CSE 250
Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Lec 35: Spatial Data Structures (pt 2)

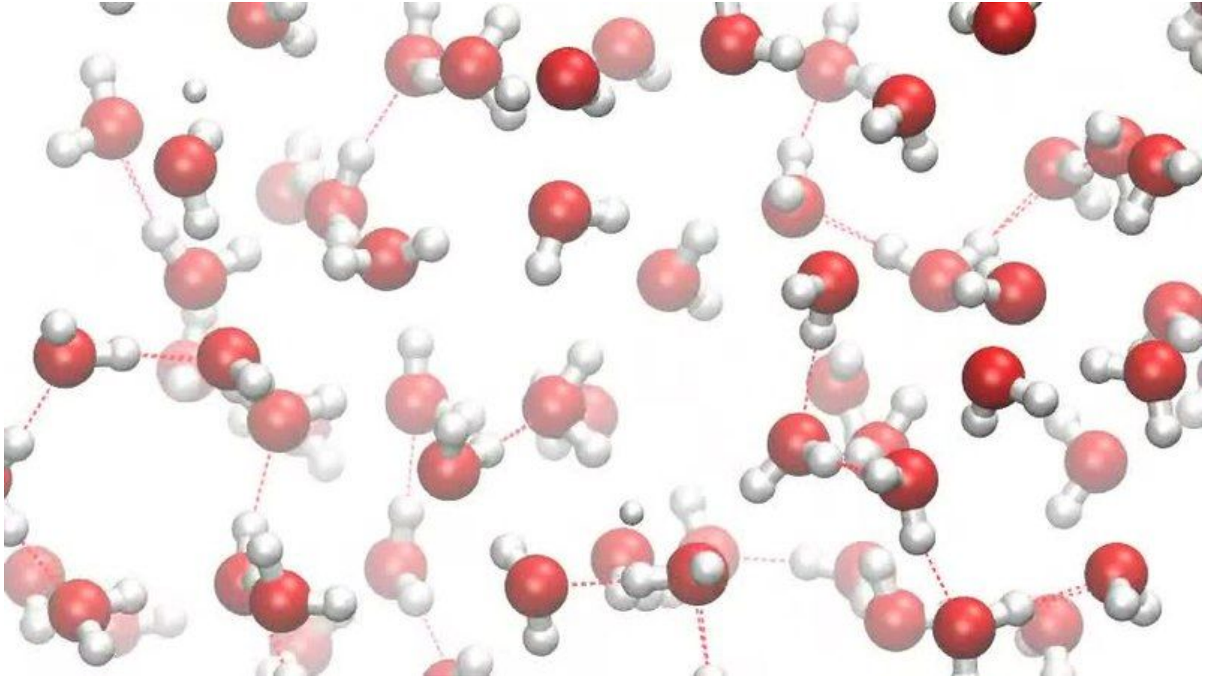
Announcements

- PA3 due Sunday
- Do your course evaluations!

Some Problems are REALLY Big

ESA/Hubble and NASA: http://www.spacetelescope.org/images/potw1006a/

Some Problems are REALLY Small



Molecular Dynamics Simulation of Liquid Water
https://commons.wikimedia.org/wiki/File:A_Molecular_Dynamics_Simulation_of_Liquid_Water_at_298_K.webm

Some Problems are REALLY Detailed

This is **NOT** a photo. It is a computer generated image.

https://en.wikipedia.org/wiki/Ray_tracing_%28graphics%29#/media/File:Glasses_800_edit.png

Summary from Last Time

- We used Quad Trees and k-D Trees to organize multidimensional points of data (ie (x,y) coordinates)
- To find points in either data structure, we could search in the same way we would search a BST, with small tweaks to handle >1 dimension
  - Quad Trees have 4 children per node to handle 2 dimensions
  - k-D Trees still have 2 children per node, but alternate which dimension is used to partition the data at each level of the tree
- Basic searching was therefore $\boldsymbol{O(d)}$ ($\boldsymbol{O(\log(n))}$ if trees are balanced)

Summary from Last Time

- We also looked at more complex questions we could as: what points fall within a given range, what points are closest to a target point
- **Common theme: prune the search space as much as we can**
  - In both cases we could come up with an $\boldsymbol{O(1)}$ check to determine whether we needed to explore a subtree further
  - This means in some scenarios we can ignore large parts of the tree
- Depending on the data and structure of the tree, these searches can be done in $\boldsymbol{O(\log(n))}$ if they can ignore significant parts of the tree

Summary from Last Time

- We also looked at more complex questions we could as: what points fall within a given range, what points are closest to a target point
- <u>**Common theme: prune the search space as much as we can**</u>
  - In both cases we could come up with an $O(1)$ check to determine whether we needed to explore a subtree further
  - This means in som
- Depending on the dat

done in $O(\log(n))$ if they can ignore significant parts of the tree

We'll see this more today as well!

Other Problems: N-Body Problem

What if we want to compute interactions between one body and every other body? How long would we expect that to take?

Other Problems: N-Body Problem

What if we want to compute interactions between one body and every other body? How long would we expect that to take?

Naively, this would take $O(n^2)$...but likely we don't care as much about interactions with bodies that are very very far away.

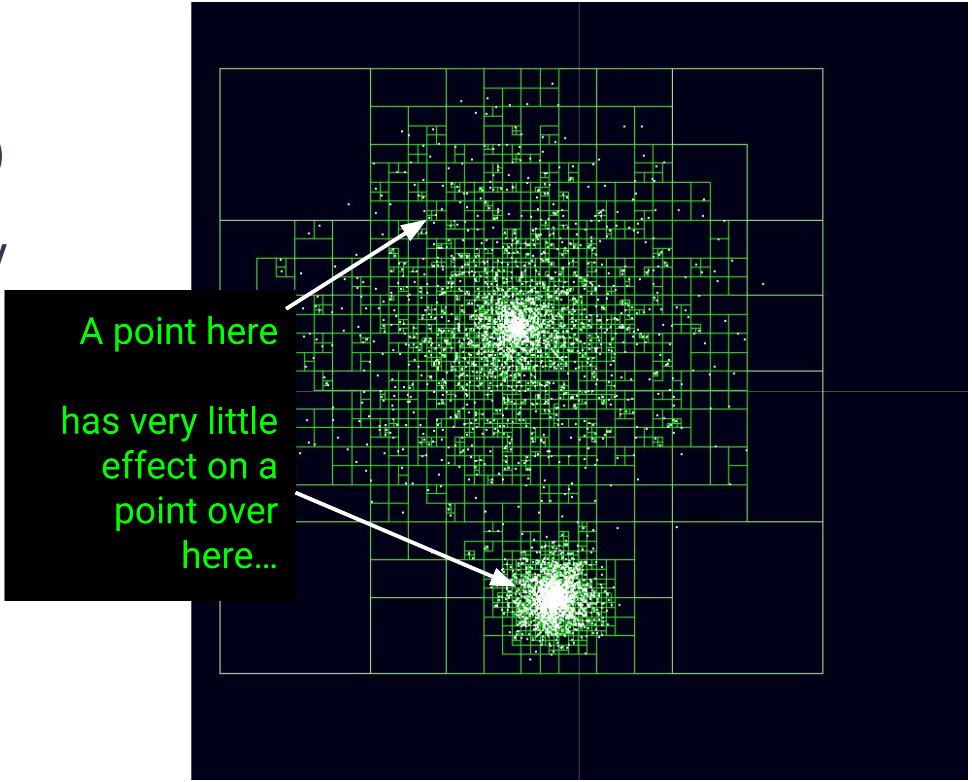
Other Problems: N-Body Problem

Idea: Divide our points into a quadtree (or octree in 3 dimensions)

Do full calculation for points closeby (in the same box)

Compute a summary (ie total force and center of mass) for each box that can be applied to far away boxes

