Lecture 10:

Accelerating Ray-Scene Intersection

Computer Graphics and Imaging

UC Berkeley CS184/284A

Ray Tracing – Performance Challenges

San Miguel Scene, 10.7M triangles

Ray Tracing – Performance Challenges

Simple ray-scene intersection

- Exhaustively test ray-intersection with every object

Problem:

- Exhaustive algorithm = #pixels $\times$ #objects
- Very slow!

Ray Tracing – Acceleration

- Brute-force algorithm = #pixels × #objects
- Acceleration structures ≈ #pixels × log (#objects)

Bounding Volumes

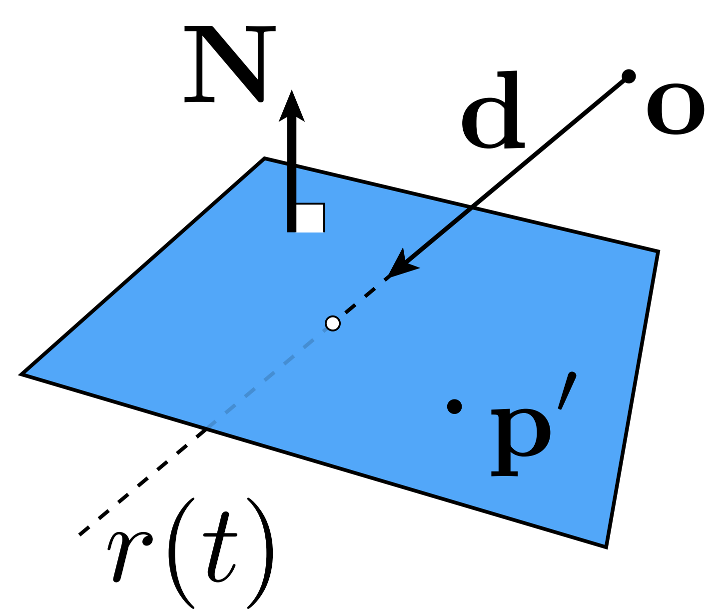
Bounding Volumes

Quick way to avoid intersections: bound complex object with a simple volume

- Object is fully contained in the volume
- If it doesn't hit the volume, it doesn't hit the object
- So test bvol first, then test object if it hits

Optimize Ray-Plane Intersection For Axis-Aligned Planes?

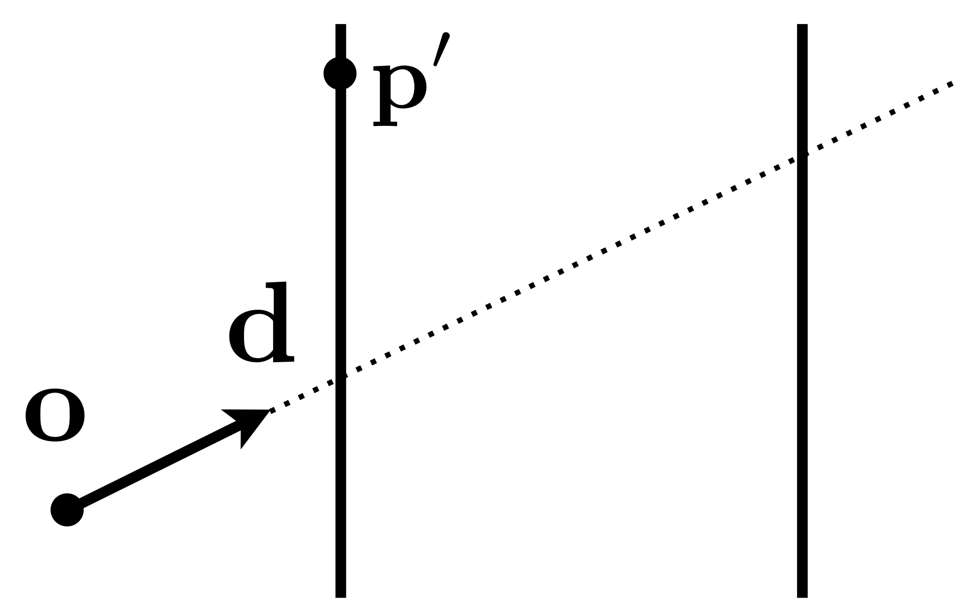
General



$$t = \frac{(\mathbf{p}' - \mathbf{o}) \cdot \mathbf{N}}{\mathbf{d} \cdot \mathbf{N}}$$

3 subtractions, 6 multiplies, 1 division

Perpendicular to x-axis



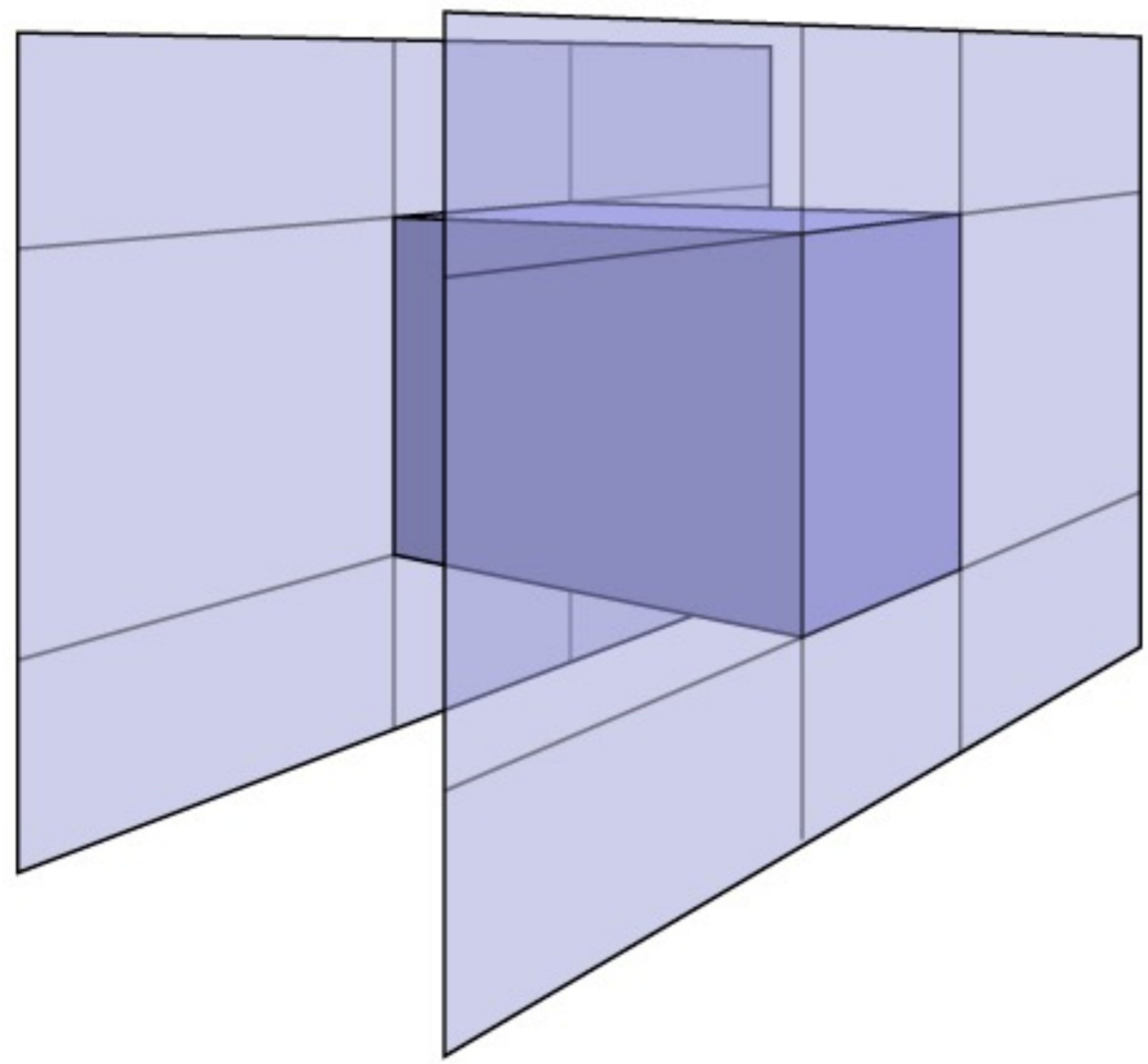
$$t = \frac{\mathbf{p}'_x - \mathbf{o}_x}{\mathbf{d}_x}$$

1 subtraction, 1 division

Ray-Intersection With Box

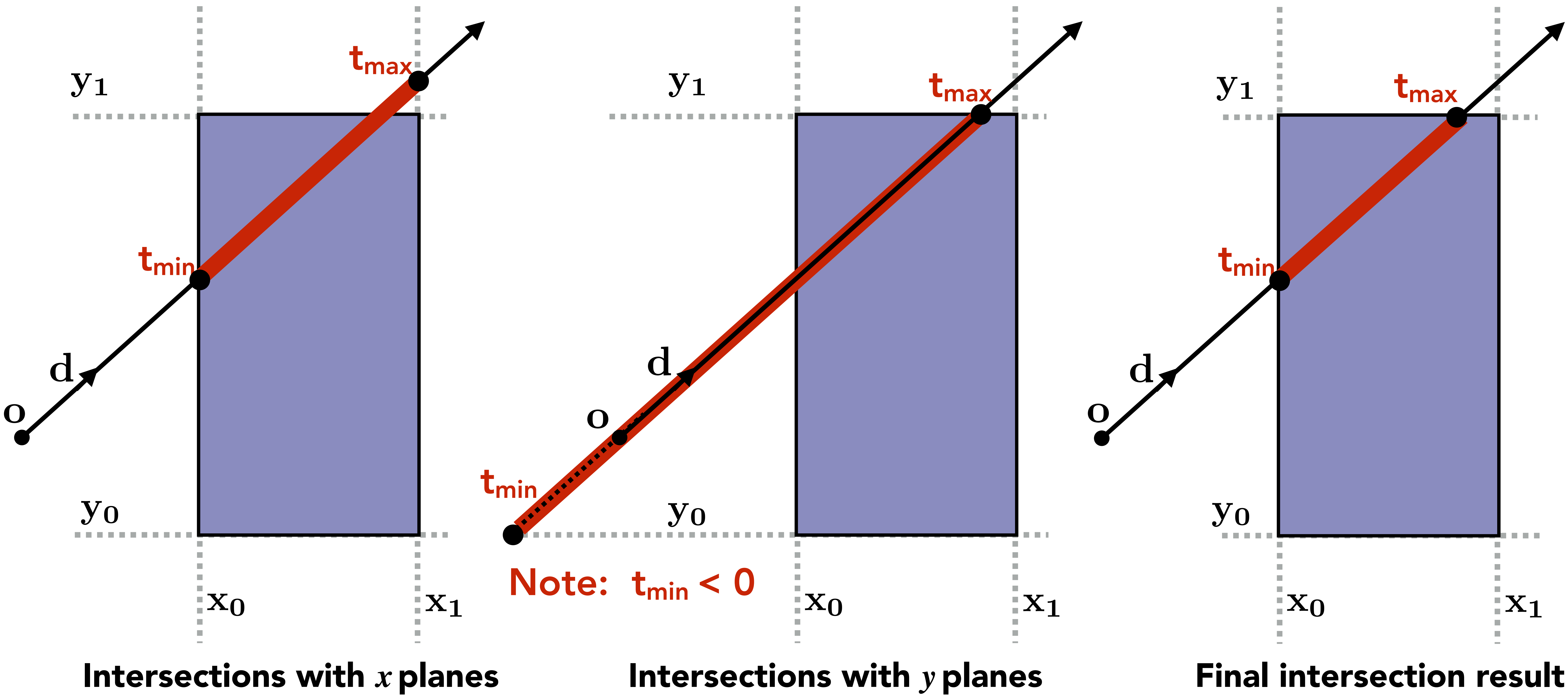
Could intersect with 6 faces individually

Better way: box is the intersection of 3 slabs



Ray Intersection with Axis-Aligned Box

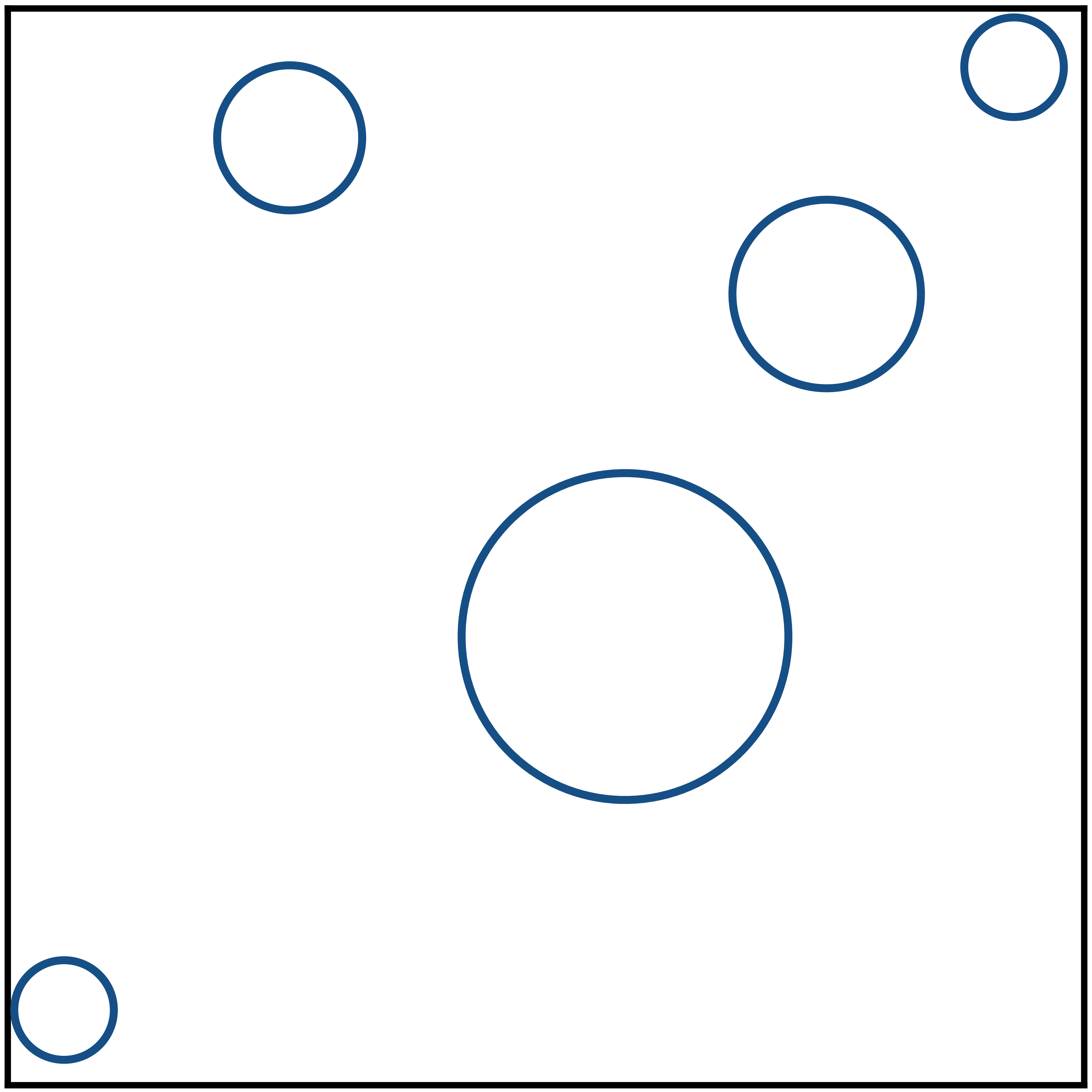
2D example; 3D is the same! Compute intersections with slabs and take intersection of t_{min}/t_{max} intervals



How do we know when the ray misses the box?

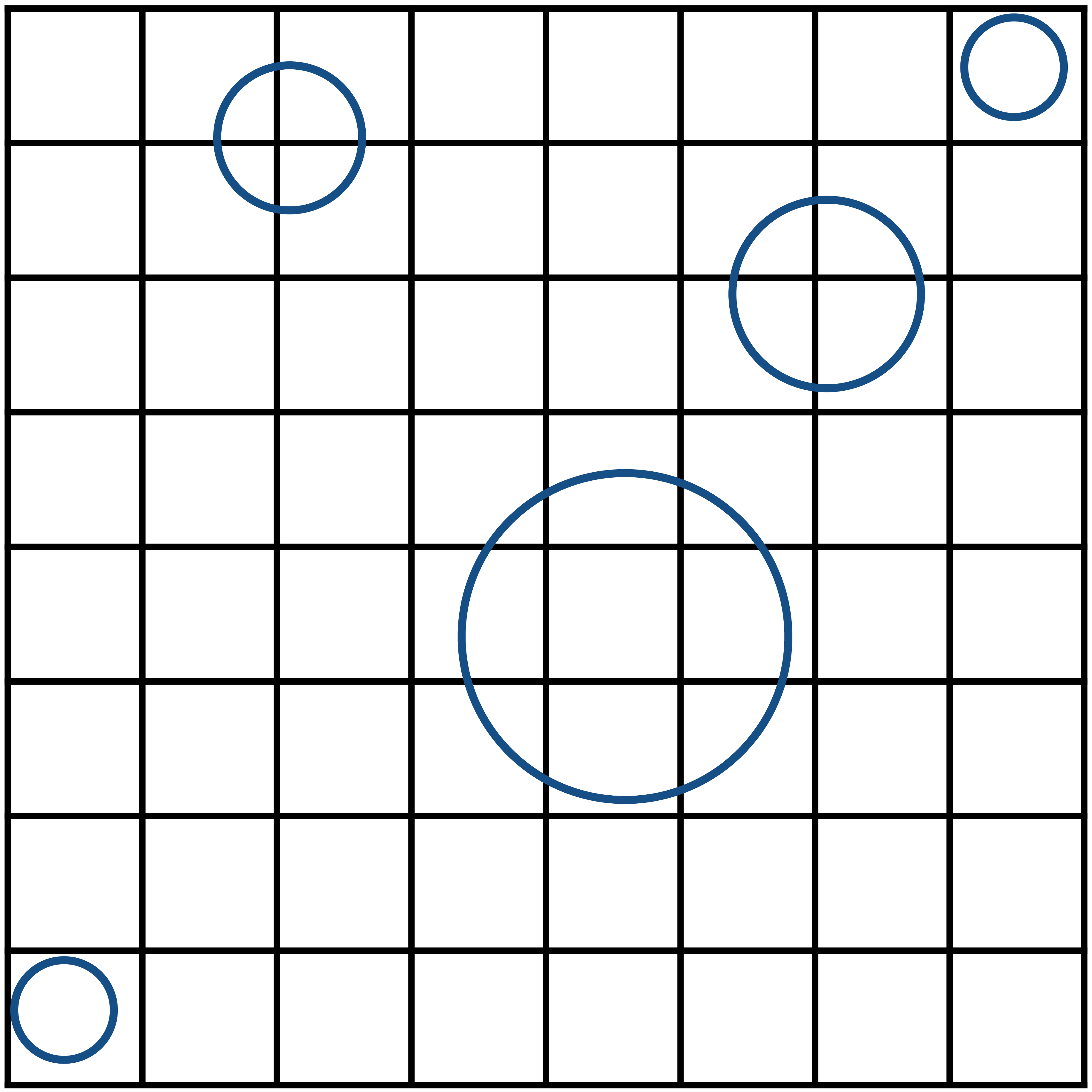
Uniform Spatial Partitions (Grids)

