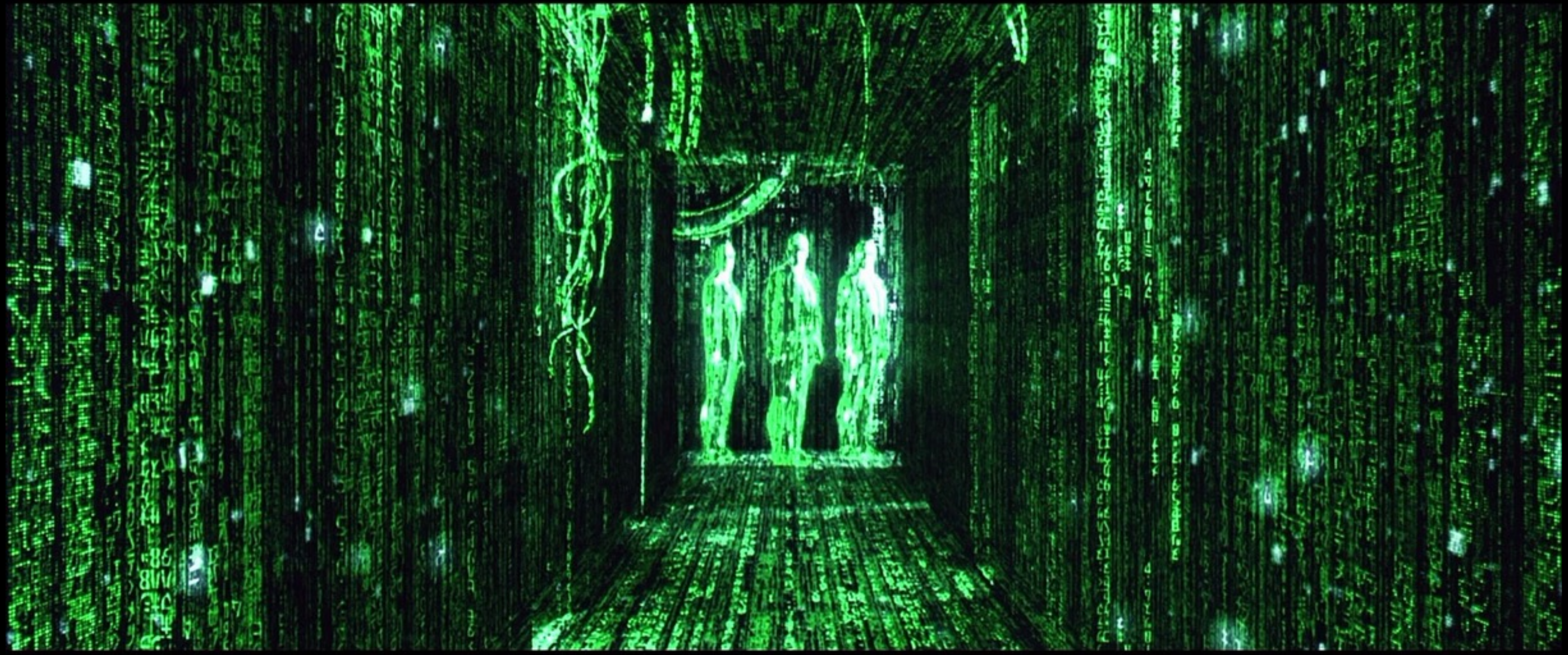
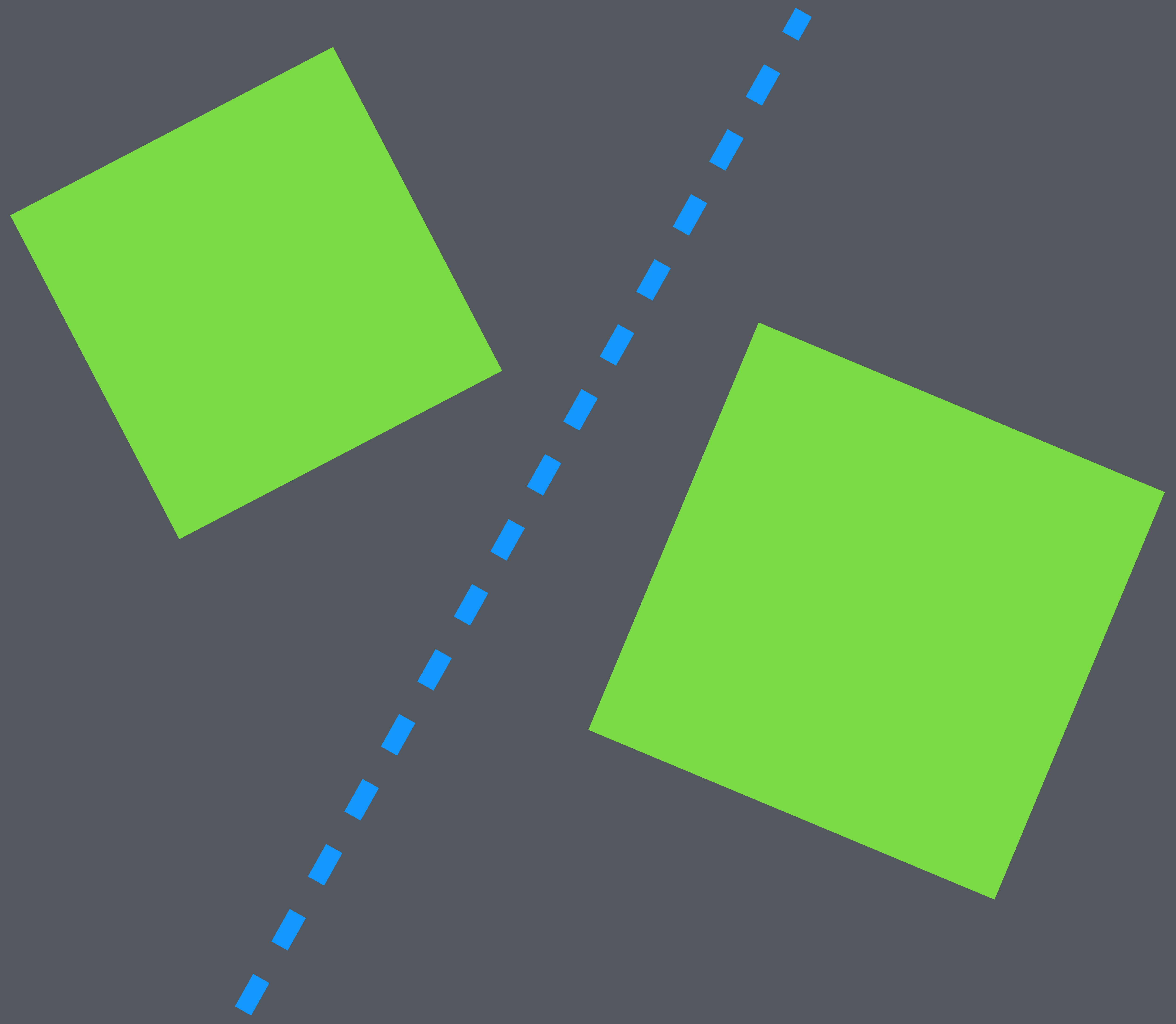
Matrix transformations.

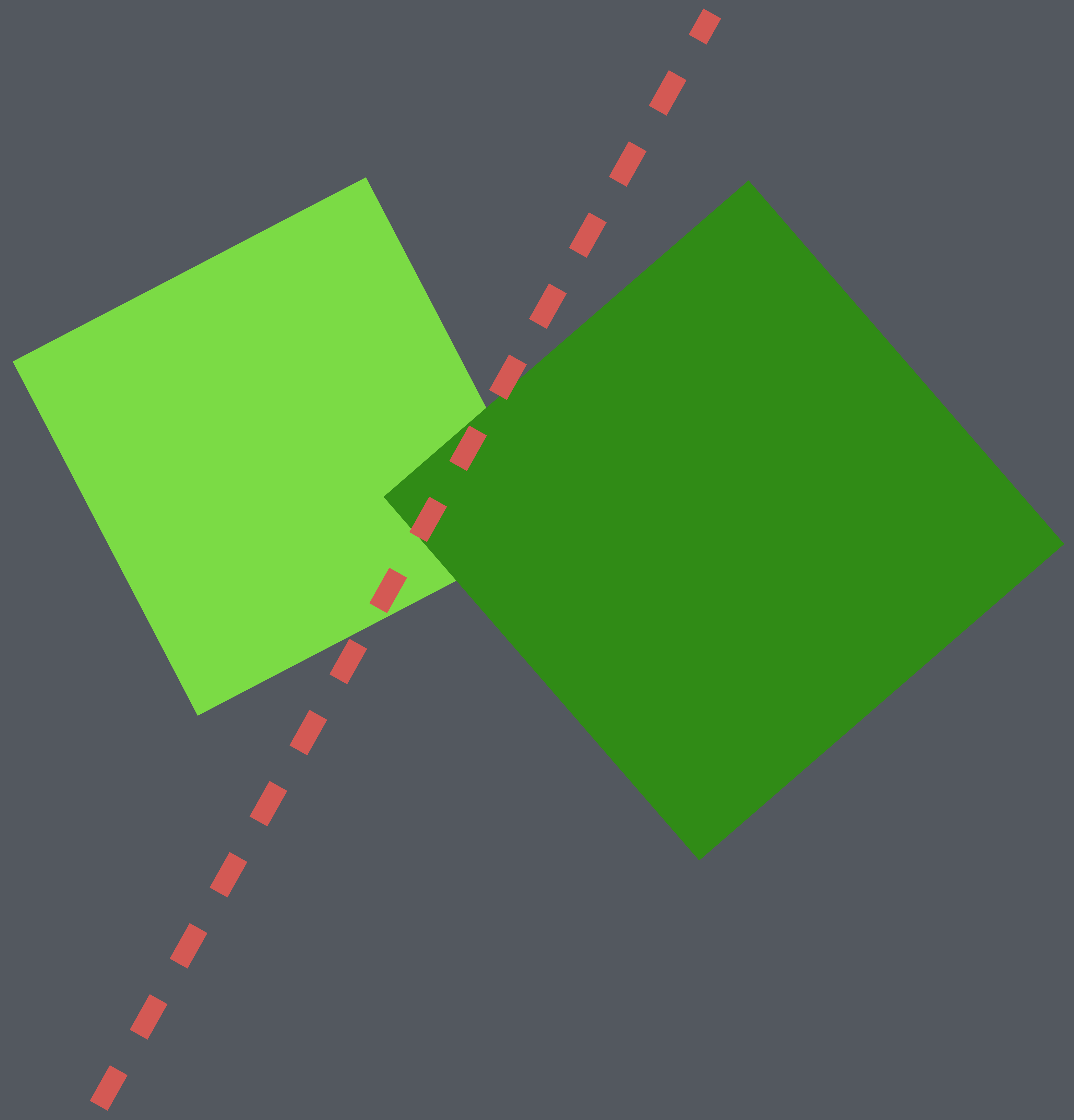
Part 3



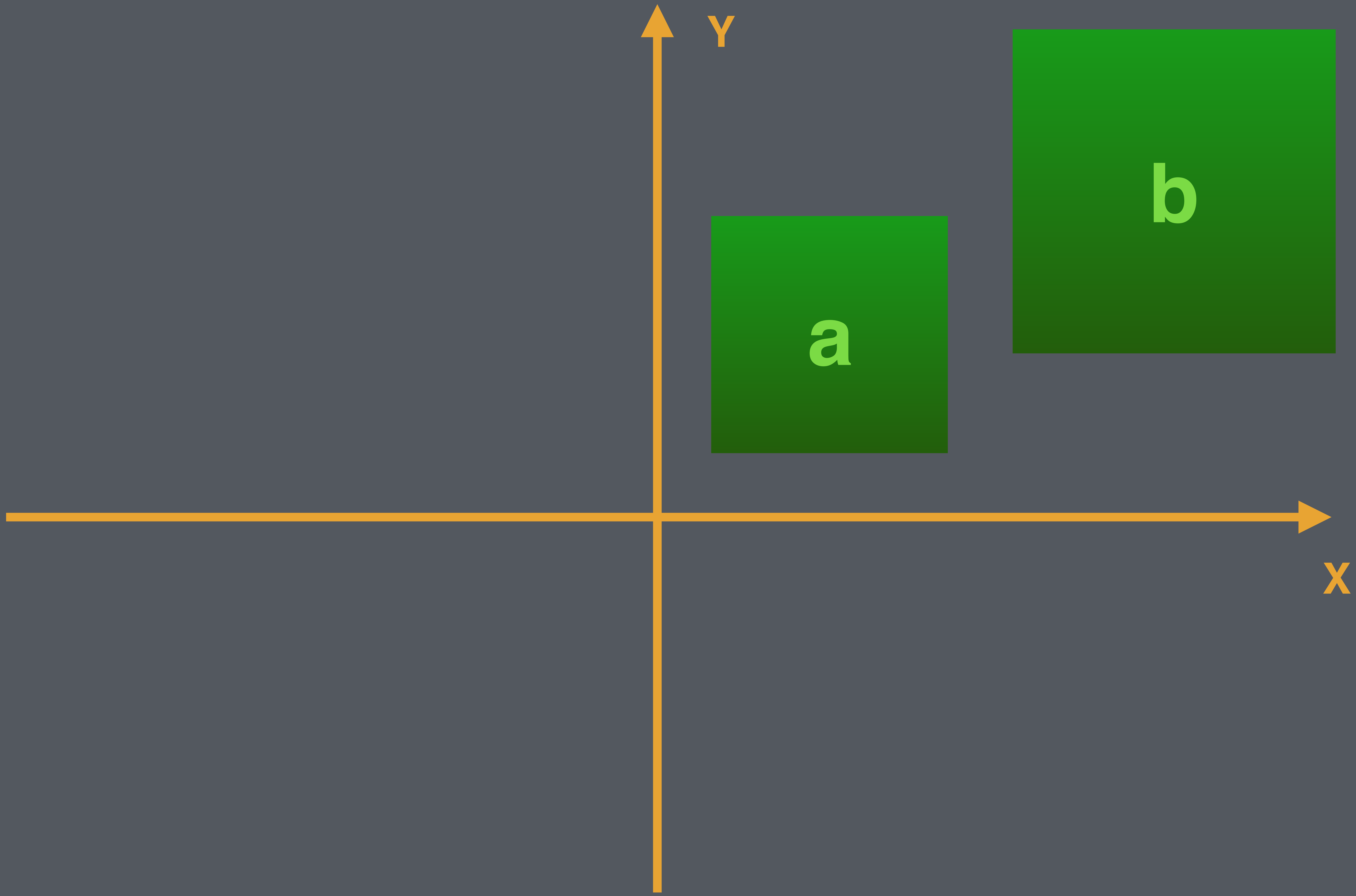
Complex collision

Separating axis theorem.

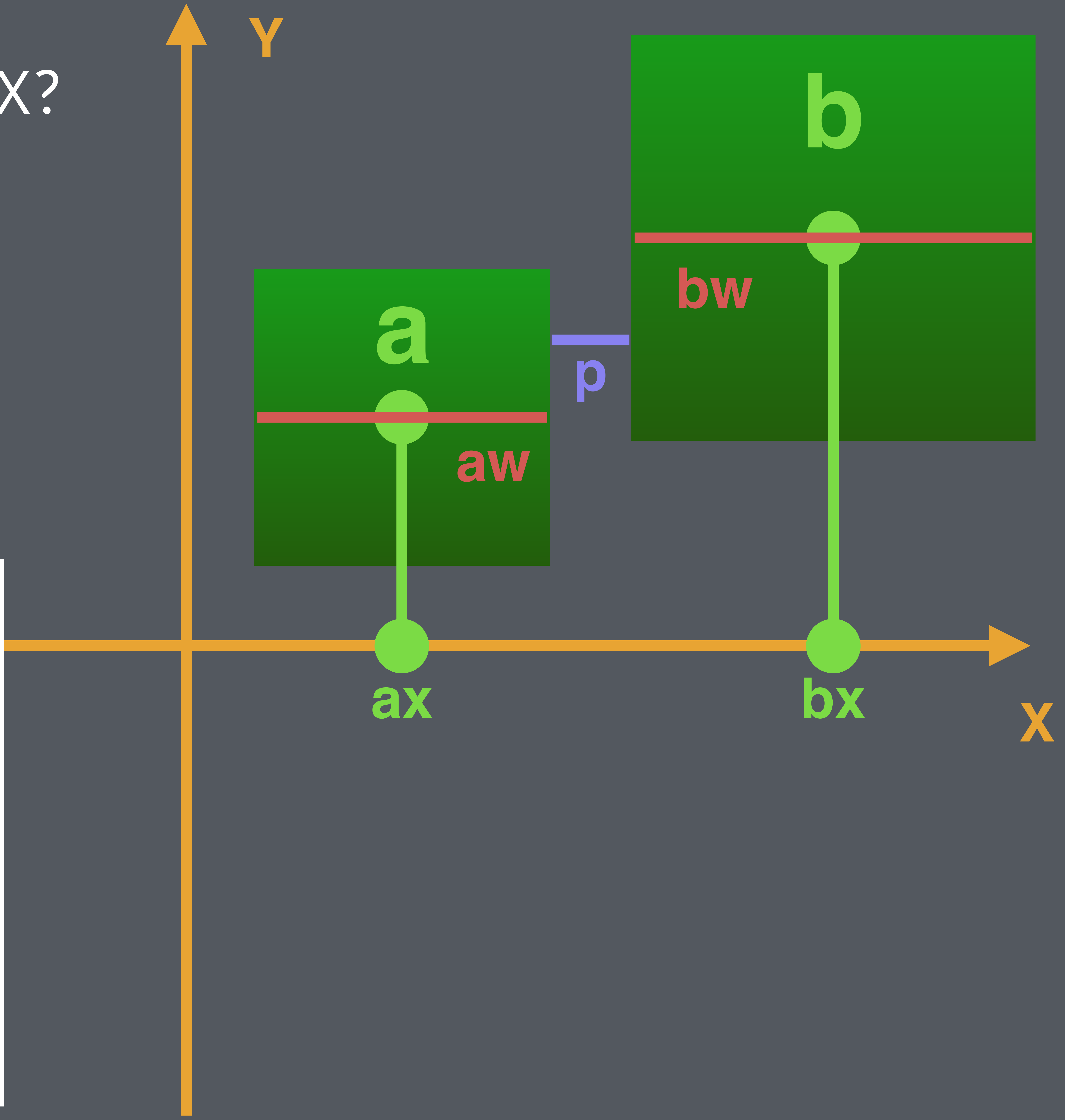




An **axis-aligned** example.

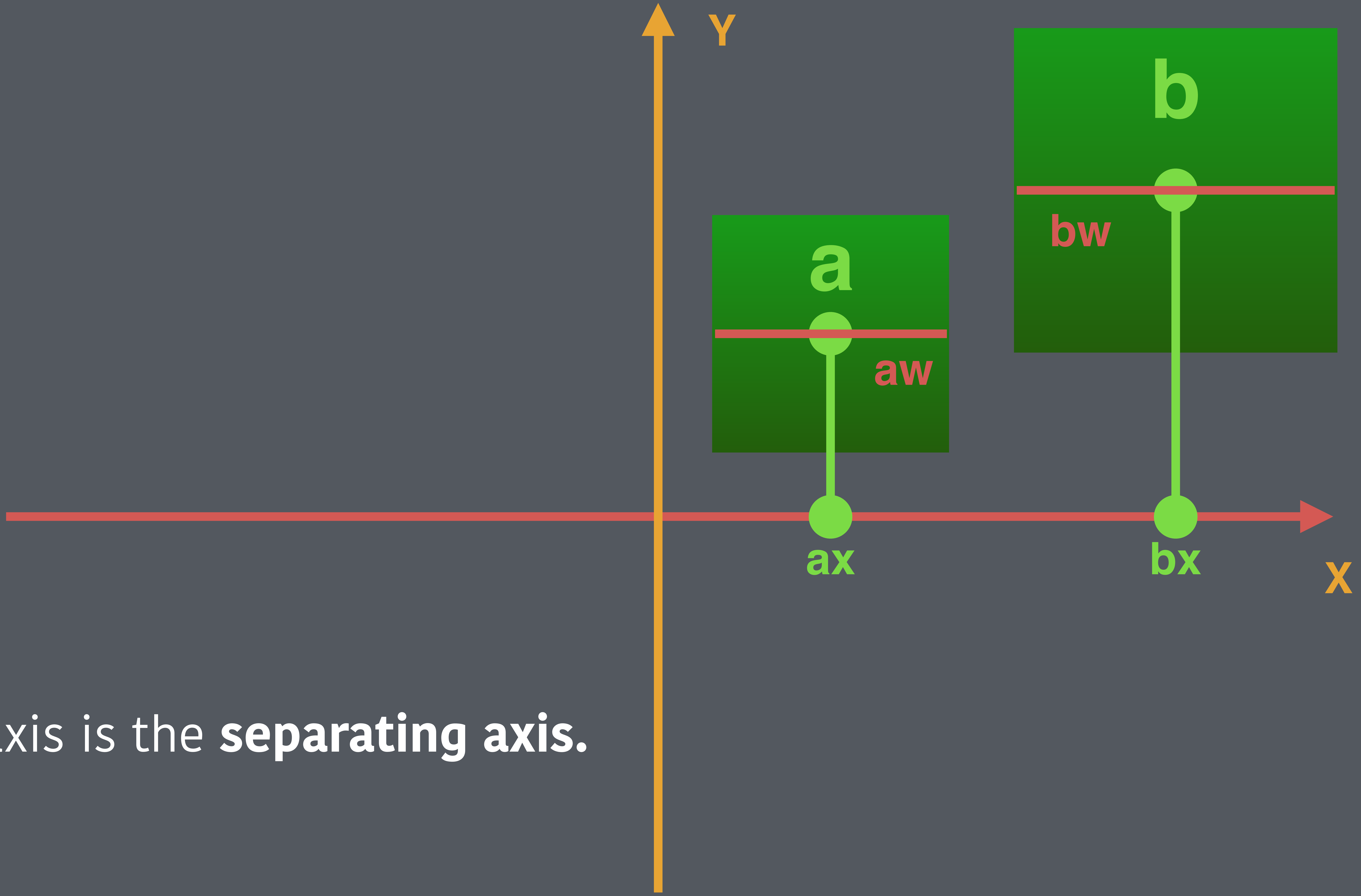

Y
b
a
X

How far away are they on X?



$$p = |x_1 - x_2| - \frac{w_1 + w_2}{2}$$

if p >= 0, we are not colliding!