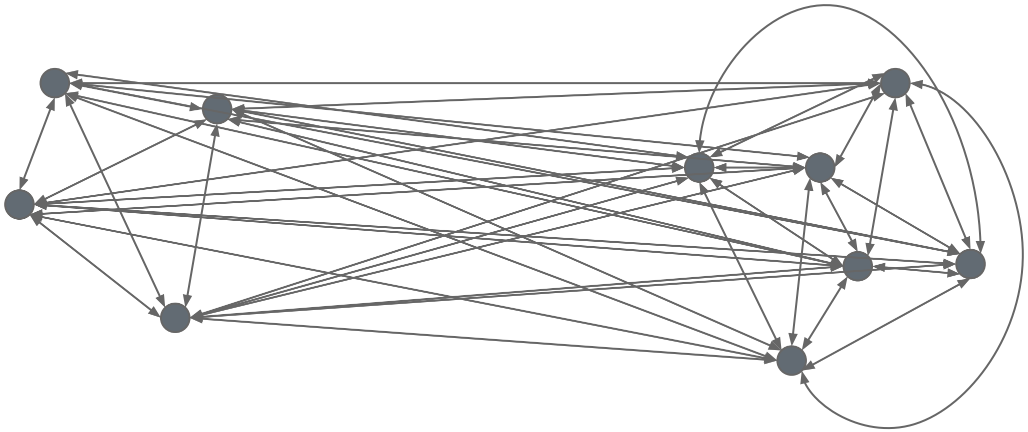
Target runtime: ~***O*(*n*log(*n*))**



Example

This diagram contains 10 bodies interacting with one another…

***O*(*n*^{2}) = ~**100 interactions (arrows)

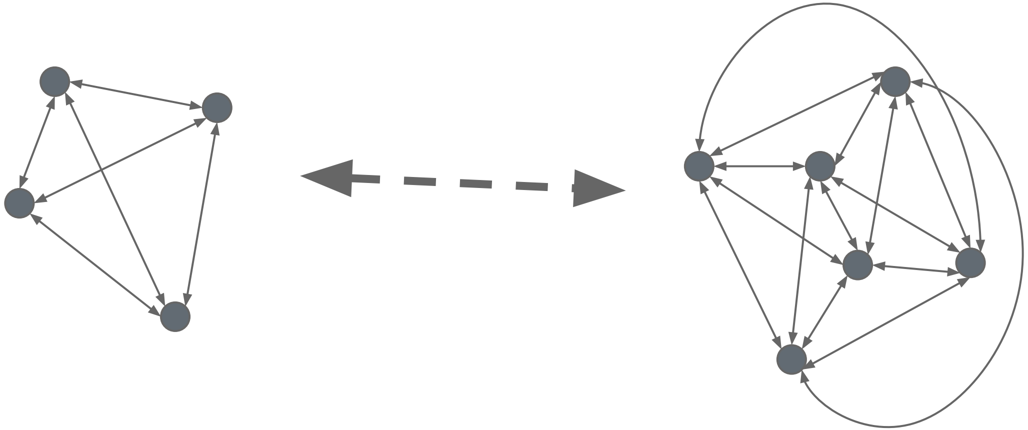


Example

This diagram contains 10 bodies interacting with one another…

***$O(n^2)$* = ~**100 interactions (arrows)

Idea: Estimate the interactions between far away points



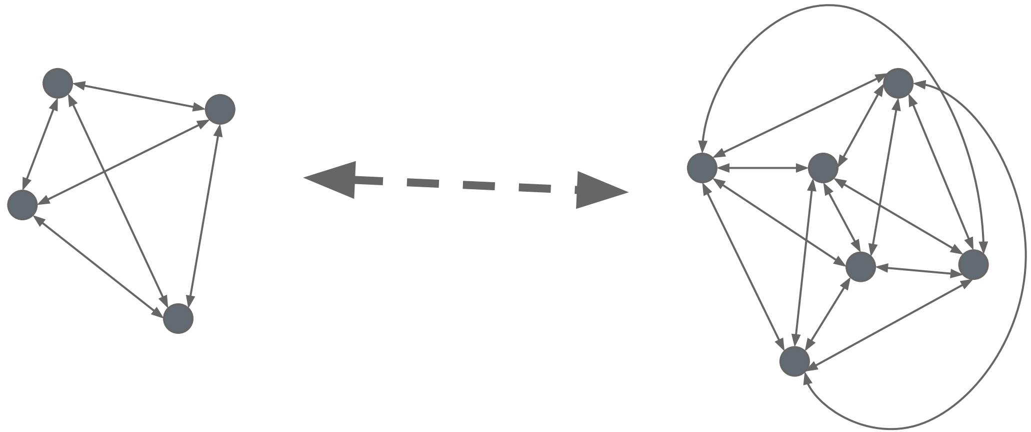
Example

This diagram contains 10 bodies interacting with one another…

***O*(n^2) = ~**100 interactions (arrows)

How can we do this systematically?

Idea: Estimate the interactions between far away points



Quad/Oct Trees Revisited

Idea: Let's organize the data (spatially) in a tree structure

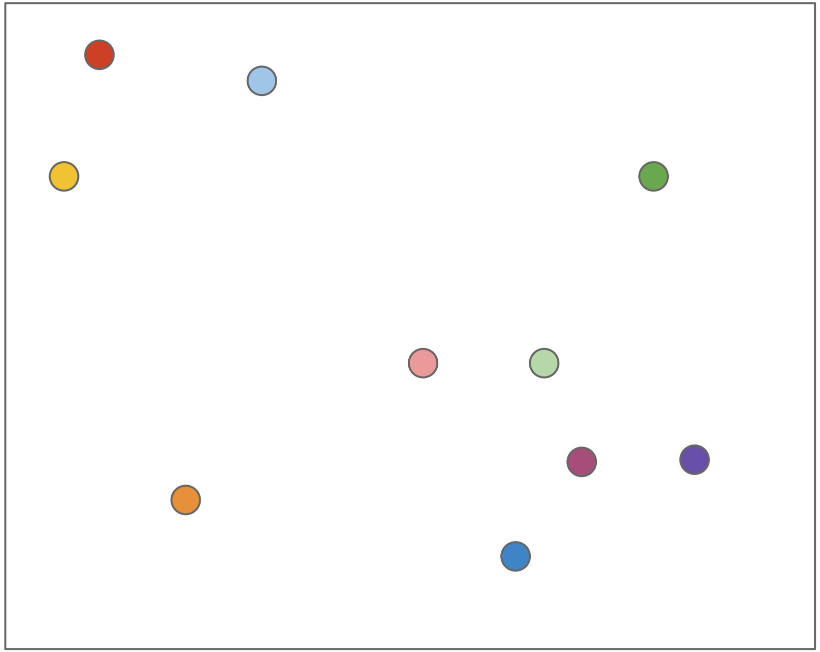
- 2D space → use a quad tree
- 3D space → use an oct tree (each node has at most 8 children)

Unlike last time, let's partition the space we are simulating, rather than the points in the space

Space Partitioning - 2D Example

Create a quad-tree by recursively partitioning the space

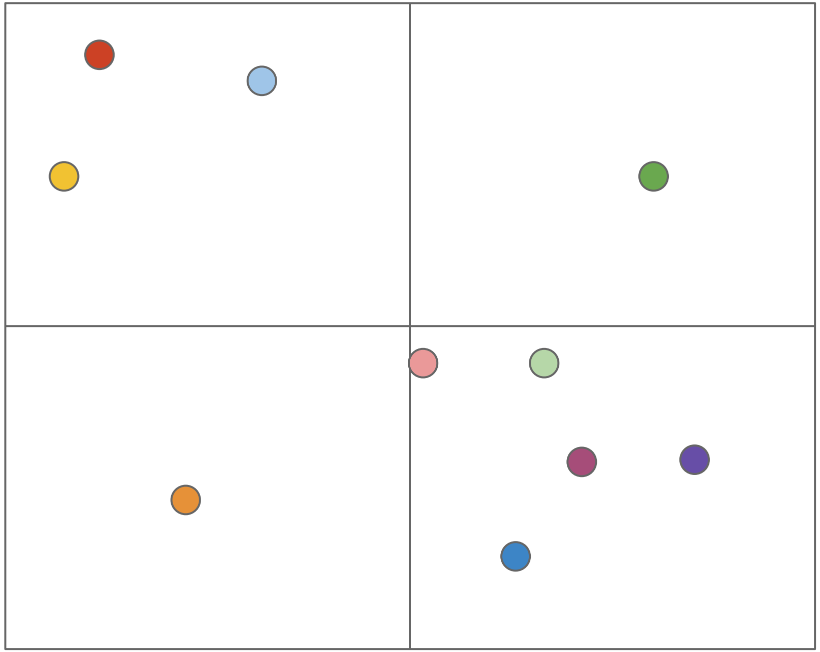
- Divide the space evenly until there is only one element per partition
- Internal tree nodes represent the partitions, leaves are the actual elements



Space Partitioning - 2D Example

Create a quad-tree by recursively partitioning the space

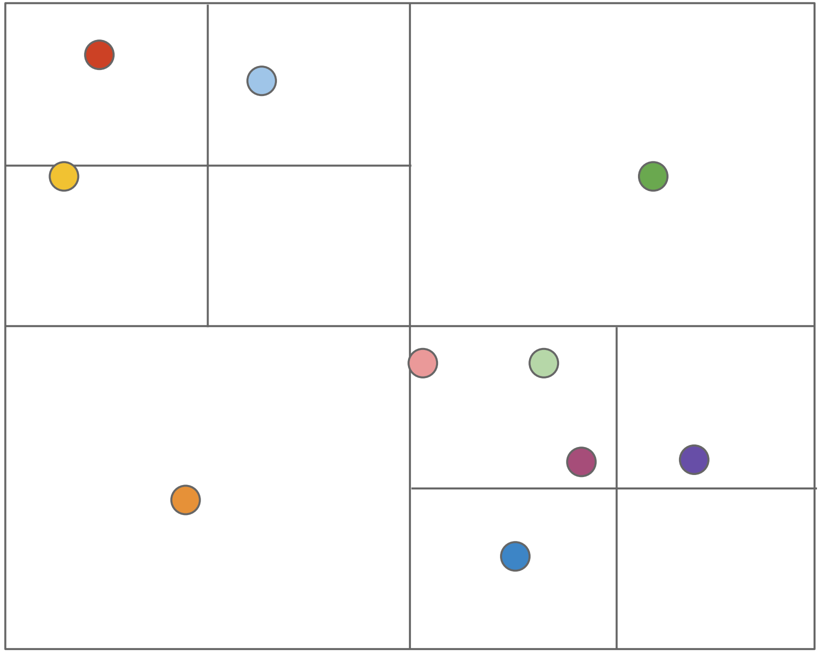
- Divide the space evenly until there is only one element per partition
- Internal tree nodes represent the partitions, leaves are the actual elements



Space Partitioning - 2D Example

Create a quad-tree by recursively partitioning the space

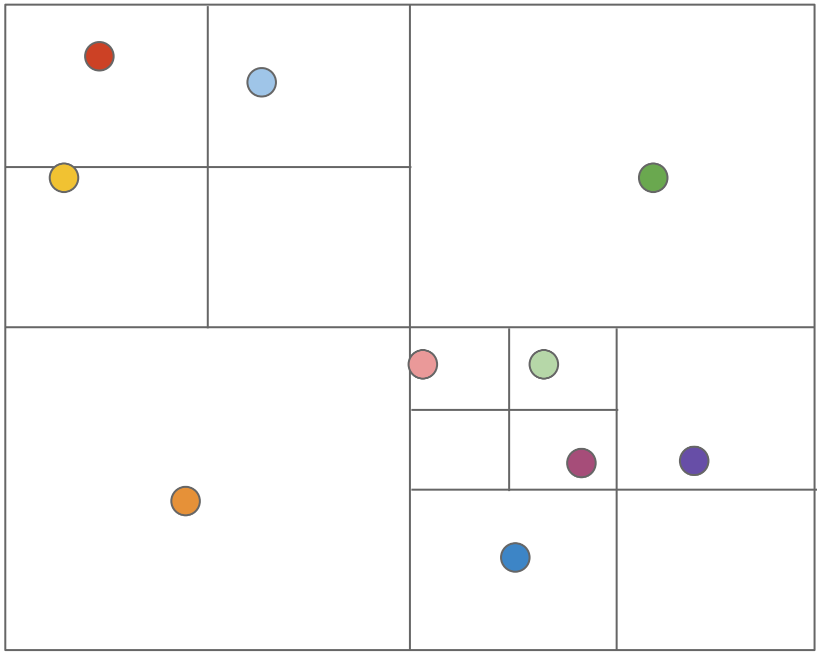
- Divide the space evenly until there is only one element per partition
- Internal tree nodes represent the partitions, leaves are the actual elements



