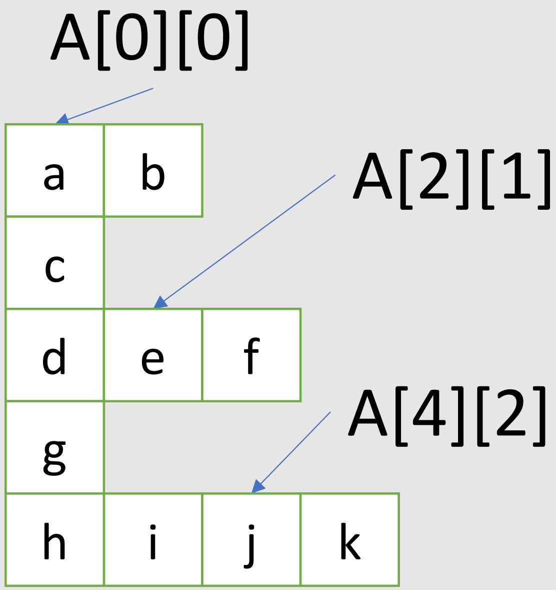
Jagged Array

Jagged Array: Irregular 2D array

- Rows of the array has variable sizes

A = [[a,b],[c],[d,e,f],[g],[h,i,j,k]]



Array of array is inefficient !

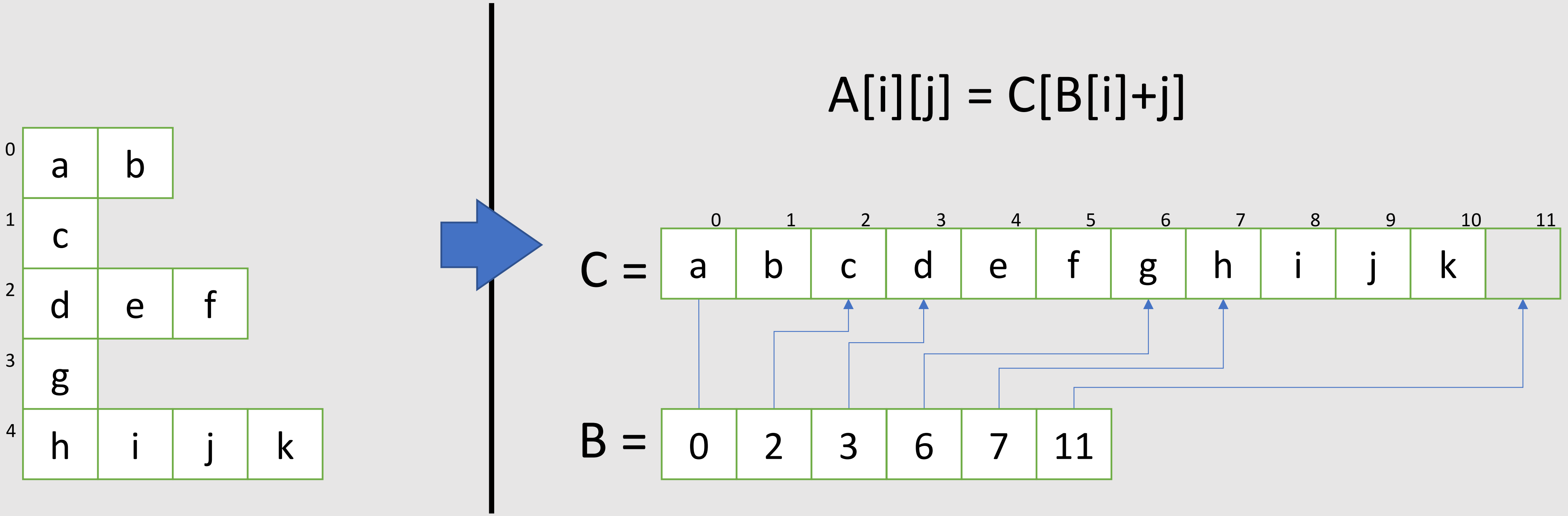
```
std::vector< std::vector<int> > arrayOfArray;
```

Jagged Array: Irregular 2D array

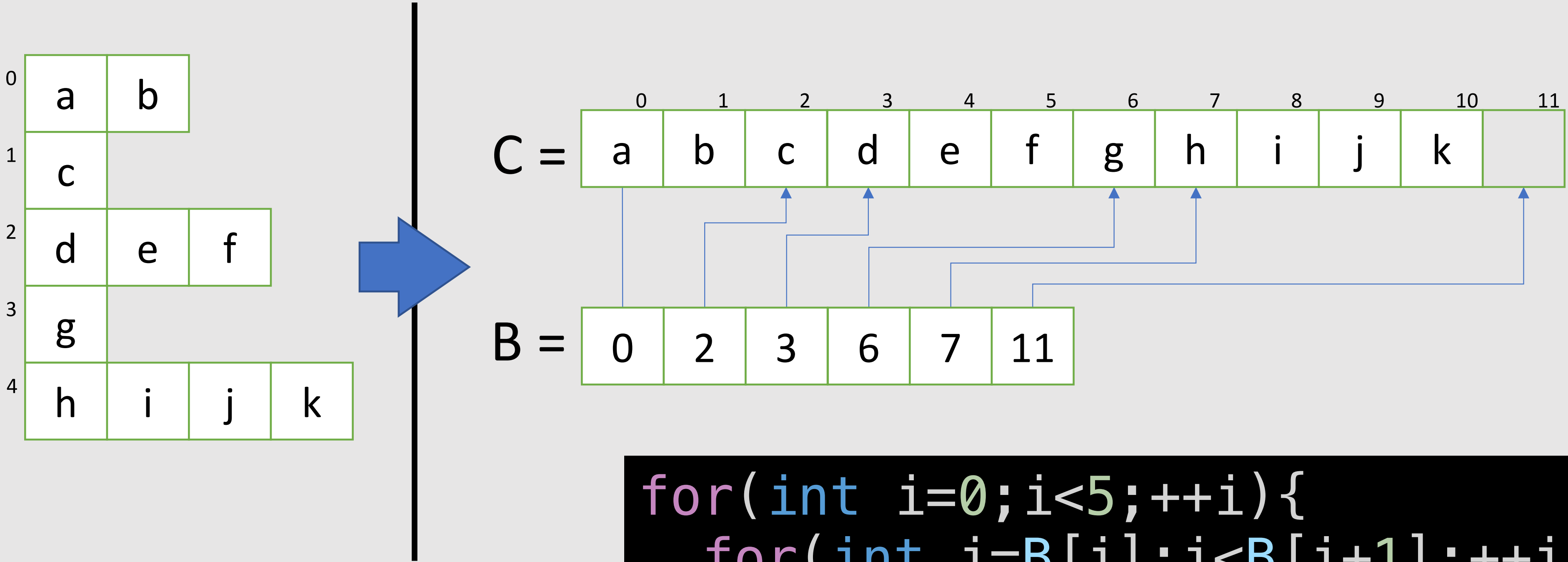
- A jagged array can be expressed by two 1D arrays

A = [[5,7],[1],[9,3,4],[3],[5,5,4,3]]

A[i][j] = C[B[i]+j]



Loop Over Jagged Array




```
for(int i=0;i<5;++i){
  for(int j=B[i];j<B[i+1];++j){
    float v = C[j];
  }
}
```

Collision Detection

衝突検出

Applications

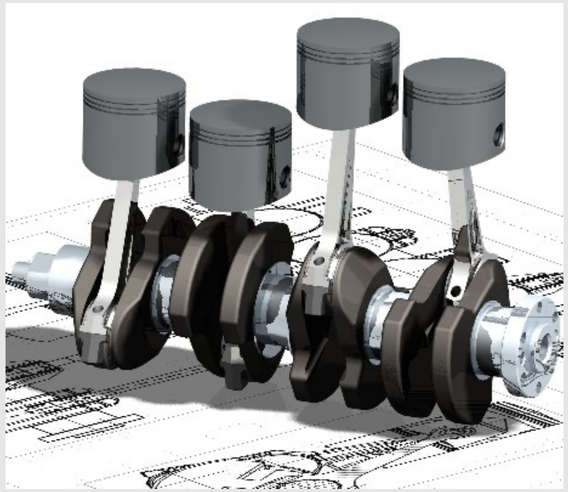
Computer Graphics

(Wikipedia)

Robotics

(Wikipedia)

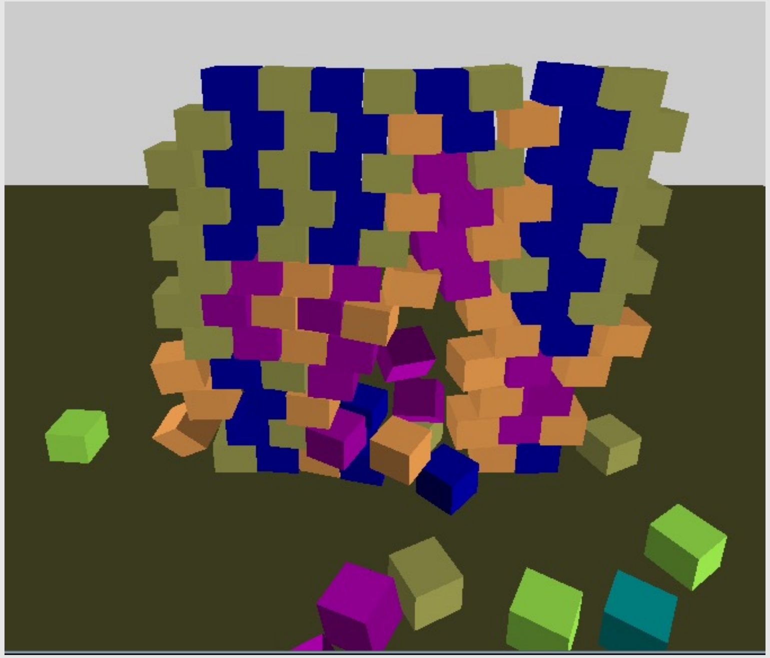
CAD



(Credit: freeformer @ Wikipedia)

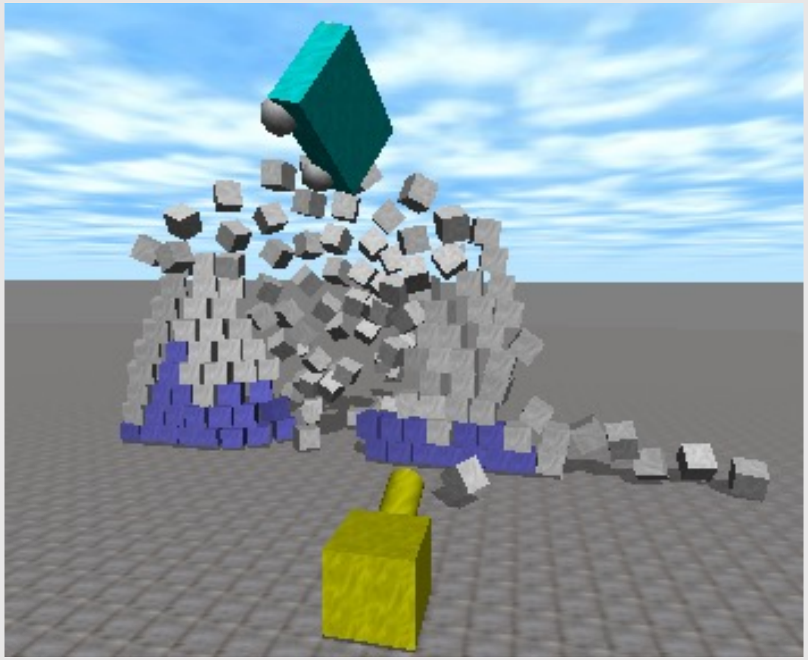
Popular Rigid Body Simulation Engine

Bullet



(Credit: SteveBaker at Wikipedia)

