Lectures 9-11:

Intro to Raytracing

Computer Graphics and Imaging

UC Berkeley CS184

Towards Photorealistic Rendering

Credit: Bertrand Benoit. “Sweet Feast,” 2009. [Blender /VRay]

Course Roadmap

Rasterization Pipeline

Core Concepts

- Sampling
- Antialiasing
- Transforms

Geometric Modeling

Core Concepts

- Splines, Bezier Curves
- Topological Mesh Representations
- Subdivision, Geometry Processing

Lighting & Materials

Core Concepts

- Measuring Light
- Unbiased Integral Estimation
- Light Transport & Materials

Cameras & Imaging

Rasterization
Transforms & Projection
Texture Mapping
Visibility, Shading, Overall Pipeline — Exam 1
Intro to Geometry
Curves and Surfaces
Geometry Processing
Ray-Tracing & Acceleration — Today
Radiometry & Photometry
Monte Carlo Integration
Global Illumination & Path Tracing
Material Modeling — Exam 2

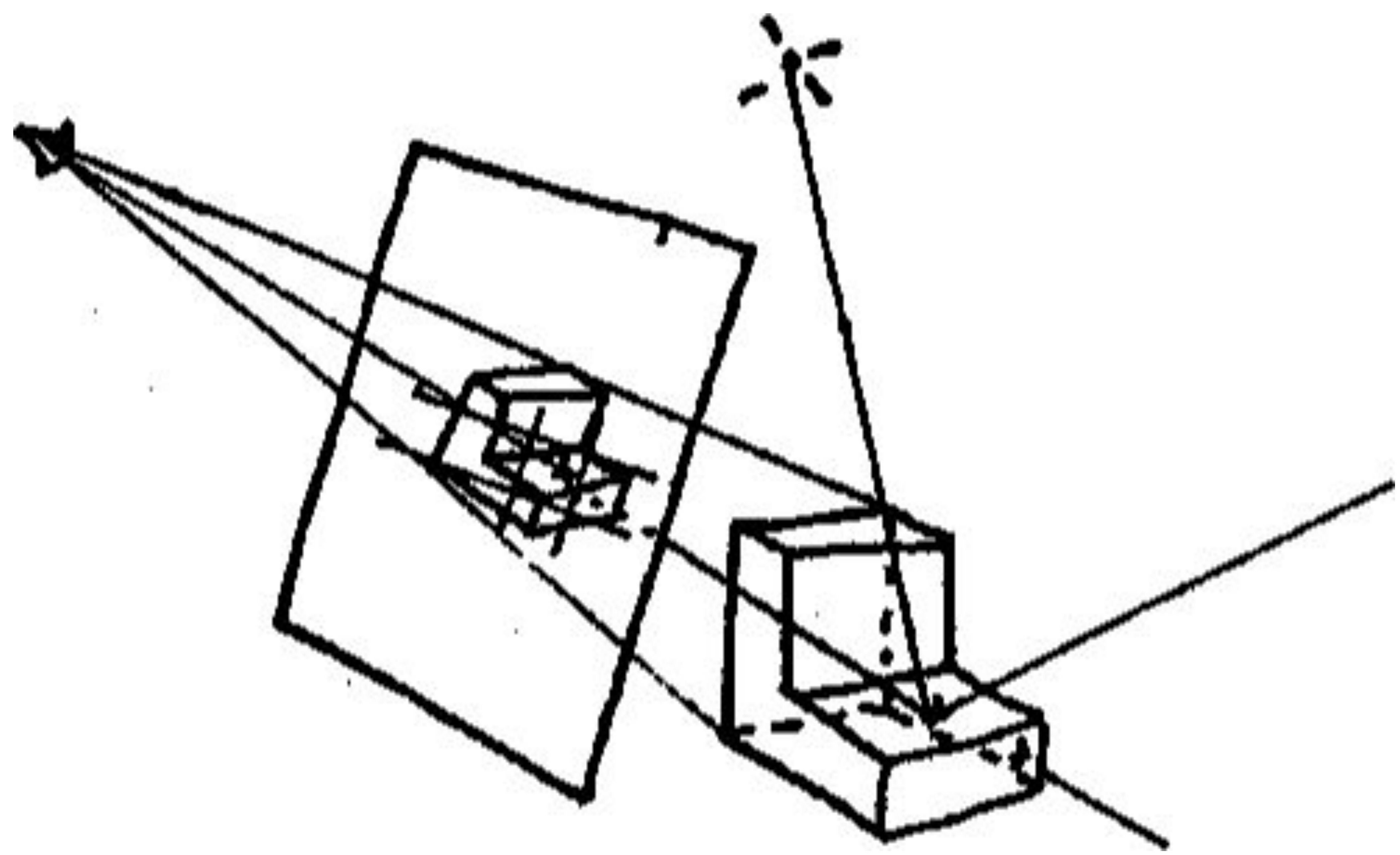
Final Project

Basic Ray-Tracing Algorithm

Ray Casting

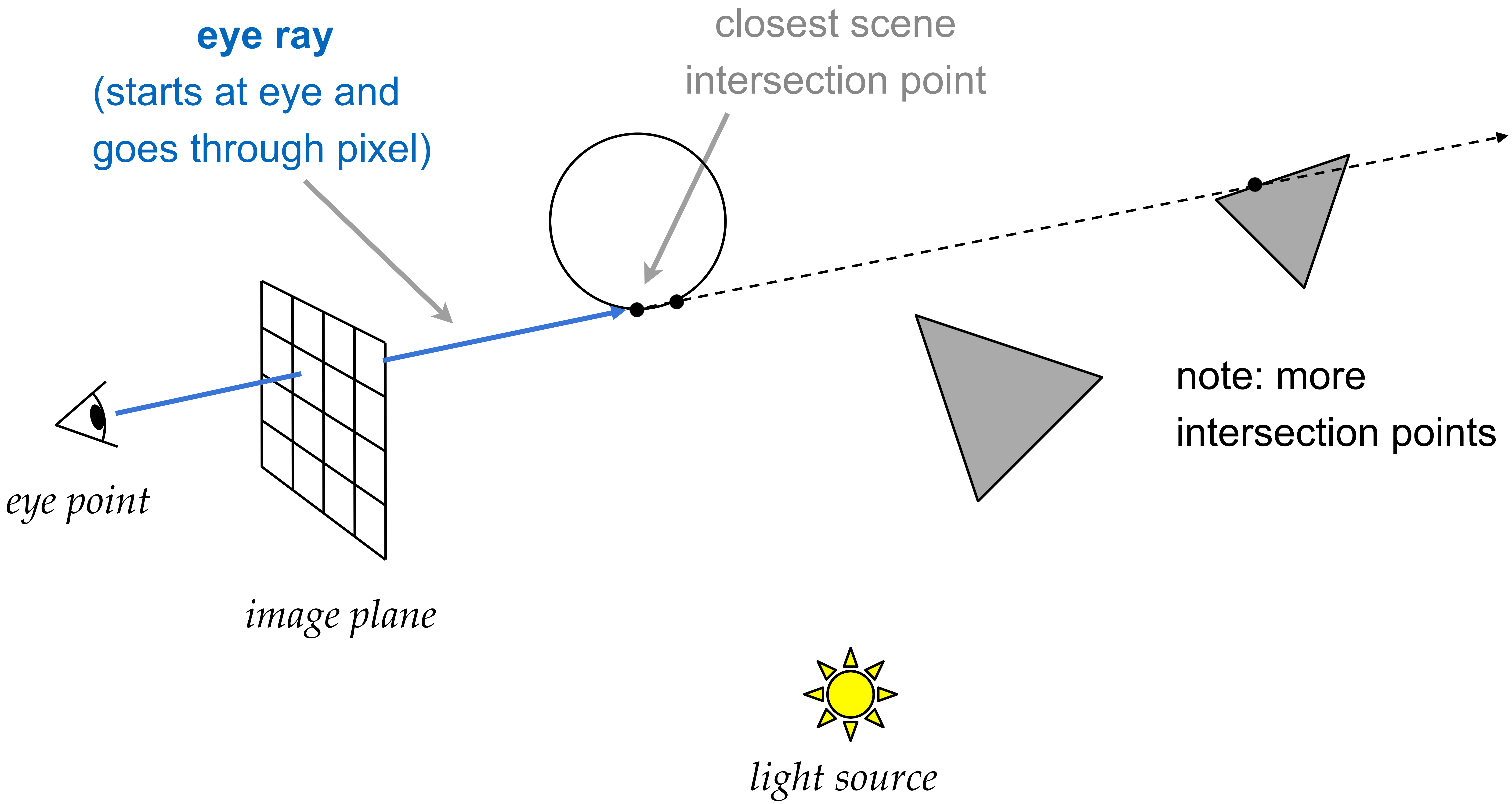
Appel 1968 - Ray casting

1. **Generate an image by casting one ray per pixel**
2. **Check for shadows by sending a ray to the light**



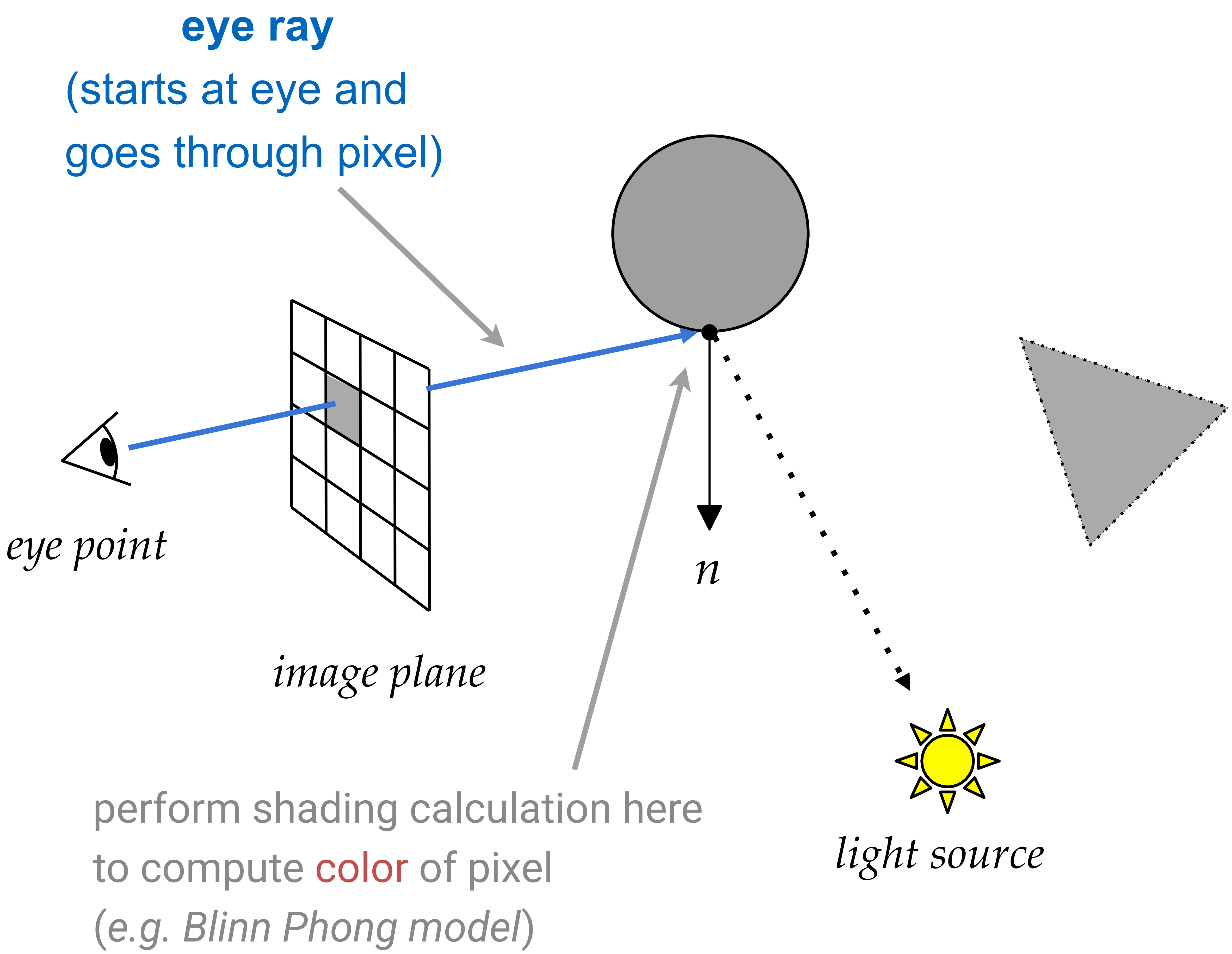
Ray Casting - Generating Eye Rays

Pinhole Camera Model



Ray Casting - Shading Pixels (Local Only)

Pinhole Camera Model

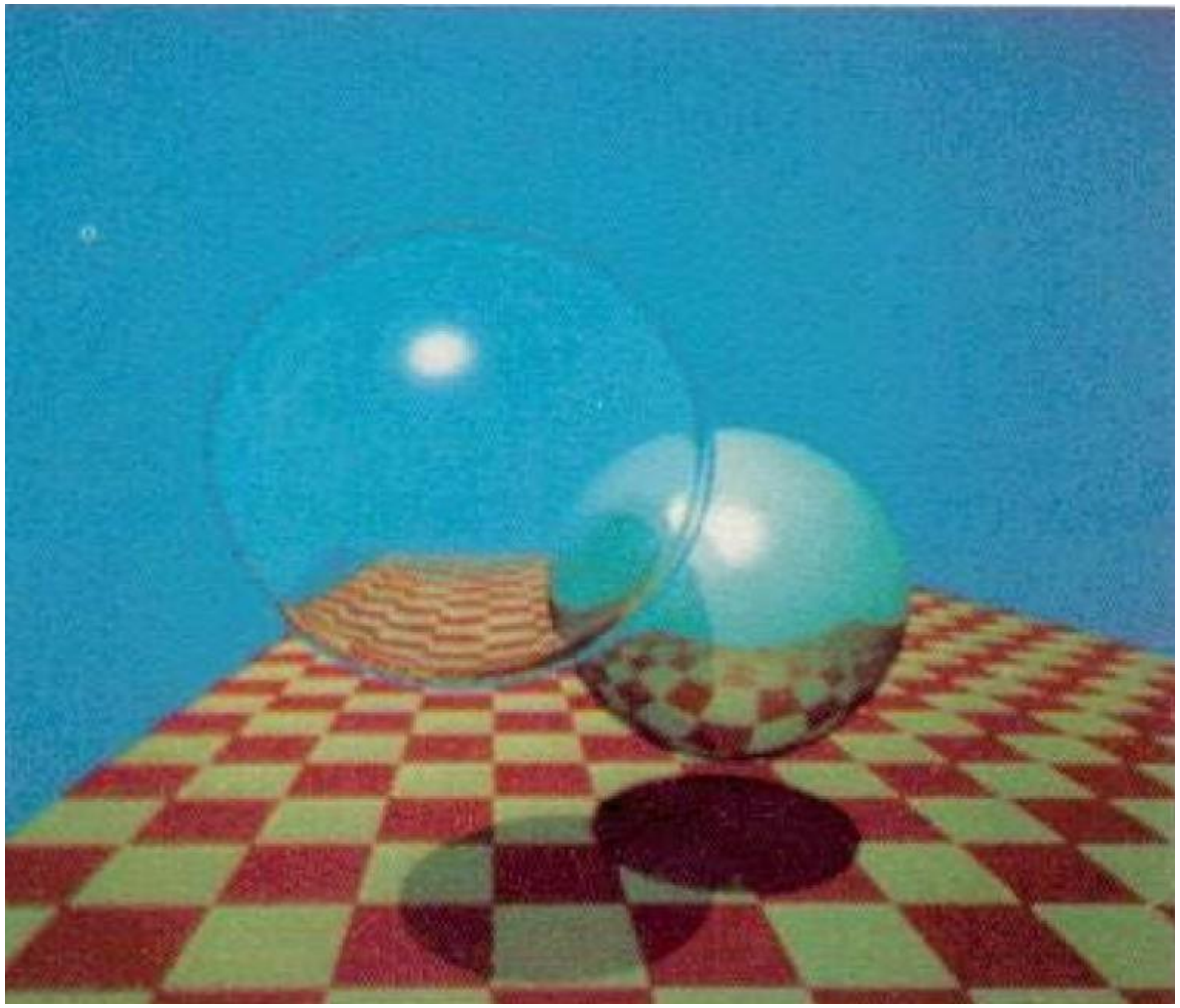


Recursive Ray Tracing

“An improved Illumination model for shaded display”
T. Whitted, CACM 1980

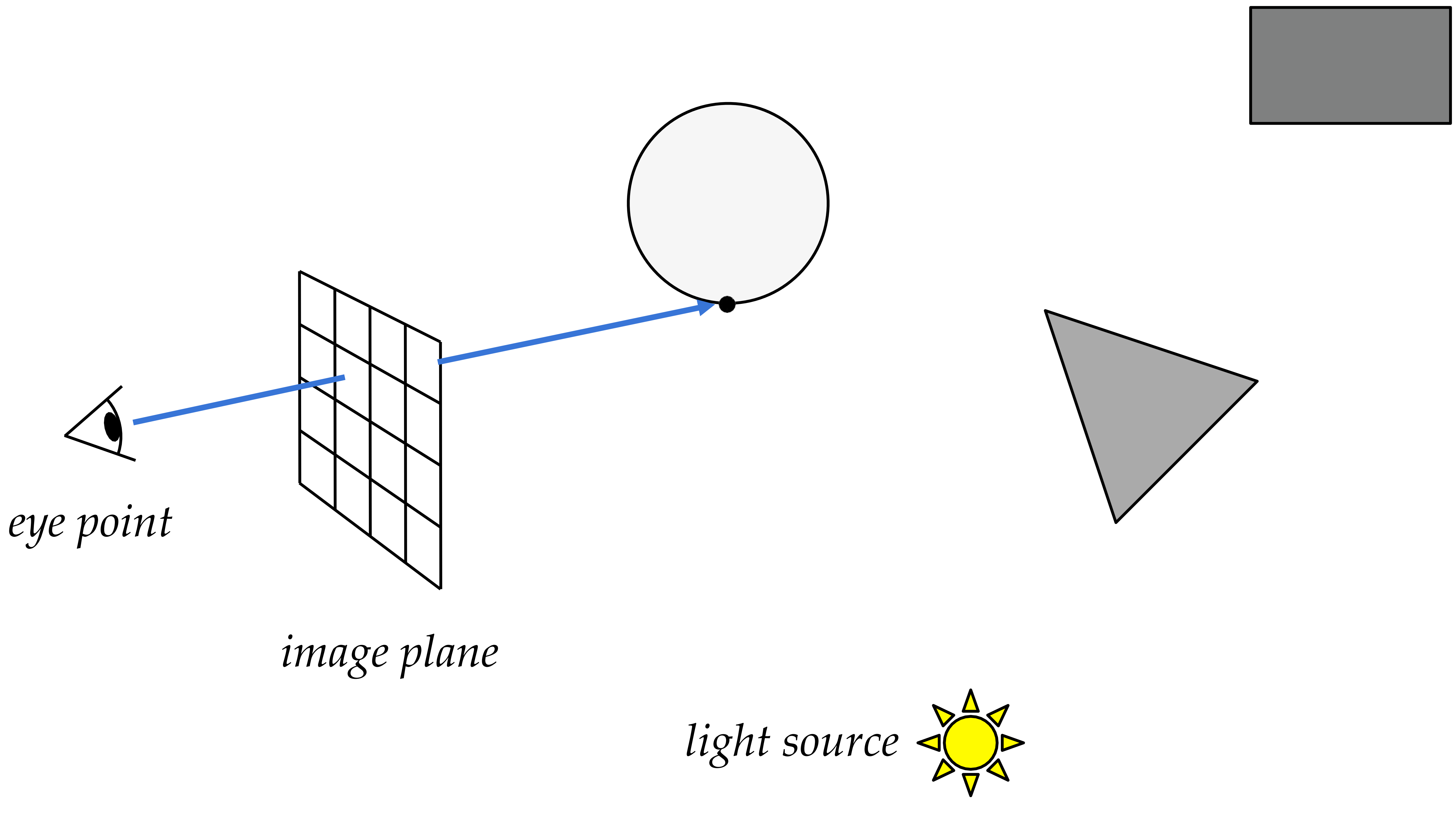
Time:

- VAX 11/780 (1979) 74min
- PC (2009) 3sec
- GPU (2019) 1/240sec

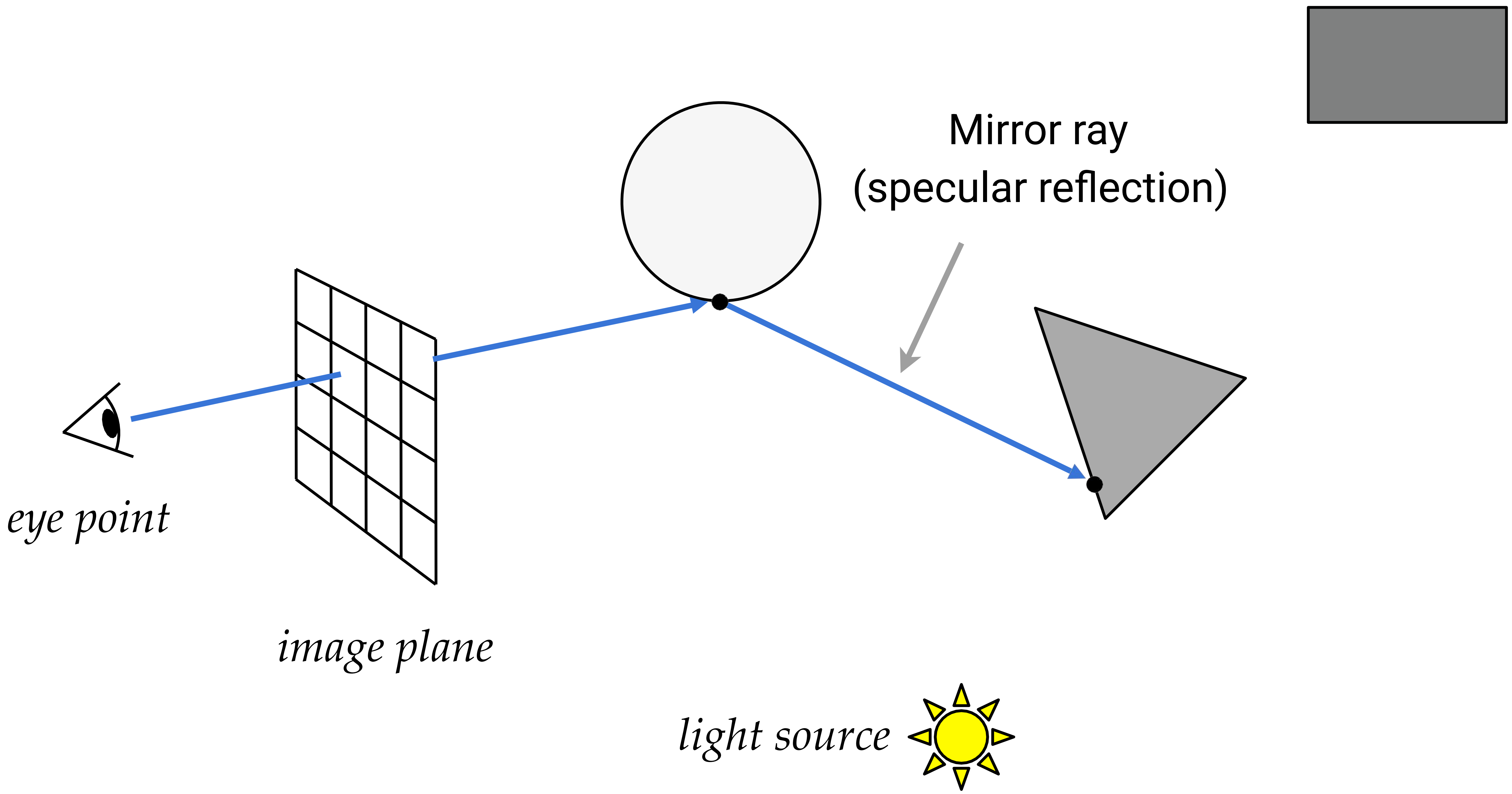


Spheres and Checkerboard, T. Whitted, 1979

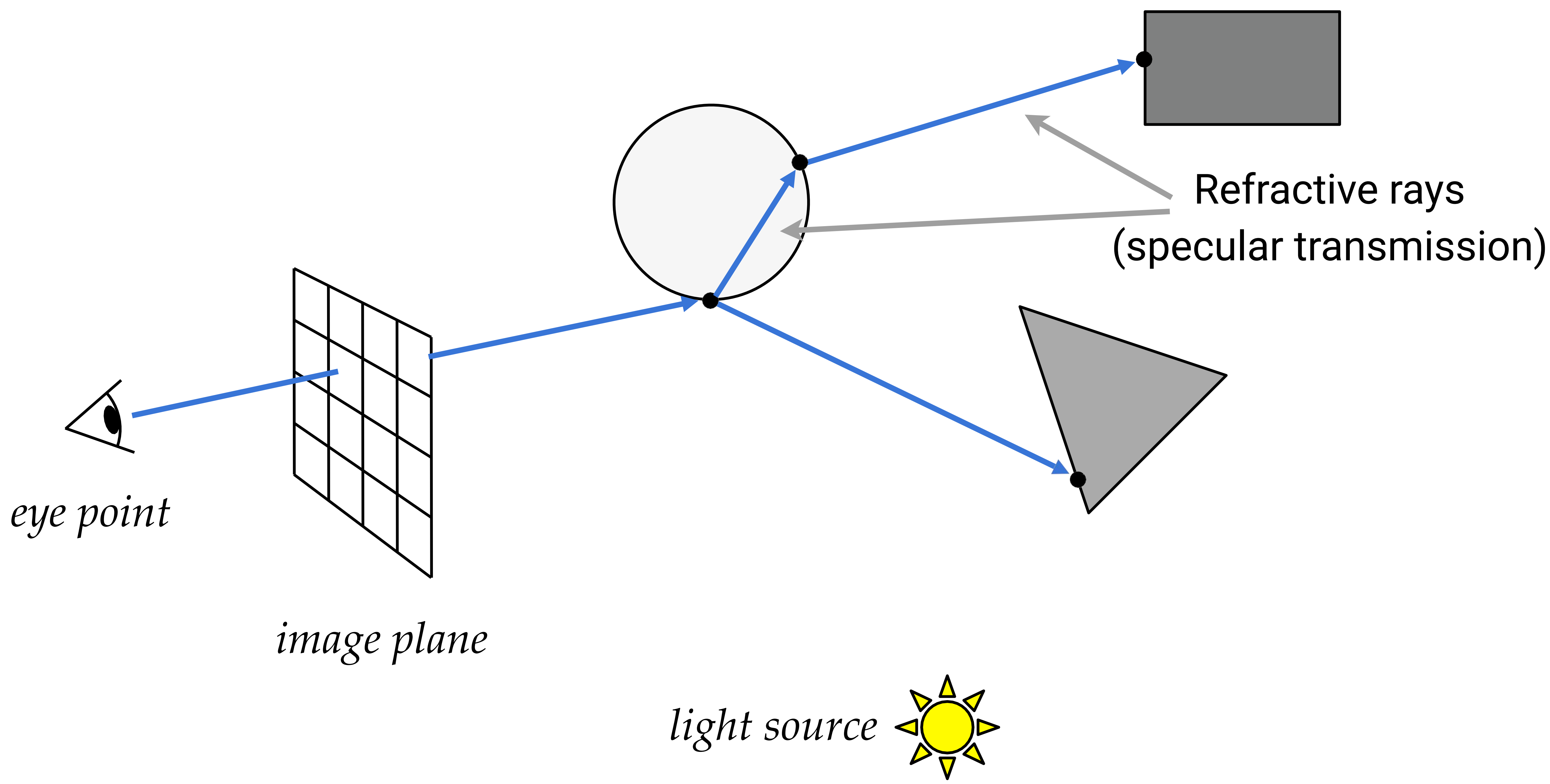
Recursive Ray Tracing



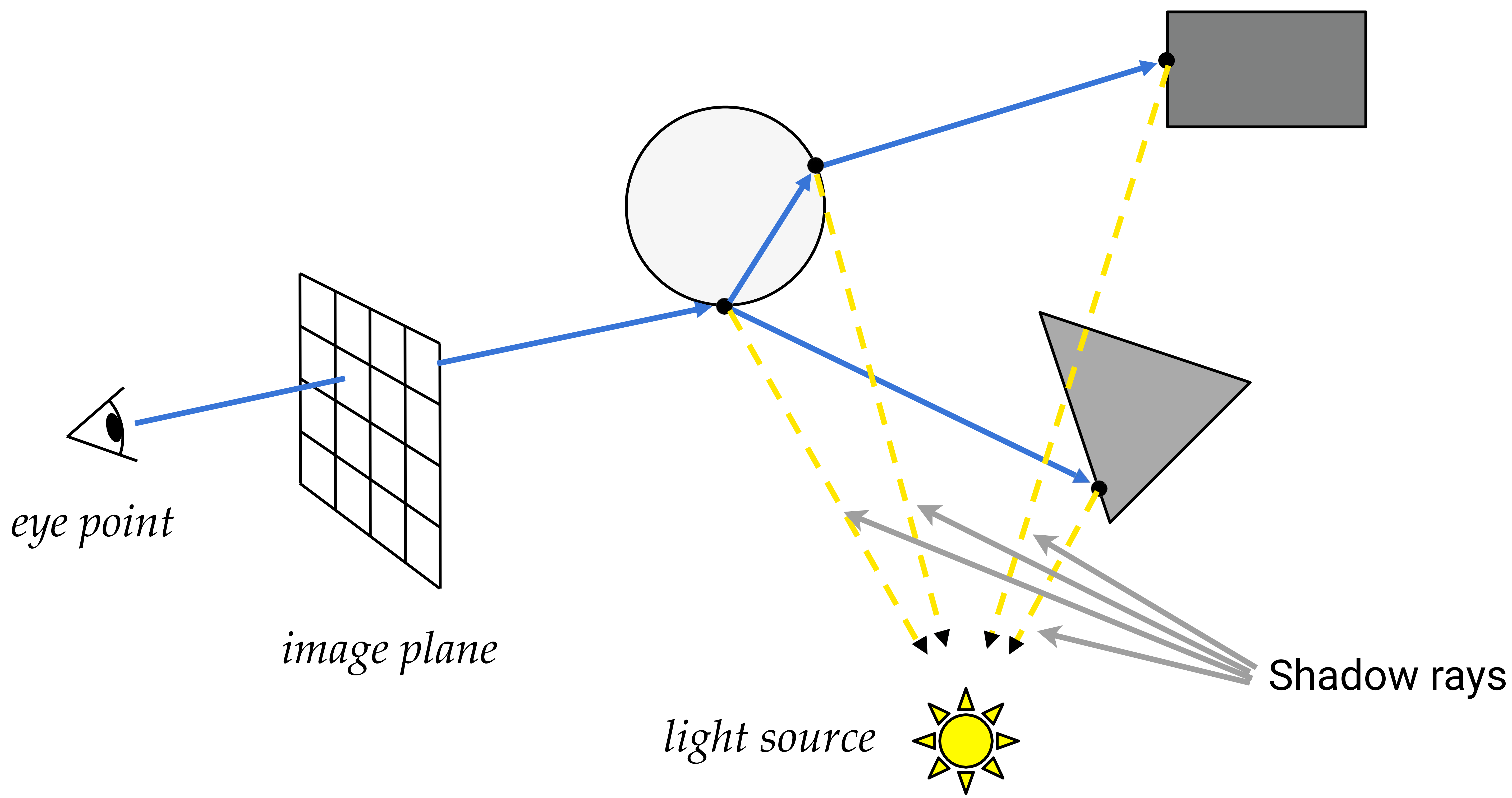
Recursive Ray Tracing



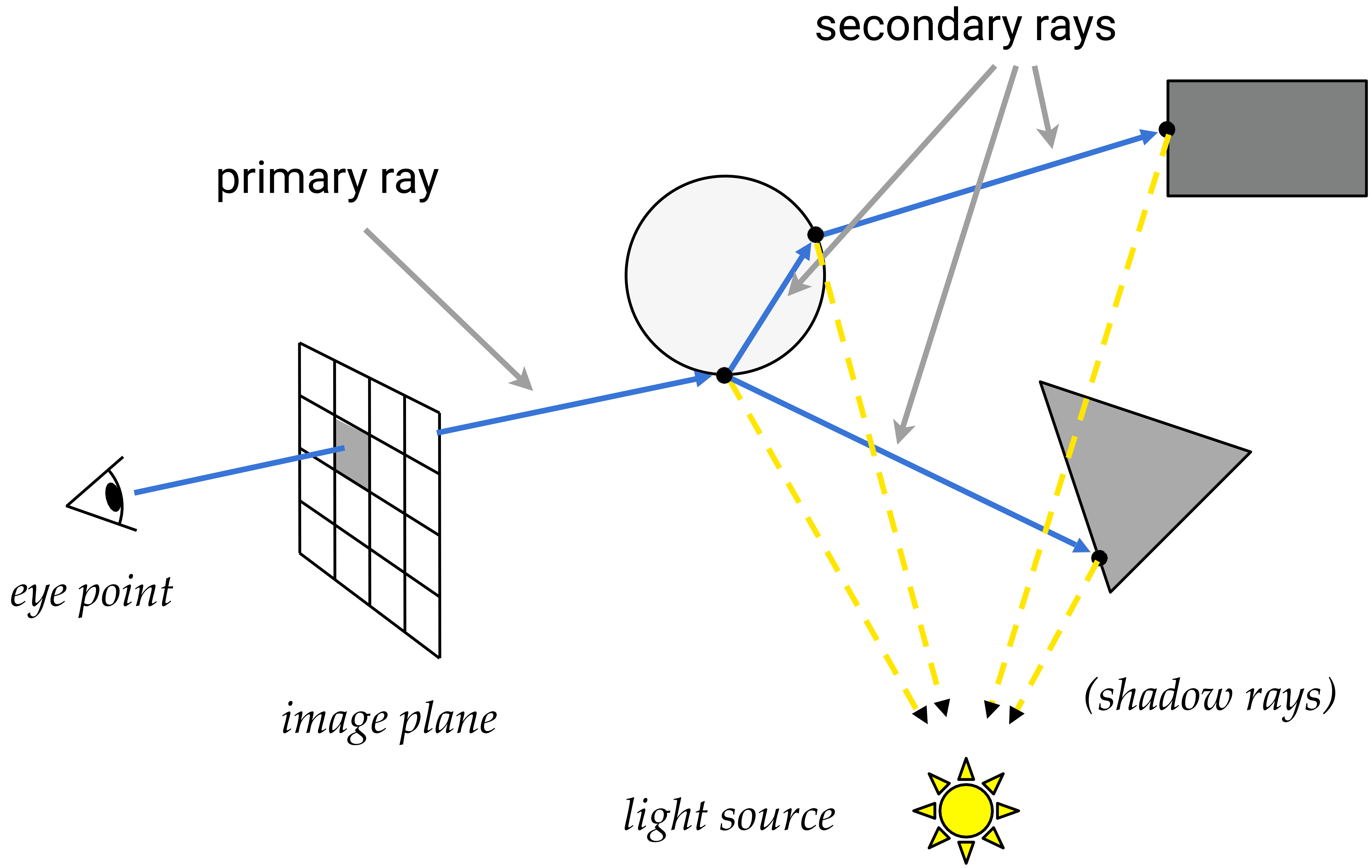
Recursive Ray Tracing



Recursive Ray Tracing



Recursive Ray Tracing



- Trace secondary rays recursively until you hit a non-specular surface.
- Final pixel color is weighted sum of contributions along rays
- Results in more sophisticated effects (e.g. **specular reflection**, **refraction**, **shadows**)