Preprocess – Build Acceleration Grid



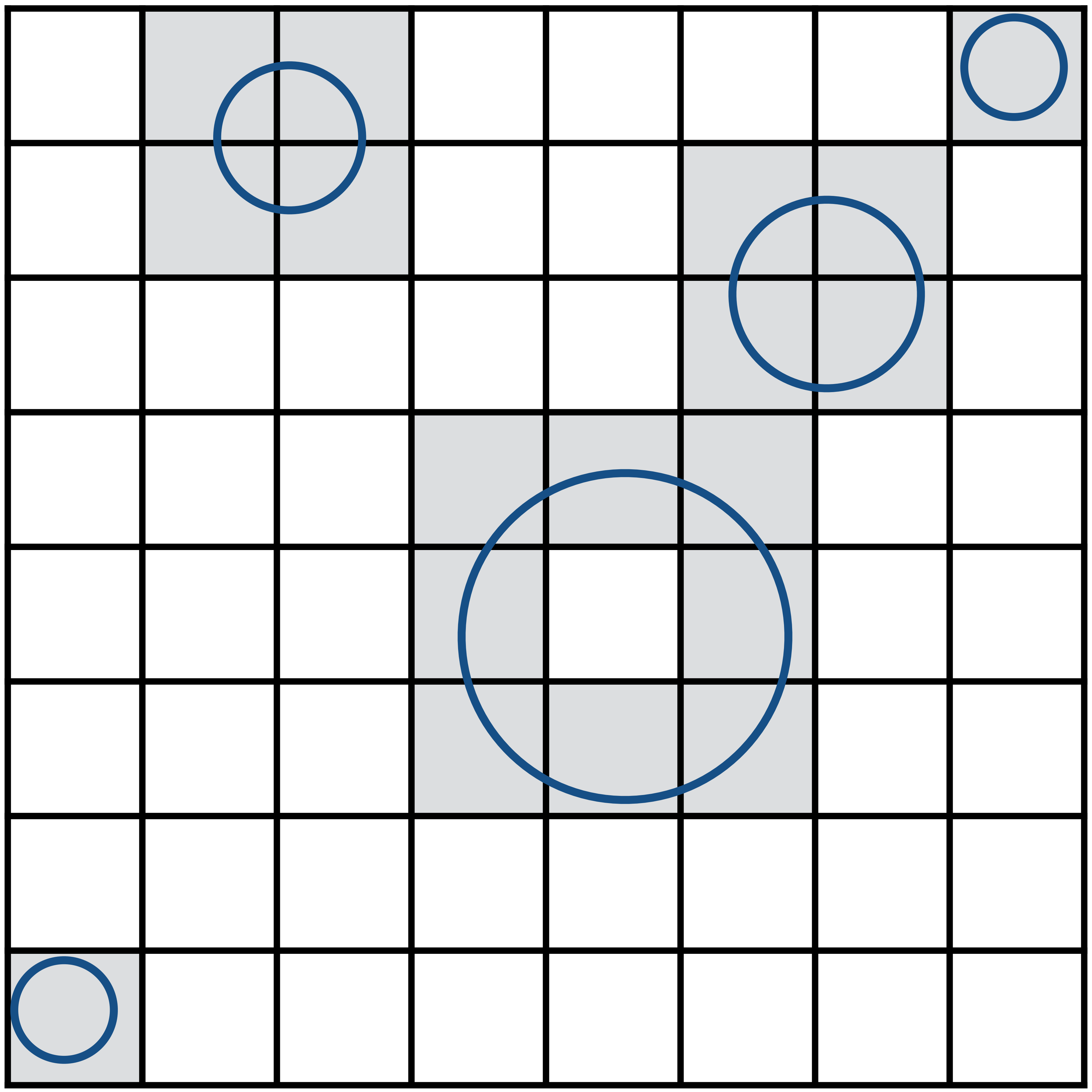
1. Find bounding box

Preprocess – Build Acceleration Grid



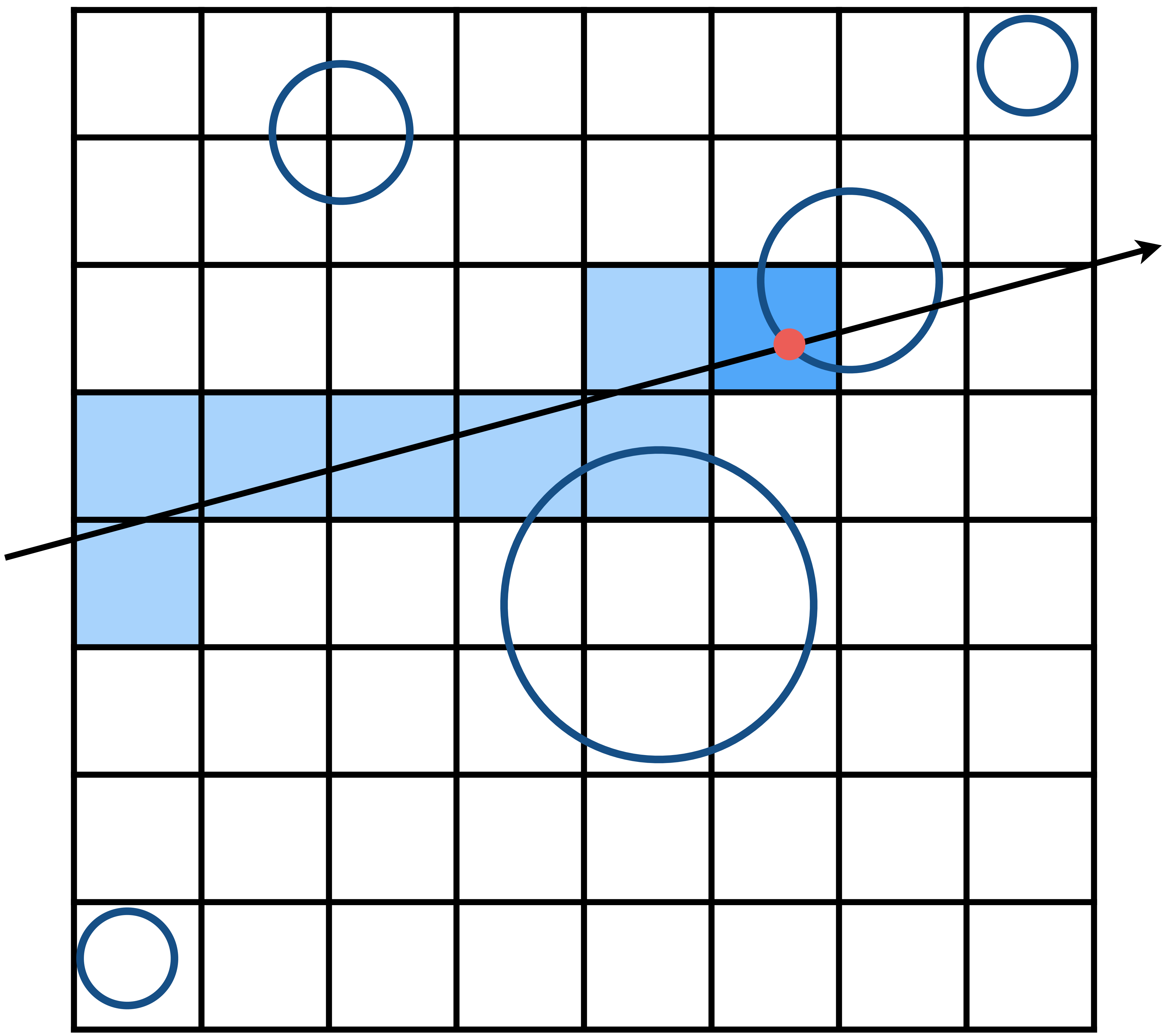
1. Find bounding box
2. Create grid

Preprocess – Build Acceleration Grid



1. Find bounding box
2. Create grid
3. Store each object in overlapping cells

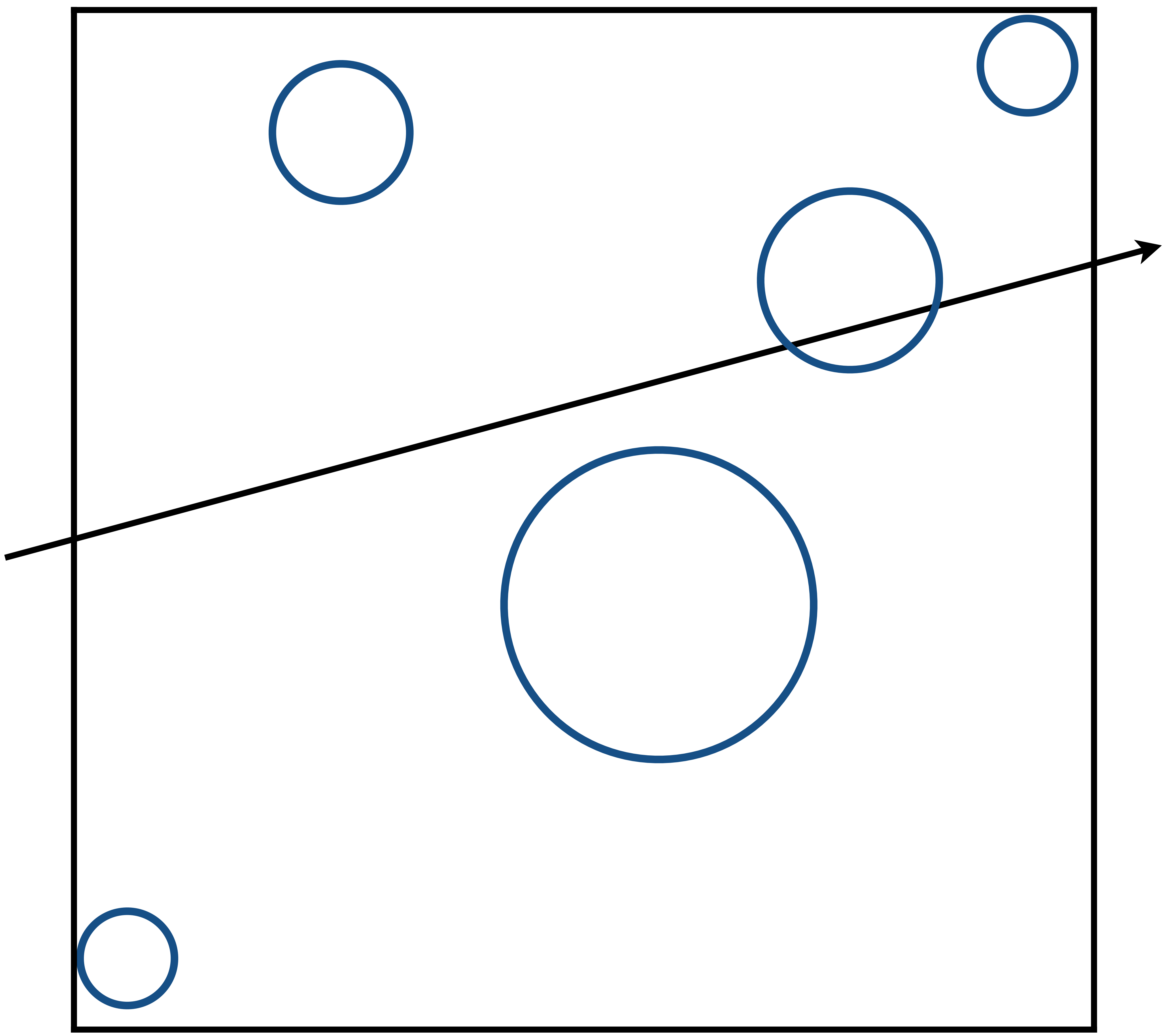
Ray-Scene Intersection



Step through grid in ray traversal order (3D line - 3D DDA)

For each grid cell
  Test intersection with all objects stored at that cell

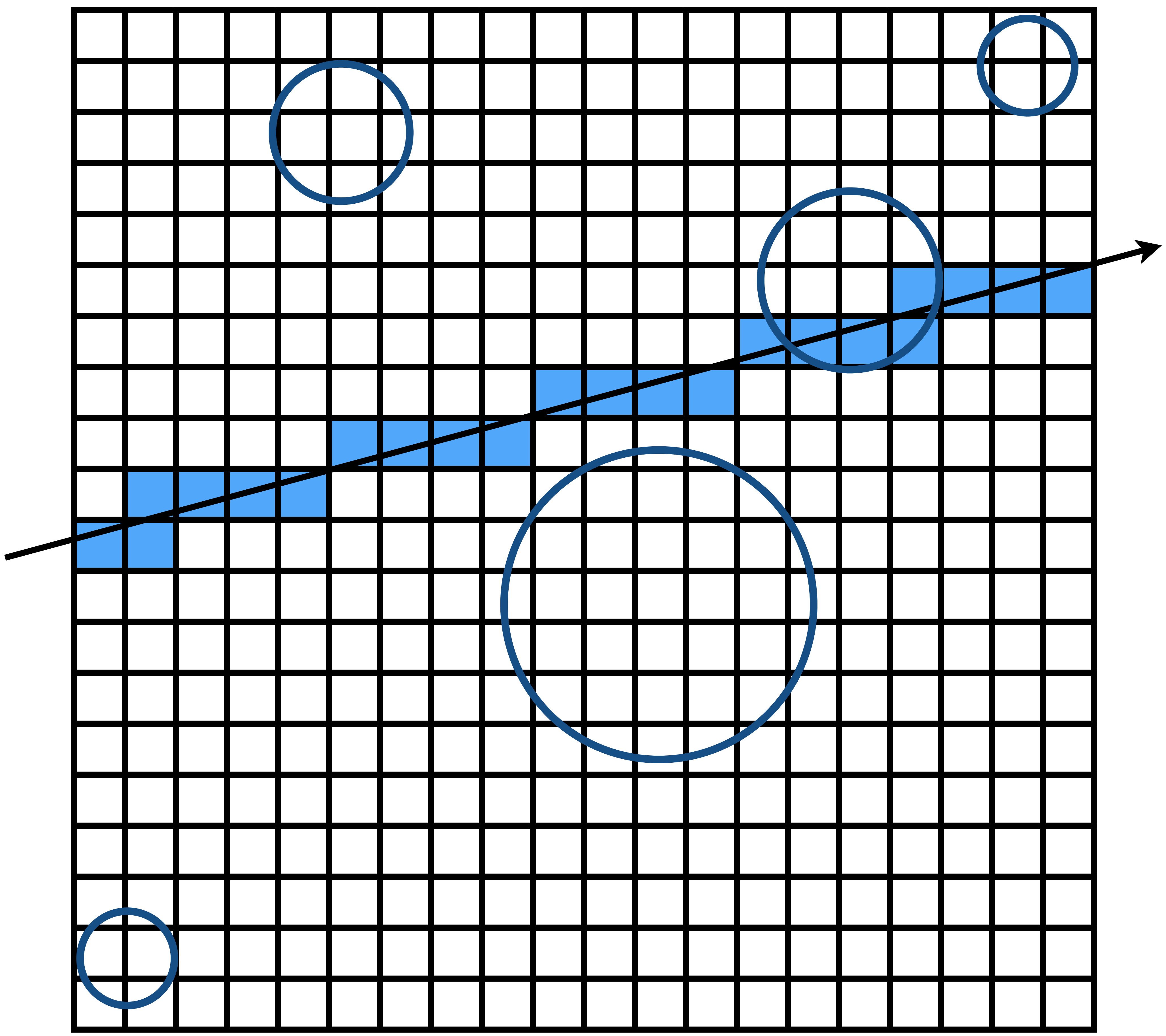
Grid Resolution?



One cell

- No speedup

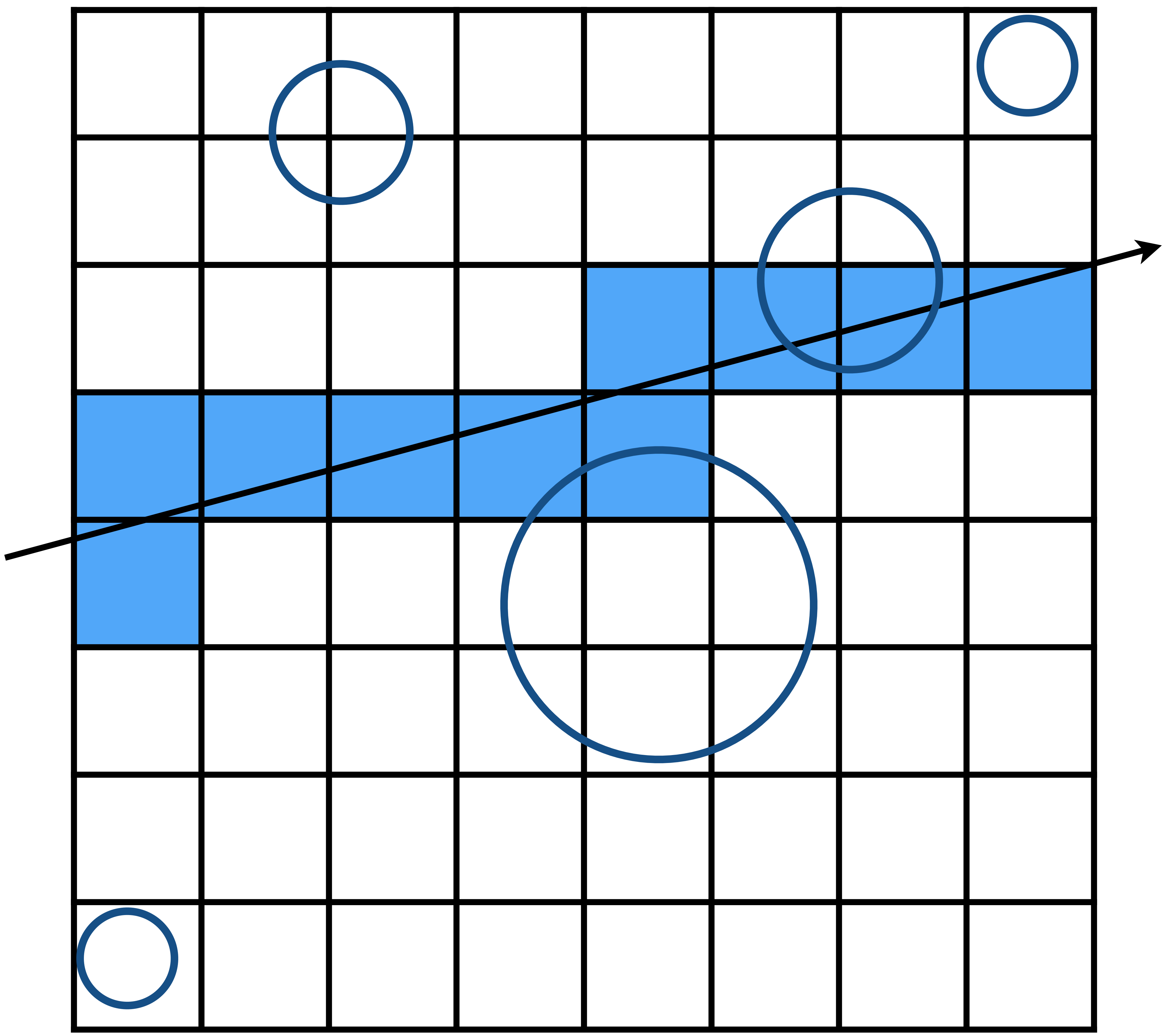
Grid Resolution?



Too many cells

- Inefficiency due to extraneous grid traversal

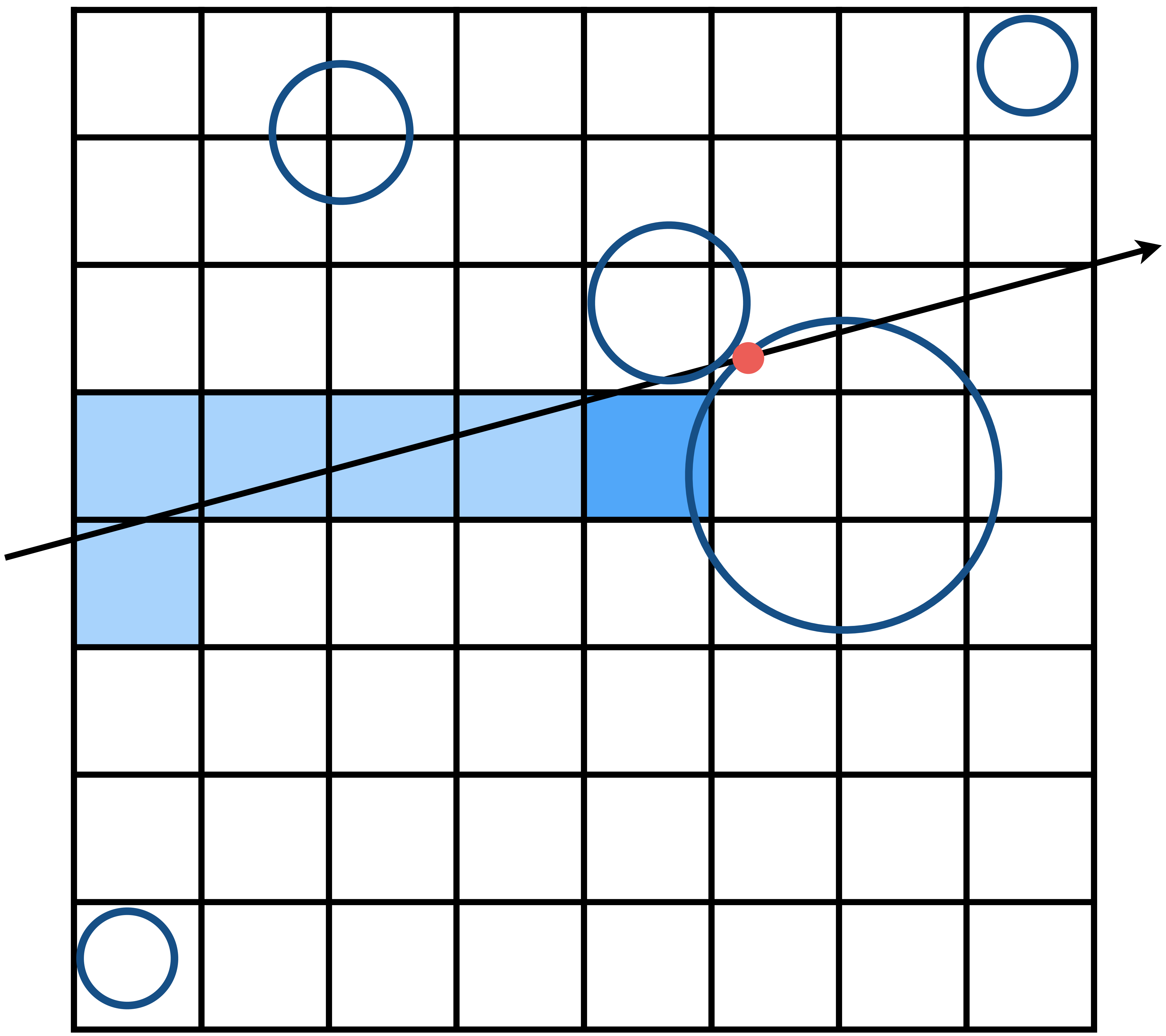
Grid Resolution?



Heuristic:

- #cells = C * #objs
- C ≈ 27 in 3D

Careful! Objects Overlapping Multiple Cells



What goes wrong here?

- First intersection found (red) is not the nearest!

Solution?

- Check intersection point is inside cell

Optimize

- Cache intersection to avoid re-testing (mailboxing)

Uniform Grids – When They Work Well

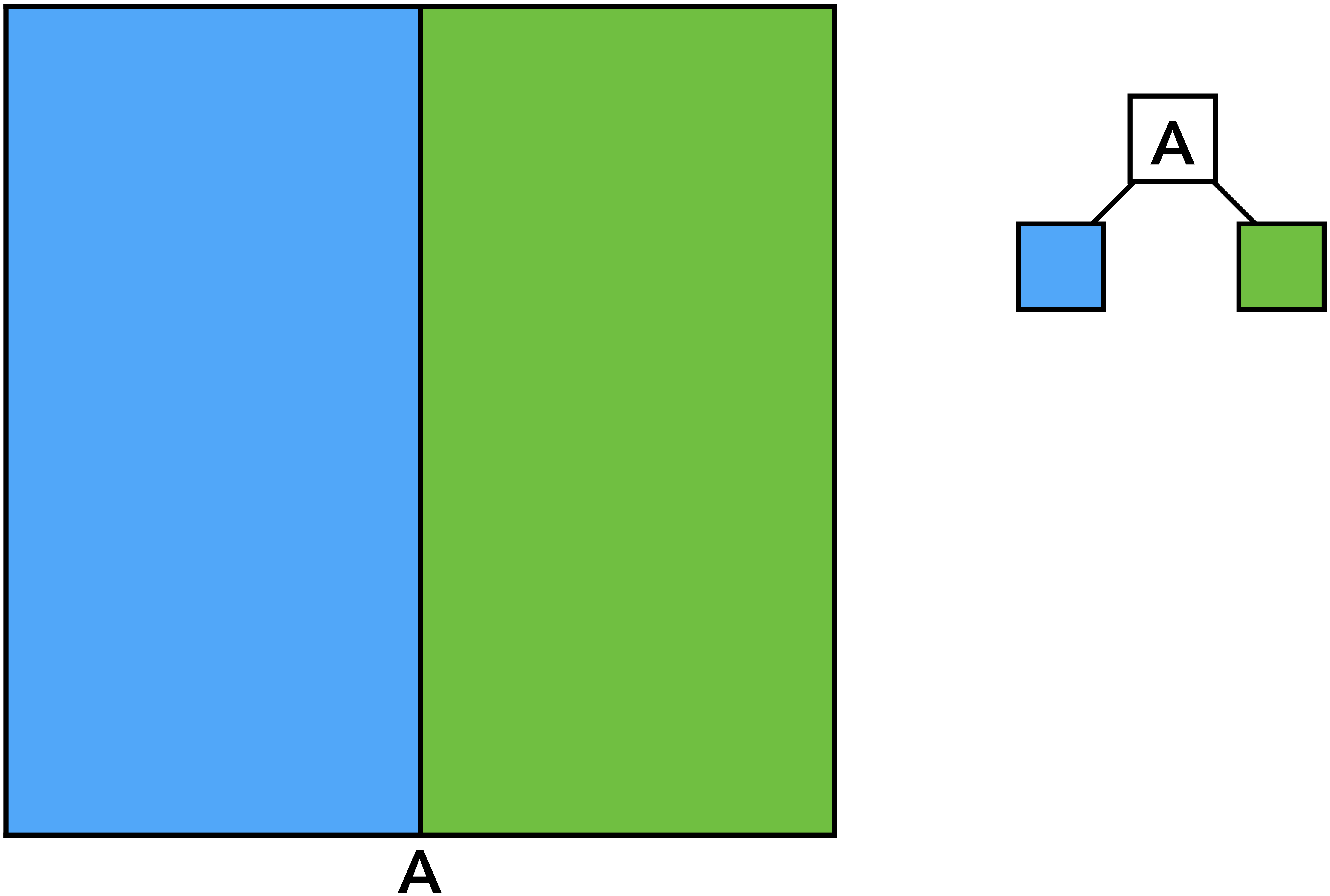
Grids work well on large collections of objects
that are distributed evenly in size and space