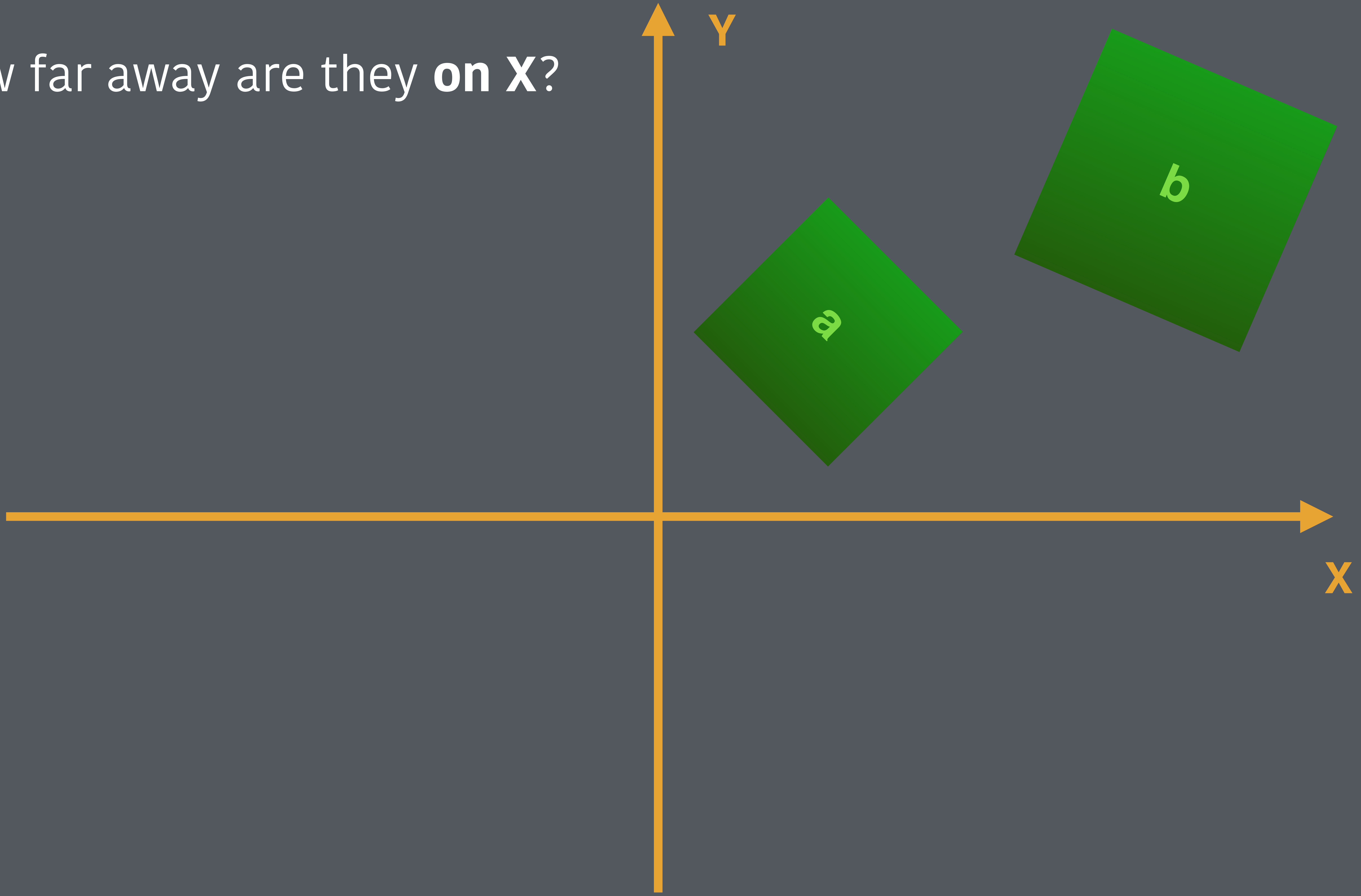


X axis is the **separating axis.**

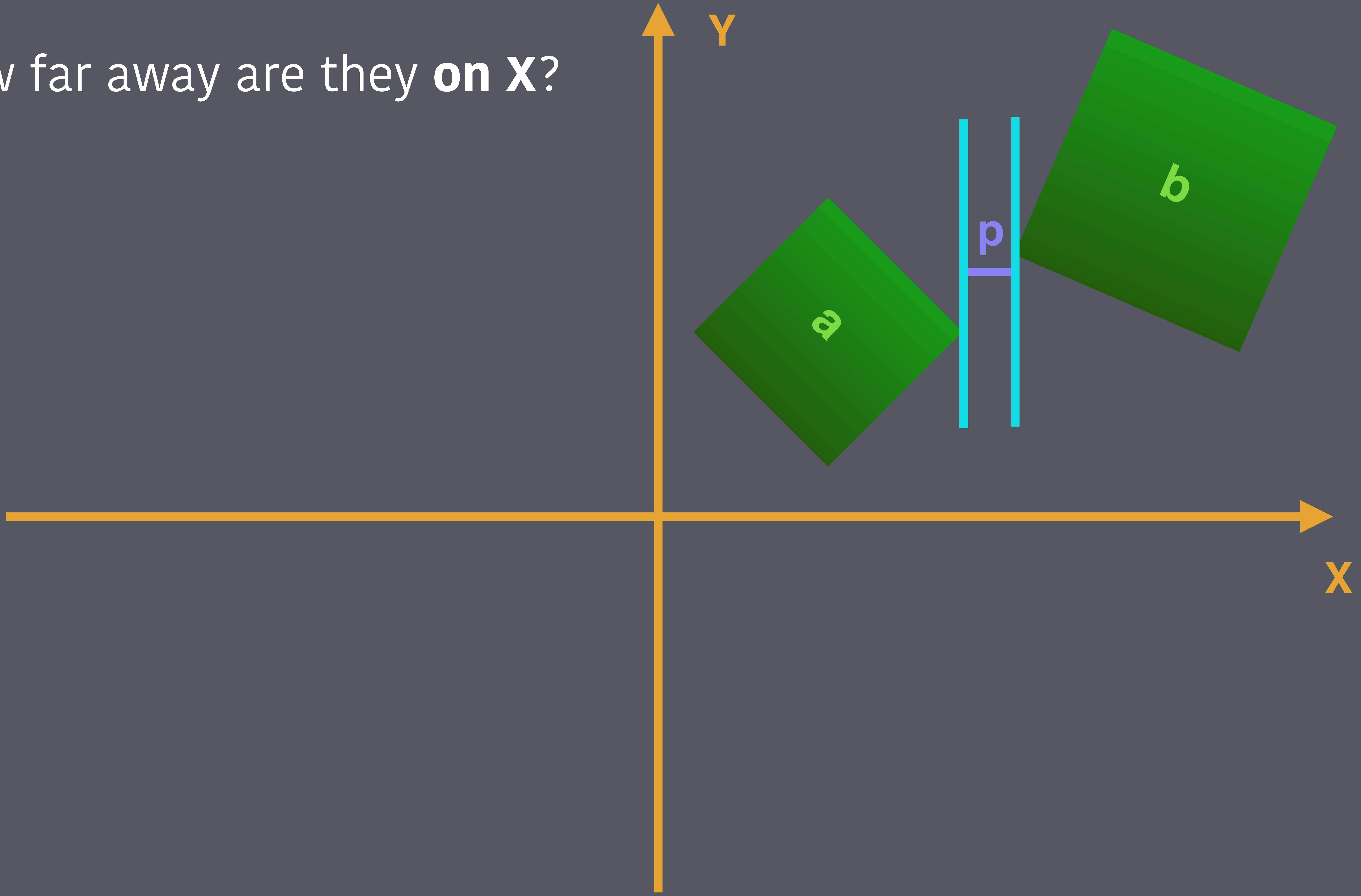
Do the same on the **Y-axis** with box heights if X is not separating.

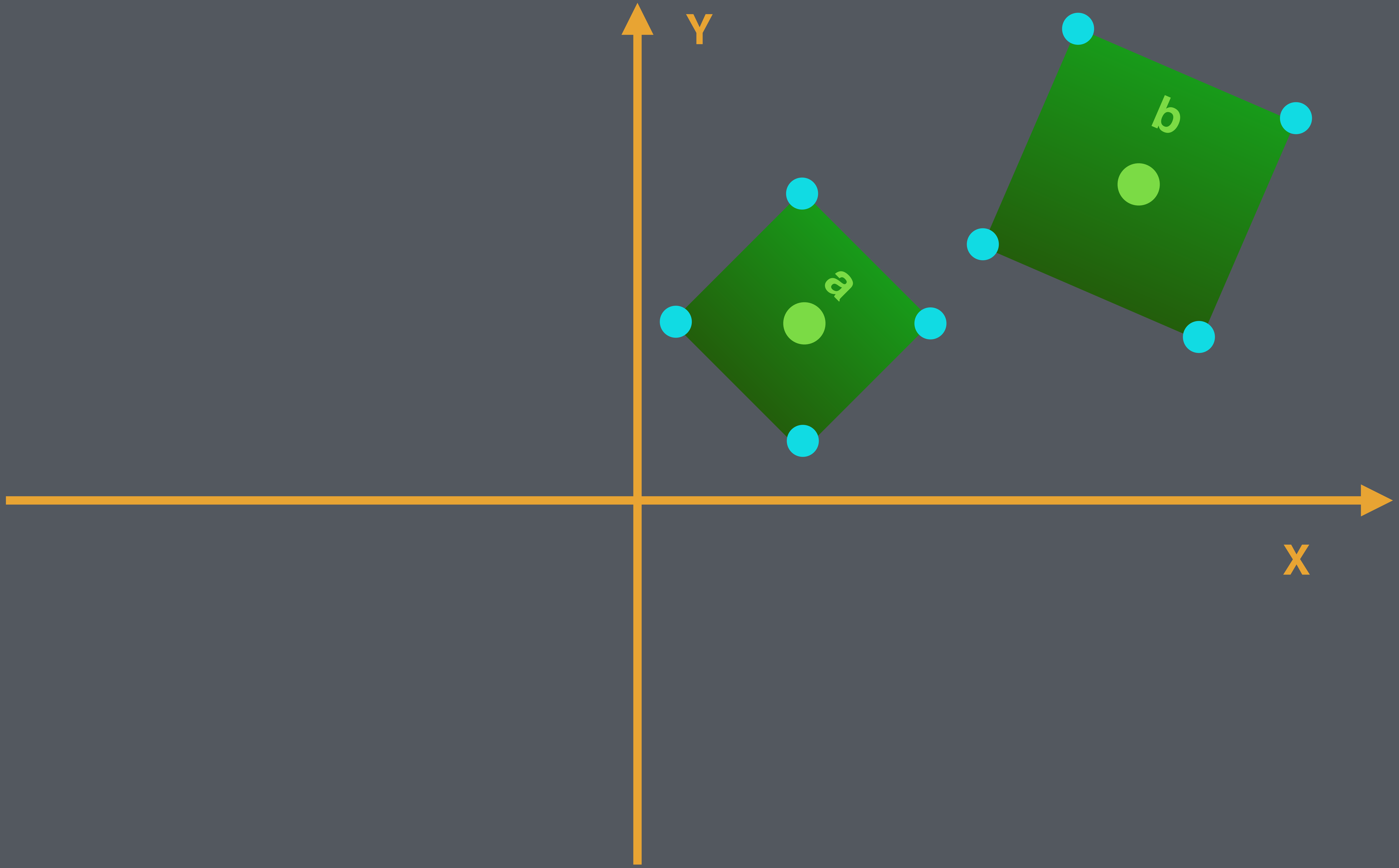
If **neither axis is separating**, we have a **collision**!

How far away are they **on X**?



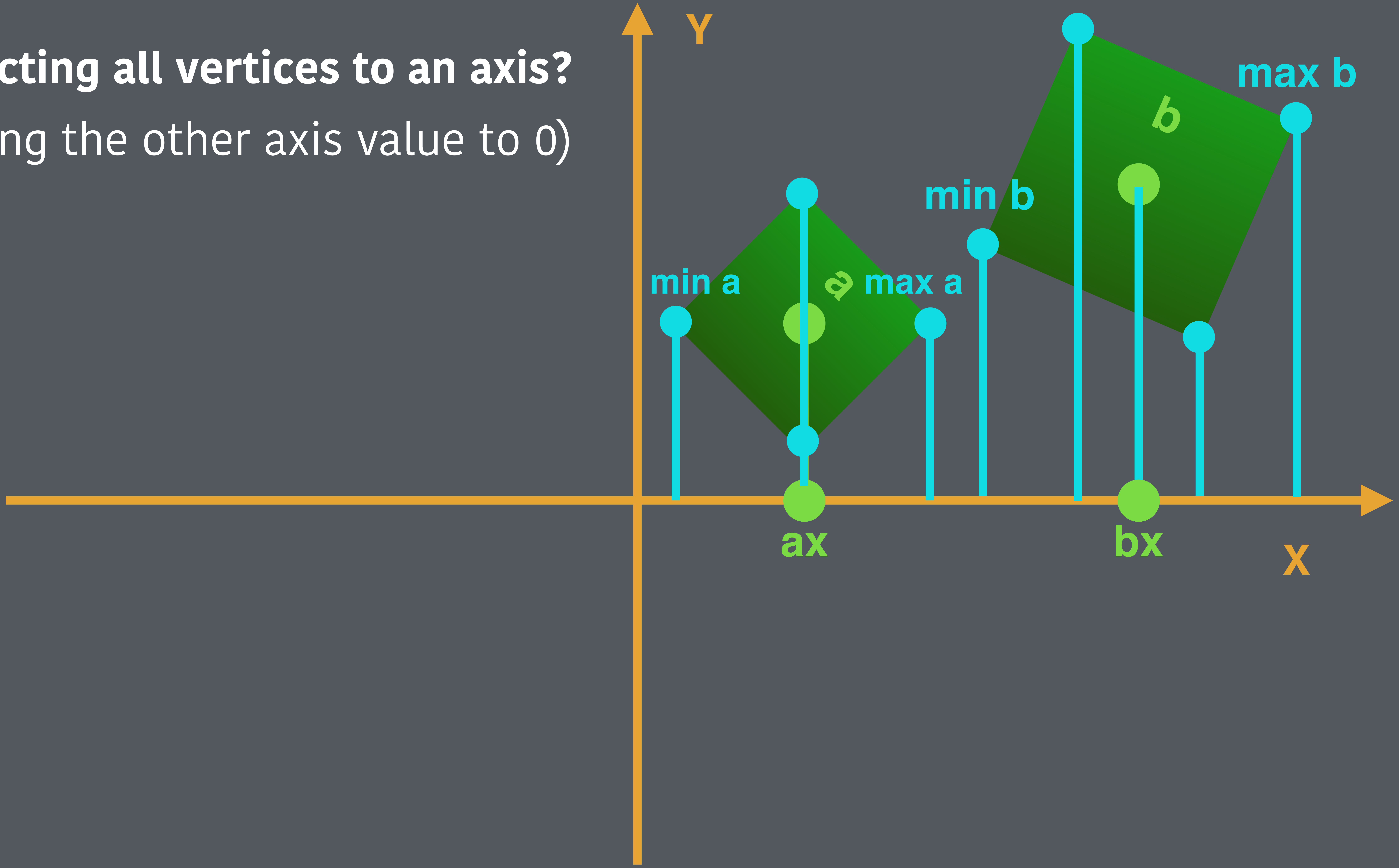
How far away are they **on X**?



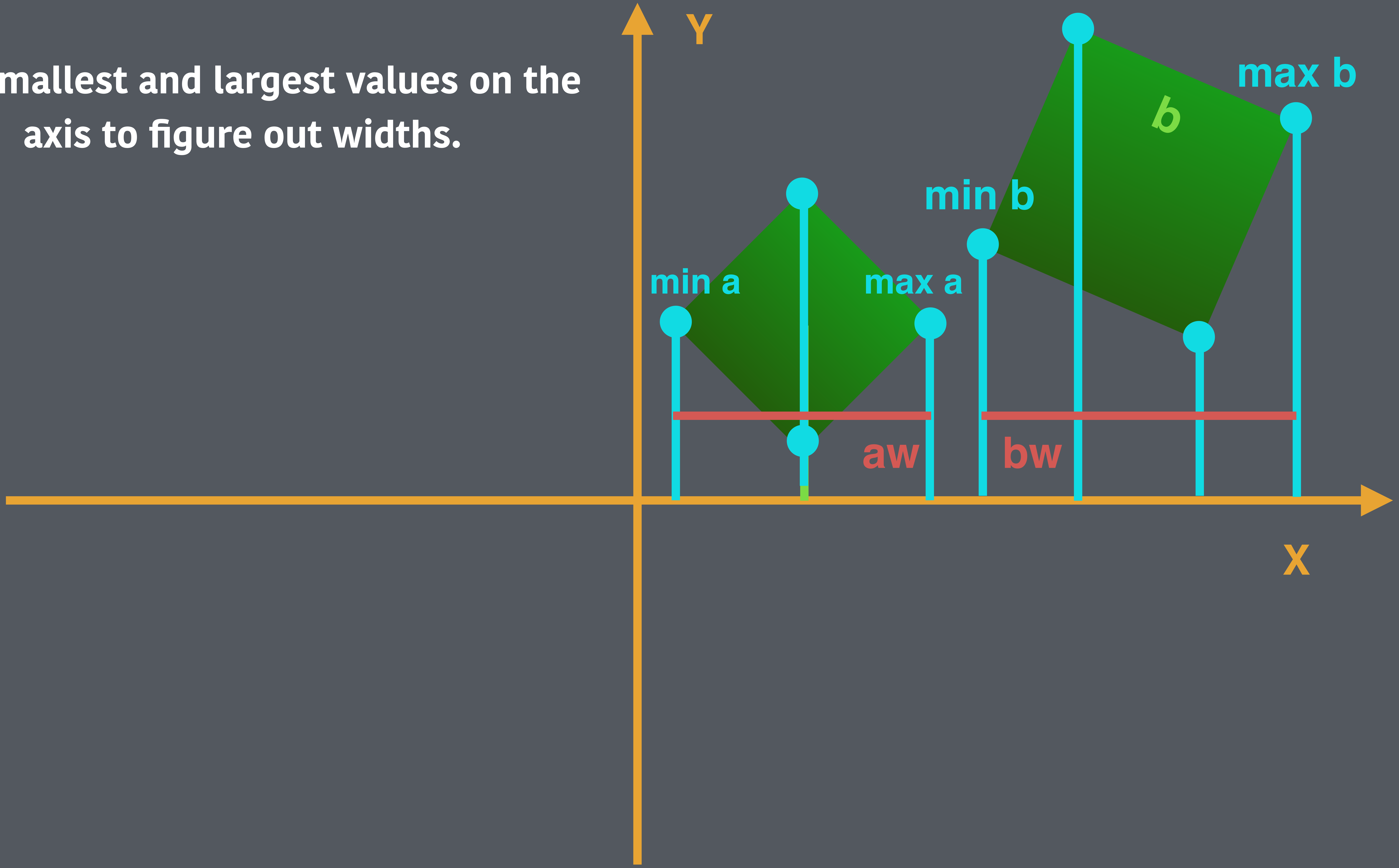

Y
X
a
b

Projecting all vertices to an axis?

