Ray Tracing II
(Acceleration Structures)

Last Lecture

- Why ray tracing?
- Whitted-style ray tracing
- Ray-object intersections
  - Implicit surfaces
  - Triangles

Today

- Axis-Aligned Bounding Boxes (AABBs)
    - Understanding — pairs of slabs
    - Ray-AABB intersection
- Using AABBs to accelerate ray tracing
    - Uniform grids
    - Spatial partitions

Accelerating Ray-Surface Intersection

Ray Tracing – Performance Challenges

Simple ray-scene intersection

- Exhaustively test ray-intersection with **every triangle**
- Find the closest hit (i.e. minimum t)

Problem:

- Naive algorithm = #pixels × # traingles (× #bounces)
- Very slow!

Ray Tracing – Performance Challenges

Jun Yan, Tracy Renderer

San Miguel Scene, 10.7M triangles

Ray Tracing – Performance Challenges

Plant Ecosystem, 20M triangles

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

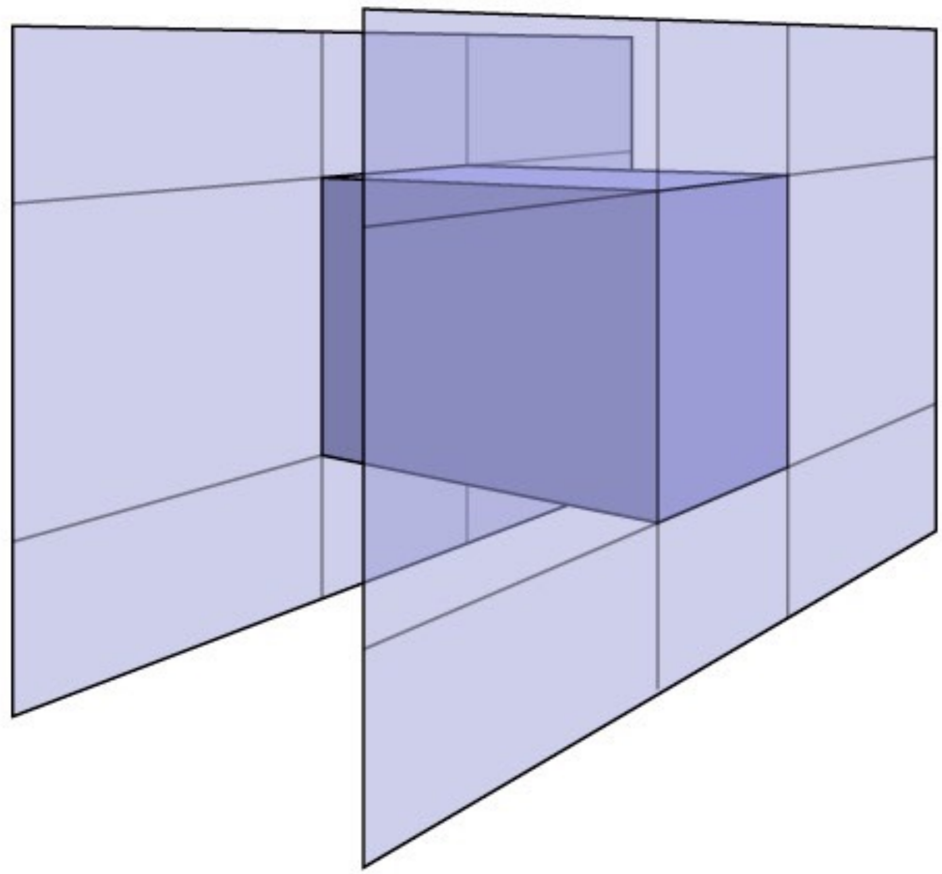
Ray-Intersection With Box

Understanding: **box is the intersection of 3 pairs of slabs**

Specifically:

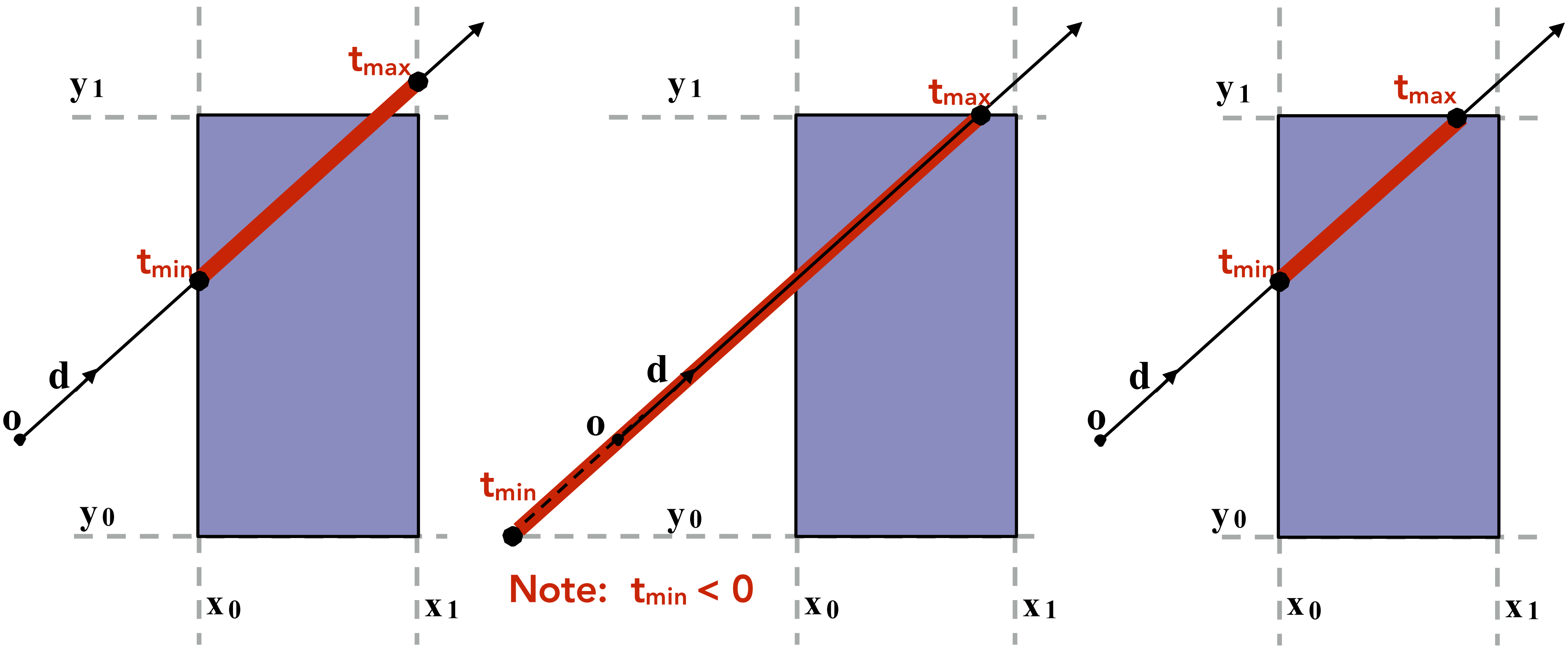
We often use an **Axis-Aligned Bounding Box (AABB)**

i.e. any side of the BB is along either x, y, or z axis



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



Intersections with *x* planes **Intersections with *y* planes** **Final intersection result**

How do we know when the ray intersects the box?

Ray Intersection with Axis-Aligned Box

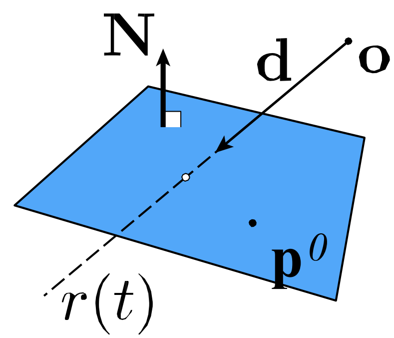
- Recall: a box (3D) = three pairs of infinitely large slabs
- Key ideas
  - The ray enters the box **only when** it enters all pairs of slabs
  - The ray exits the box **as long as** it exits any pair of slabs
- For each pair, calculate the t_{min} and t_{max} (negative is fine)
- For the 3D box, $t_{enter} = \textbf{max}\{t_{min}\}$, $t_{exit} = \textbf{min}\{t_{max}\}$
- If $t_{enter} < t_{exit}$, we know the ray **stays a while** in the box (so they must intersect!) (not done yet, see the next slide)

Ray Intersection with Axis-Aligned Box

- However, ray is not a line
  - Should check whether t is negative for physical correctness!
- What if $t_{exit} < 0$?
  - The box is “behind” the ray — no intersection!
- What if $t_{exit} >= 0$ and $t_{enter} < 0$?
  - The ray’s origin is inside the box — have intersection!
- In summary, ray and AABB intersect iff
  - $t_{enter} < t_{exit}$ && $t_{exit} >= 0$

Why Axis-Aligned?

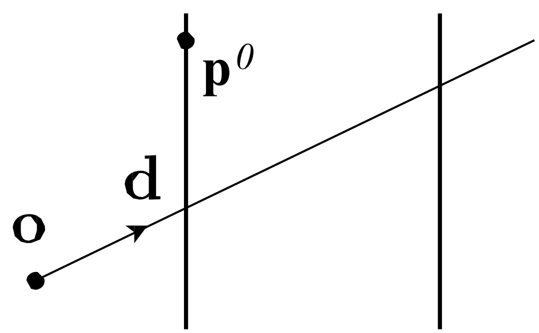
General



$$t = \frac{(\mathbf{p}^{0} - \ \mathbf{o}) \cdot \mathbf{N}}{\mathbf{d} \quad \mathbf{N}}$$