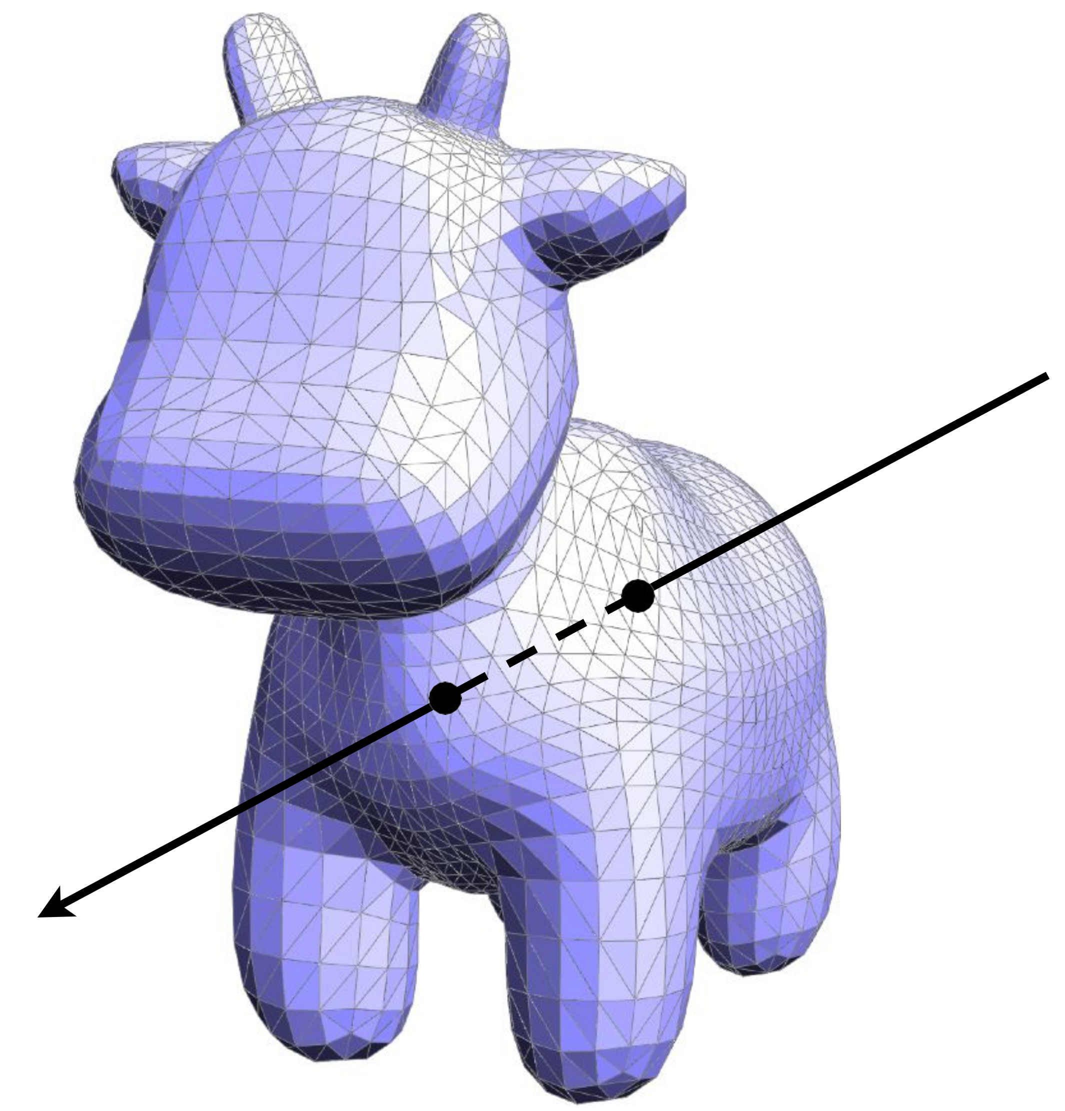
Ray-Surface Intersection

Ray Intersection With Mesh

Why?

- Rendering: visibility, shadows, lighting …
- Geometry: inside/outside test

How to compute?

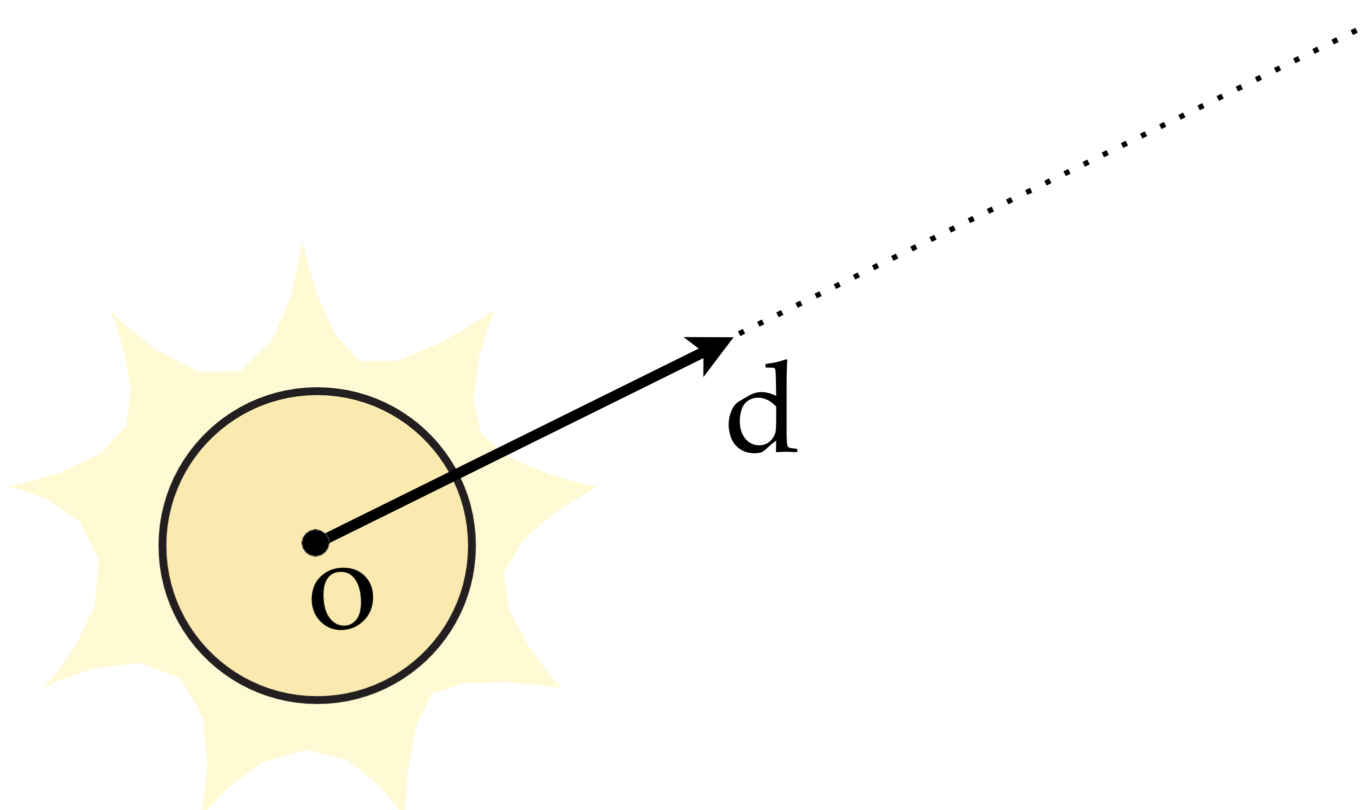


- Simple idea: just intersect ray with each triangle
- Simple, but slow *(implement acceleration later)*
- Note: A triangle can have 0, 1 or multiple intersections

Ray Equation

Ray is defined by its origin and a direction vector

Example:



Ray equation:

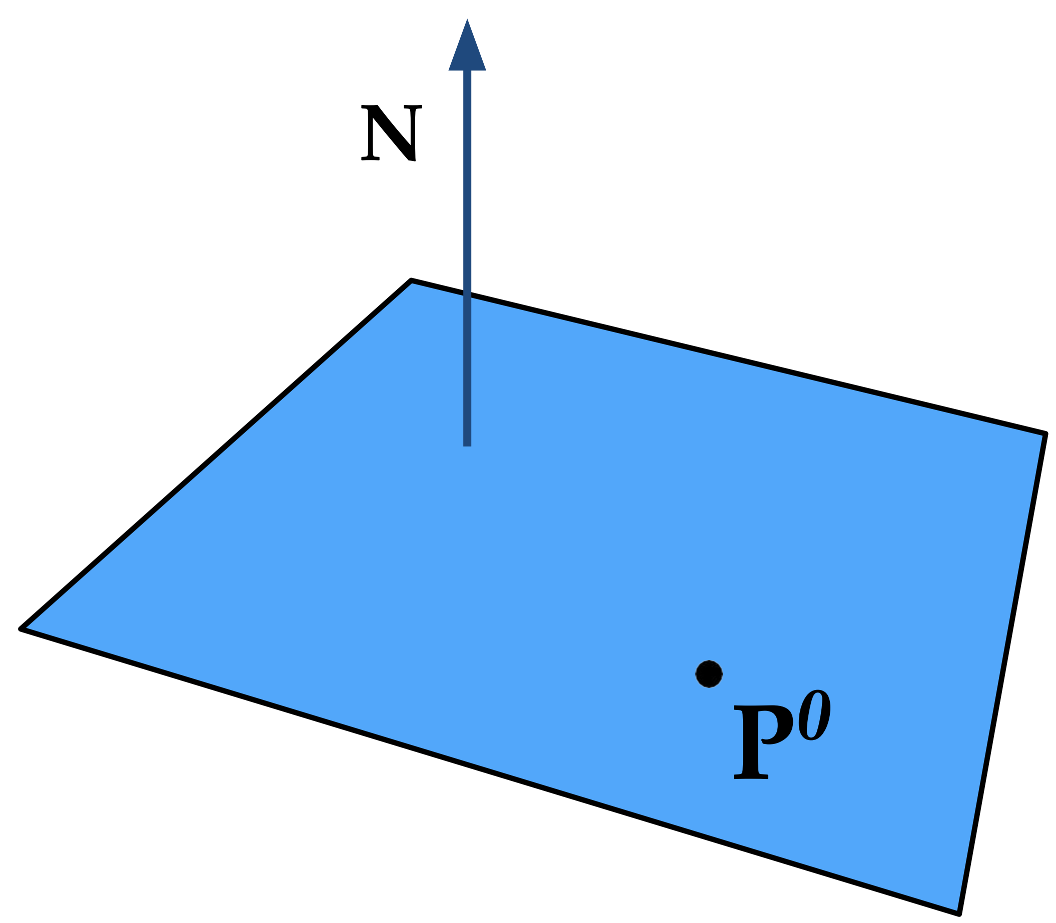
$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d} \qquad 0 \leq t < \infty$$

point along ray *“time”* origin unit direction

Plane Equation

A plane is defined by a normal vector and a point on the plane

Example:



Plane Equation:

$$\mathbf{p} : (\mathbf{p} - \mathbf{p}^0) \cdot \mathbf{N} = 0$$

all points on plane **point on plane** **normal vector**

$$ax + by + cz + d = 0$$

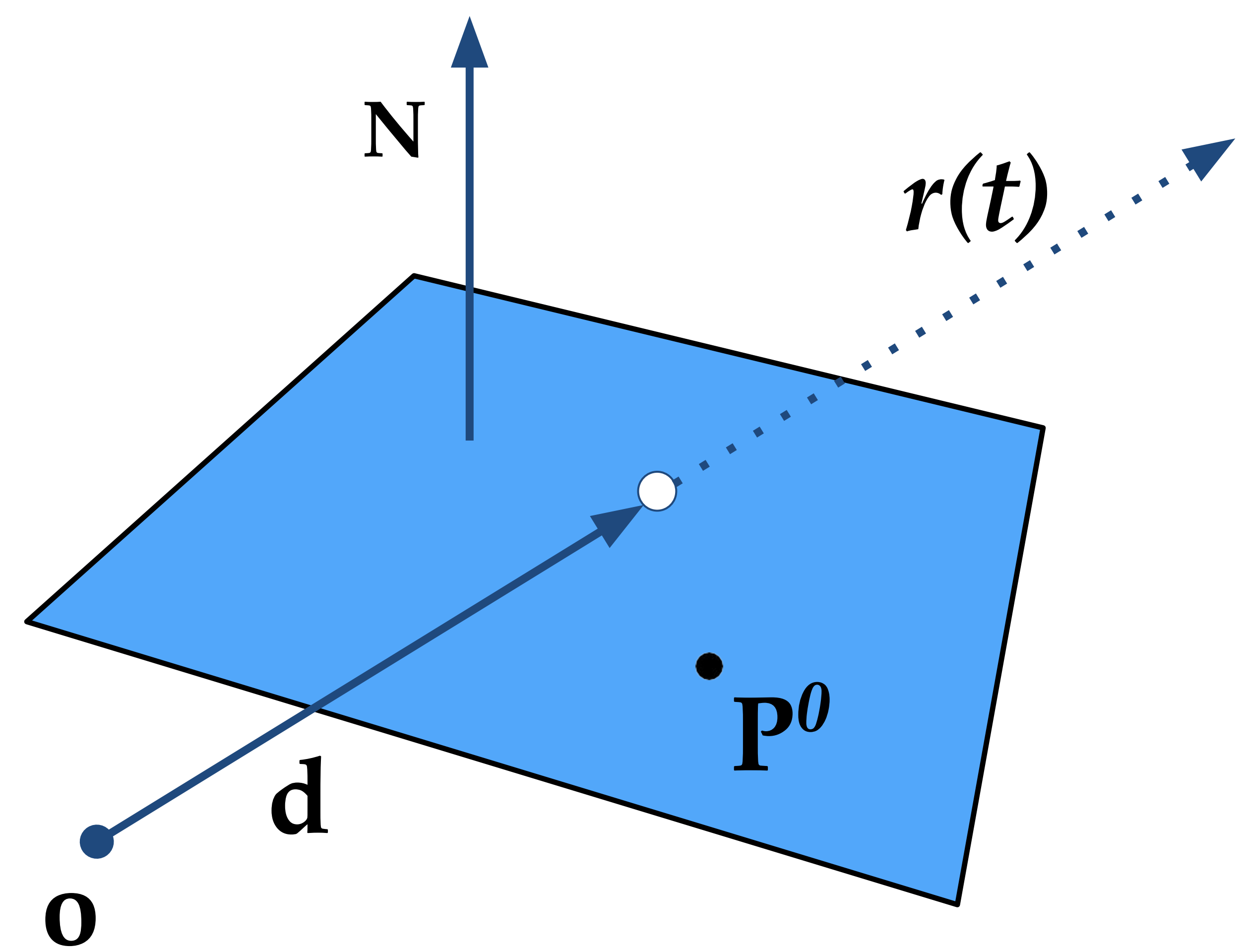
Ray Intersection With Plane

Ray equation:

$$\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d},\ 0 \leq t < \infty$$

Plane equation:

$$\mathbf{p} : (\mathbf{p} - \mathbf{p}^{0}) \cdot \mathbf{N} = 0$$



Solve for intersection

Set $\mathbf{p} = \mathbf{r}(t)$ and solve for t

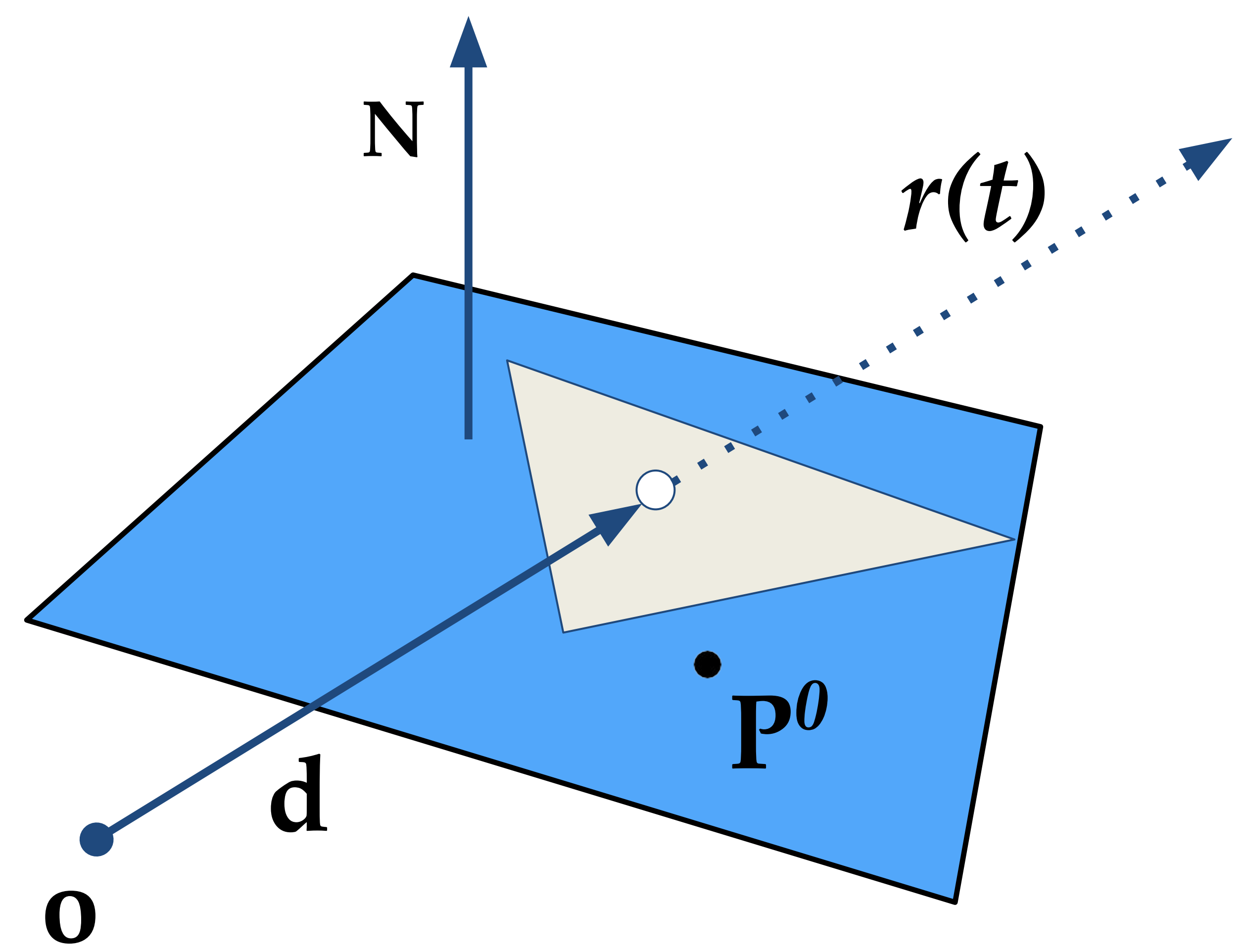
$$(\mathbf{p} - \mathbf{p}^{0}) \cdot \mathbf{N} = (\mathbf{o} + t\,\mathbf{d} - \mathbf{p}^{0}) \cdot \mathbf{N} = 0$$

$$\boldsymbol{t} = \frac{(\mathbf{p}^{0} - \mathbf{o}) \cdot \mathbf{N}}{\mathbf{d} \cdot \mathbf{N}}$$ Check: $0 \leq t < \infty$

Ray Intersection With Triangle

Triangle is in a plane

- Ray-plane intersection
- Test if hit point is inside triangle *(Assignment 1!)*



Many ways to optimize

One Optimization: e.g. Möller Trumbore Algorithm

$$\vec{\mathbf{O}} + t\vec{\mathbf{D}} = (1 - b_1 - b_2)\vec{\mathbf{P}}_0 + b_1\vec{\mathbf{P}}_1 + b_2\vec{\mathbf{P}}_2$$

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\vec{\mathbf{S}}_1 \bullet \vec{\mathbf{E}}_1} \begin{bmatrix} \vec{\mathbf{S}}_2 \bullet \vec{\mathbf{E}}_2 \\ \vec{\mathbf{S}}_1 \bullet \vec{\mathbf{S}} \\ \vec{\mathbf{S}}_2 \bullet \vec{\mathbf{D}} \end{bmatrix}$$

Where:

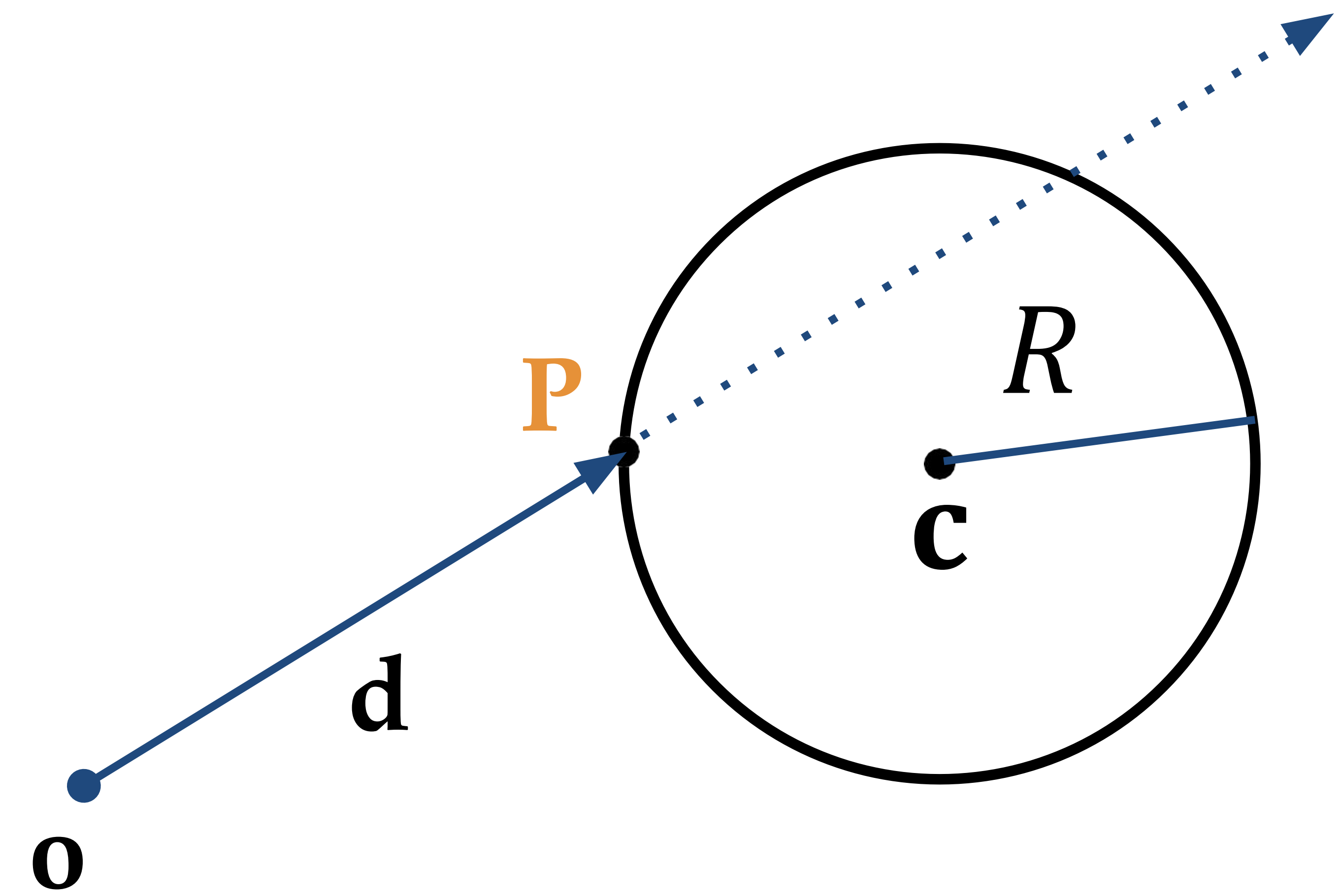
$$\vec{\mathbf{E}}_1 = \vec{\mathbf{P}}_1 - \vec{\mathbf{P}}_0$$
$$\vec{\mathbf{E}}_2 = \vec{\mathbf{P}}_2 - \vec{\mathbf{P}}_0$$
$$\vec{\mathbf{S}} = \vec{\mathbf{O}} - \vec{\mathbf{P}}_0$$
$$\vec{\mathbf{S}}_1 = \vec{\mathbf{D}} \times \vec{\mathbf{E}}_2$$
$$\vec{\mathbf{S}}_2 = \vec{\mathbf{S}} \times \vec{\mathbf{E}}_1$$

Cost = (1 div, 27 mul, 17 add)

Ray Intersection With Sphere

Ray: $\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d},\ \ 0 \leq t < \infty$

Sphere: $\mathbf{p} : (\mathbf{p} - \mathbf{c})^2 - R^2 = 0$



Solve for intersection:

$$(\mathbf{o} + t\,\mathbf{d} - \mathbf{c})^2 - R^2 = 0$$

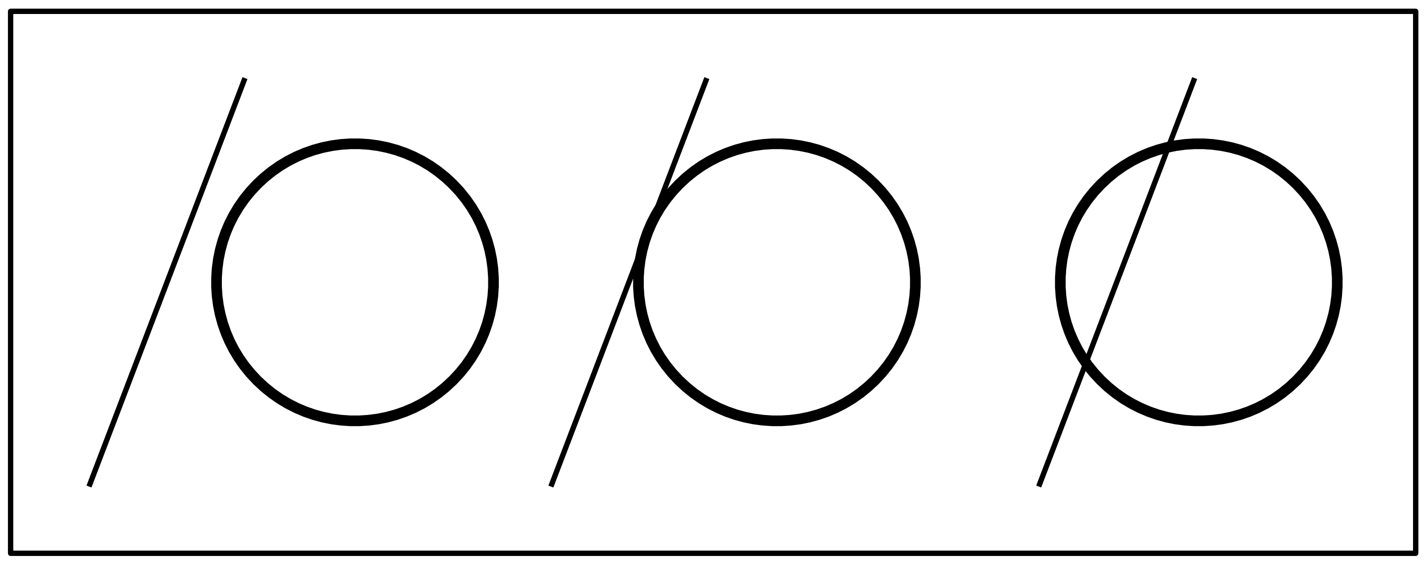
$$at^2 + bt + c = 0, \text{ where}$$

$$a = \mathbf{d} \cdot \mathbf{d}$$

$$b = 2(\mathbf{o} - \mathbf{c}) \cdot \mathbf{d}$$

$$c = (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2$$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