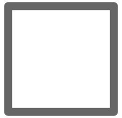
Space Partitioning - 2D Example

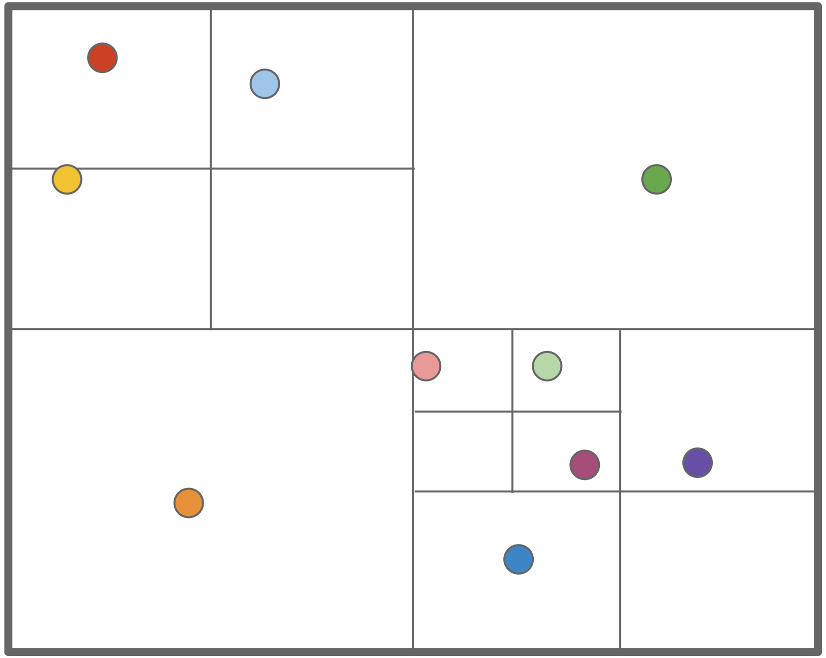
Create a quad-tree by recursively partitioning the space

- Divide the space evenly until there is only one element per partition
- Internal tree nodes represent the partitions, leaves are the actual elements