Uniform Grids – When They Fail

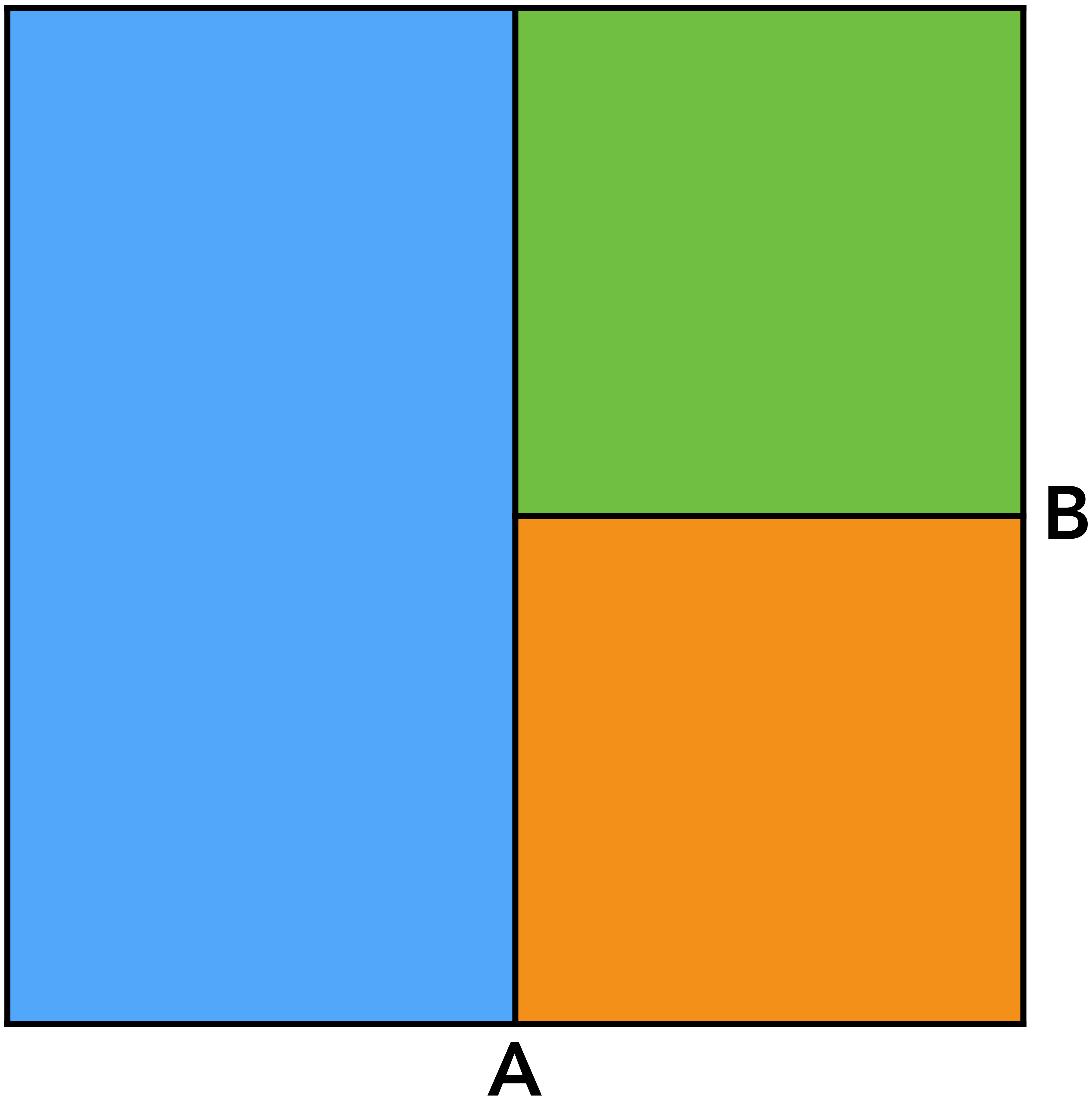
"Teapot in a stadium" problem

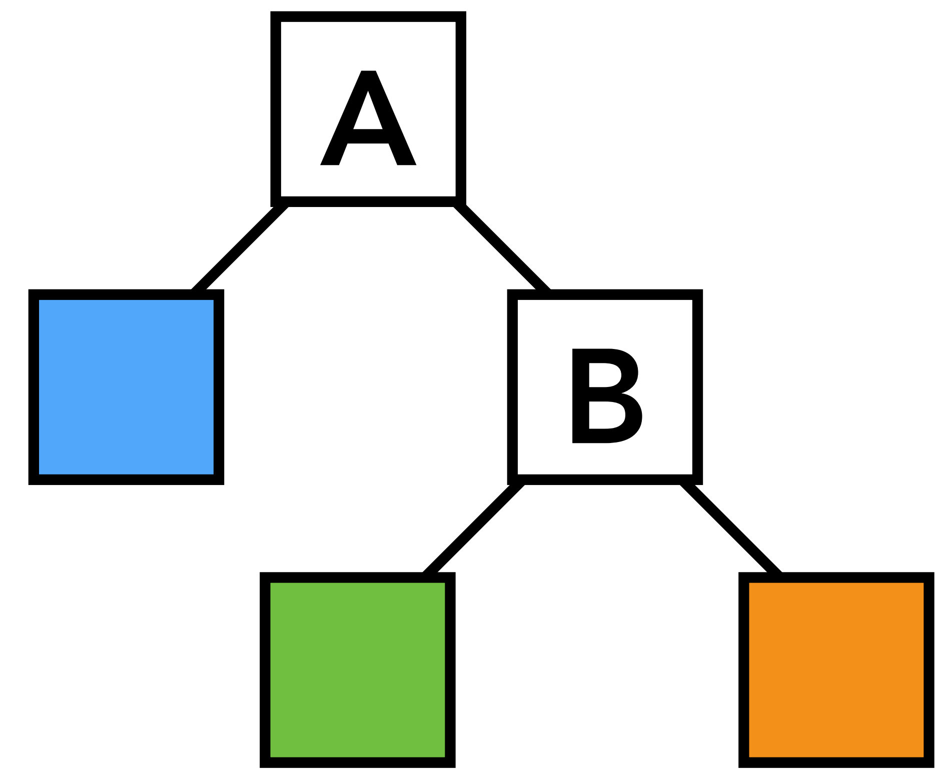
Non-Uniform Spatial Partitions: Spatial Hierarchies