Ray Intersection With Implicit Surface

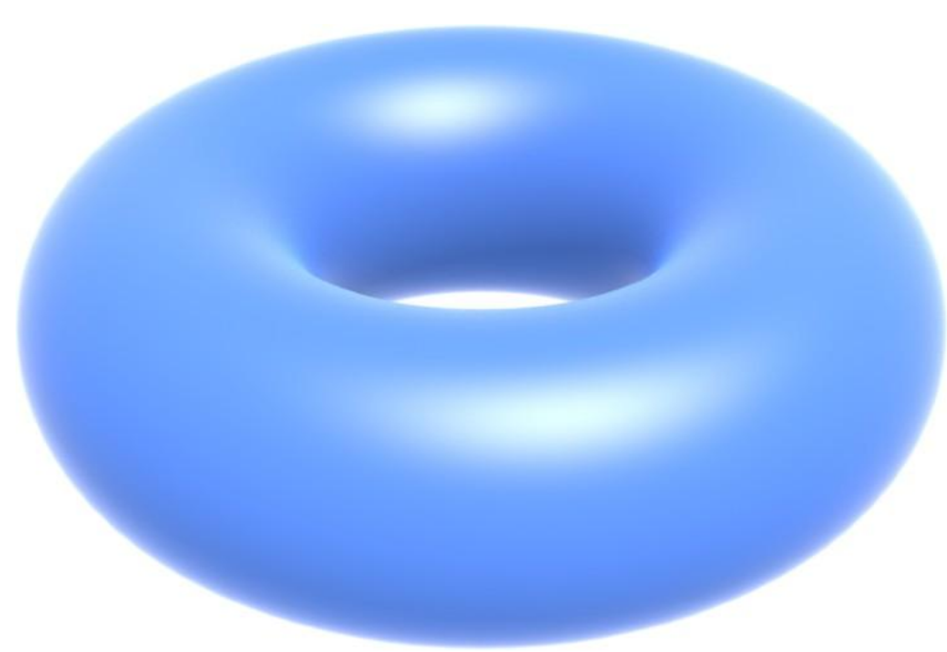
Ray: $\mathbf{r}(t) = \mathbf{o} + t\ \mathbf{d},\ \ 0 \leq\ t < \infty$

General implicit surface: $\mathbf{p} : f\,(\mathbf{p}) = 0$

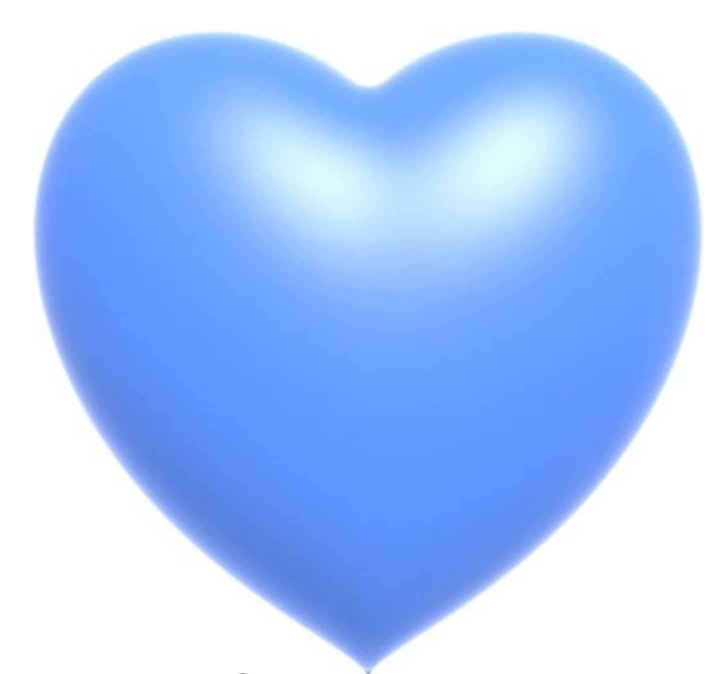
Substitute ray equation: $f\,(\mathbf{o} + t\ \mathbf{d}) = 0$

Solve for real, positive roots

$x^2 + y^2 + z^2 = 1$



$(R - \sqrt{x^2 + y^2})^2 + z^2 = r^2$



$= \ (x^2 + \frac{9y^2}{4} + z^2 - 1)^3$

$x^2 z^3 + \frac{9y^2 z^3}{80}$

Accelerating Ray-Surface Intersection

Ray Tracing – Performance Challenges

Simple ray-scene intersection

- Exhaustively test ray-intersection with every object

Problem:

- Exhaustive algorithm = $(x \cdot y)$ pixels $\times$ objects
- Very slow!

Ray Tracing – Performance Challenges

San Miguel Scene, 10.7M triangles

Ray Tracing – Performance Challenges

Plant Ecosystem, 20M triangles

Bounding Volumes

Bounding Volumes

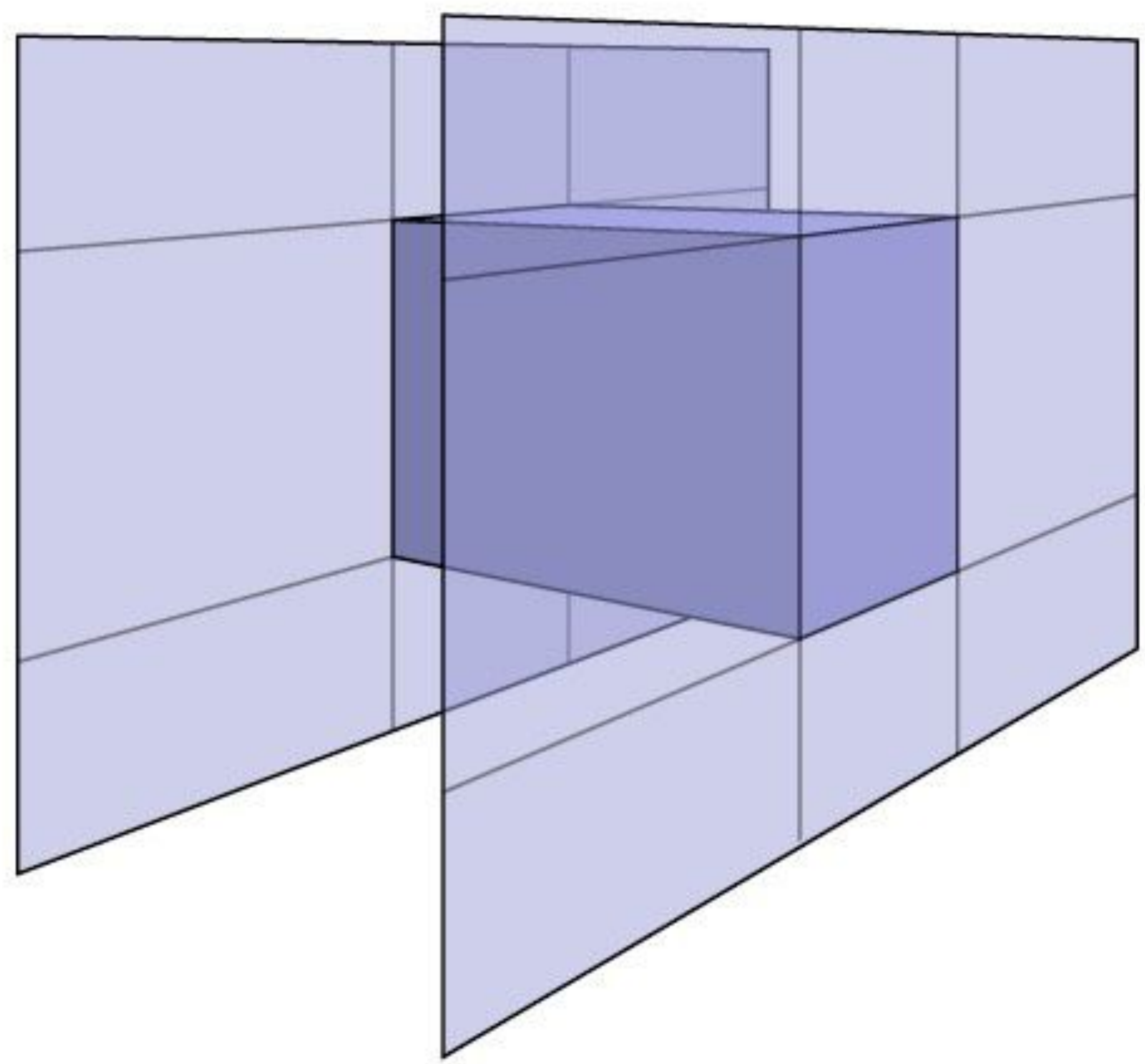
Quick way to avoid intersections: bound complex object with a simple volume

- 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

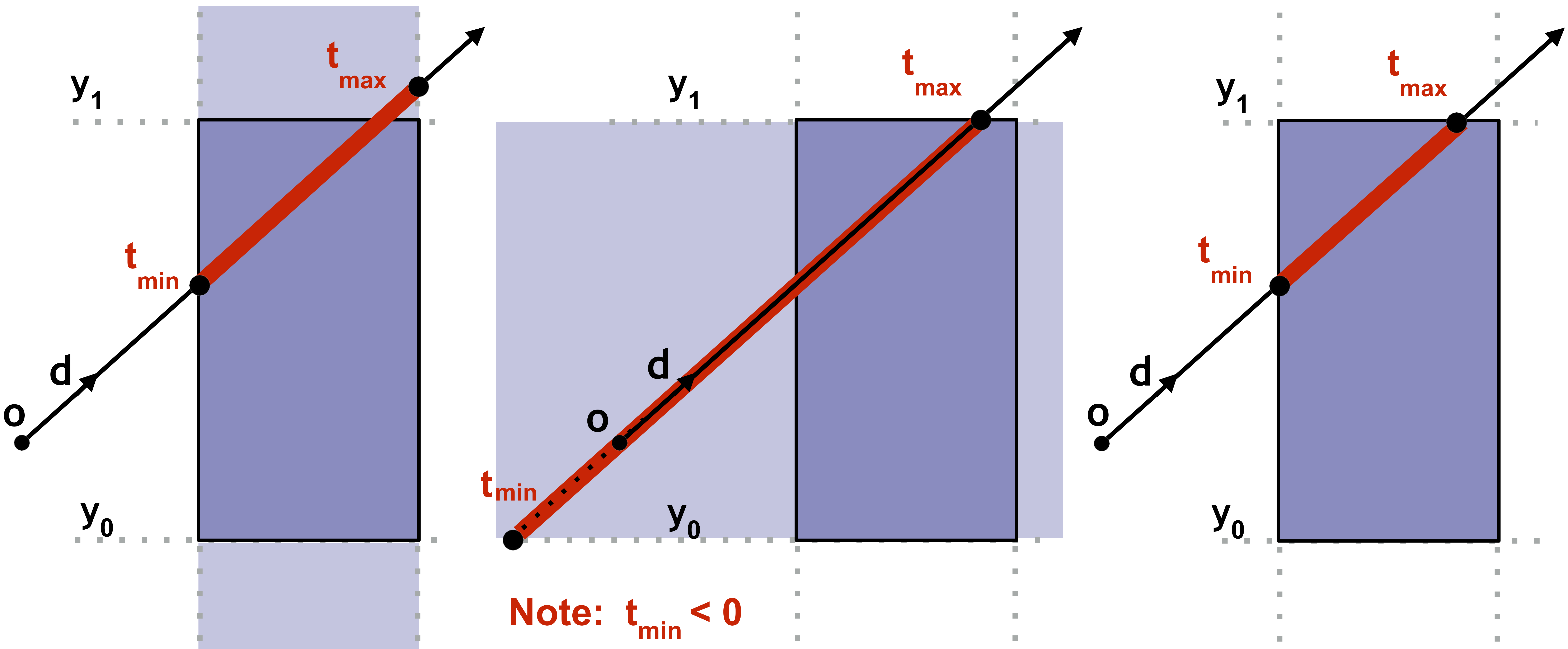
Could intersect with 6 faces individually Better way:

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 the two t_{min} and t_{max} intervals**

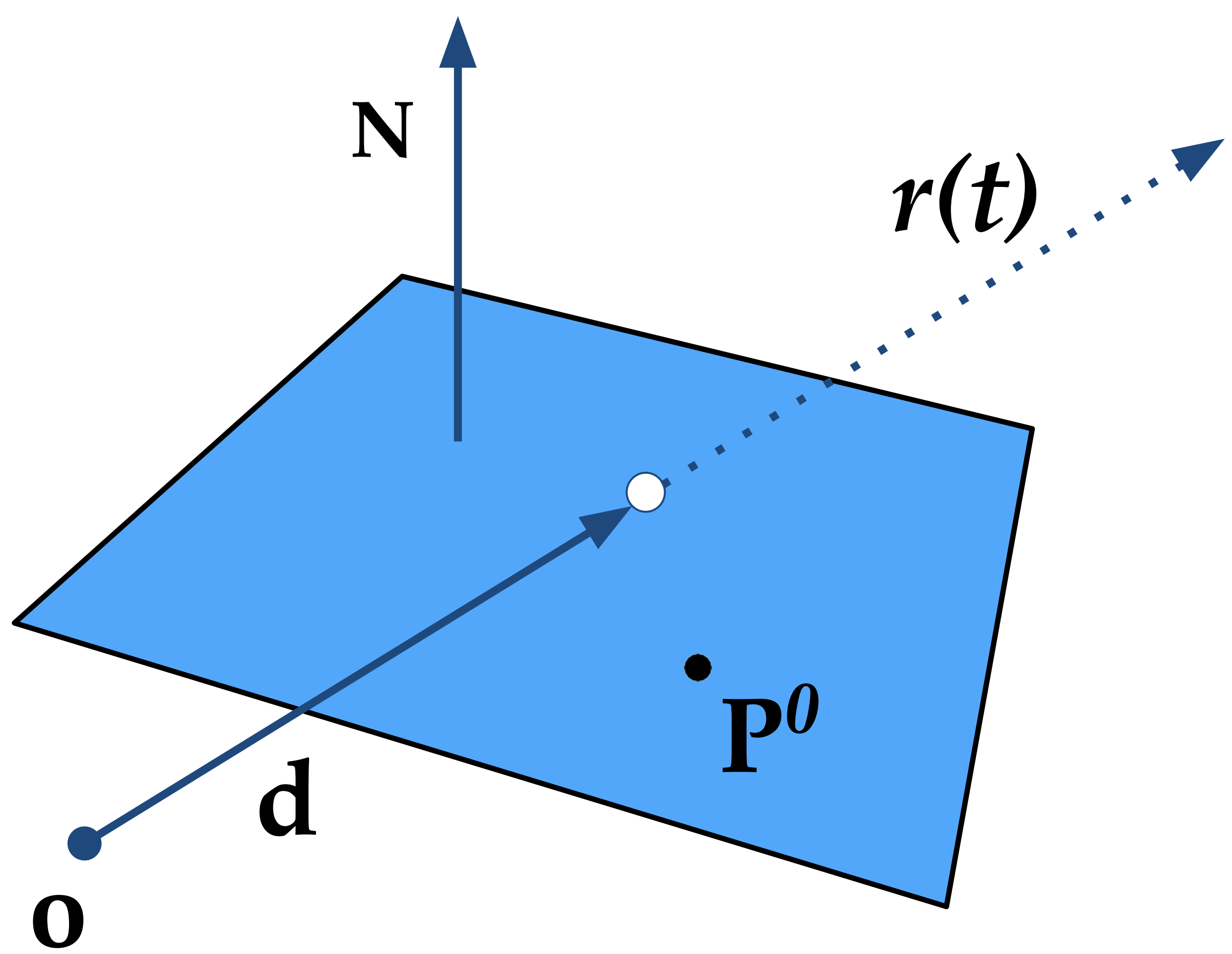


Intersections with *x* plane Intersections with *y* plane Final intersection result

How do we know when the ray misses the box?

Optimize Ray-Plane Intersection For Axis-Aligned Planes?

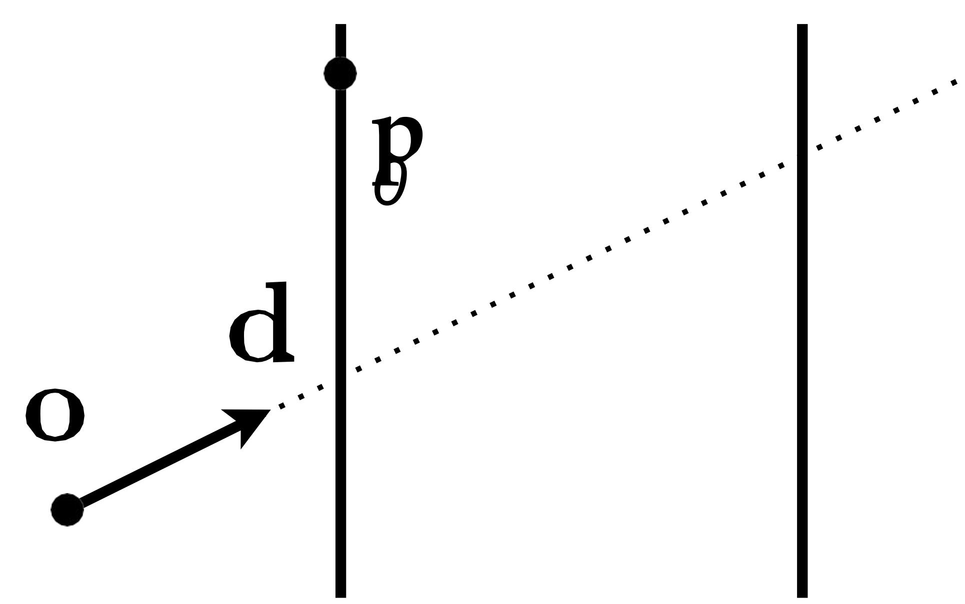
General



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

3 subtractions, 6 multiplies, 1 division

Perpendicular to x-axis

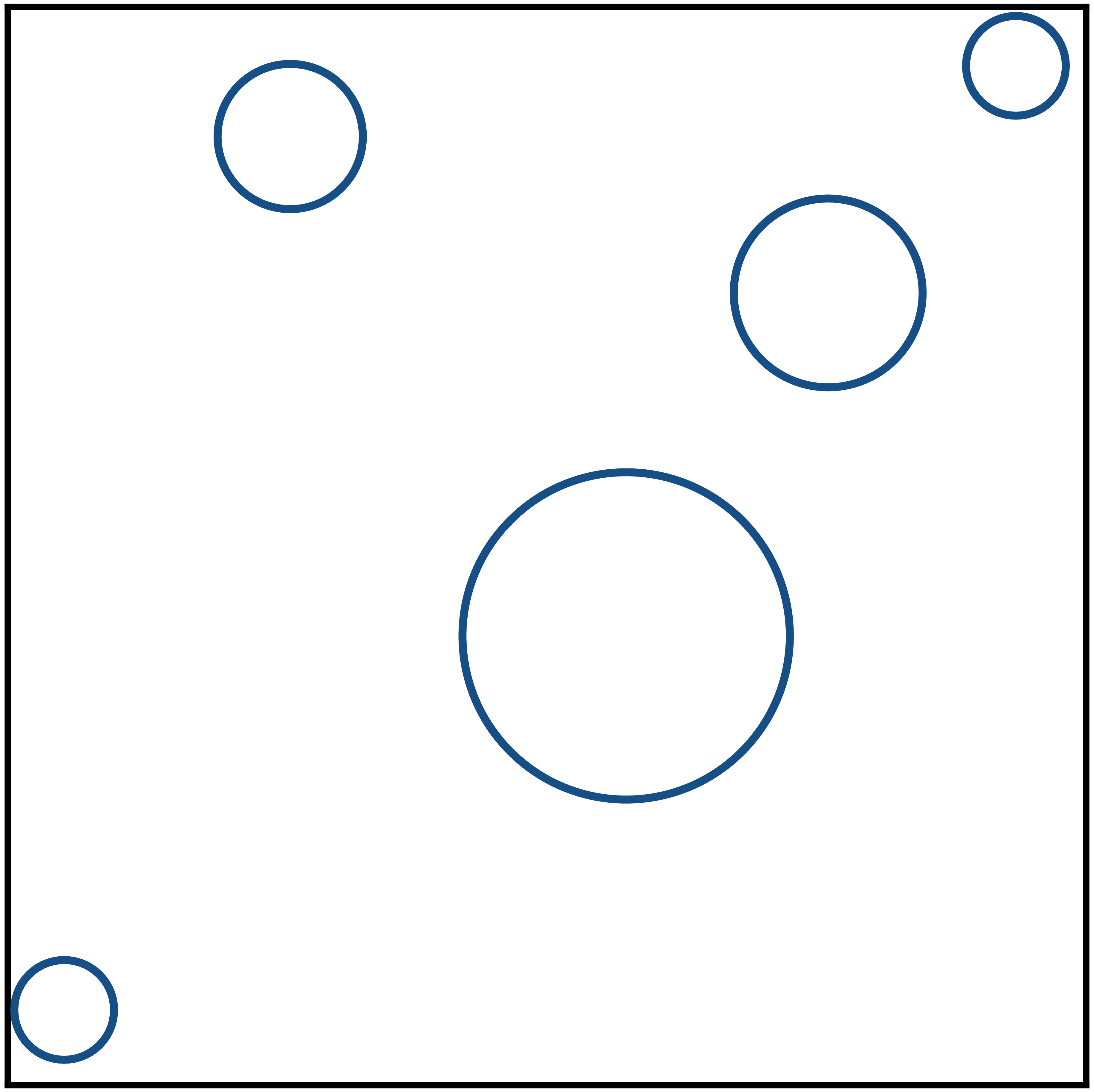


$$t = \frac{\mathbf{p}^0{}_x - \mathbf{o}_x}{\mathbf{d}_x}$$

1 subtraction, 1 division

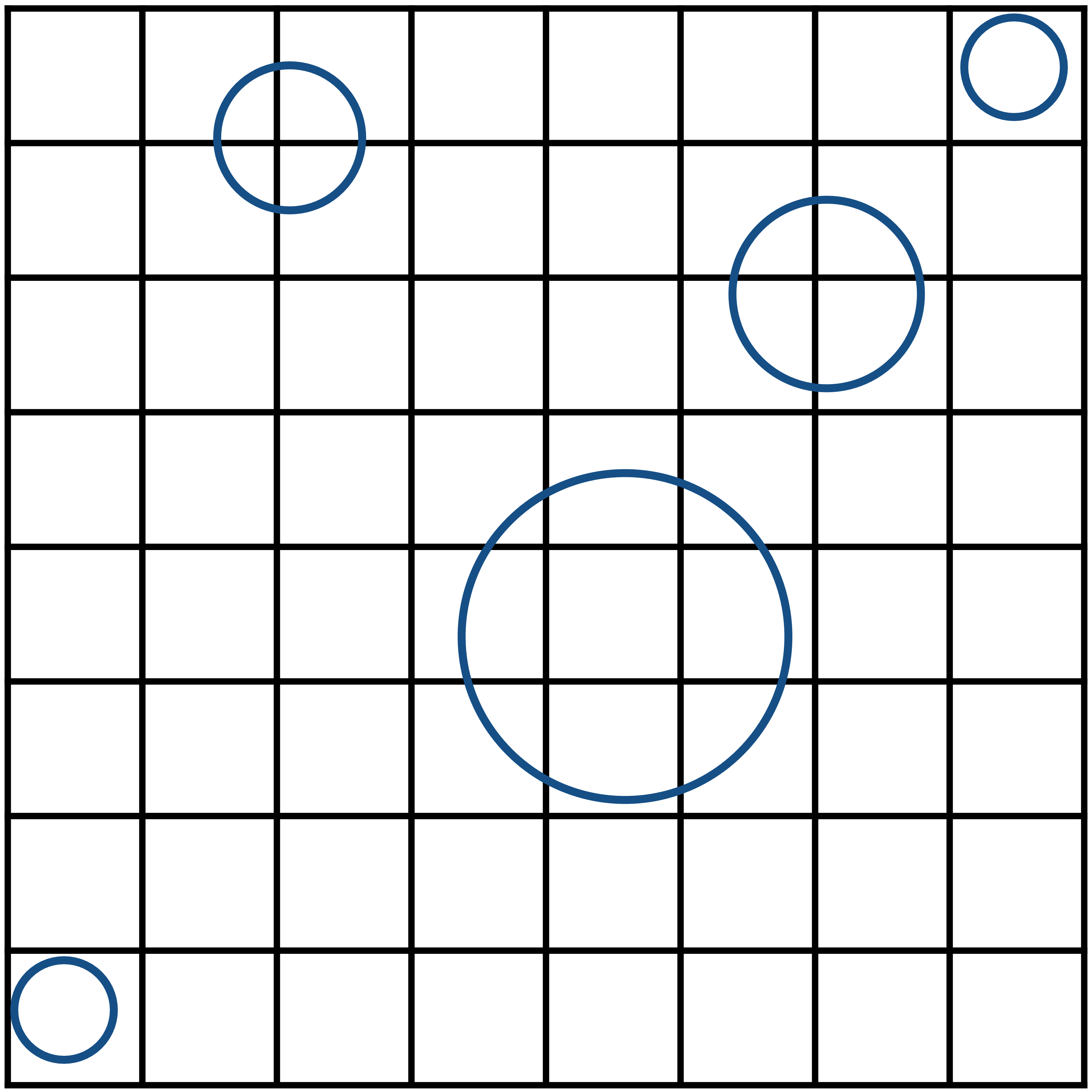
Uniform Spatial Partitions (Grids)