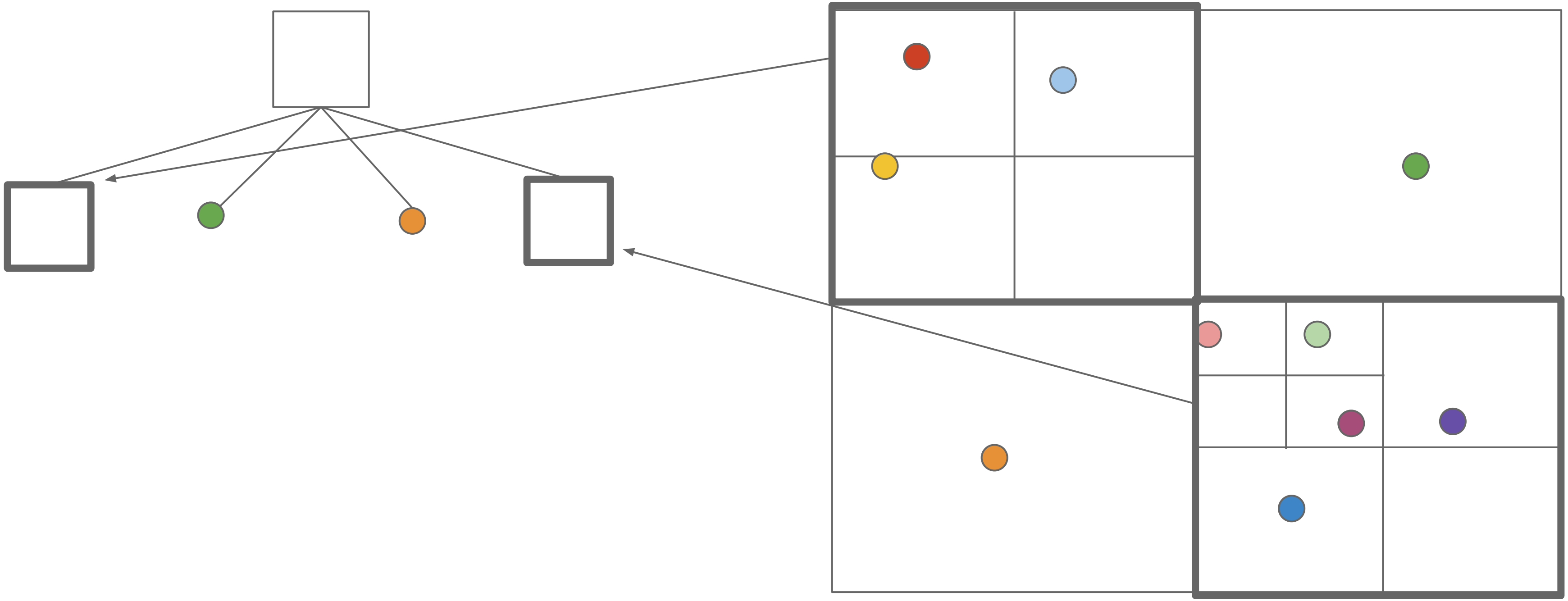


Space Partitioning - 2D Example

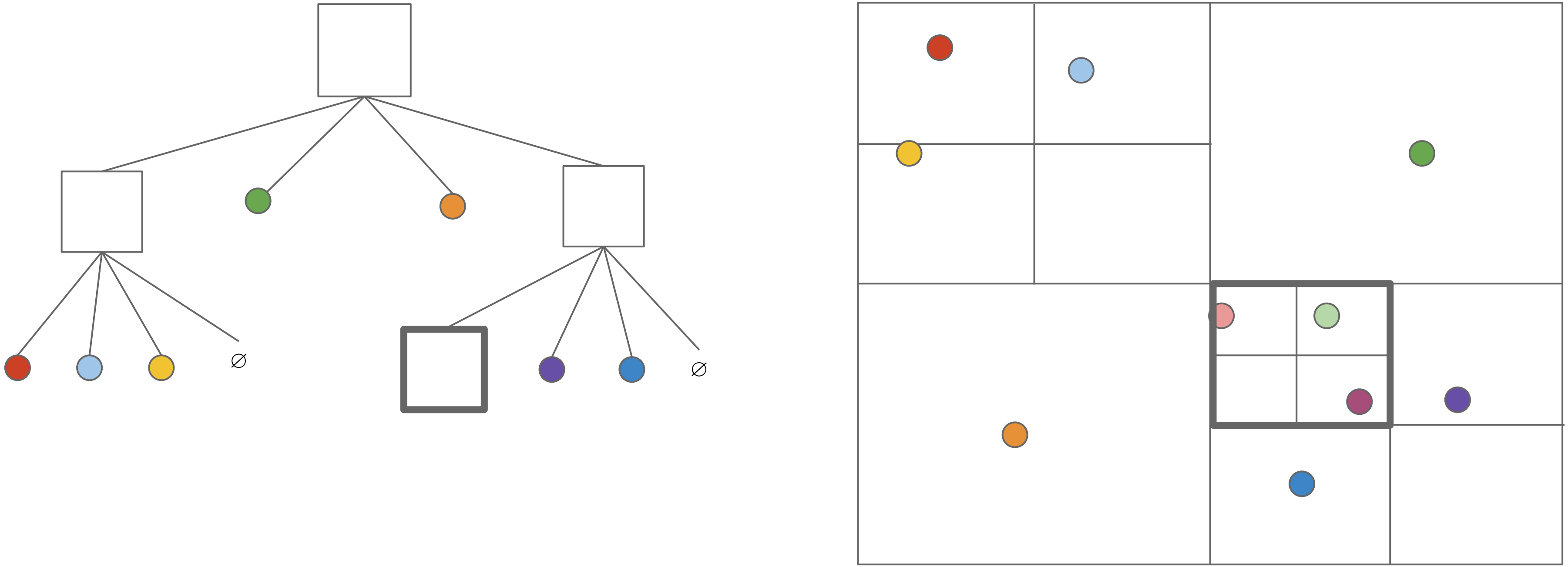




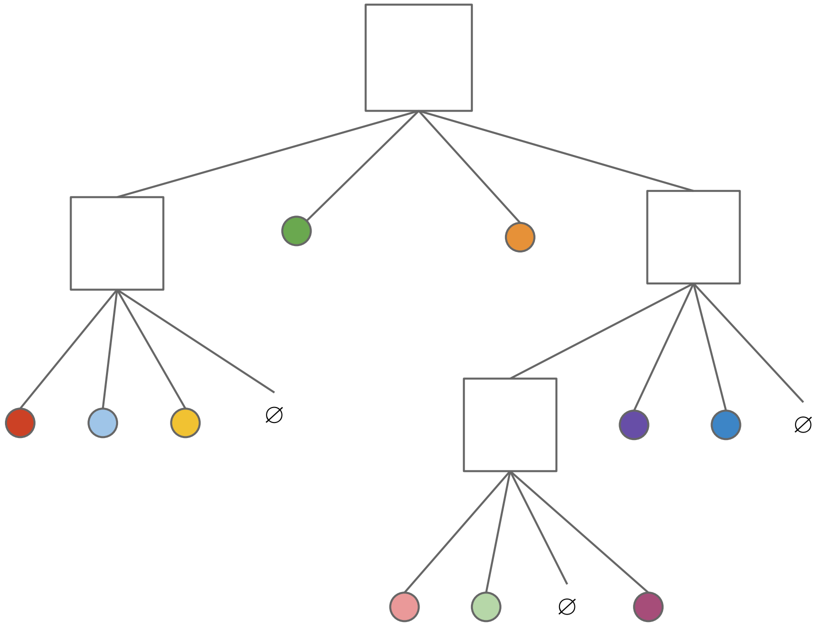
Space Partitioning - 2D Example

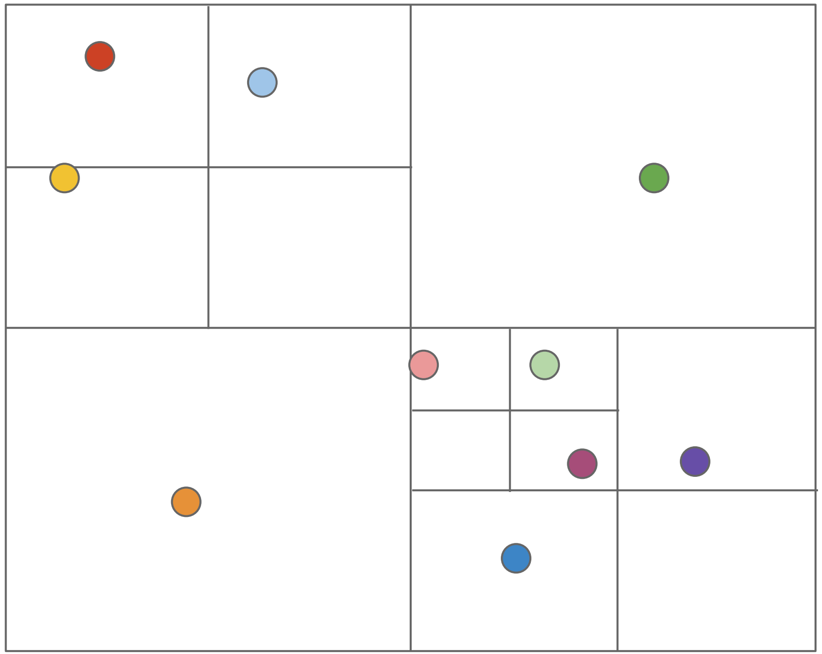


Space Partitioning - 2D Example

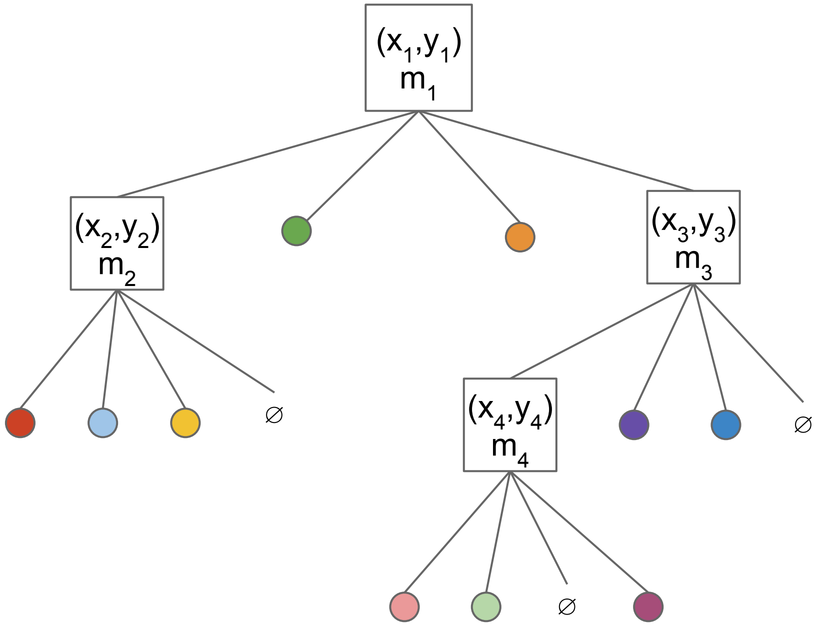


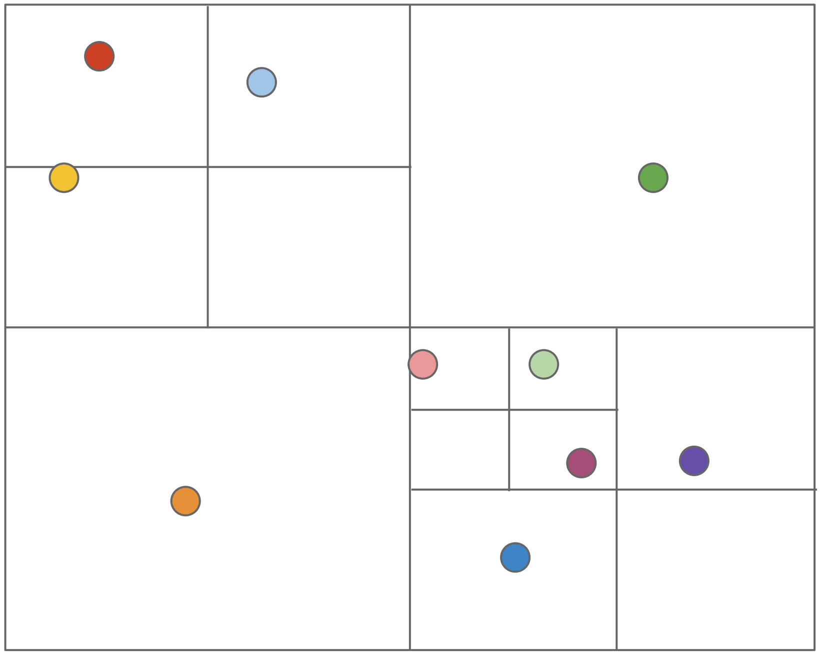
Space Partitioning - 2D Example





Space Partitioning - 2D Example





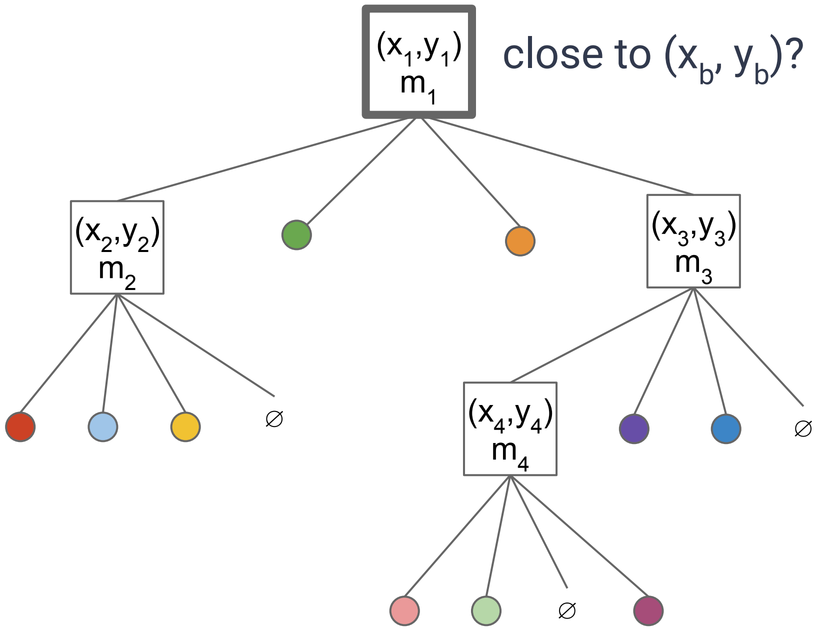
For each internal node, we can compute the center of mass and total mass

Barnes-Hut Algorithm (simplified)

Now to use the tree:

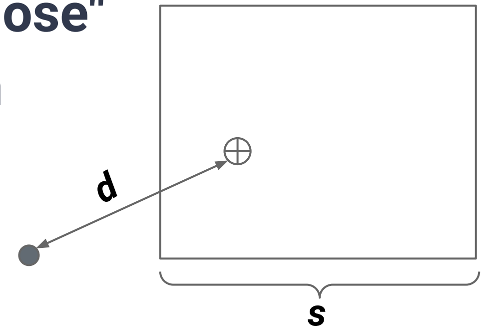
For a body with coordinates (x_b, y_b):

1. Start at the root
2. If (x_1, y_1) is "far" from (x_b, y_b) then just treat it as a single body with mass m_1, no need to "open the box"
3. If it is "close", then repeat this process with the children…What's in the box?



Barnes-Hut Algorithm (simplified)

So what is considered "far", and what is considered "close"



- Find the ratio ***s*/*d*** where ***s*** is the width of the region in question and ***d*** is the distance from the body to the center of mass of that region
- Pick a threshold, θ
    - If ***s*/*d*** > θ then we are close enough to check children in more detail
    - If ***s*/*d*** < θ then we are far away and can treat the region as a single body
- Larger θ means more fudging the numbers, but faster execution (**~*O*(n log n)** to process all ***n*** bodies)
- θ = 0 means finding an exact answer, but at a cost of $\boldsymbol{O(n^2)}$

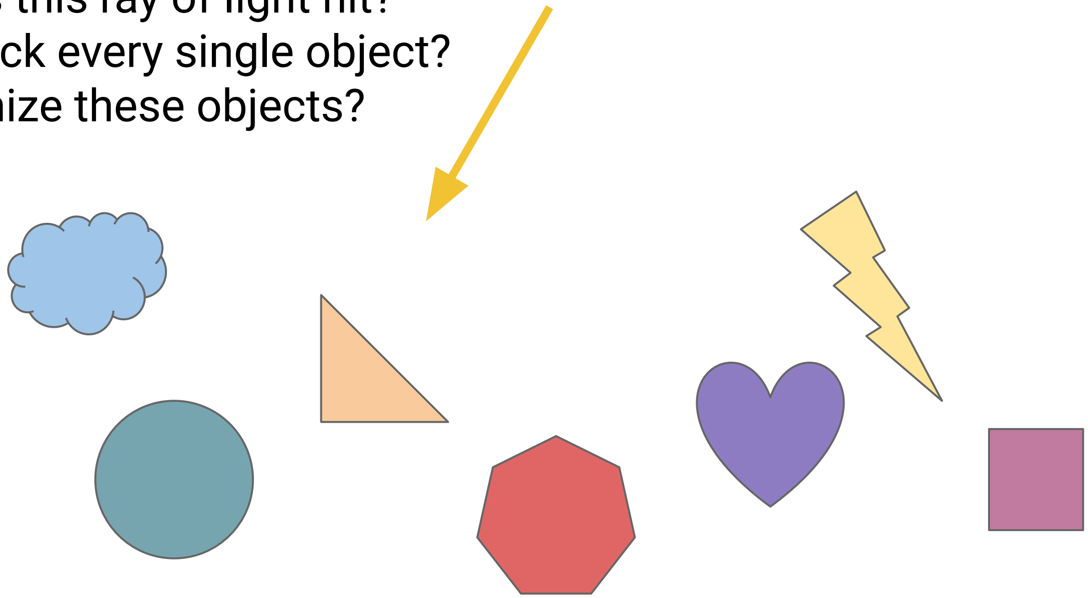
Trees as a Hierarchy

In the n-body problem, we used a tree to ***hierarchically*** organize our data

- When using this hierarchy, for each internal node we could decide whether or not to explore further with a very cheap $\boldsymbol{O(1)}$ check
  - This allows us to avoid checking all ***n*** elements in a systematic fashion
  - We saw a similar idea with **`range()`** and **`nearestNeighbor()`** last time
- This style of algorithm has other applications as well