(setting the other axis value to 0)



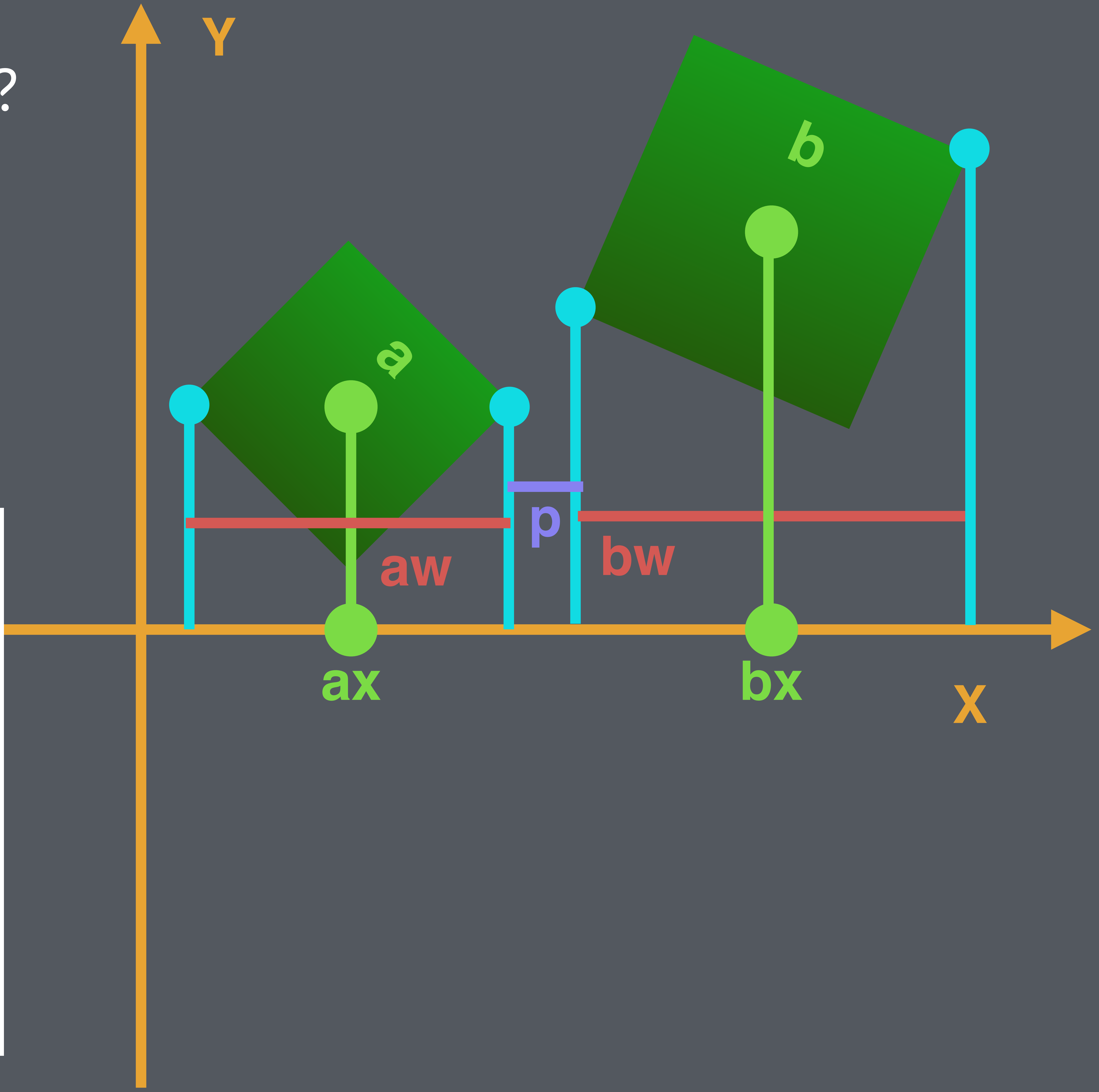
Use smallest and largest values on the axis to figure out widths.



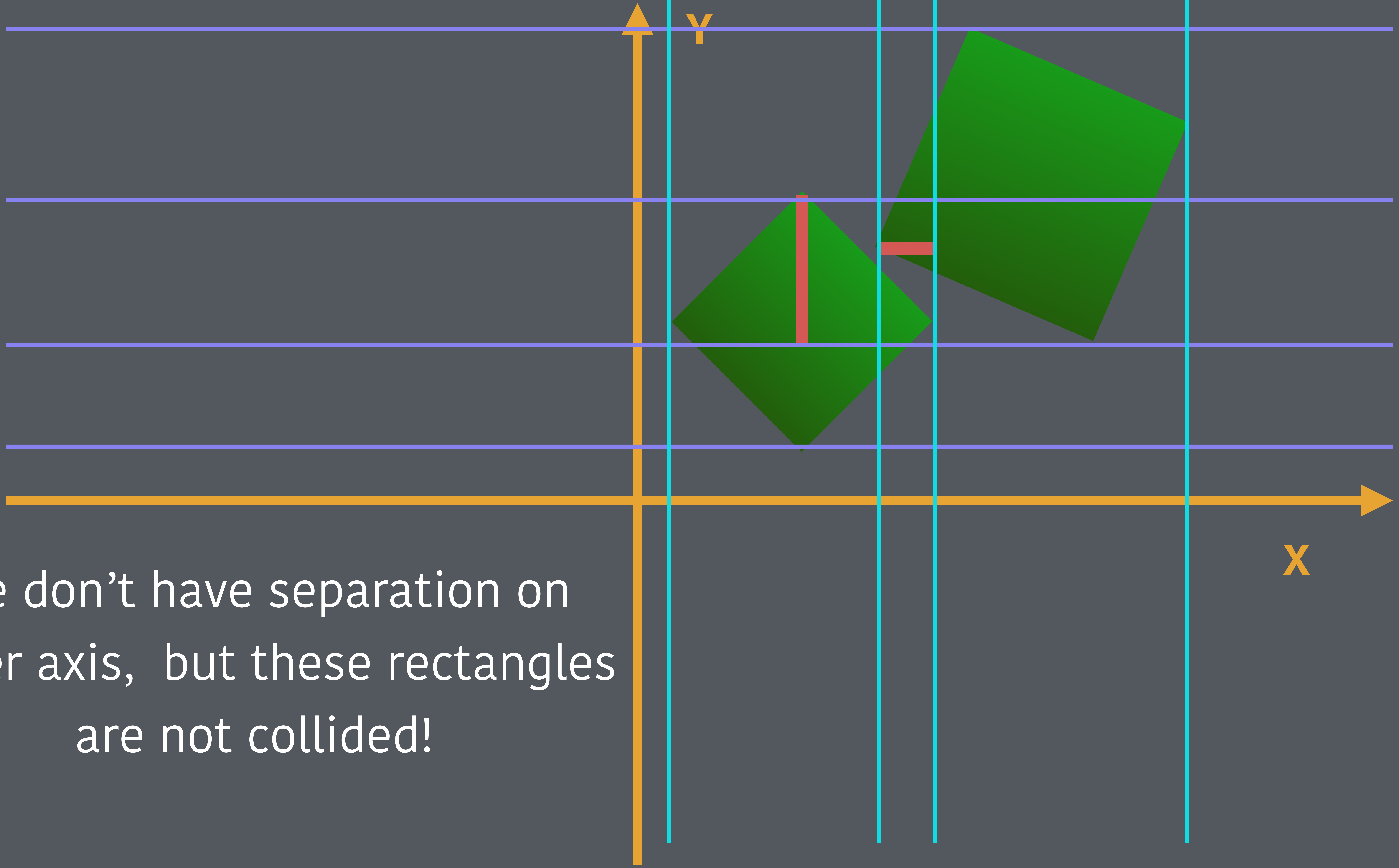
How far away are they on X?

$$p = |x_1 - x_2| - \frac{w_1 + w_2}{2}$$

if p >= 0, we are not colliding!

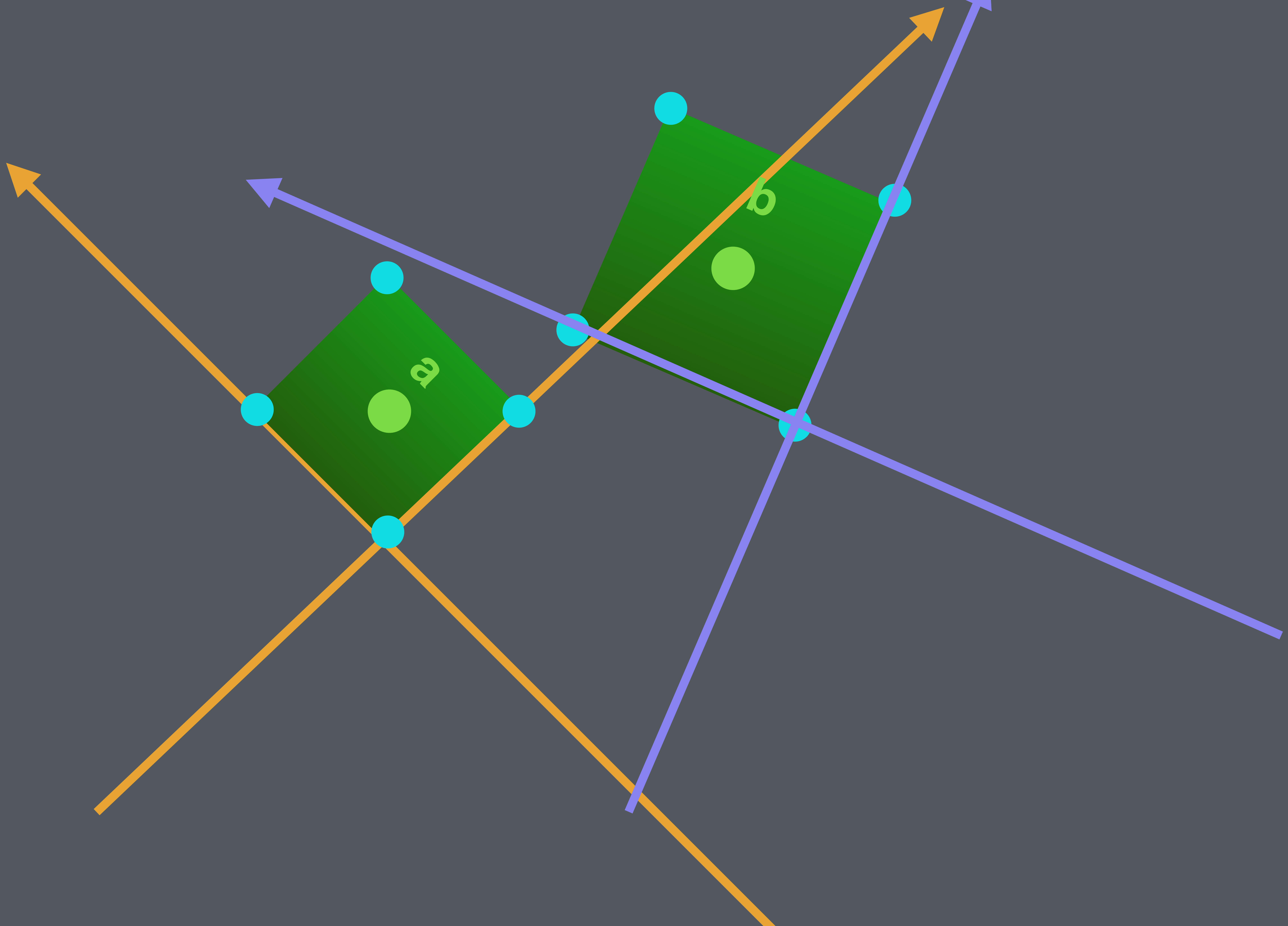


We cannot check rotated separation
on X and Y axes!



We don't have separation on either axis, but these rectangles are not collided!

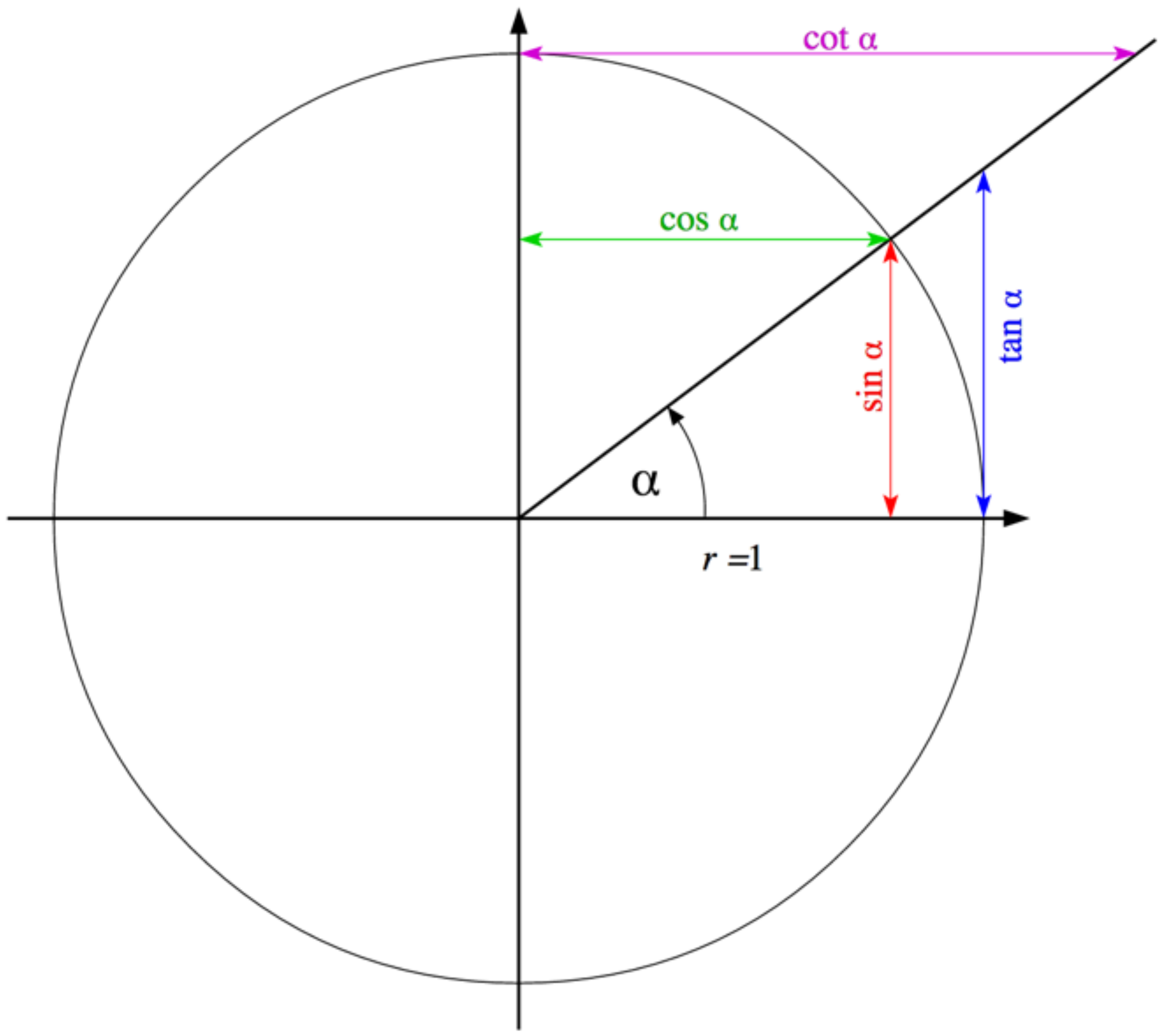
We need to check on **both axes**
of **each rectangle**.



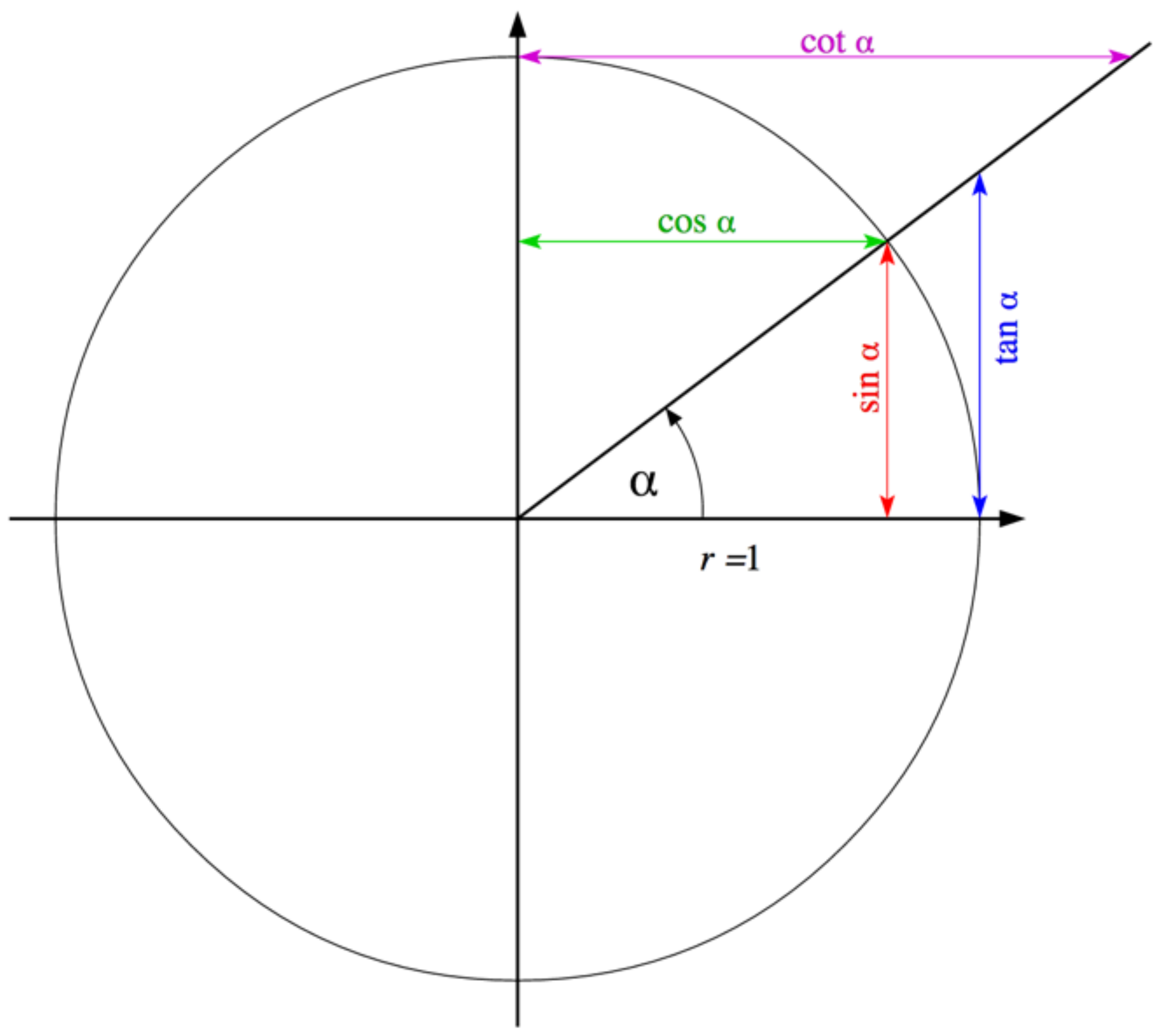
a
b

What is an **axis**?

An **axis** is just a **unit vector** representing a **direction**.



An **axis** is just a **unit vector** representing a **direction**.



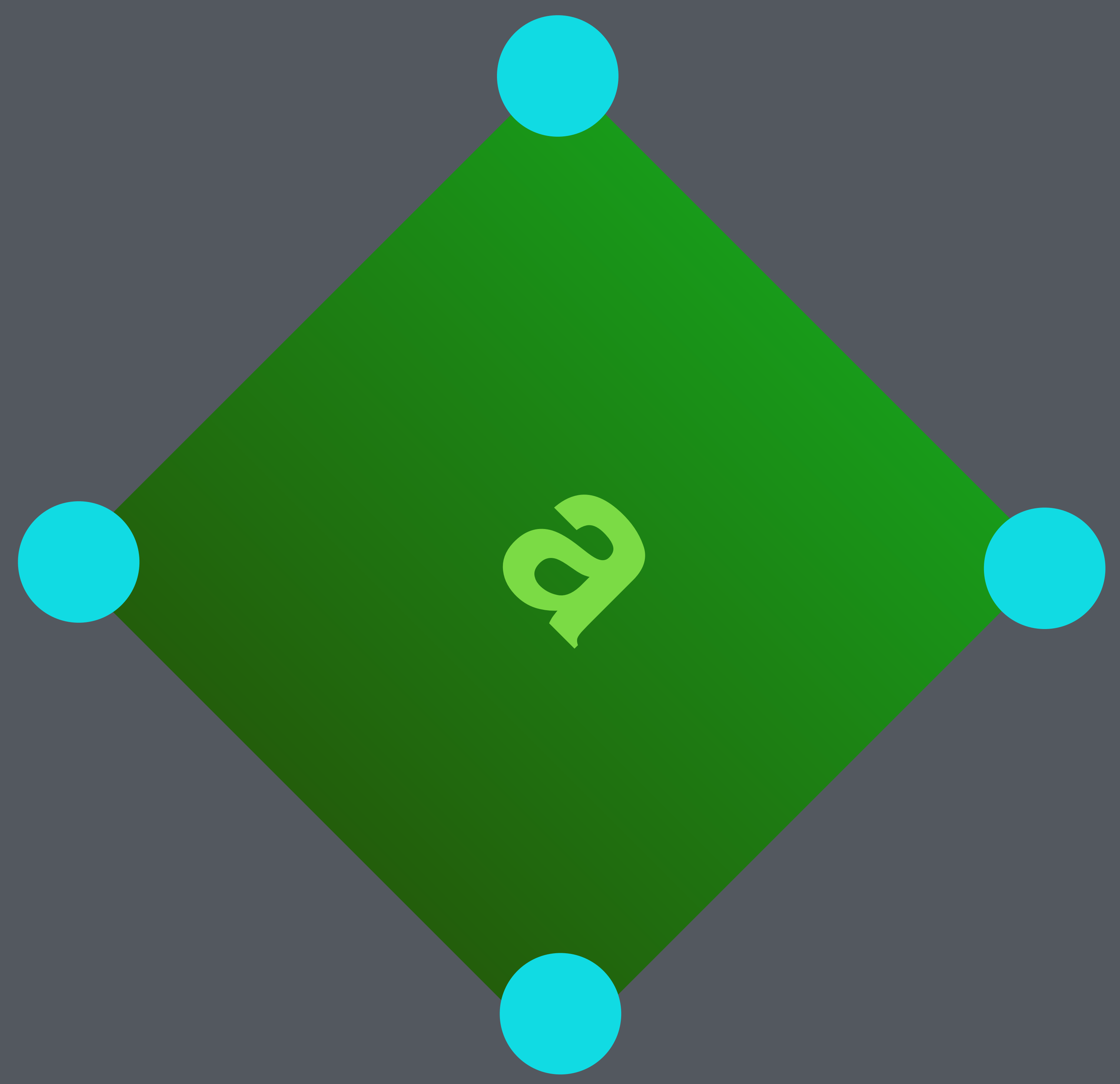
Our usual X axis is (1.0, 0.0) and Y is (0.0, 1.0).