Preprocess – Build Acceleration Grid



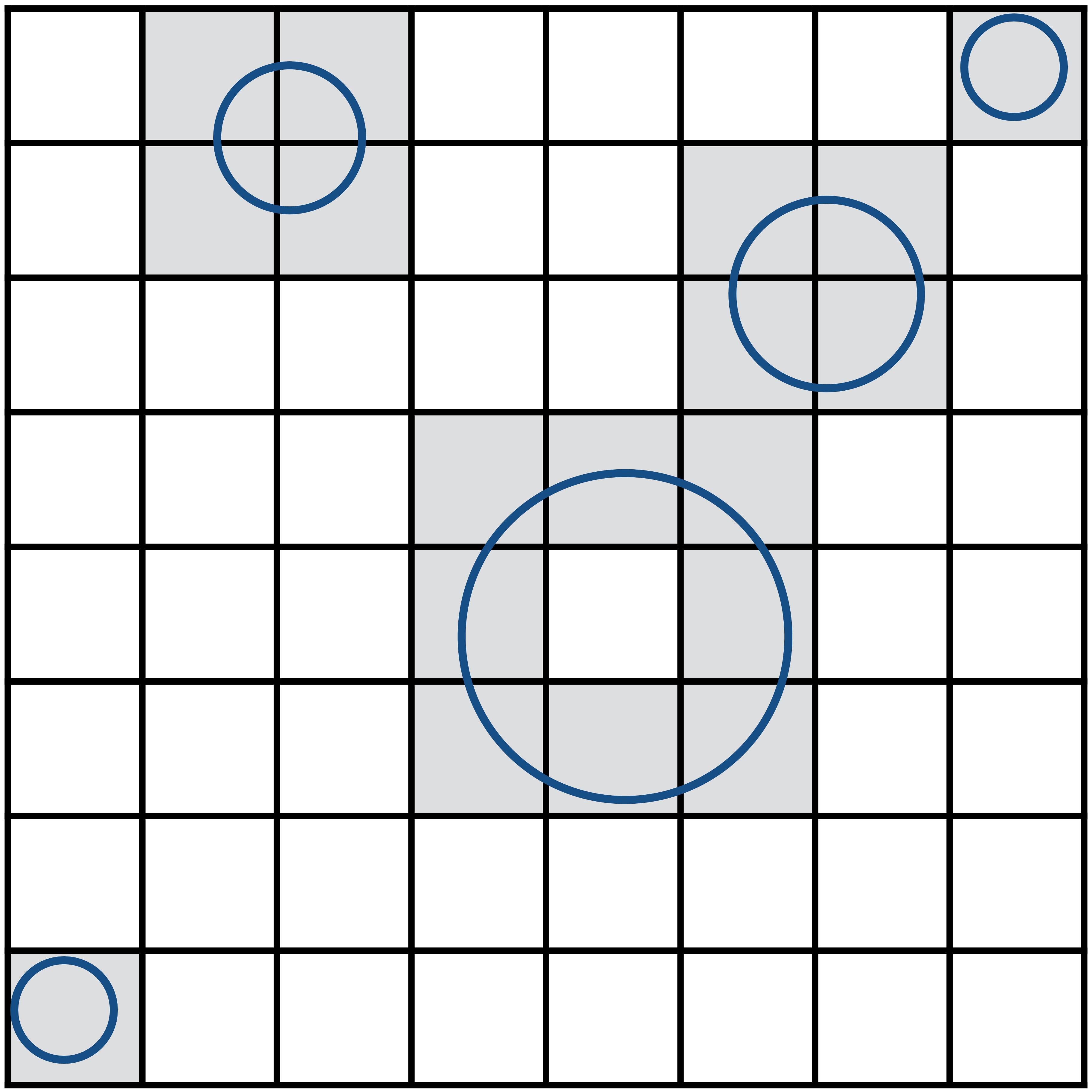
1. Find bounding box

Preprocess – Build Acceleration Grid



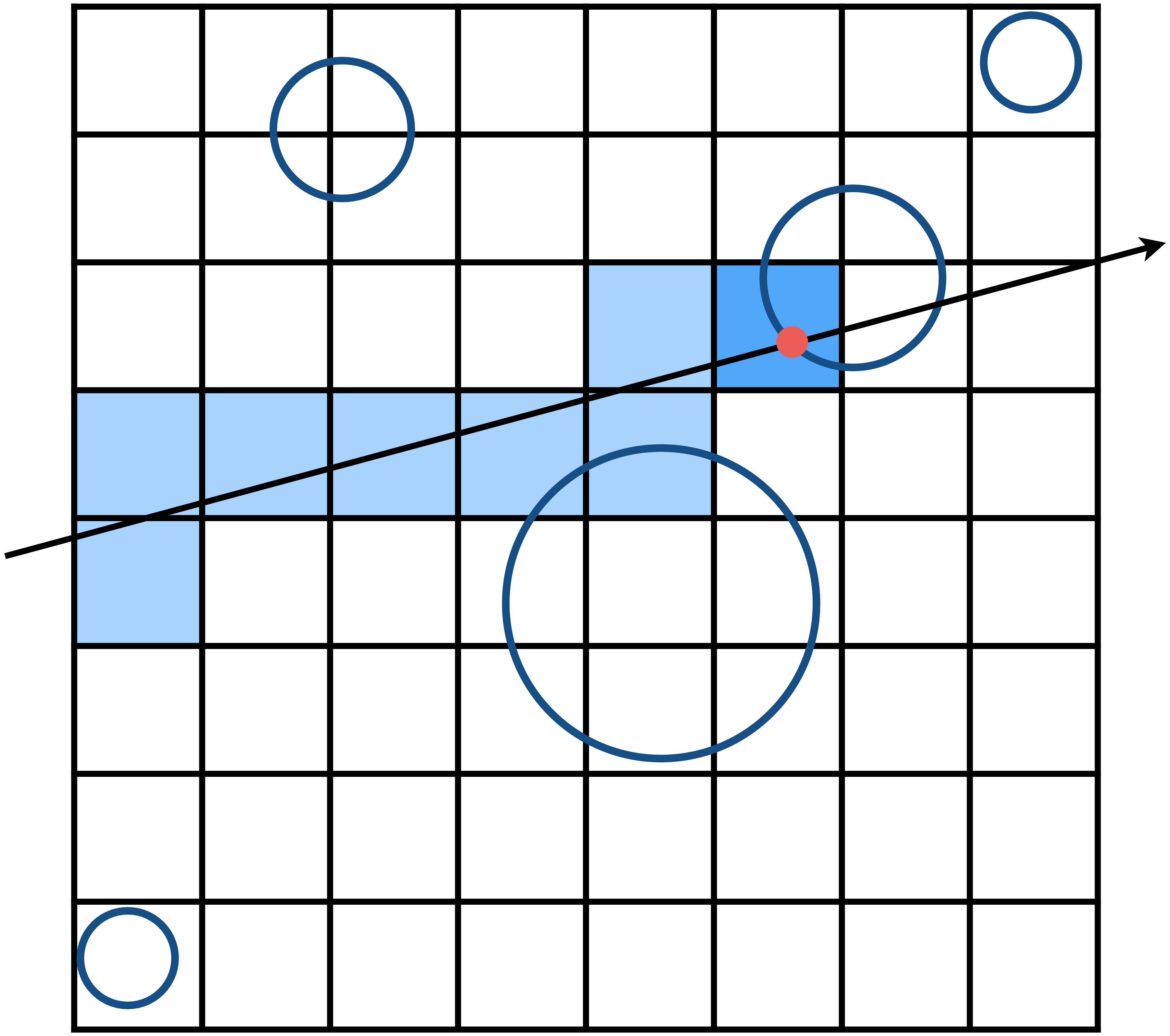
1. Find bounding box
2. Create grid

Preprocess – Build Acceleration Grid



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

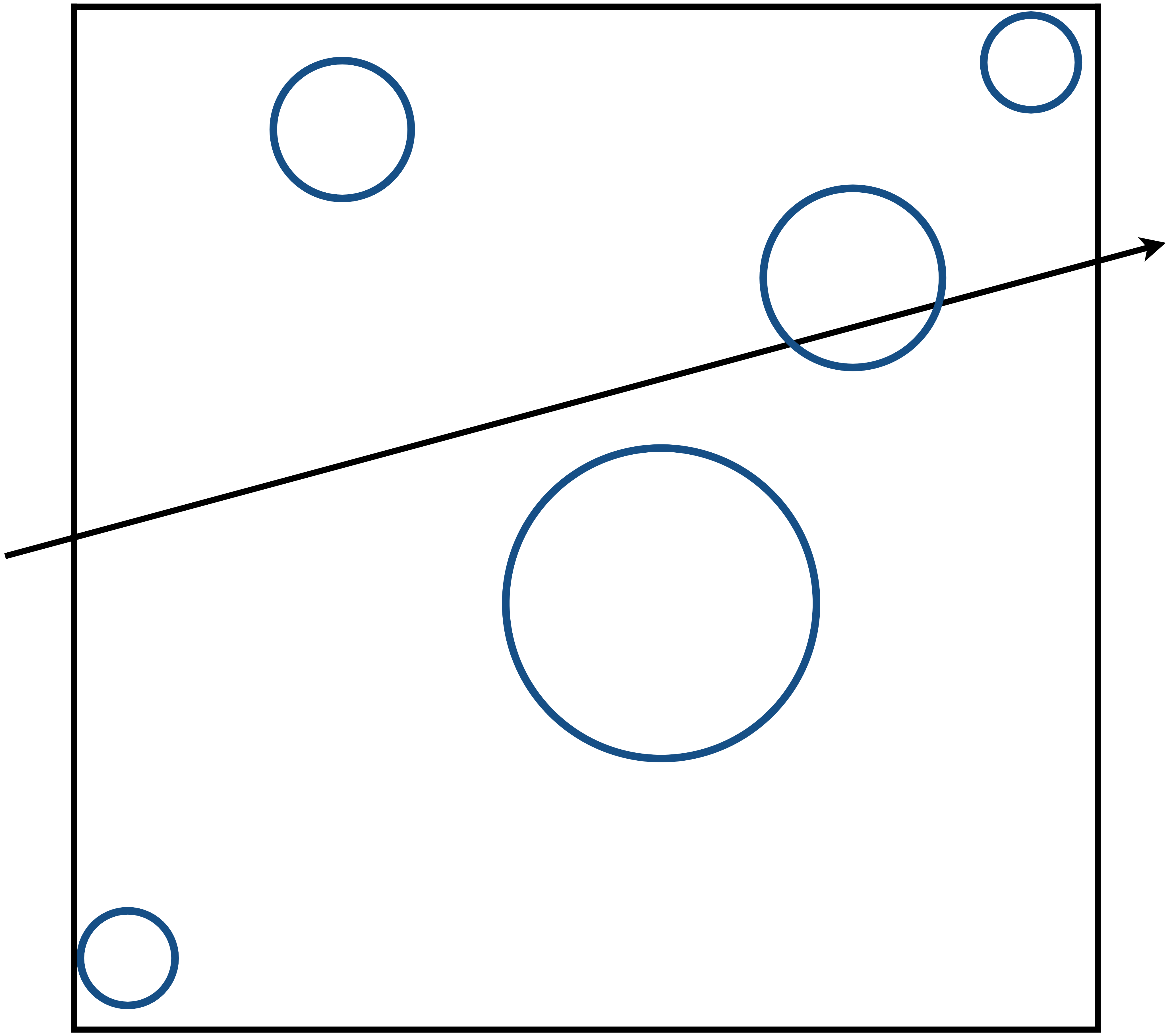
Ray-Scene Intersection



Step through grid in ray traversal order

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

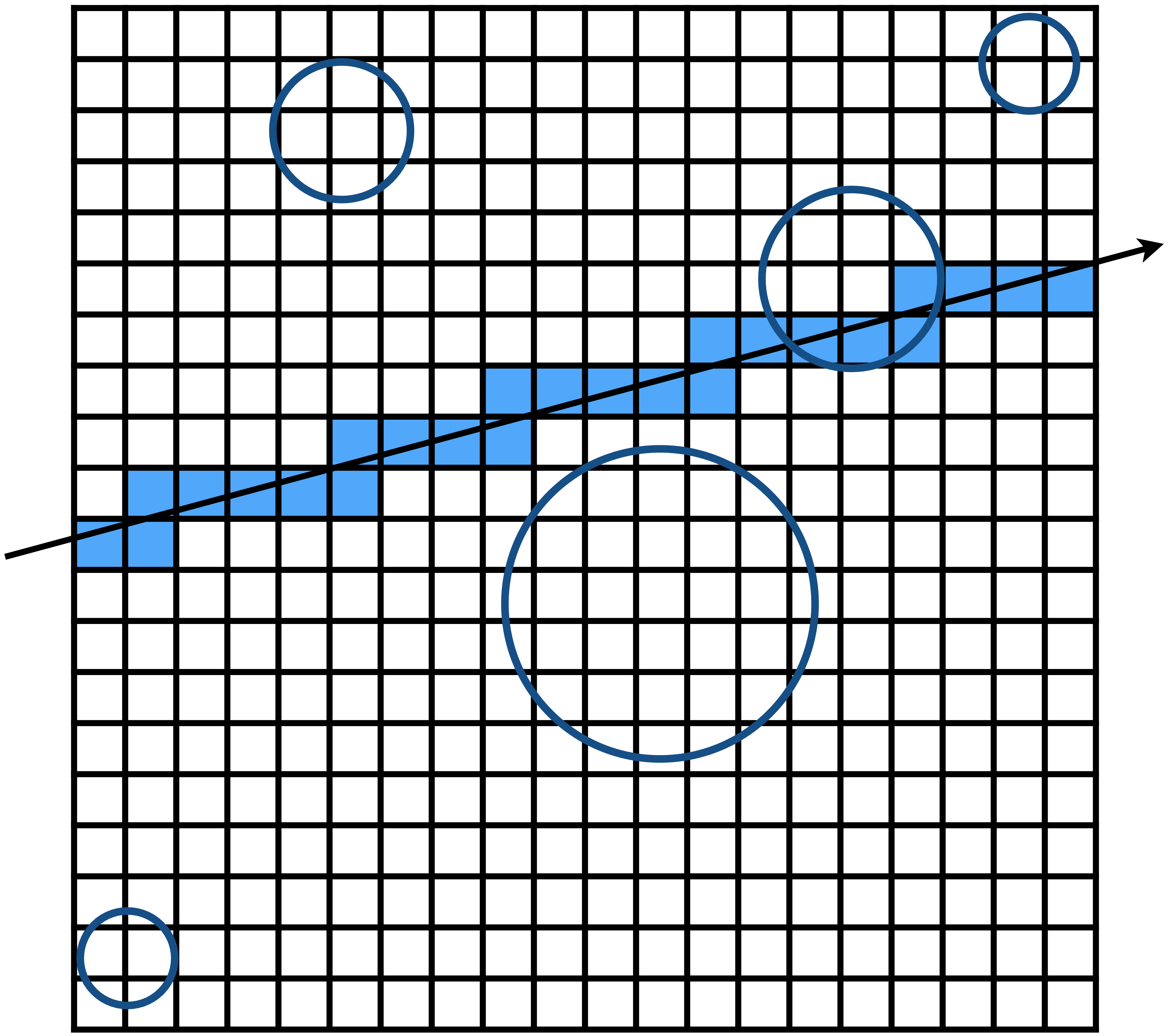
Grid Resolution?



One big cell

- No speedup

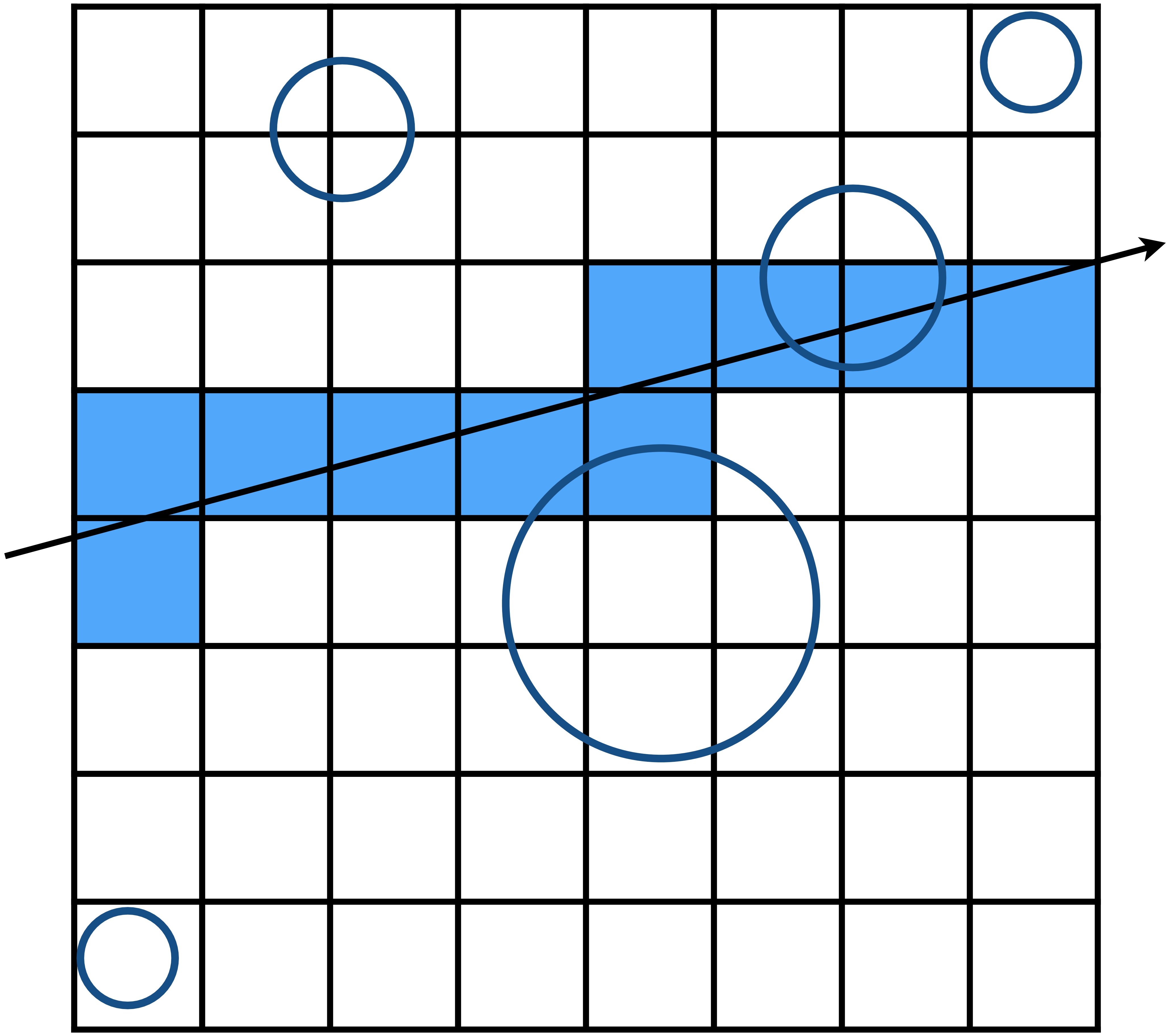
Grid Resolution?



Too many cells

- Inefficiency due to extraneous grid traversal

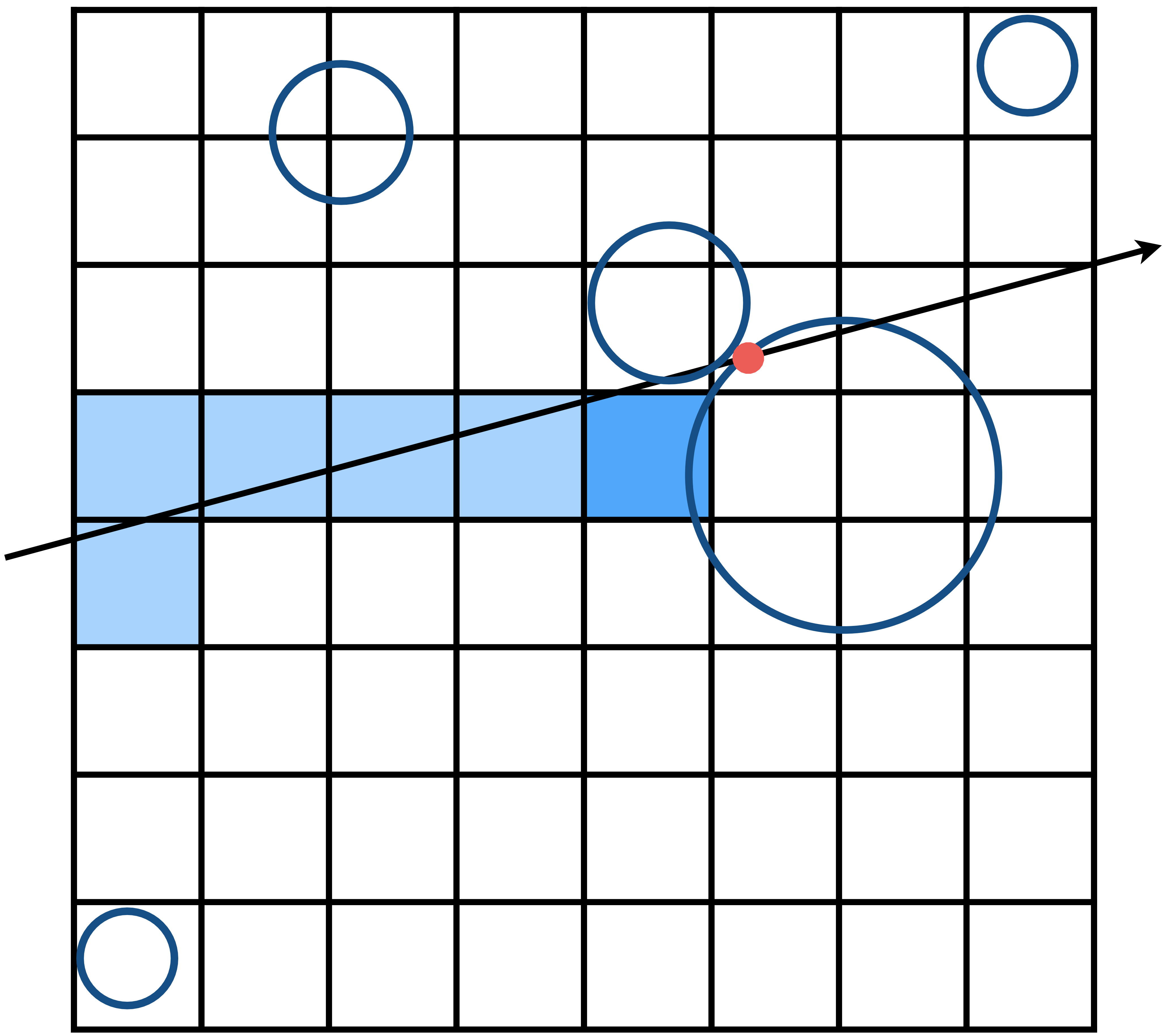
Grid Resolution?



Heuristic:

- $cells = C * n_{objs}$
- $C \approx 27$ 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

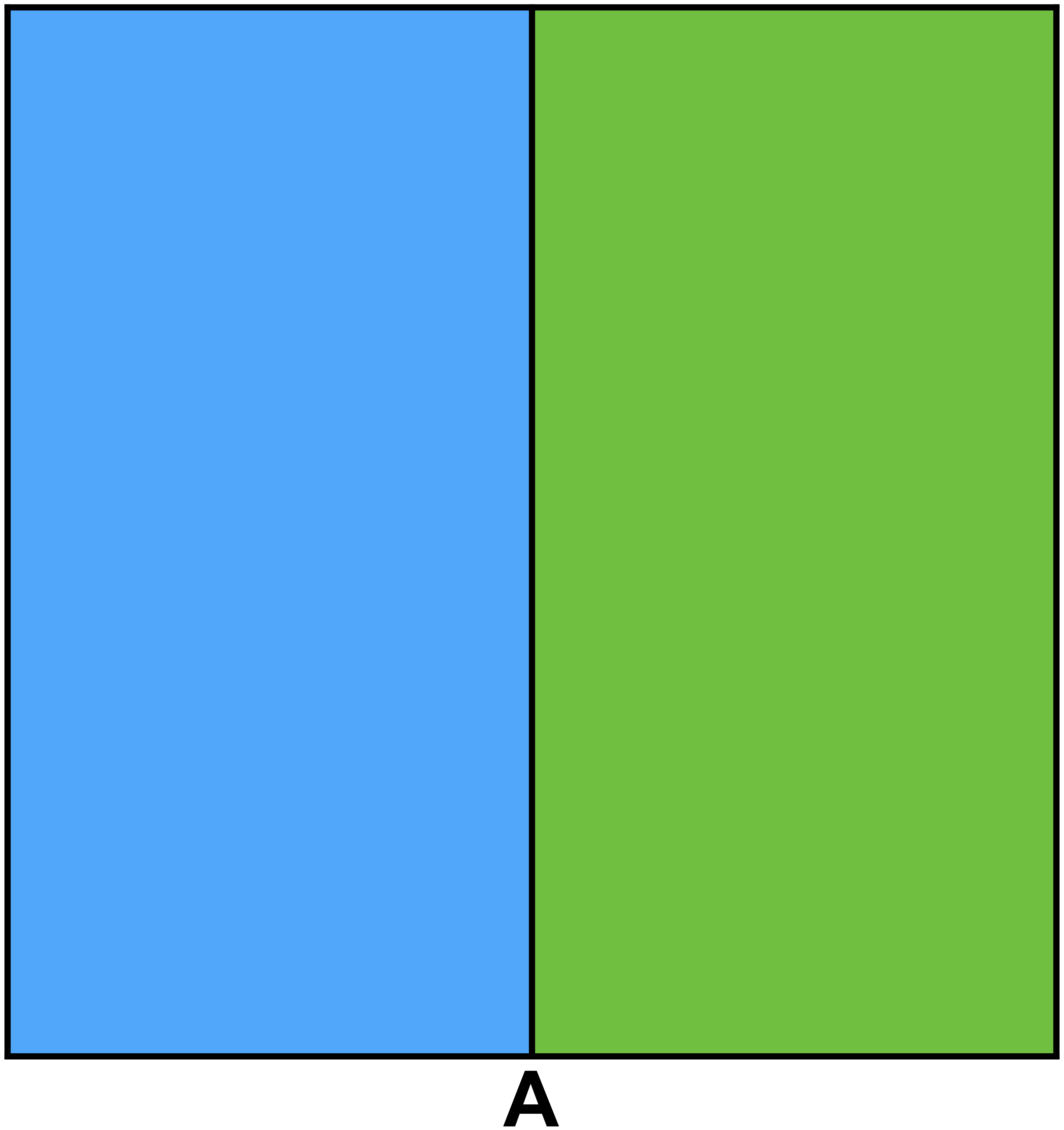
Grids work well on large collections of objects that are distributed evenly in size and space.

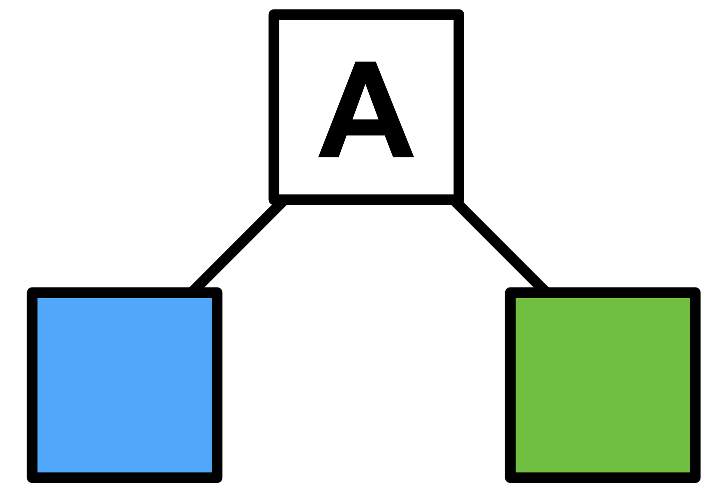
Uniform Grids – When They Fail

"Ball in a stadium" problem

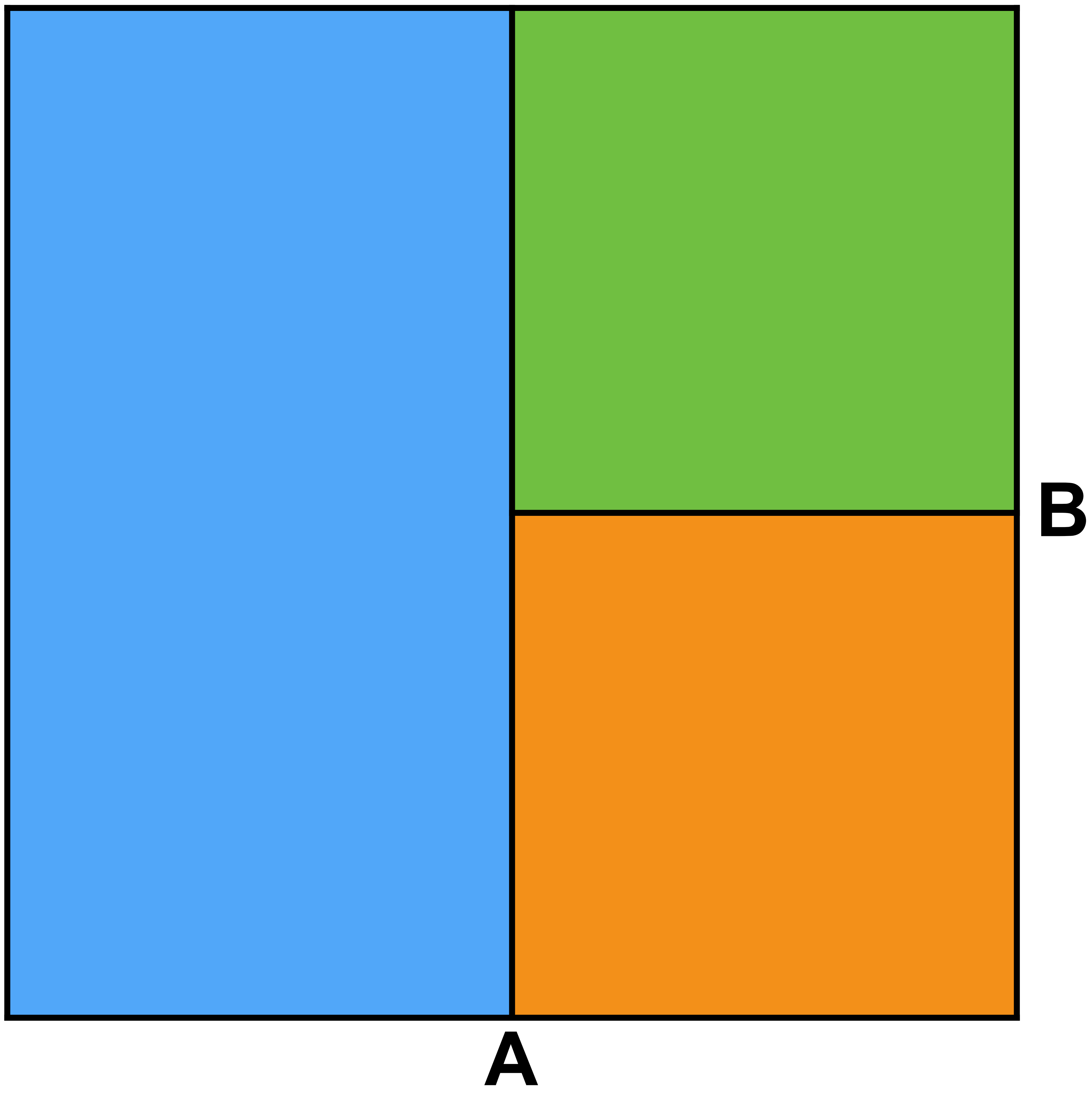
Non-Uniform Spatial Partitions:
Spatial Hierarchies

