Game Physics

GPR350, Fall 2019
Daniel S. Buckstein

Intro to Collisions
Weeks 5 – 6

License

Intro to Collisions

- Physics: continuous vs. discrete integration
  - (still relevant)
- Intro to collision detection
- Intro to collision response

Intro to Collisions

- Introduction to ***rigid body kinematics***
- Rigid body: non-deformable primitive with physical properties
- Opposite is soft-body dynamics (e.g. cloth)

- We'll explore two main uses for rigid bodies: ***collision detection and response***

Intro to Collisions

- “Coarse intersection testing” is a fast method of checking whether objects’ *bounding volumes* intersect with each other

- …what’s a *bounding volume*???

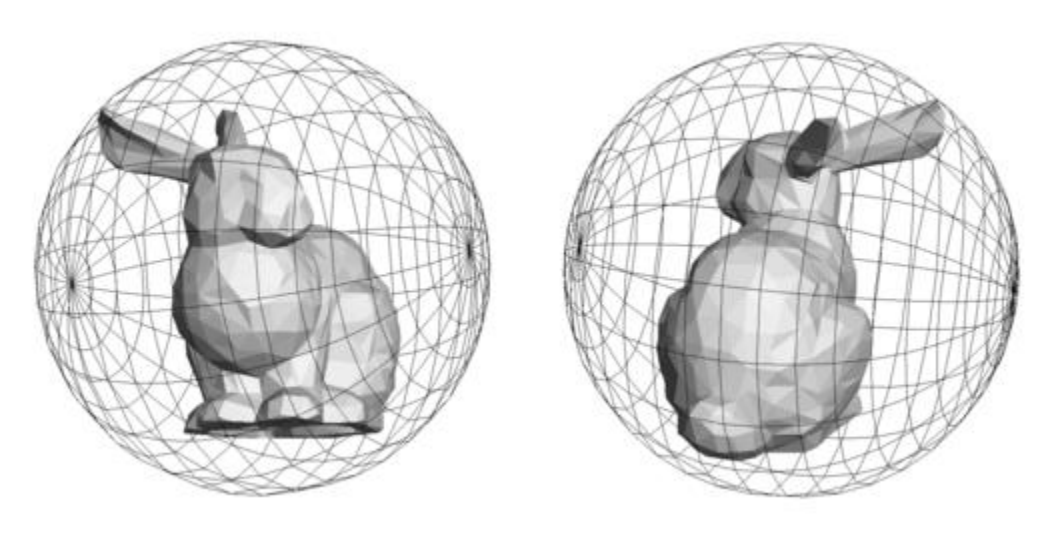
Intro to Collisions

- ***Bounding volume***: the space that the object is contained within
- Same principles in 2D and 3D
- Use bounding volumes for *coarse* intersection testing: rough estimates whether the object is colliding with another

- What is the fastest intersection test???