An axis that’s at a 45 degree angle (PI/4) can be represented by **(cos(PI/4), sin(PI/4)).**

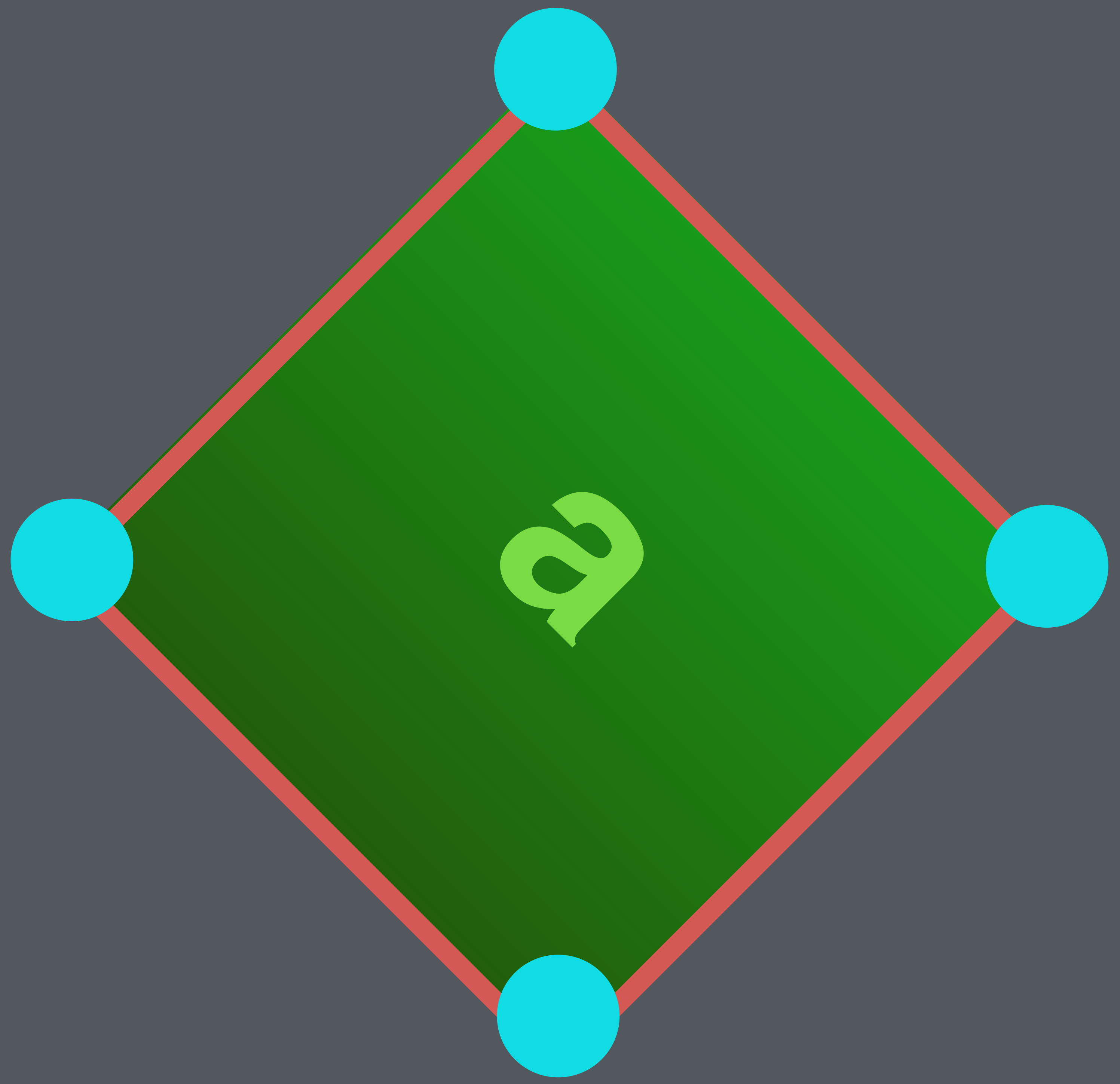
How do we figure out our **rectangle axes**?

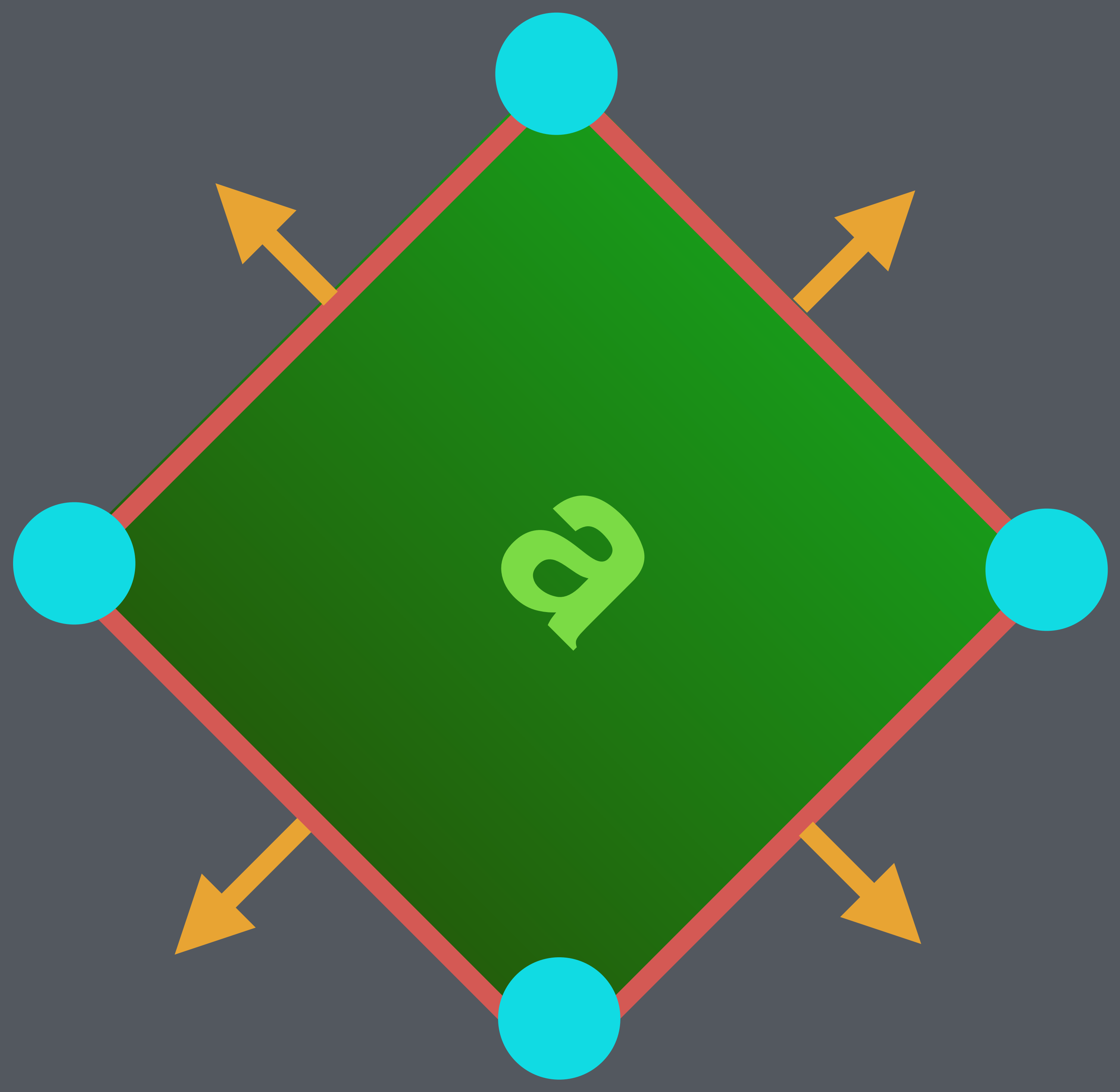
Normals.

Polygon



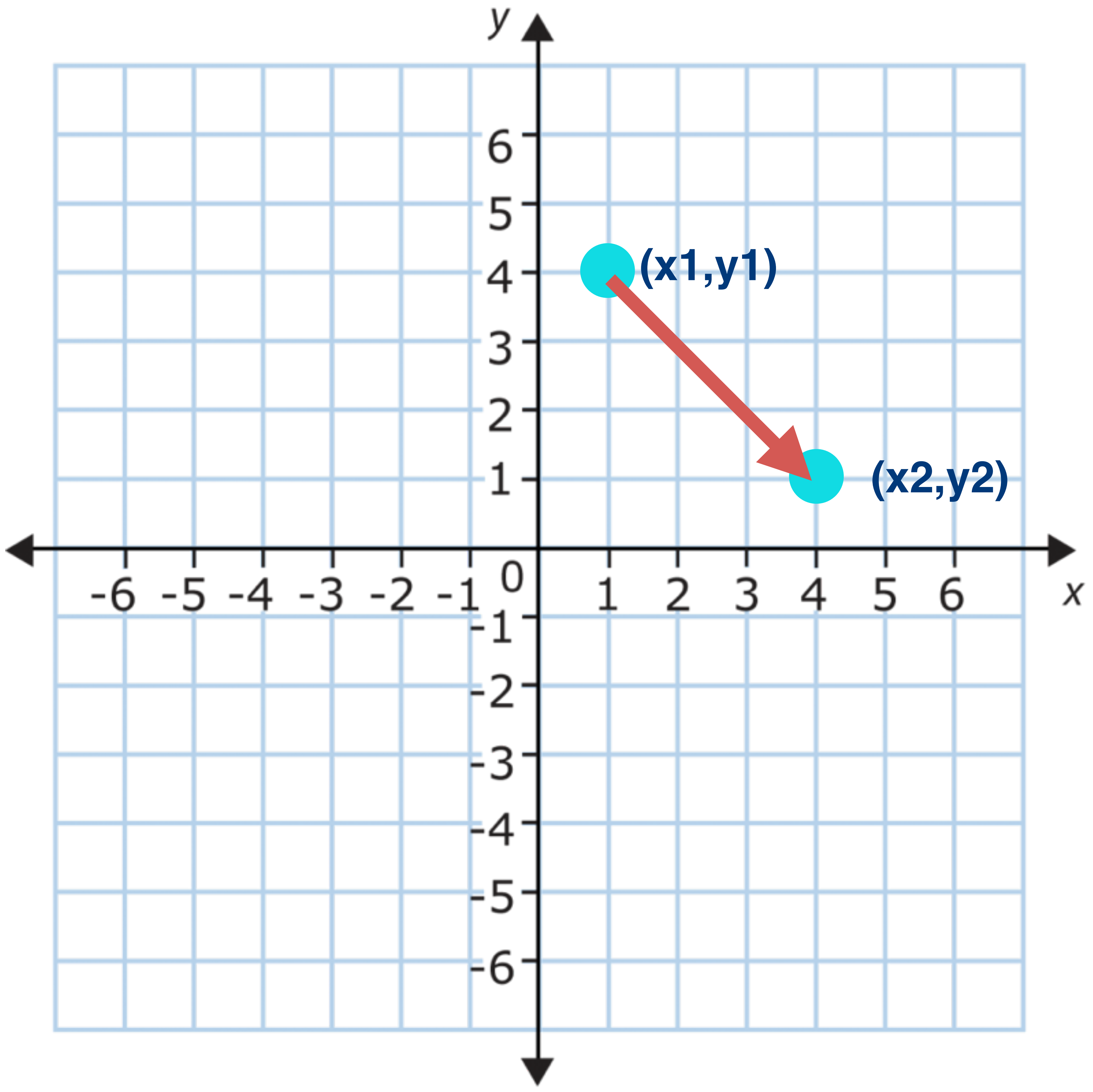
Polygon edges or sides.





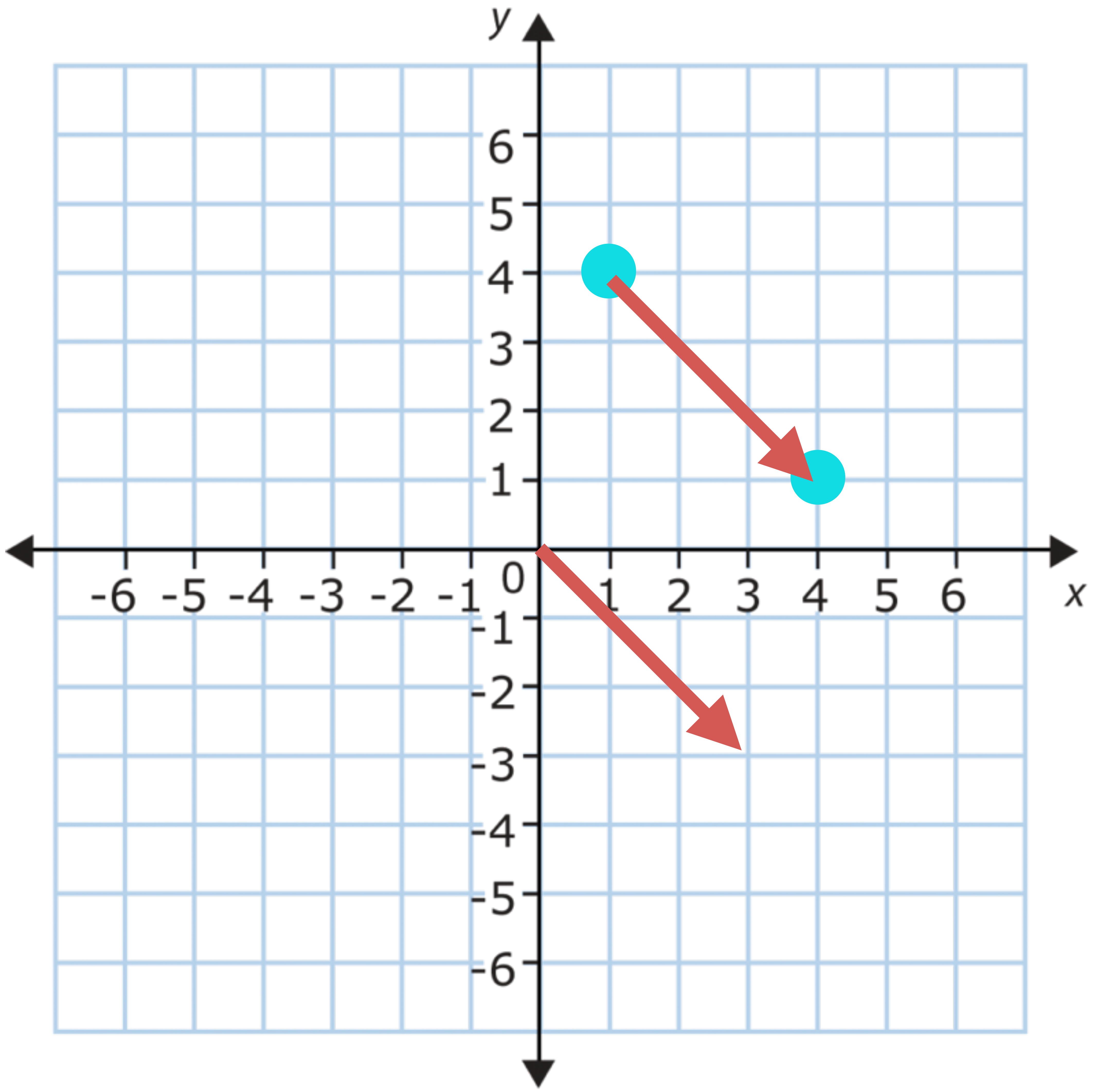
Edge normals.

Normalized (unit) vectors perpendicular to the edge.



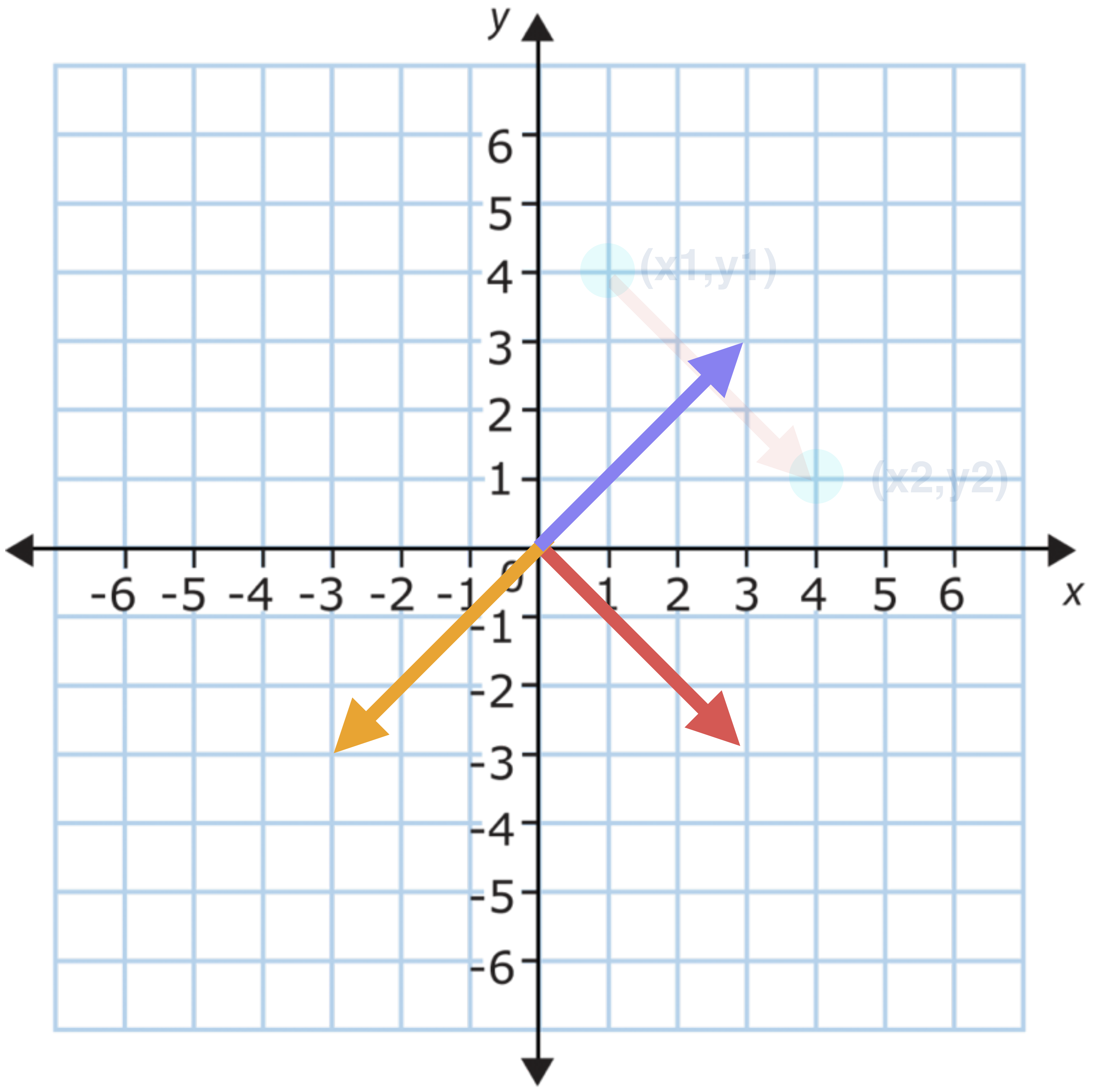
An edge is a vector from one vertex to another.

edge_x = x2-x1
edge_y = y2-y1
edge = (edge_x, edge_y)



An edge is a vector from one vertex to another.

edge_x = x2-x1
edge_y = y2-y1
edge = (edge_x, edge_y)