Open Dynamic Engine



(Credit: Kborer at Wikipedia)

Real-time Collision Detection using GPU

Vivace: a Practical Gauss-Seidel Method for Stable Soft Body Dynamics
Marco Fratarcangeli, Valentina Tibaldo, Fabio Pellacini
ACM Transactions on Graphics (SIGGRAPH Asia), 2016
http://www.cse.chalmers.se/~marcof

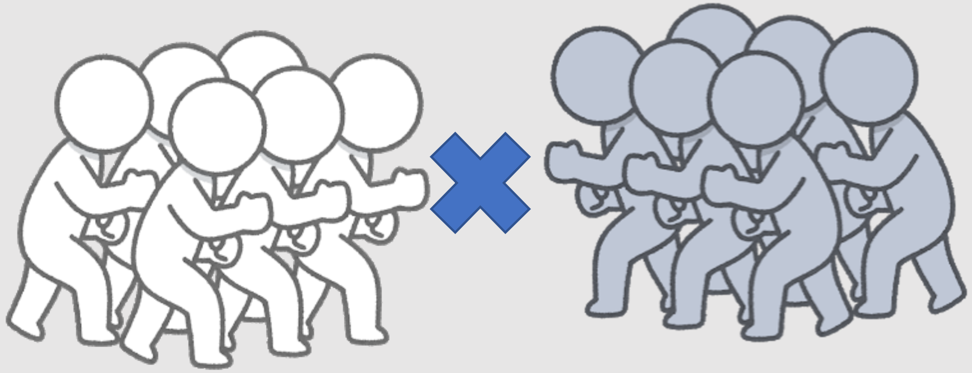
Brute-force Collision Detection Never Works

- If there are N objects, there are N(N-1)/2 number of pair

→ $\mathcal{O}(N^2)$ complexity is too slow!

$\mathcal{O}(N)$

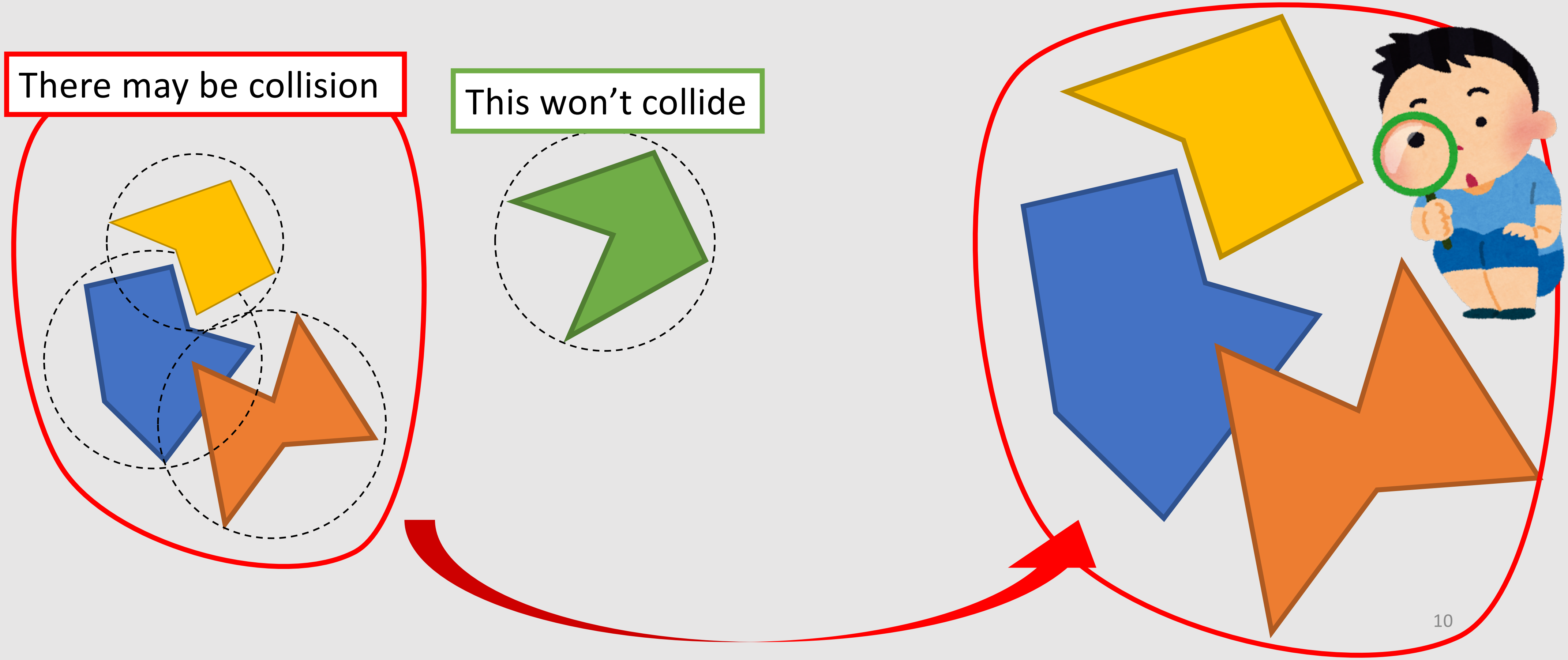
$\mathcal{O}(N^2)$



Collision Detection in Two Stages

Broad Phase: extract candidate

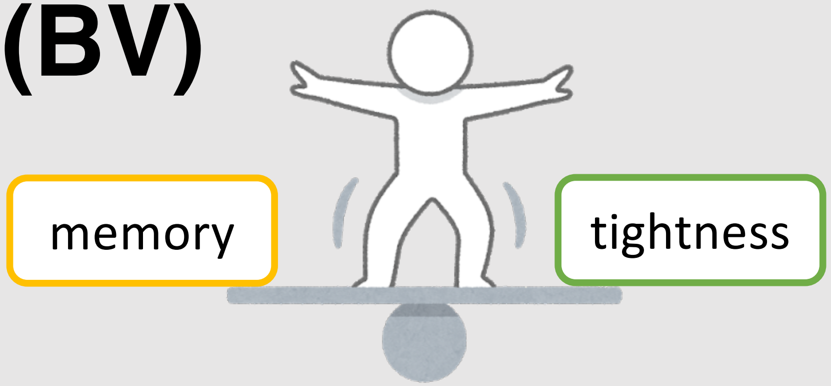
Narrow Phase: actual check

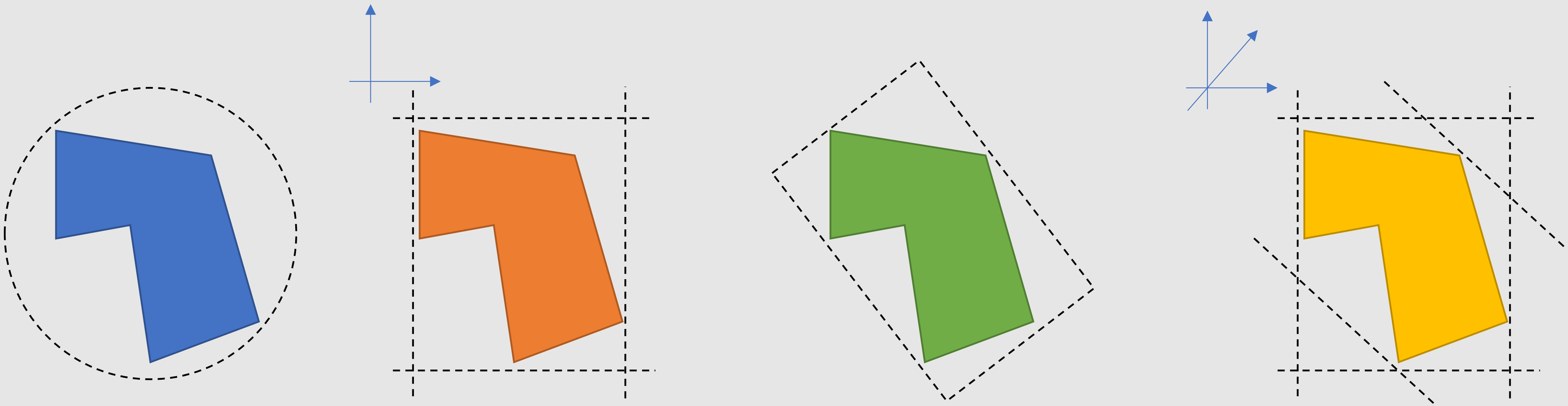


Idea of Finding Collision (like a Garimpeiro)

Types of Bounding Volume (BV)

- Easy evaluation (convex shape!)
- Tightly fit to object's shape
- Low memory footprint





Sphere

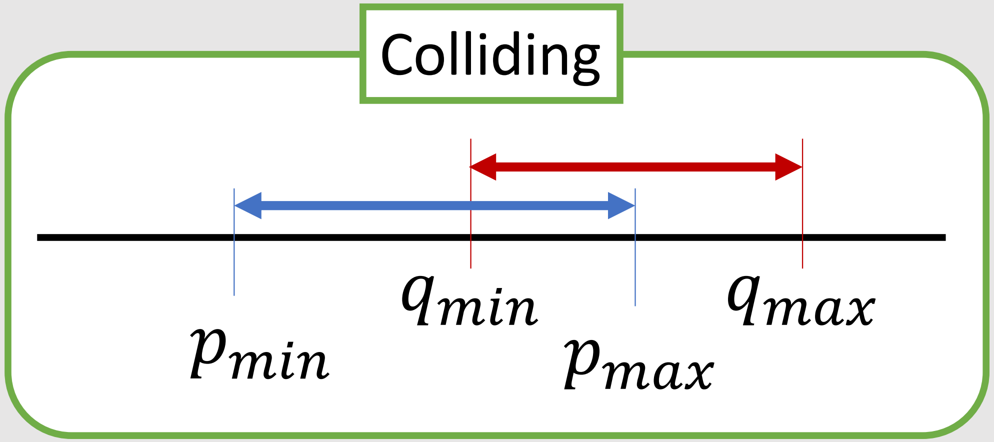
AABB
Axis-Aligned Bounding Box

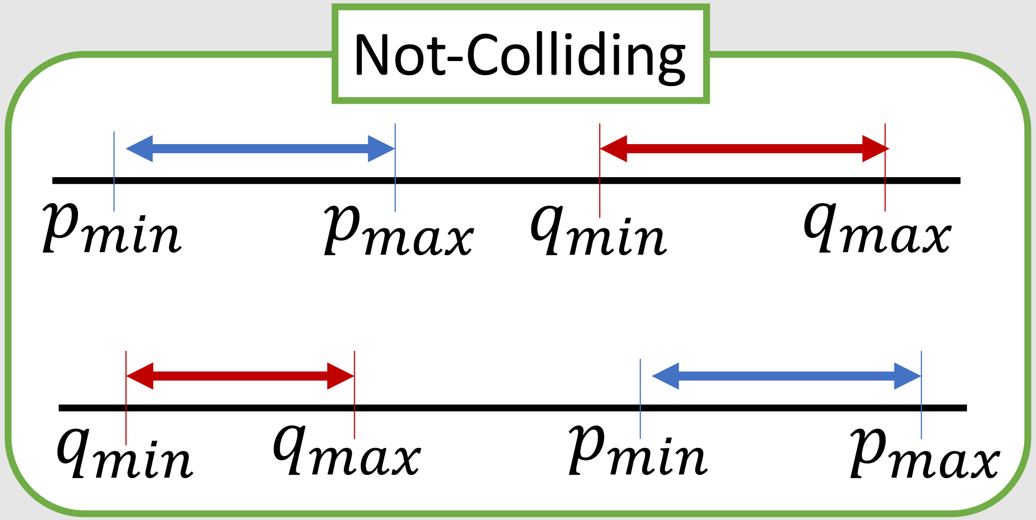
OOBB
Object-Oriented Bounding Box

k-DOP
discrete orientation polytope

1D Collision Detection

- What is the condition two line segments intersect?