Spatial Hierarchies

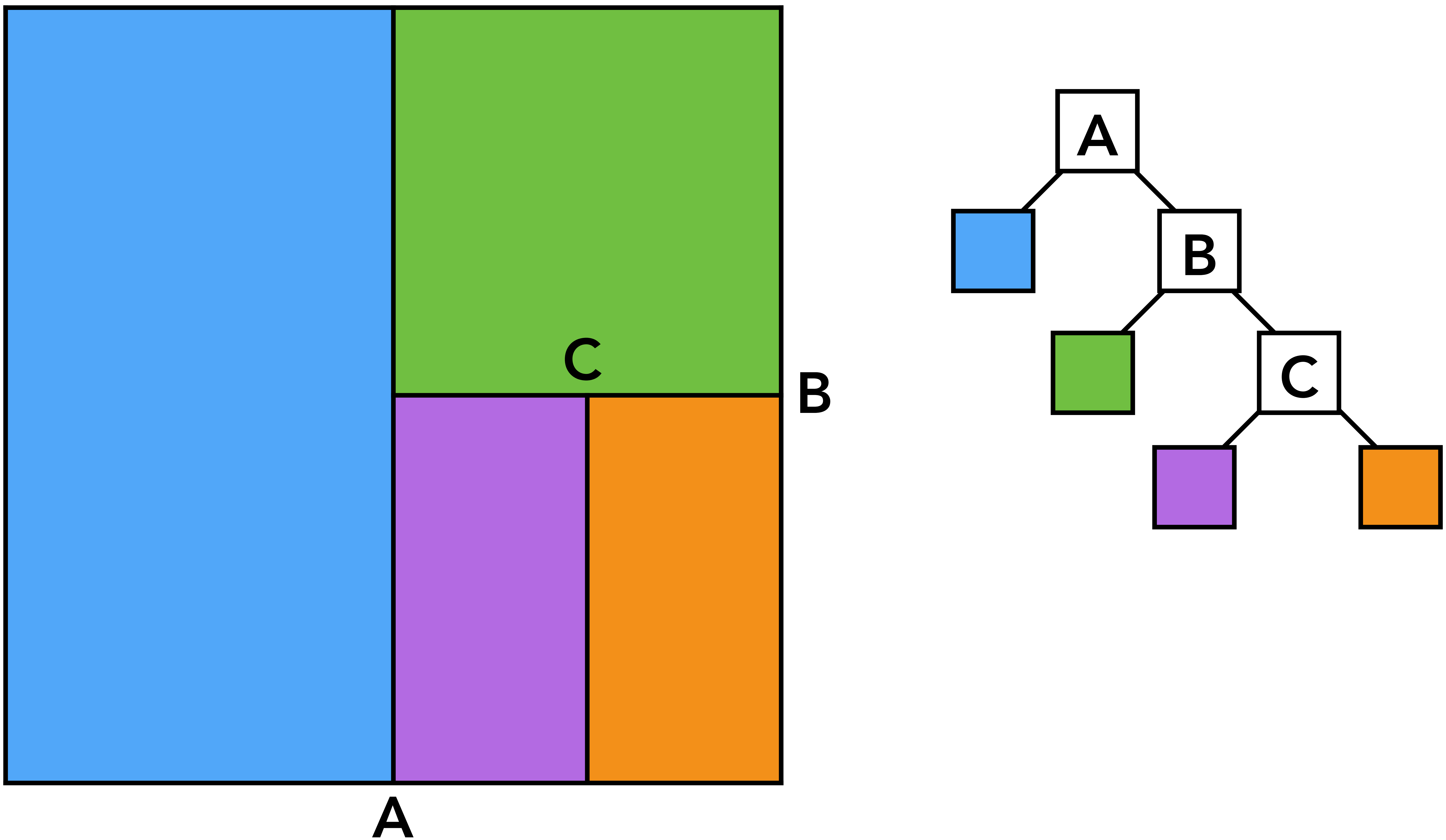


Spatial Hierarchies

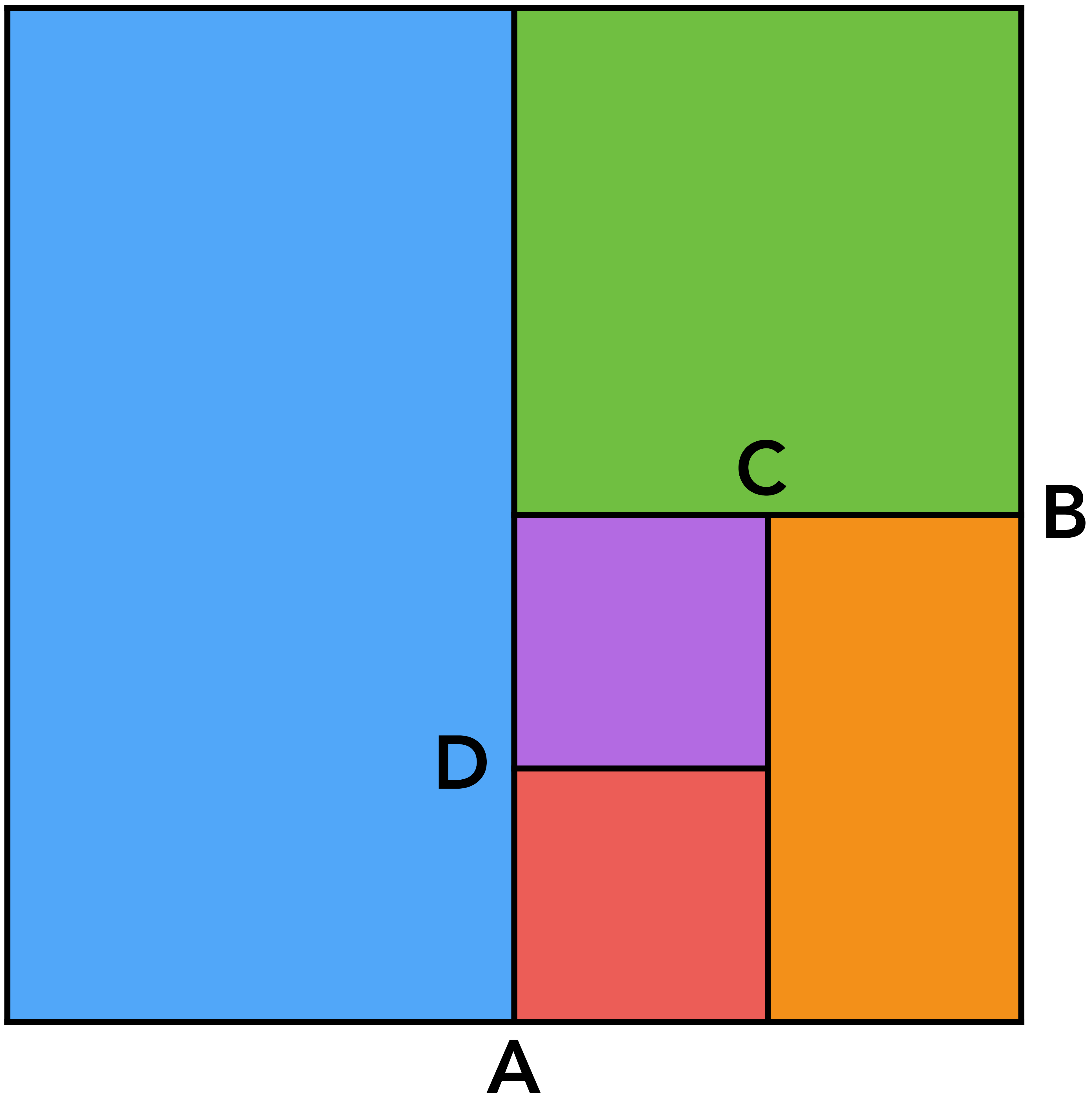


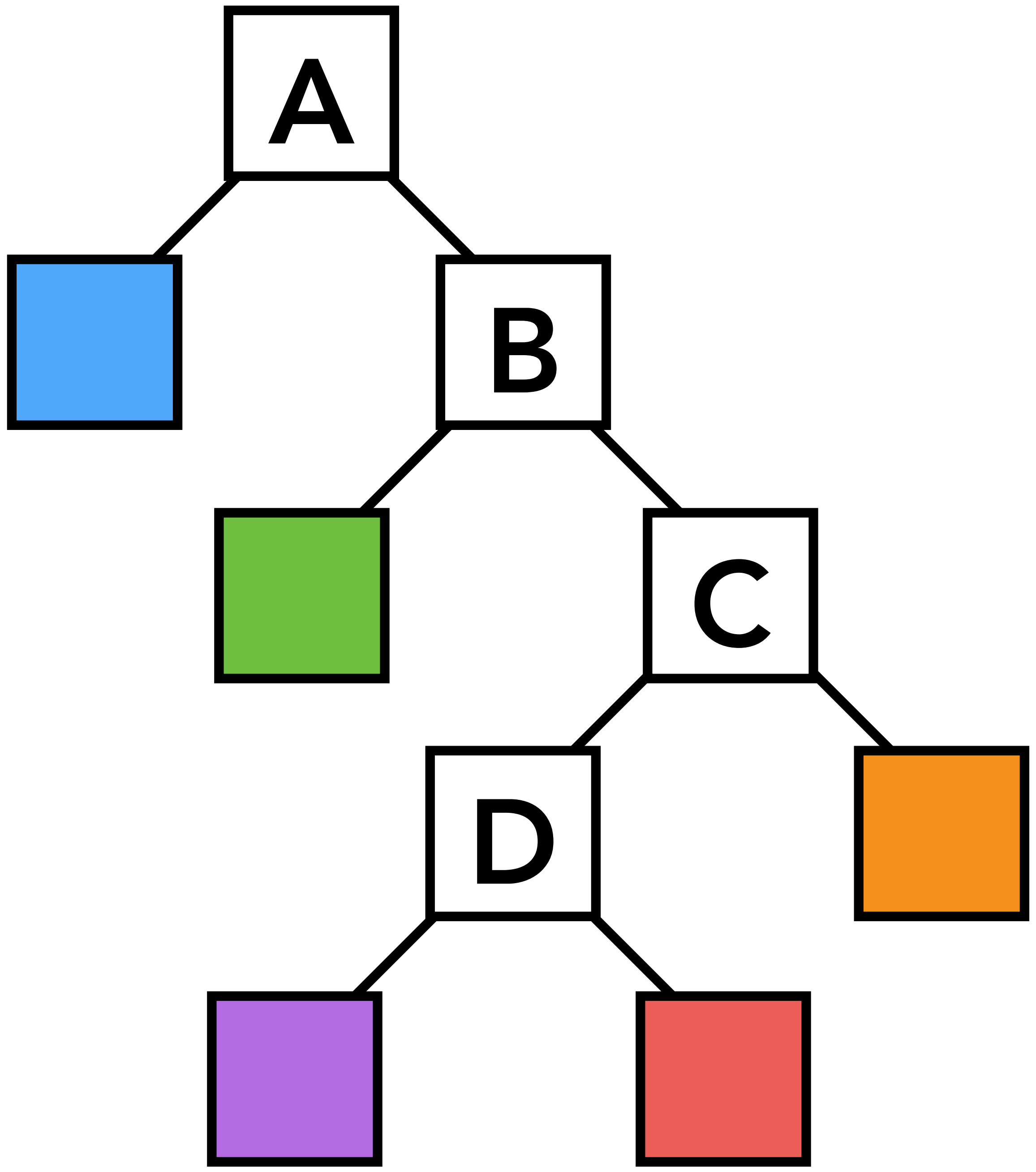


Spatial Hierarchies

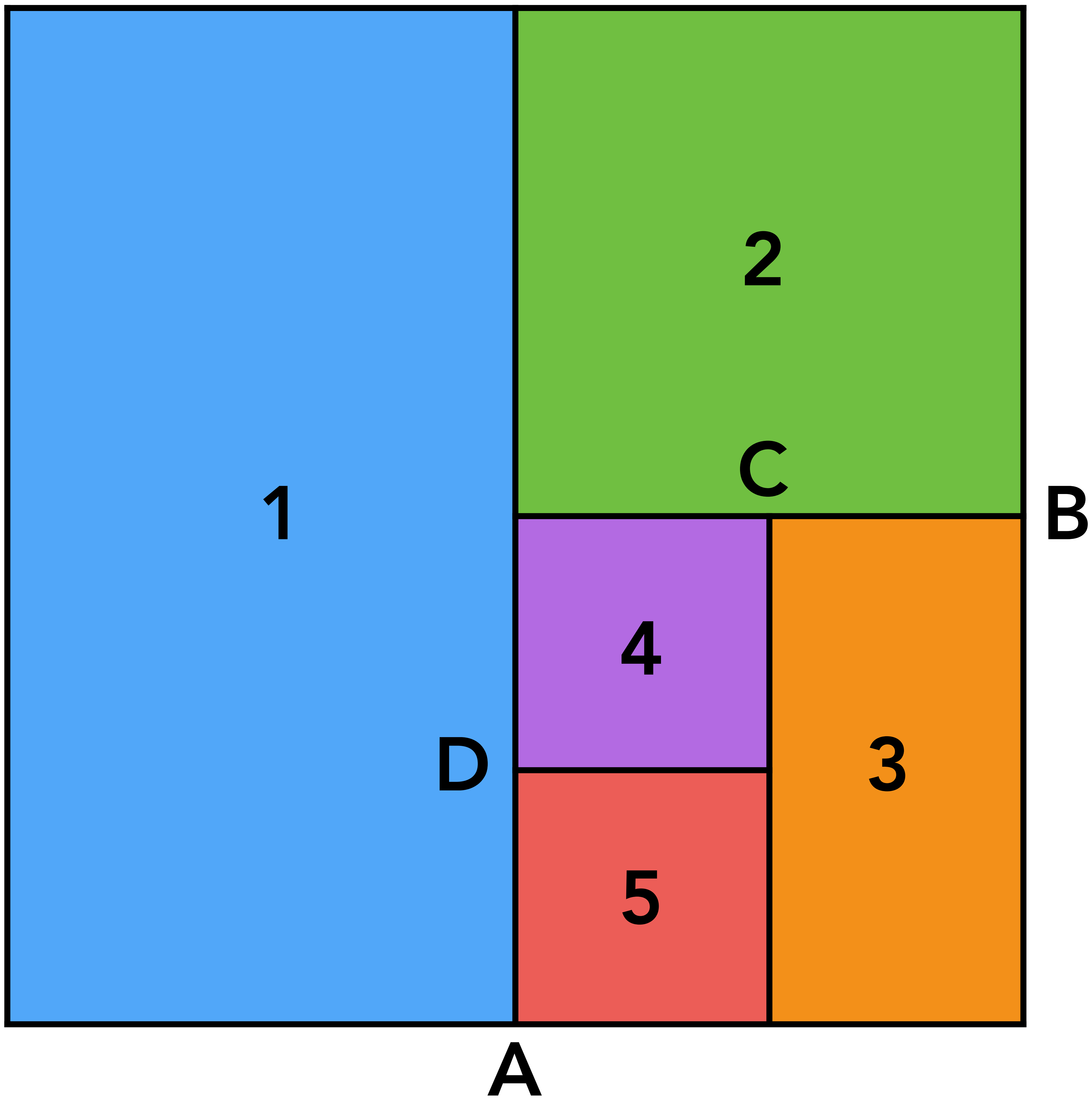


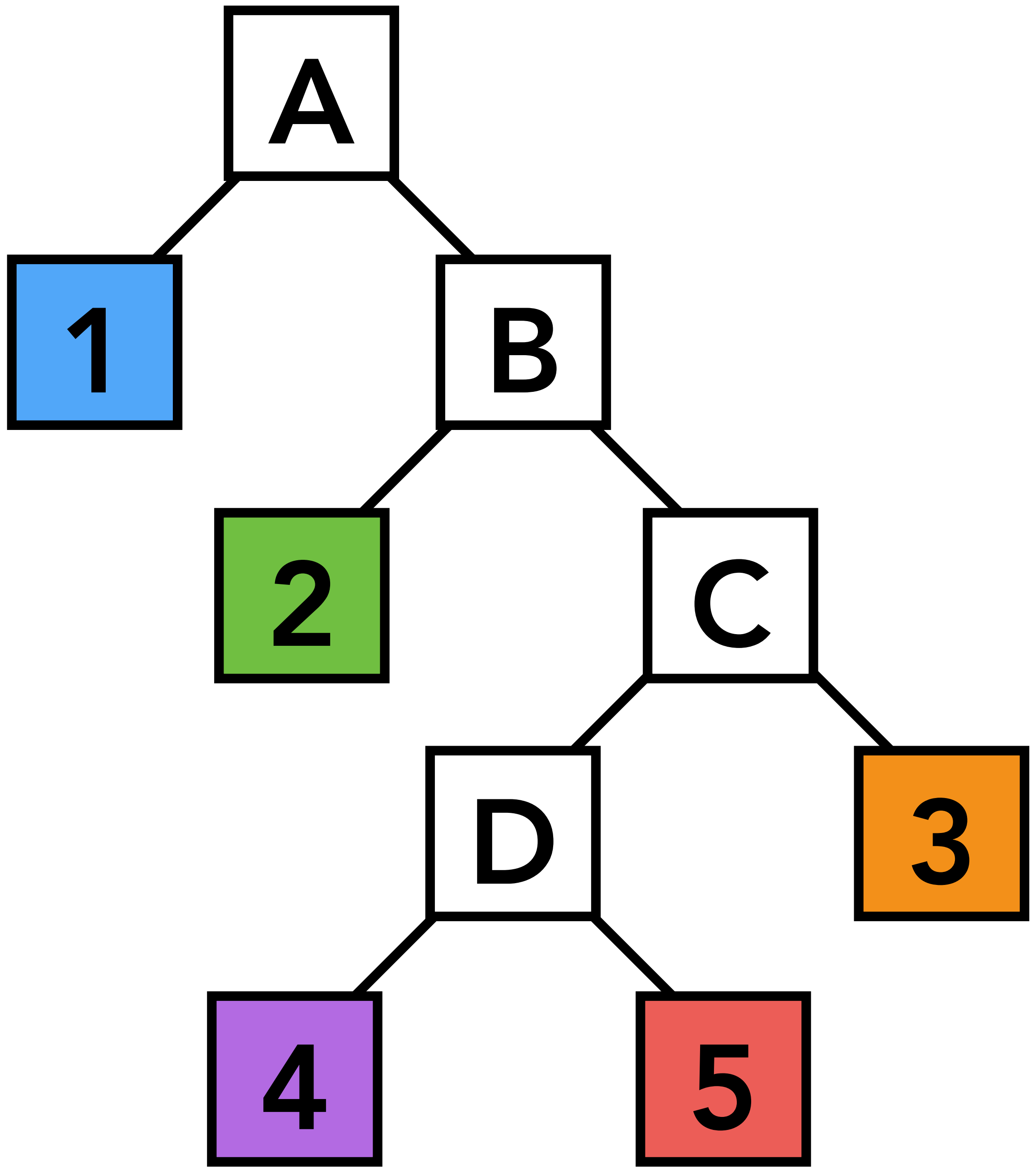
Spatial Hierarchies



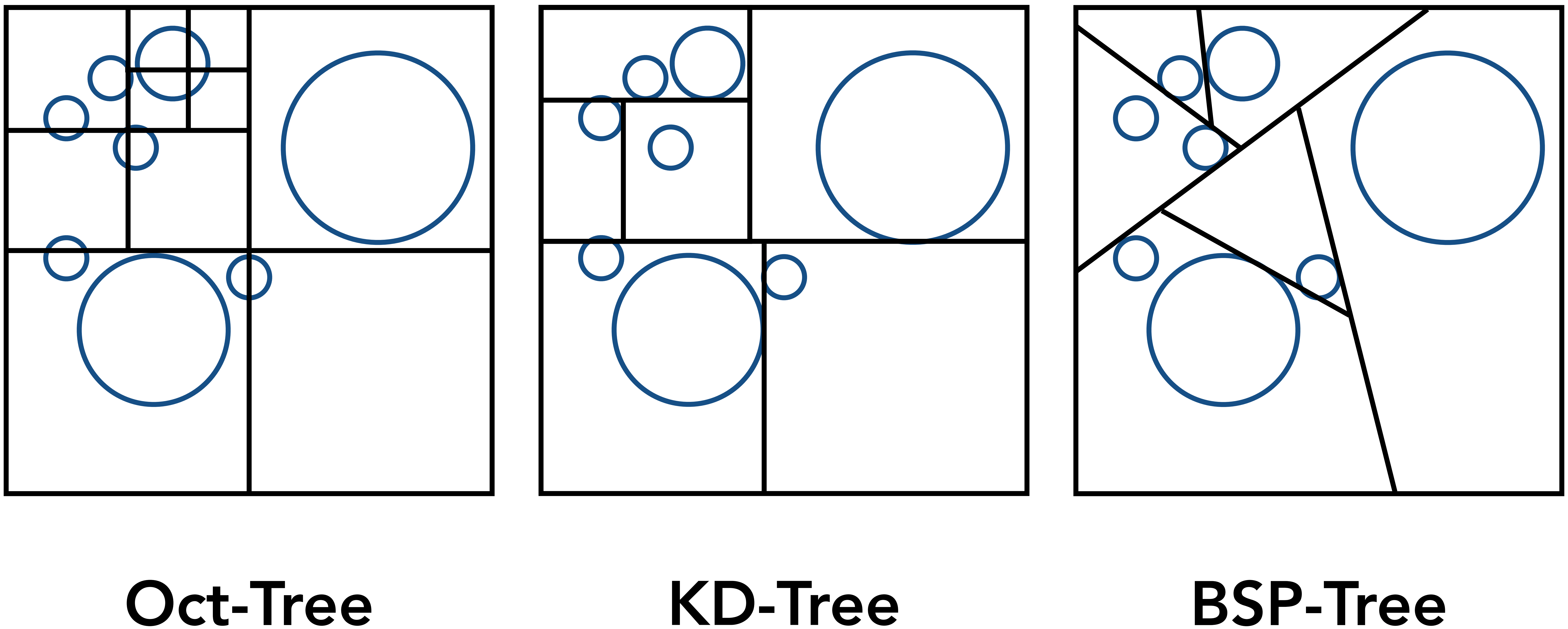


Spatial Hierarchies





Spatial Partitioning Variants



Note: you could have these in both 2D and 3D. In lecture we will illustrate principles in 2D, but for assignment you will implement 3D versions.

KD-Trees

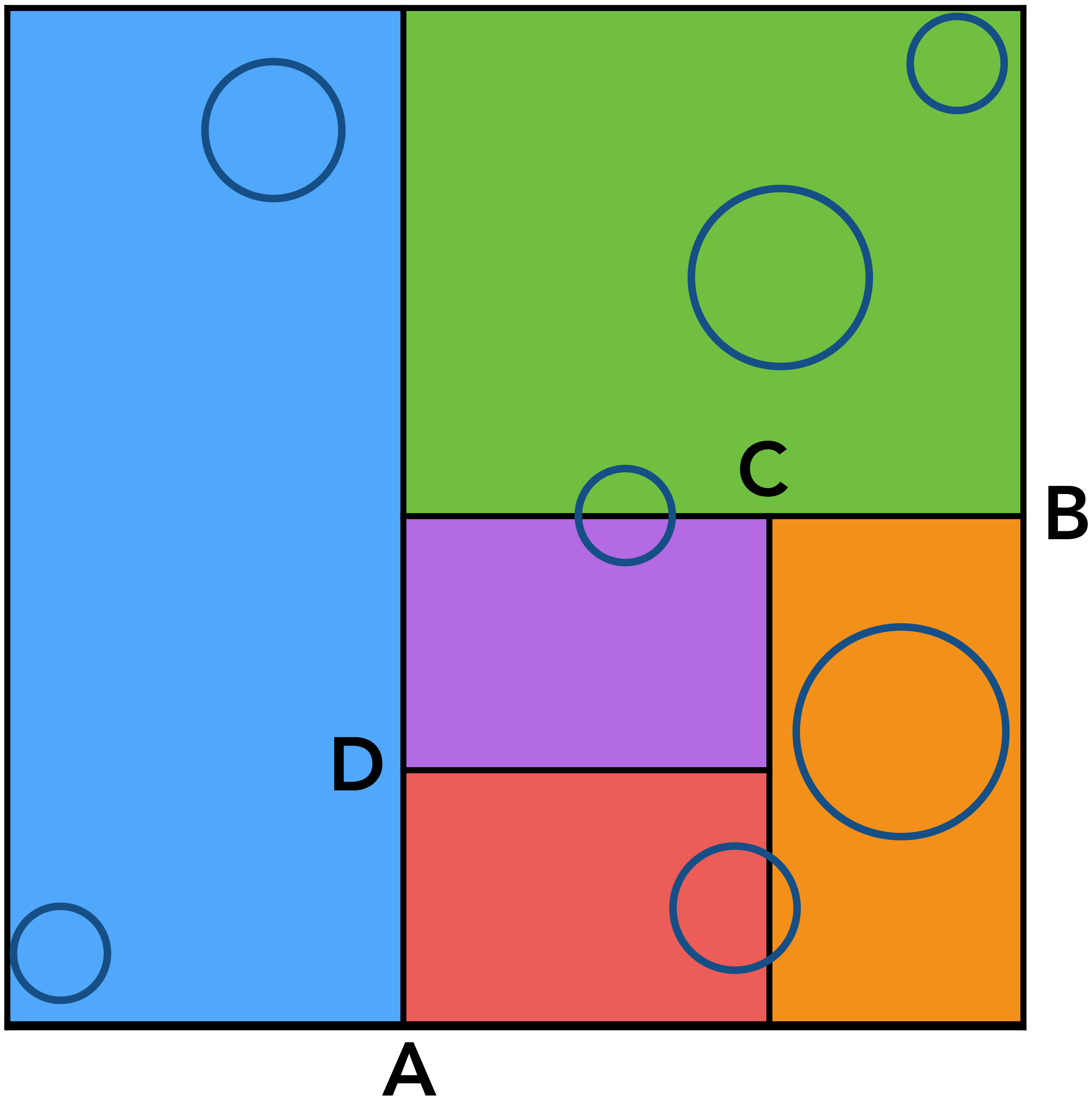
Internal nodes store

- split axis: x-, y-, or z-axis
- split position: coordinate of split plane along axis
- children: reference to child nodes

Leaf nodes store

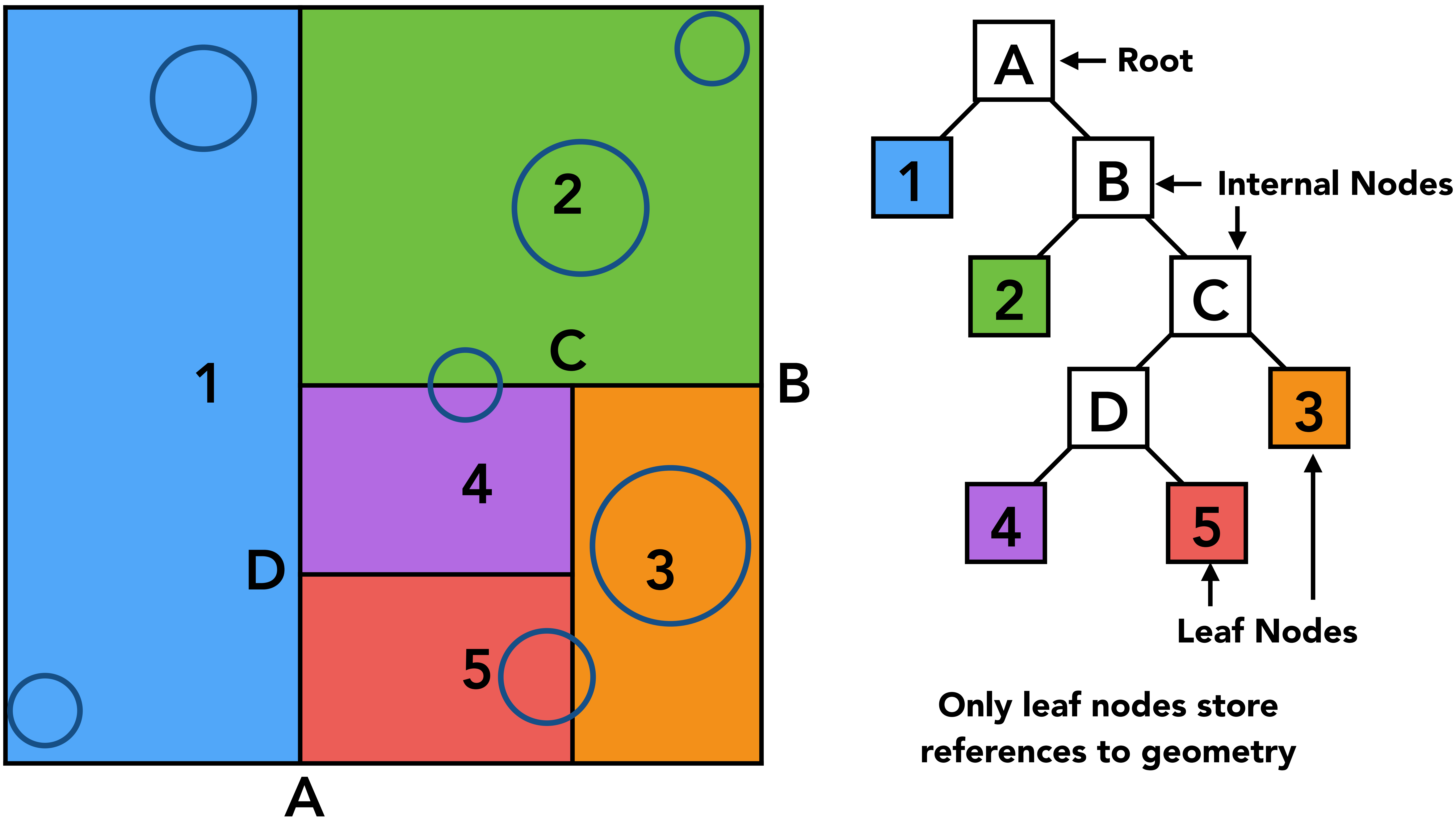
- list of objects
- mailbox information

KD-Tree Pre-Processing



- Find bounding box
- Recursively split cells, axis-aligned planes
- Until termination criteria met (e.g. max #splits or min #objs)
- Store obj references with each leaf node

KD-Tree Pre-Processing



Only leaf nodes store references to geometry

KD-Tree Pre-Processing

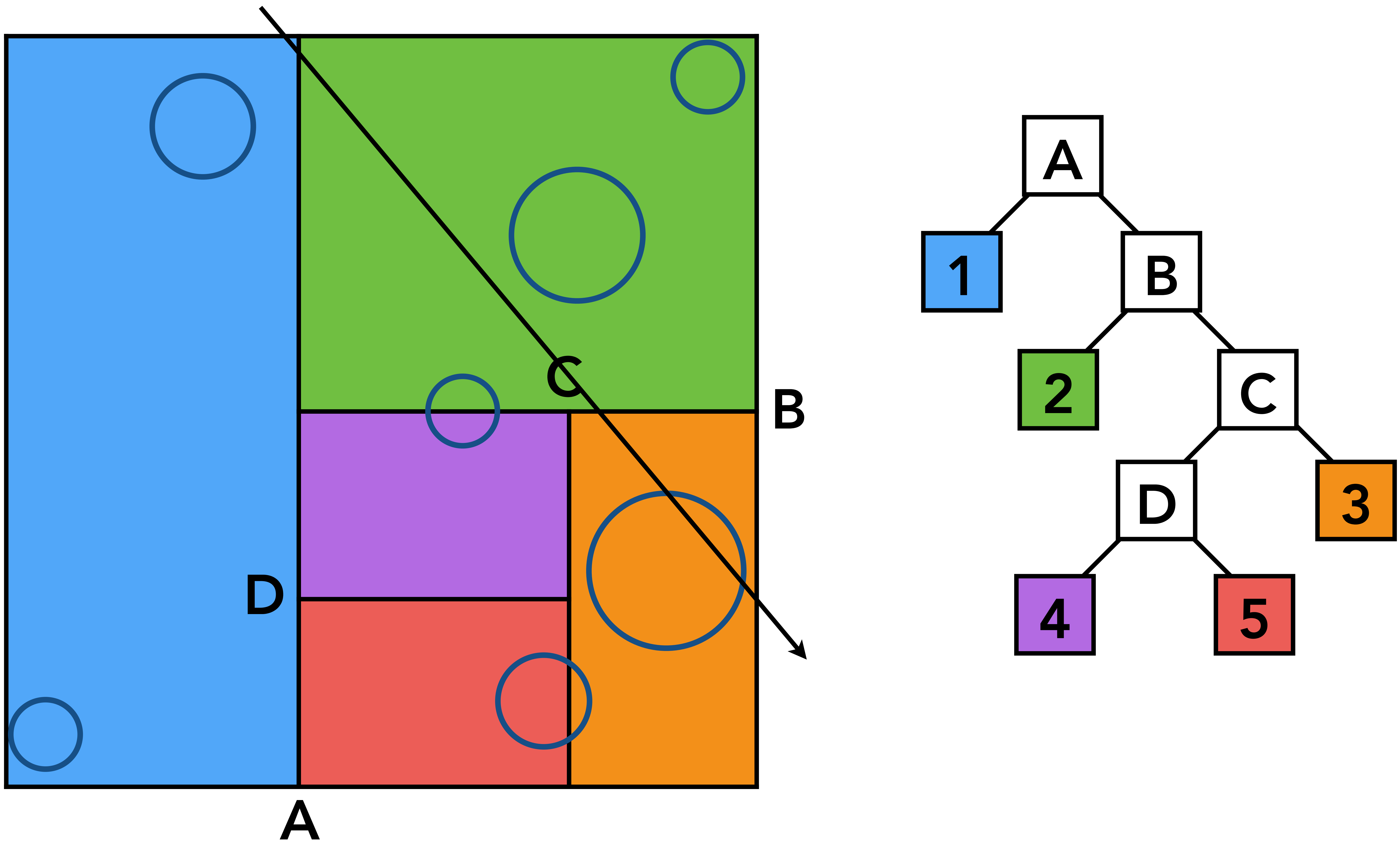
Choosing the split plane

- Simple: midpoint, median split
- Ideal: split to minimize expected cost of ray intersection

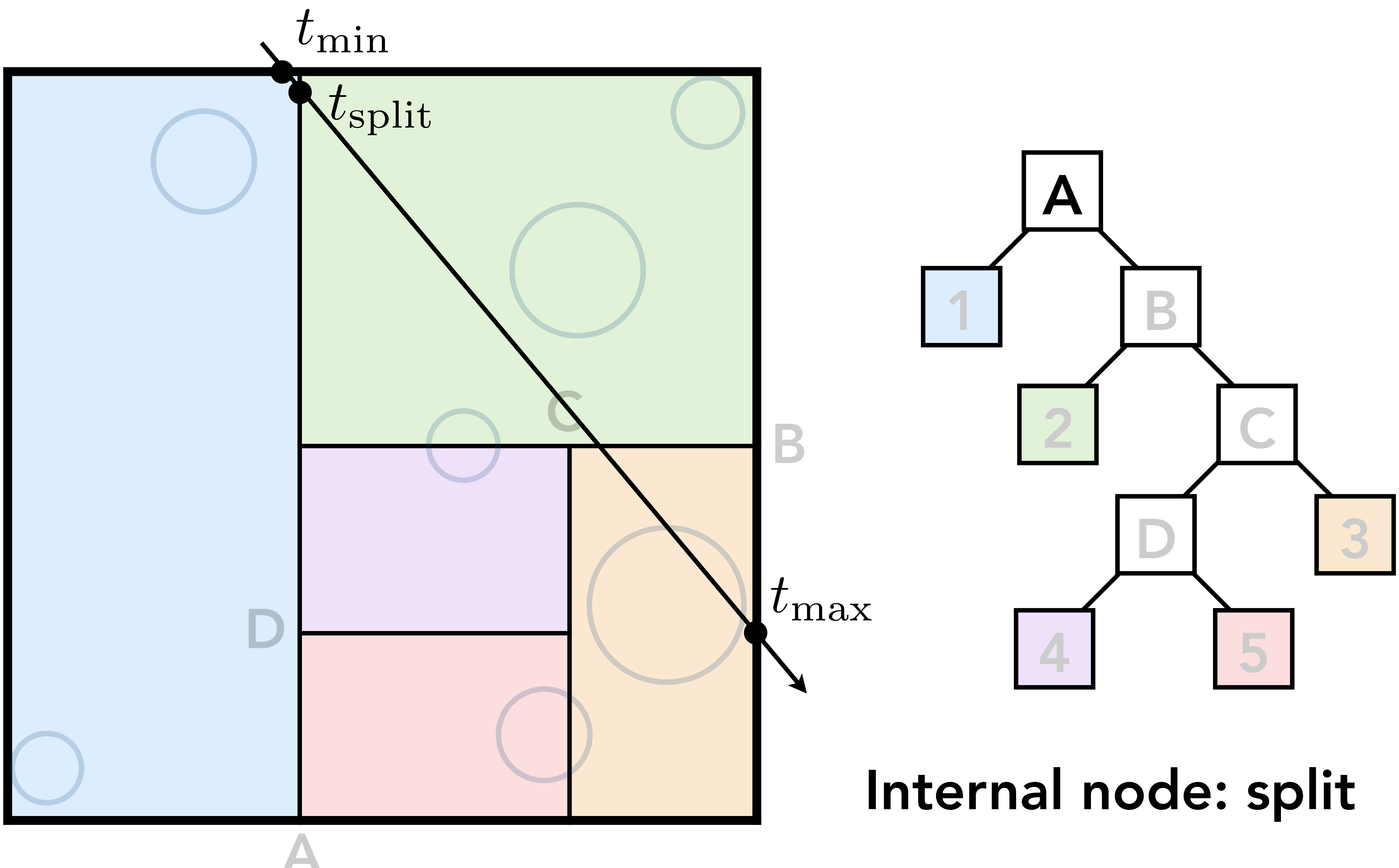
Termination criteria?