3 subtractions, 6 multiplies, 1 division

Slabs perpendicular to x-axis

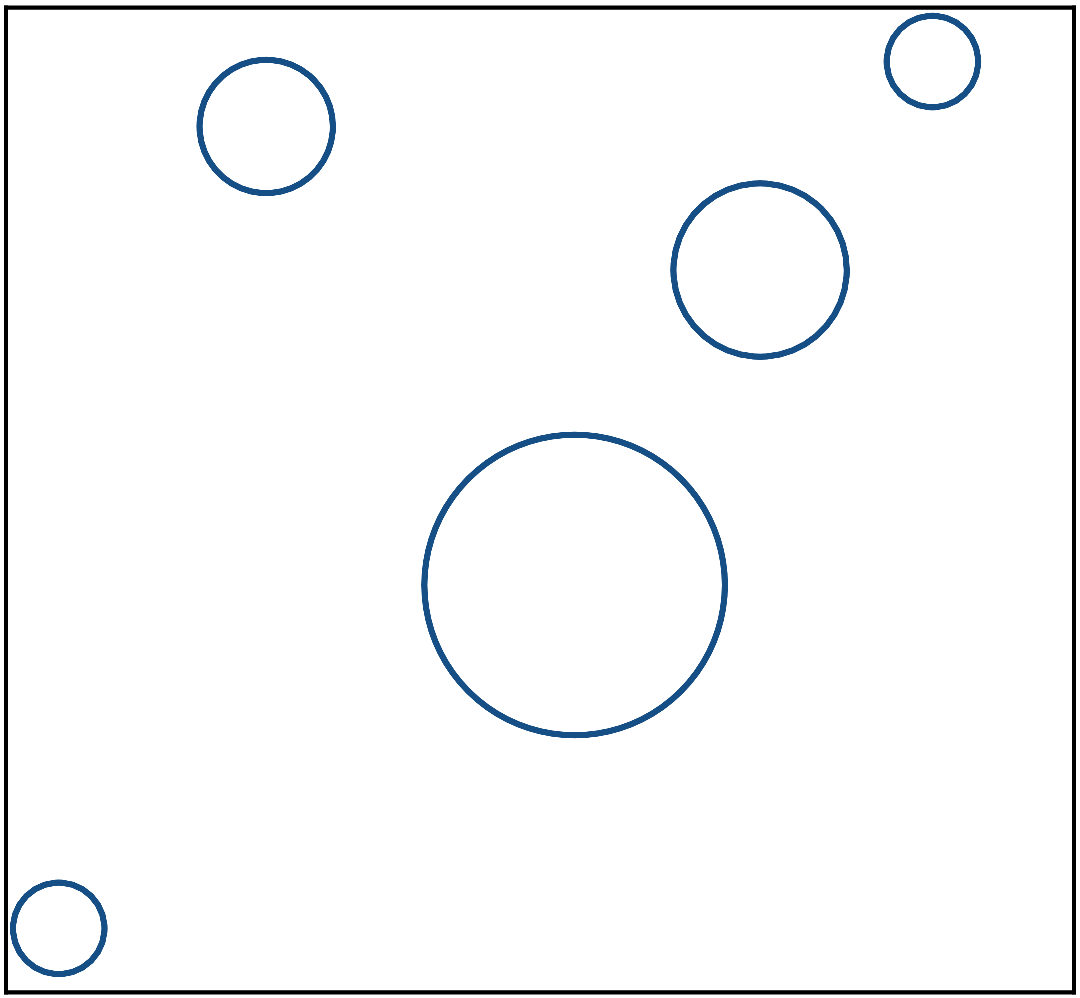


$$t = \frac{\mathbf{p}'_x - \mathbf{o}_x}{\mathbf{d}_x}$$

1 subtraction, 1 division

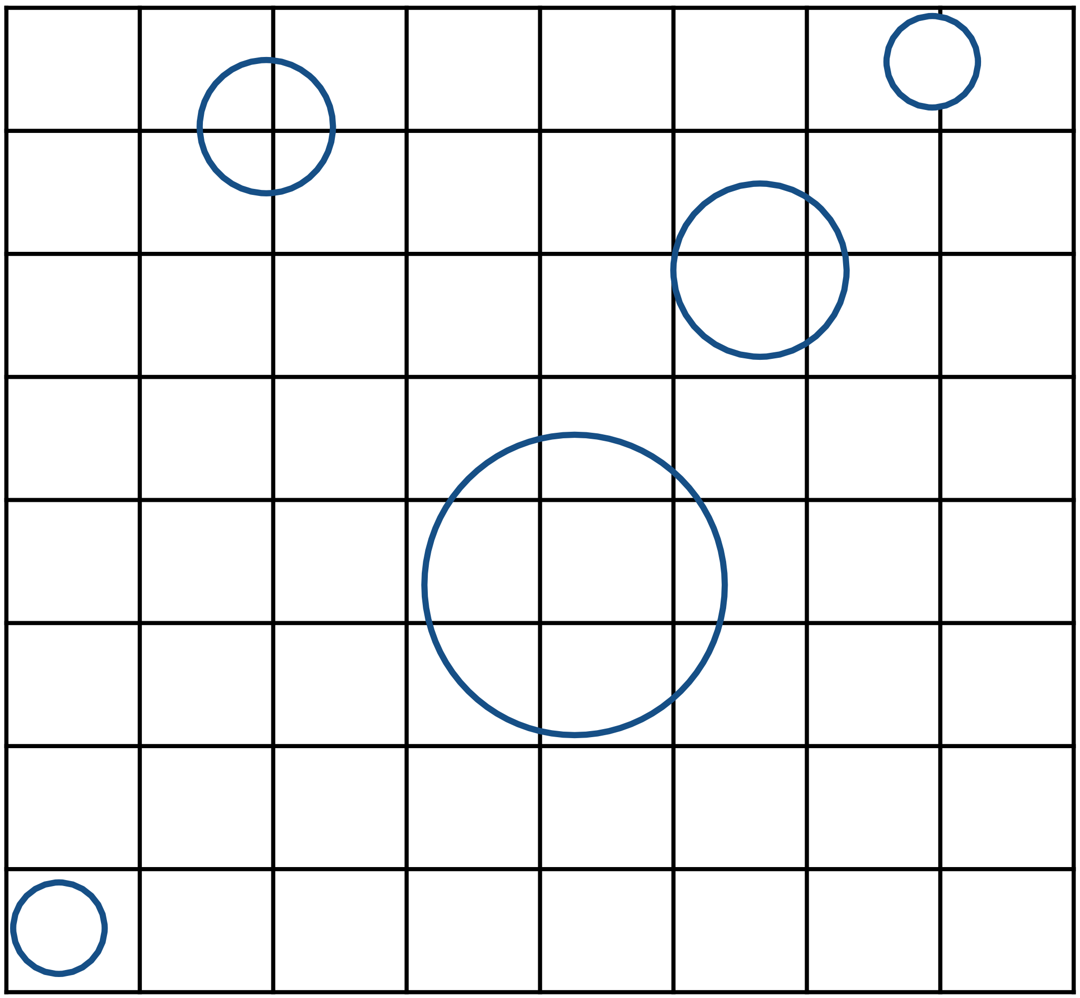
Uniform Spatial Partitions (Grids)