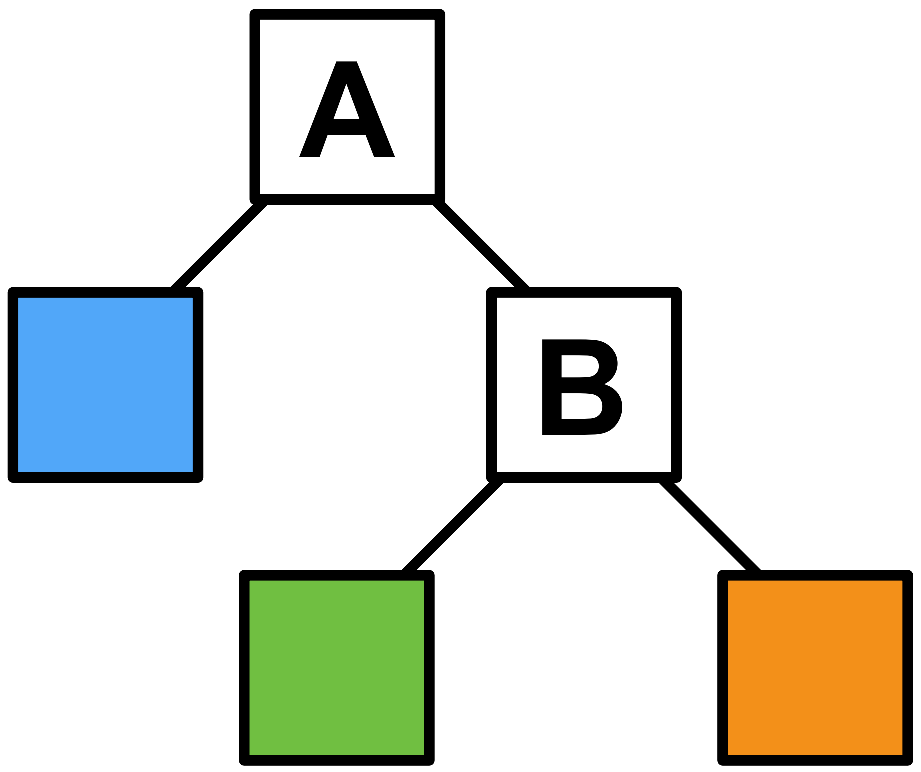
Spatial Hierarchies



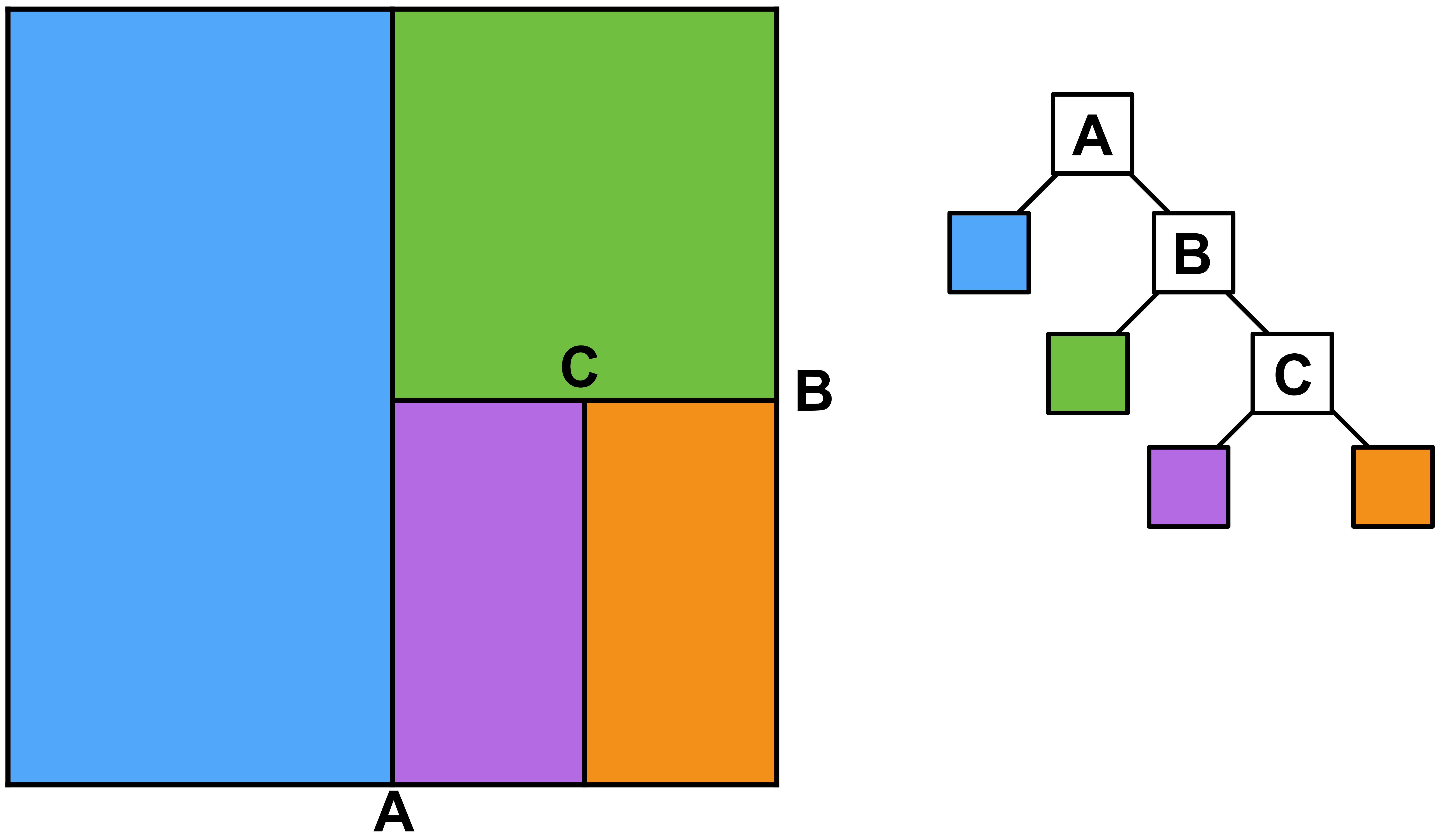


Spatial Hierarchies

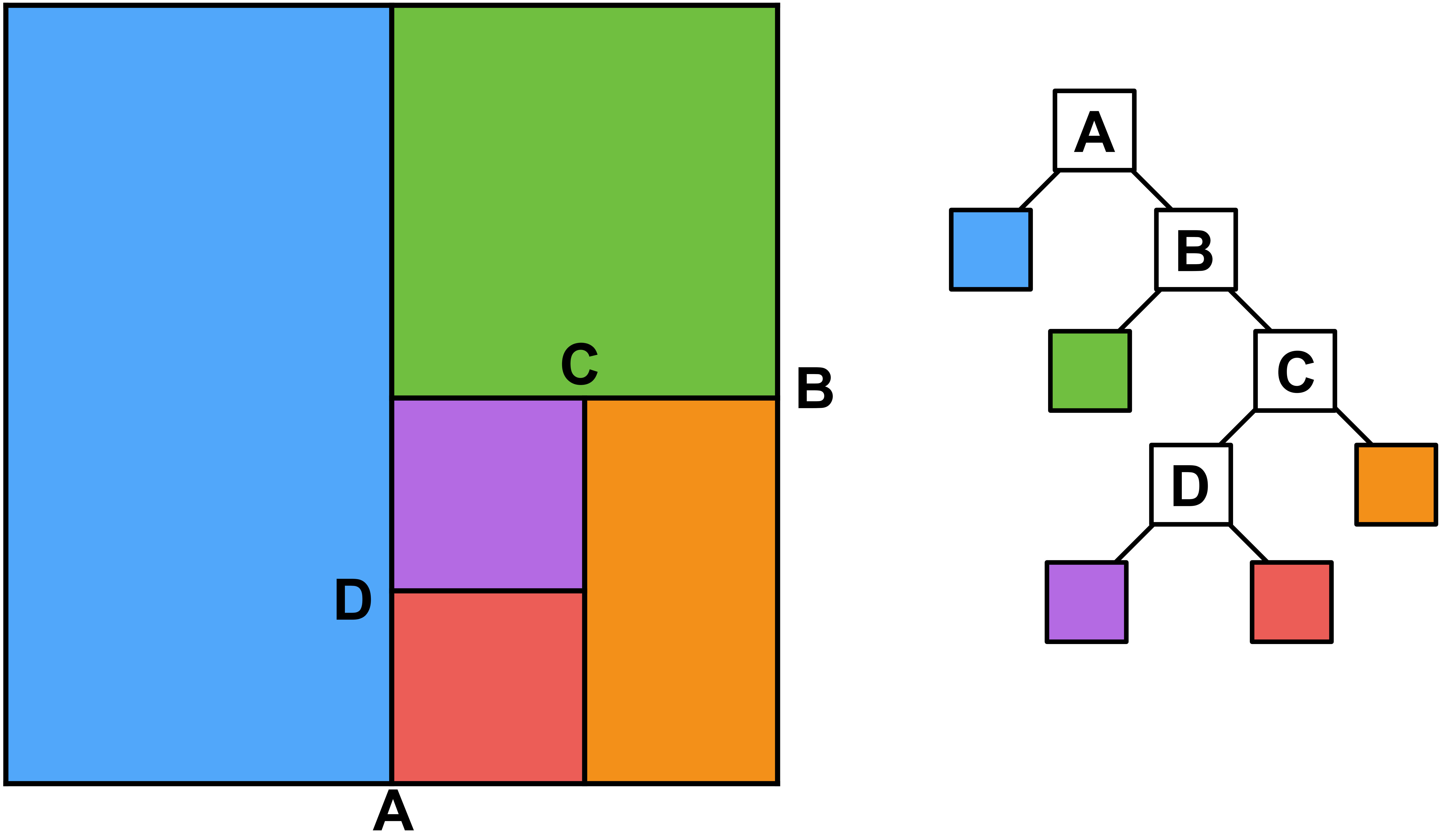




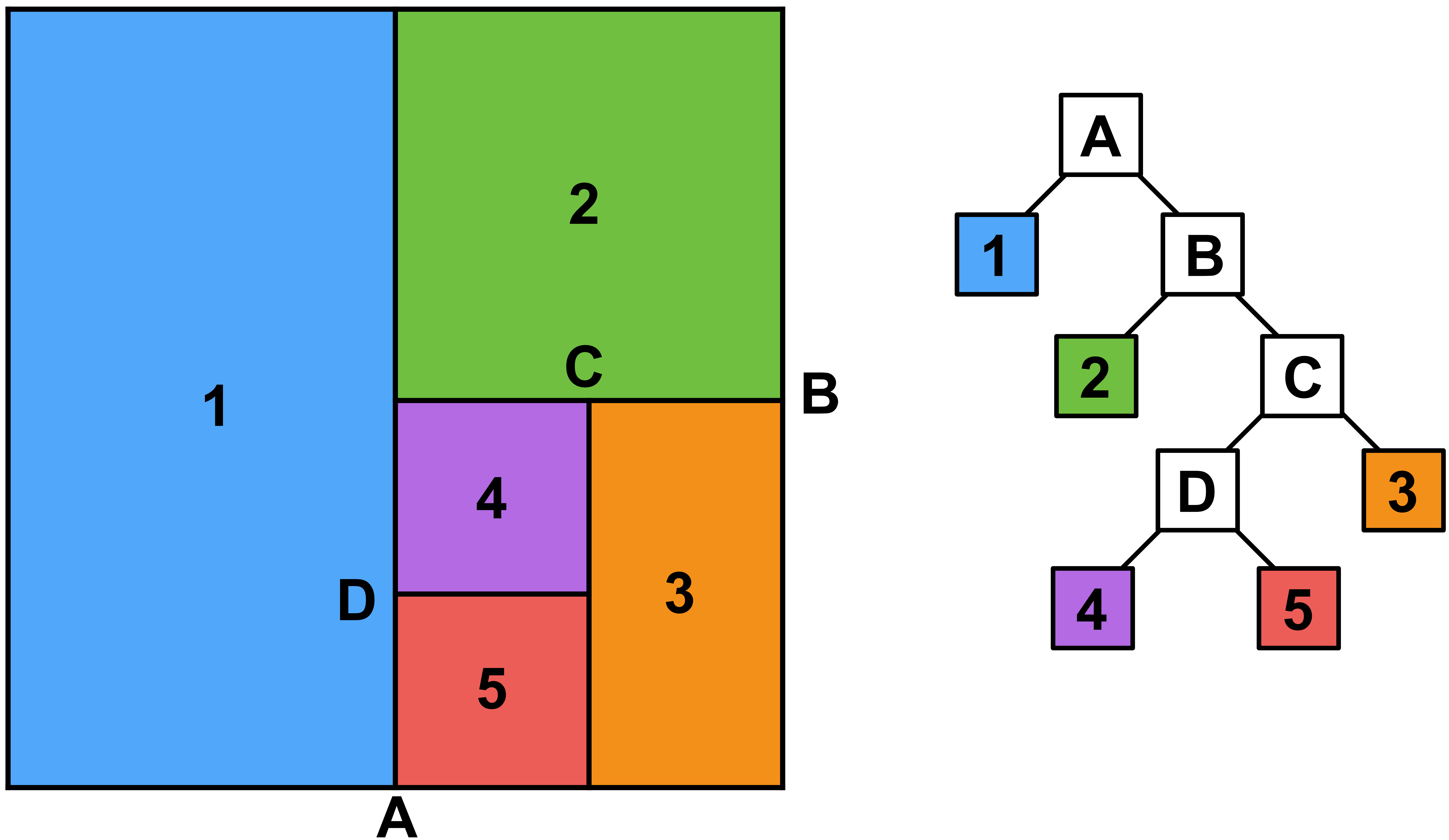
Spatial Hierarchies



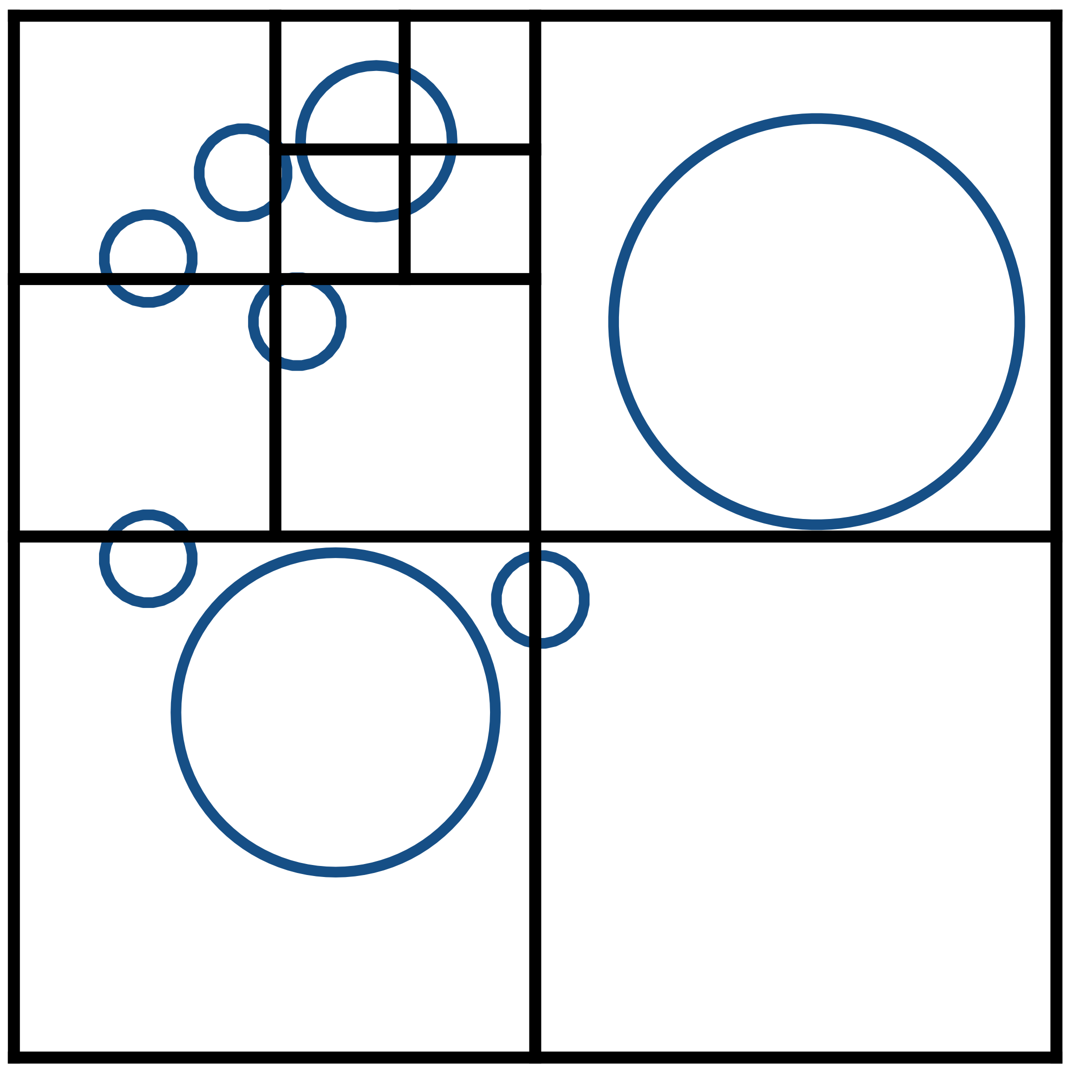
Spatial Hierarchies



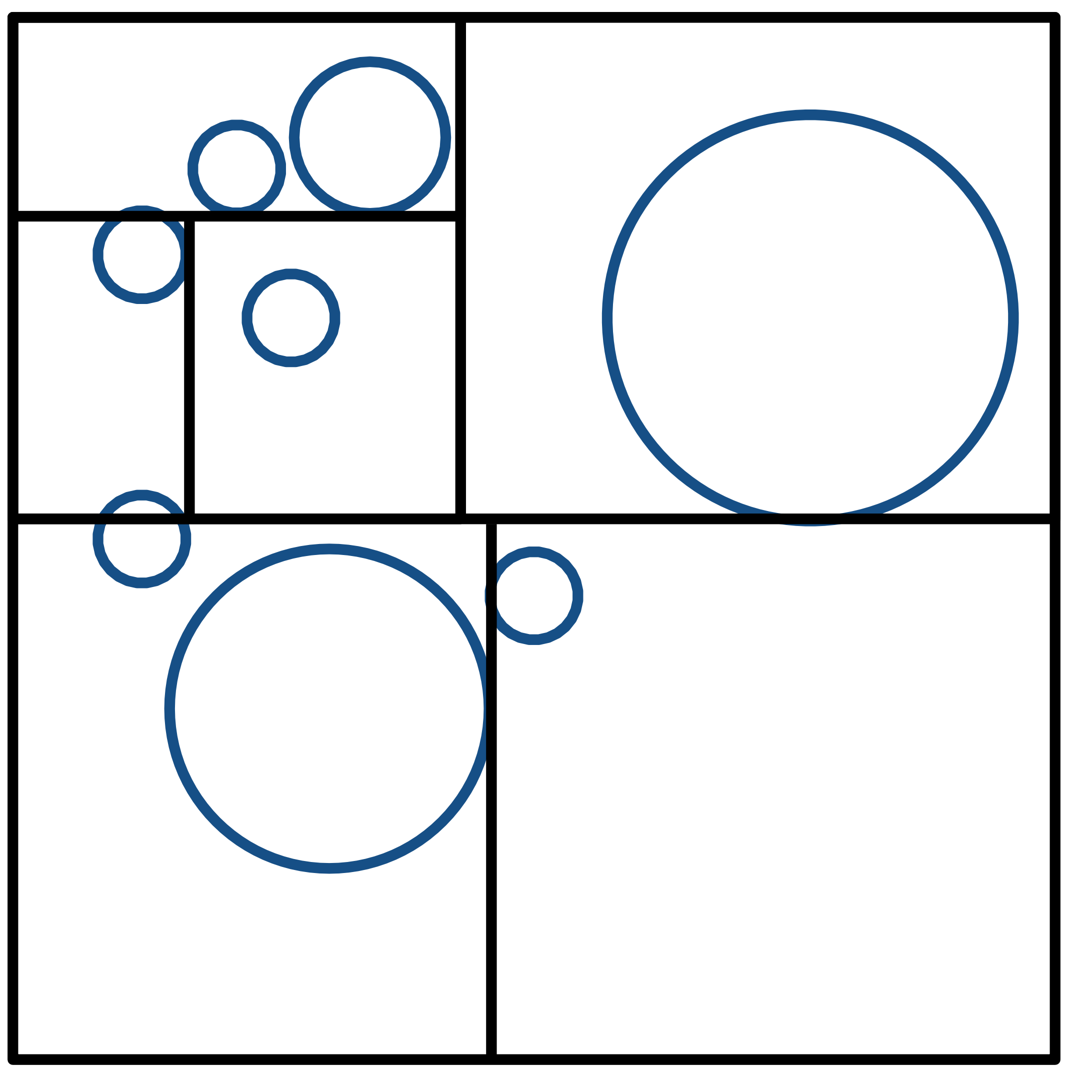
Spatial Hierarchies



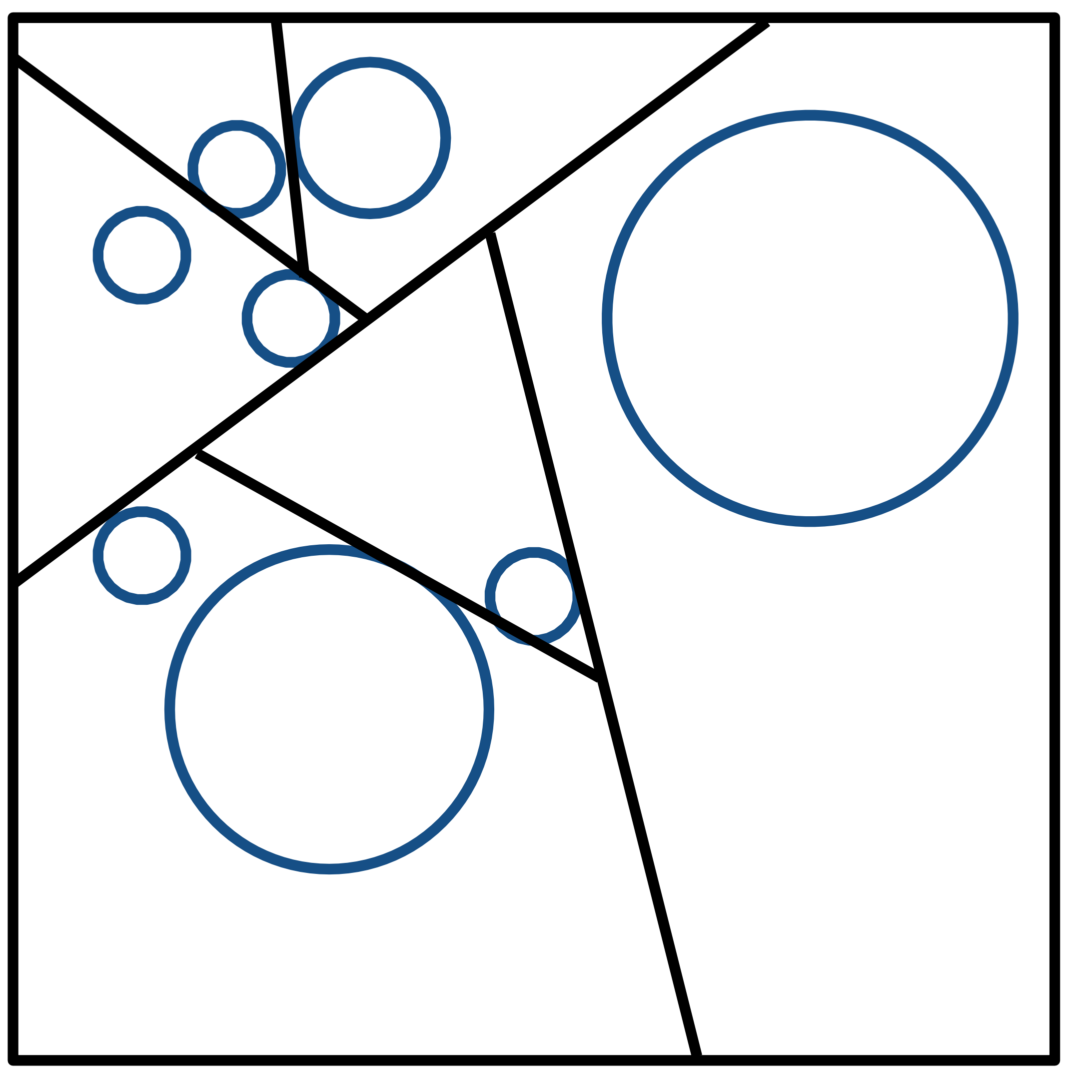
Spatial Partitioning Variants



Oct-Tree



KD-Tree

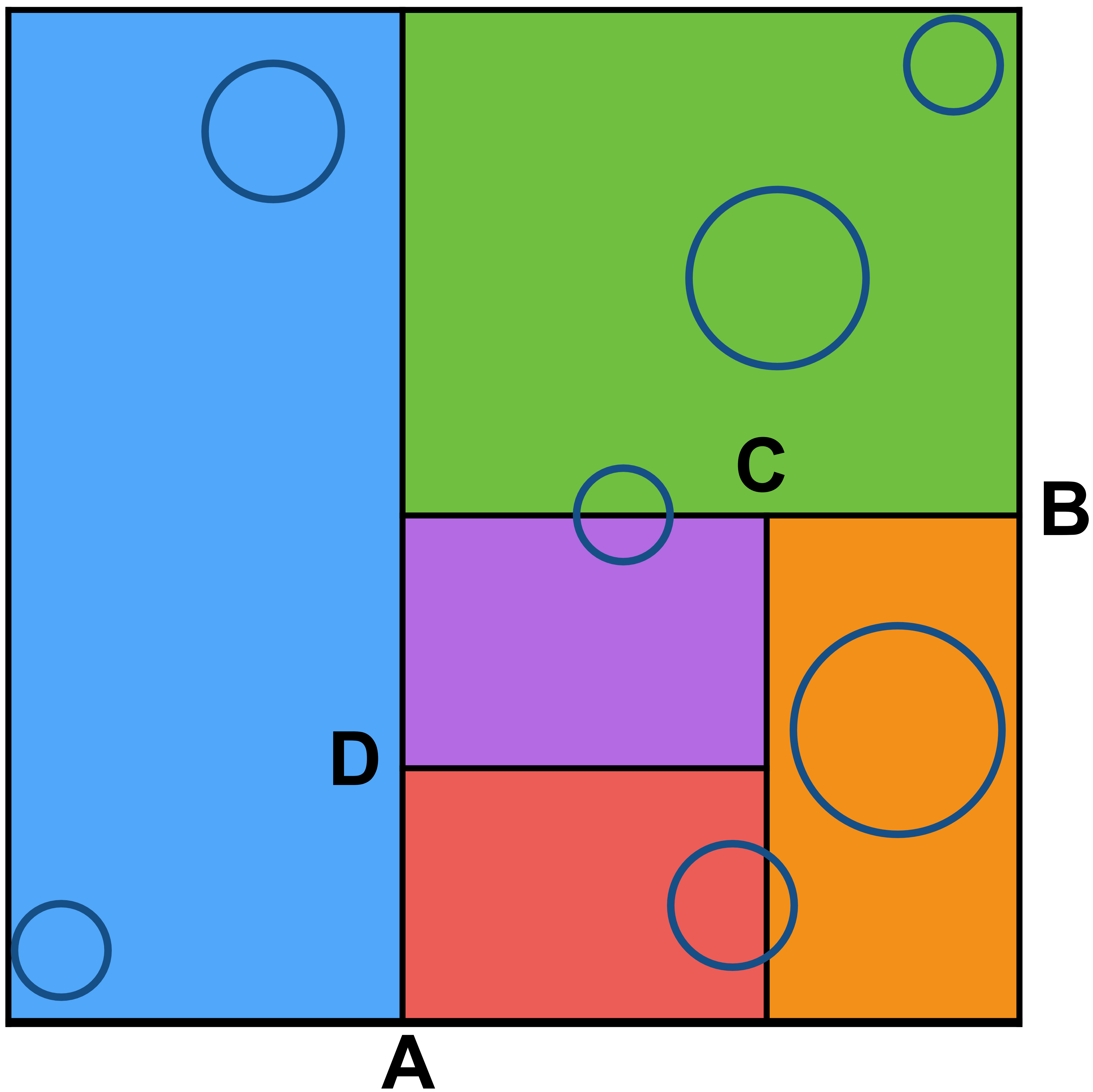


BSP-Tree

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.