$(p_{max} > q_{min})$ and $(q_{max} > p_{min})$

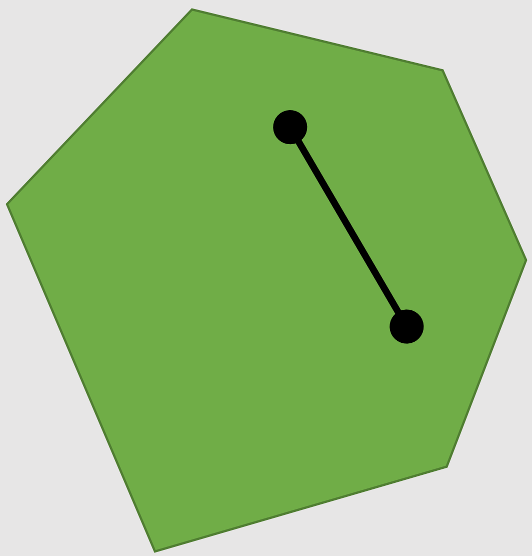
$(p_{max} < q_{min})$ or $(q_{max} < p_{min})$

Logical inverse

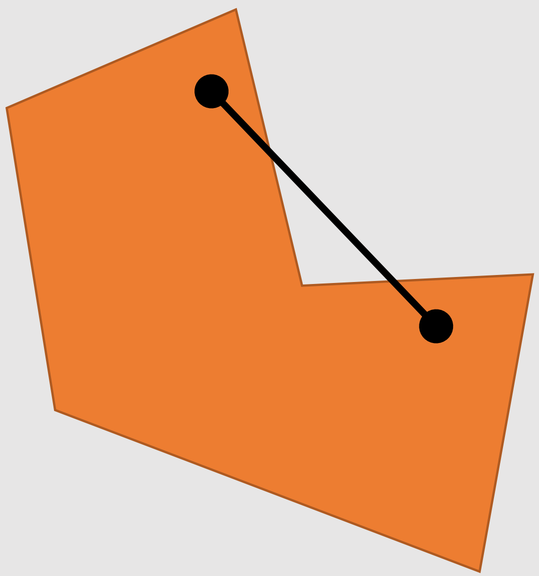
What is "Convex" Shape

- Interpolation of two points is always included

Convex

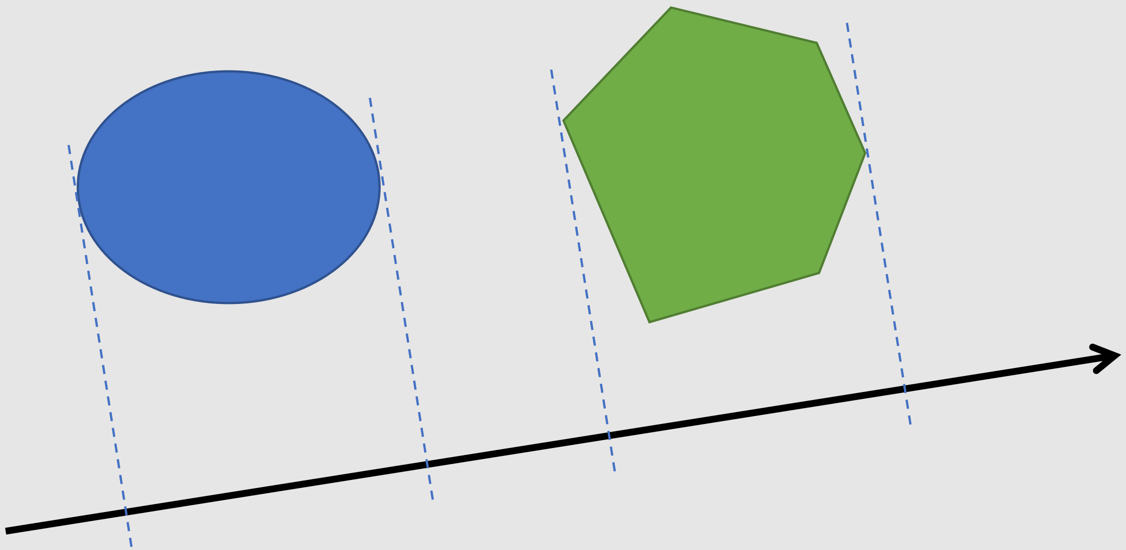


Non-Convex



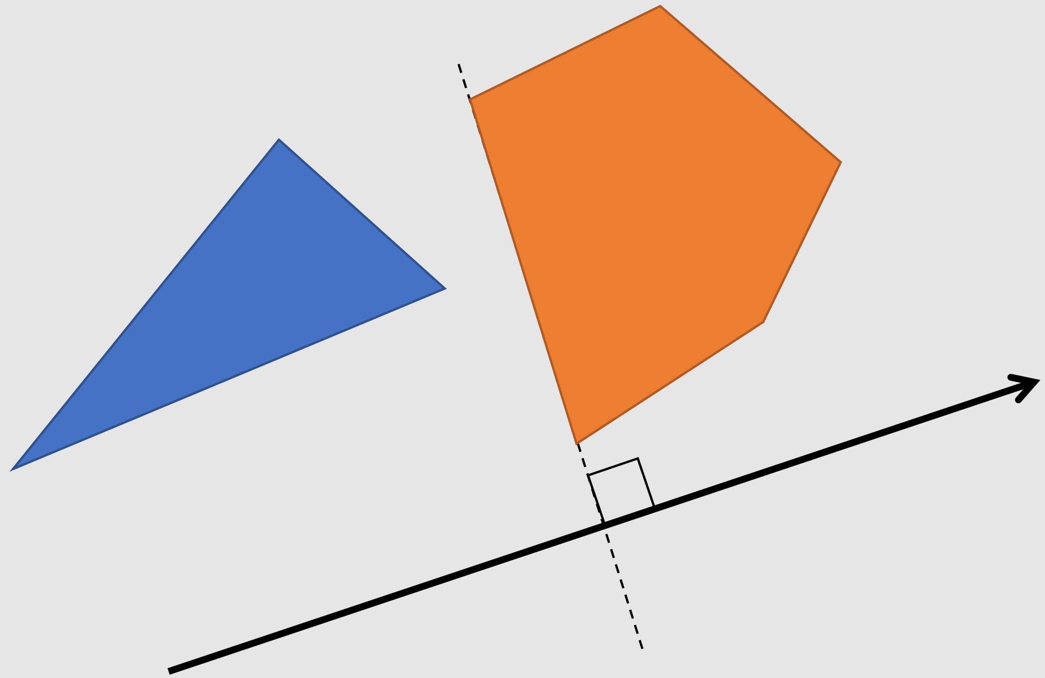
Separation Axis Theorem (SAT)

- If two convex shapes do not collide, there exists an axis where their projections will not overlap