Preprocess – Build Acceleration Grid



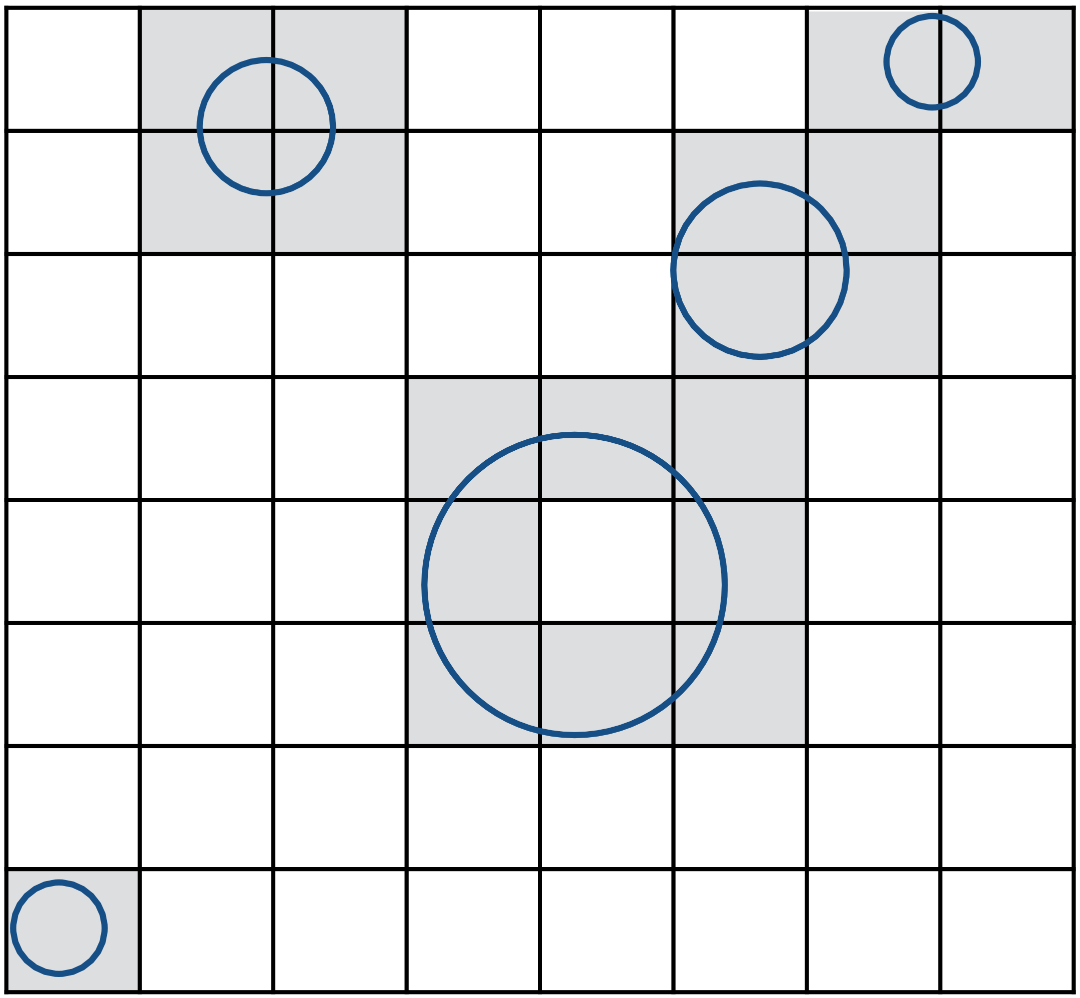
1. Find bounding box

Preprocess – Build Acceleration Grid



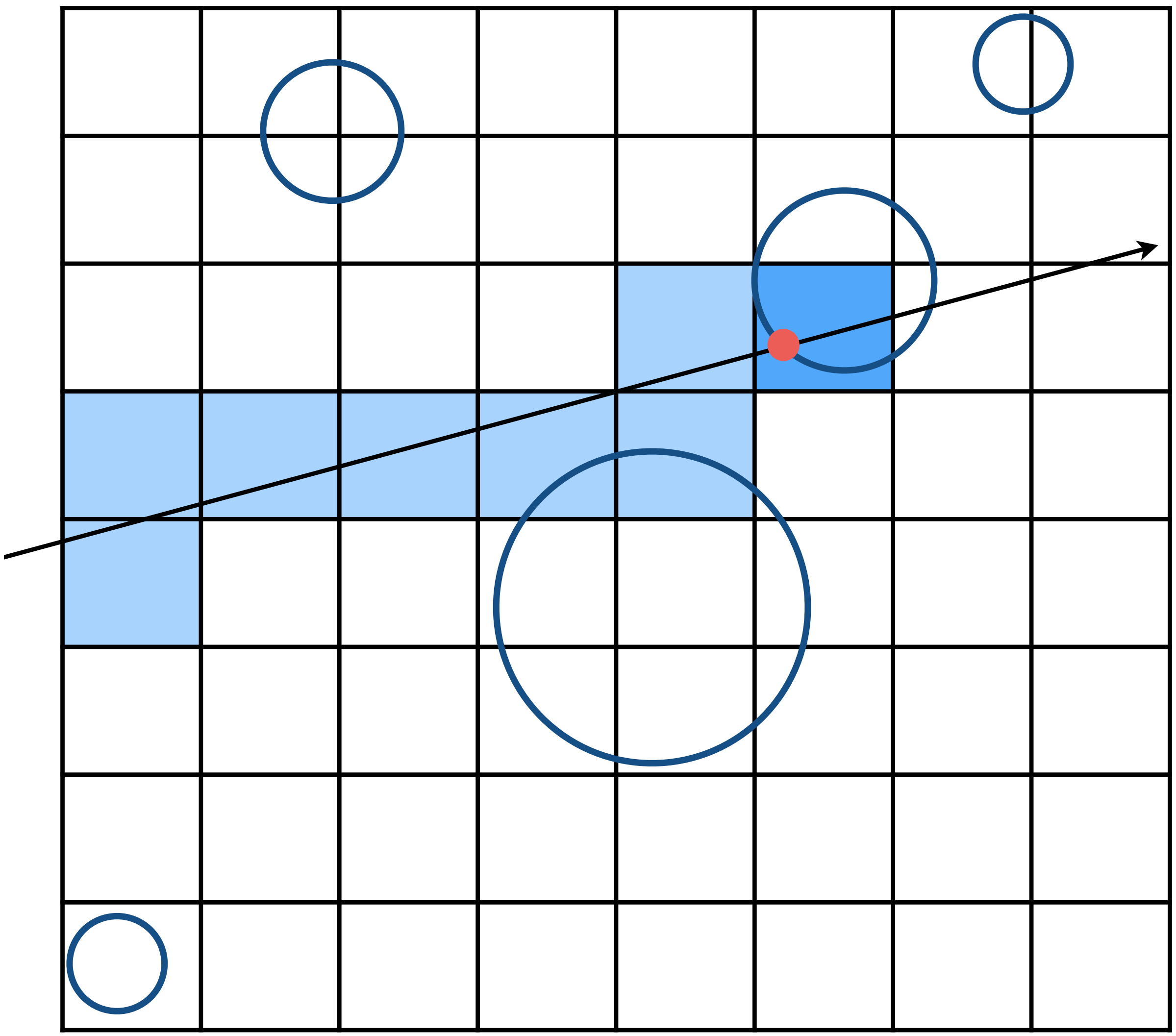
1. Find bounding box
2. Create grid

Preprocess – Build Acceleration Grid



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

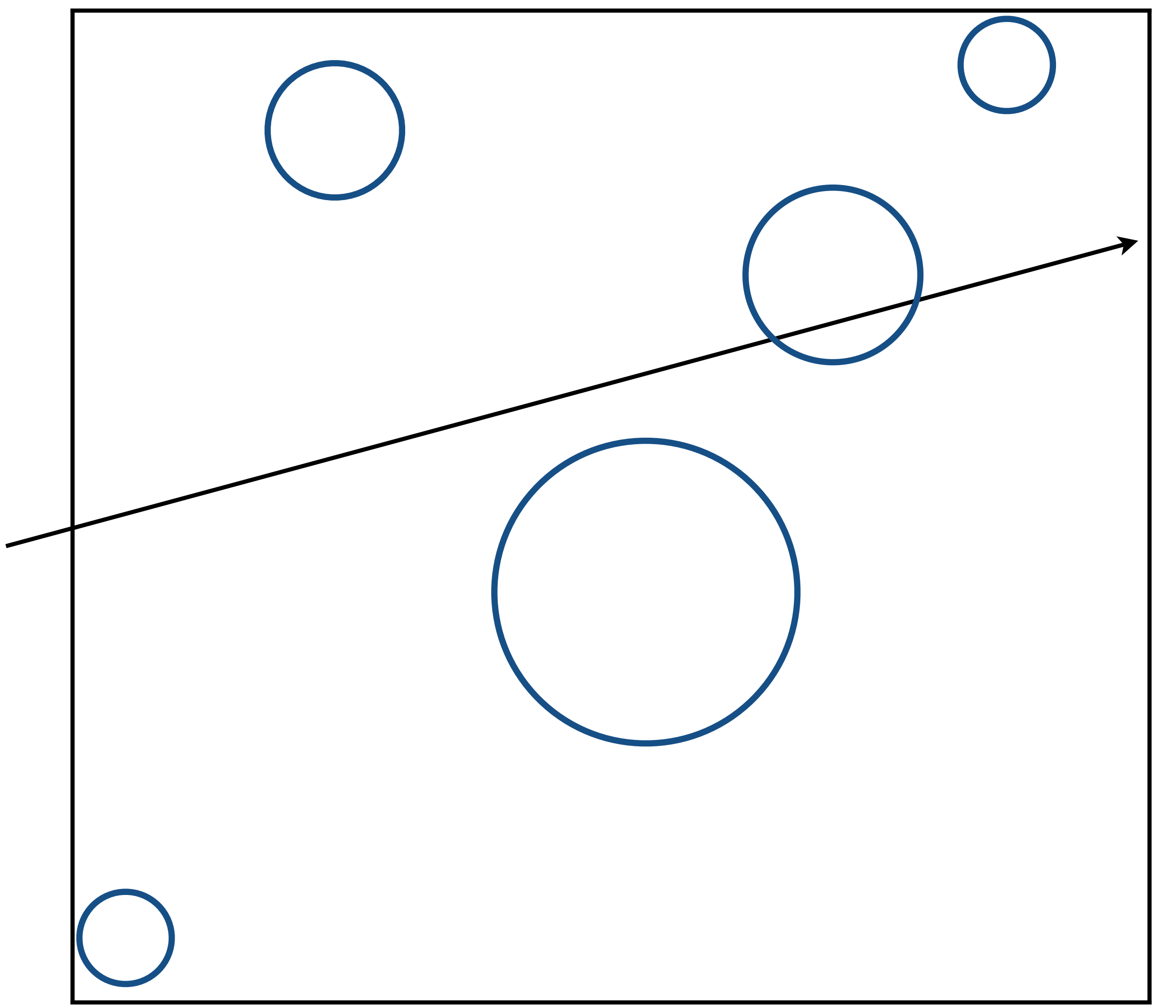
Ray-Scene Intersection



Step through grid in ray traversal order

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

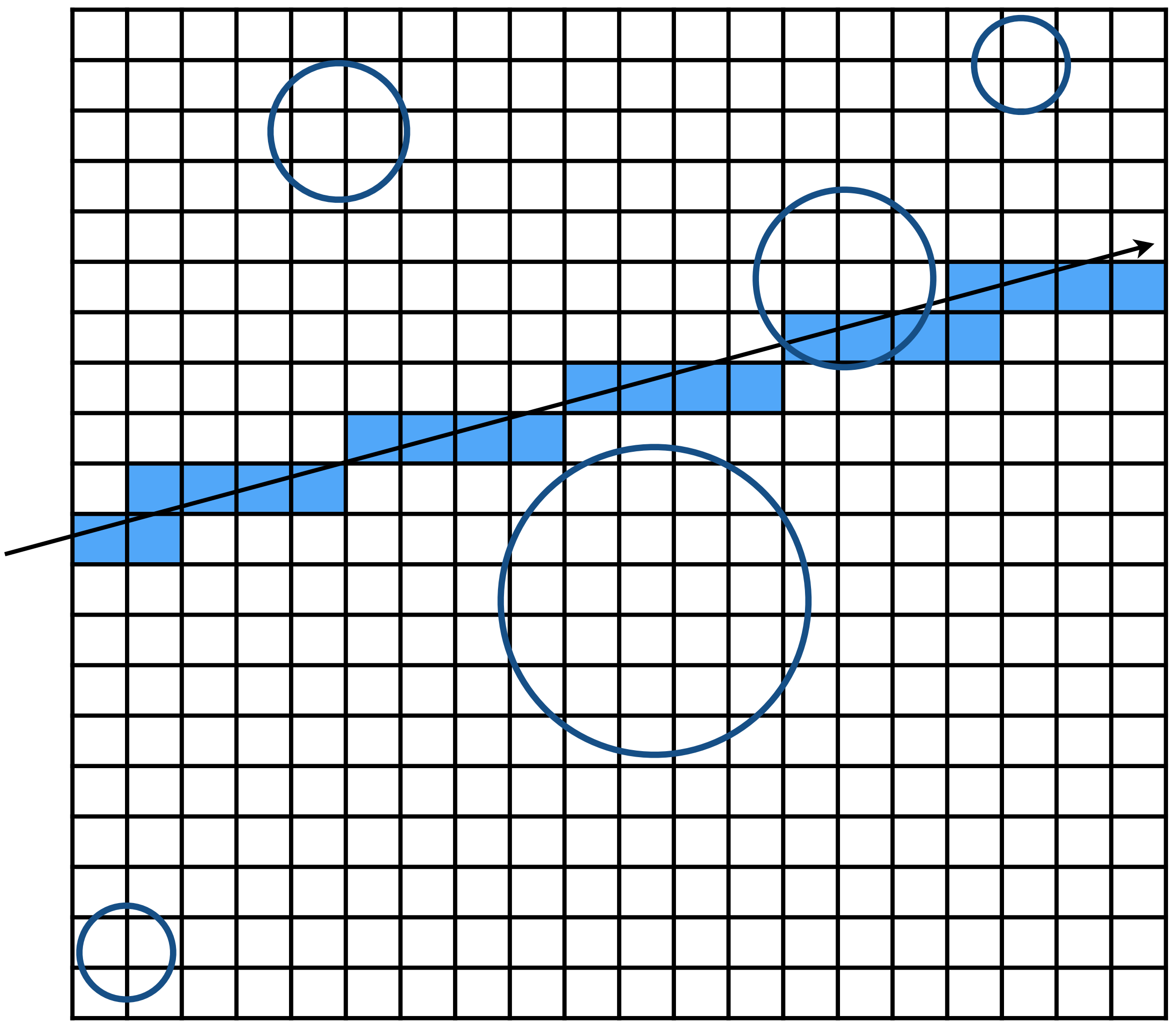
Grid Resolution?



One cell

- No speedup

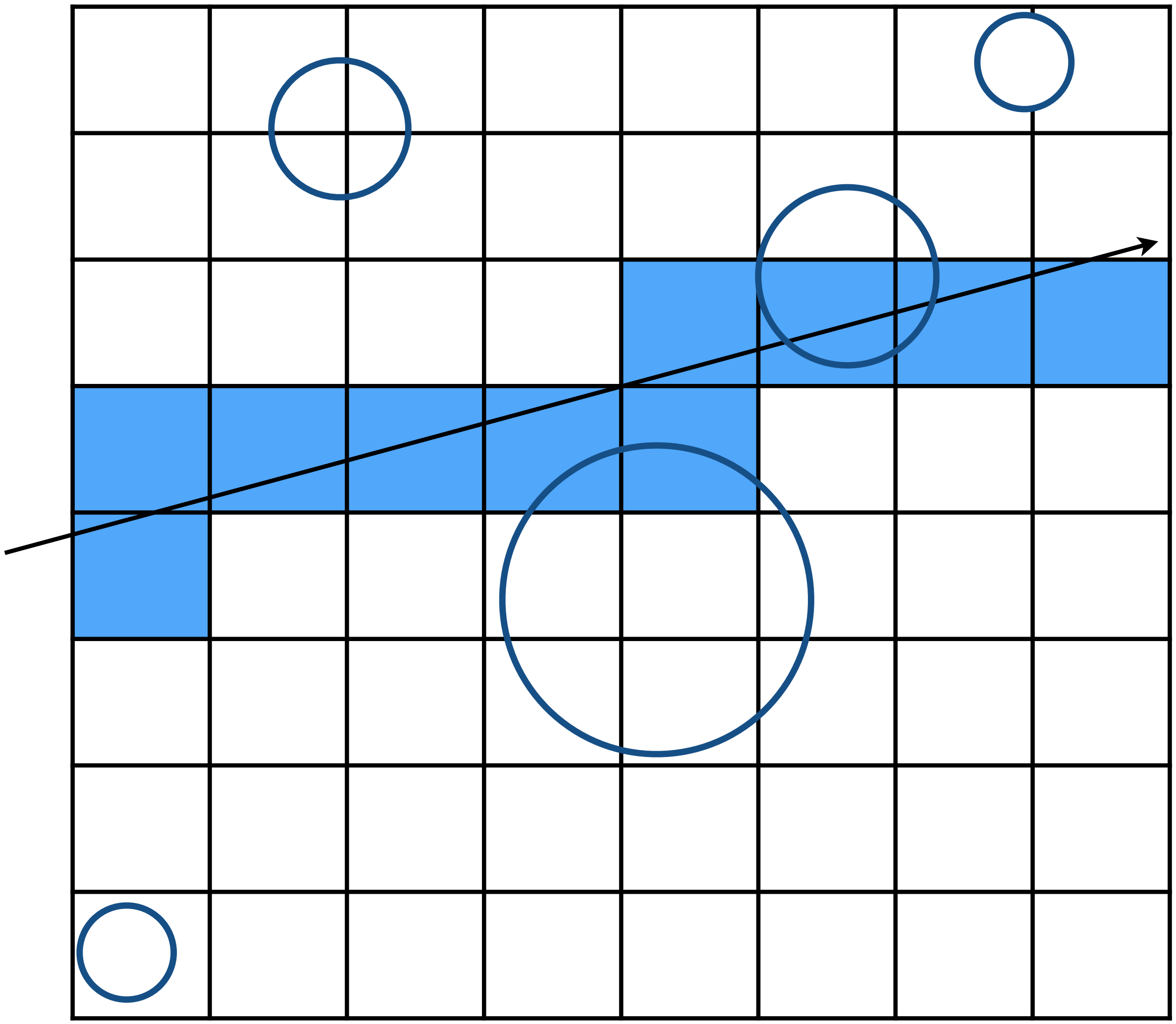
Grid Resolution?



Too many cells

- Inefficiency due to extraneous grid traversal

Grid Resolution?



Heuristic:

- #cells = C * #objs
- C ≈ 27 in 3D

Uniform Grids – When They Work Well

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

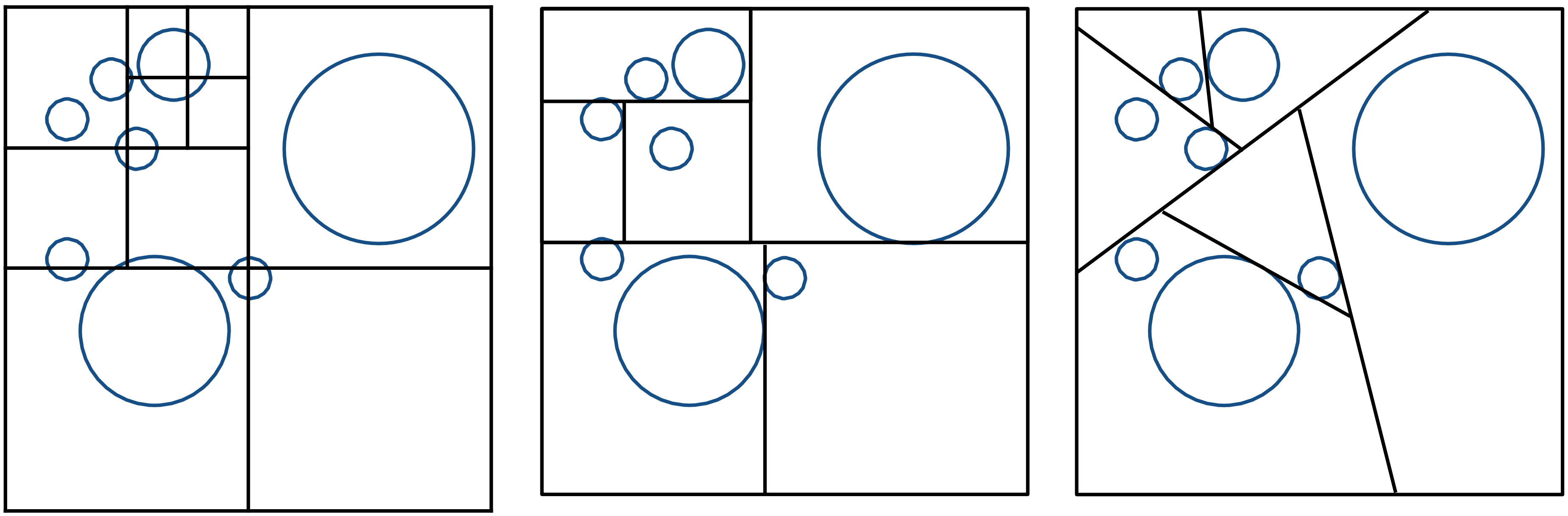
Uniform Grids – When They Fail


Jun Yan, Tracy Renderer

"Teapot in a stadium" problem

Spatial Partitions

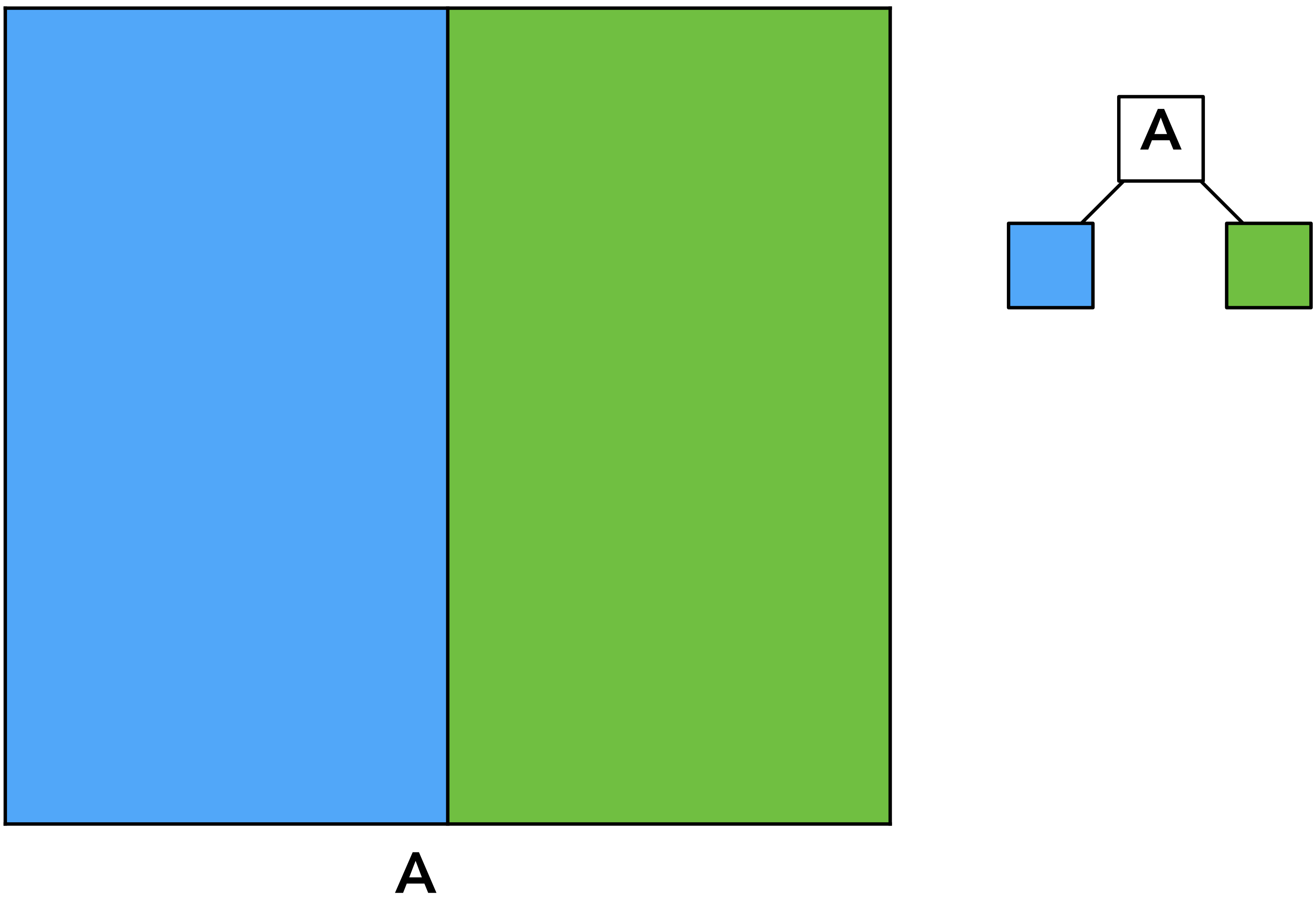
Spatial Partitioning Examples



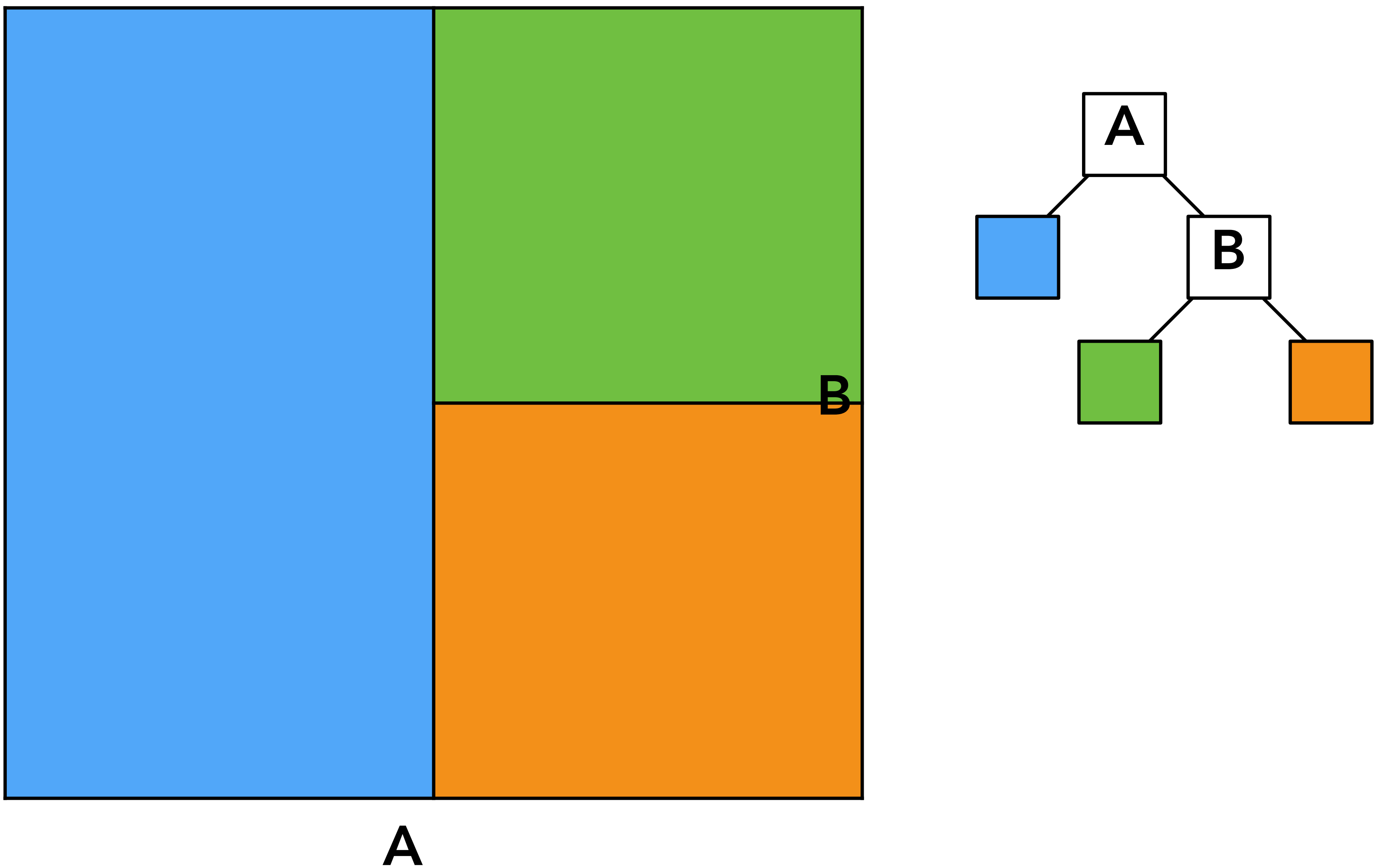
Oct-Tree **KD-Tree** BSP-Tree

Note: you could have these in both 2D and 3D. In lecture we will illustrate principles in 2D.

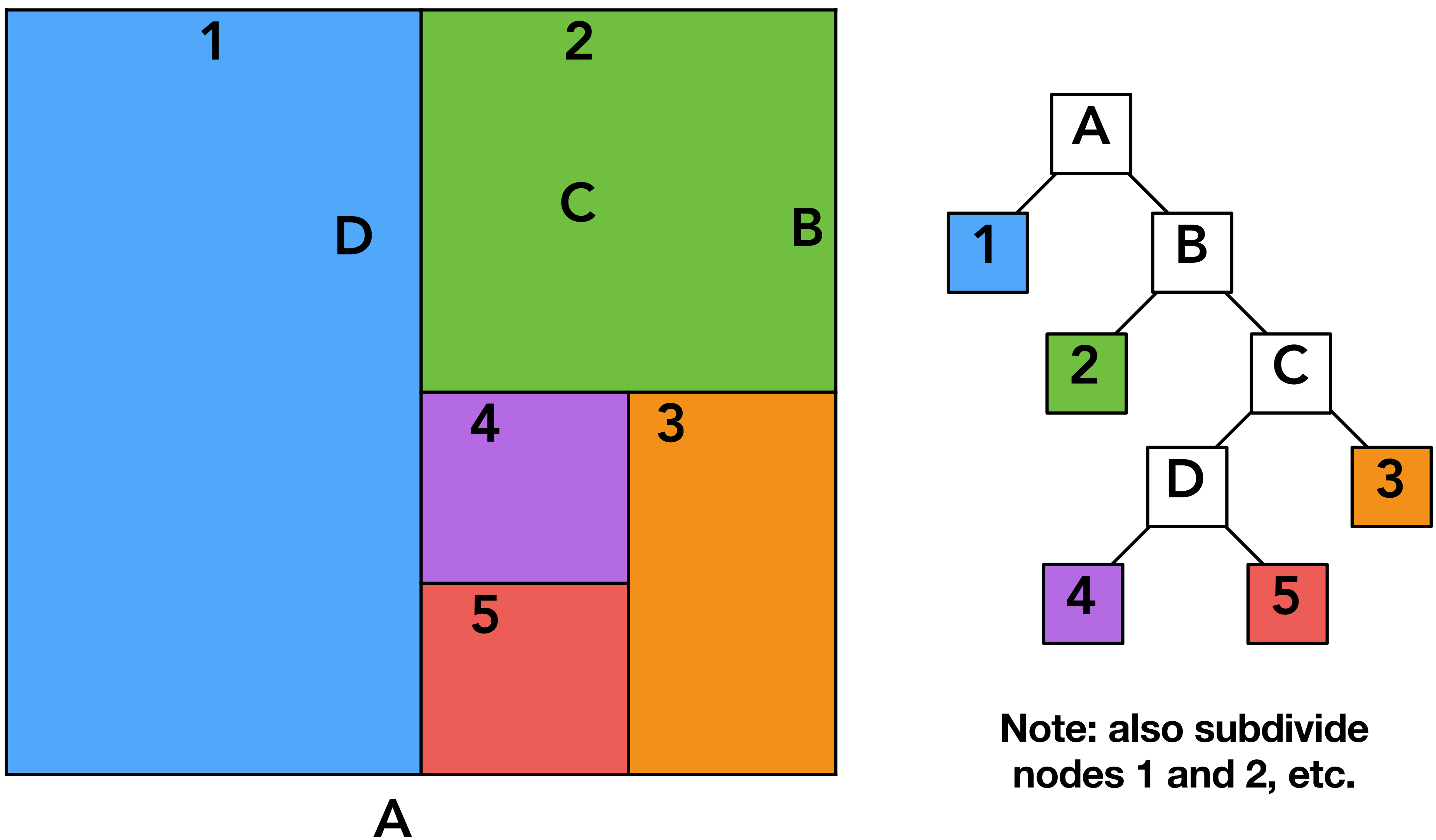
KD-Tree Pre-Processing



KD-Tree Pre-Processing



KD-Tree Pre-Processing



Note: also subdivide nodes 1 and 2, etc.