- Simple: common to prescribe maximum tree depth (empirical 8 + 1.3 log N, N = #objs) [PBRT]
- Ideal: stop when splitting does not reduce expected cost of ray intersection

Top-Down Recursive In-Order Traversal

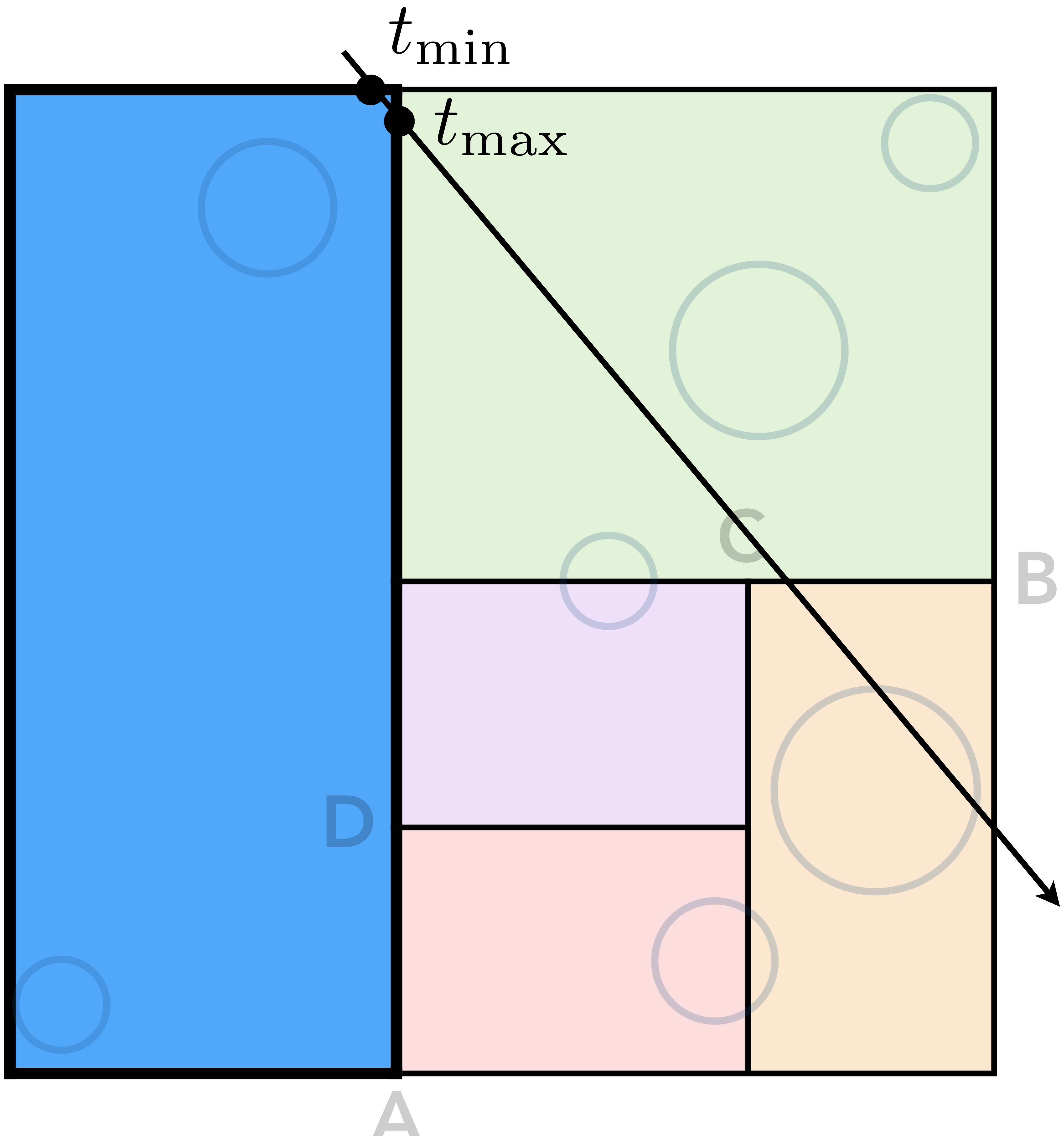


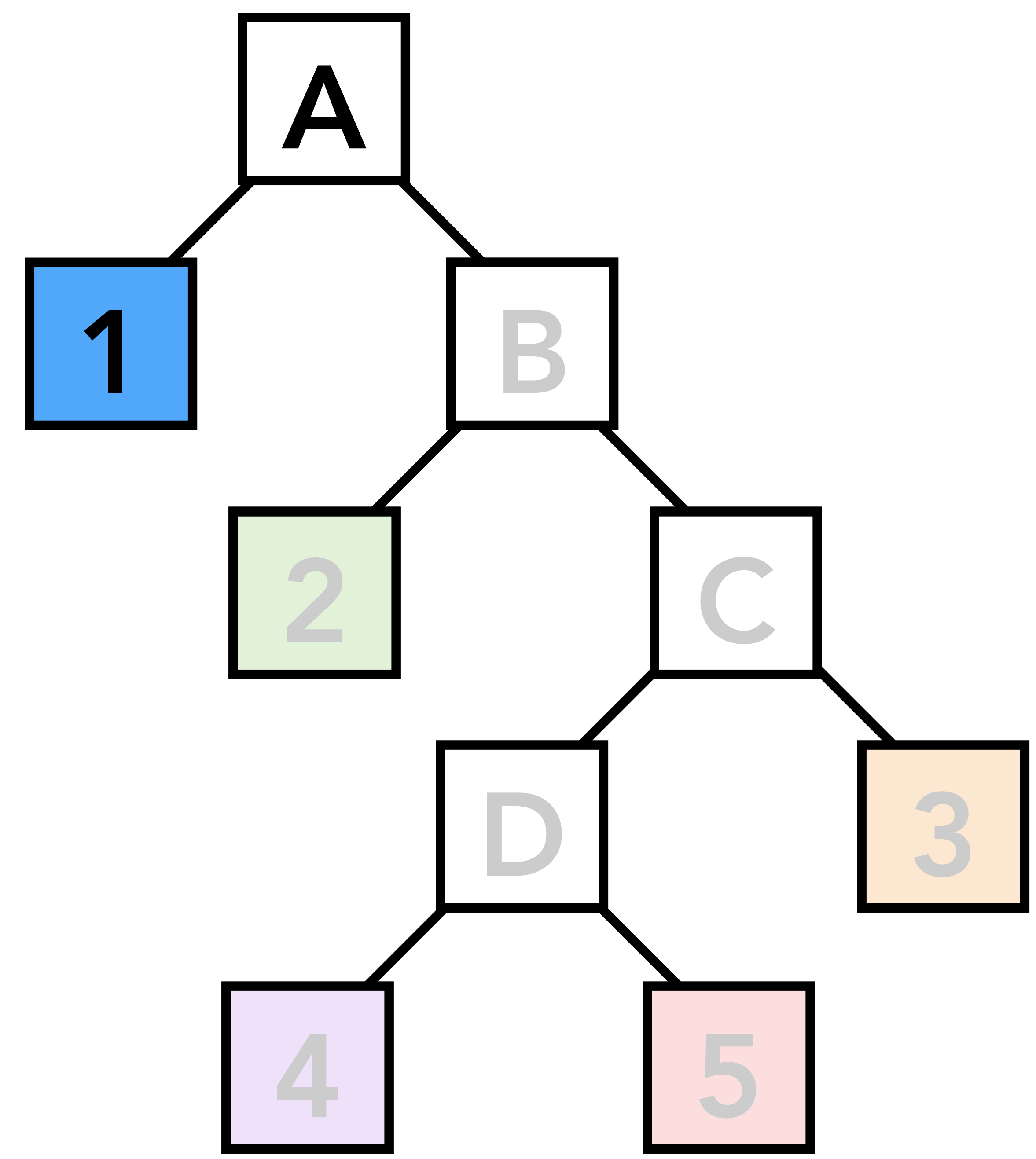
Top-Down Recursive In-Order Traversal



Internal node: split

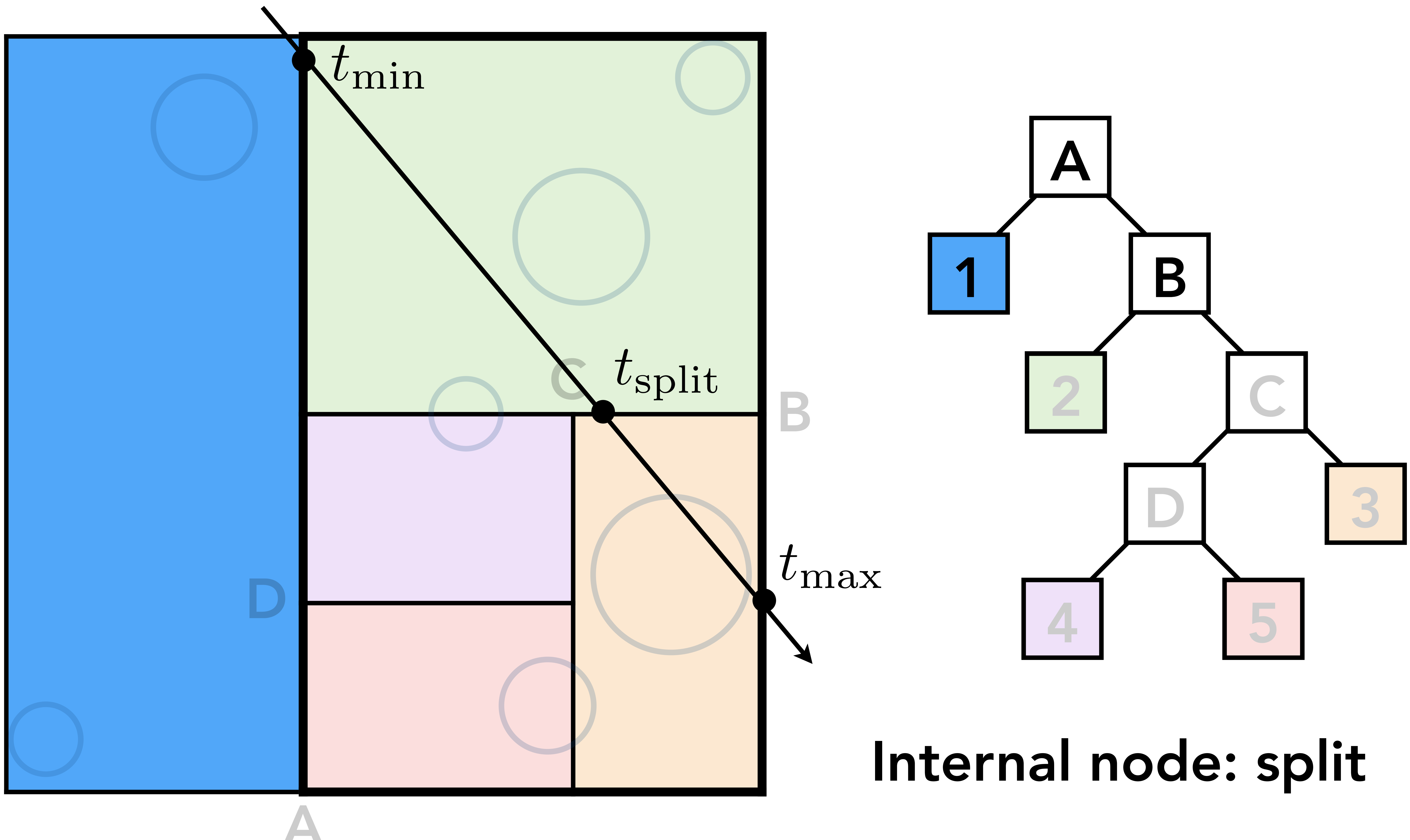
Top-Down Recursive In-Order Traversal





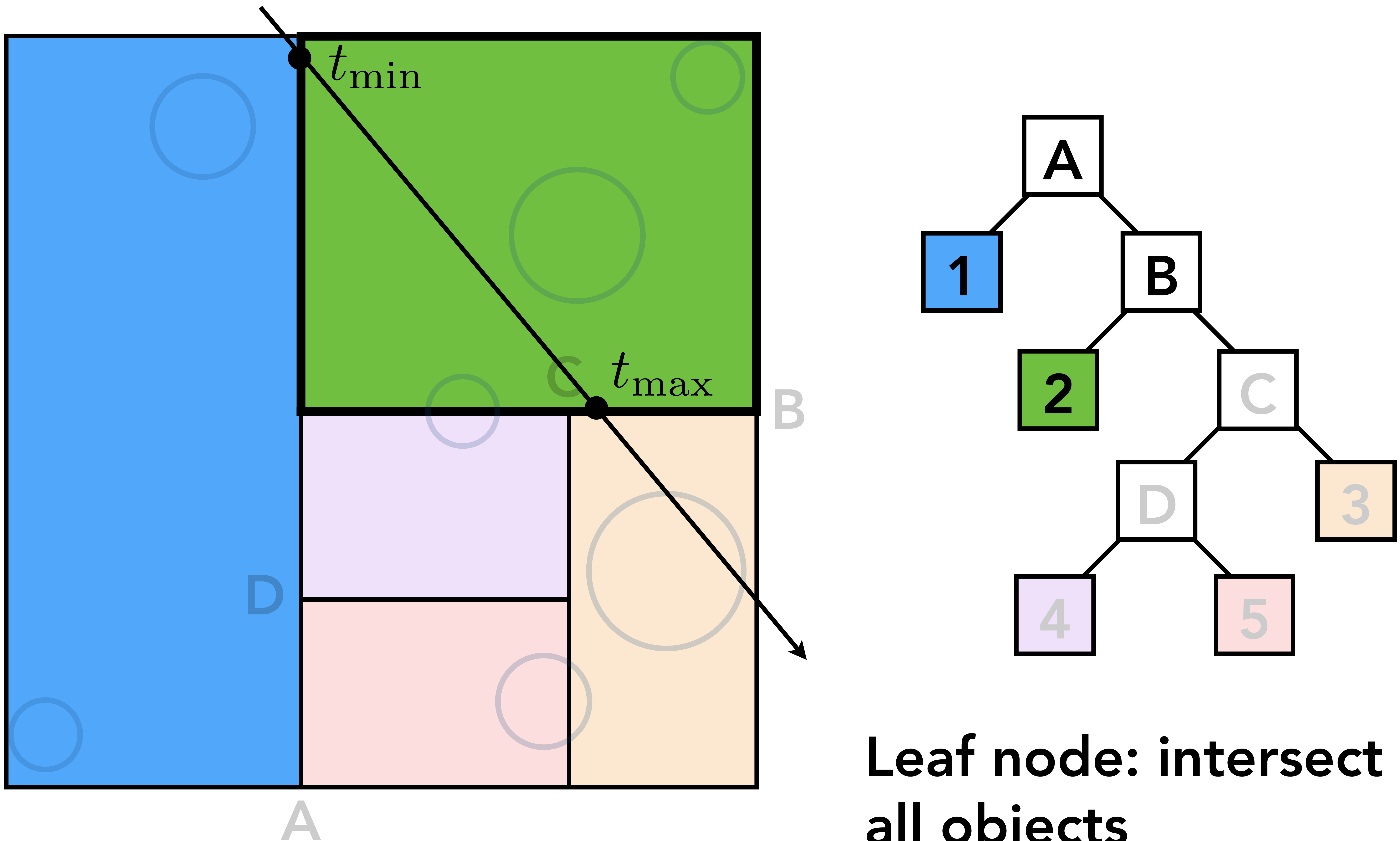
Leaf node: intersect all objects

Top-Down Recursive In-Order Traversal



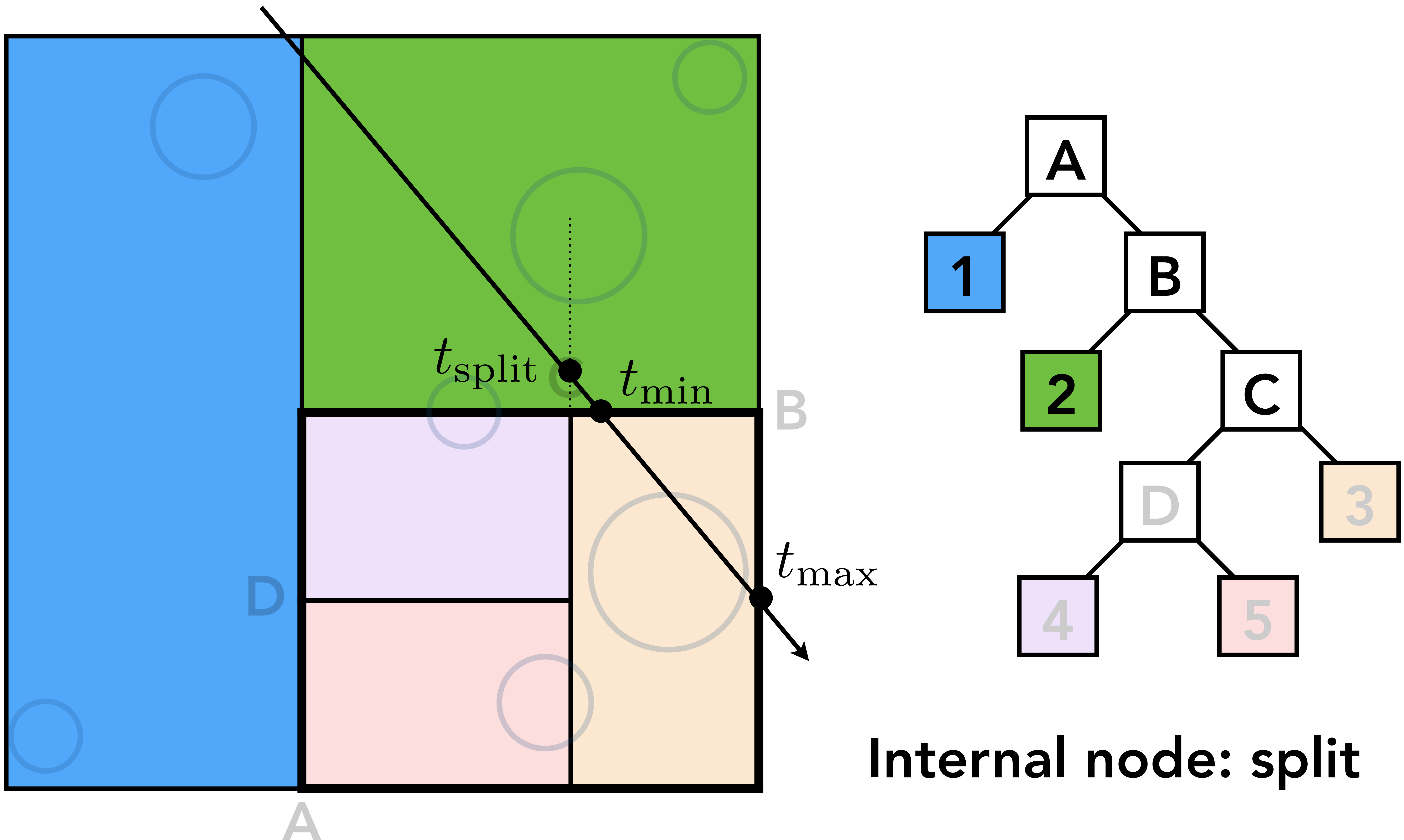
Internal node: split

Top-Down Recursive In-Order Traversal



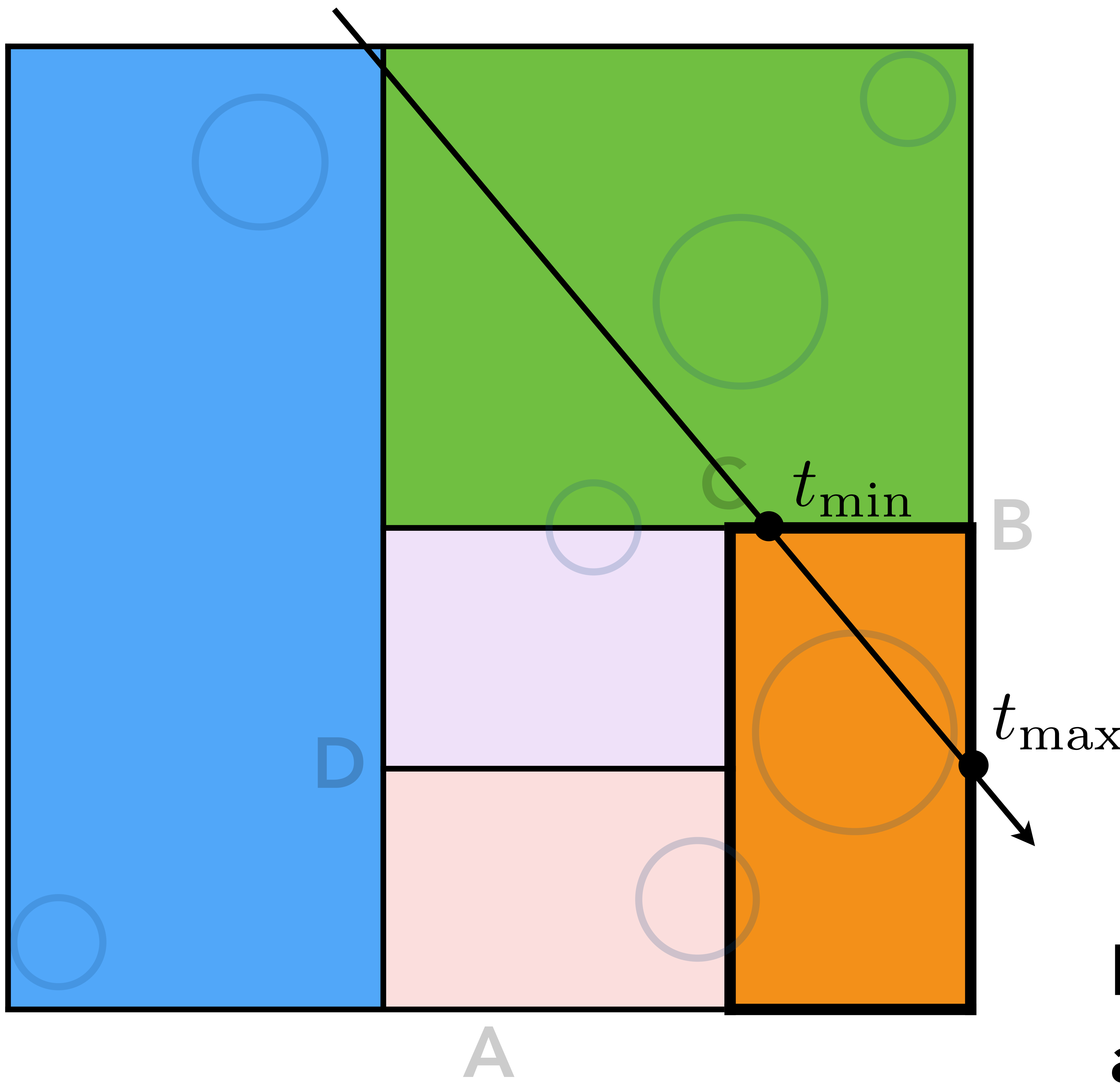
Leaf node: intersect all objects

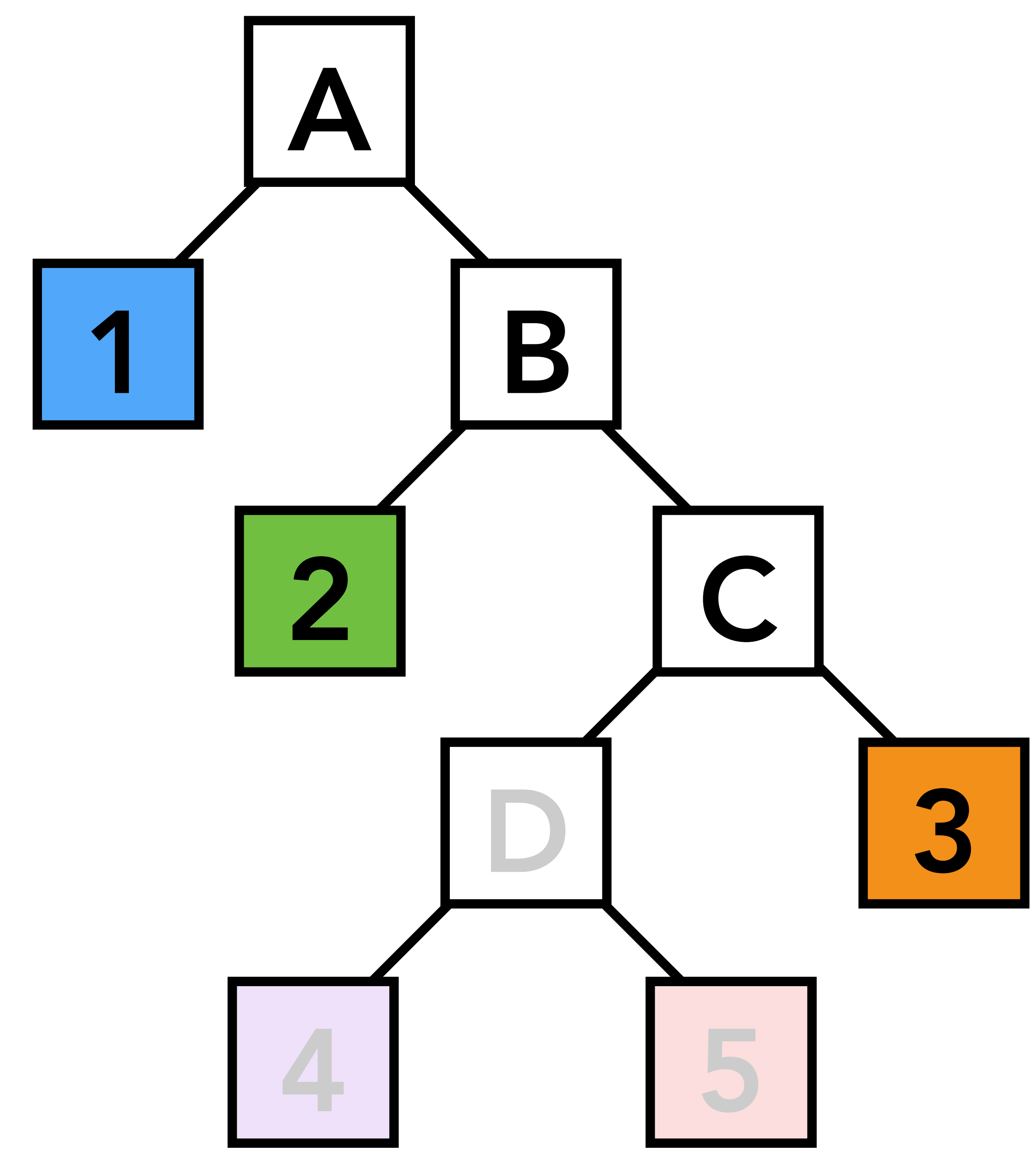
Top-Down Recursive In-Order Traversal



Internal node: split

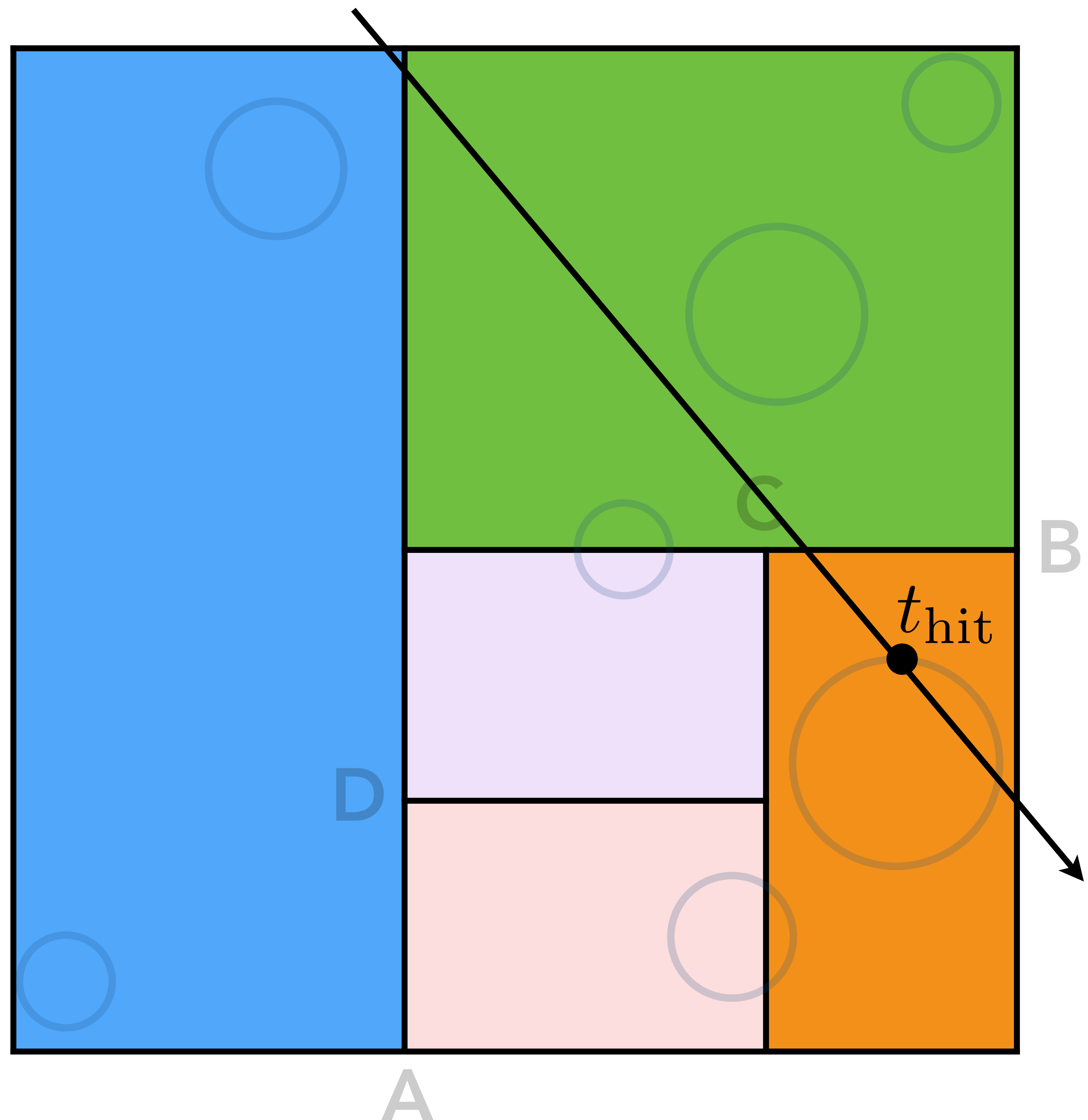
Top-Down Recursive In-Order Traversal

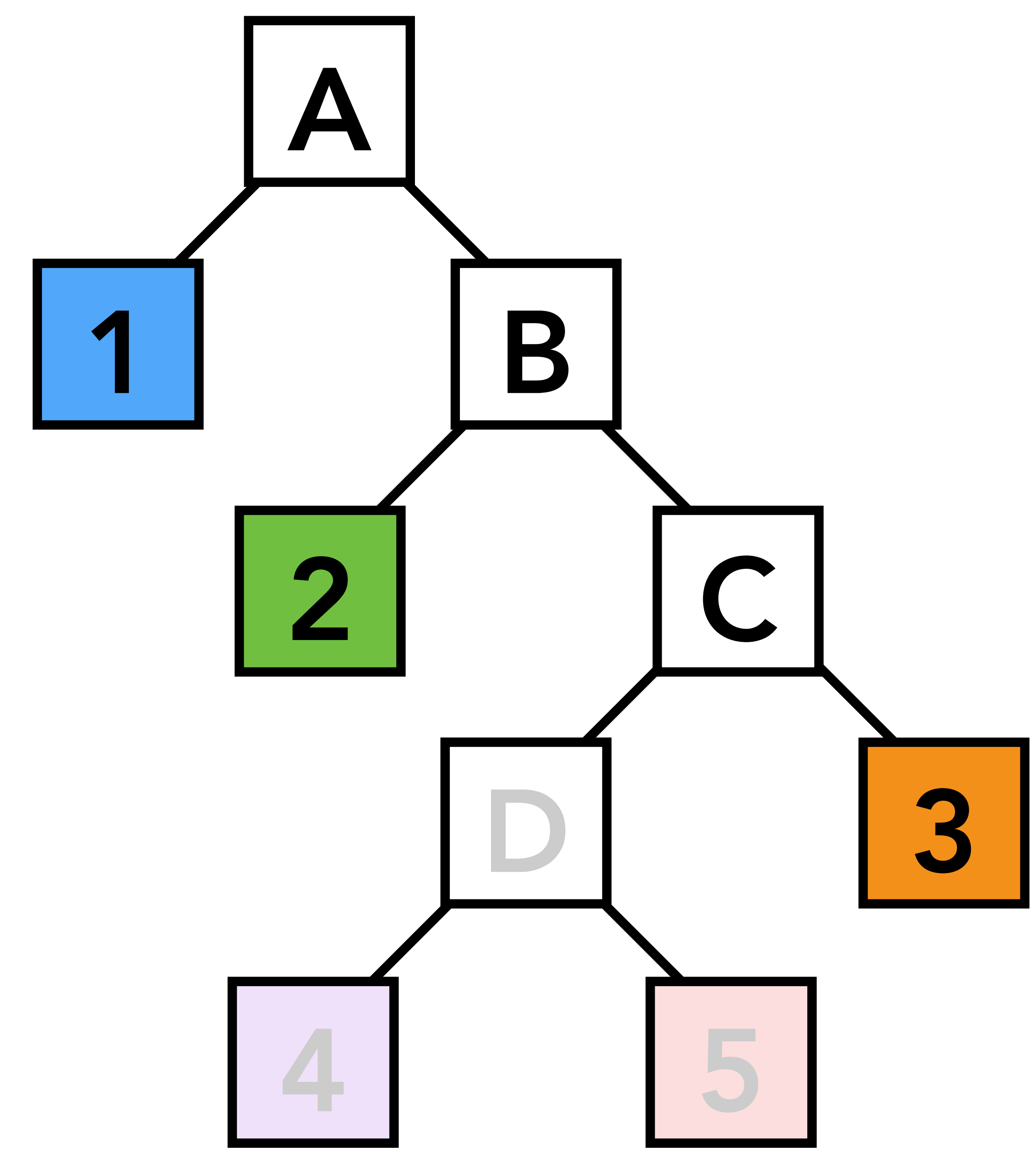