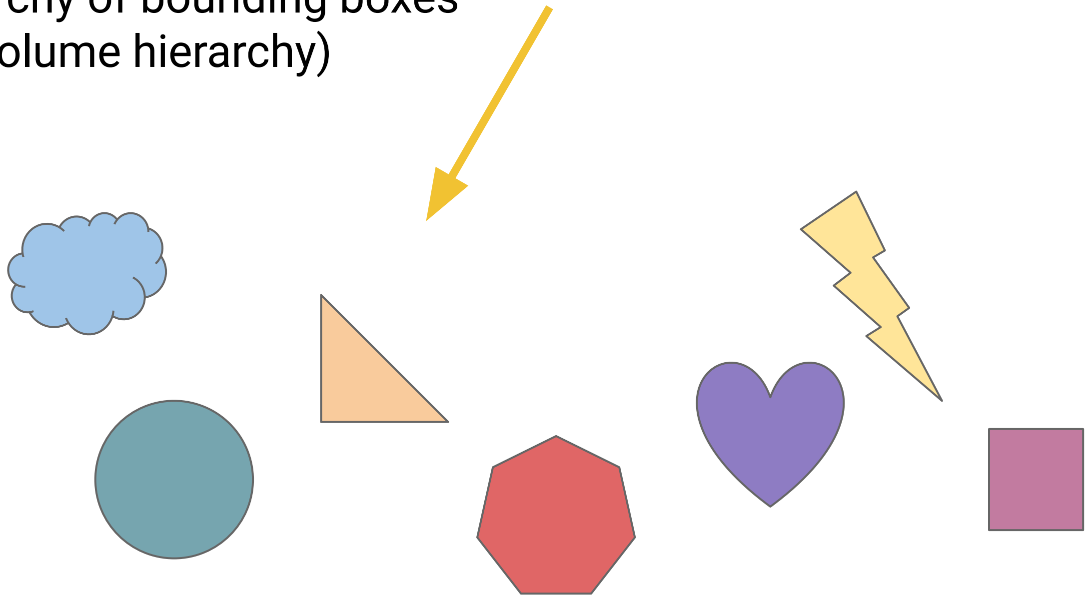
Other Problems: Ray/Path Tracing

Which object does this ray of light hit?
Do we have to check every single object?
How can we organize these objects?



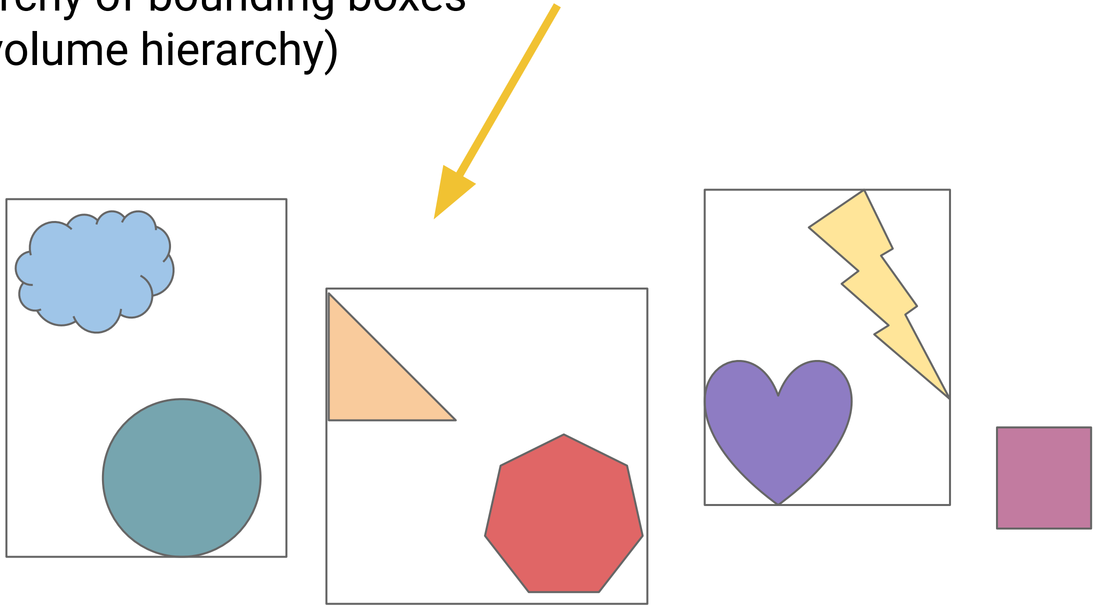
Other Problems: Ray/Path Tracing

Idea: Build a hierarchy of bounding boxes
(BVH - Bounding volume hierarchy)



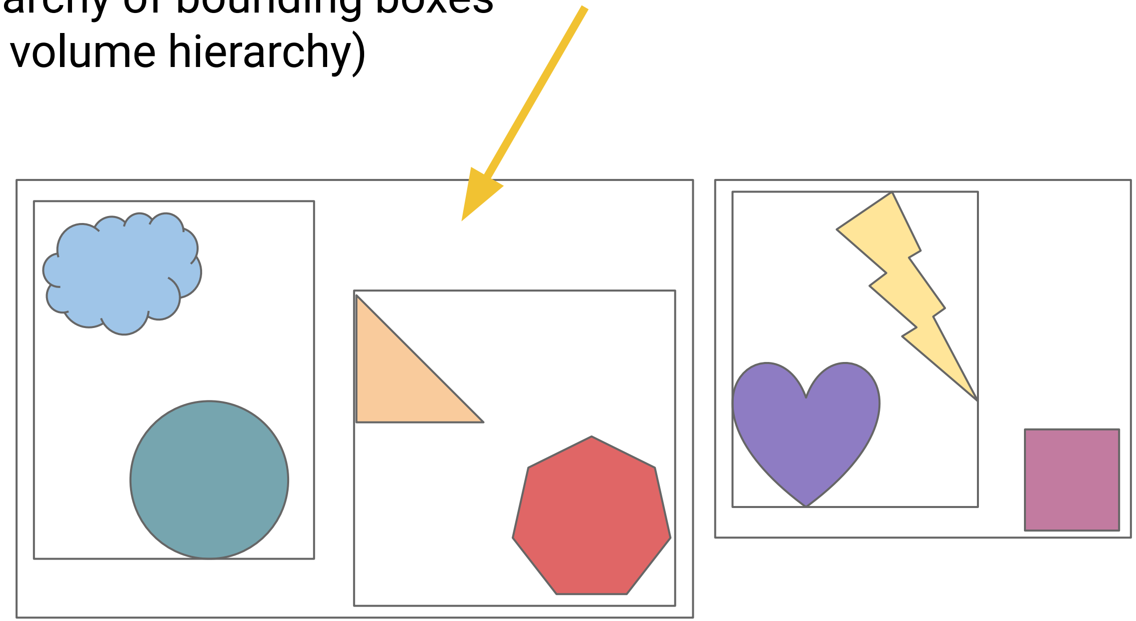
Other Problems: Ray/Path Tracing

Idea: Build a hierarchy of bounding boxes
(BVH - Bounding volume hierarchy)



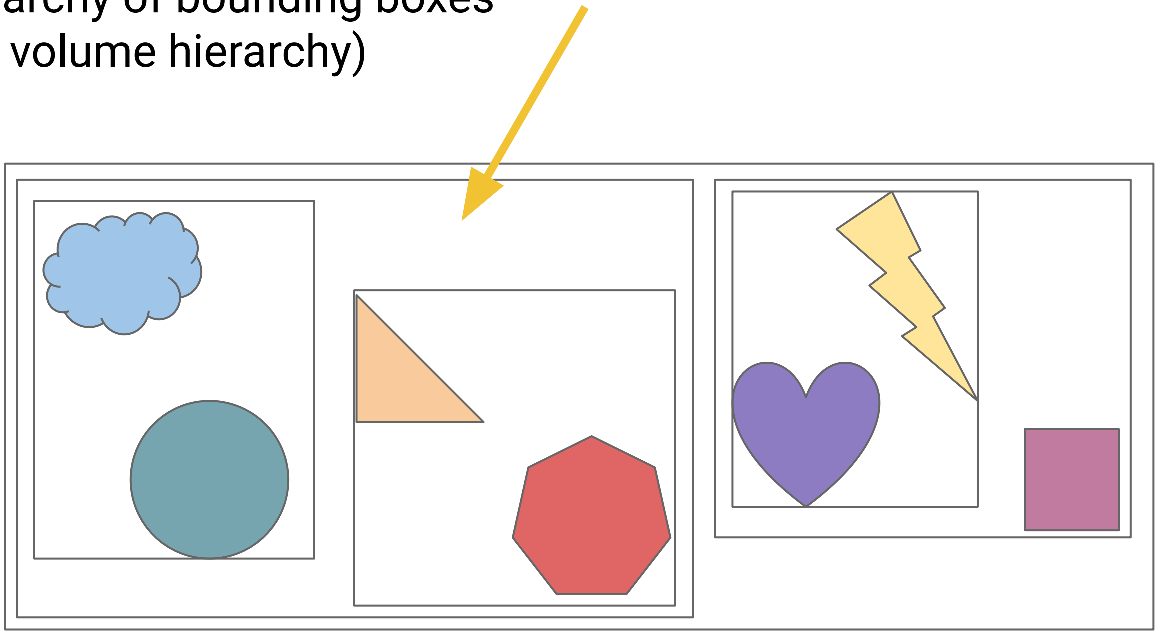
Other Problems: Ray/Path Tracing

Idea: Build a hierarchy of bounding boxes
(BVH - Bounding volume hierarchy)