Data Structure for KD-Trees

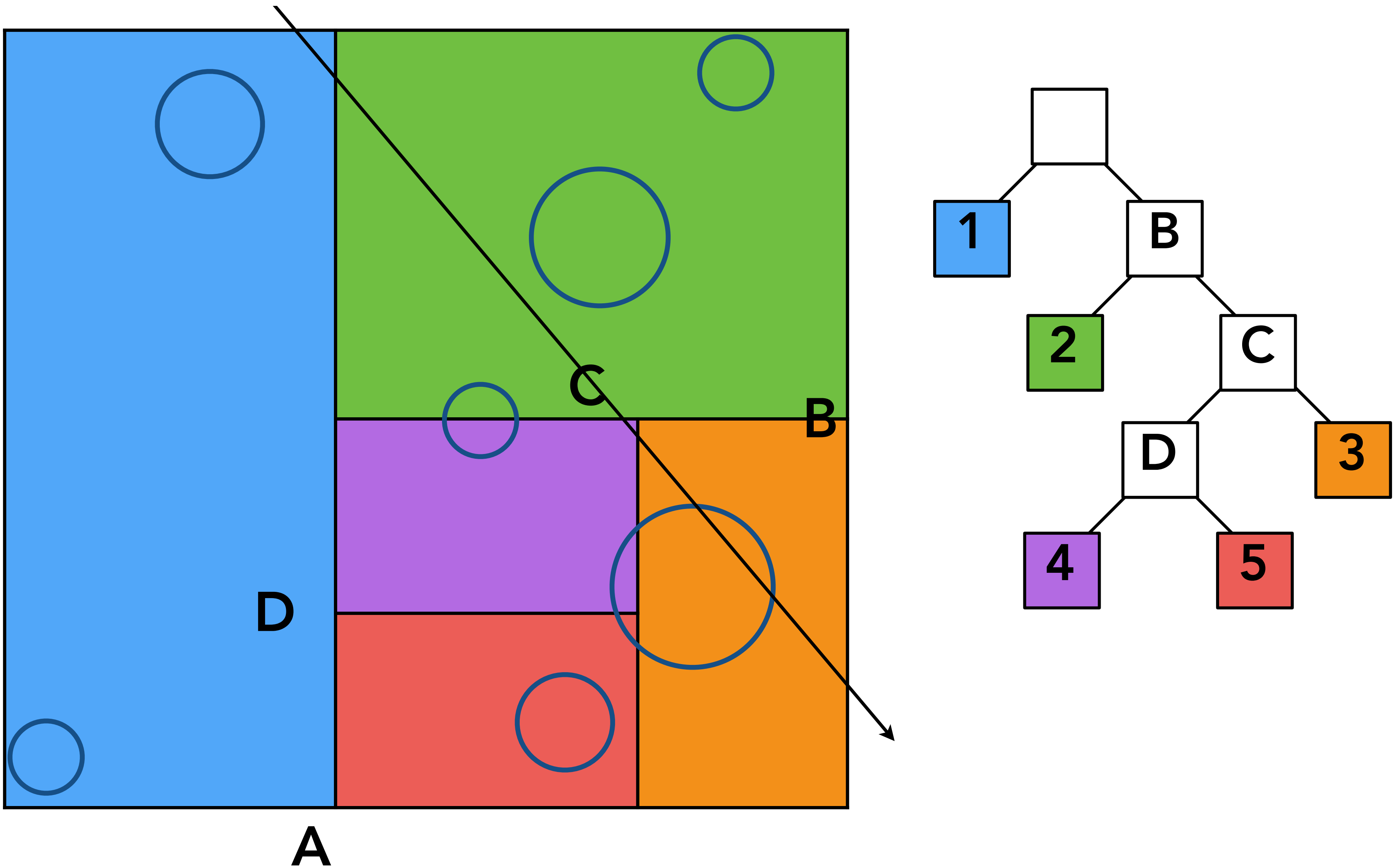
Internal nodes store

- split axis: x-, y-, or z-axis
- split position: coordinate of split plane along axis
- children: pointers to child nodes
- **No objects are stored in internal nodes**