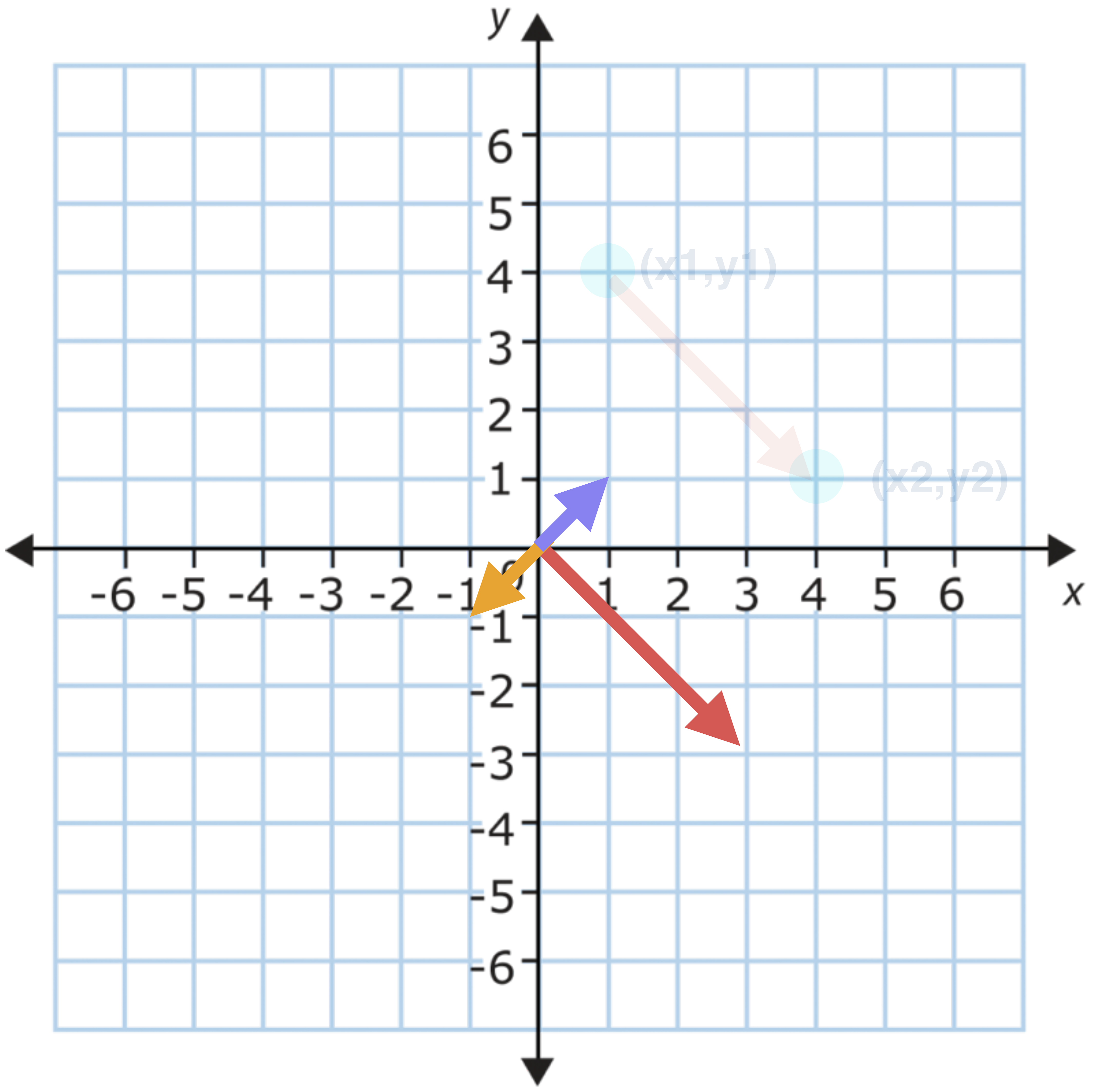
An edge is a vector from one vertex to another.

edge_x = x2-x1
edge_y = y2-y1
edge = (edge_x, edge_y)

Its normals are the vectors perpendicular to that vector.
normal1 = (edge_y, -edge_x)

and

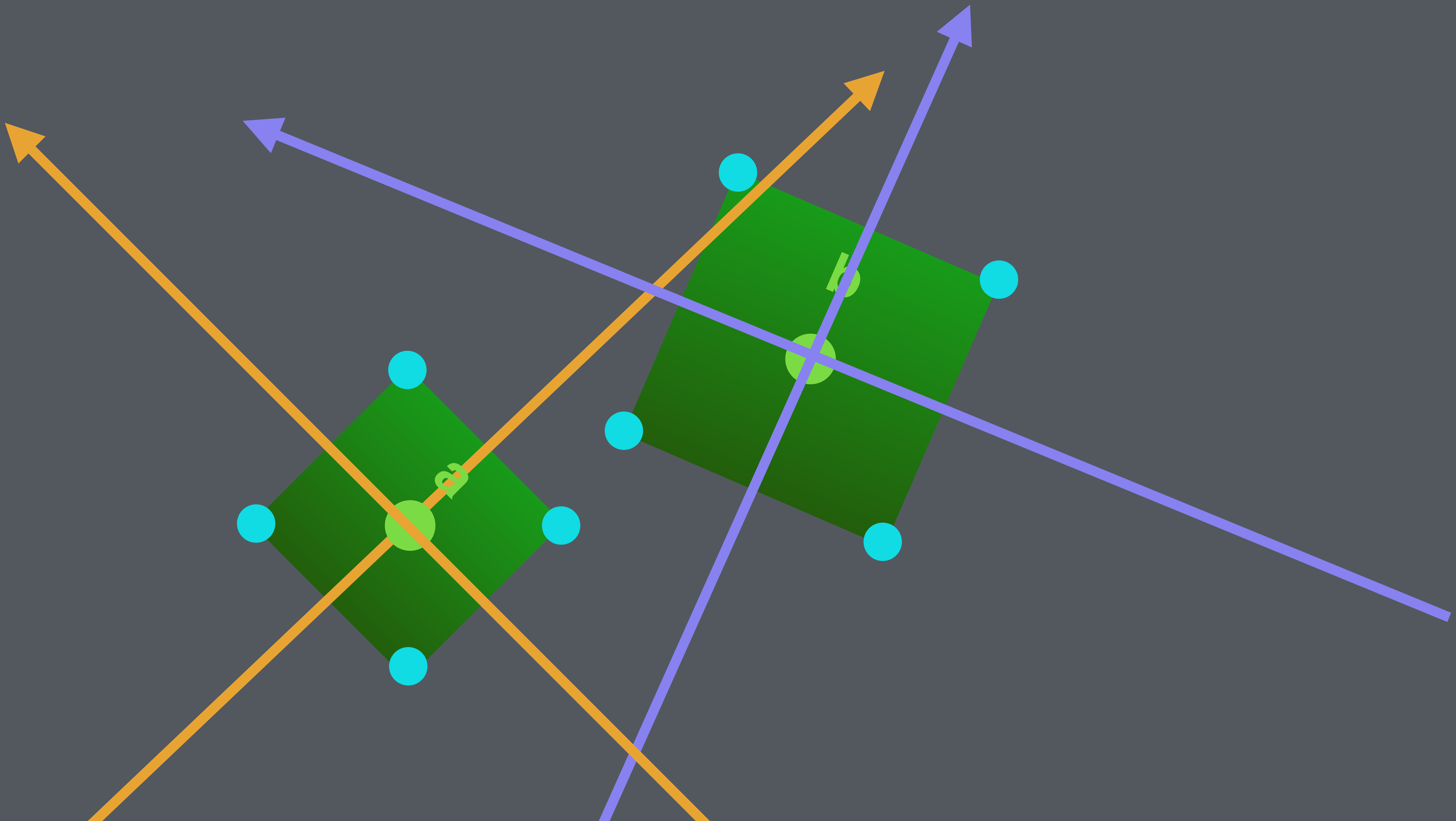
normal2 = (-edge_y, edge_x)



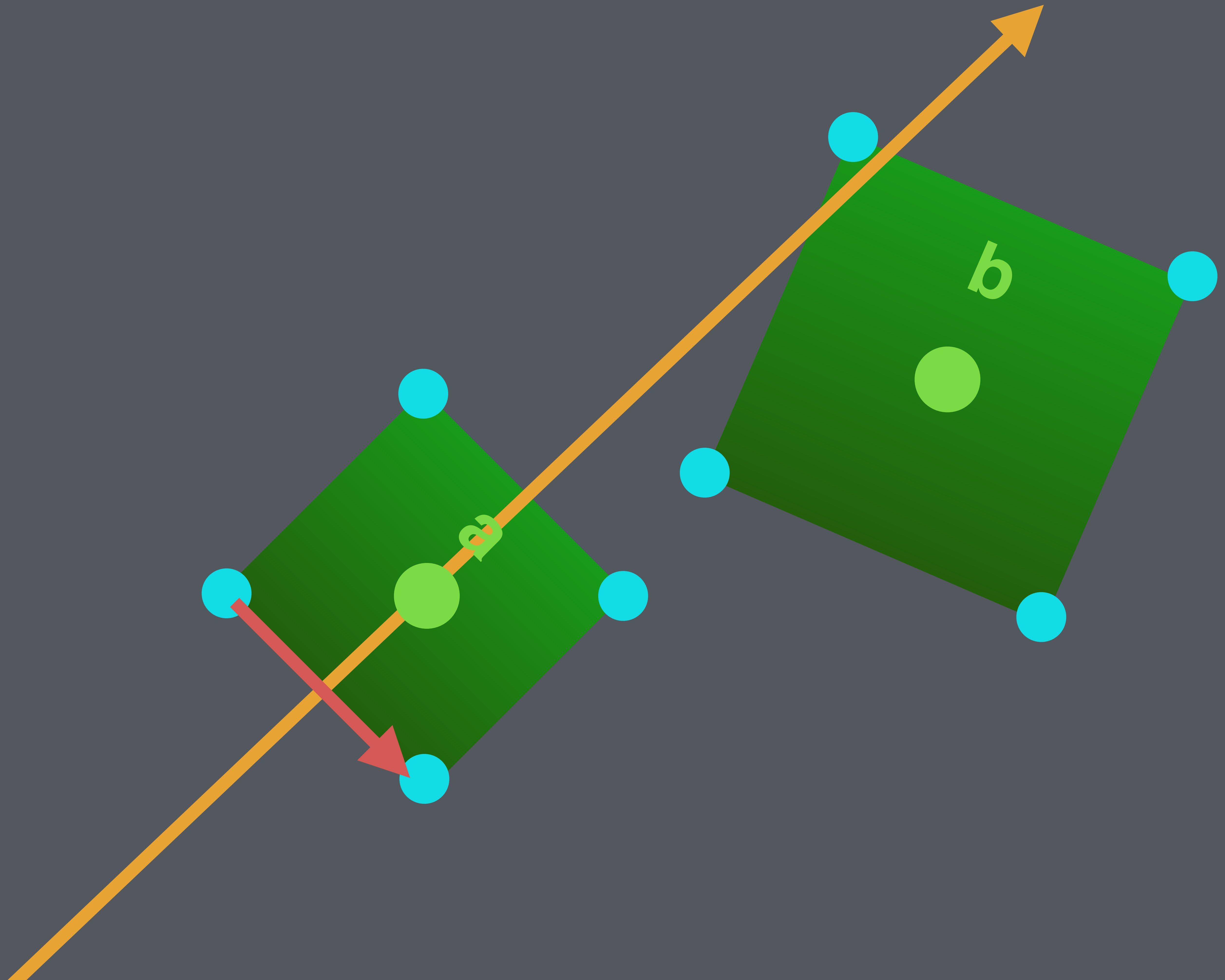
Now, normalize the normal vectors.

```
length = sqrt(x*x + y*y)
x /= length
y /= length
```

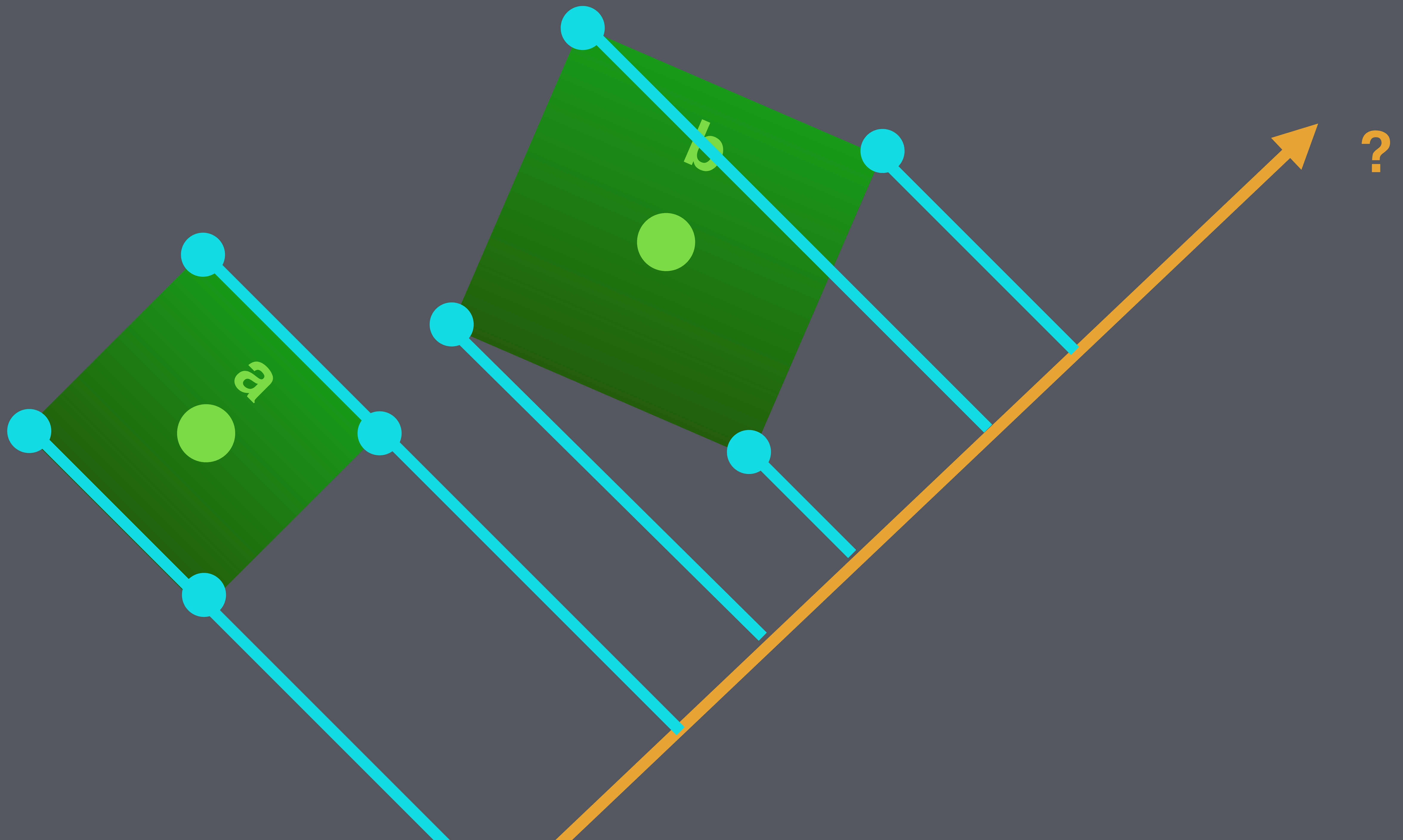
Our **normals** are the **axes** on which we **check for separation.**



For each edge **find the normal.**



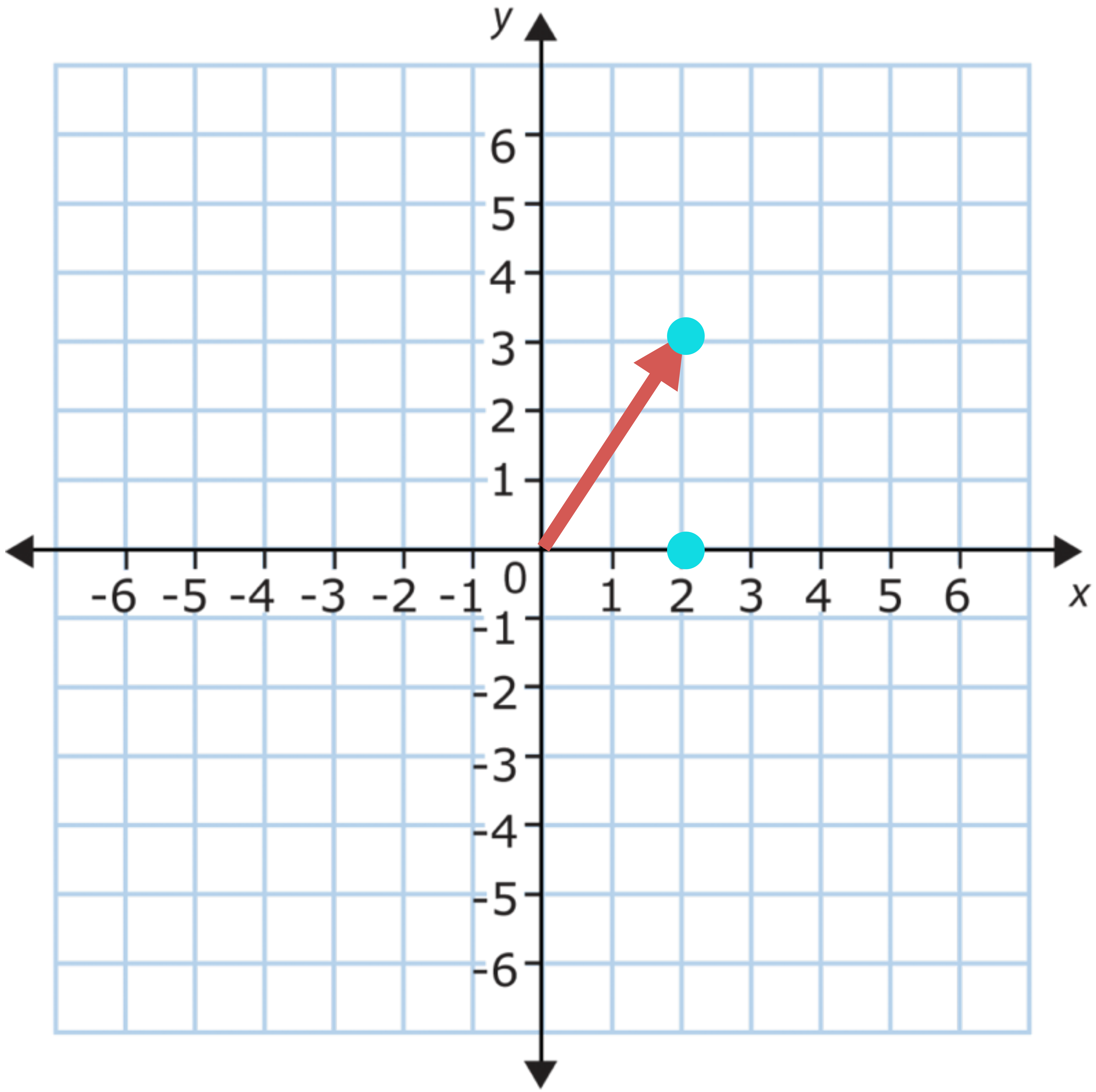
Projecting onto an **arbitrary axis.**



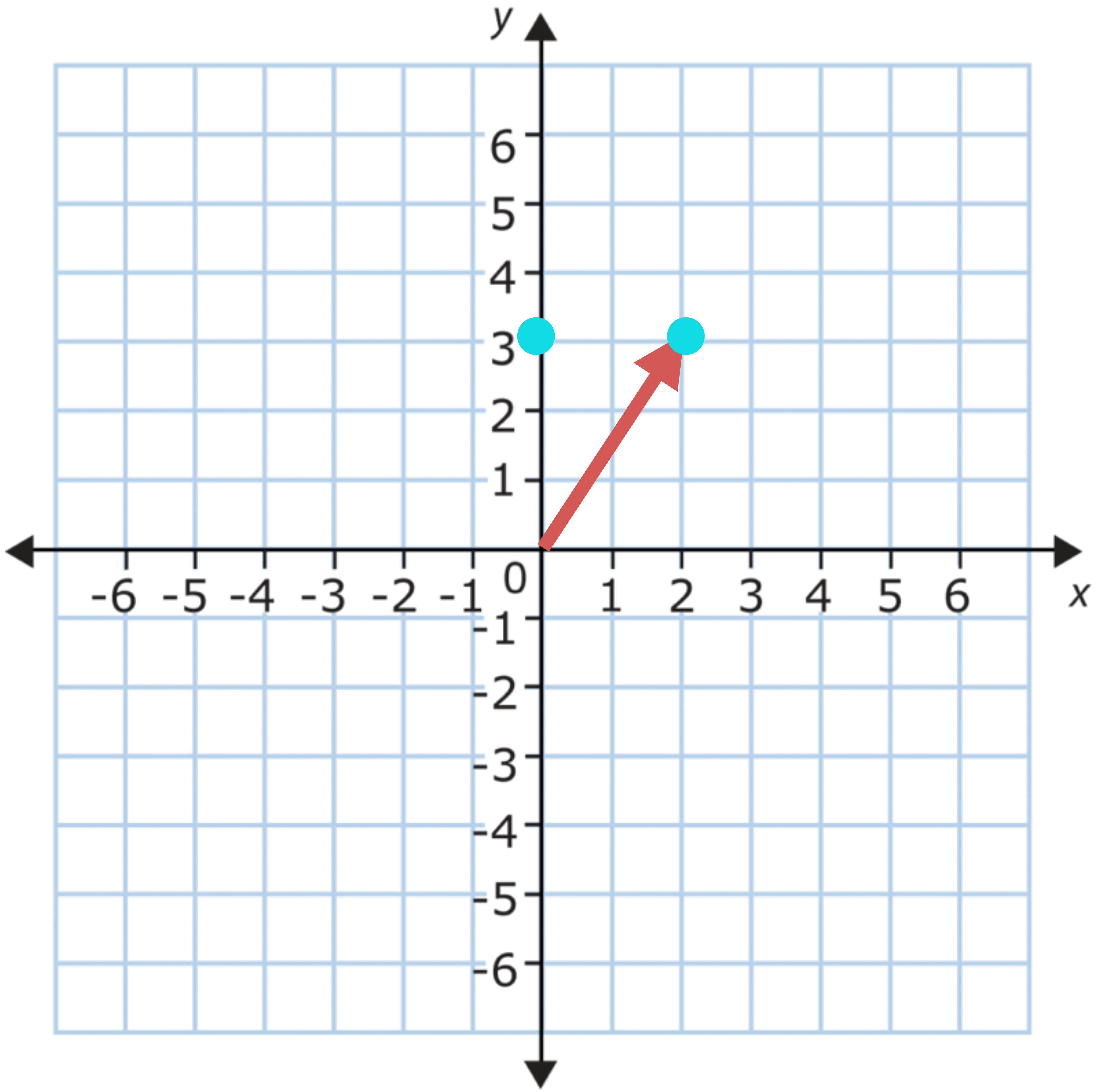
The dot product.

```
(x1*x2) + (y1*y2)
```