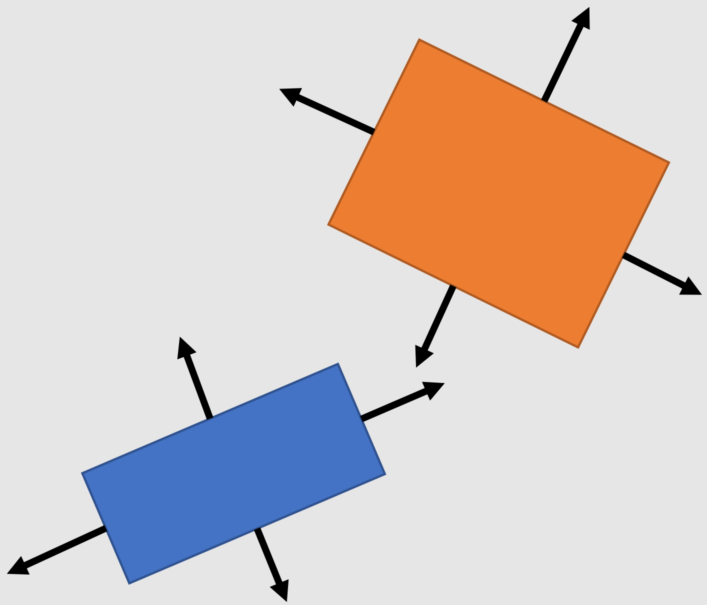
Separation Axis Theorem for 2D Polygons

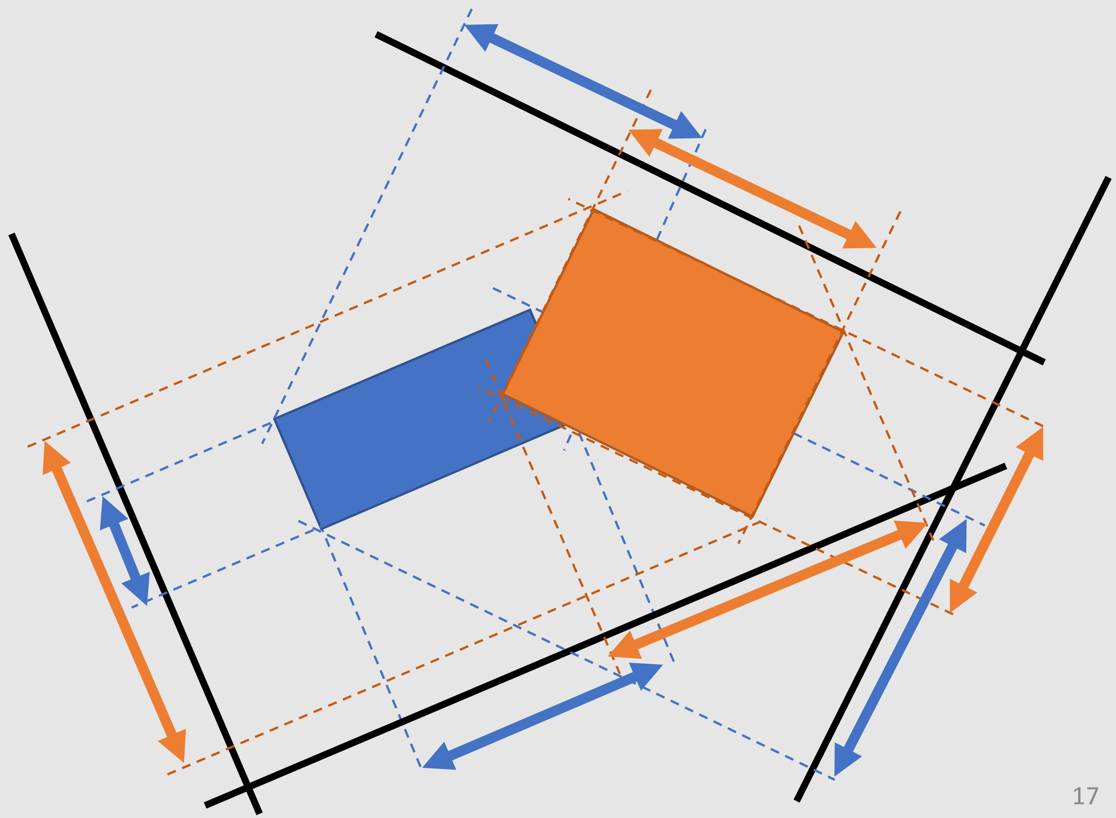
- One of the edges will be perpendicular to the separation axis



Collision Detection for 2D Polygons

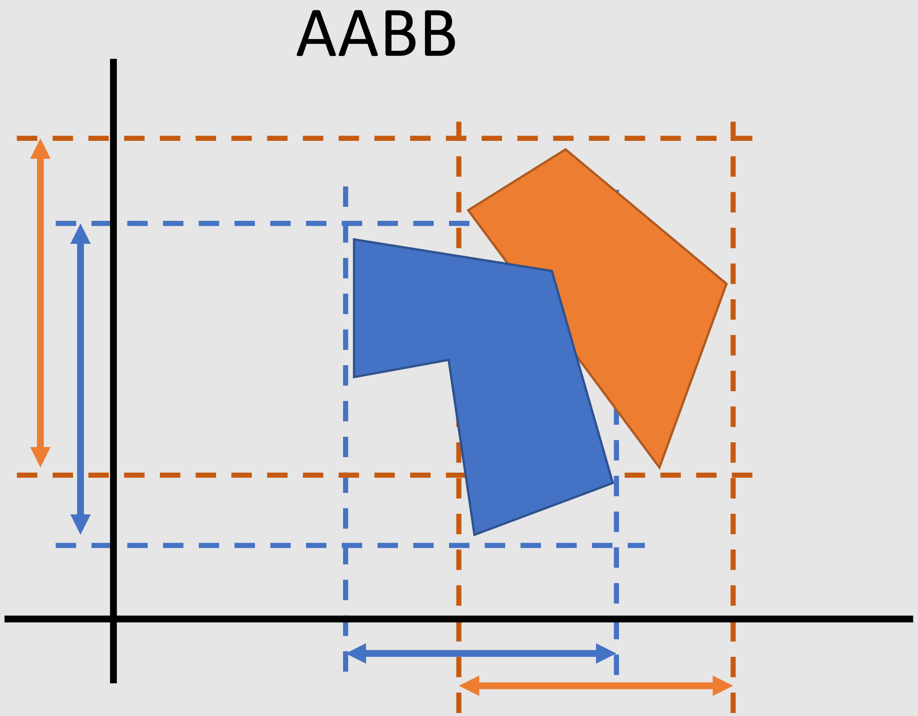
- Check all the axes perpendicular to polygon's edges

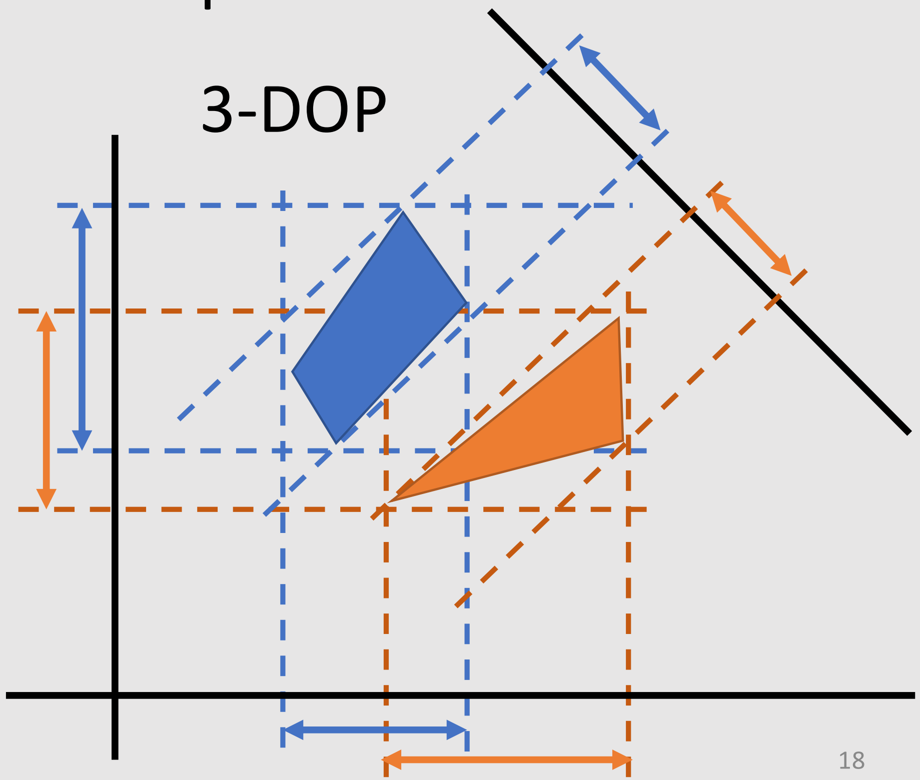




Collision of AABB and k-DOP

- Project the Bounding Volume (BV) on axes
- Two BVs collide if all projections overlap





Data Structure of AABB & k-DOP

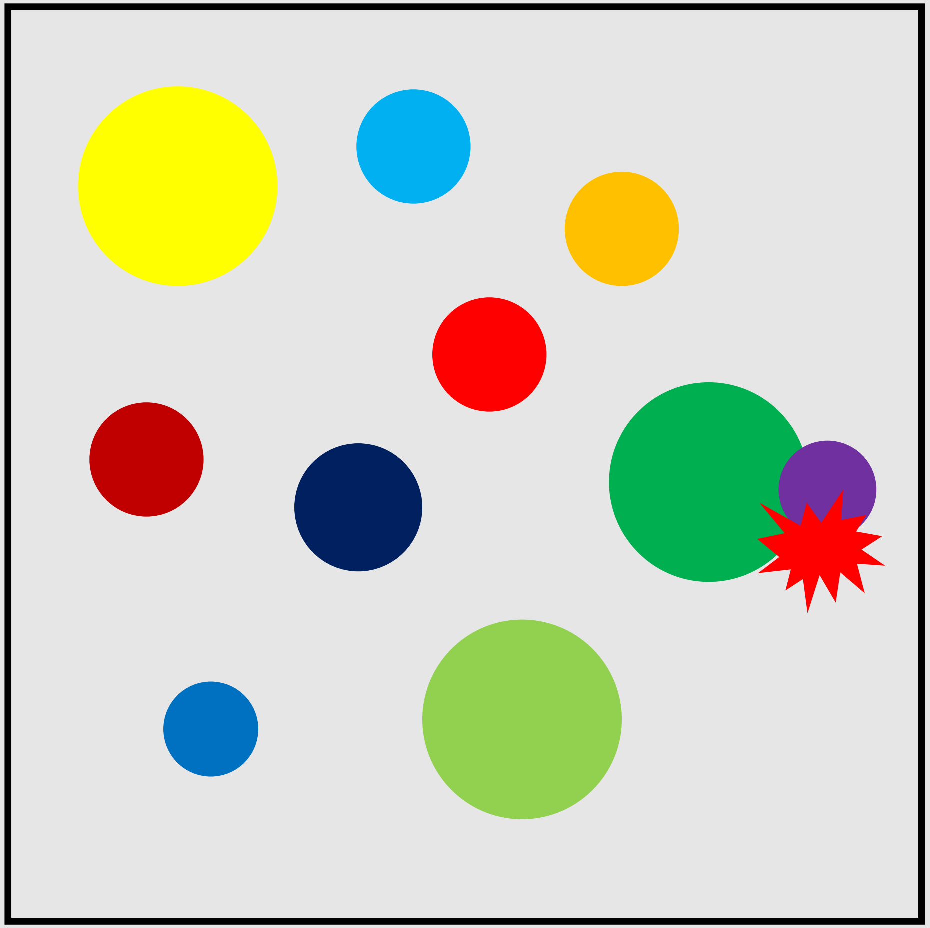
- Minimum and maximum along the axis

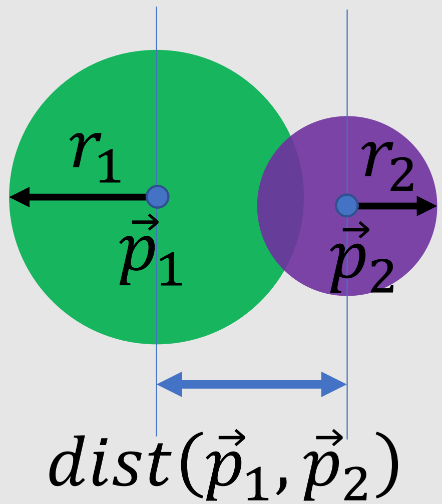
```
template <int naxis>
class CKdops
{
public:
  double minmax[naxis][2];
};

constexpr double axes[3][2] = {
  {0,1},
  {1,0},
  {1,1} };
std::vector< CKdops<3> > aKdops;
```

Non-type template parameter
(compile time argument)

Broad-phase Collision Detection

How We can Find Collisions of Circles?