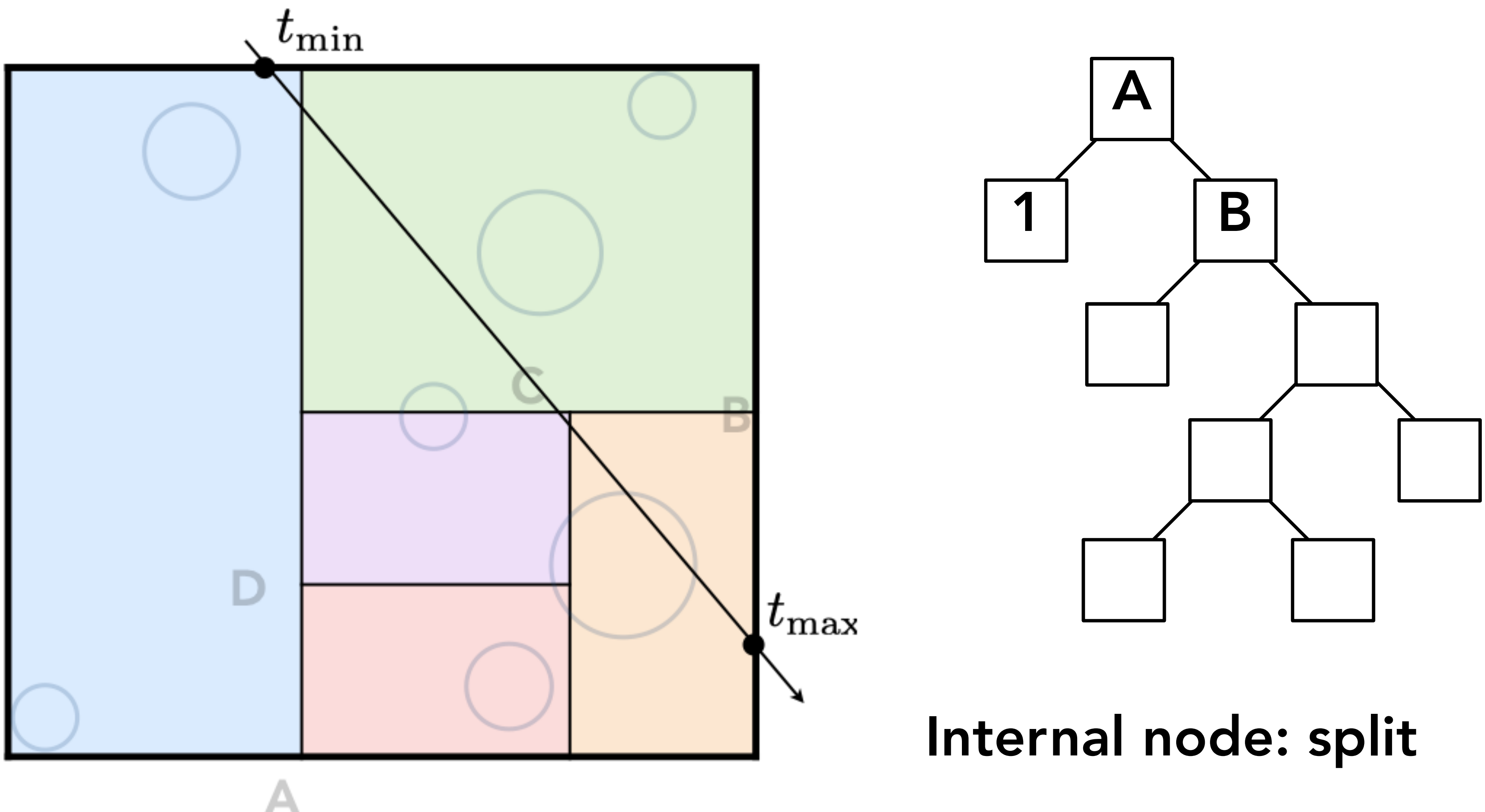
Leaf nodes store

- list of objects

Traversing a KD-Tree

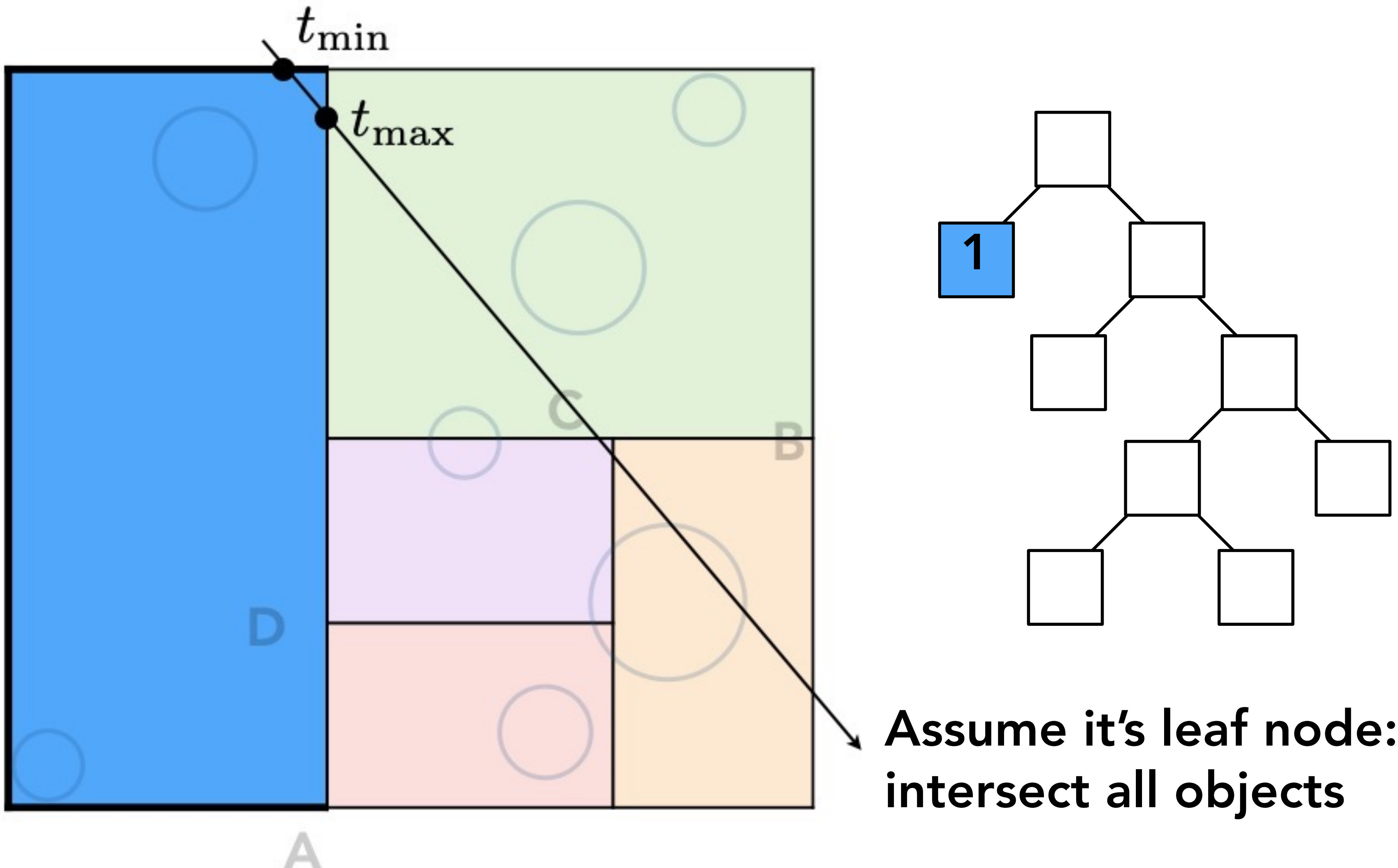


Traversing a KD-Tree



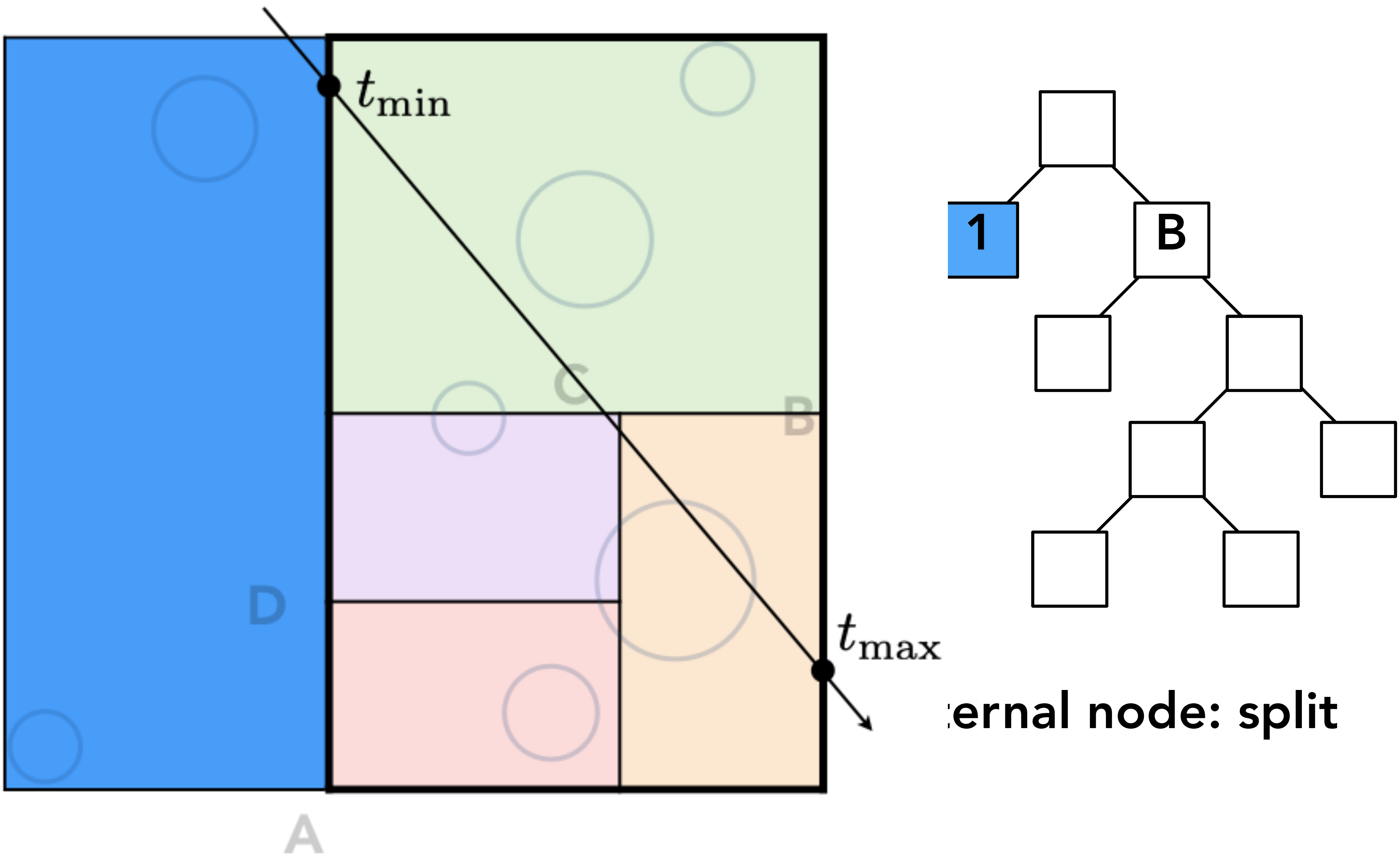
Internal node: split

Traversing a KD-Tree



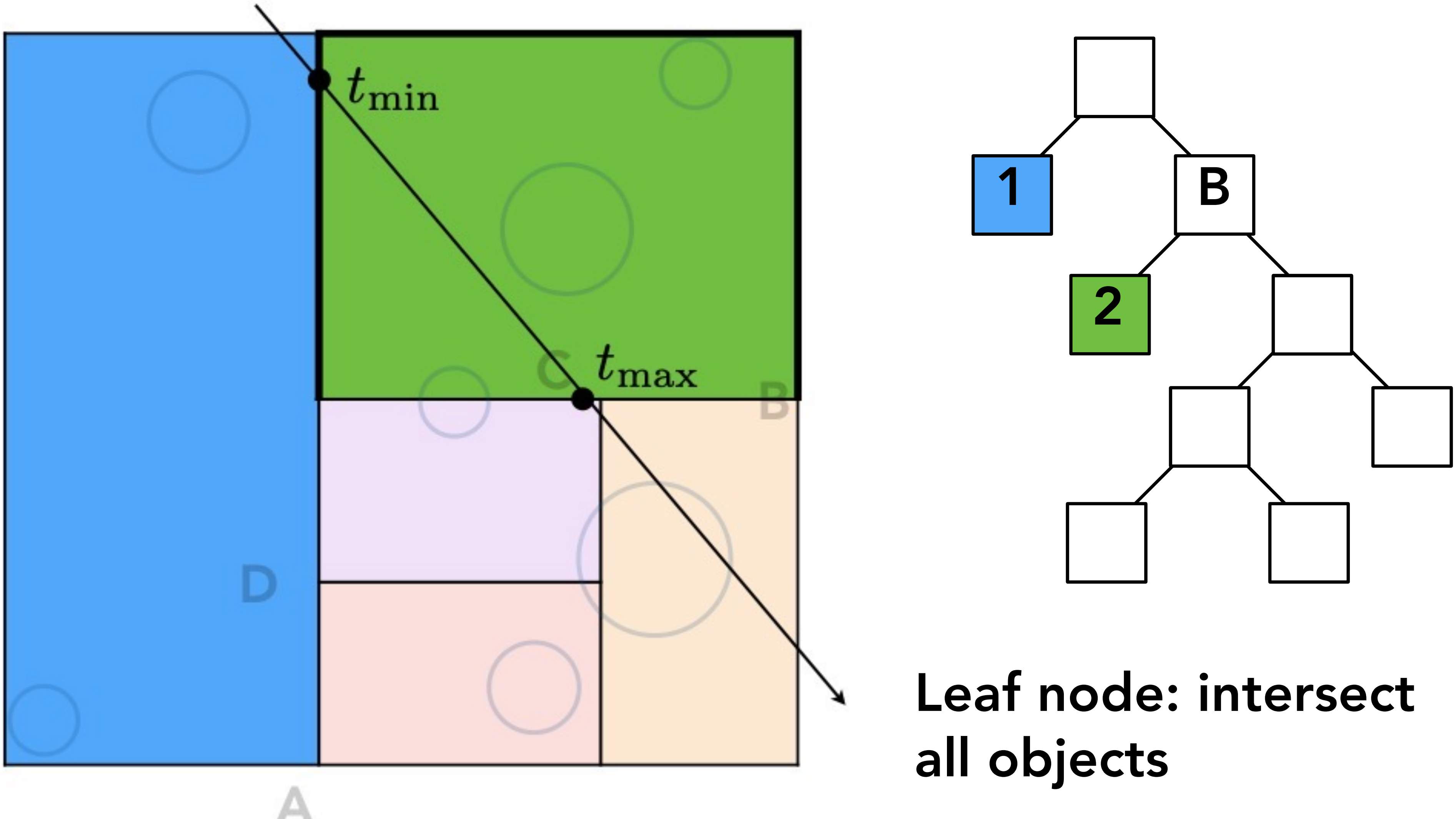
Assume it's leaf node: intersect all objects