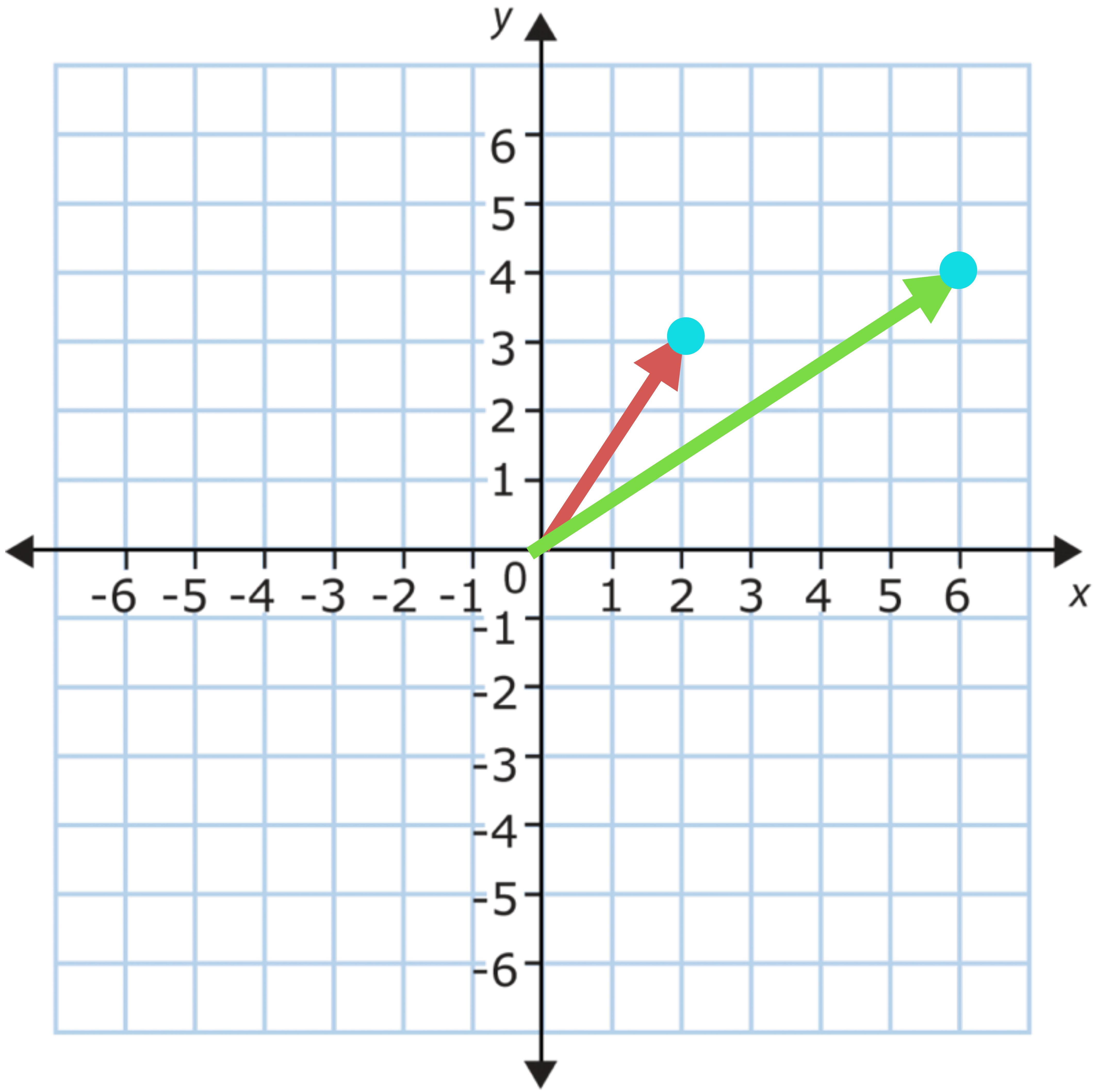
Applies one vector to another.



$$(2,3) \cdot (1,0) = (2*1) + (3 * 0) = 2$$



$$(2,3) \cdot (0,1) = (2*0) + (3 * 1) = 3$$



Normalize (6,4):

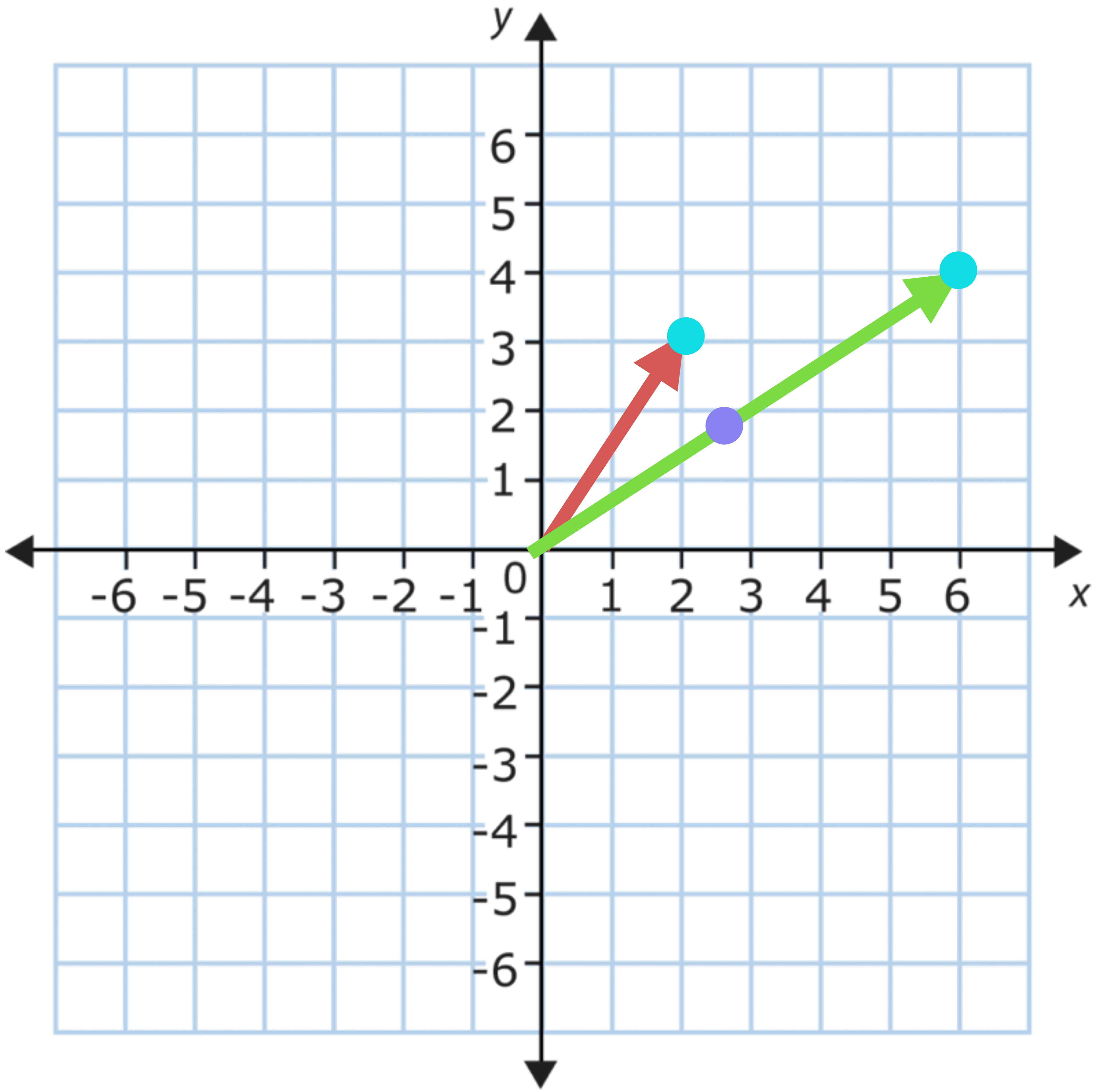
length = sqrt(6*6 + 4*4) = 7.2111

x = 6 / 7.2111 = 0.832

y = 4 / 7.2111 = 0.5547

(2,3) · (0.832,0.555) = (2*0.832) + (3 * 0.555)

= 1.664 + 1.665 = **3.329**



Normalize (6,4):

length = sqrt(6*6 + 4*4) = 7.2111

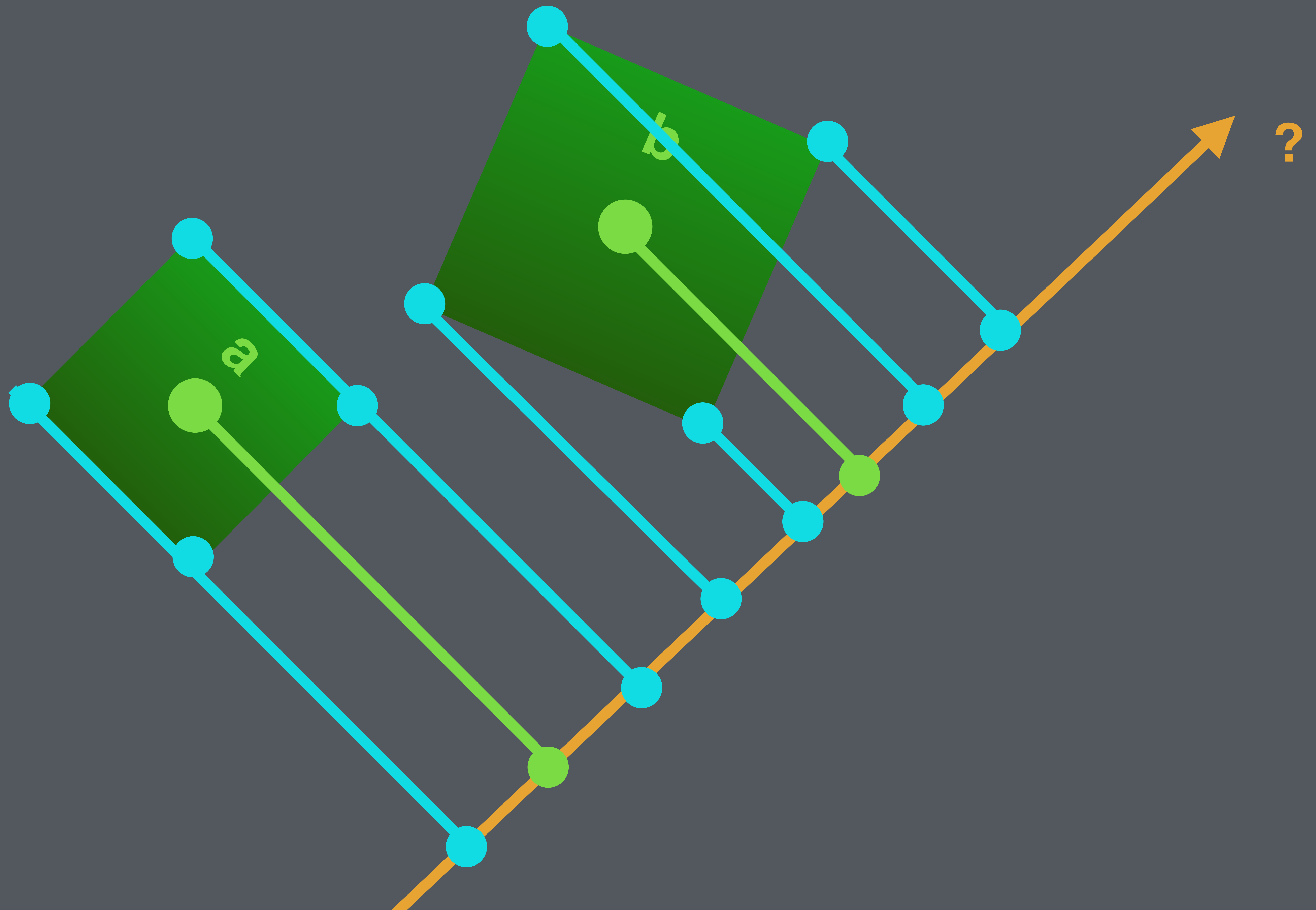
x = 6 / 7.2111 = 0.832

y = 4 / 7.2111 = 0.5547

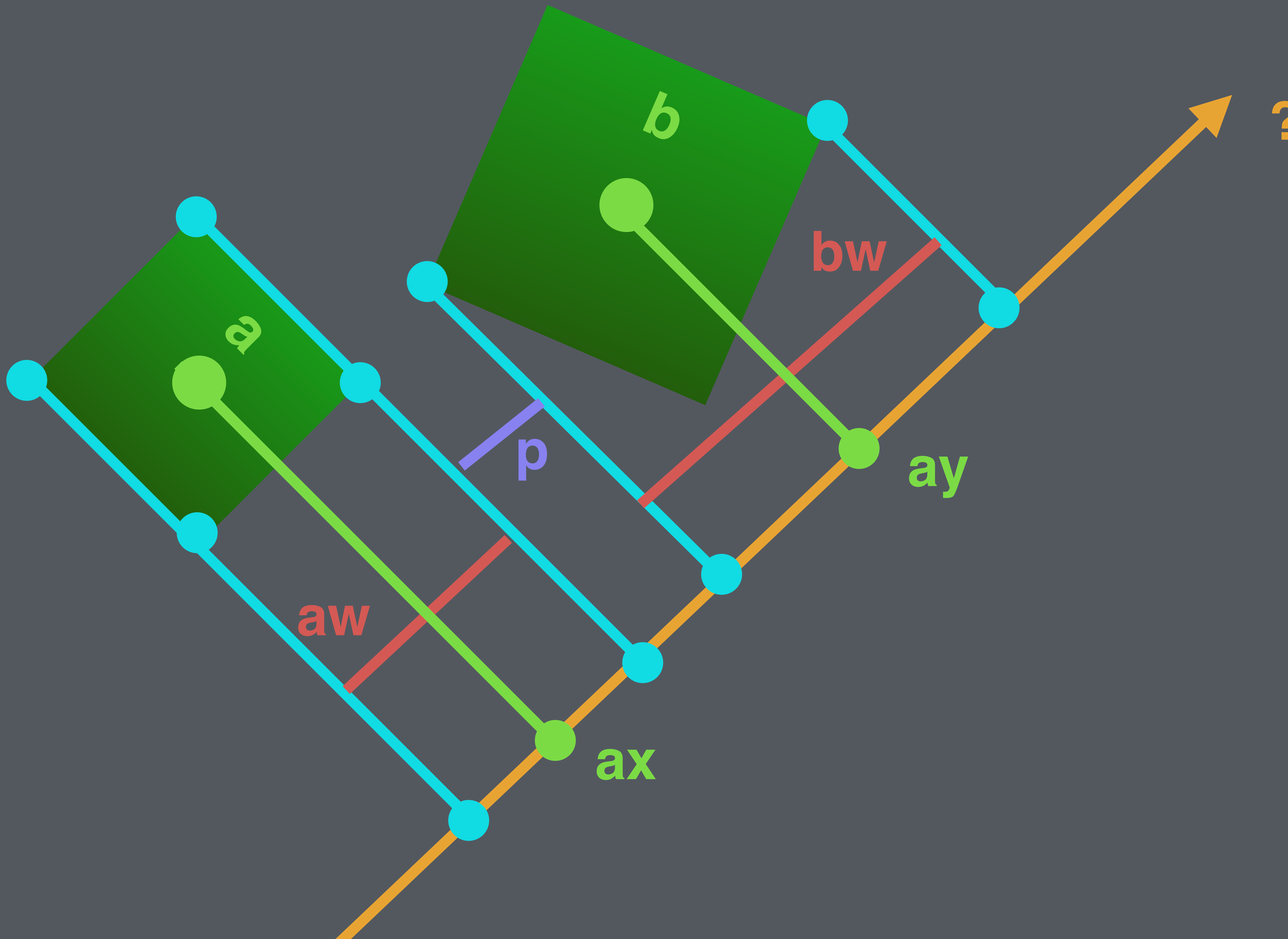
(2,3) · (0.832,0.555) = (2*0.832) + (3 * 0.555)

= 1.664 + 1.665 = **3.329**

Find dot product of each vertex with the normalized axis vector.



How far away are they **on this axis**?



$$p = |x_1 - x_2| - \frac{w_1 + w_2}{2}$$

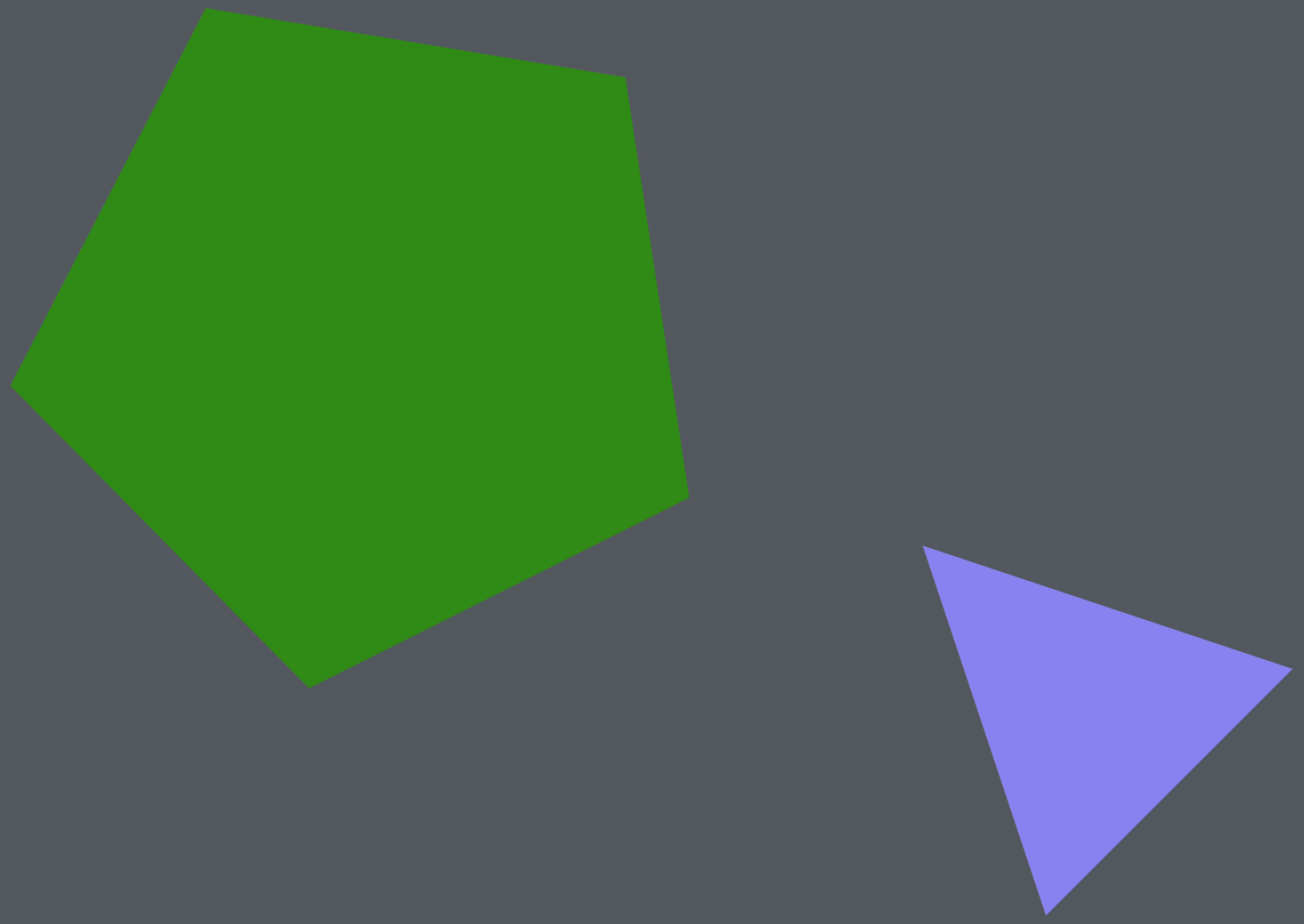
if p >= 0, we are not colliding!