$dist(\vec{p}_1, \vec{p}_2) \leq r_1 + r_2 \Rightarrow$Collision

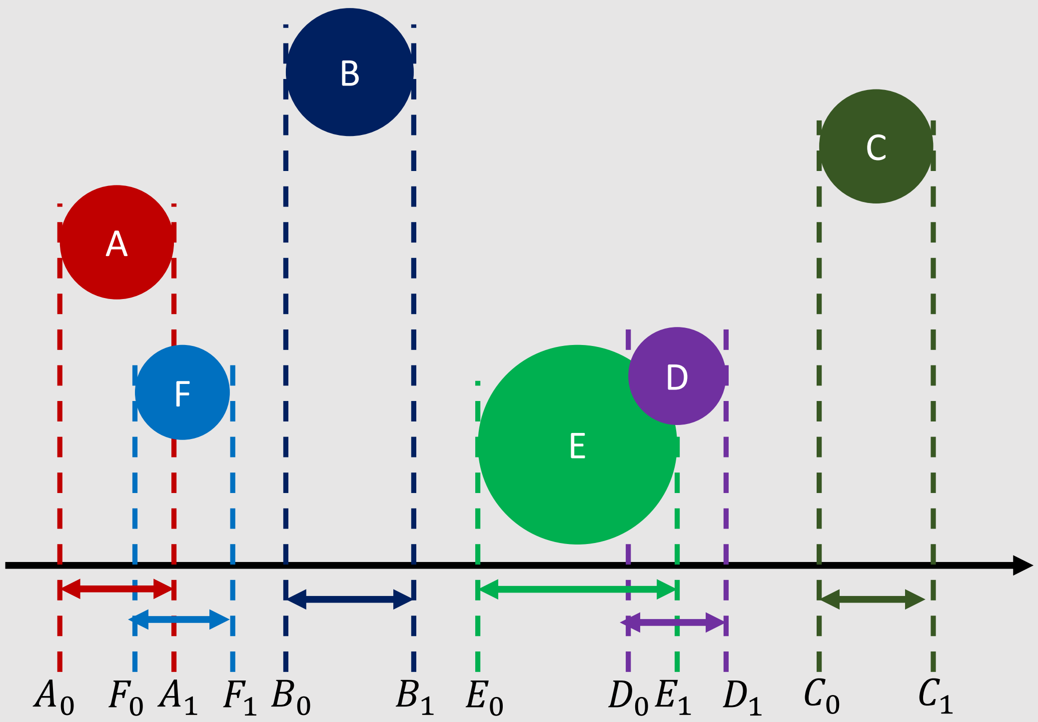
Approaches

- ~~Brute force approach~~
- Sweep & Prune method
- Spatial Hashing (e.g., Regular grid)
- Spatial Partitioning (e.g., KD-tree)
- Bounding Volume Hierarchy (BVH)

We four are awesome!

Sweep & Prune (Sort & Sweep) Method

- Simple but effective culling method



$\{A_0, A_1, B_0, B_1, C_0, C_1, D_0, D_1, E_0, E_1, F_0, F_1\}$

sort

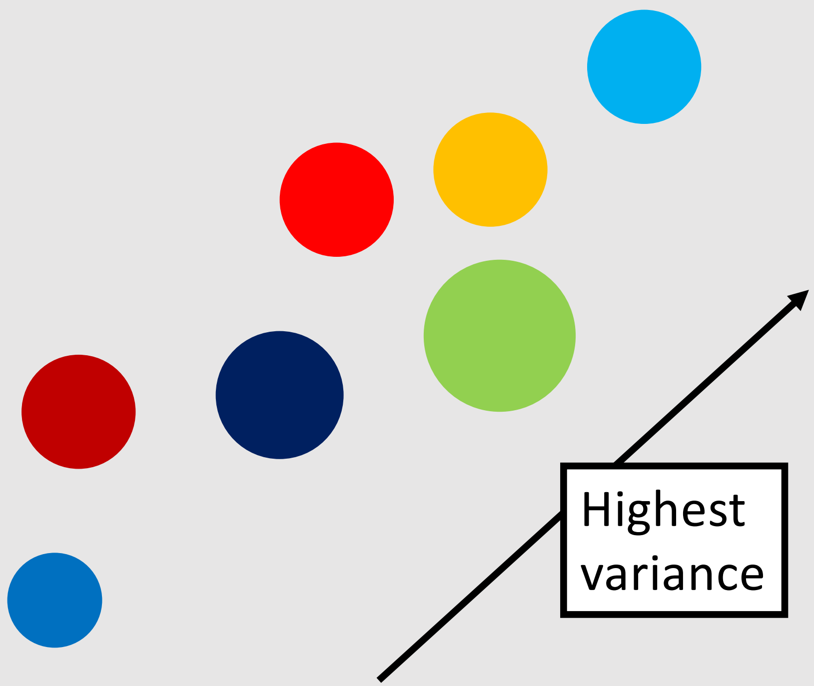
$\{A_0, F_0, A_1, F_1, B_0, F_1, E_0, D_0, E_1, D_1, C_0, C_1\}$

X_0: put X in the stack

X_1: remove X in the stack

How to Choose Sweeping Axis ?

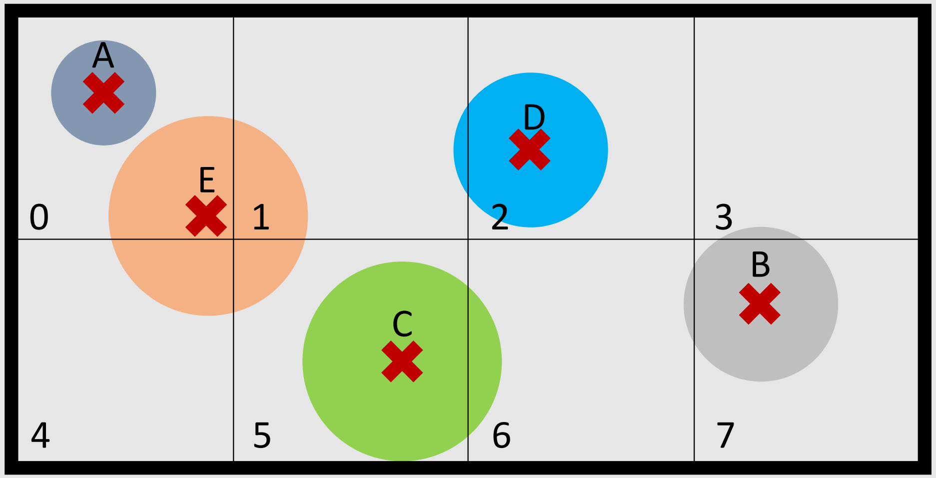
- kDOPs -> Sweep in the kDOPs' axis
- Sphere, AABB, OOBB -> XYZ-axis or PCA



Spatial Hashing using Regular Grid

- Putting circles in a grid based on circles' center positions
- Grid length is maximum diameter of the circle

  ➡ Look only 1-ring neighborhood



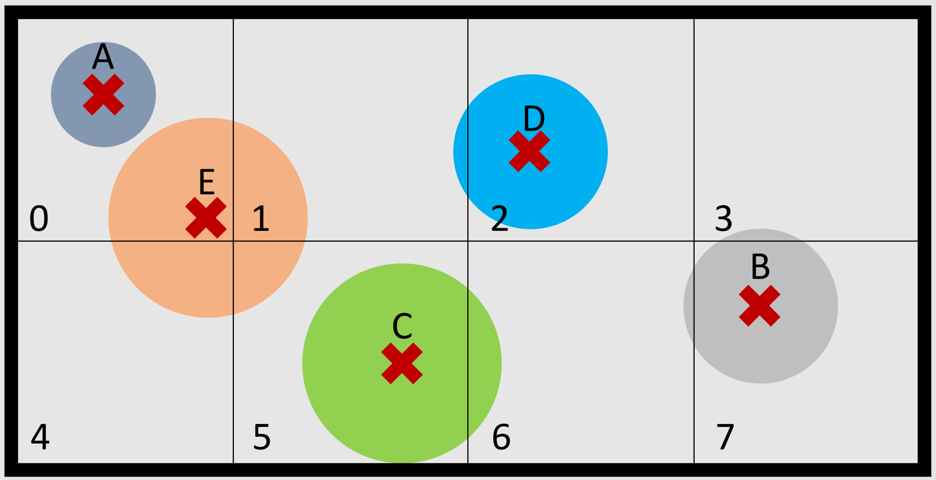
Possible collisions:

{A,E}, {E,C}, {C,D}, {D,B}

No need to check for {E,D},{C,B}…etc

Spatial Hashing using Regular Grid

- Creating look-up table from grid index to circle index



circle index	A	B	C	D	E
grid index	0	7	5	2	0

sort by the grid index

A=

circle index	A	E	D	C	B	
grid index	0	0	2	5	7	

B=

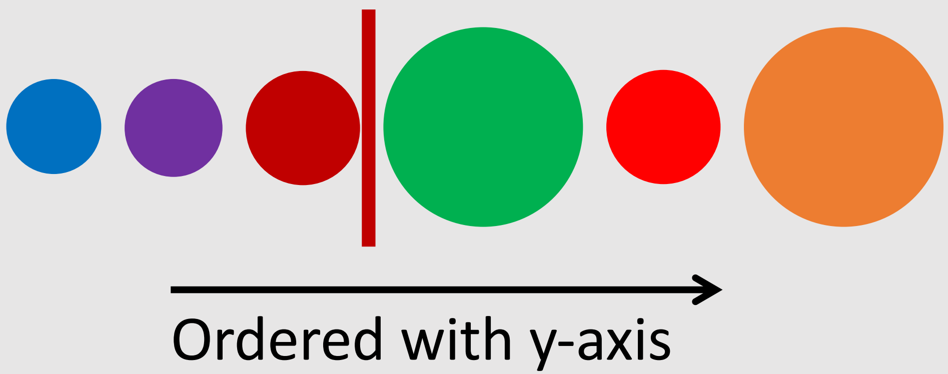
index of A	0	2	2	3	3	3	4	4	5

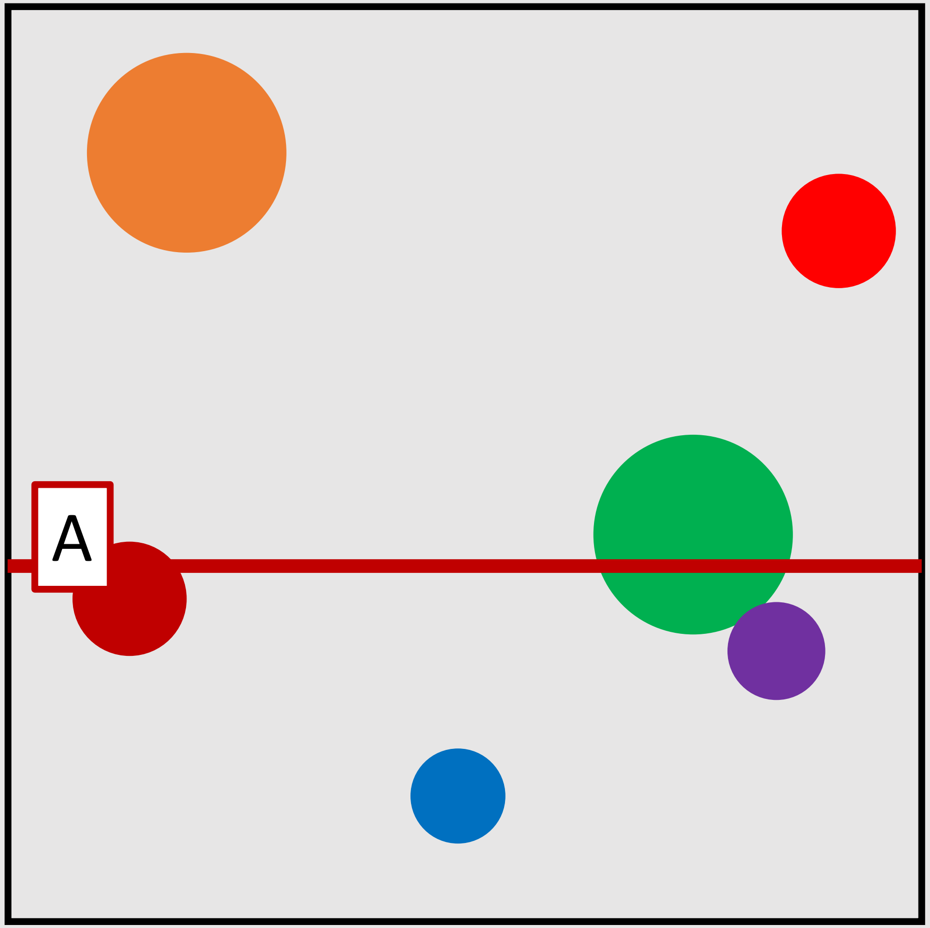
jagged array

```
B[igrid] <= j < B[igrid+1]
icircle=A[j]
```