Traversing a KD-Tree



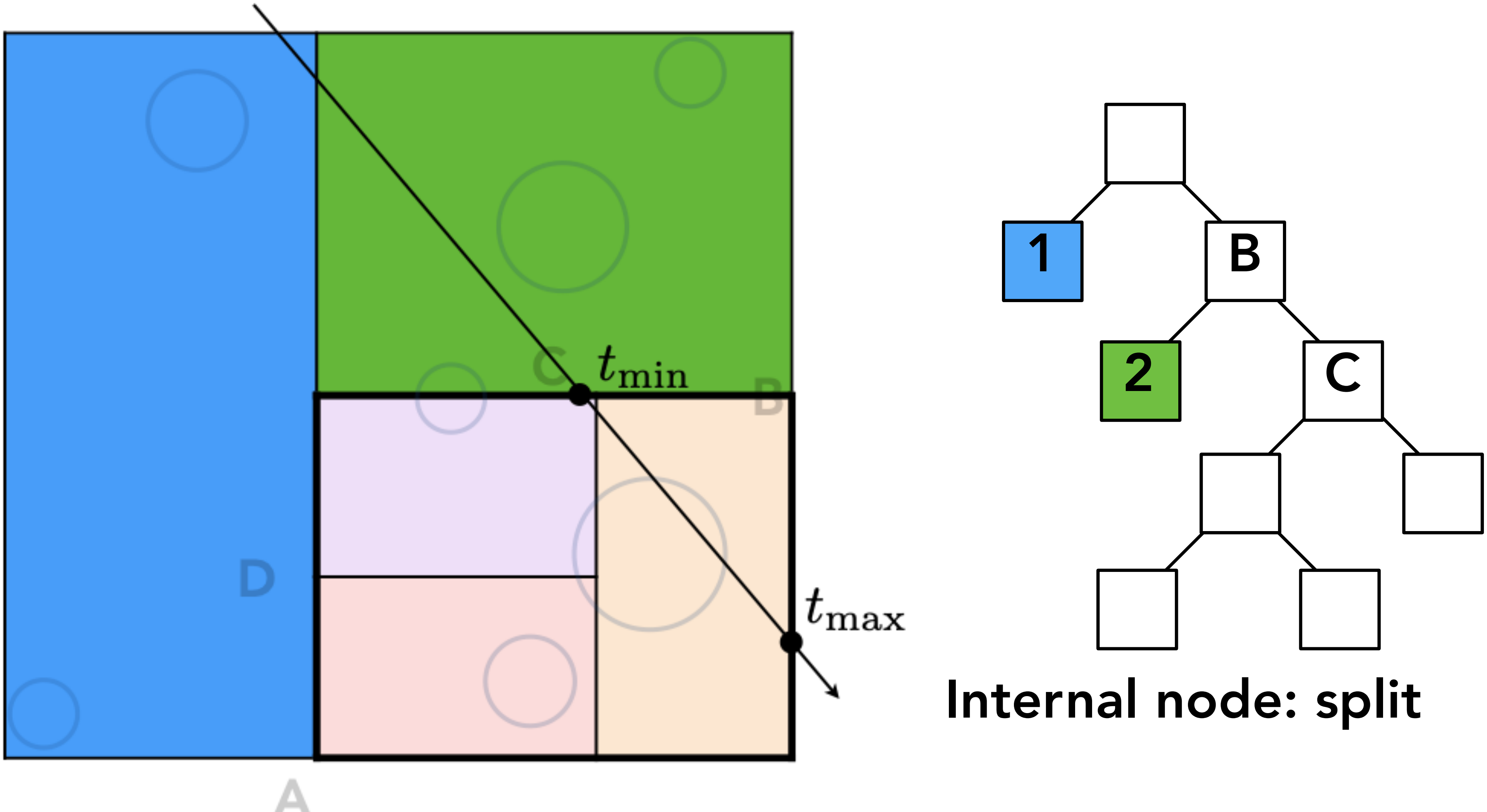
ernal node: split

Traversing a KD-Tree



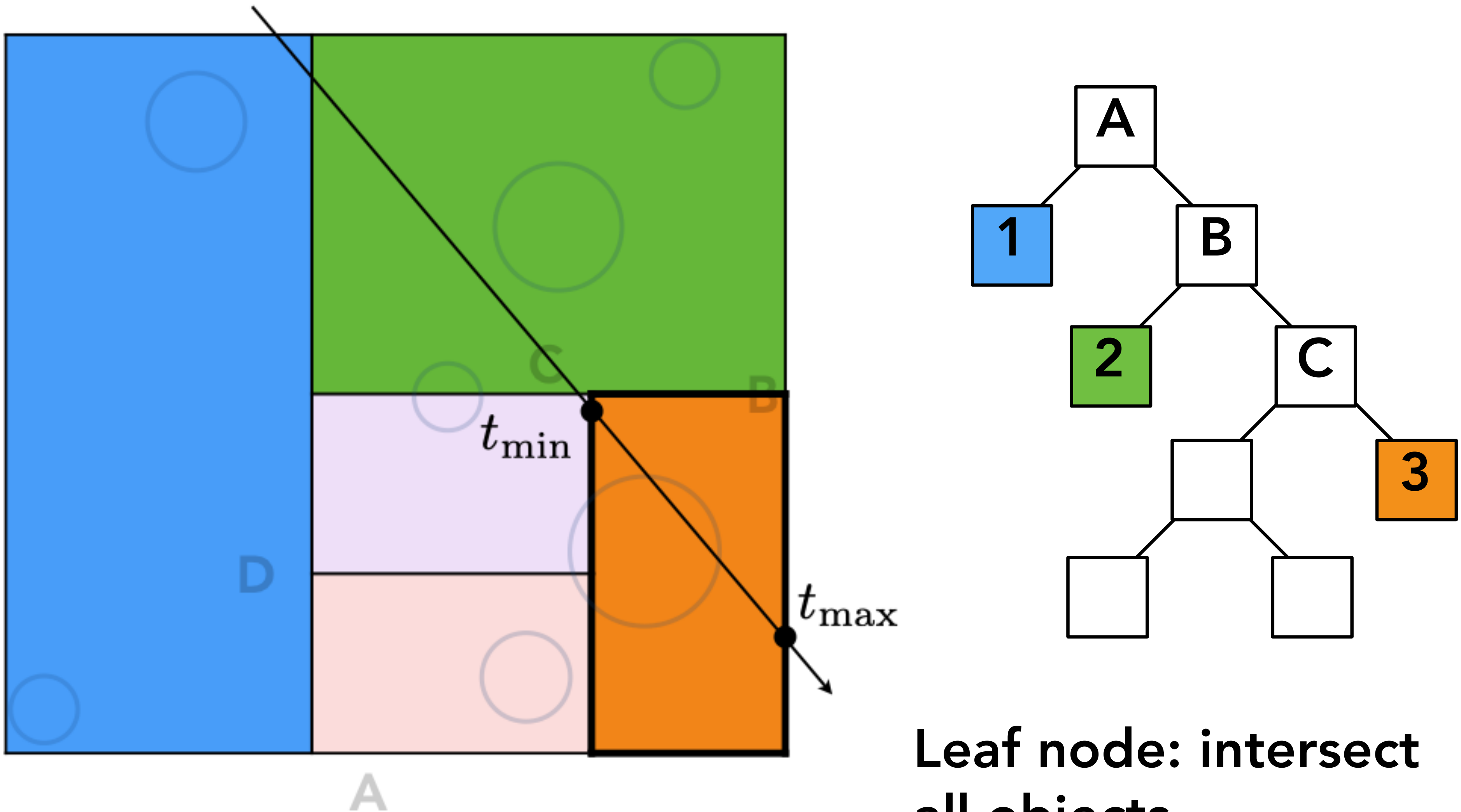
Leaf node: intersect all objects

Traversing a KD-Tree



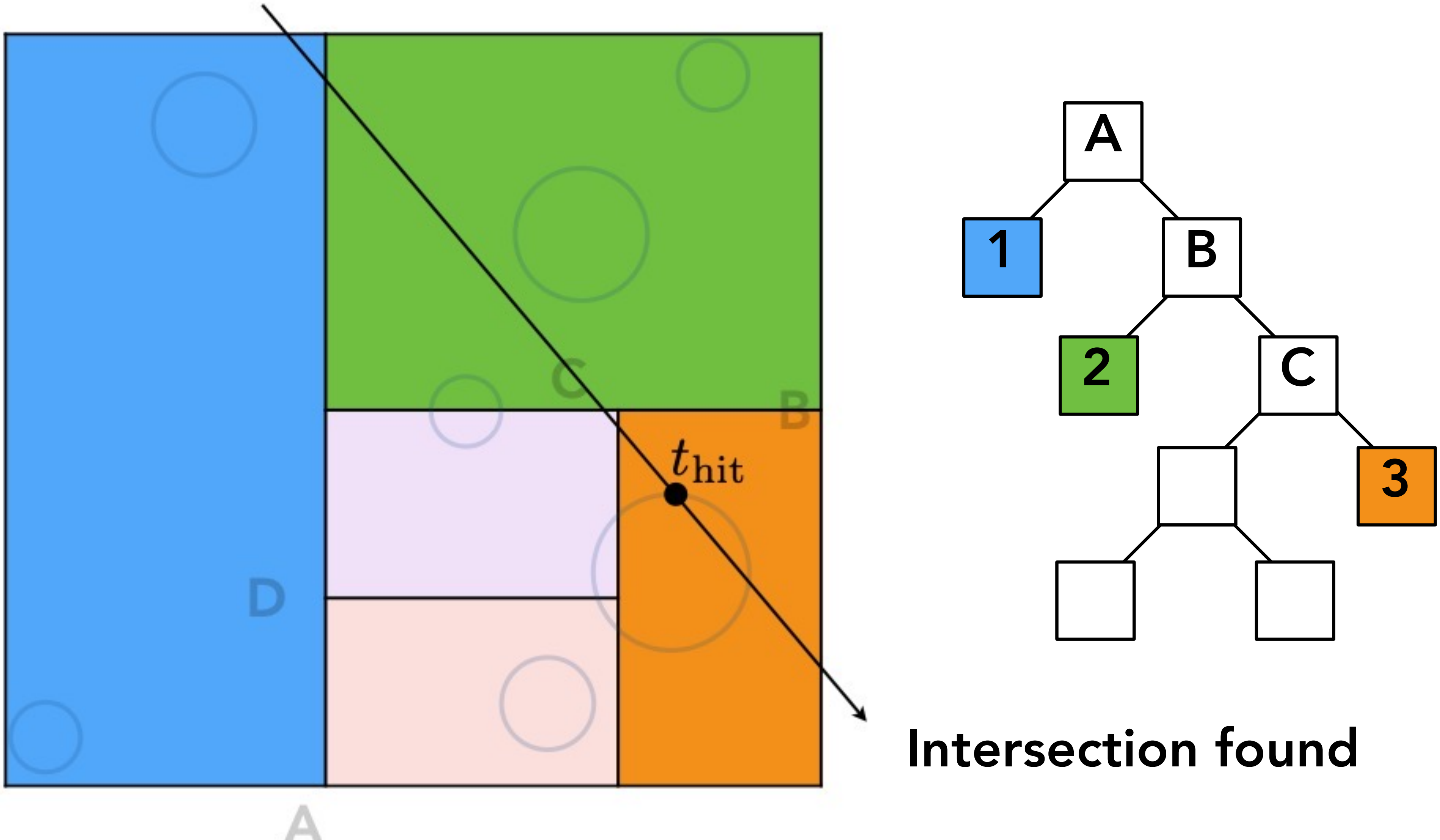
Internal node: split

Traversing a KD-Tree



Leaf node: intersect all objects

Traversing a KD-Tree

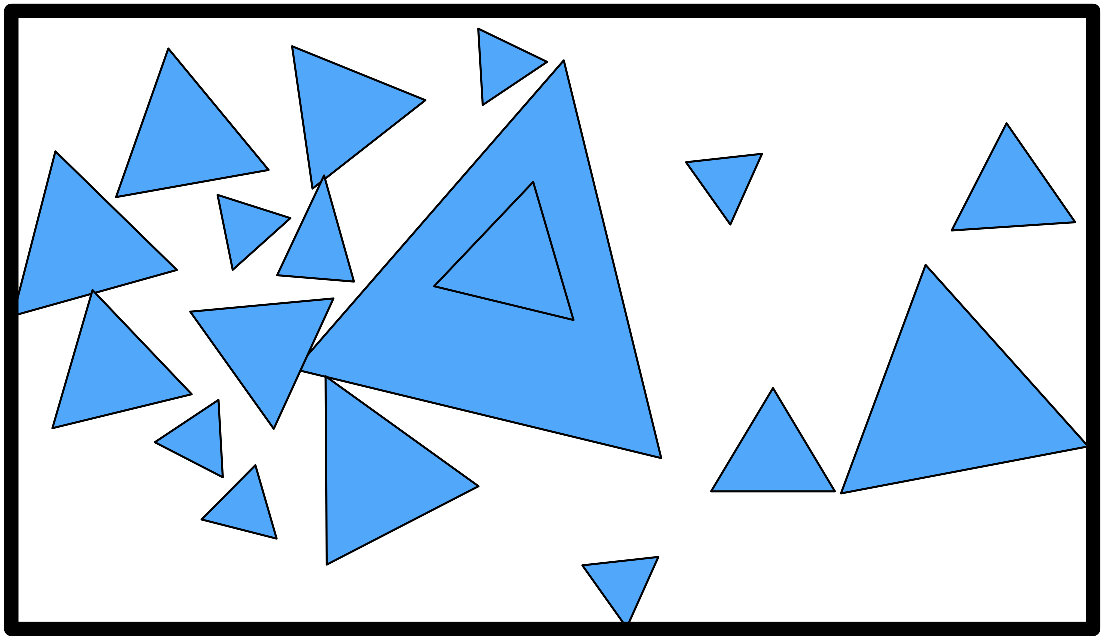


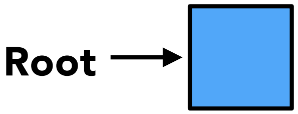
Intersection found

Object Partitions &

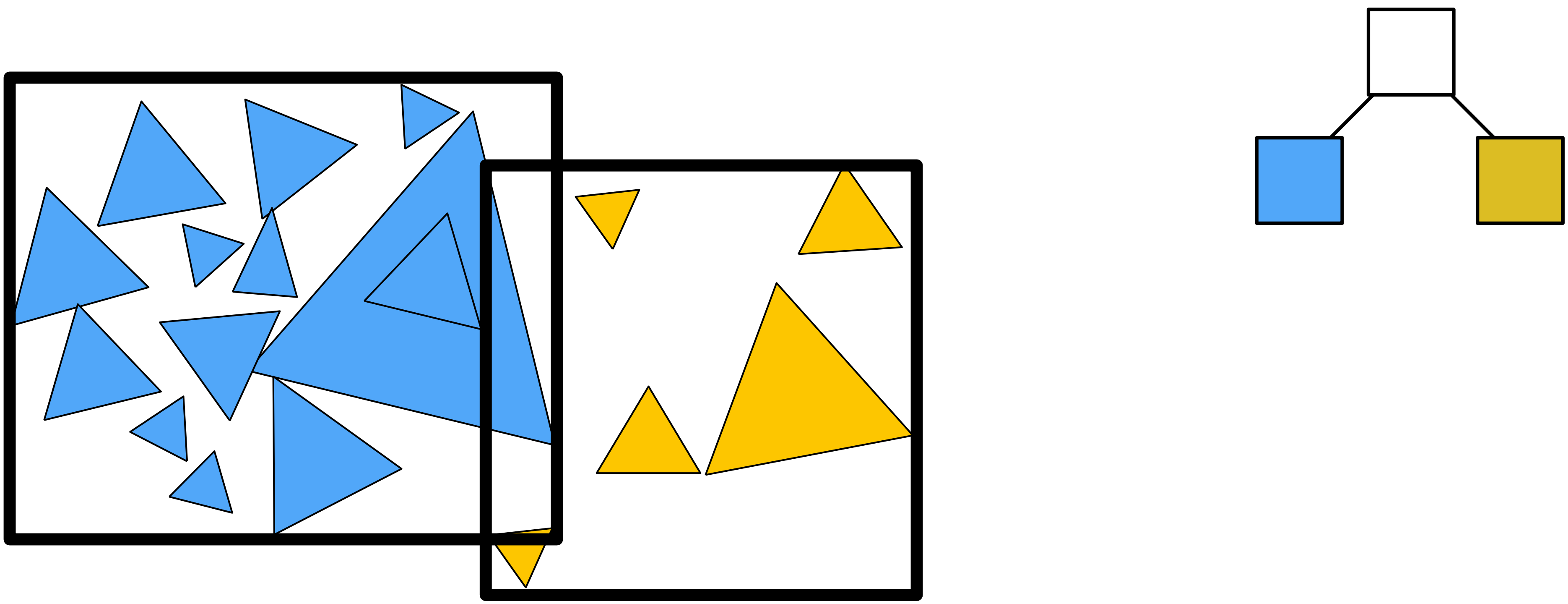
Bounding Volume Hierarchy (BVH)

Bounding Volume Hierarchy (BVH)