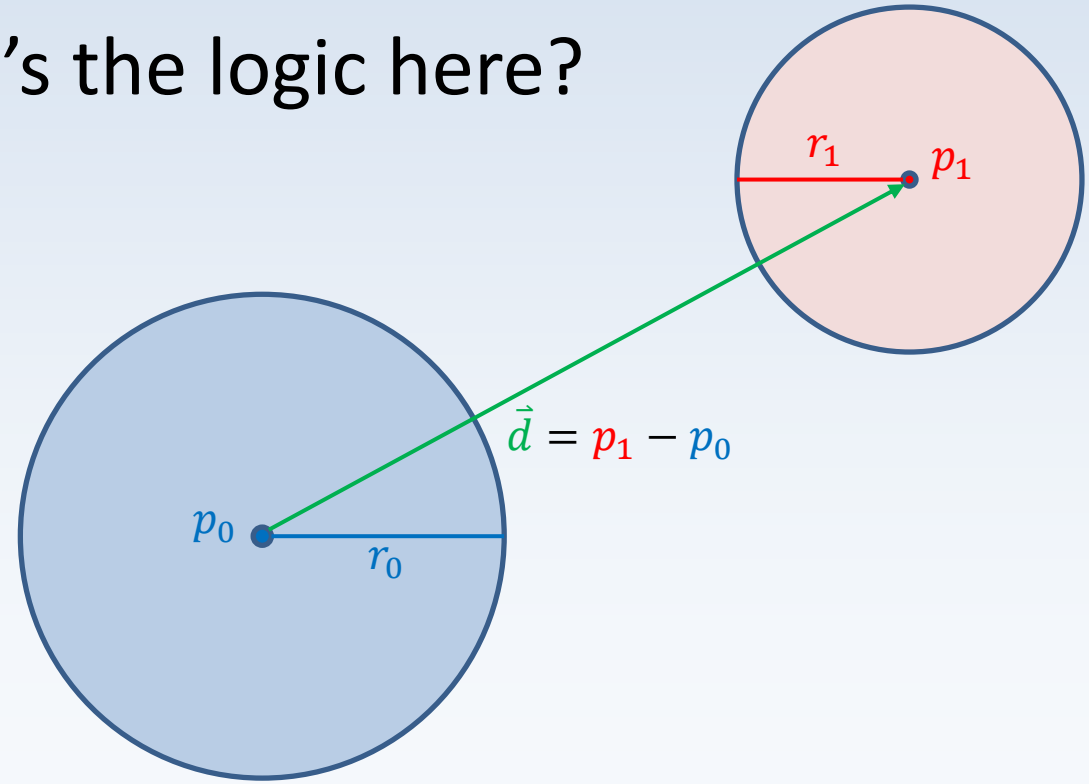
Intro to Collisions

- Bounding volumes & intersection testing:
- The fastest intersection test???
- Circle-circle or sphere-sphere



Intro to Collisions

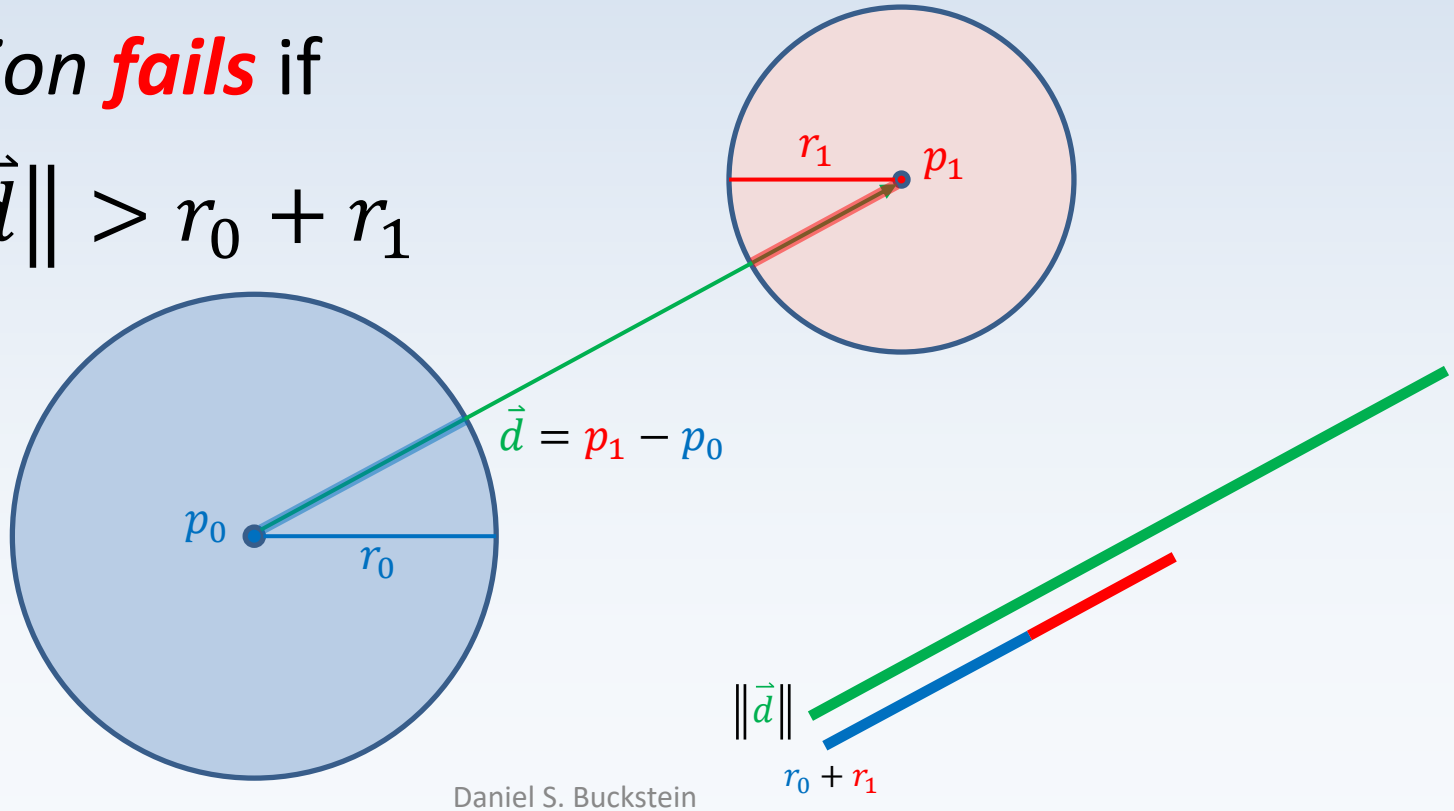
- Bounding volumes & intersection testing:
- ***Circle-circle*** or ***sphere-sphere***
- What's the logic here?