Space Partitioning with K-D Tree

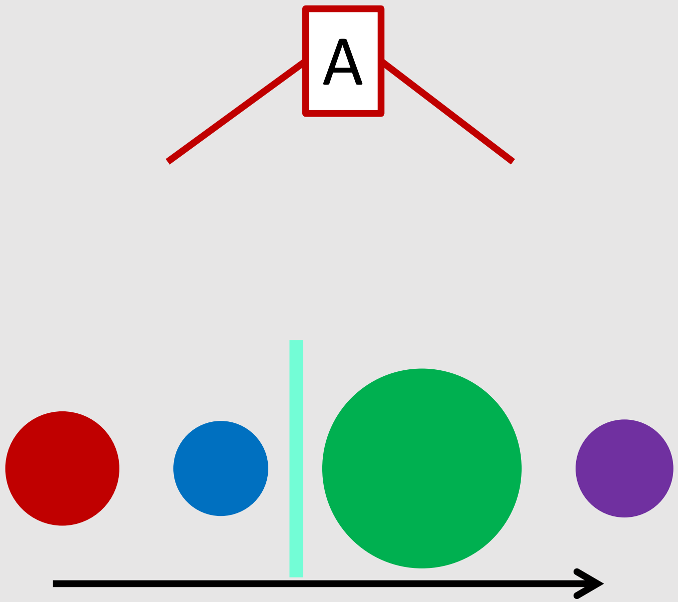
1. Select axis (e.g., y-axis)
2. Split the space along median



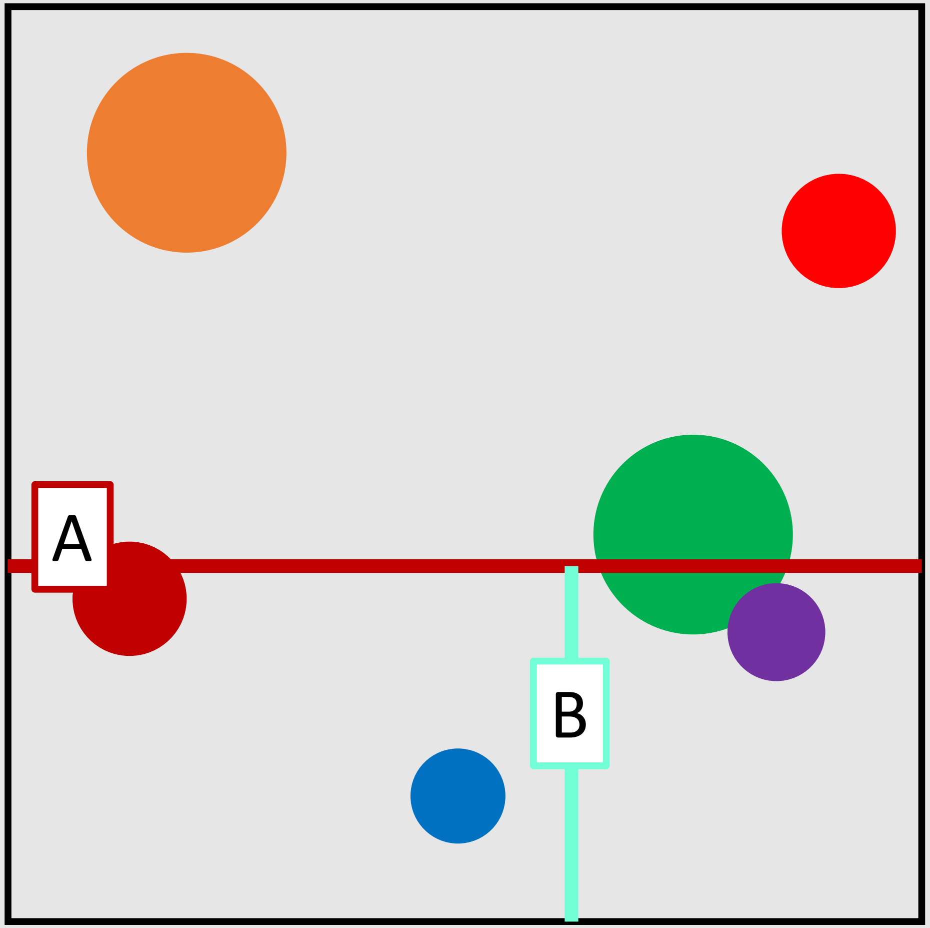


Space Partitioning with K-D Tree

1. Select axis (e.g., y-axis)
2. Split the space along median
3. Repeat along other axis (e.g., x-axis)

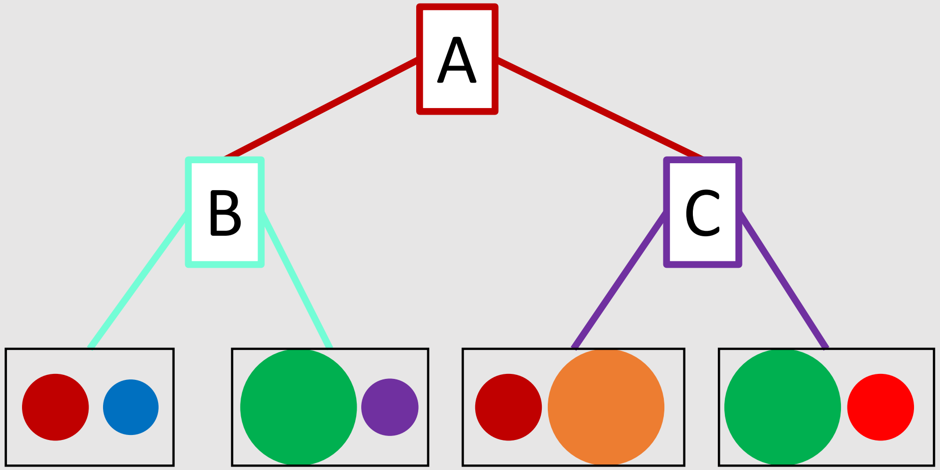


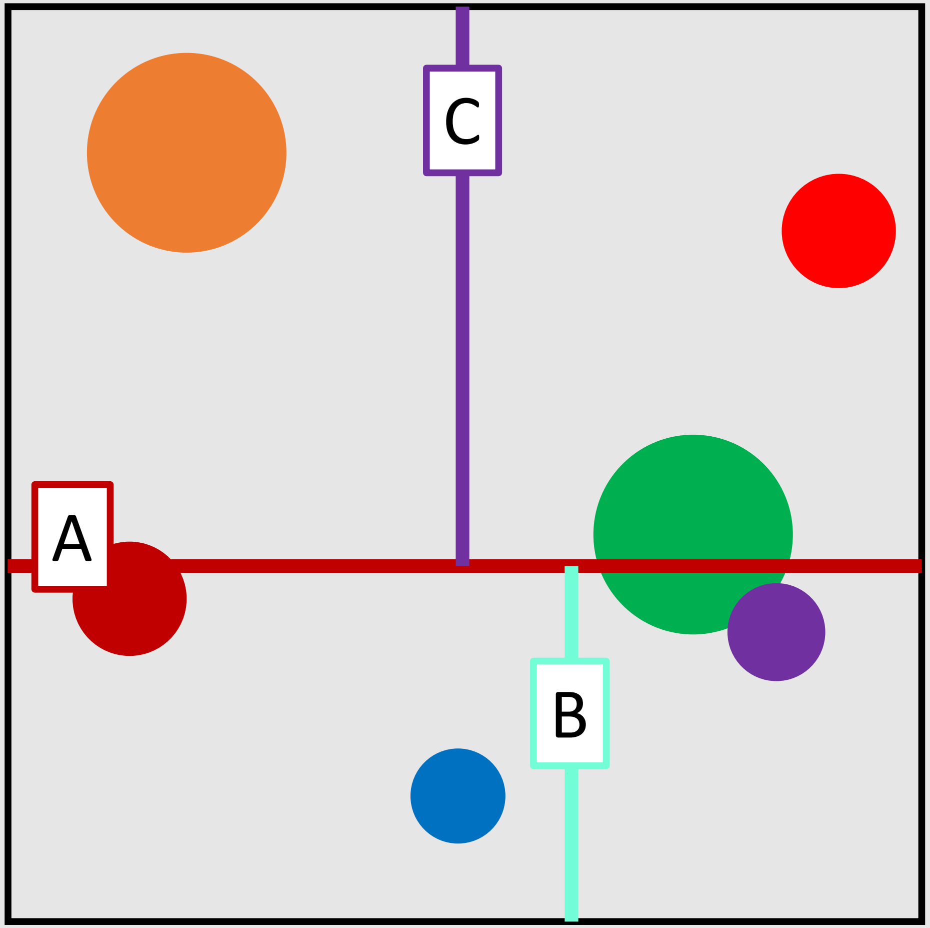
Ordered with x-axis



Space Partitioning with K-D Tree

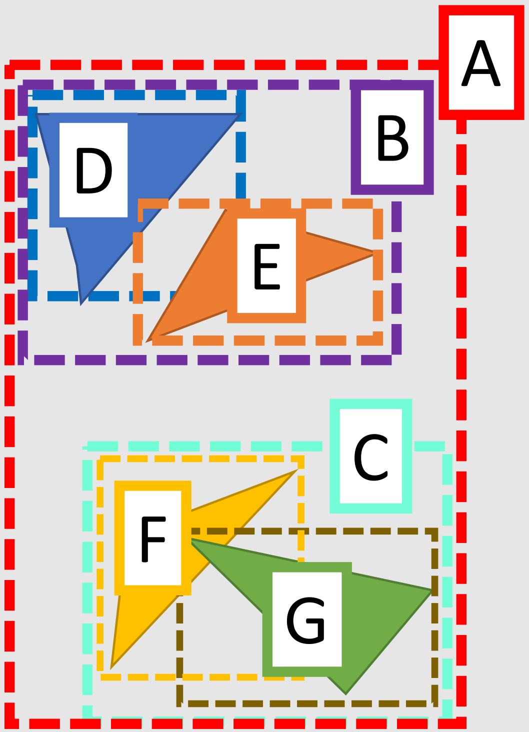
1. Select axis (e.g., y-axis)
2. Split the space along median
3. Repeat along other axis (e.g., x-axis)