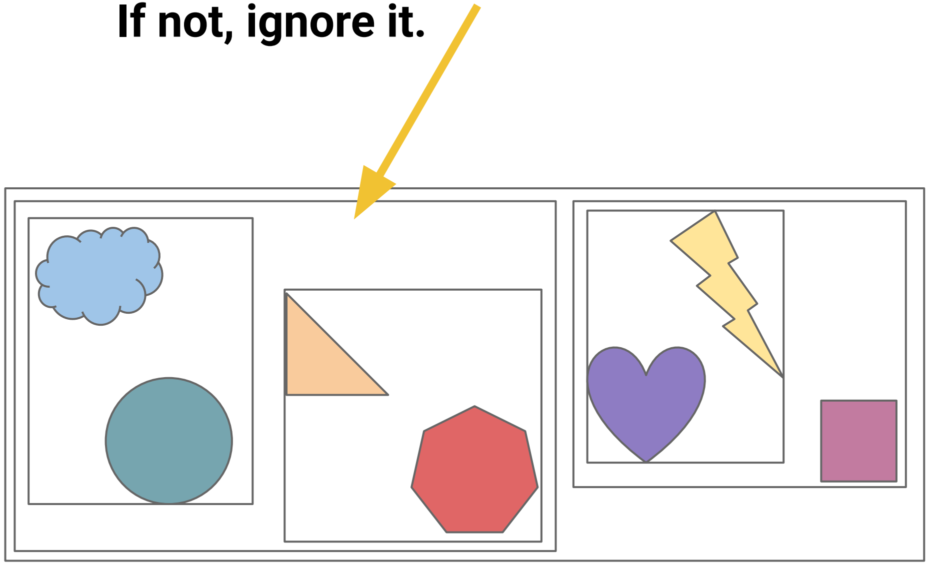
Other Problems: Ray/Path Tracing

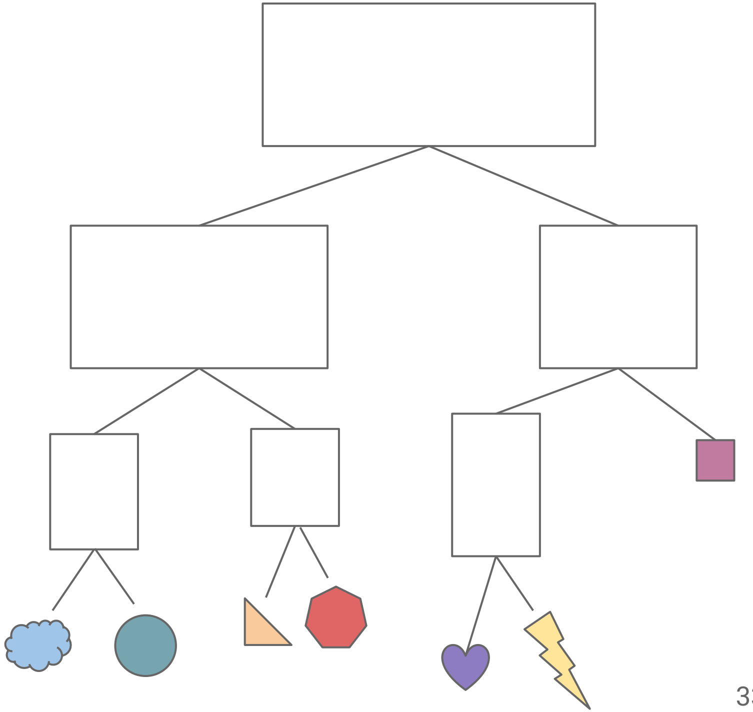
Idea: Build a hierarchy of bounding boxes
(BVH - Bounding volume hierarchy)



Other Problems: Ray/Path Tracing

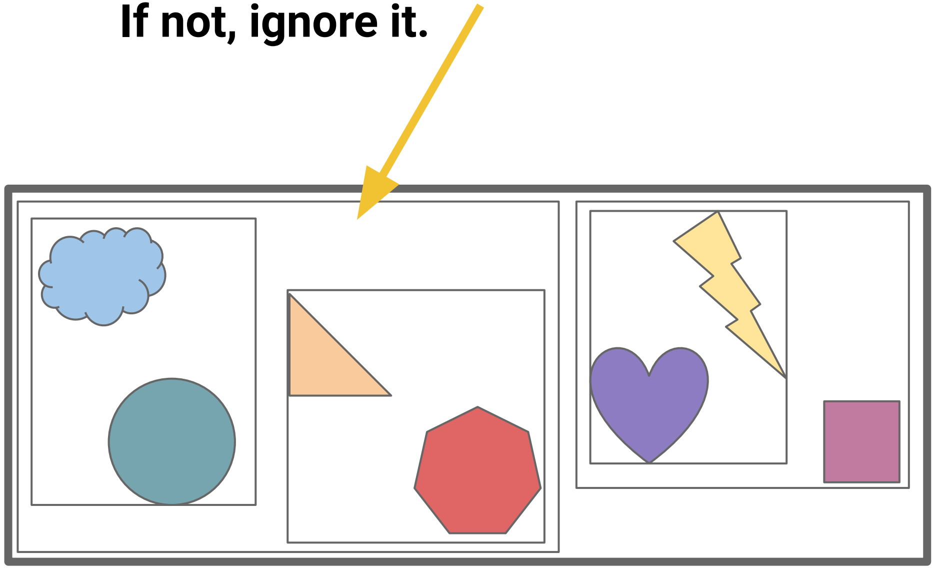
These bounding boxes form a tree…
We can check if the ray intersects a bounding box.
- **If it does, explore its children.**
- **If not, ignore it.**

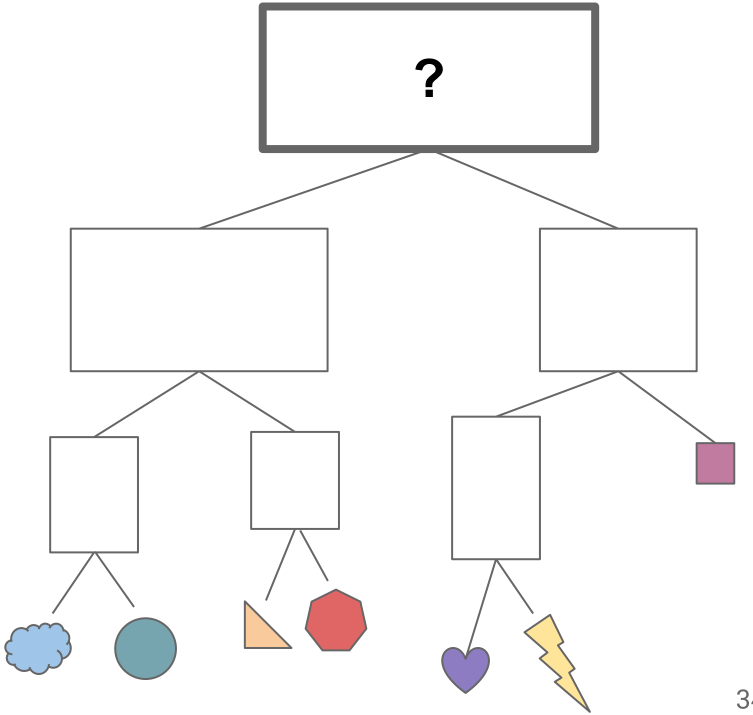




Other Problems: Ray/Path Tracing

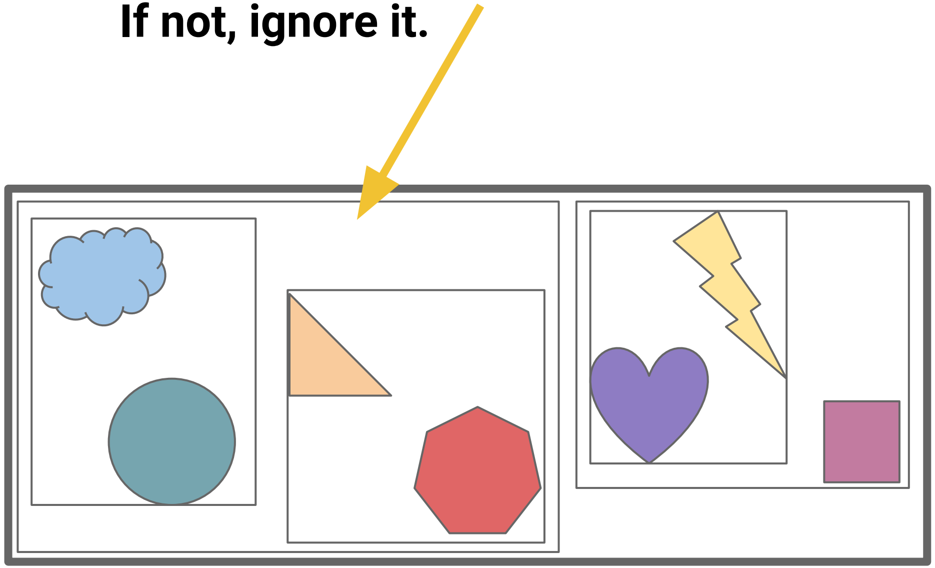
These bounding boxes form a tree…
We can check if the ray intersects a bounding box.
If it does, explore its children.
If not, ignore it.