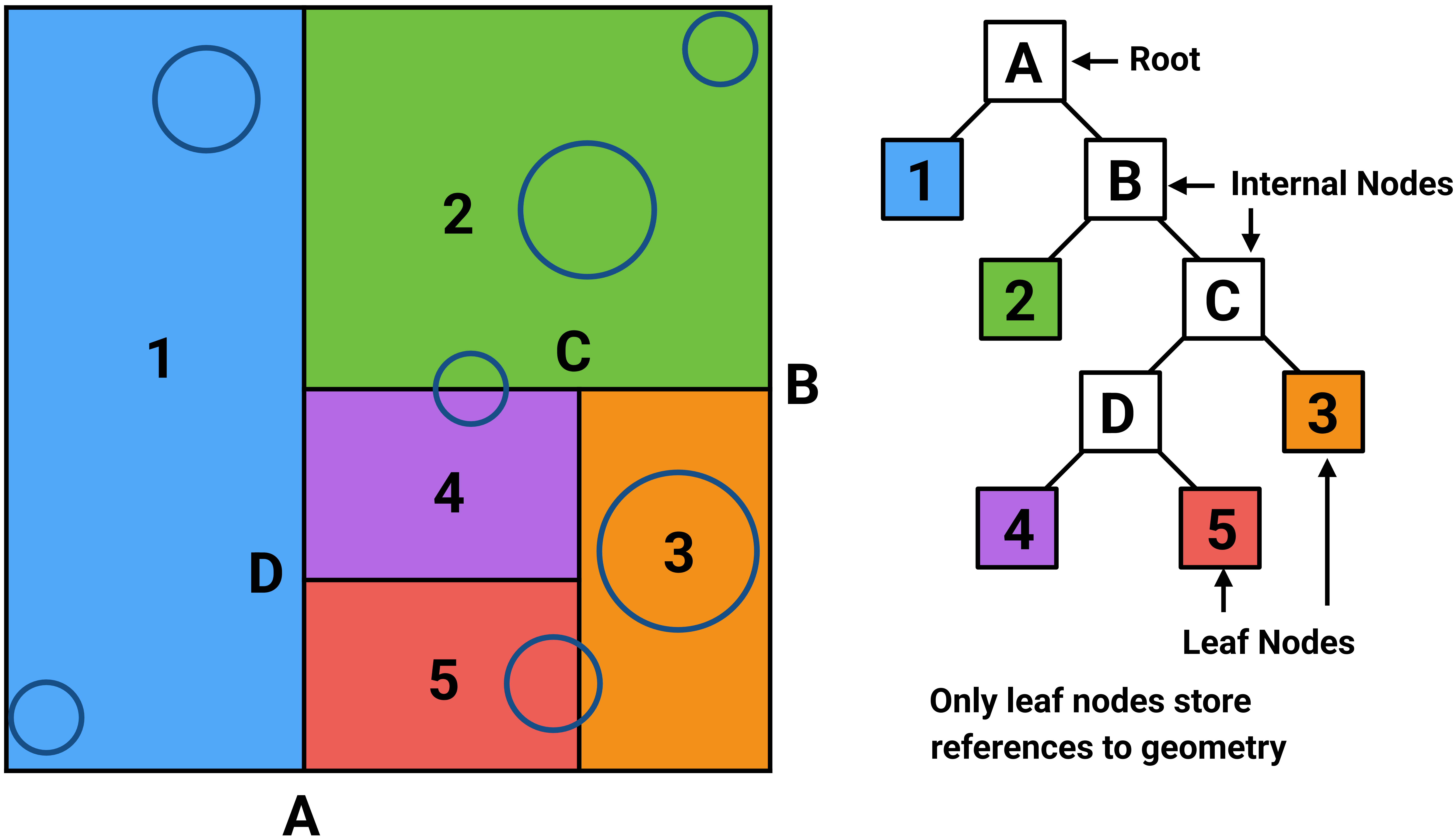
KD-Tree Spatial Partitioning

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



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
- ❑ Intersection Cache

KD-Tree Pre-Processing

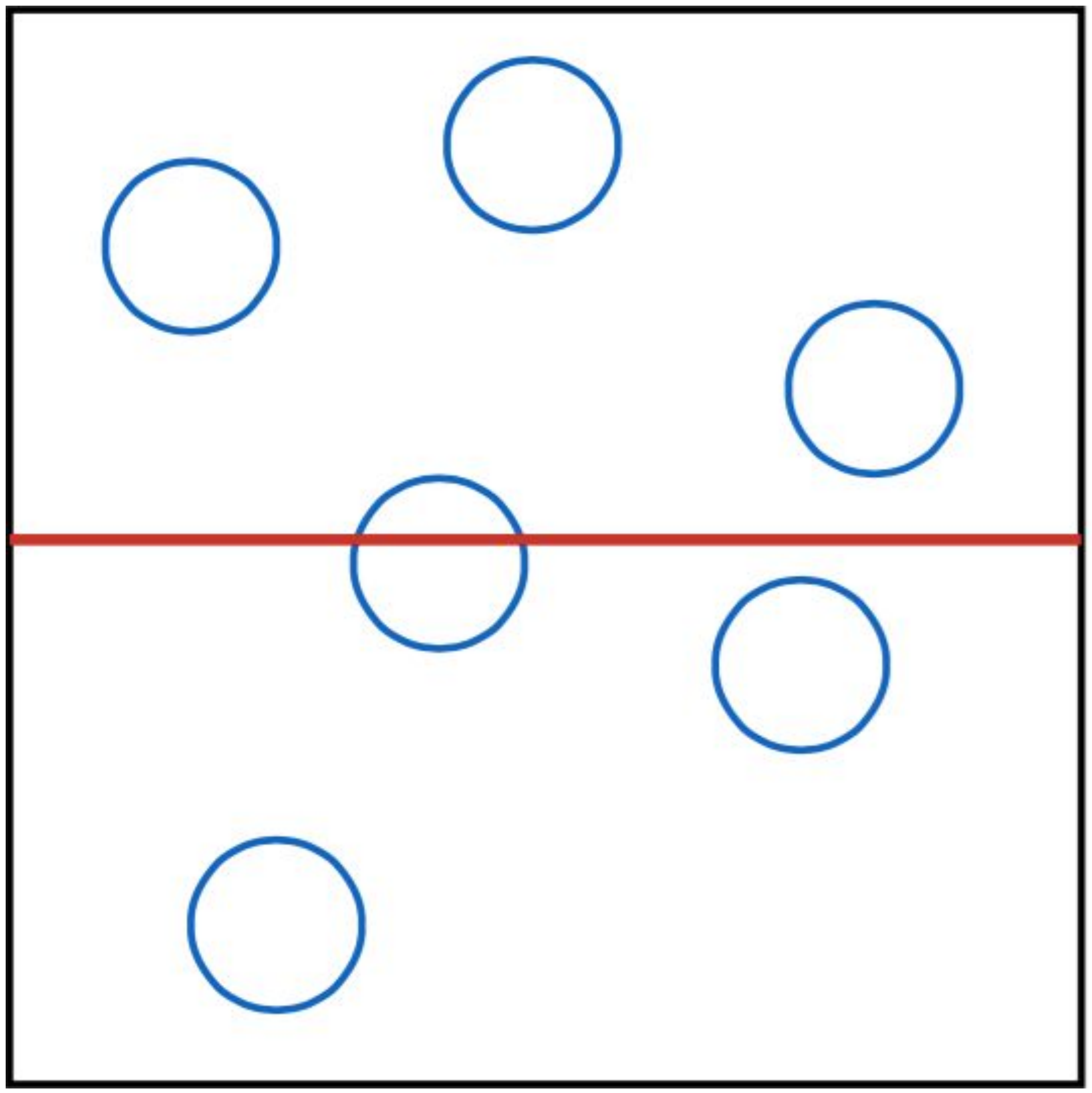
Choosing the split plane

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

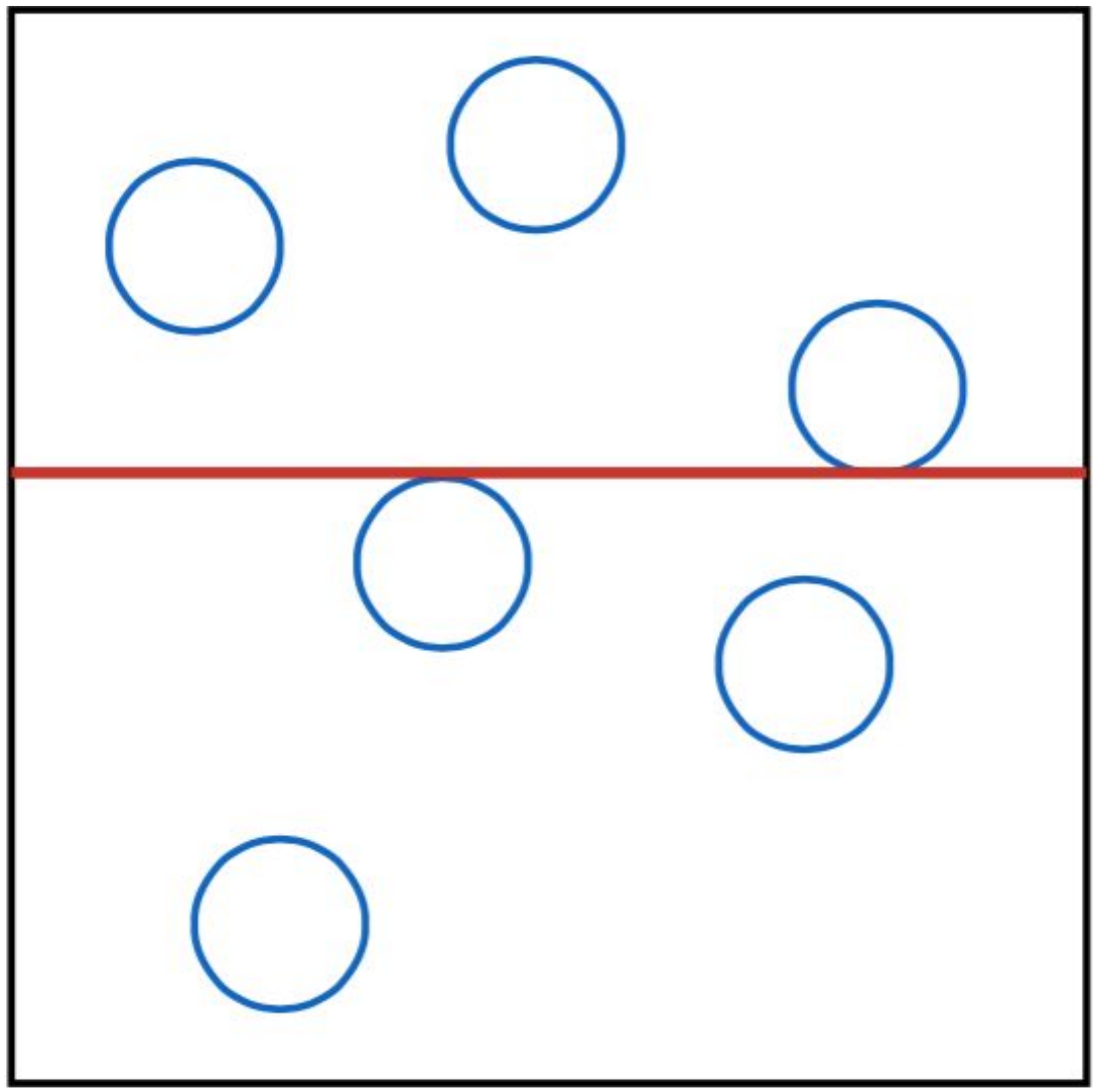
Termination criteria?

- **Simple:** common to prescribe maximum tree depth (empirical $8 + 1.3 \log \mathbf{N_{objs}}$) [PBRT]
- **Ideal:** stop when splitting does not reduce expected cost of ray intersection