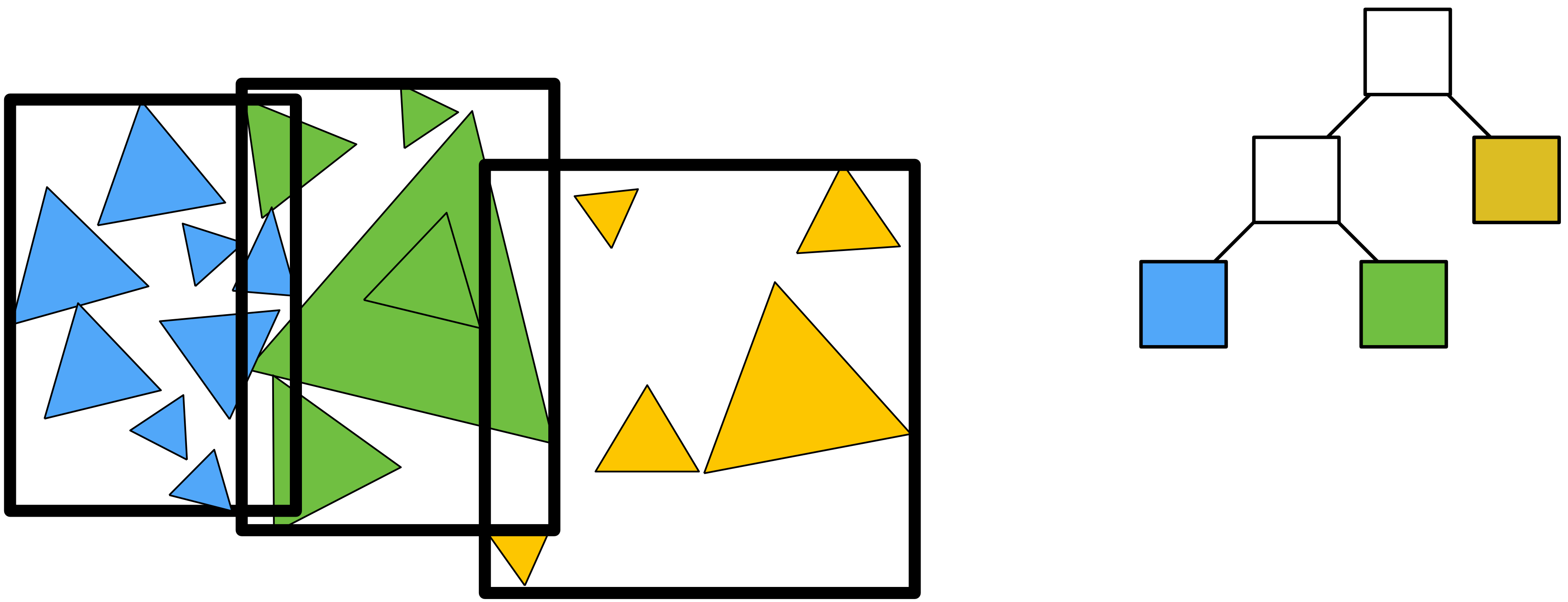




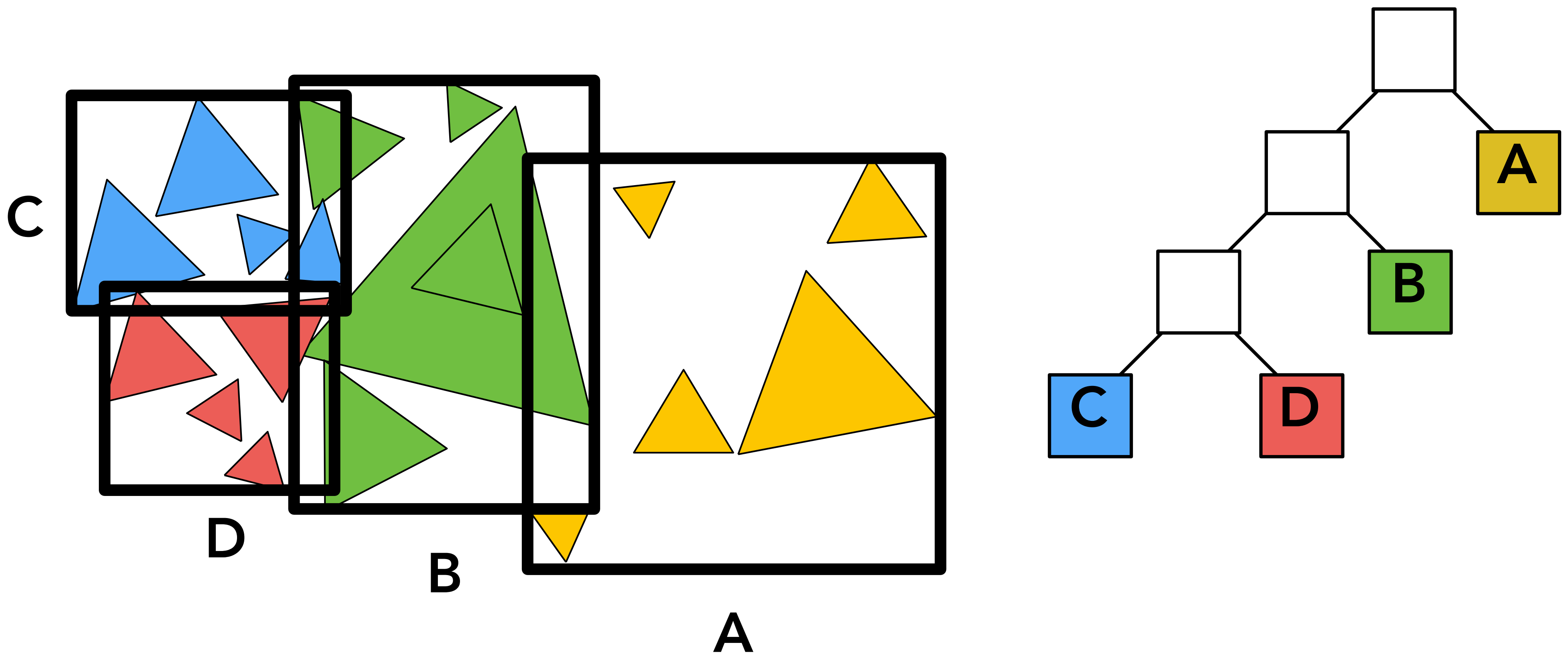
Bounding Volume Hierarchy (BVH)



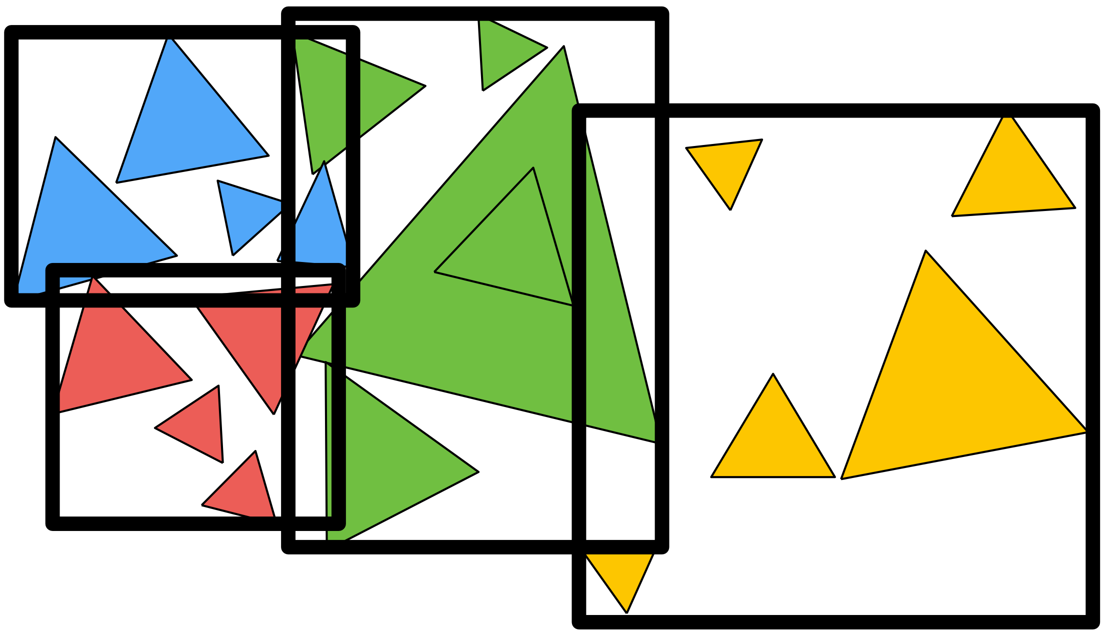
Bounding Volume Hierarchy (BVH)



Bounding Volume Hierarchy (BVH)



Summary: Building BVHs



- Find bounding box
- Recursively split set of objects in two subsets
- **Recompute** the bounding box of the subsets
- Stop when necessary
- Store objects in each leaf node

Building BVHs

How to subdivide a node?

- Choose a dimension to split
- Heuristic #1: Always choose the longest axis in node
- Heuristic #2: Split node at location of **median** object

Termination criteria?

- Heuristic: stop when node contains few elements (e.g. 5)

Data Structure for BVHs

Internal nodes store

- Bounding box
- Children: pointers to child nodes

Leaf nodes store

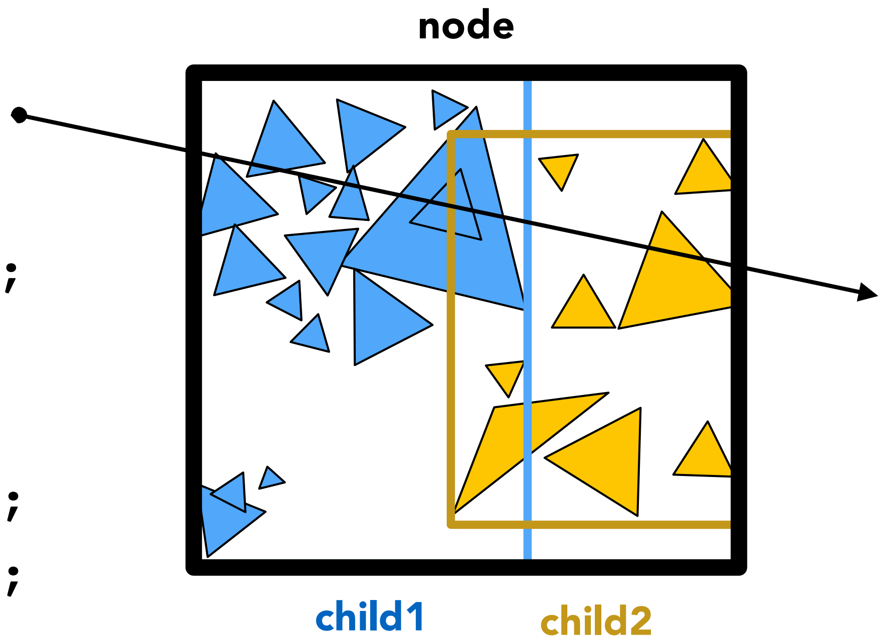
- Bounding box
- List of objects

Nodes represent subset of primitives in scene

- All objects in subtree

BVH Traversal

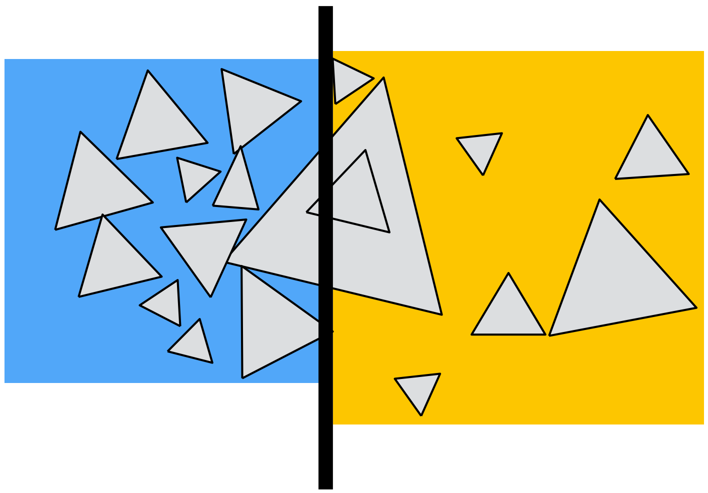
```
Intersect(Ray ray, BVH node) {
  if (ray misses node.bbox) return;

  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 the closer of hit1, hit2;
}
```



Spatial vs Object Partitions

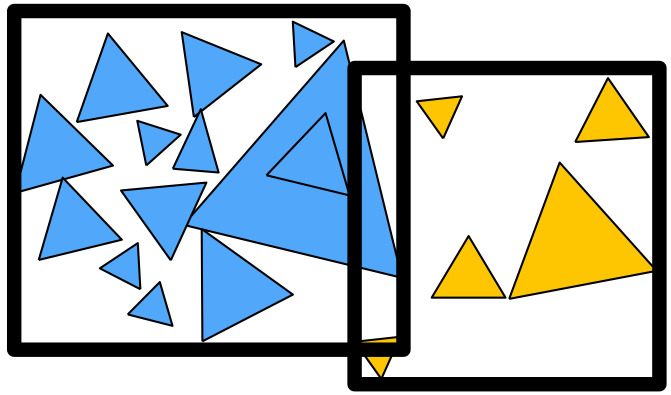
Spatial partition (e.g.KD-tree)

- Partition space into non-overlapping regions
- An object 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



Thank you!

(And thank Prof. Ravi Ramamoorthi (UCSD), Prof. Ren Ng (UC Berkeley),
Prof. Lingqi Yan (UCSB) for many of the slides!)