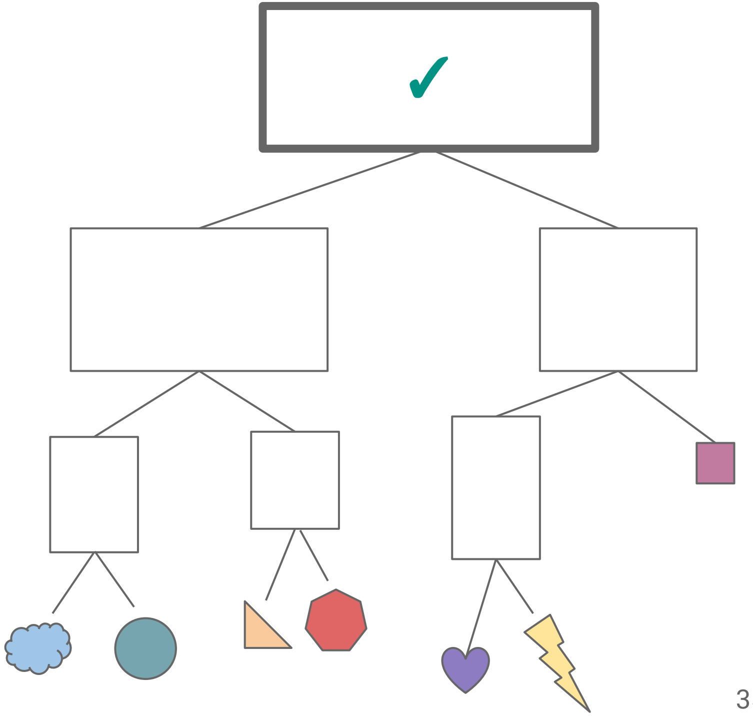




Other Problems: Ray/Path Tracing

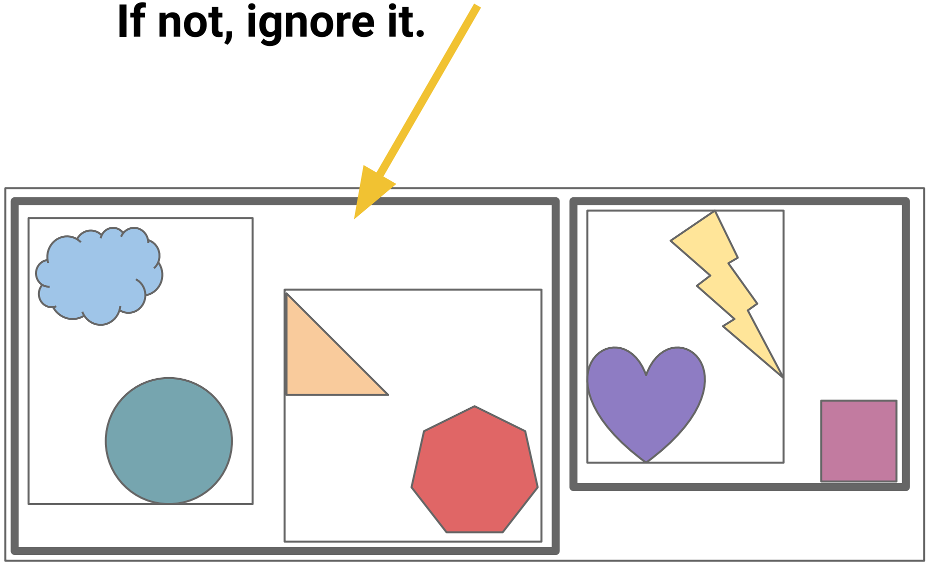
These bounding boxes form a tree…
We can check if the ray intersects a bounding box.
If it does, explore its children.
If not, ignore it.

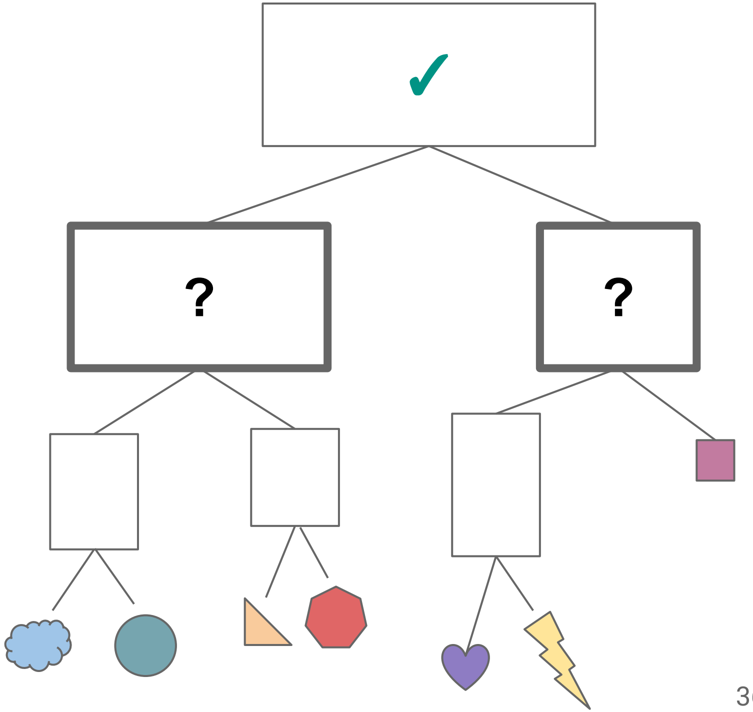




Other Problems: Ray/Path Tracing

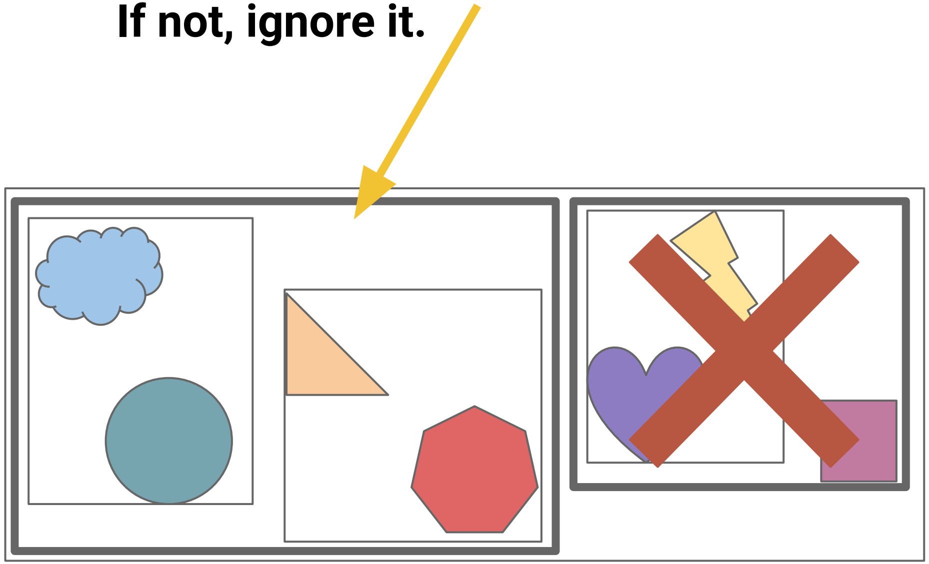
These bounding boxes form a tree…
We can check if the ray intersects a bounding box.
  If it does, explore its children.
  If not, ignore it.

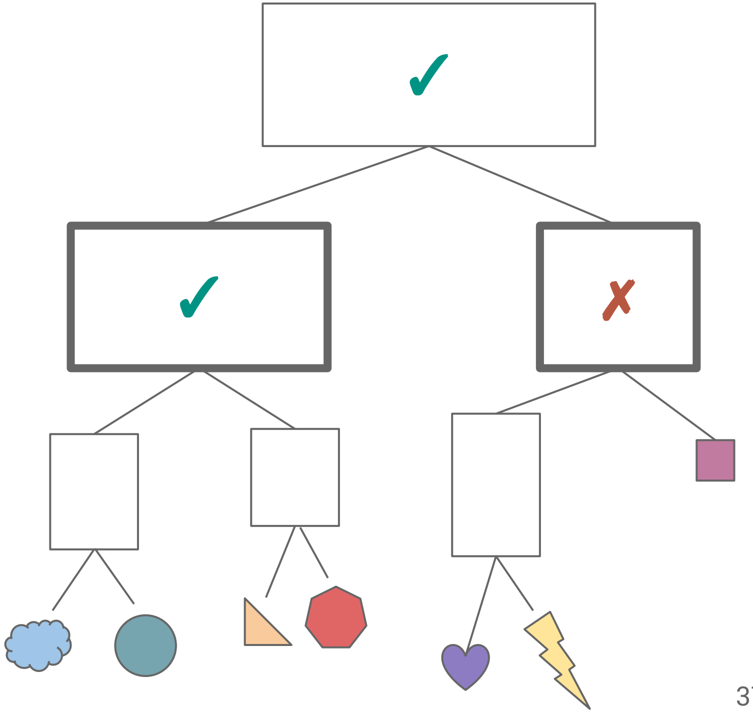




Other Problems: Ray/Path Tracing

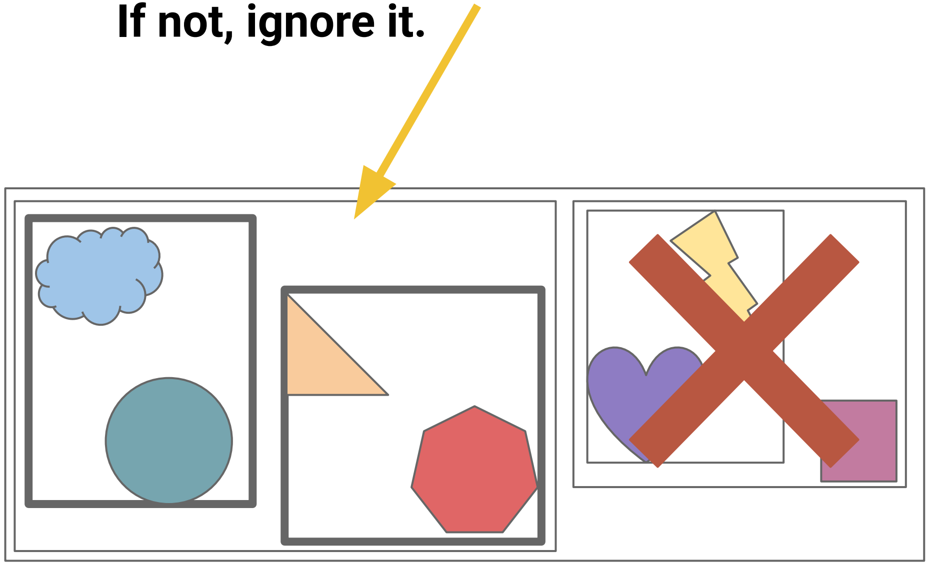
These bounding boxes form a tree…
We can check if the ray intersects a bounding box.
  If it does, explore its children.
  If not, ignore it.