Simple Hierarchy Construction

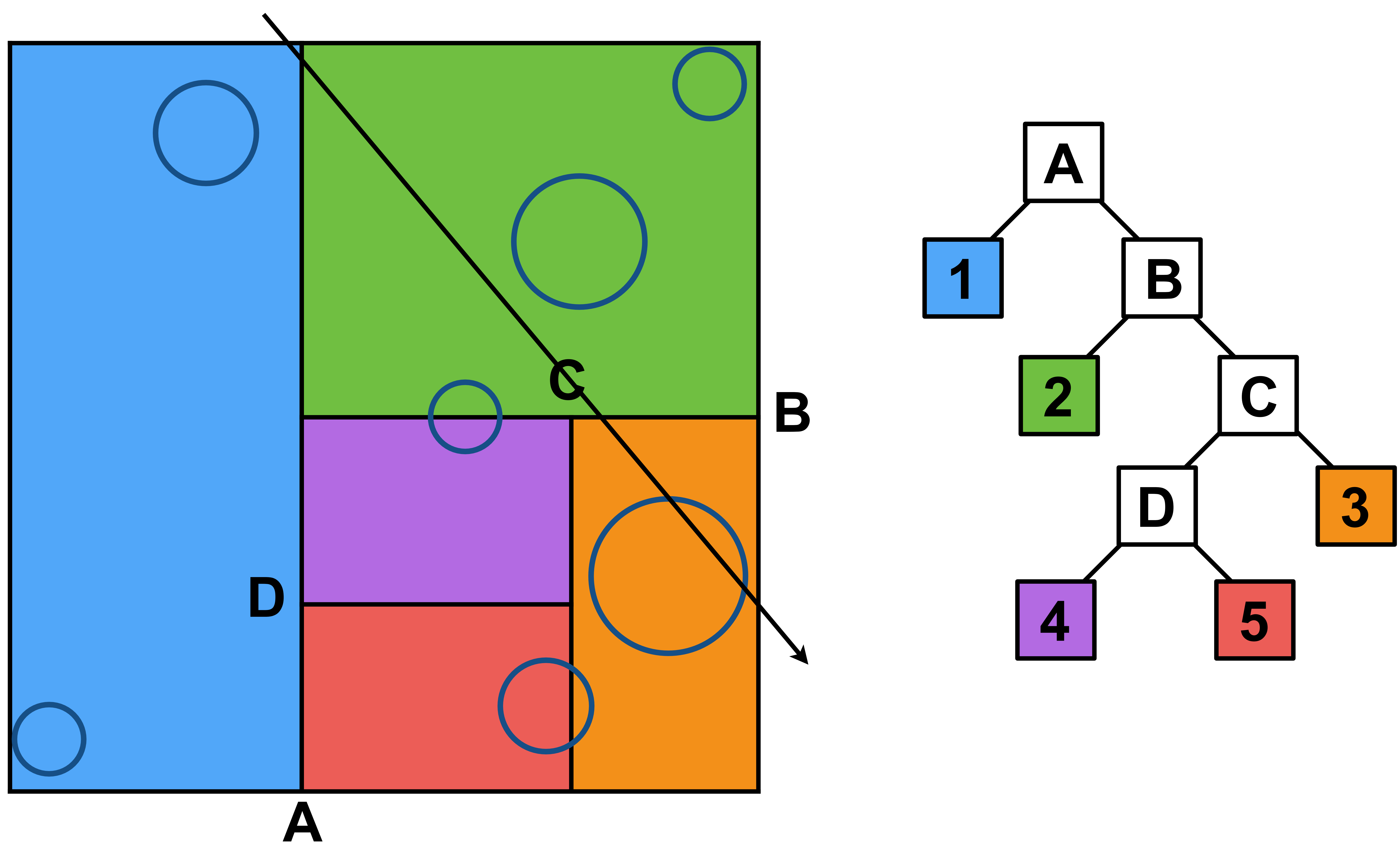


Split at midpoint

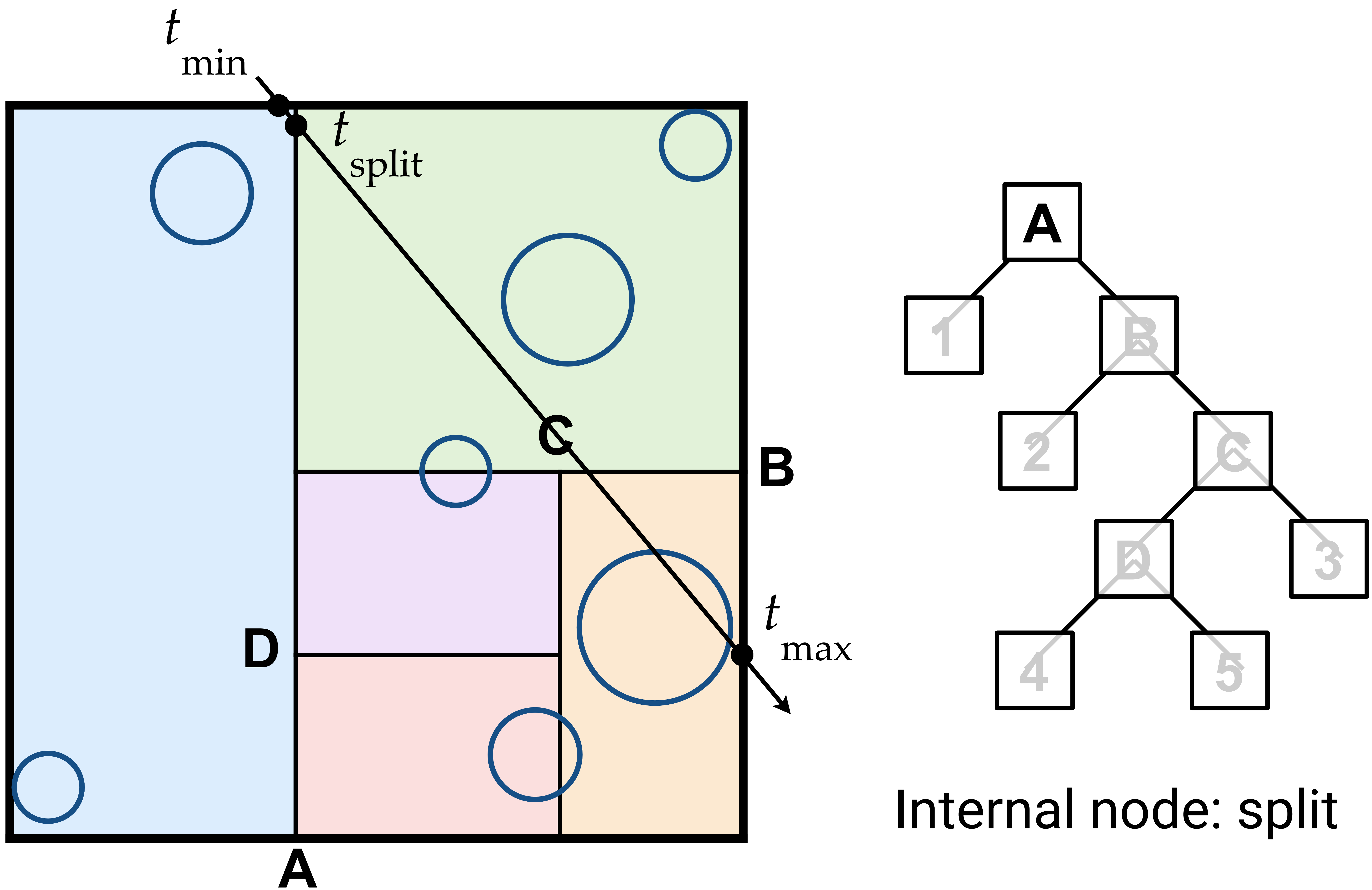


Split at median

Top-Down Recursive In-Order Traversal

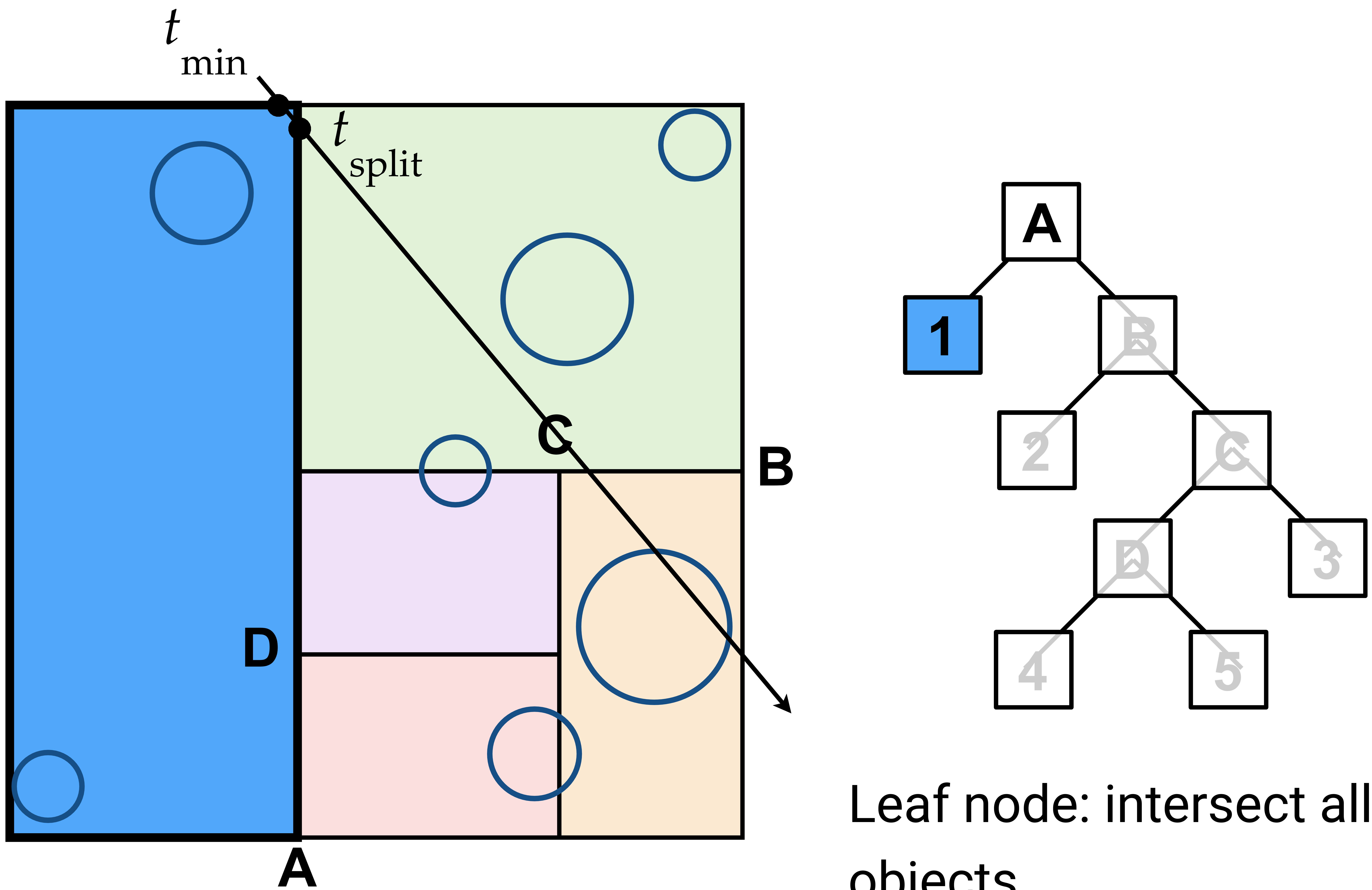


Top-Down Recursive In-Order Traversal



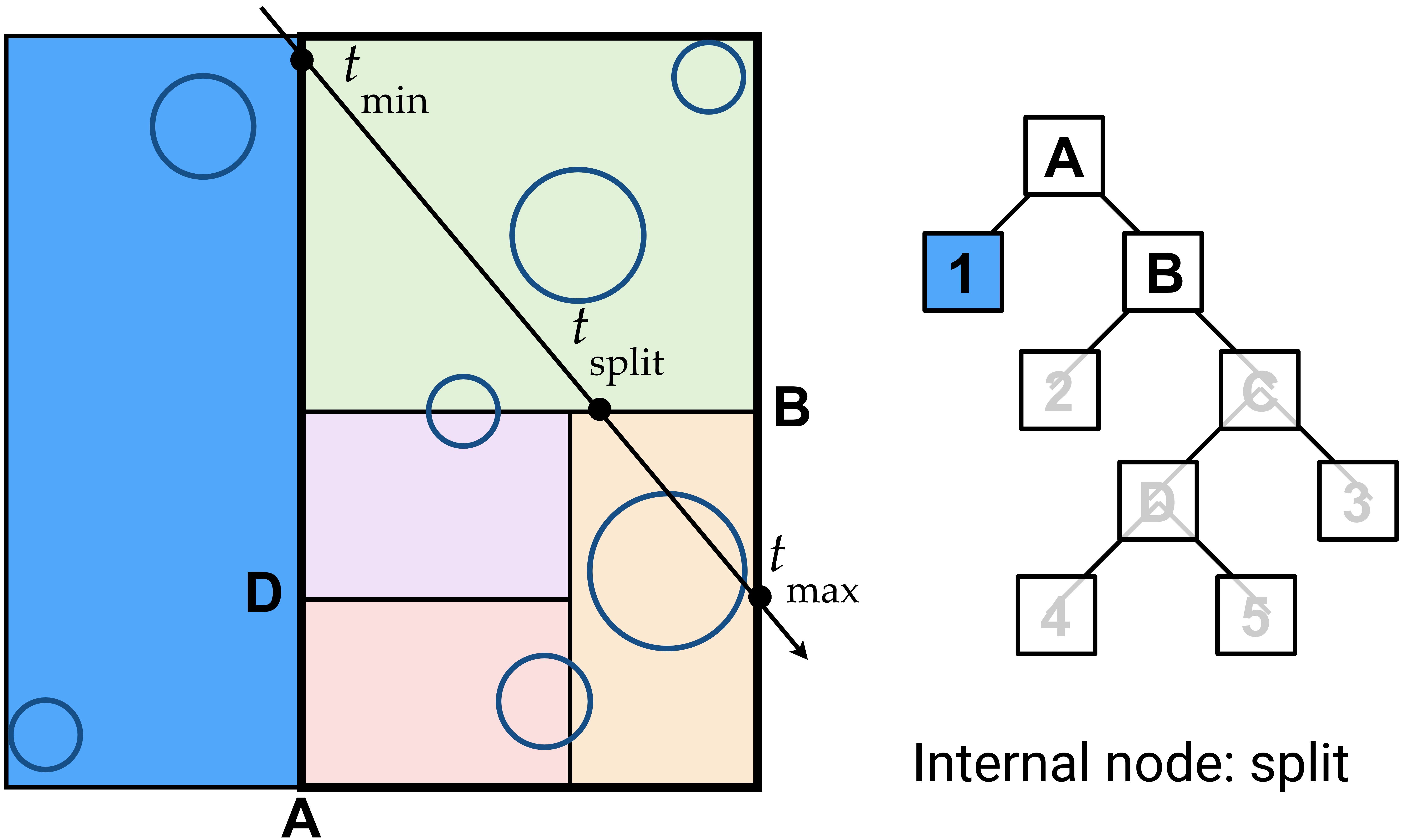
Internal node: split

Top-Down Recursive In-Order Traversal



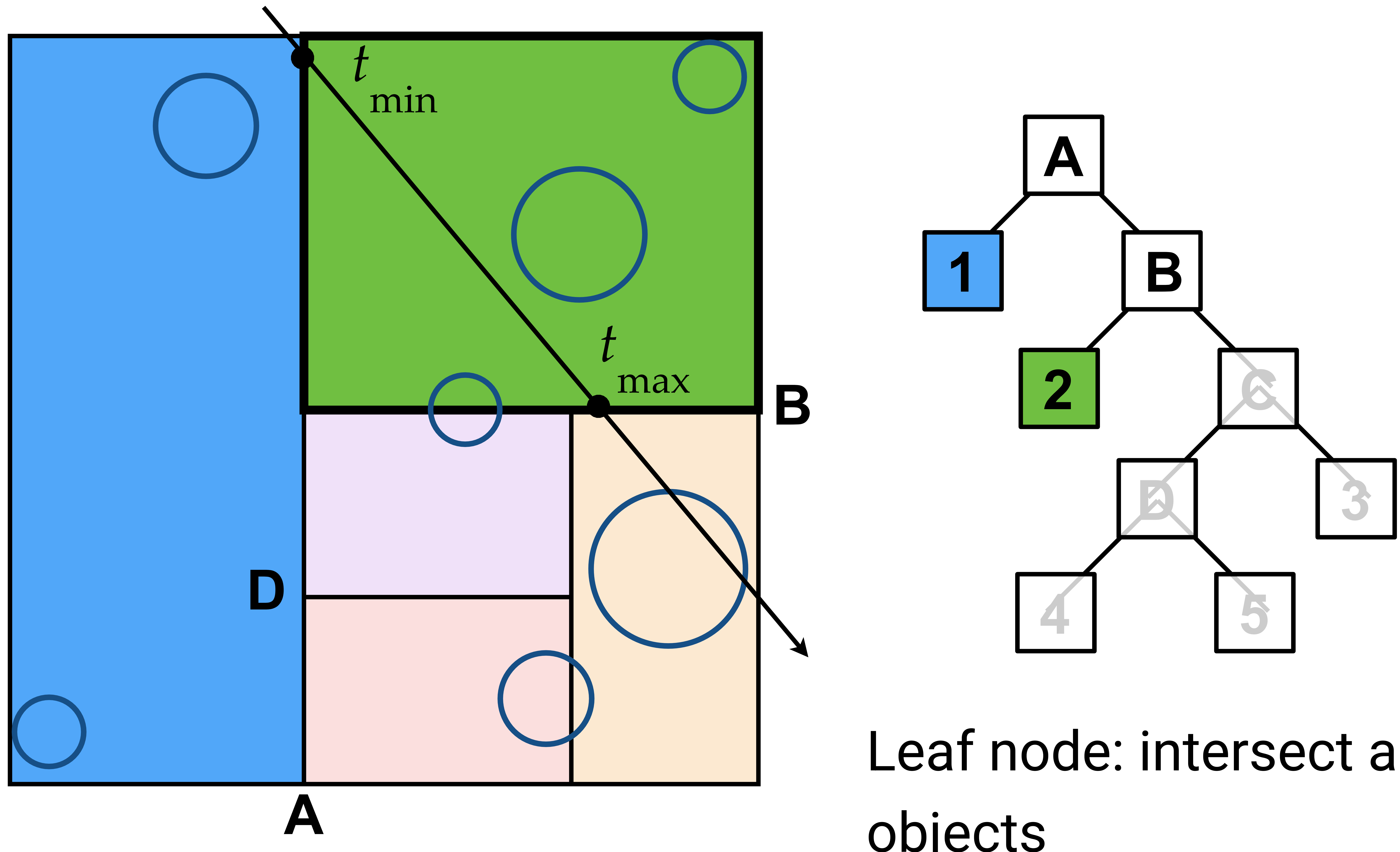
Leaf node: intersect all objects

Top-Down Recursive In-Order Traversal



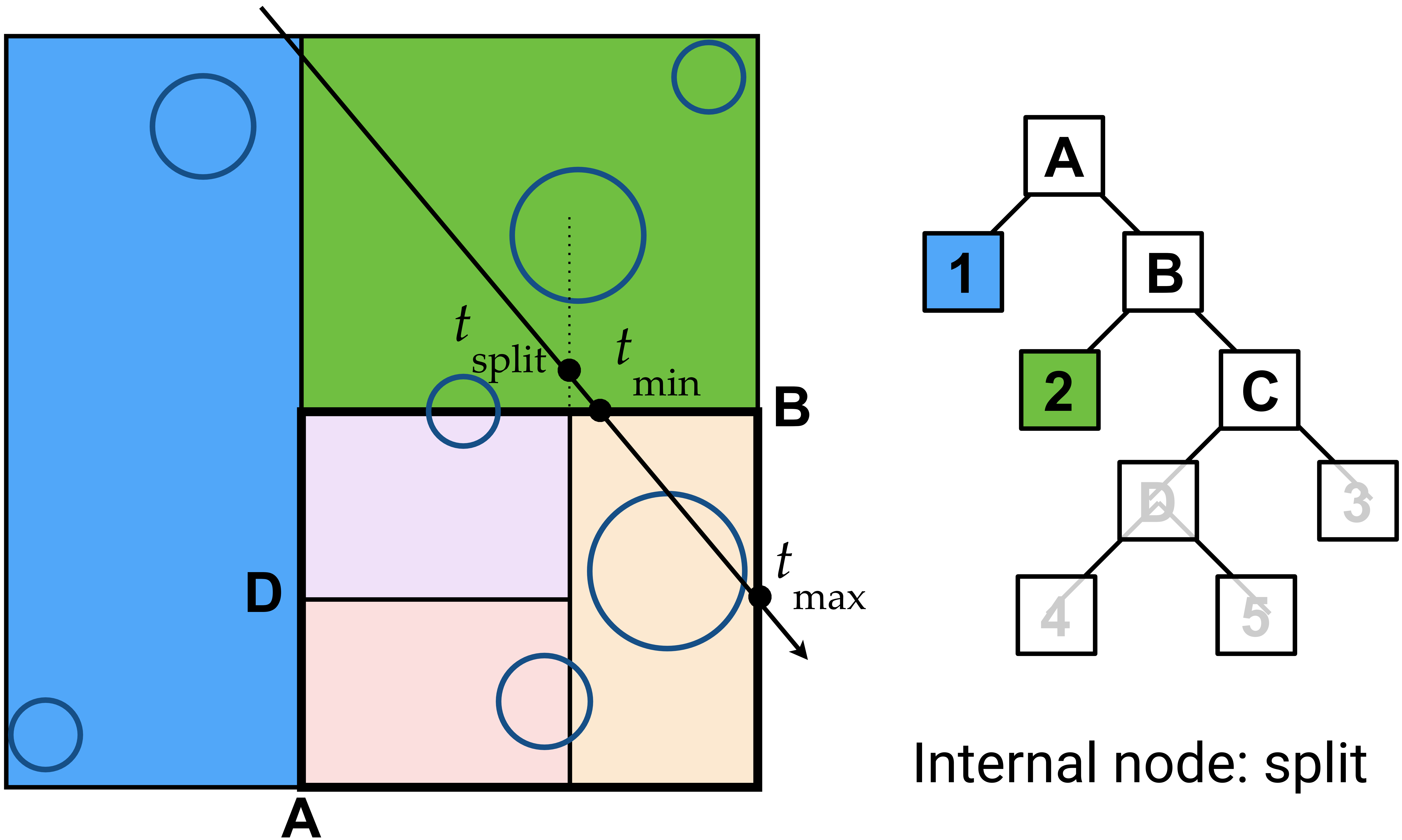
Internal node: split

Top-Down Recursive In-Order Traversal



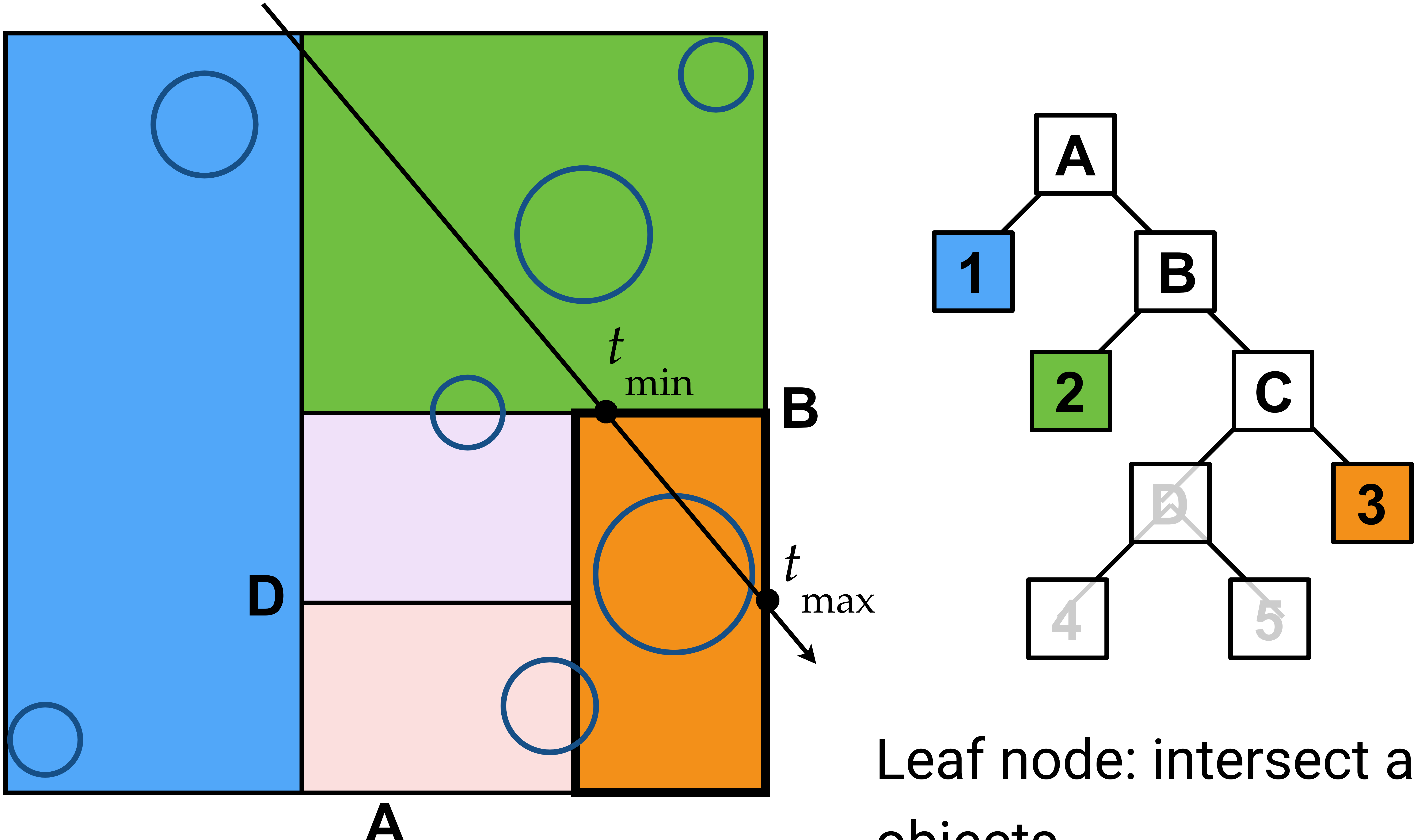
Leaf node: intersect all objects

Top-Down Recursive In-Order Traversal



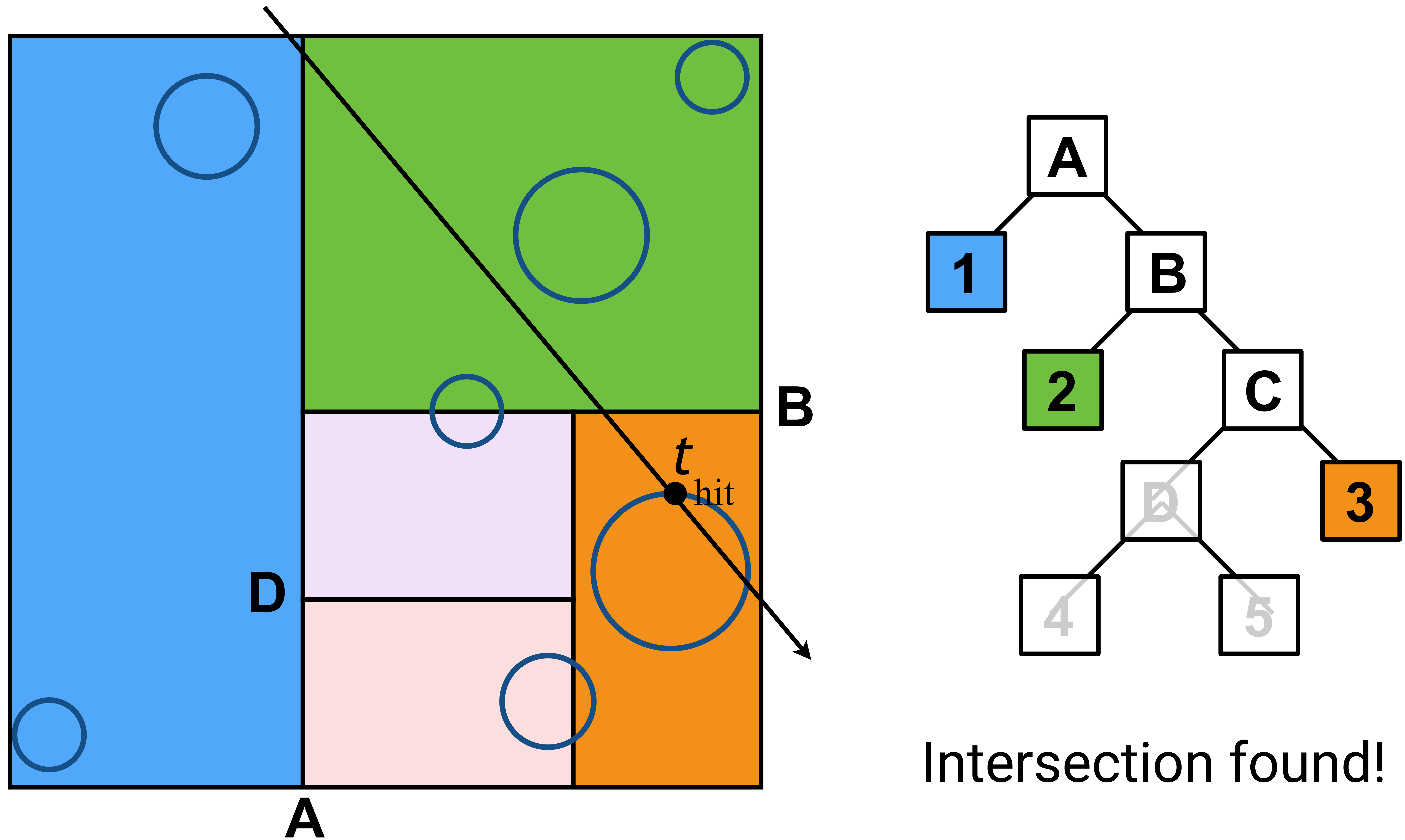
Internal node: split

Top-Down Recursive In-Order Traversal



Leaf node: intersect all objects

Top-Down Recursive In-Order Traversal

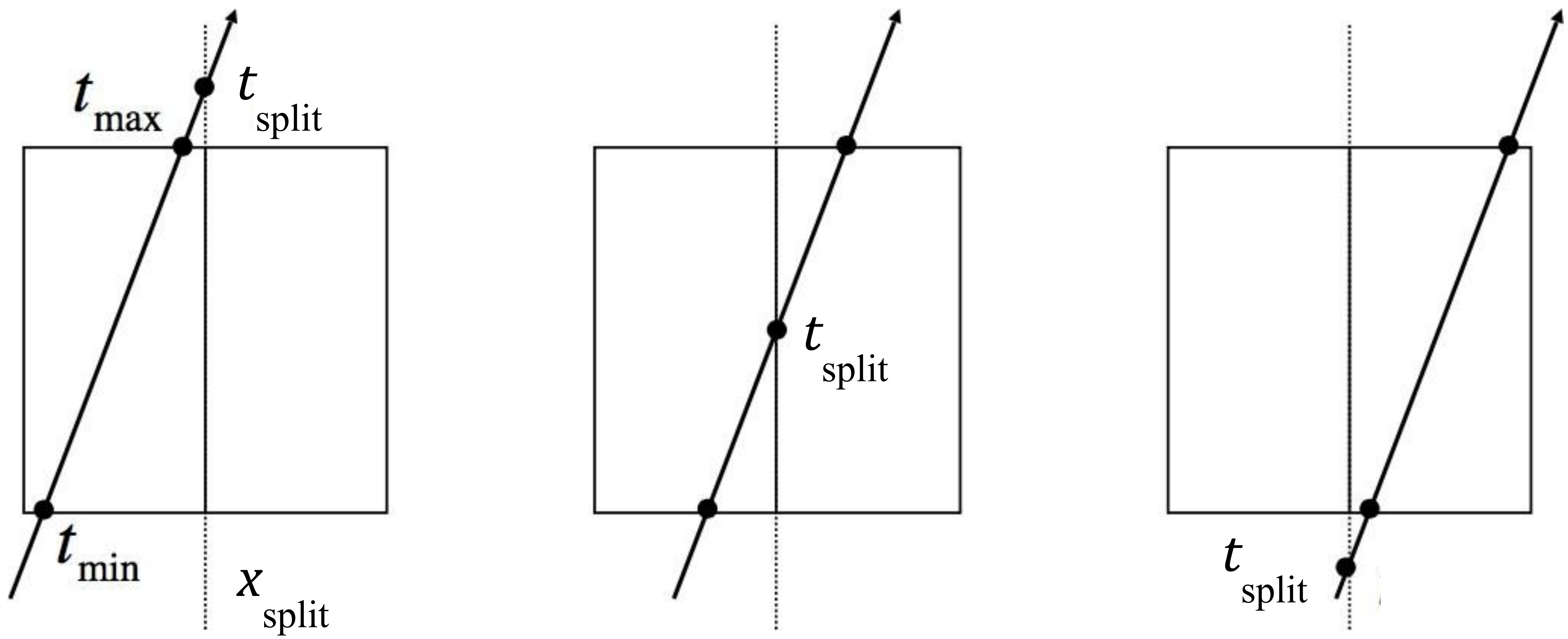


Intersection found!

KD-Trees Traversal – Recursive Step

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

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



$t_{max} < t_{split}$

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

$t_{min} < t_{split} < t_{max}$

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

$t_{split} < t_{min}$

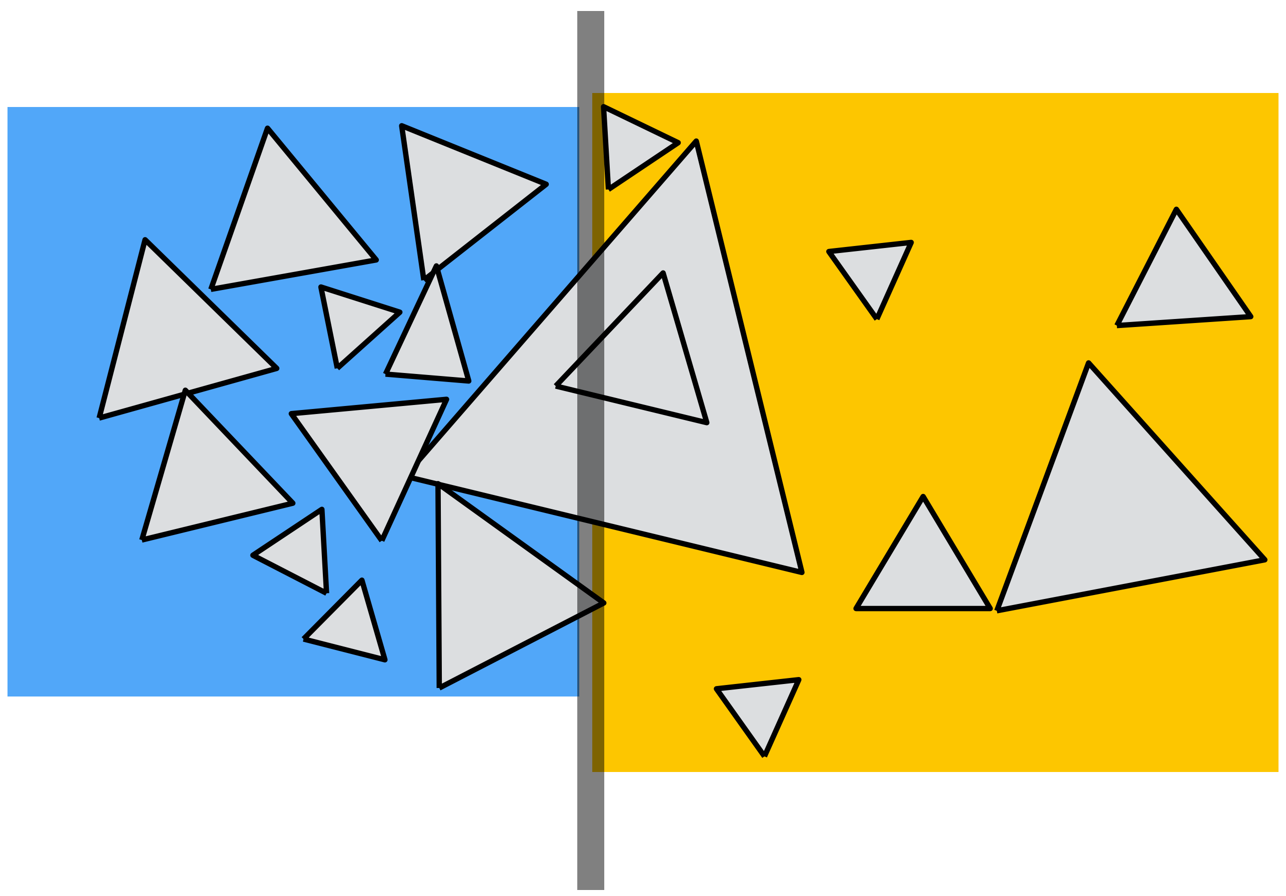
```
Intersect(R,tmin,tmax)
```

Object Partitions & Bounding Volume Hierarchy (BVH)

Spatial vs Object Partitions

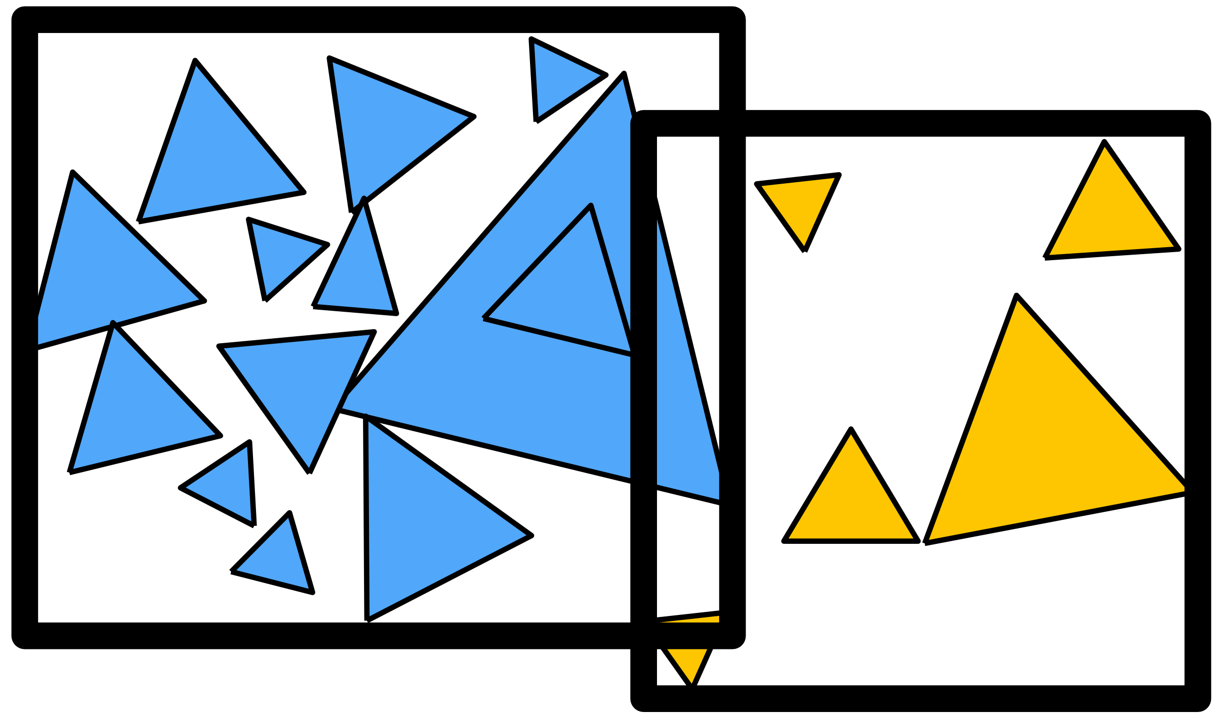
Spatial partition (e.g.KD-tree)

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

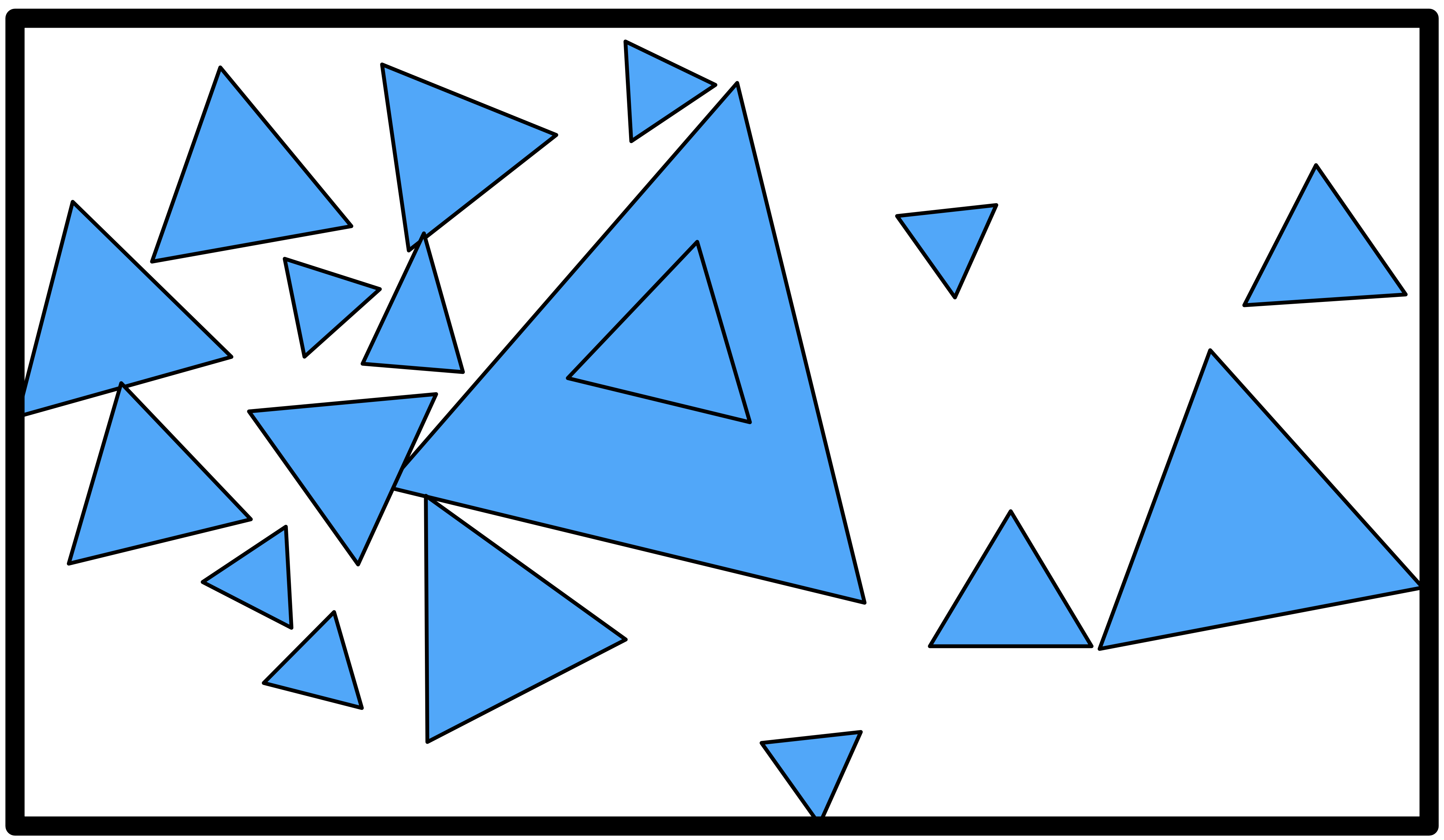


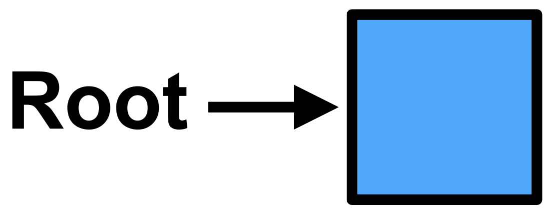
Object partition (e.g. BVH)

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

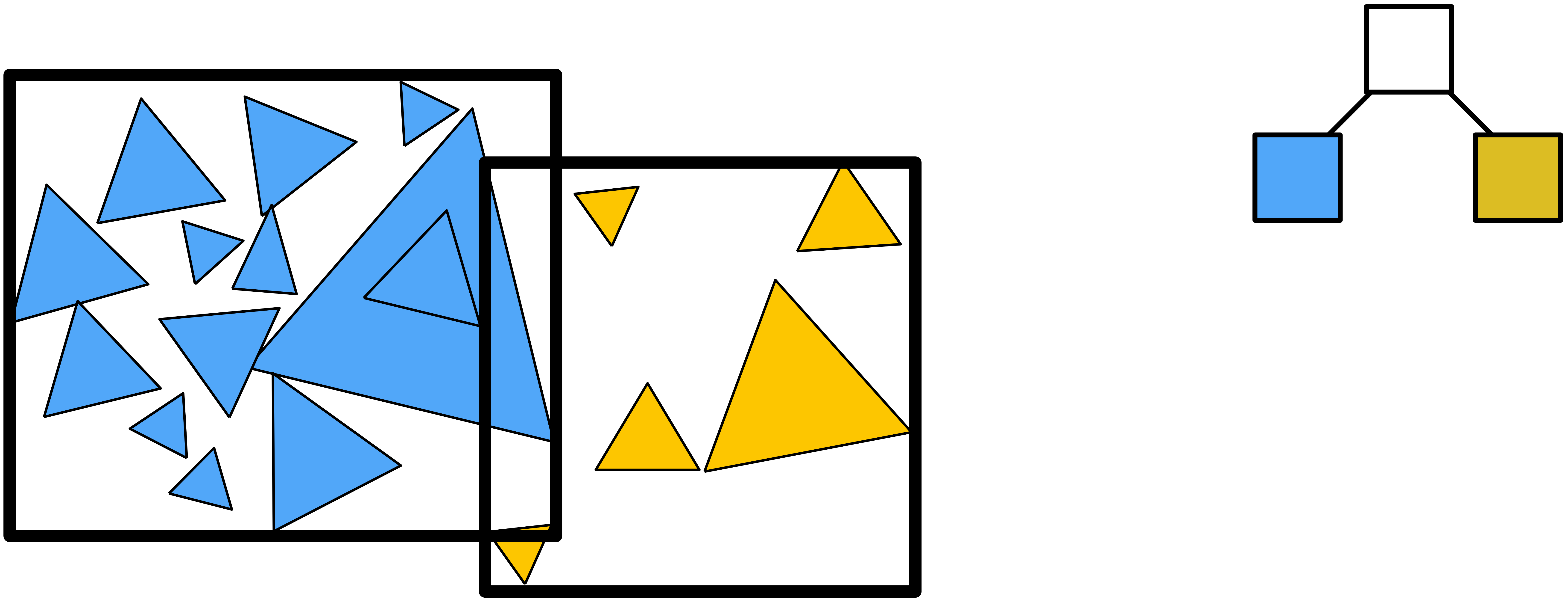


Bounding Volume Hierarchy (BVH)

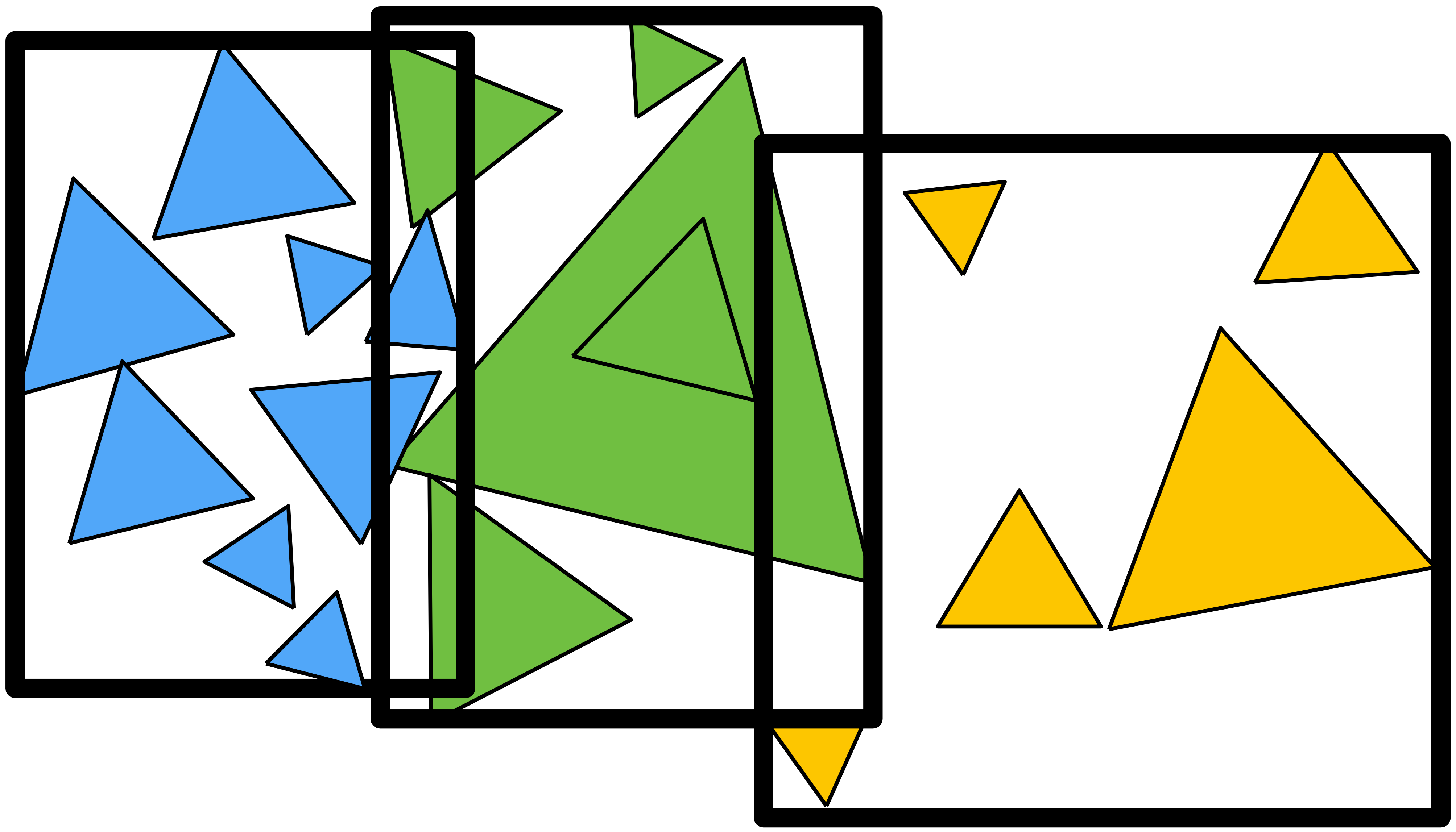


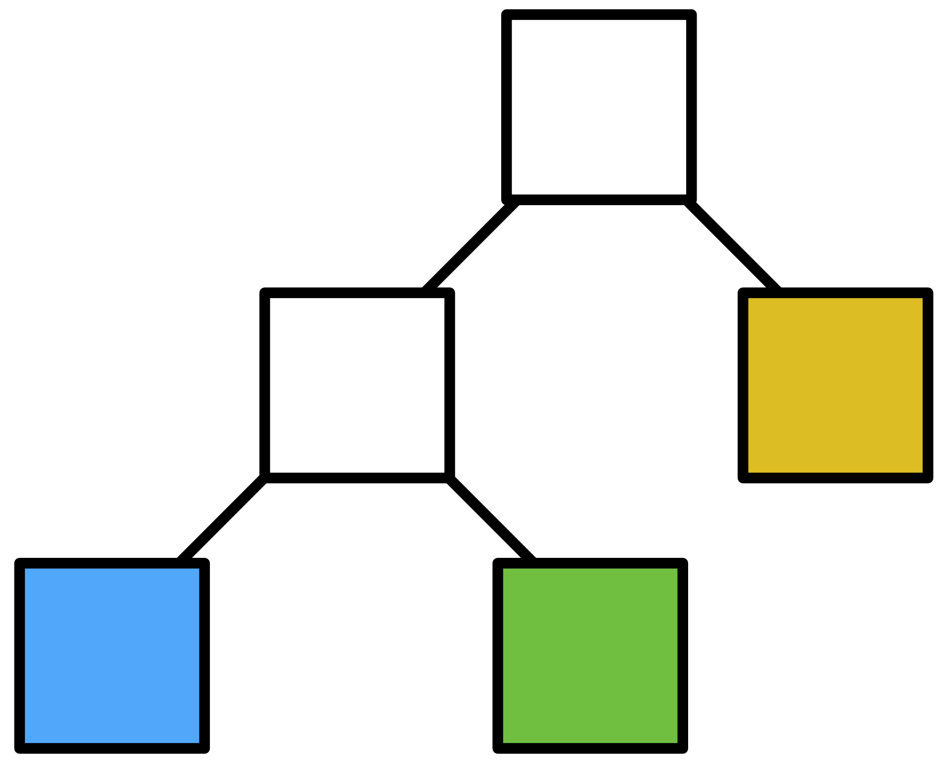


Bounding Volume Hierarchy (BVH)

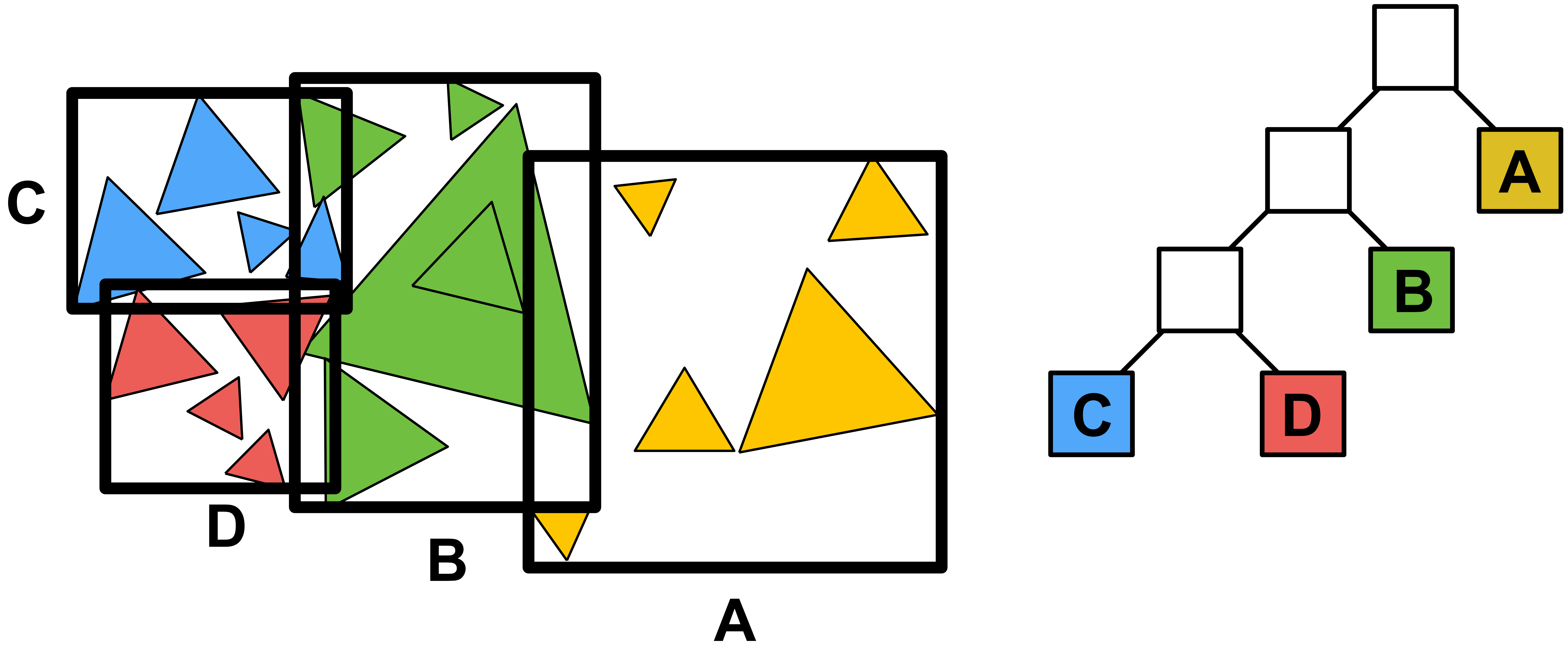


Bounding Volume Hierarchy (BVH)