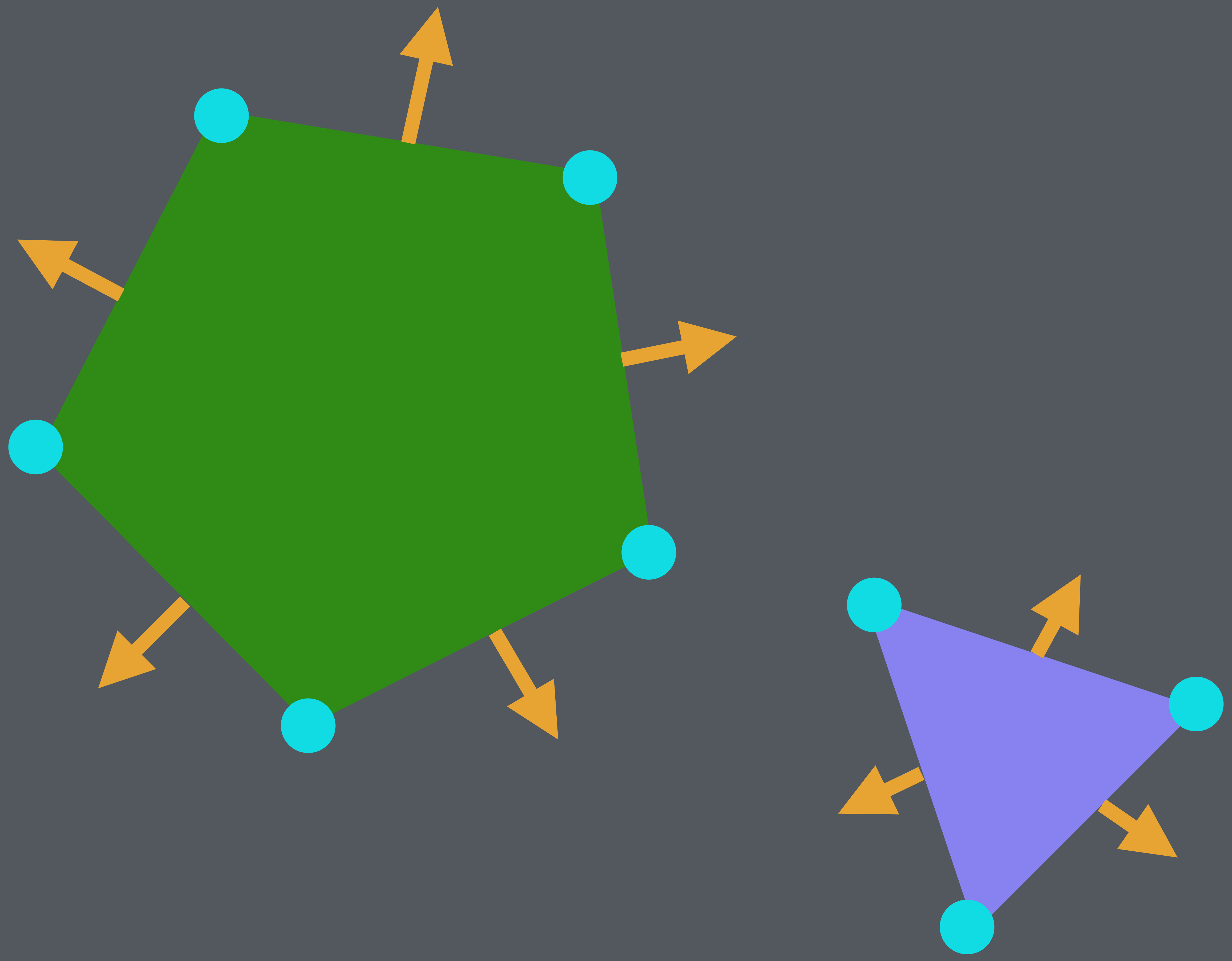
Check the **separation** on **each of the 4 normal axes**

(we don't have to check all 8 since 2 sides of each rectangle are parallel).

If on **any axis**, there is a **separation**, the collision is **not occurring**.

Arbitrary polygon collision.

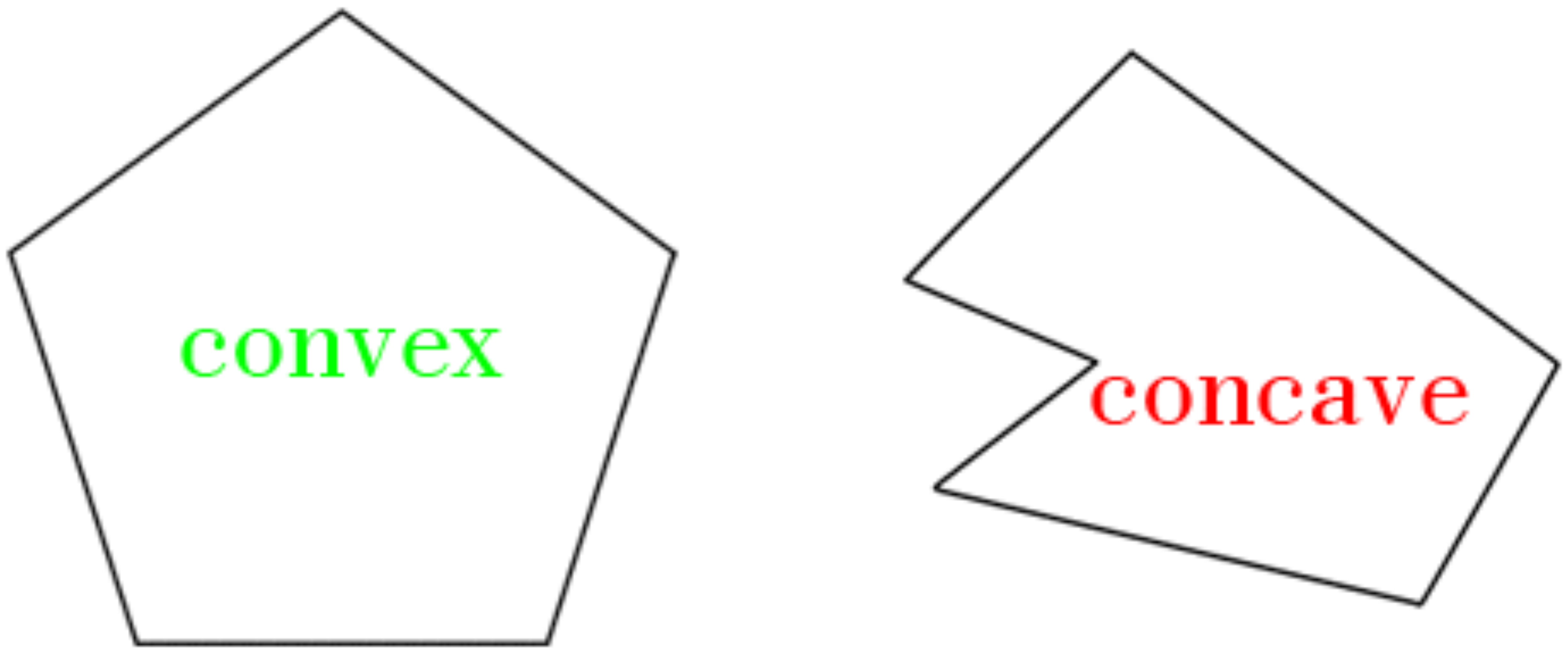




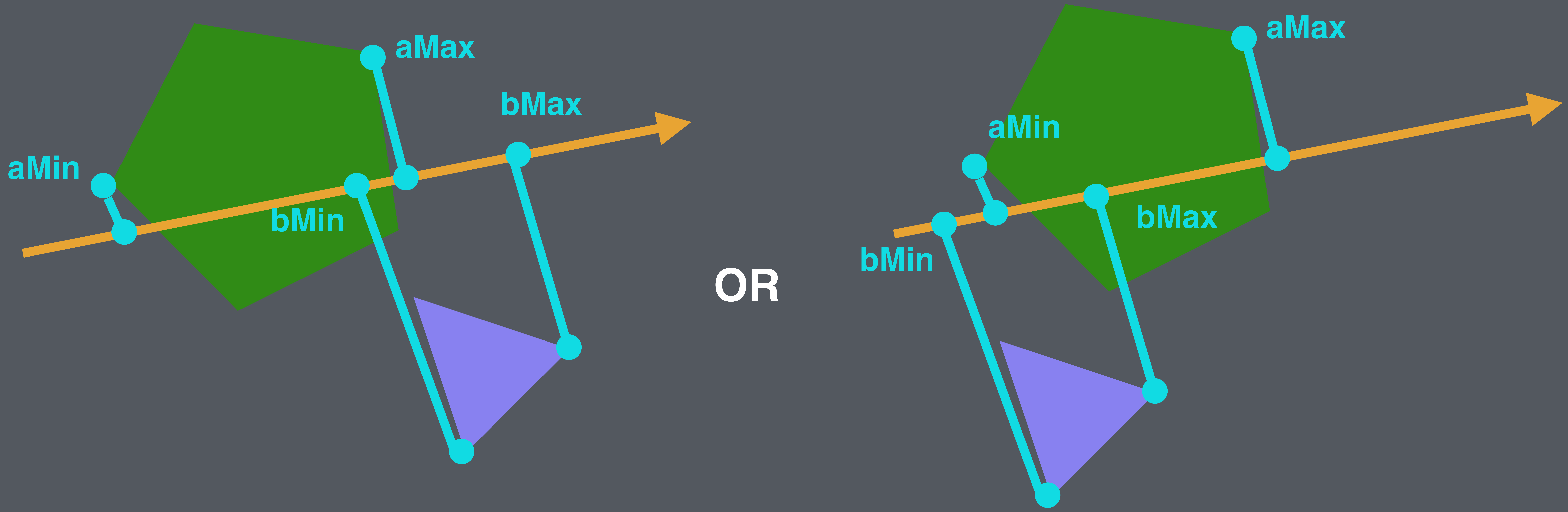
If aMin <= bMax and aMax >= bMin, we have a collision on this axis

Only works with **convex polygons**!

(every internal angle < 180 degrees
and it's not self intersecting)

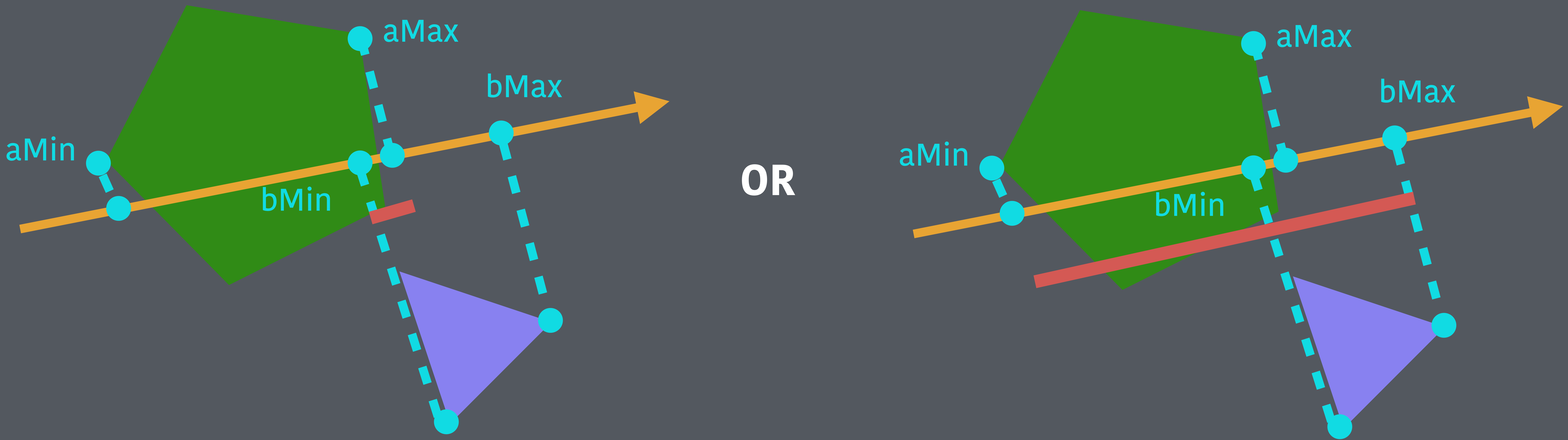


Responding to collisions.



If aMin <= bMax and aMax >= bMin, we have a collision on this axis

Find the **smaller penetration distance** for each axis



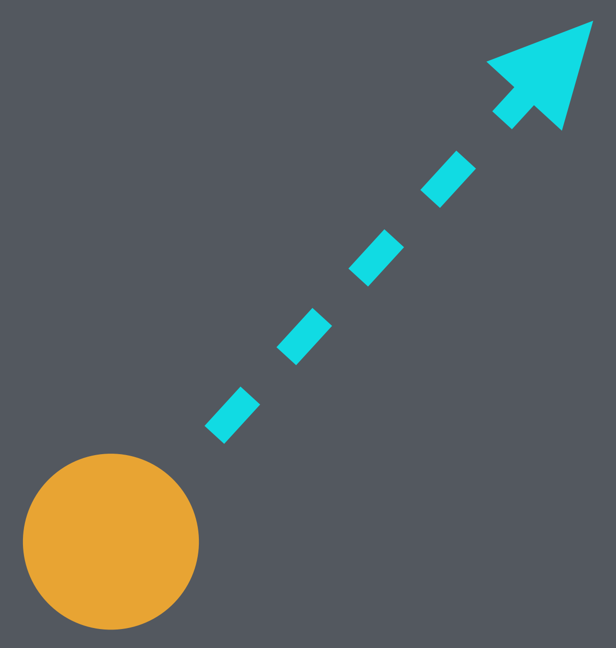
aMax - bMin OR bMax - aMin

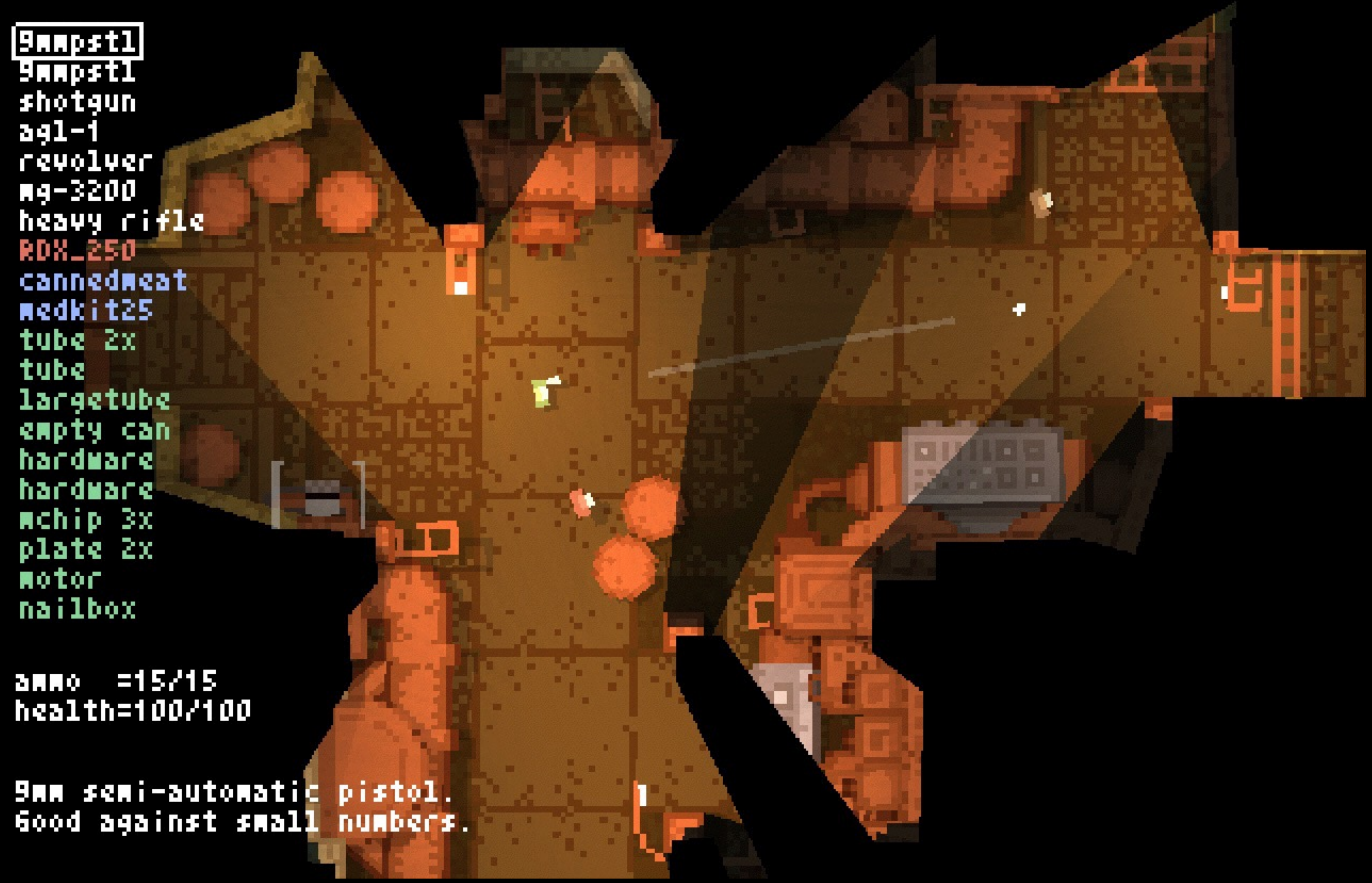
Raycasting.

What is a **ray**?

A ray has an origin position and a direction.

It can be defined as a two vectors, one defining the position and another (unit!) vector defining the direction.





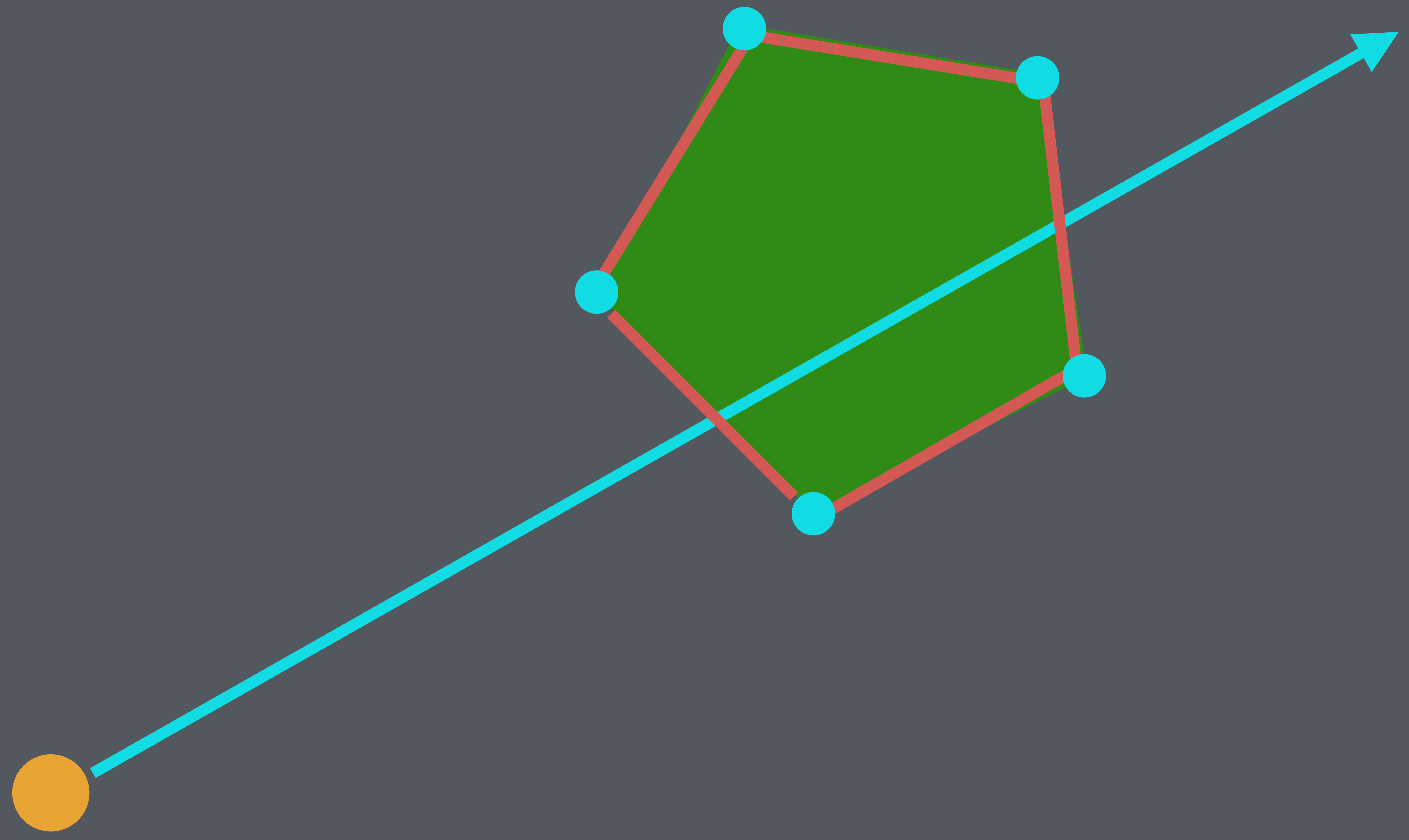
9mmpstl
9mmpstl
shotgun
agl-1
revolver
mg-3200
heavy rifle
RDX_250
cannedmeat
medkit25
tube 2x
tube
largetube
empty can
hardware
hardware
mchip 3x
plate 2x
motor
nailbox
ammo =15/15
health=100/100
9mm semi-automatic pistol.
Good against small numbers.