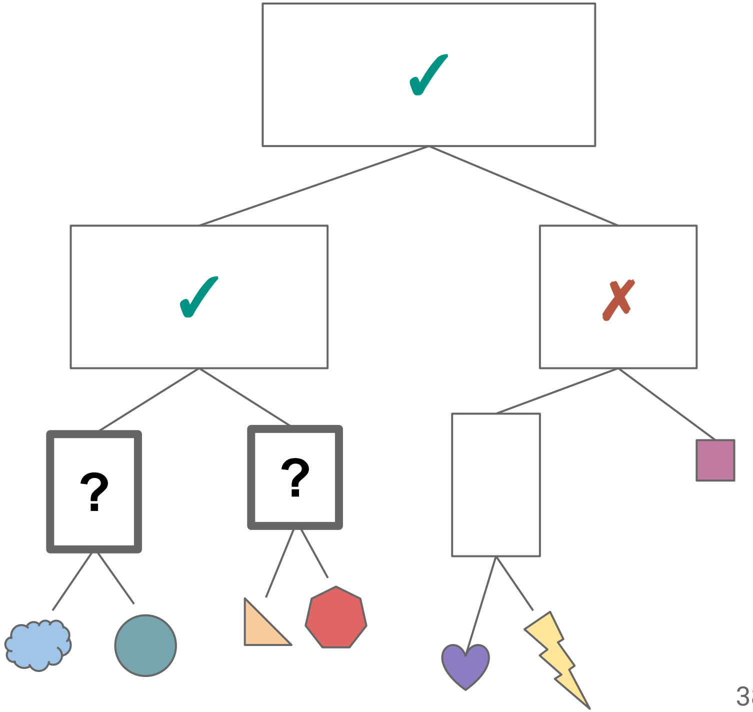




Other Problems: Ray/Path Tracing

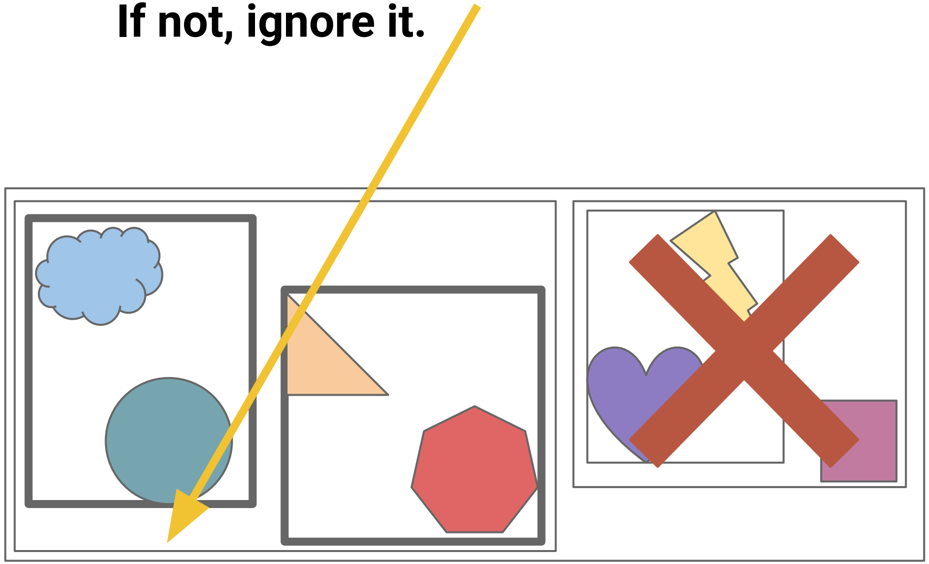
These bounding boxes form a tree…
We can check if the ray intersects a bounding box.
If it does, explore its children.
If not, ignore it.

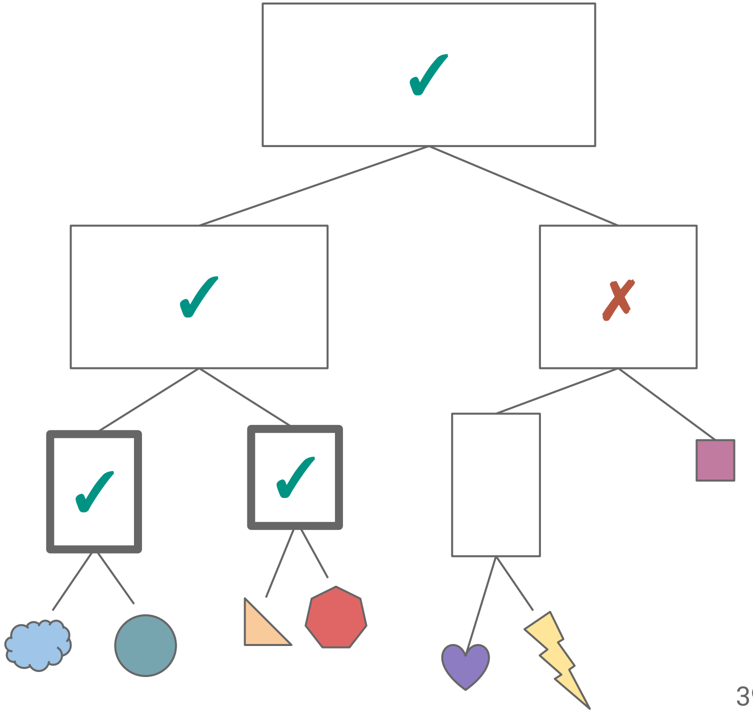




Other Problems: Ray/Path Tracing

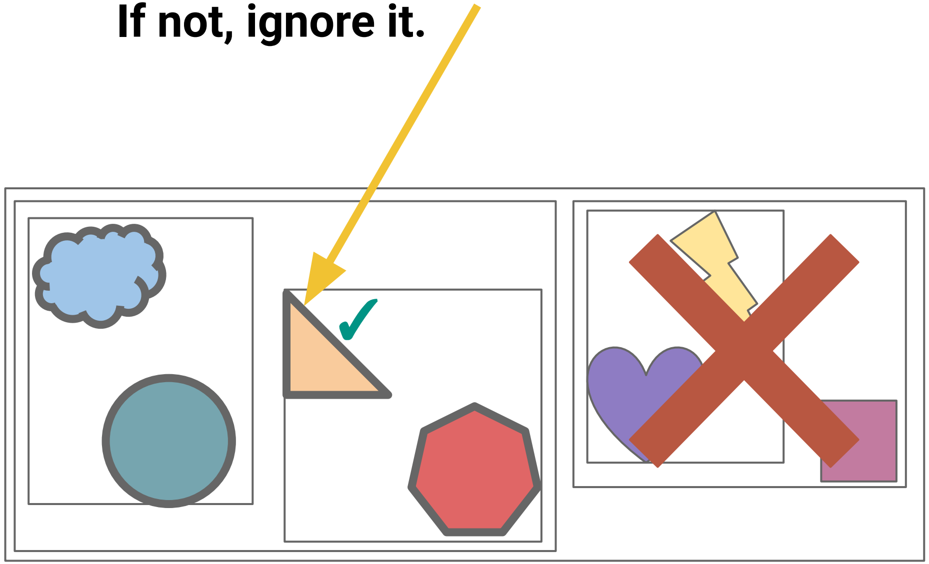
These bounding boxes form a tree…
We can check if the ray intersects a bounding box.
If it does, explore its children.
If not, ignore it.

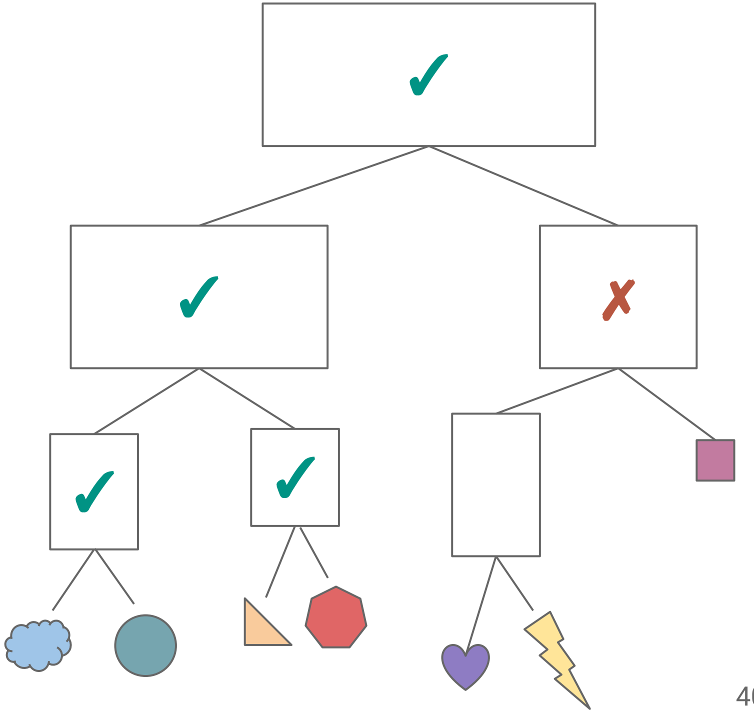




Other Problems: Ray/Path Tracing

These bounding boxes form a tree…
We can check if the ray intersects a bounding box.
  If it does, explore its children.
  If not, ignore it.





Other Problems: Ray/Path Tracing

- By using a ***bounding-volume hierarchy***, we can avoid checking all ***n*** objects for collisions
  - When we are projecting millions+ rays of light, this is a **huge** savings
- In practice, we hope to end up with a runtime of **~*O*(*m* log *n*)** where ***m*** is the number of rays and ***n*** is the number of objects
  - This depends on how effectively we can build our BVH
- In both ray tracing and Barnes-Hut, the exact structure of the hierarchy will vary based on the specific data we are using

Taking it a Step Further

The data in these problems can get **HUGE**…

What if it gets so big we can't fit all the data on one computer? Or even if we could, it would take forever to compute?

Taking it a Step Further

The data in these problems can get **HUGE**…

What if it gets so big we can't fit all the data on one computer? Or even if we could, it would take forever to compute?

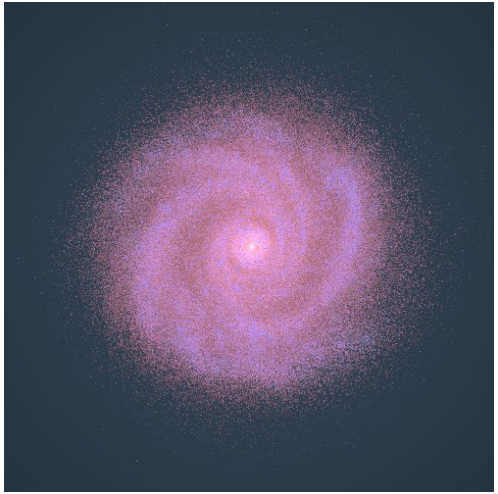
Distribute the data (and computation) across multiple computers!

An Example from my Past

ChaNGa (the Charm++ N-Body Gravity solver)

- Uses Barnes-Hut to simulate various cosmological phenomena
- Breaks up the Oct-Tree across multiple compute cores
- Has been run on at least 512,000 cores (as of 2015)

This image took 100,000 core-hours to simulate! →



This [video](http://faculty.washington.edu/trq/hpcc/homepage/picture/picture.html) simulated over 50 million particles

http://faculty.washington.edu/trq/hpcc/homepage/picture/picture.html

High-Level Summary

- We've seen both trees and hash tables as effective ways to organize our data if we know we are going to be searching it often
- **HashTables** can be great for exact lookups
  - Think PA3: you may want to lookup a person with an exact (bday, zipcode) pair, and HashTable lets you do that very fast
- **Trees** and tree like structures work very well for "fuzzier" searches
  - What is "close" to this point? What object might this projectile hit? etc
  - The input to your search is not necessarily an exact element in your tree, but the tree organizes the data in a way that effectively directs the search