Leaf node: intersect all objects

Top-Down Recursive In-Order Traversal



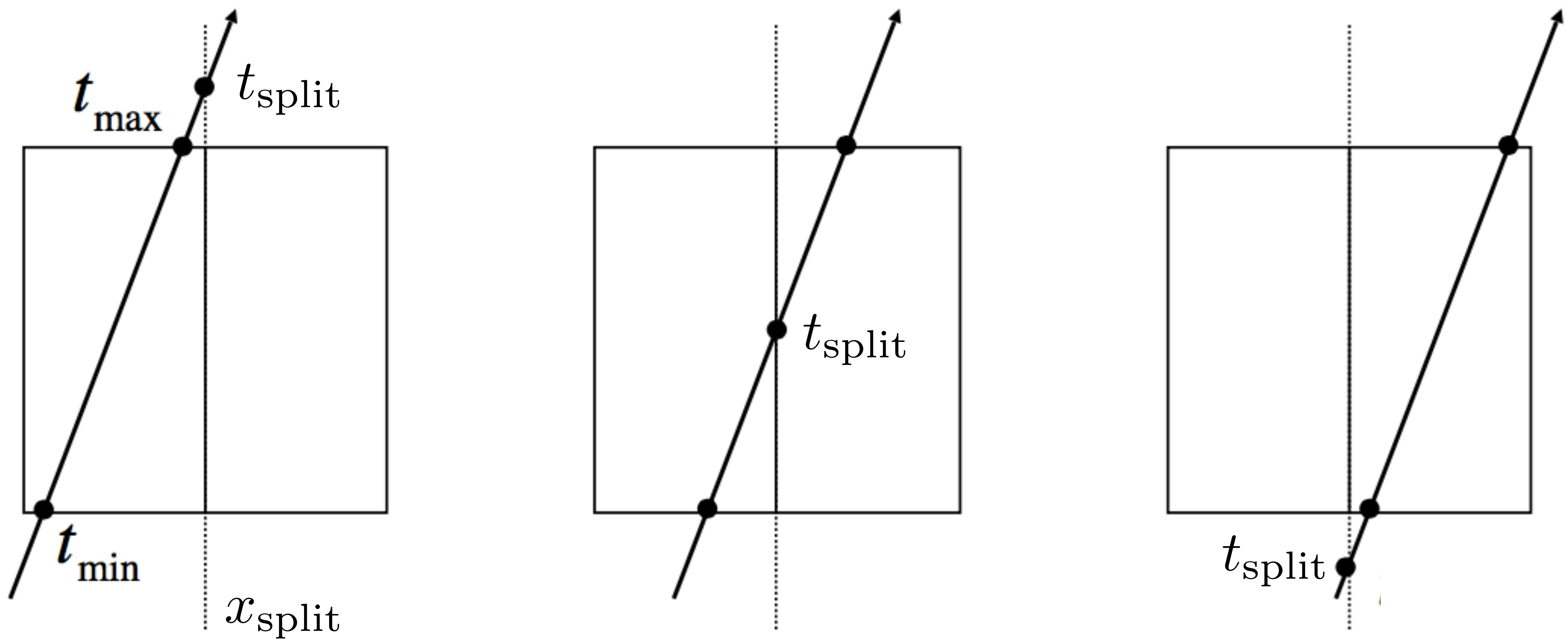


Intersection found

KD-Trees Traversal – Recursive Step

W.L.O.G. consider x-axis split with ray moving right

$$t_{\text{split}} = (x_{\text{split}} - \mathbf{o}_x)/\mathbf{d}_x$$



$t_{\text{max}} < t_{\text{split}}$ $\qquad$ $t_{\text{min}} < t_{\text{split}} < t_{\text{max}}$ $\qquad$ $t_{\text{split}} < t_{\text{min}}$

```
Intersect(L,tmin,tmax)
```

```
Intersect(L,tmin,tsplit)
Intersect(R,tsplit,tmax)
```

```
Intersect(R,tmin,tmax)
```

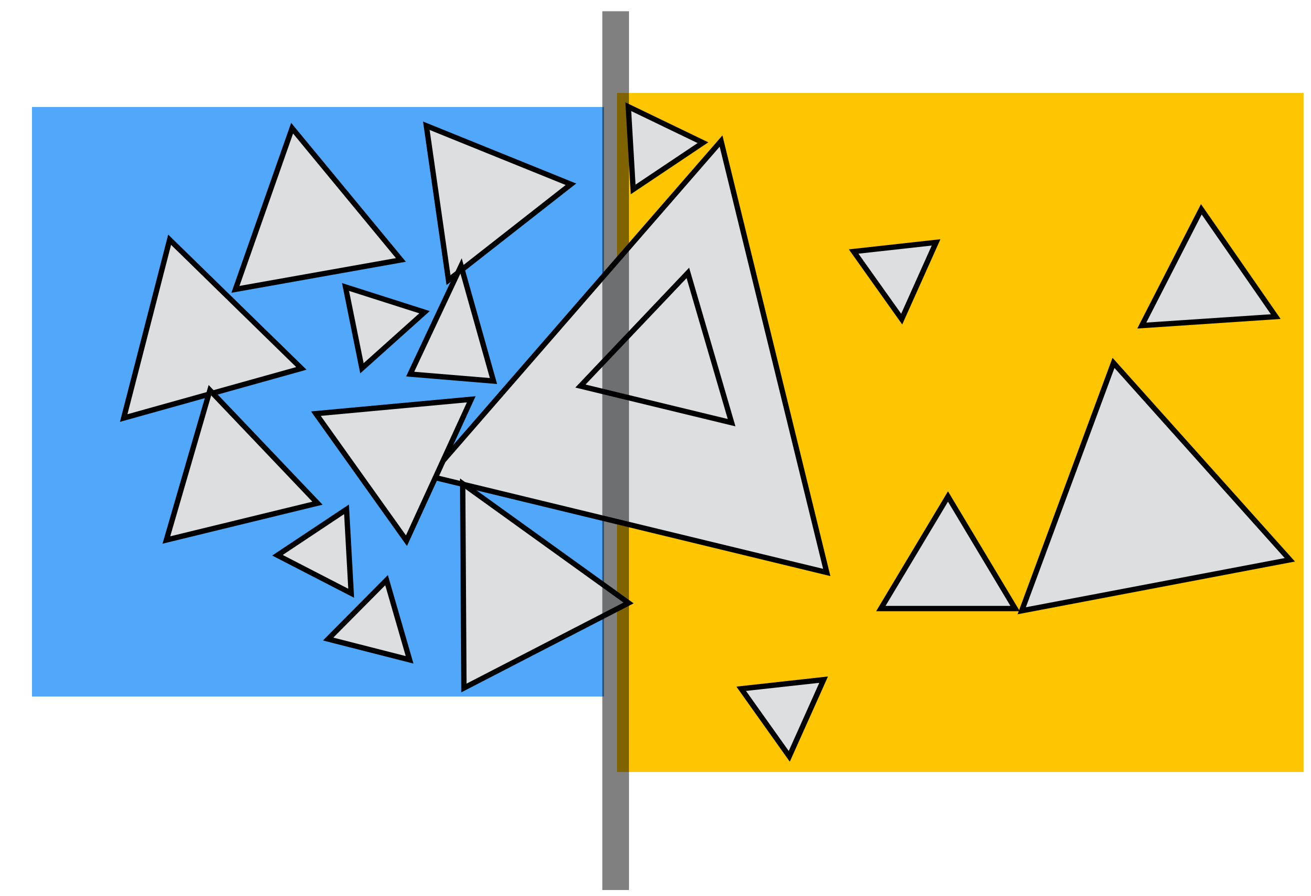
Object Partitions &

Bounding Volume Hierarchy (BVH)

Spatial vs Object Partitions

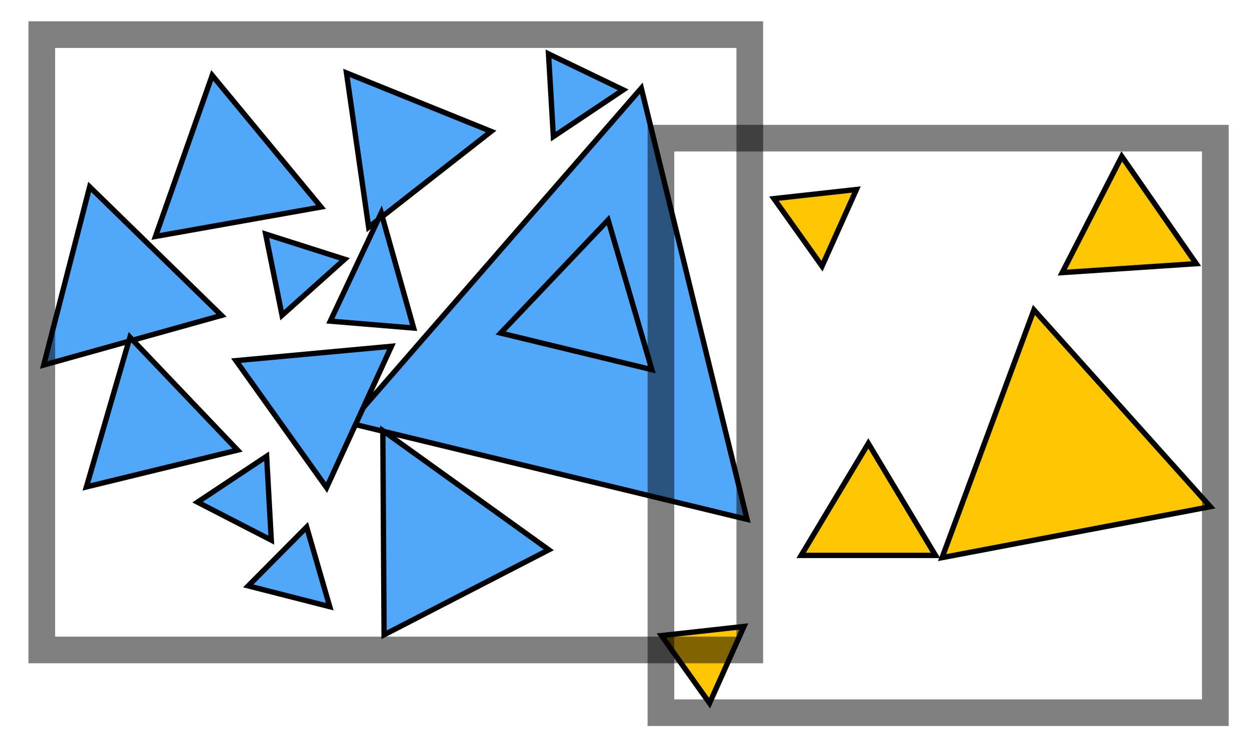
Spatial partition (e.g.KD-tree)

- Partition space into non-overlapping regions
- Objects can be contained in multiple regions

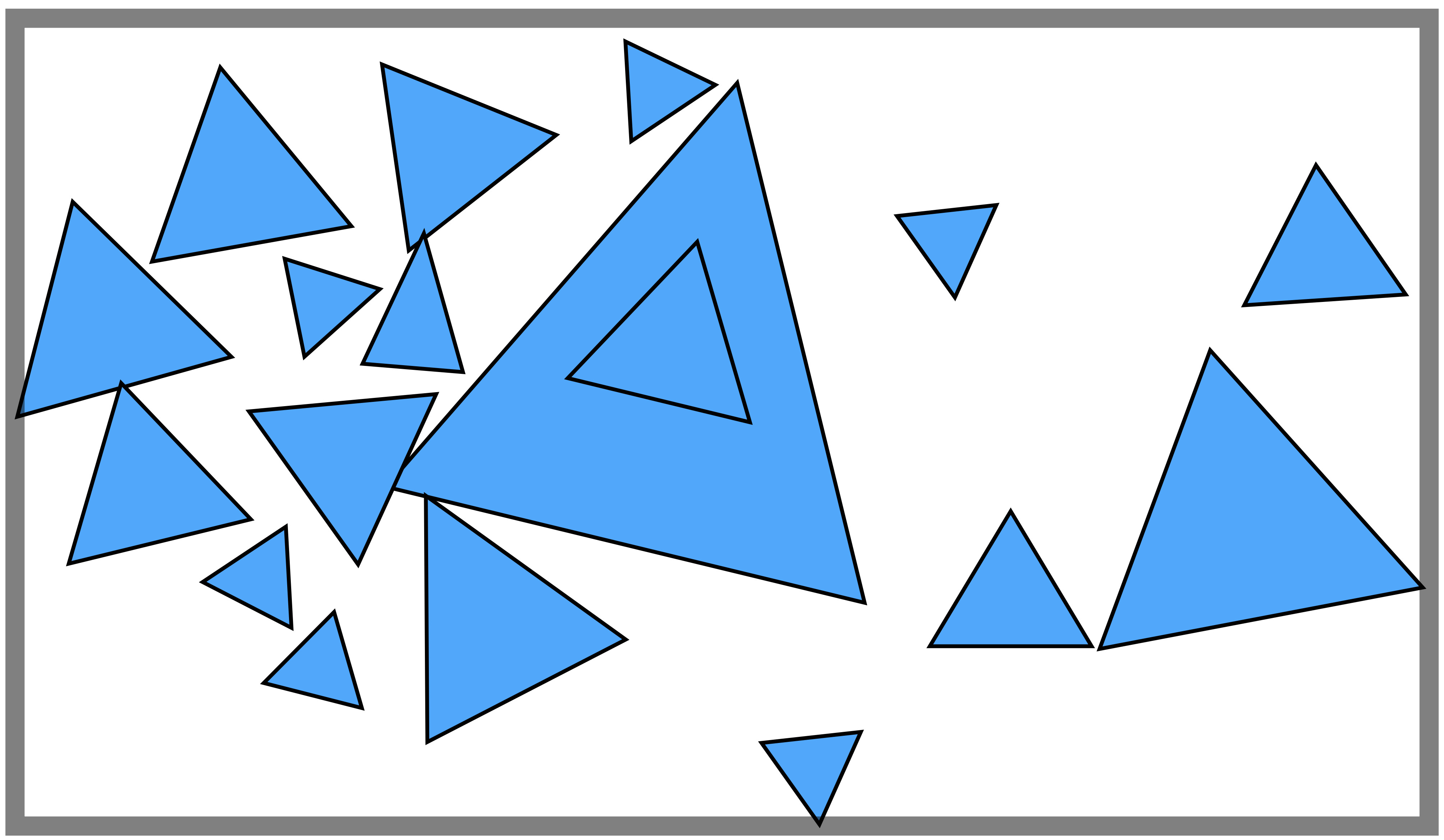


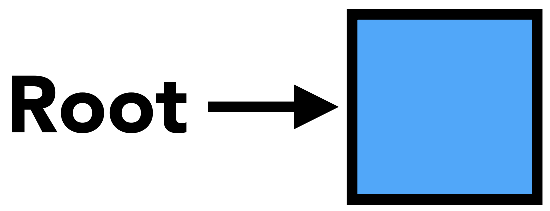
Object partition (e.g. BVH)