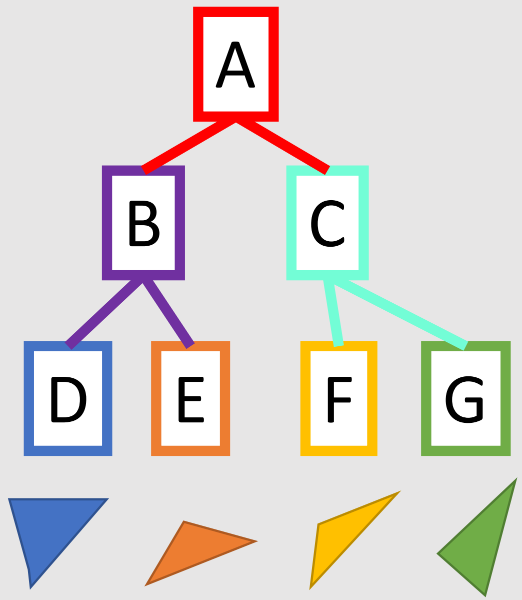




Bounding Volume Hierarchy (BVH)

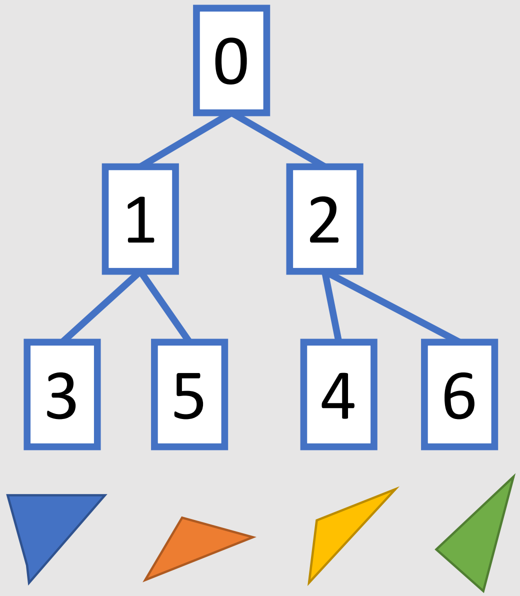
- Near triangles are in the same branch
- Each node has a BV that includes two child BVs





Example of BVH Data Structure in C++

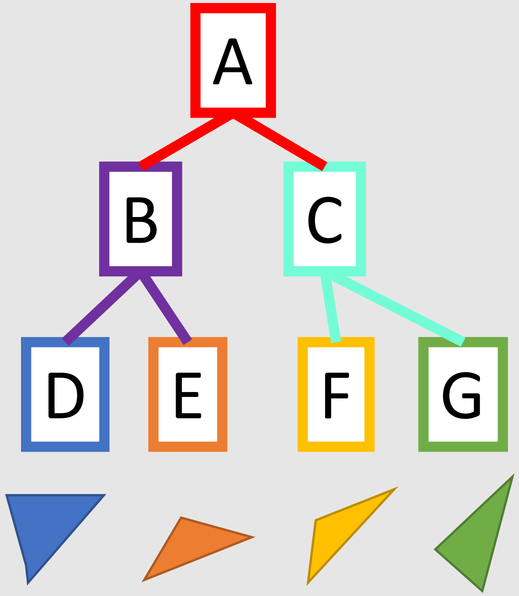
index	0	1	2	3	4	5	6
left-child index	1	3	4	tri index	tri index	tri index	tri index
Right-child index	2	5	6	-1	-1	-1	-1
BV data	…	…	…	…	…	…	…




```
template <class T>
class CNodeBVH {
        unsigned int ichild_left;
        unsigned int ichild_right;
        T BV;
};

std::vector<CNodeBVH<CAABB>> aNodeBVH;
```

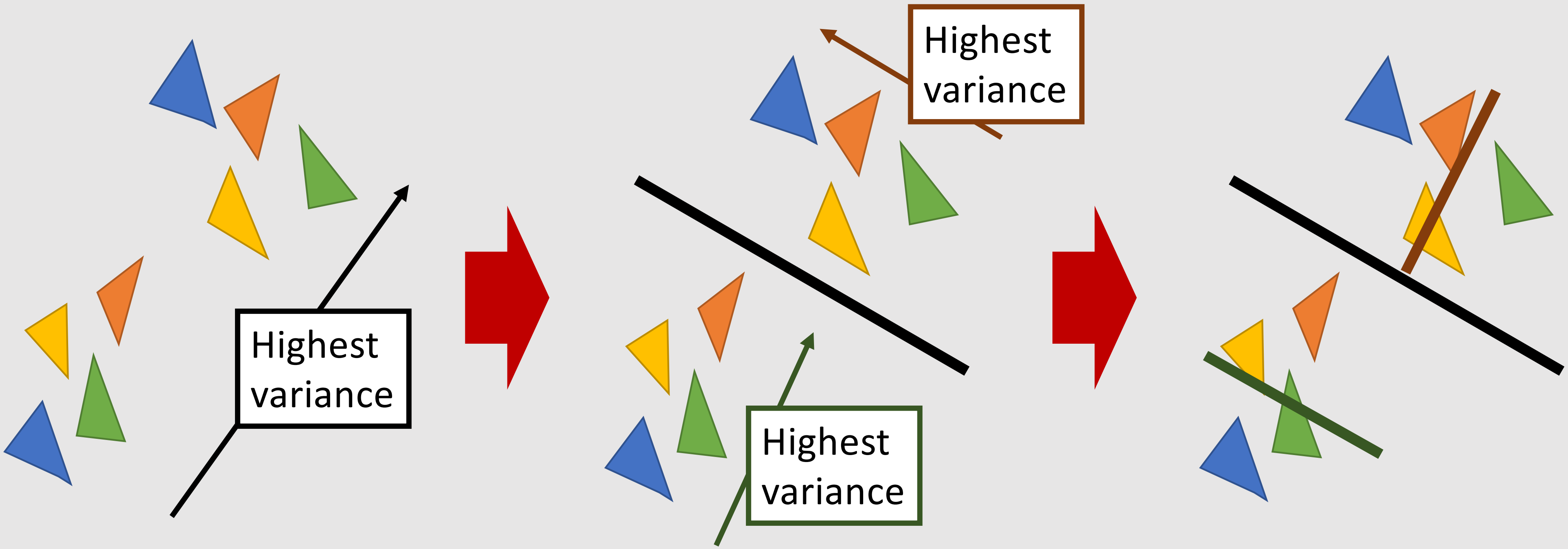
Evaluation of BVH using Recursion

- Ask question to the one node -> that node asks the same question to two child nodes



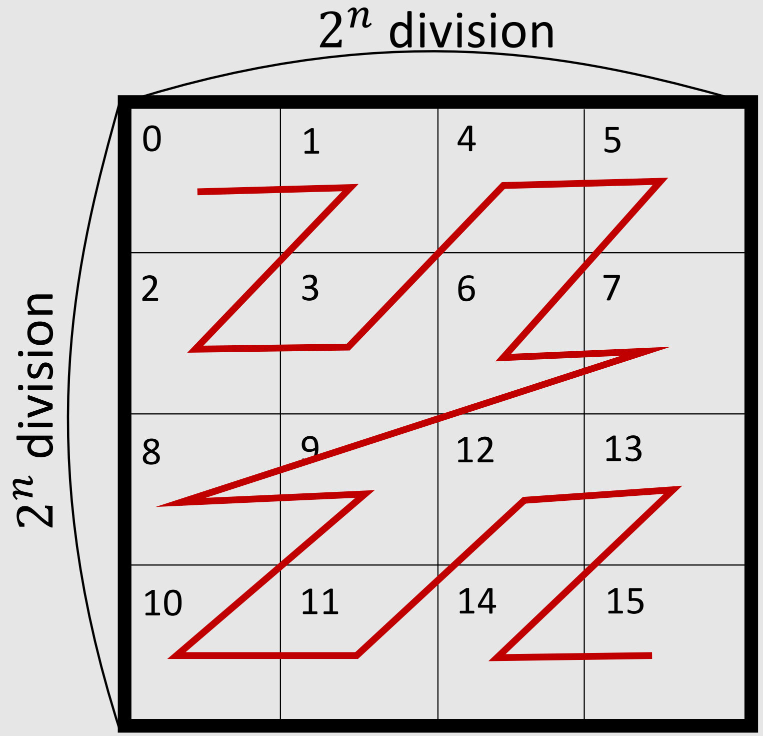
Top-down Approach to Build BVH

- Use PCA for separating triangles into two groups



Linear BVH: Fully Parallel Construction

- Construct BVH based on Morton code (i.e., Z-order curve)
- Two cells with close Morton codes tends to be near