Bounding Volume Hierarchy (BVH)



Bounding Volume Hierarchy (BVH)

Internal nodes store:

- Bounding box
- Children: reference to child

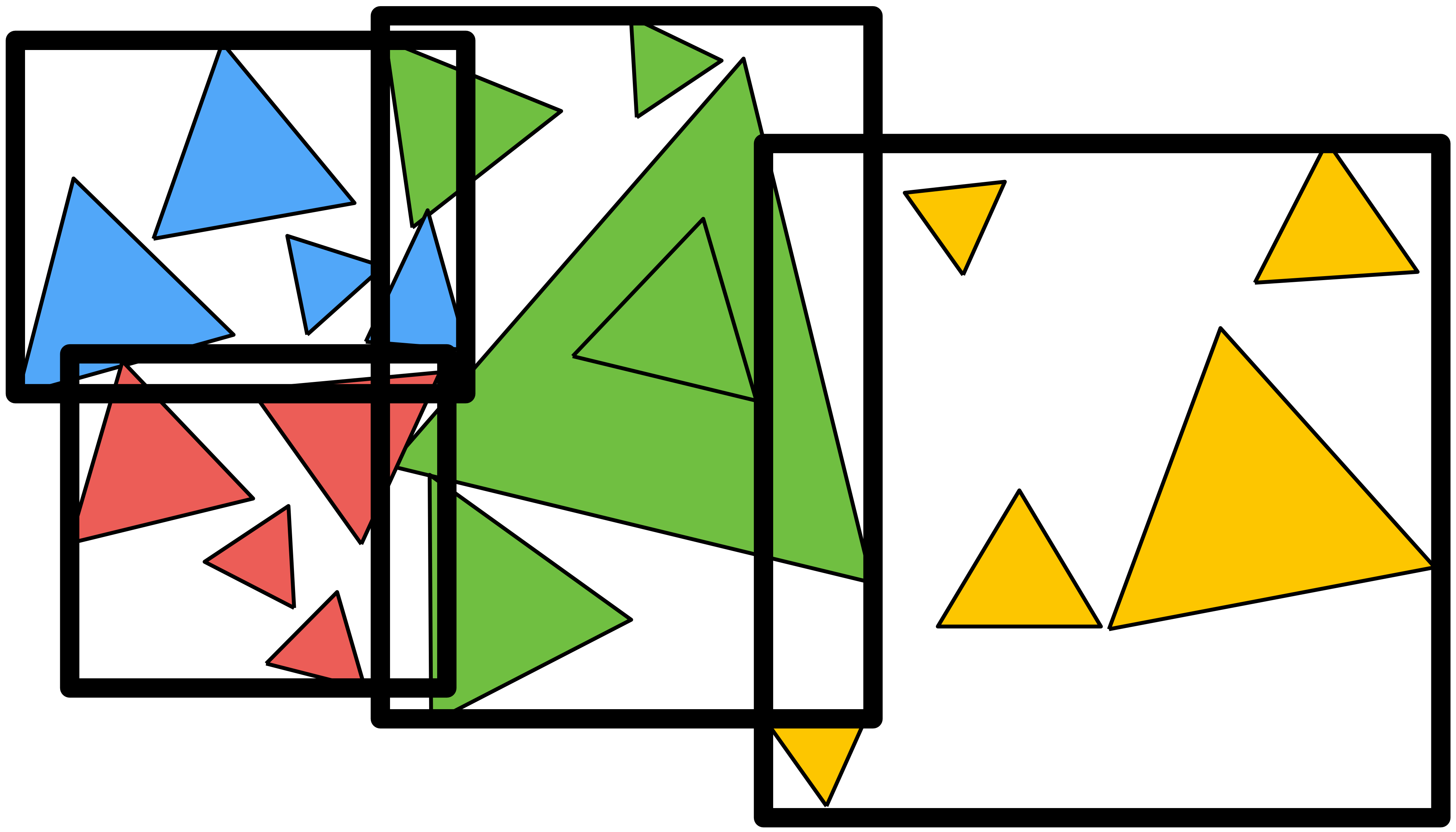
Nodes Leaf nodes store:

- Bounding box
- List of objects

Nodes represent subset of primitives in scene

- All objects in subtree

BVH Pre-Processing



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

BVH Pre-Processing

Choosing the set partition

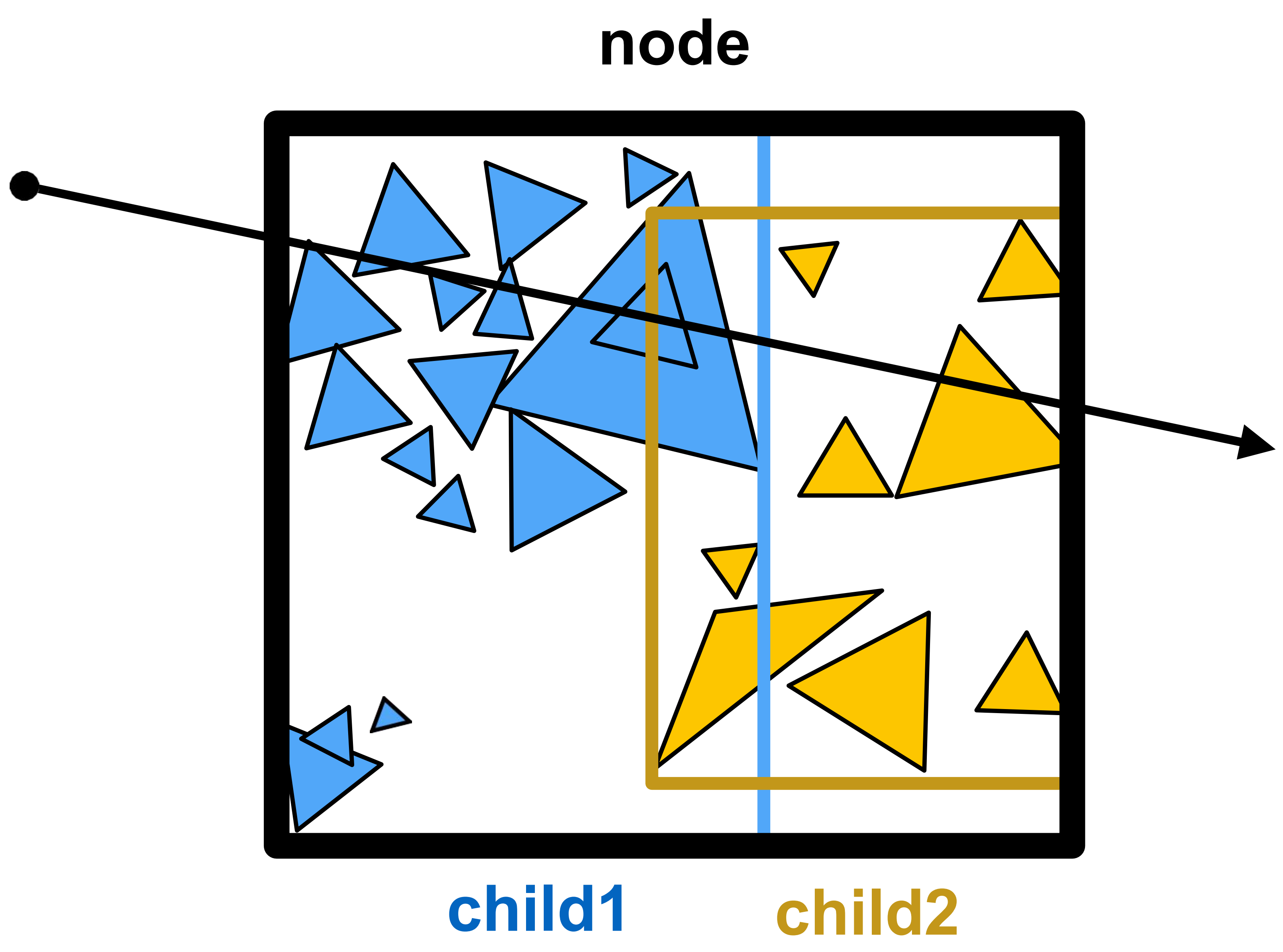
- Choose a spatial dimension to partition over (e.g. x,y,z)
- **Simple 1:** Split objects around spatial midpoint
- **Simple 2:** Split at location of median object
- **Ideal:** split to minimize expected cost of ray intersection

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

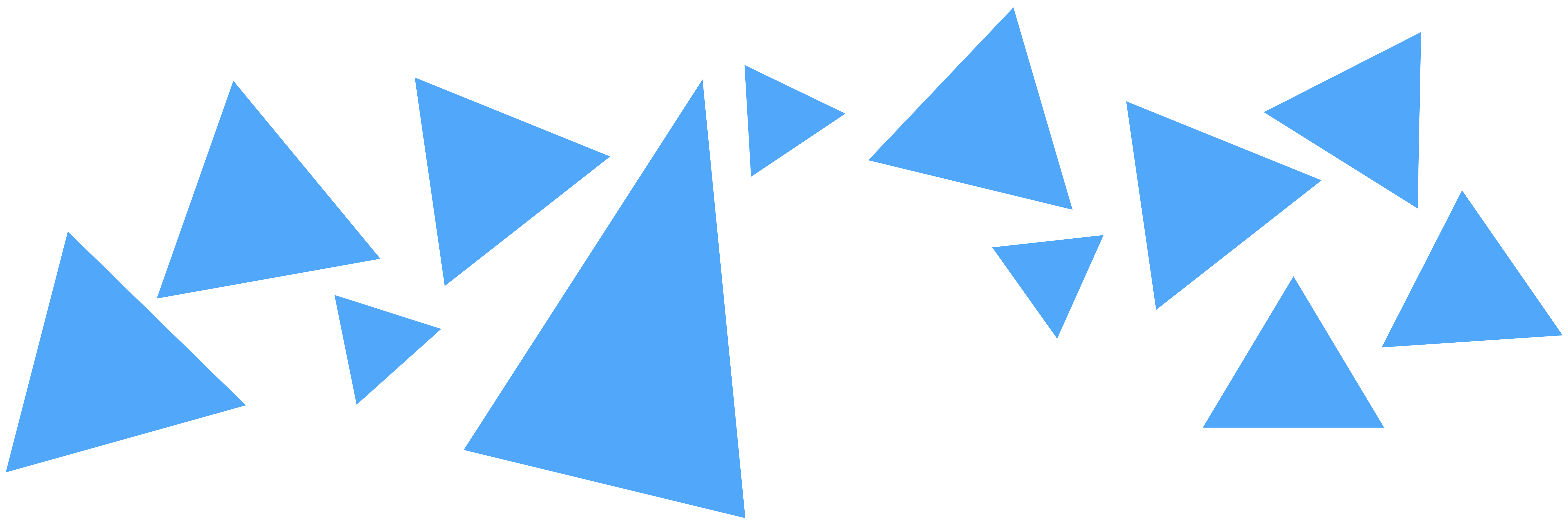
```
Intersect(Ray ray, BVH node)
  if (ray misses node.bbox)
    return;

  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;
```



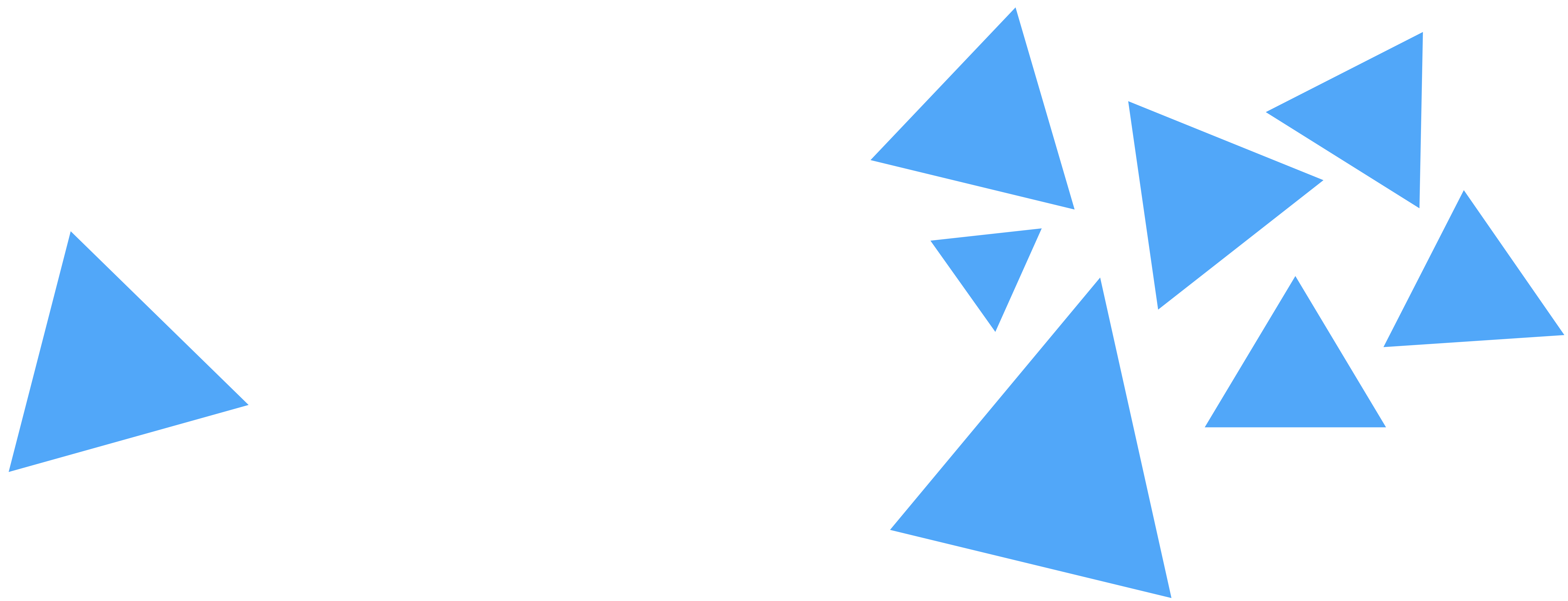
Jakarta

Optimizing Hierarchical Partitions:(How to Split?)

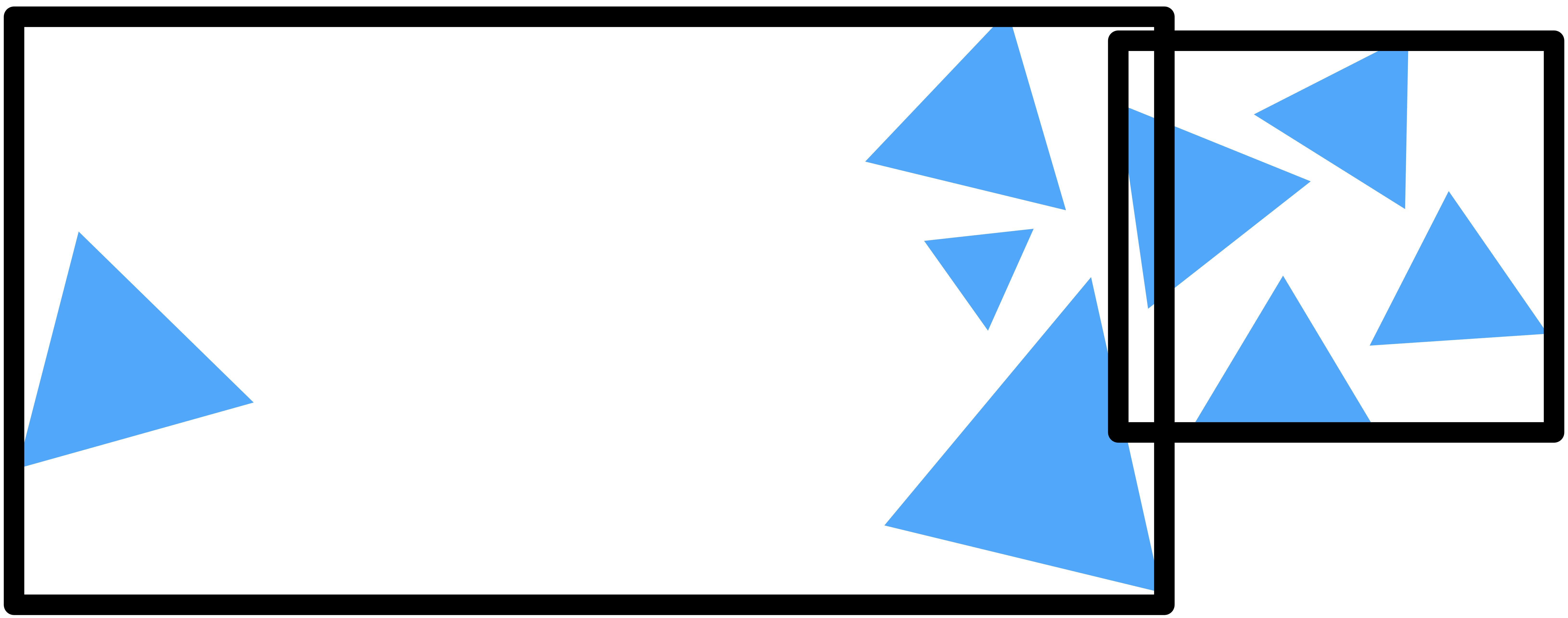
How to Split into Two Sets? (BVH)



How to Split into Two Sets? (BVH)



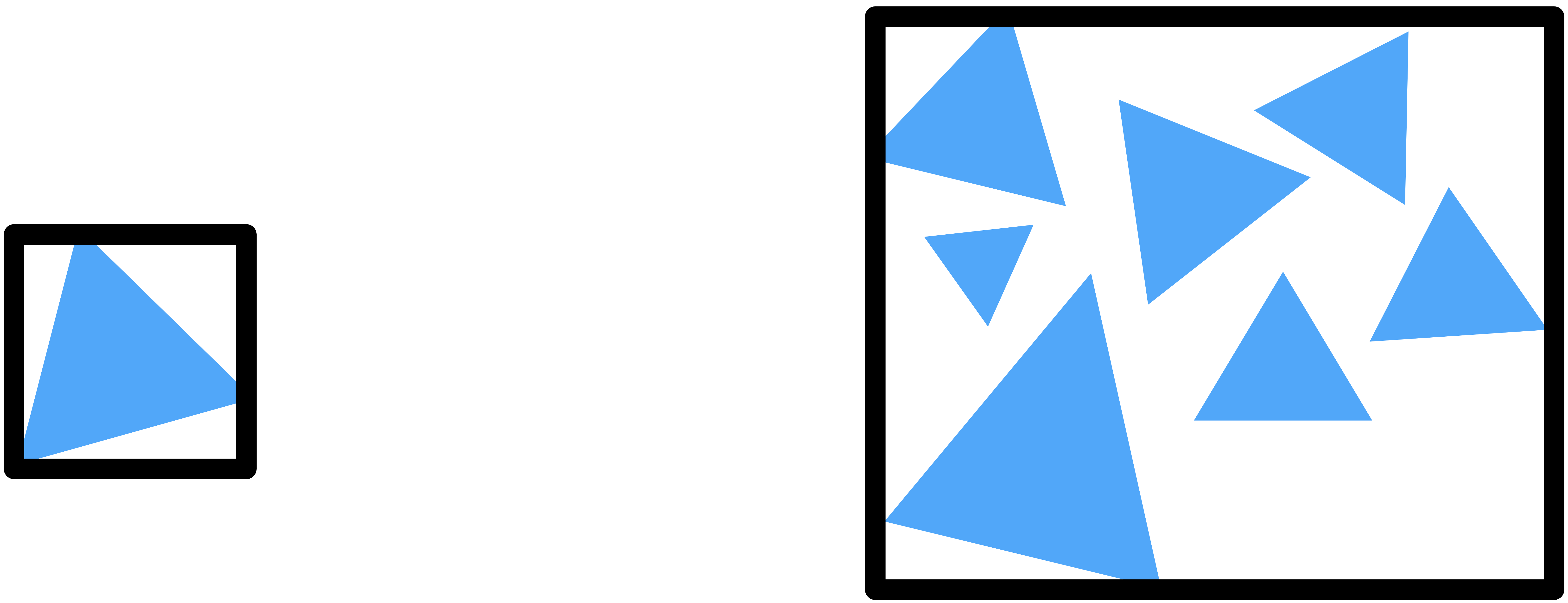
How to Split into Two Sets? (BVH)



Split at median element?

Child nodes have equal numbers of elements

How to Split into Two Sets? (BVH)



Is this 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:

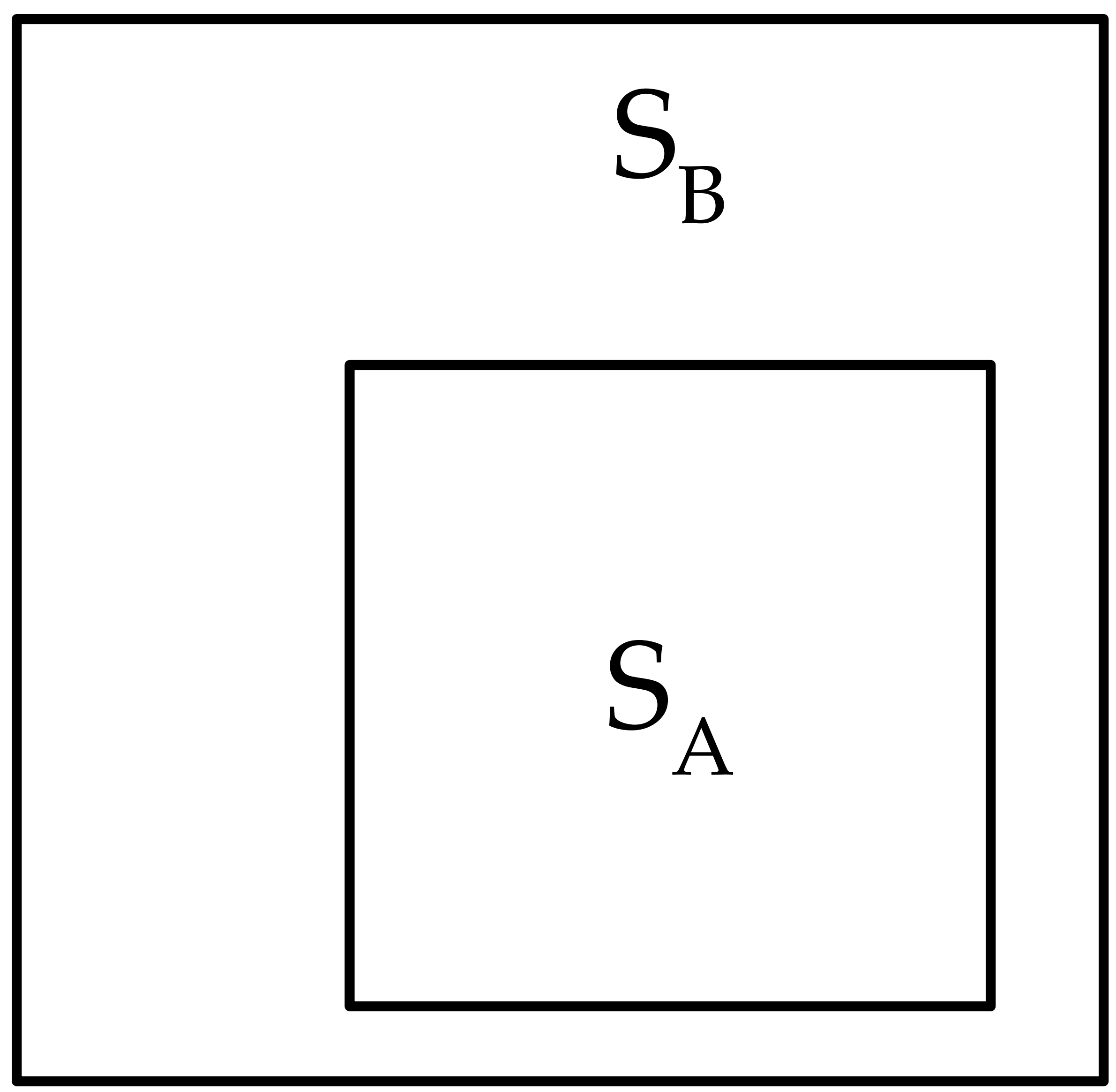
```
Cost(node) = C_trav + Prob(hit L)*Cost(L)
                    + Prob(hit R)*Cost(R)
```

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

Optimizing Hierarchical Partitions: 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**.**



$$P\,(\mathrm{hit}A \mid \mathrm{hit}B) = \frac{S_A}{S_B}$$

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

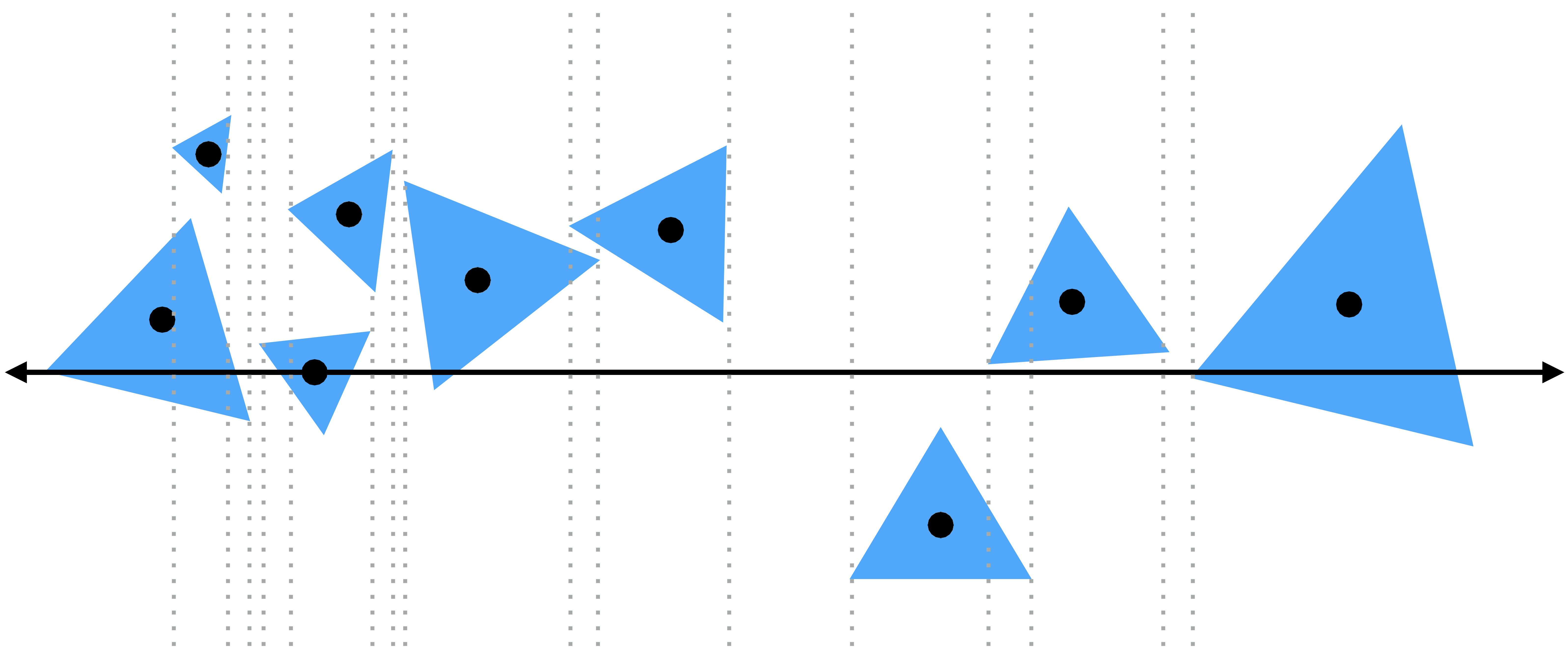
- Common approximation is number triangles in node's subtree

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

where 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

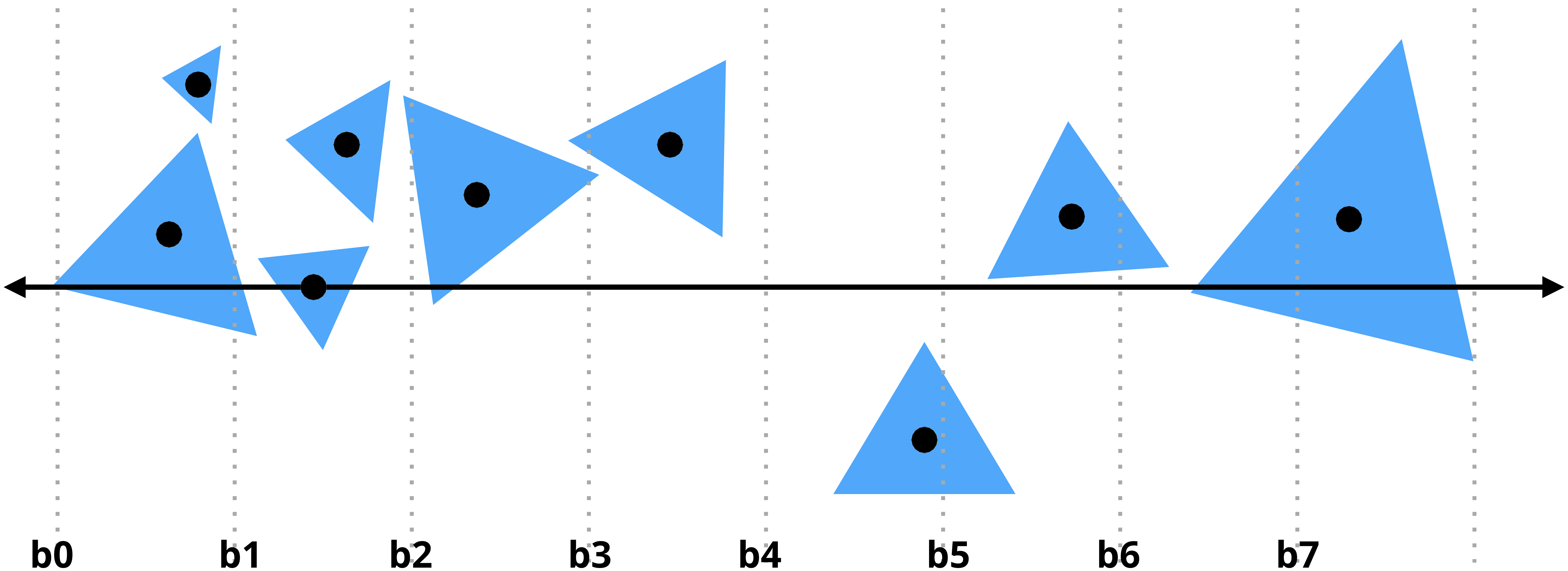
Constrain search to axis-aligned spatial partitions

- 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 Approximation)

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 innode:
    b = compute_bucket(p.centroid);
    b.bbox.union(p.bbox);
    b.prim_count++;
For each B-1 possible partitioning planes evaluate SAH
  Execute lowest cost partitioning found(or make node a leaf)
```

Cost-Optimization Applies to Spatial Partitions Too

- **We only discussed optimization for 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 apply Surface Area Heuristic
  - Note that surface area vs. number of nodes in children differs between spatial partitions and BVH

Things to Remember

- **Ray-geometry intersection as solution of ray-equation substituted into implicit geometry function**
- **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.