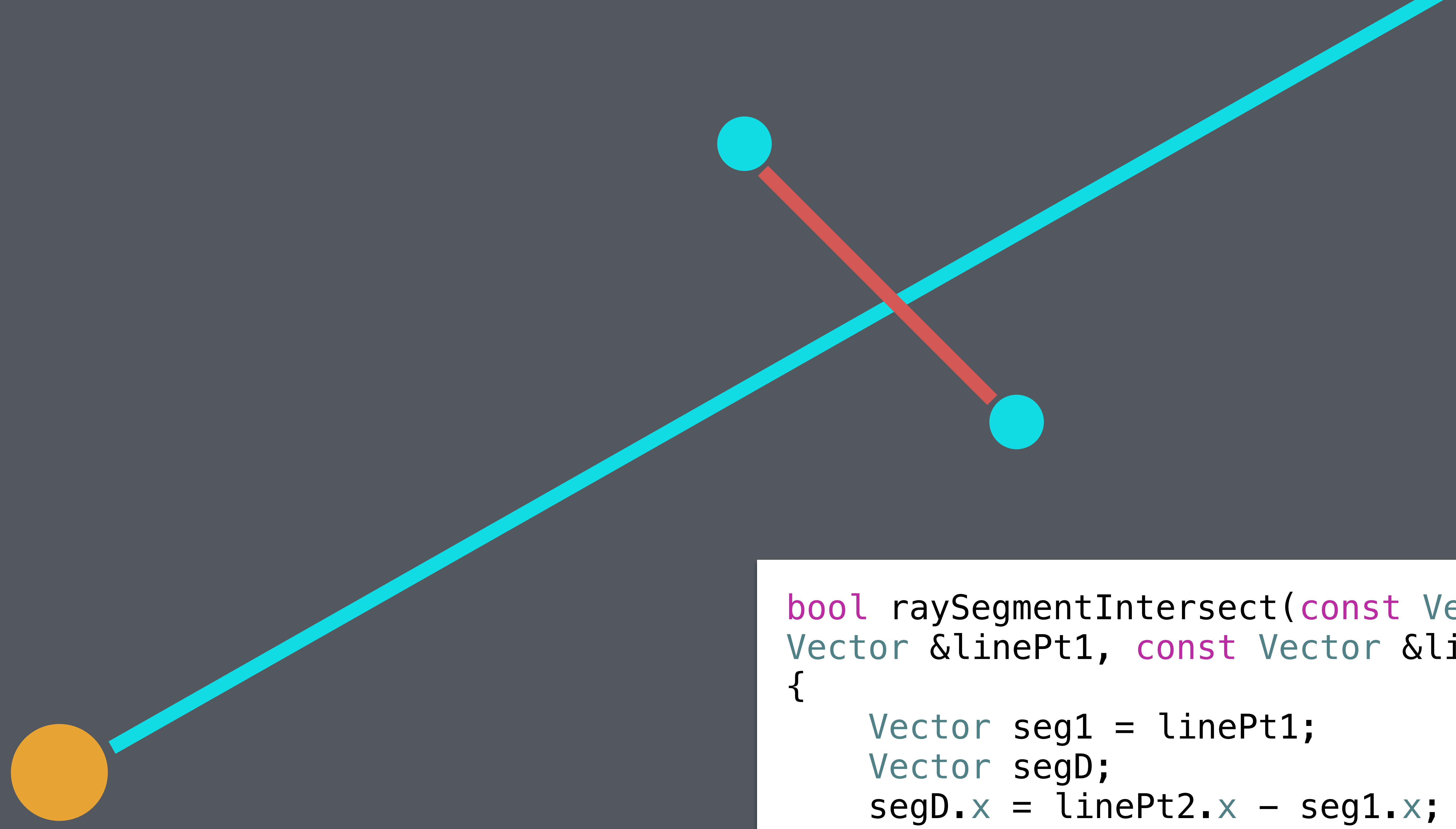

Drop
Drop
B
043

ZR
X
Y
29
A
Back
B
00
477


PICKED UP BODY ARMOR
PICKED UP BOX OF SHOTGUN SHELLS
92
21

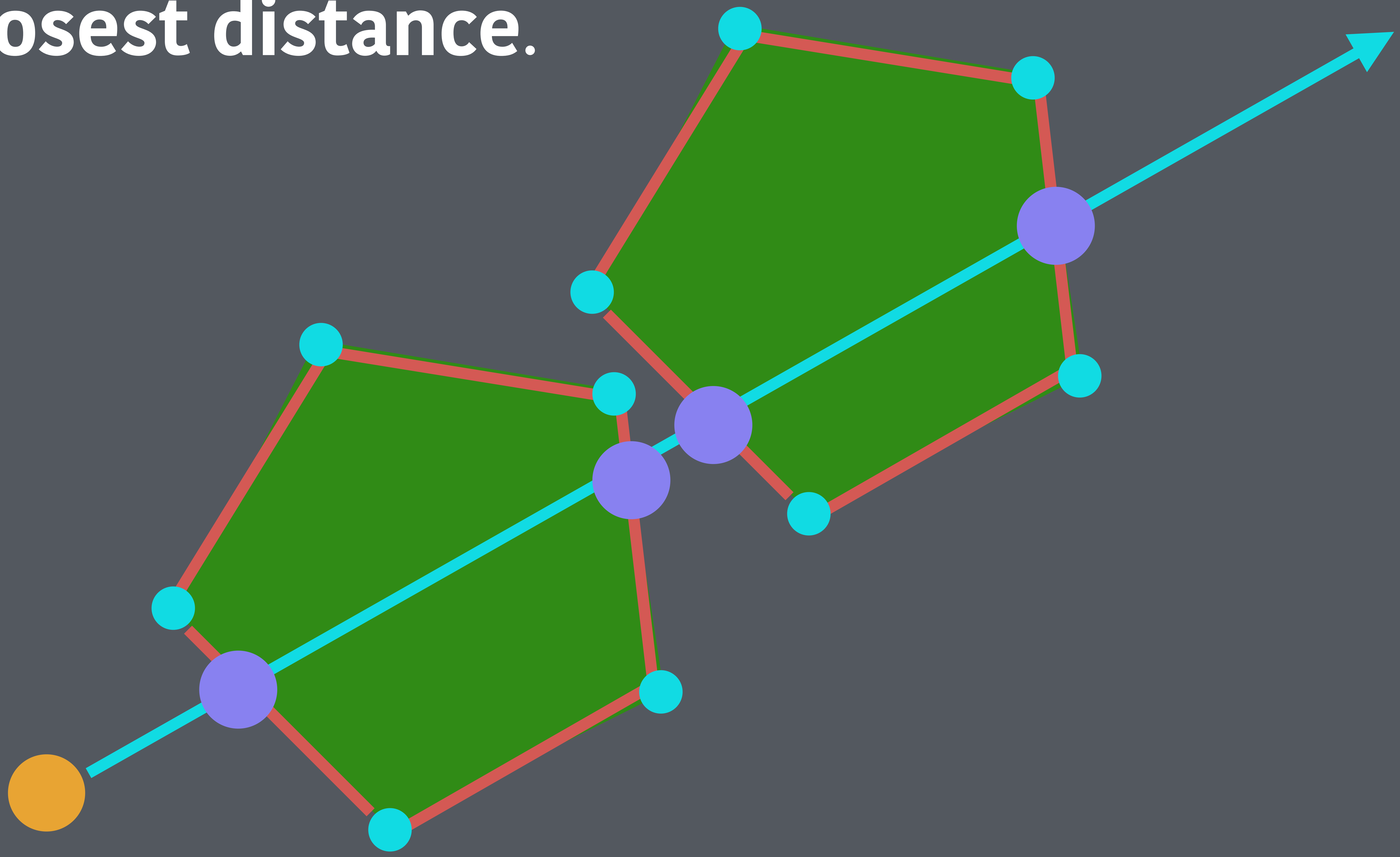
Ray/Polygon **intersection test**.



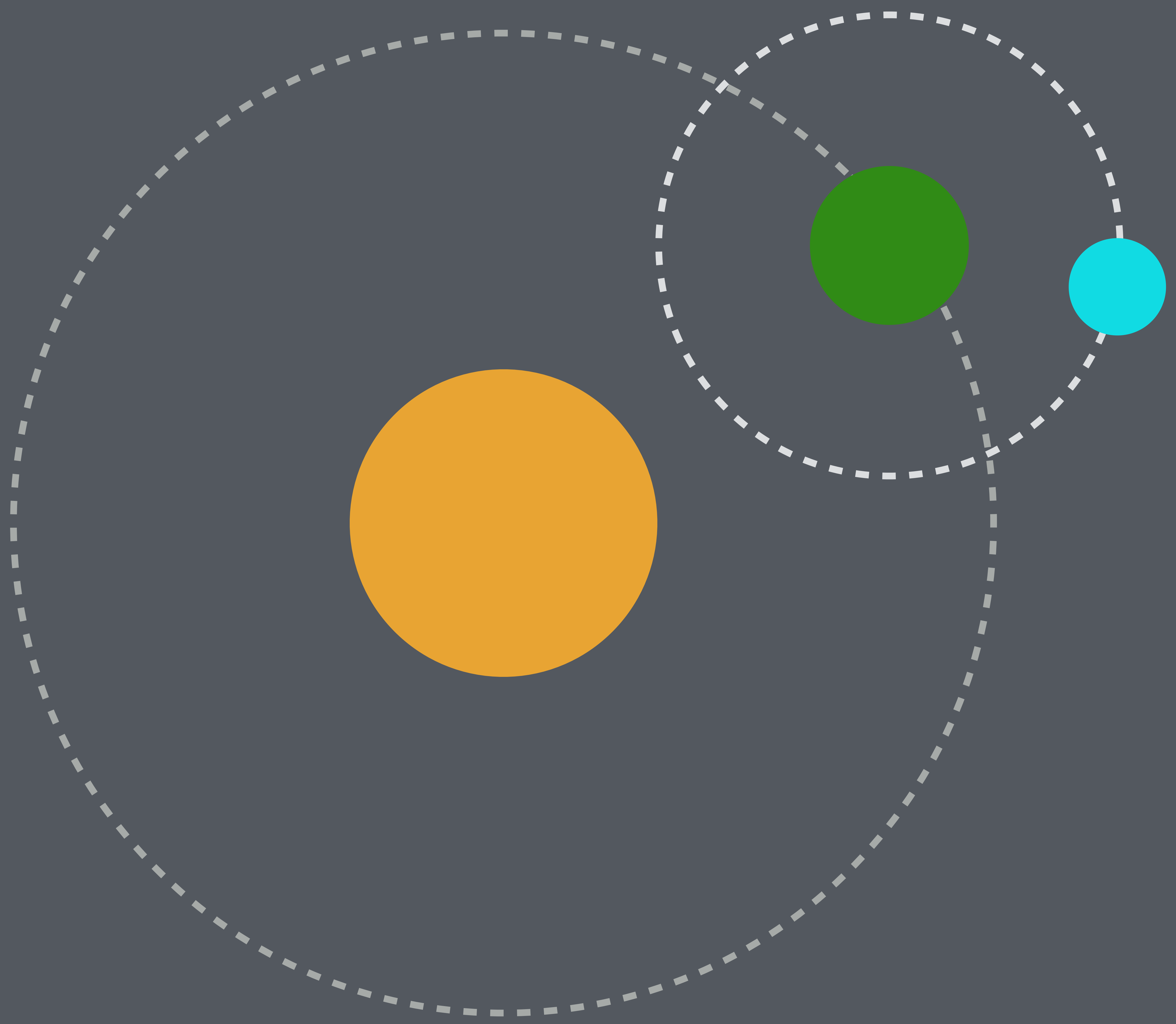


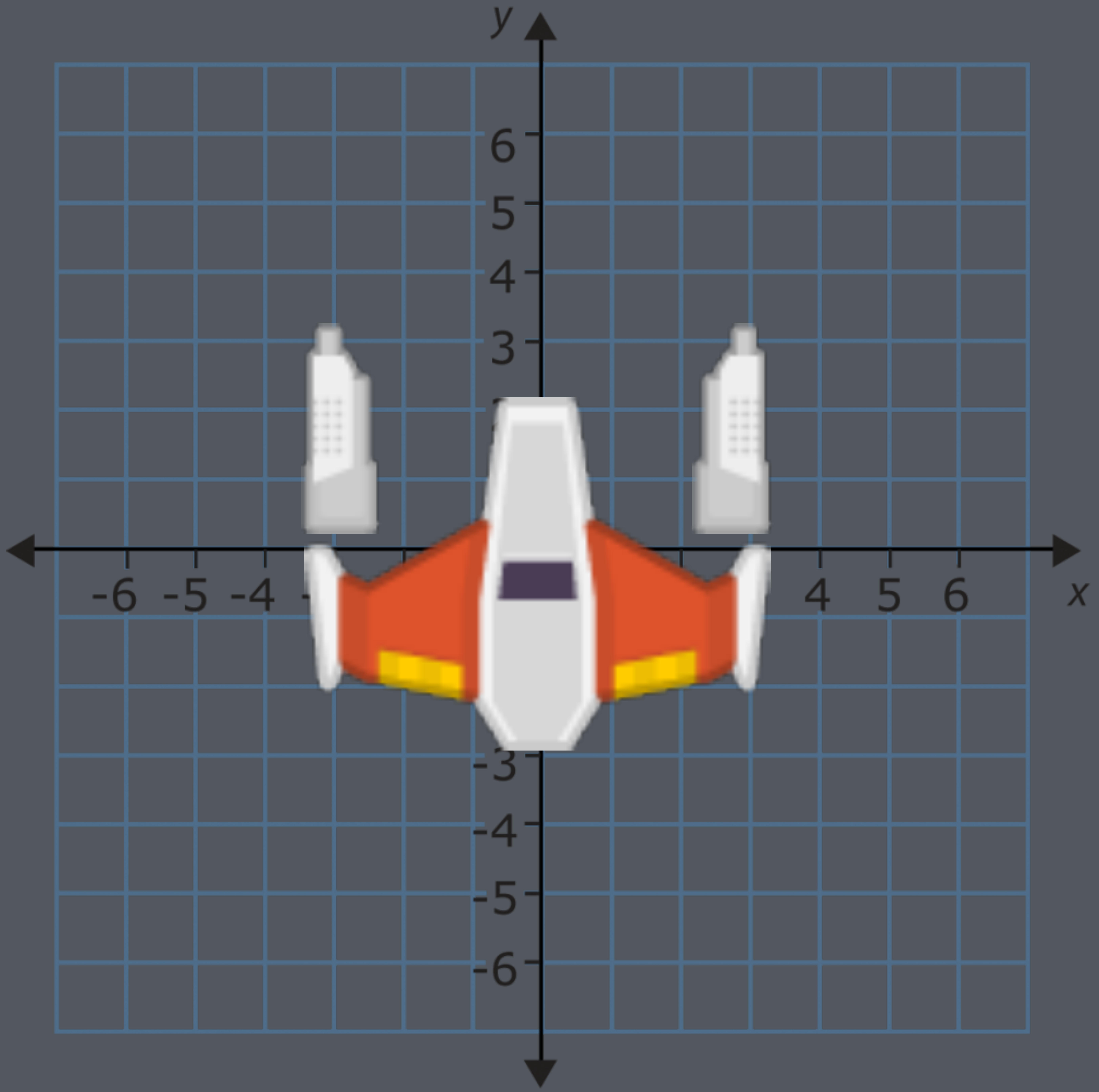
```cpp
bool raySegmentIntersect(const Vector &rayOrigin, const Vector &rayDirection, const
Vector &linePt1, const Vector &linePt2, float &dist)
{
    Vector seg1 = linePt1;
    Vector segD;
    segD.x = linePt2.x - seg1.x;
    segD.y = linePt2.y - seg1.y;

    float raySlope = rayDirection.y / rayDirection.x;
    float n  = ((seg1.x - rayOrigin.x)*raySlope + (rayOrigin.y - seg1.y)) / (segD.y -
segD.x*raySlope);

    if (n < 0 || n > 1)
        return false;

    float m = (seg1.x + segD.x * n - rayOrigin.x) / rayDirection.x;
    if (m < 0)
        return false;

    dist = m;
    return true;
}
```

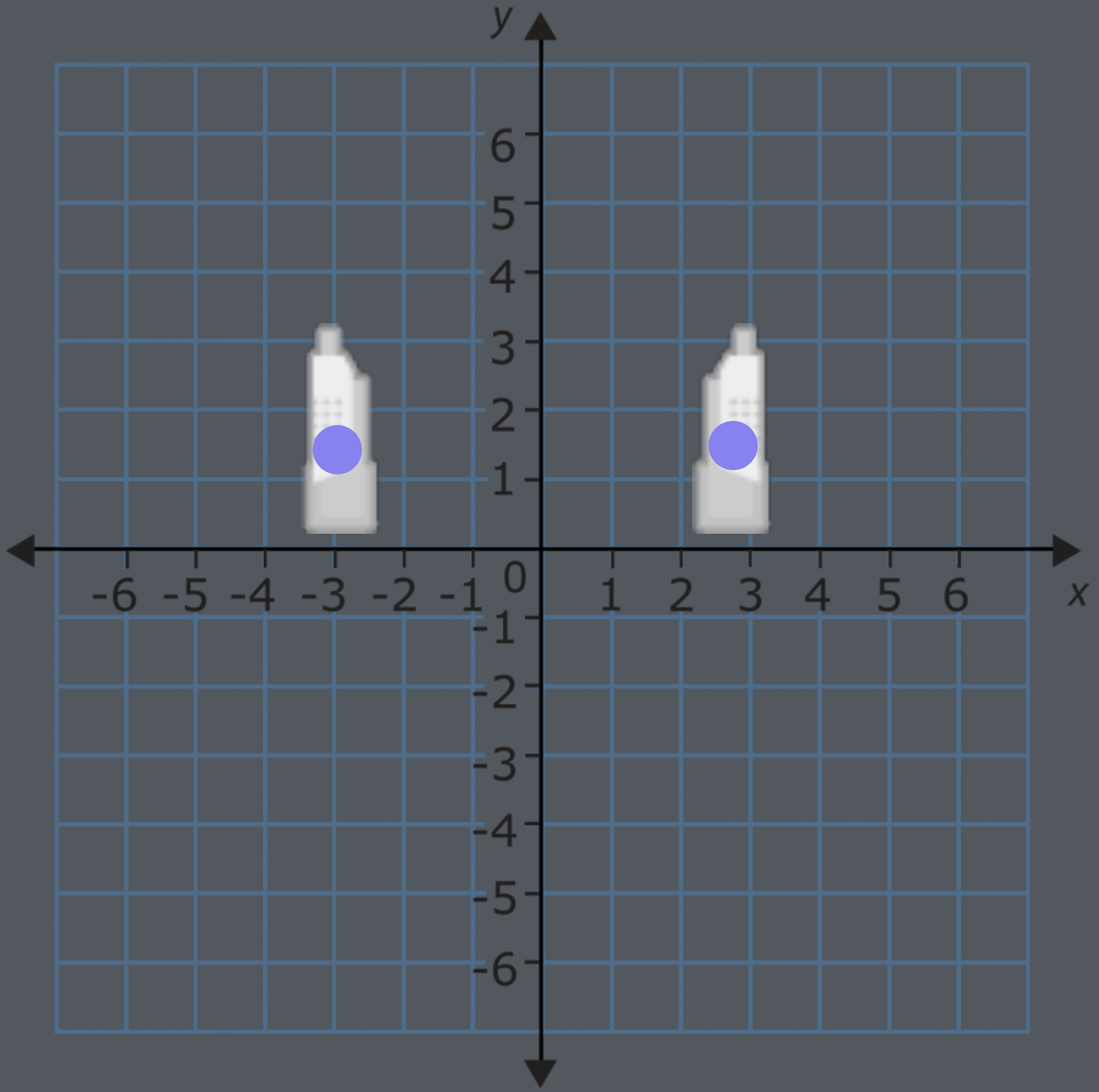
Use the **closest distance**.



Entity hierarchies




y
x
6
5
4
3
-3
-4
-5
-6
-6
-5
-4
4
5
6



y
x
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-6 -5 -4 -3 -2 -1
1 2 3 4 5 6

```cpp
class Entity {
  public:

      Entity();
      void Render();

      Entity *parentEntity;

 };


 Entity::Entity() : parentEntity(NULL) {

 }

 void Entity::Draw() {

    // instead of matrix.identity();
    if(parentEntity) {
        matrix = parentEntity->matrix;
    }

 }
```

Assigning a **parent entity.**