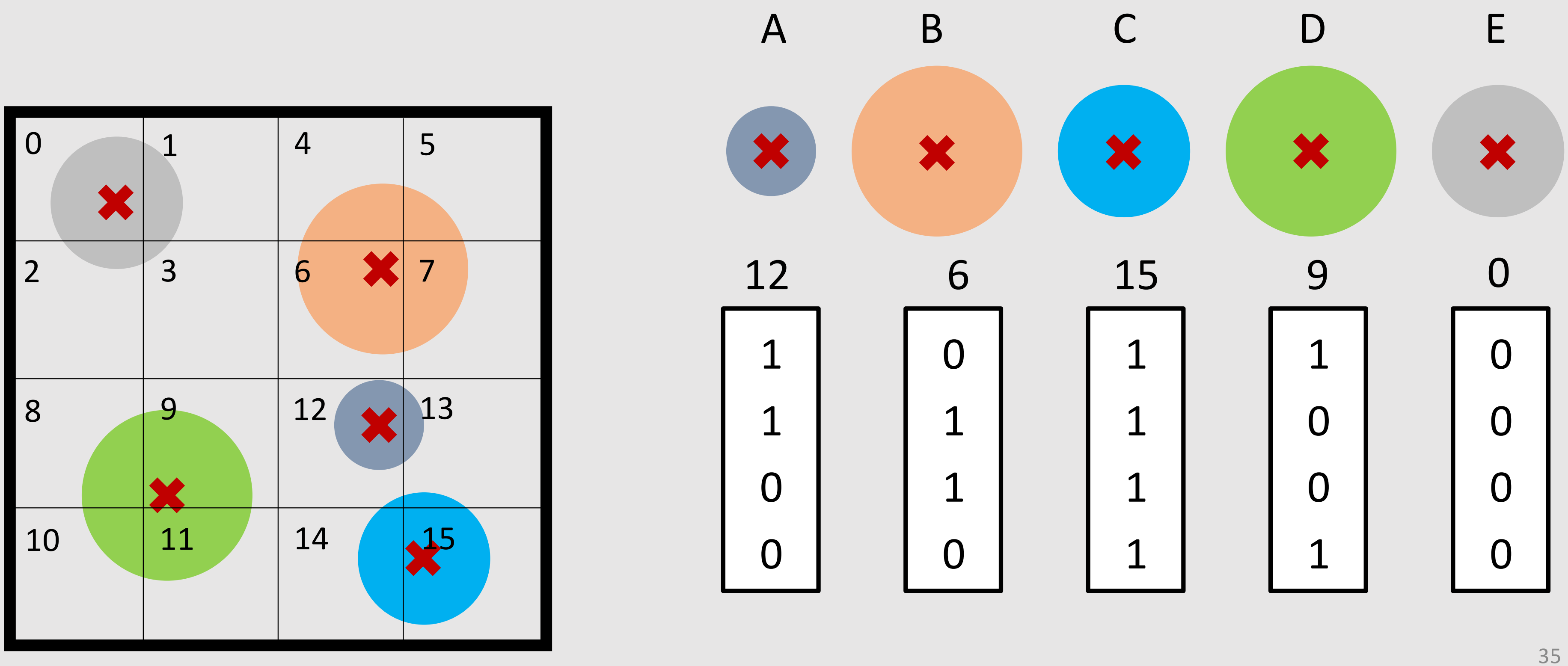


2D square domain with 2^n edge division

- ➡ 2^{2n} number of cells
- ➡ Cell index is size of $2n$ in binary

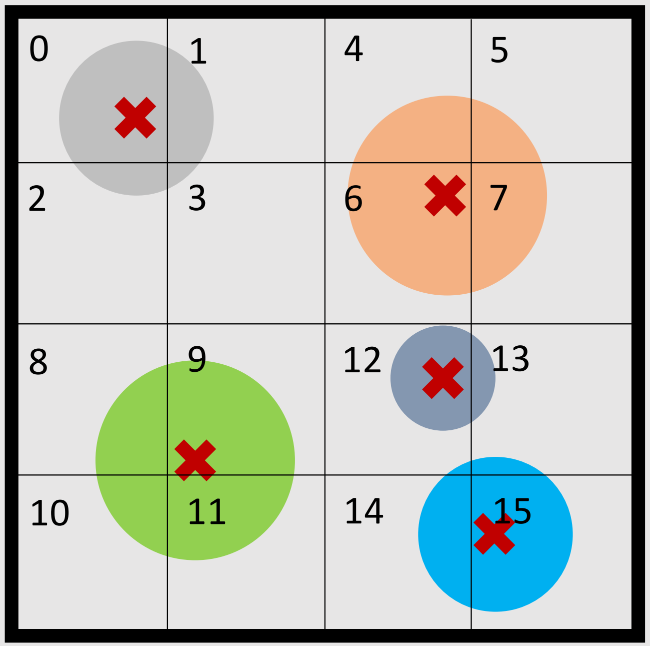
Linear BVH: Fully Parallel Construction

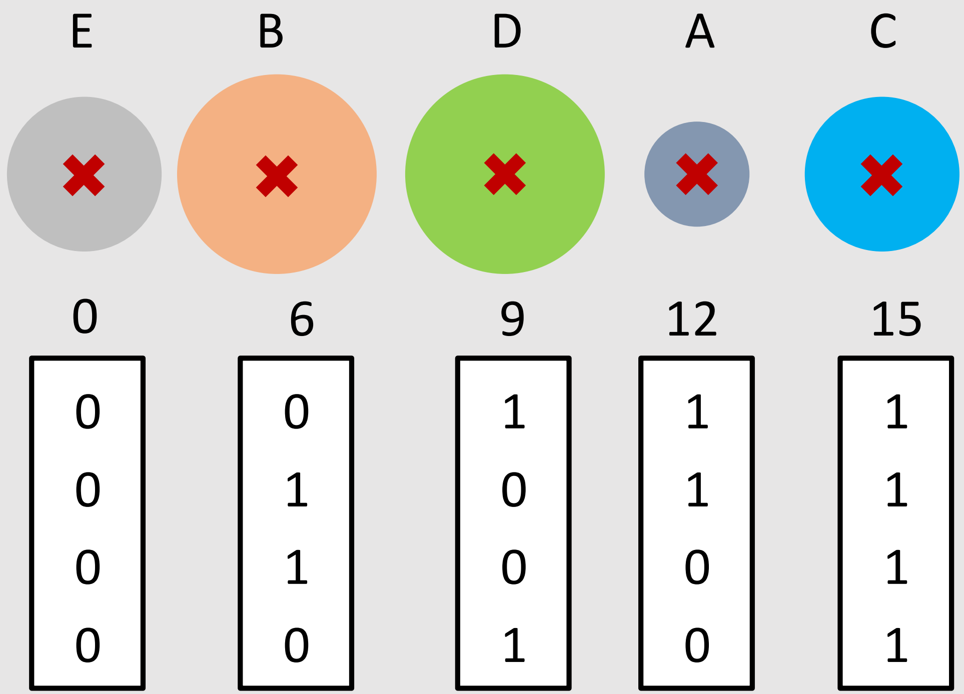
- Convert XYZ coordinate into 1D (linear) integer coordinate



Linear BVH: Fully Parallel Construction

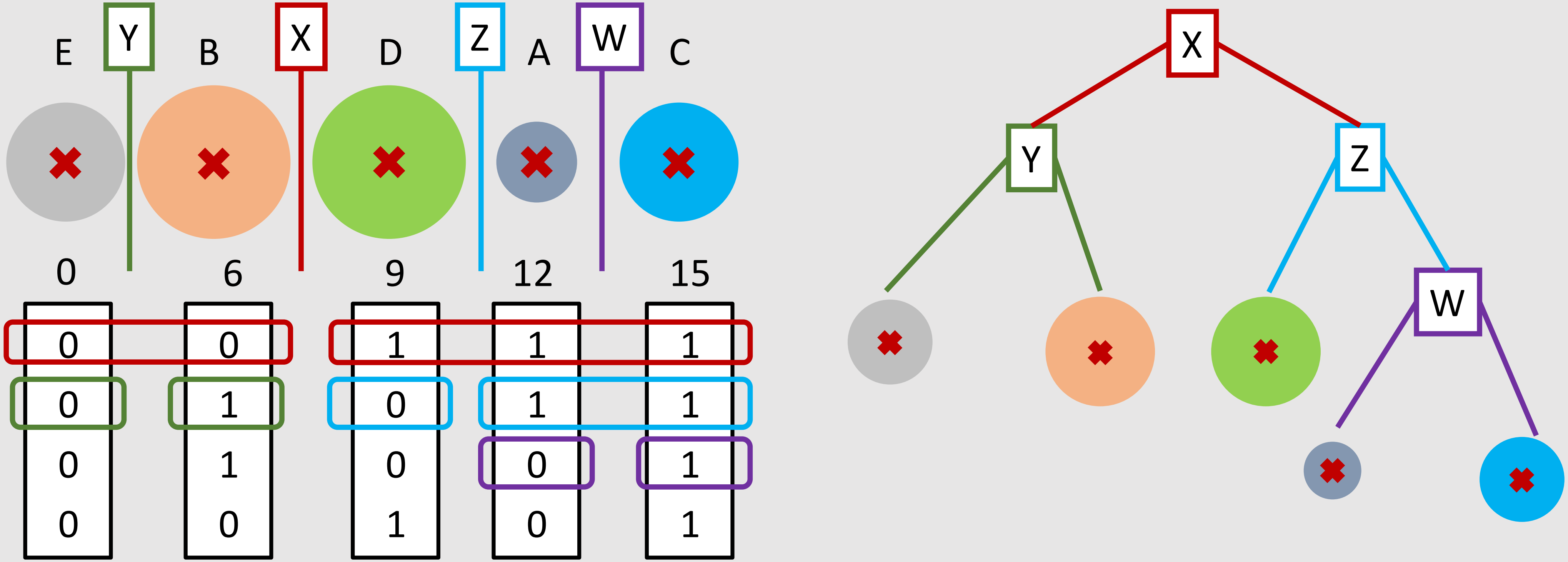
- Sort objects by their Morton codes





From Morton Code to BVH Tree

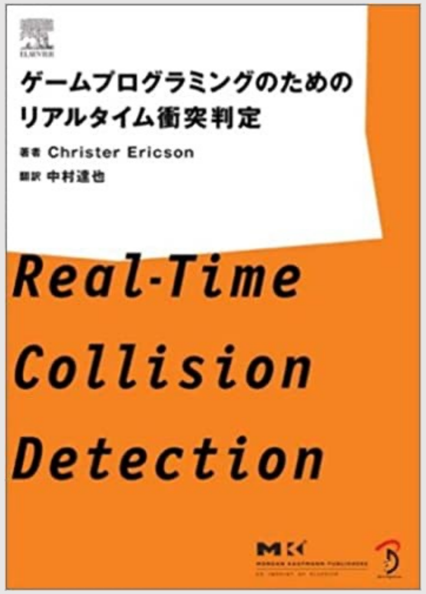
- Divide tree when digits of sorted Morton codes are different



Reference

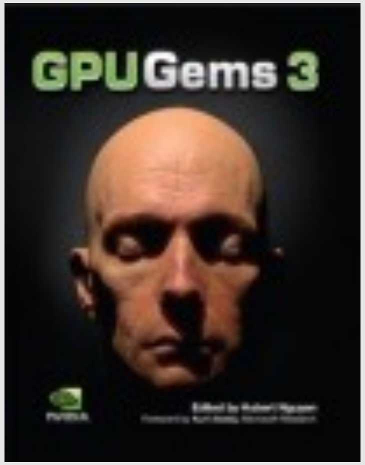
- “Real-Time Collision Detection” by Christer Ericson

Japanese translation available



Reference

- GPU Gems 3: Chapter 32. Broad-Phase Collision Detection with CUDA

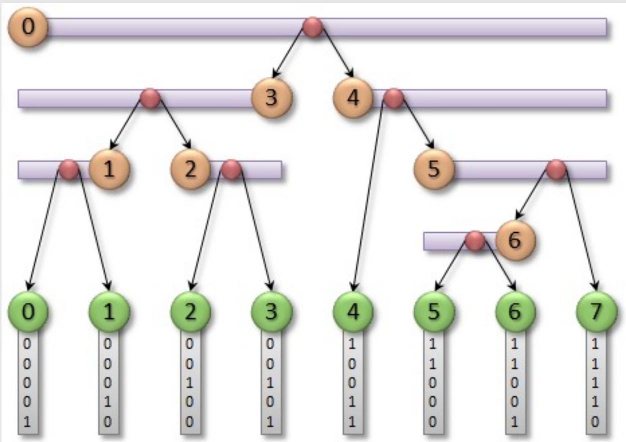


Available for free at: https://developer.nvidia.cn/gpugems/gpugems3/part-v-physics-simulation/chapter-32-broad-phase-collision-detection-cuda

Reference on Linear-BVH

- Thinking Parallel, Part III: Tree Construction on the GPU

by Tero Karras



https://developer.nvidia.com/blog/thinking-parallel-part-iii-tree-construction-gpu/