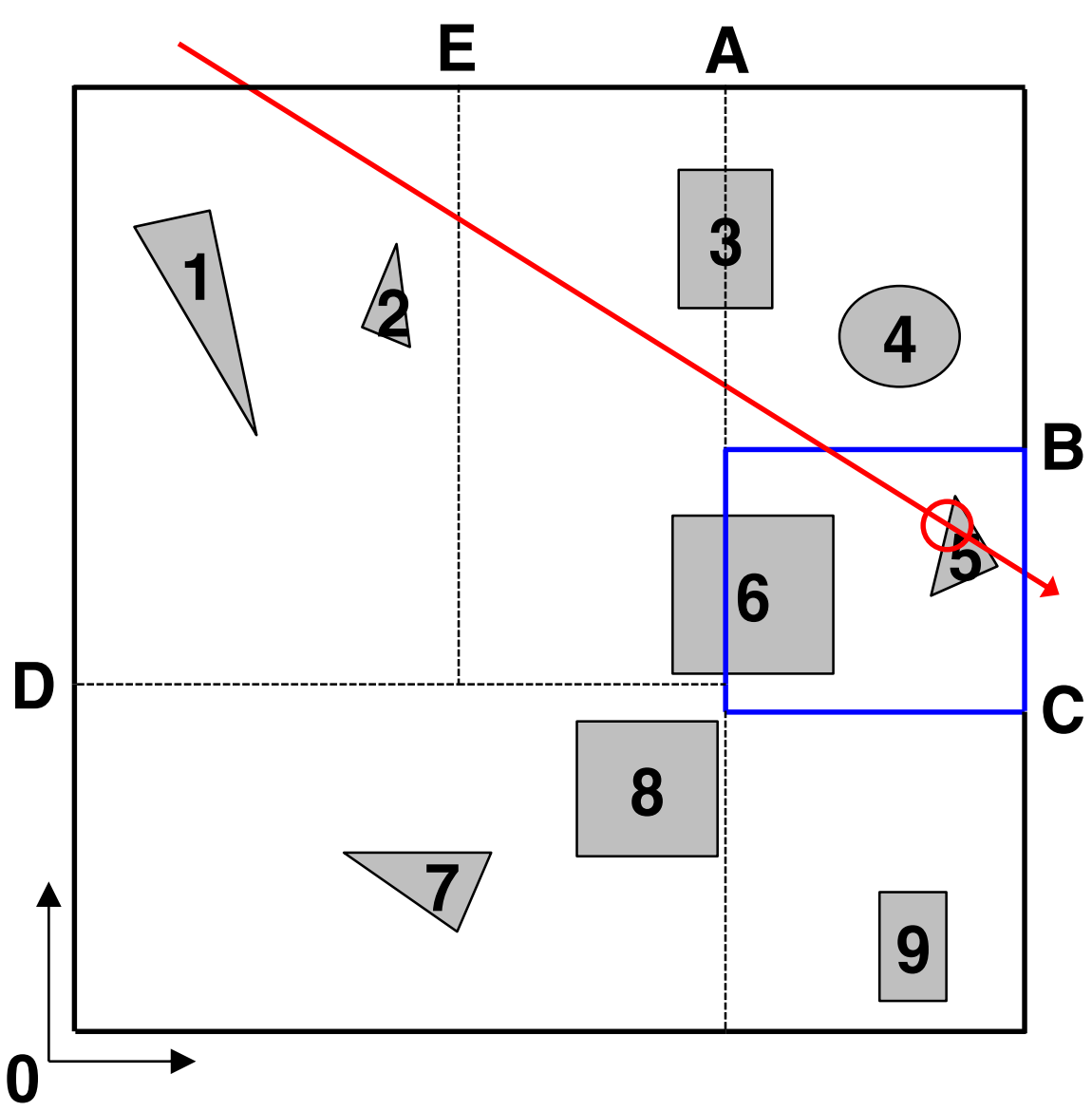
K-D Tree Traversal

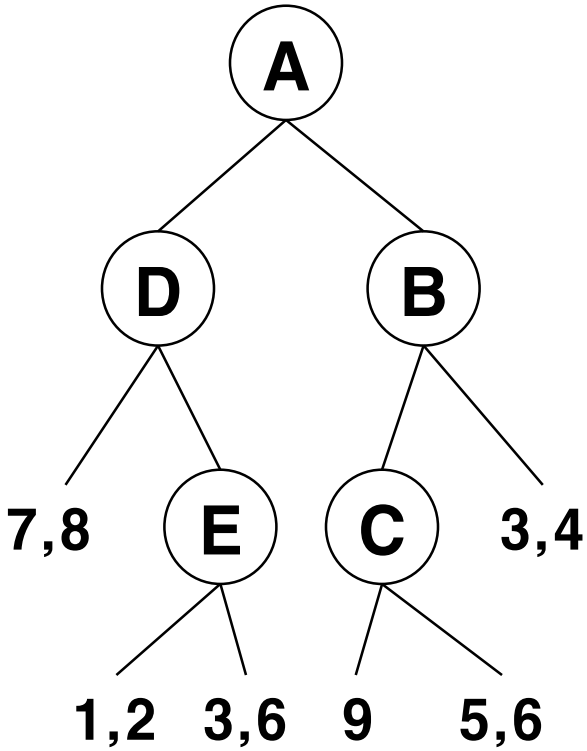




Process	Stack
5,6	9

K-D Tree Traversal





Process	Stack
5,6	9