- Partition set of objects into disjoint subsets
- Bounding boxes for each set may overlap in space

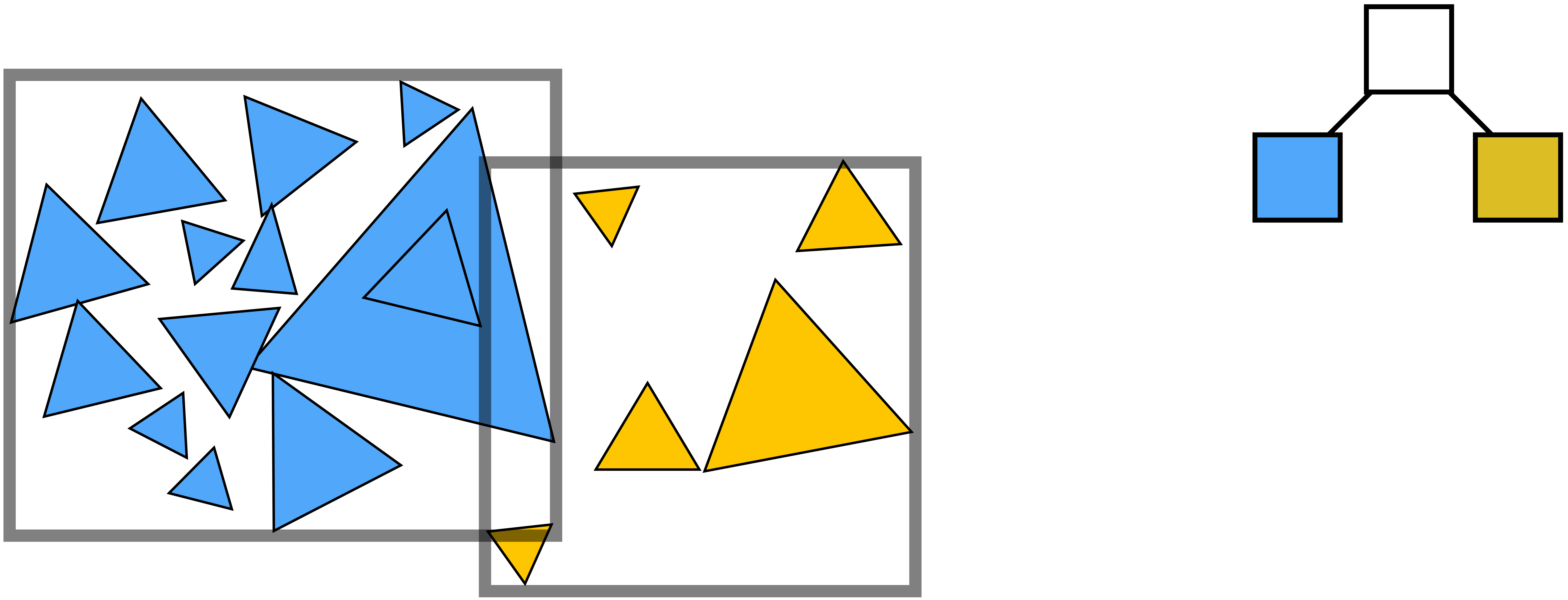


Bounding Volume Hierarchy (BVH)

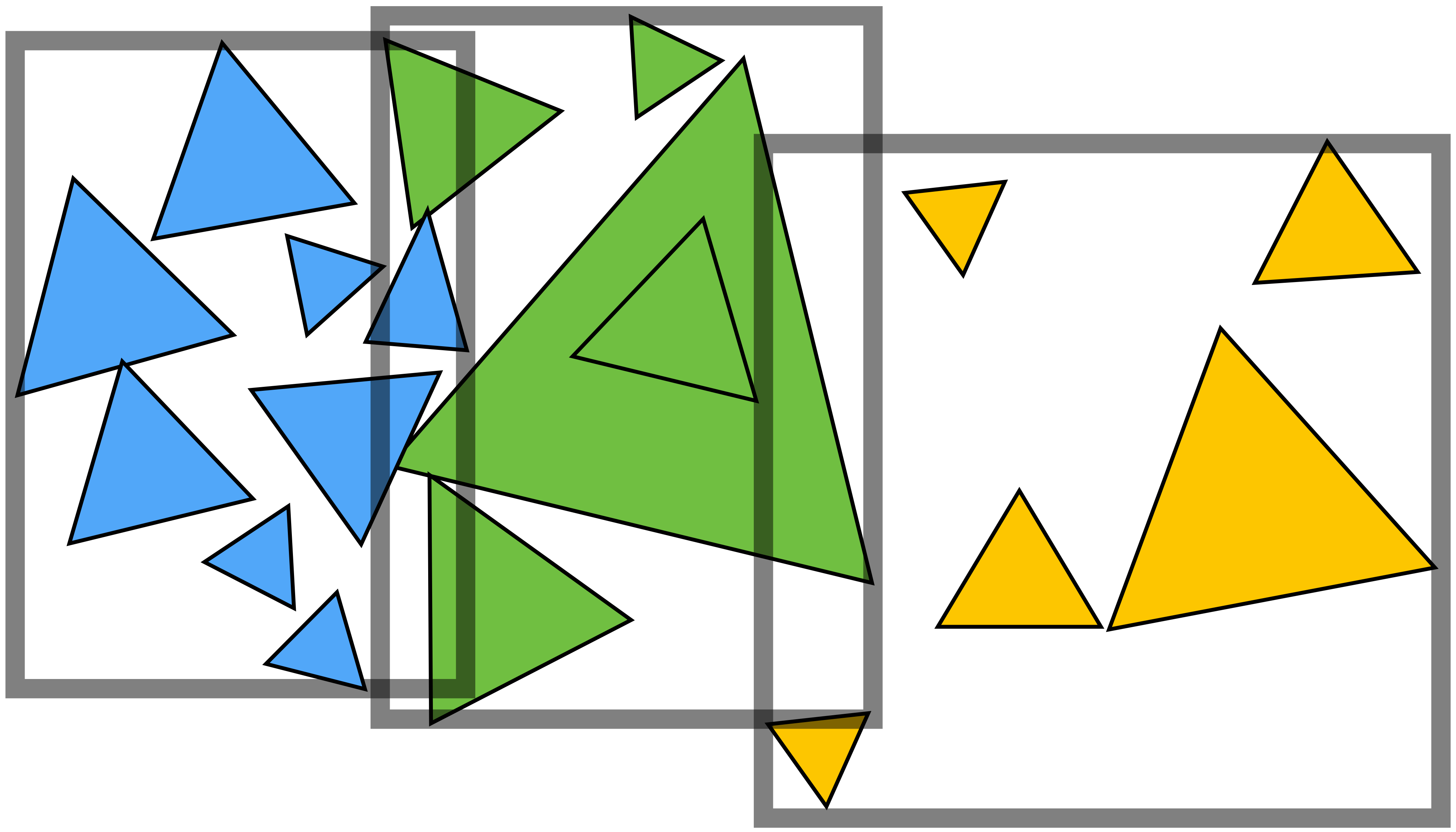


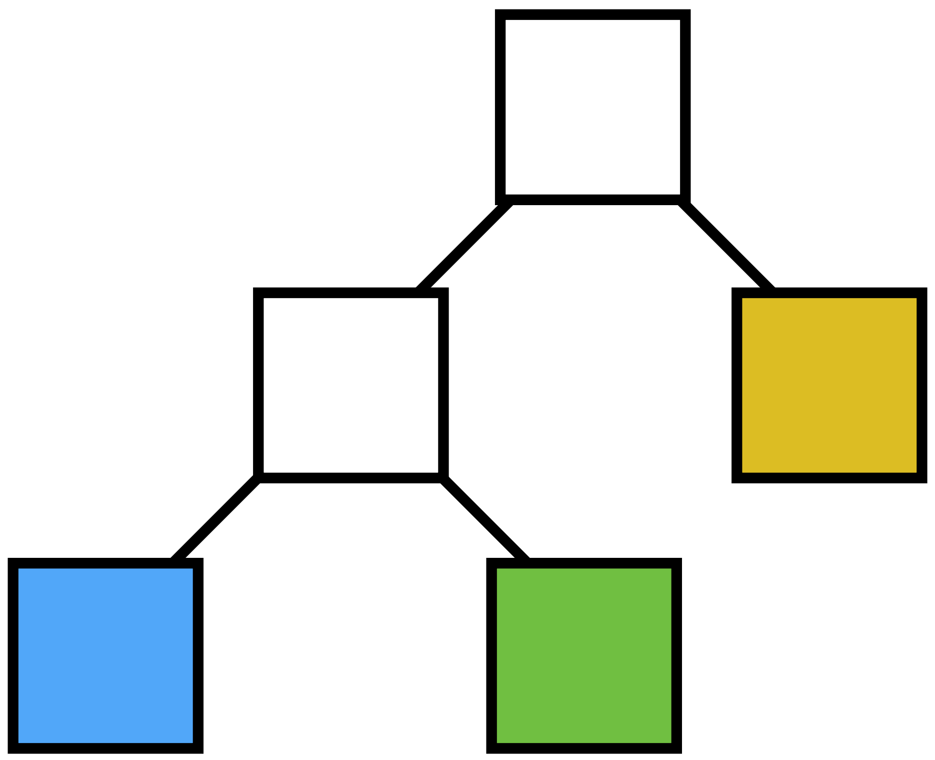


Bounding Volume Hierarchy (BVH)

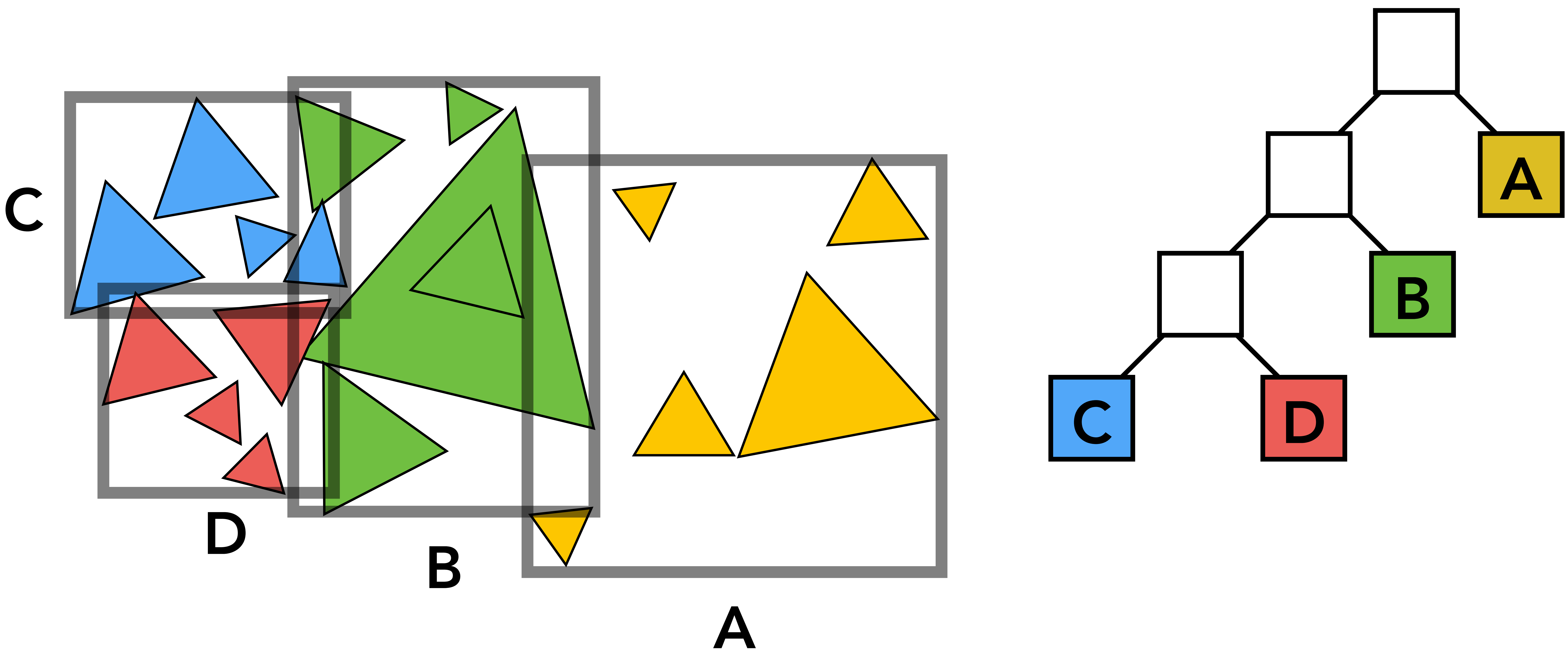


Bounding Volume Hierarchy (BVH)





Bounding Volume Hierarchy (BVH)



Bounding Volume Hierarchy (BVH)

Internal nodes store

- Bounding box
- Children: reference to child nodes

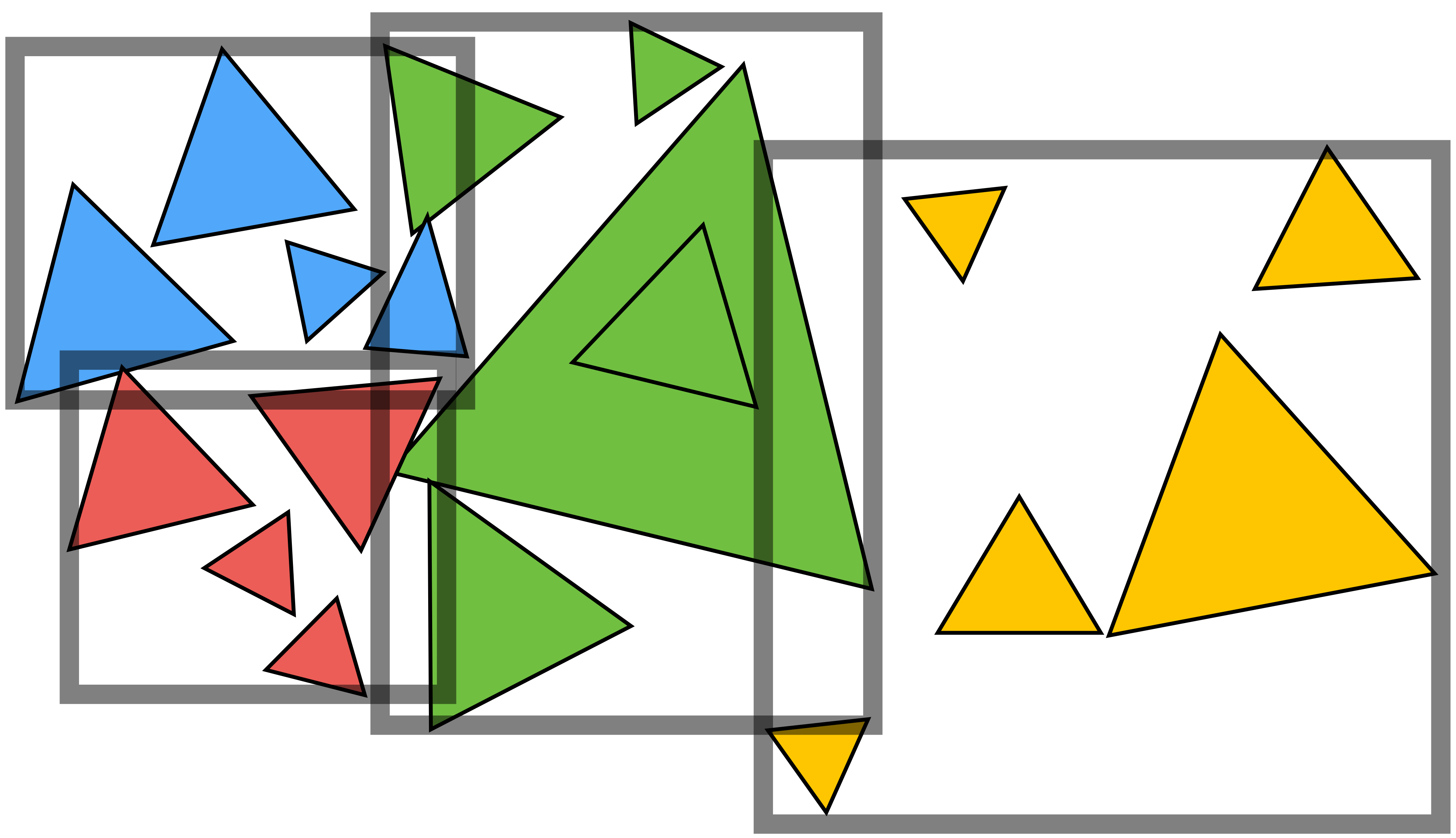
Leaf nodes store

- Bounding box
- List of objects

Nodes represent subset of primitives in scene

- All objects in subtree

BVH Pre-Processing



- Find bounding box
- Recursively split set of objects in two subsets
- Stop when there are just a few objects in each set
- Store obj reference(s) in each leaf node

BVH Pre-Processing

Choosing the set partition

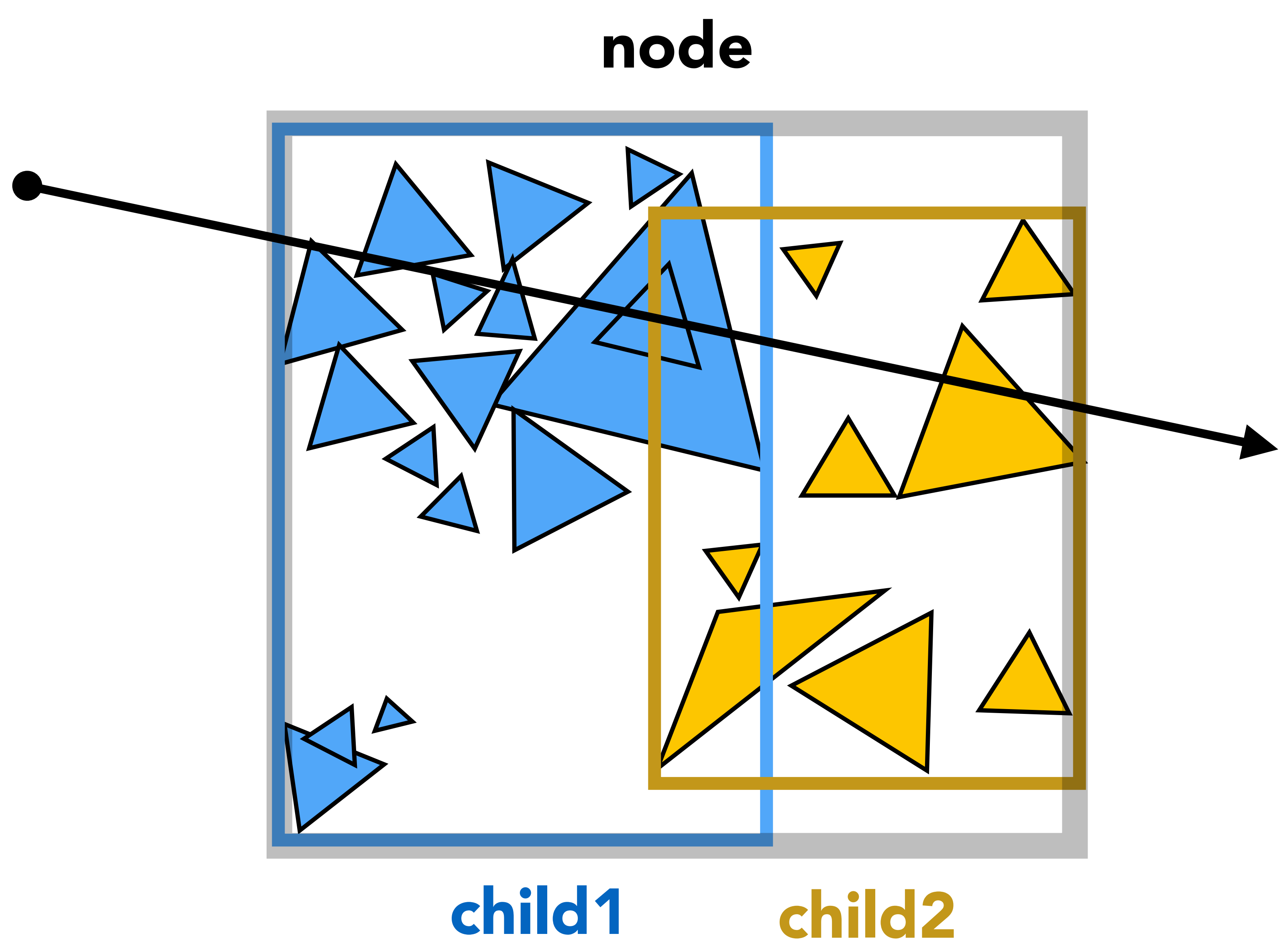
- Choose a spatial dimension to partition over (e.g. x,y,z)
- Simple #1: Split objects around spatial midpoint
- Simple #2: Split at location of median object
- Ideal: split to minimize expected cost of ray intersection

Termination criteria?

- Simple: stop when node contains few elements (e.g. 5)
- Ideal: stop when splitting does not reduce expected cost of ray intersection

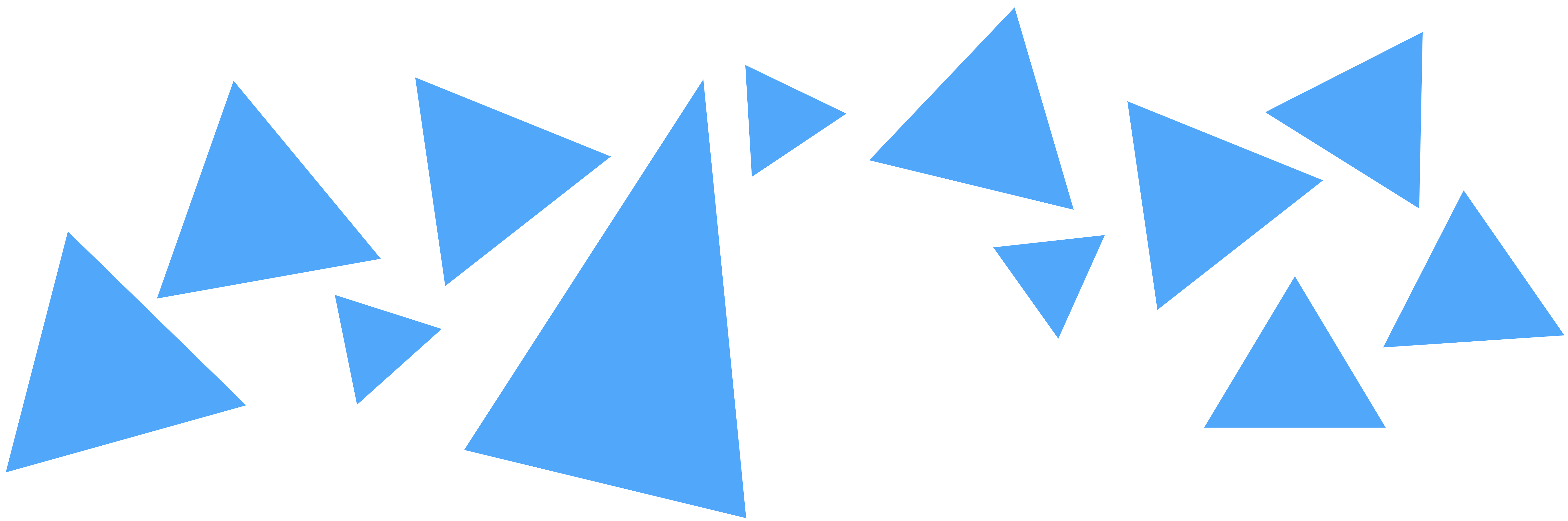
BVH Recursive Traversal

```
Intersect (Ray ray, BVH node)
  if (ray misses node.bbox) return;
  if (node is a leaf node)
     test intersection with all objs;
     return closest intersection;
  hit1 = Intersect (ray, node.child1);
  hit2 = Intersect (ray, node.child2);
  return closer of hit1, hit2;
```

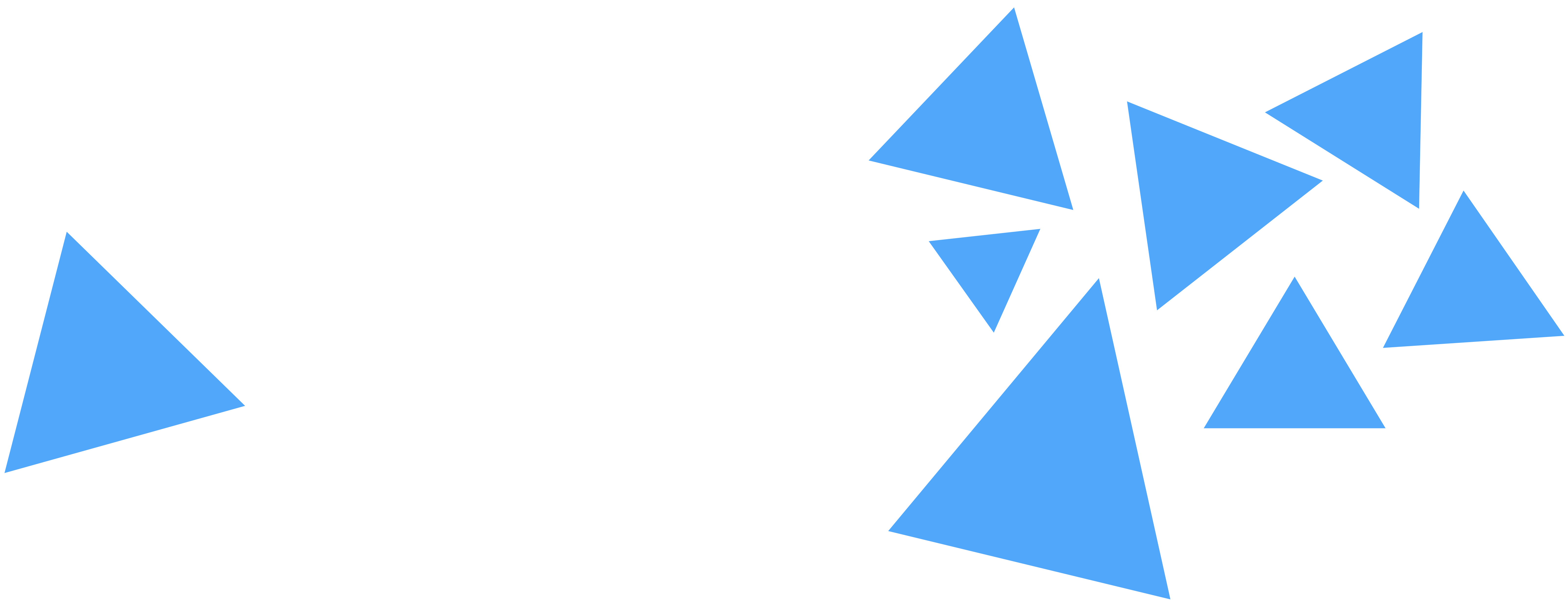


Optimizing Hierarchical Partitions (How to Split?)

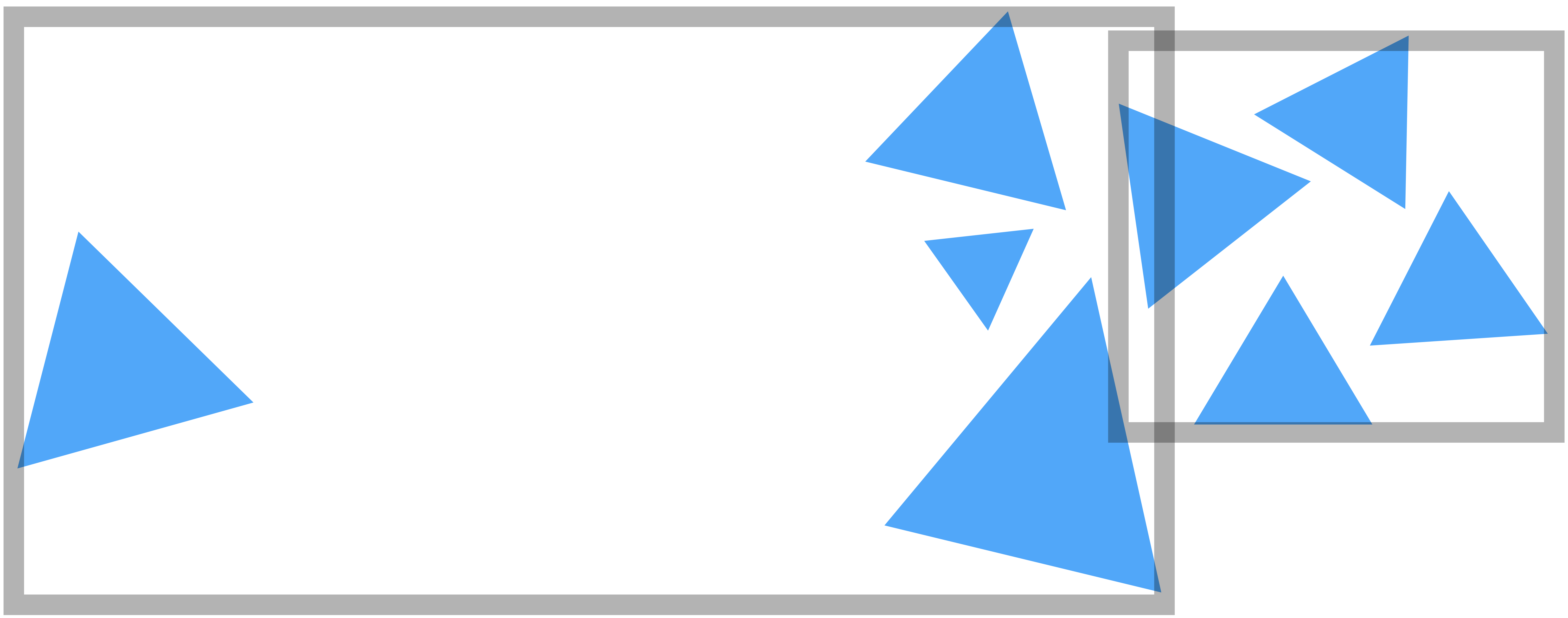
How to Split into Two Sets? (BVH)



How to Split into Two Sets? (BVH)



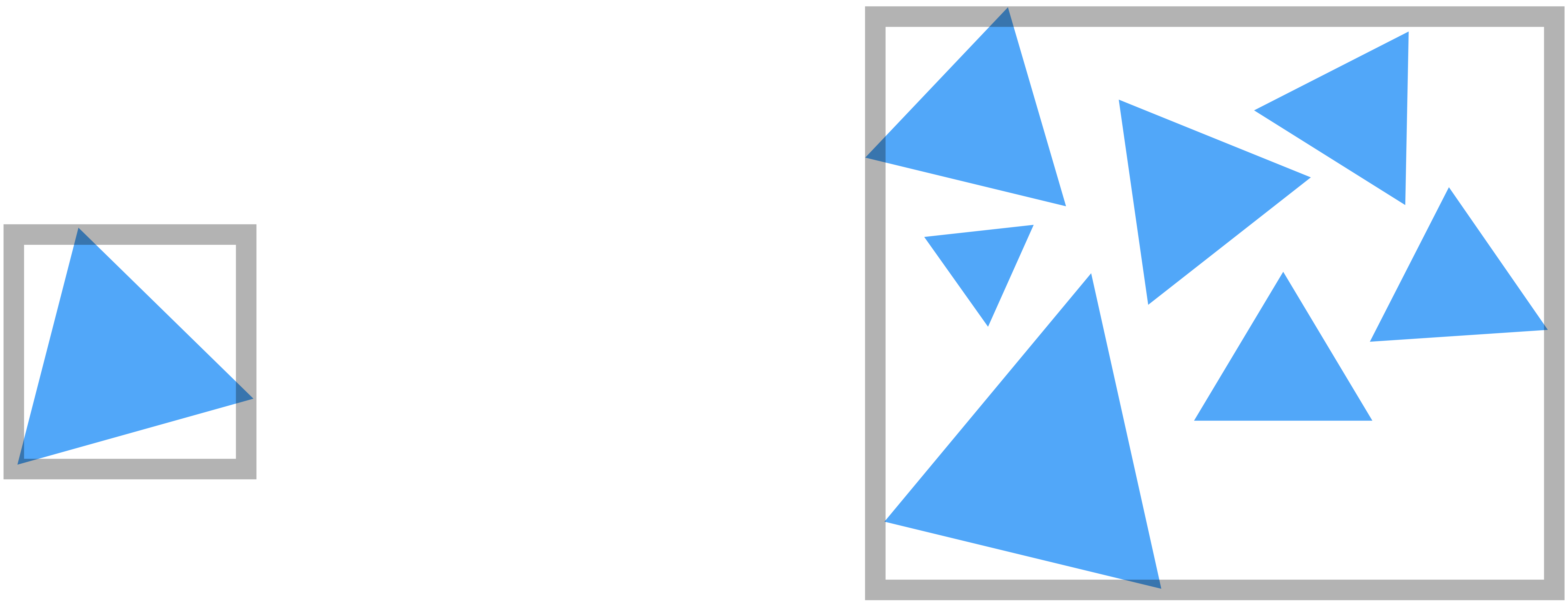
How to Split into Two Sets? (BVH)



Split at median element?

Child nodes have equal numbers of elements

How to Split into Two Sets? (BVH)



A better split?

Smaller bounding boxes, avoid overlap and empty space

Which Hierarchy Is Fastest?

Key insight: a good partition minimizes the average cost of tracing a ray

Which Hierarchy Is Fastest?

What is the average cost of tracing a ray?

For leaf node:

```
Cost(node) = cost of intersecting all triangles
           = C_isect * TriCount(node)

  C_isect        = cost of intersecting a triangle
  TriCount(node) = number of triangles in node
```

Which Hierarchy Is Fastest?

What is the average cost of tracing a ray?

For internal node:

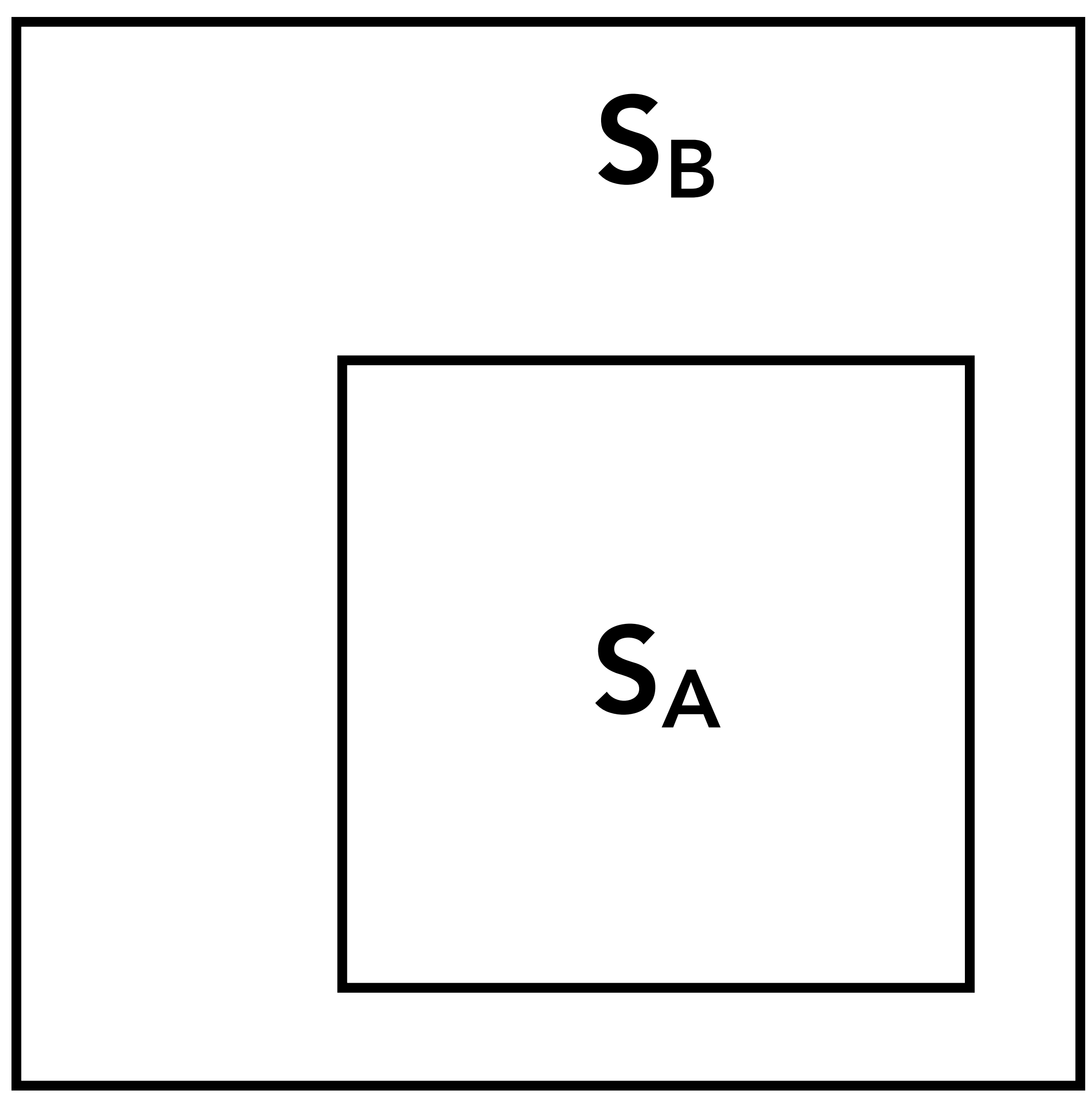
```
Cost(node) = C_trav
                + Prob(hit L)*Cost(L)
                + Prob(hit R)*Cost(R)
```

```
C_trav  = cost of traversing a cell
Cost(L) = cost of traversing left child
Cost(R) = cost of traversing right child
```

Optimizing Hierarchical Partitions Example: Surface Area Heuristic Algorithm

Ray Intersection Probability

The probability of a random ray hitting a convex shape A enclosed by another convex shape B is the ratio of their surface areas, S_A / S_B.



$$P(\text{hit}A|\text{hit}B) = \frac{S_A}{S_B}$$

Estimating Cost with Surface Area Heuristic (SAH)

Probabilities of ray intersecting a node

- If assume uniform ray distribution, no occlusions, then probability is proportional to node's surface area

Cost of processing a node

- Common approximation is #triangles in node's subtree

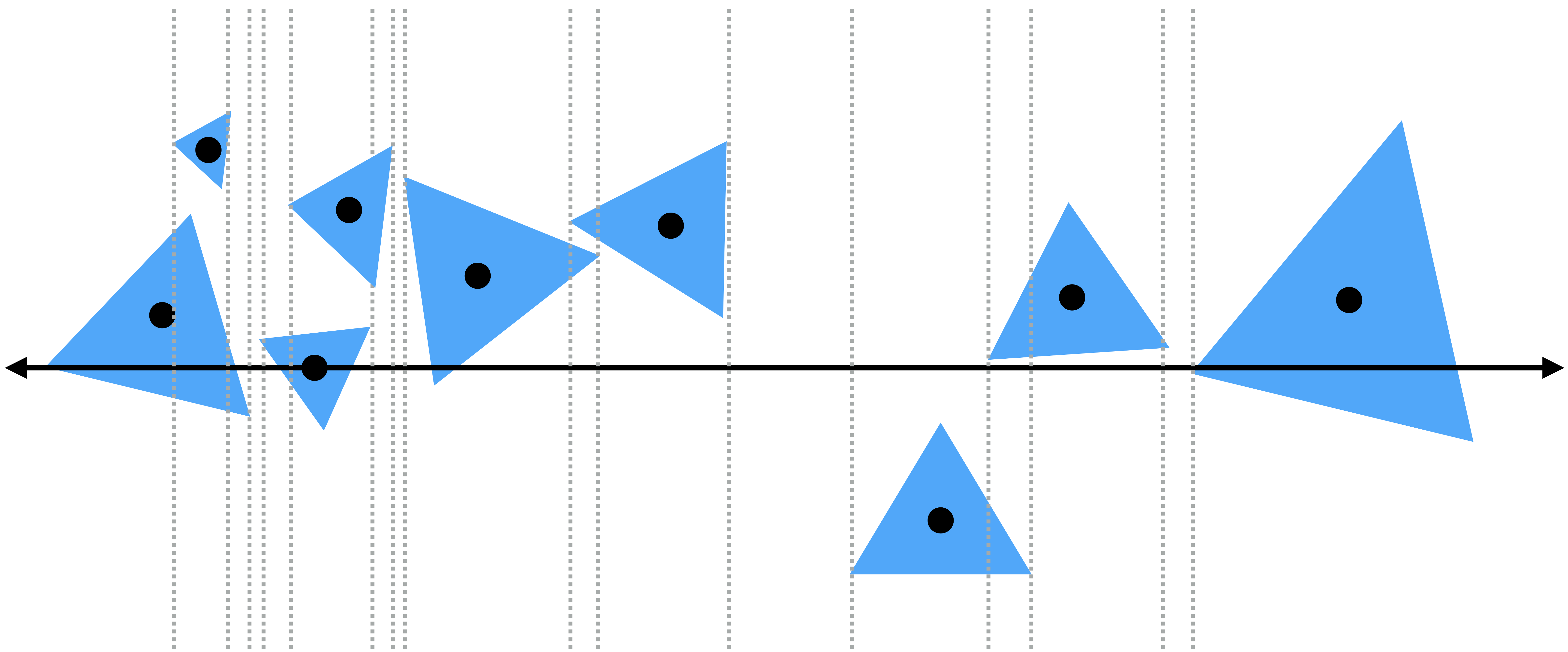
```
Cost(cell) = C_trav + SA(L)*TriCount(L) + SA(R)*TriCount(R)

SA(node) = surface area of bbox of node
C_trav = ratio of cost to traverse vs. cost to intersect tri
       C_trav = 1:8 in PBRT [Pharr & Humphreys]
       C_trav = 1:1.5 in a highly optimized version
```

Partition Implementation

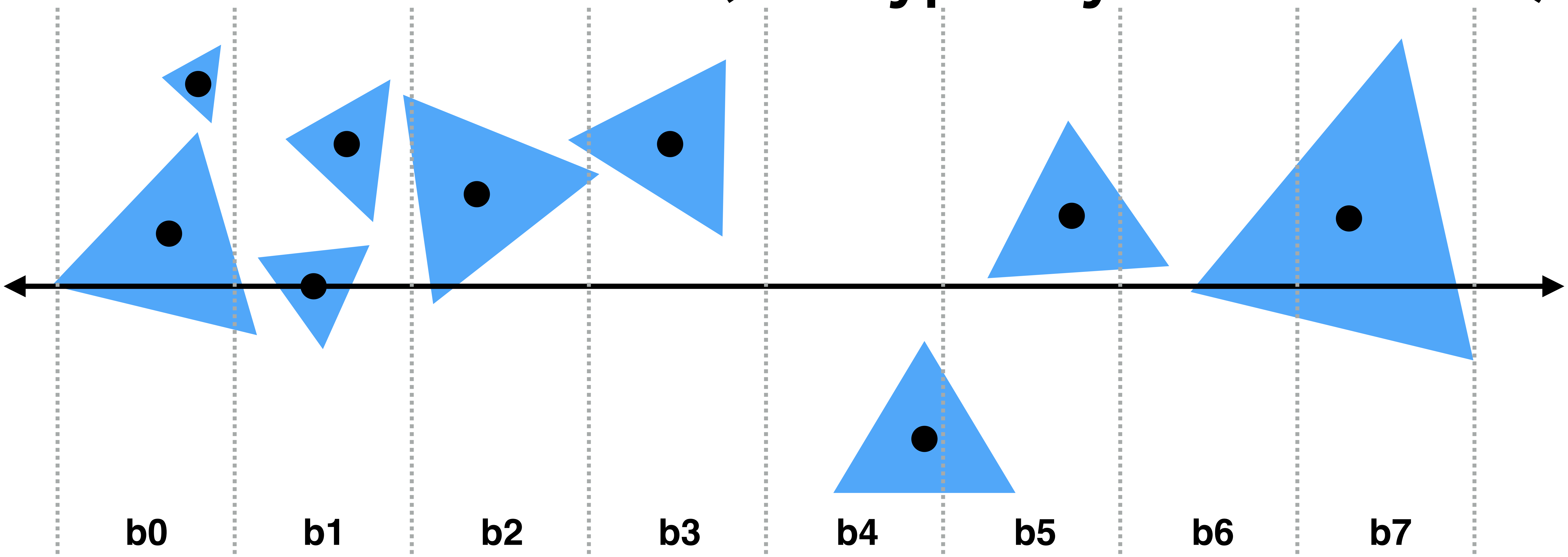
Constrain search to axis-aligned spatial partitions

- Choose an axis
- Choose a split plane on that axis
- Partition objects into two halves by centroid
- 2N–2 candidate split planes for node with N primitives. (Why?)



Partition Implementation (Efficient)

Efficient modern approximation: split spatial extent of primitives into B buckets (B is typically small: B < 32)




```
For each axis: x,y,z:
  initialize buckets
  For each object p in node:
    b = compute_bucket(p.centroid)
    b.bbox.union(p.bbox);
    b.prim_count++;
  For each of the B–1 possible partitioning planes evaluate SAH
Execute lowest cost partitioning found (or make node a leaf)
```

Cost-Optimization Applies to Spatial Partitions Too

- Discussed optimization of BVH construction
- But principles are general and apply to spatial partitions as well
- E.g. to optimize KD-Tree construction
  - Goal is to minimize average cost of intersecting ray with tree
  - Can still apply Surface Area Heuristic
  - Note that surface areas and number of nodes in children differ from BVH

Things to Remember

Linear vs logarithmic ray-intersection techniques

Many techniques for accelerating ray-intersection

- Spatial partitions: Grids and KD-Trees
- Object partitions: Bounding Volume Hierarchies

Optimize hierarchy construction based on minimizing cost of intersecting ray against hierarchy

- Leads to Surface Area Heuristic for best partition

Acknowledgments

Thanks to Pat Hanrahan, Kayvon Fatahalian, Mark Pauly and Steve Marschner for lecture resources.