Intro to Collisions

- Bounding volumes & intersection testing:
- ***Circle-circle*** or ***sphere-sphere***
- *Collision* ***fails*** if

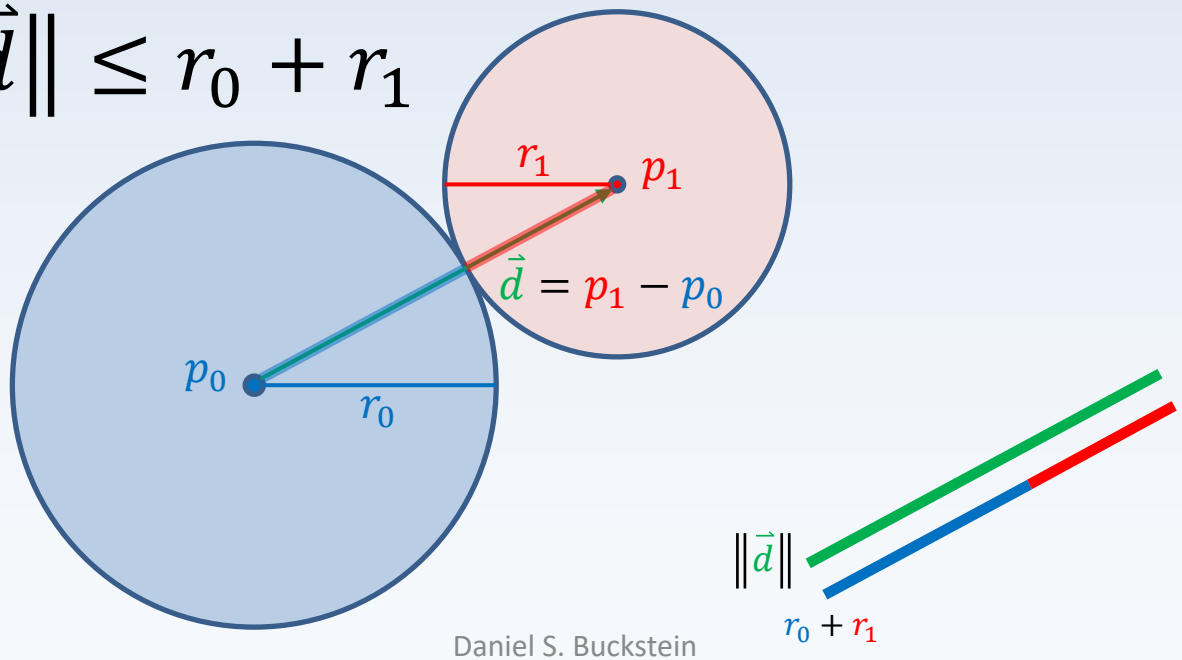
$$\|\vec{d}\| > r_0 + r_1$$



Intro to Collisions

- Bounding volumes & intersection testing:
- ***Circle-circle*** or ***sphere-sphere***
- ***"Collision"*** ***passes*** if

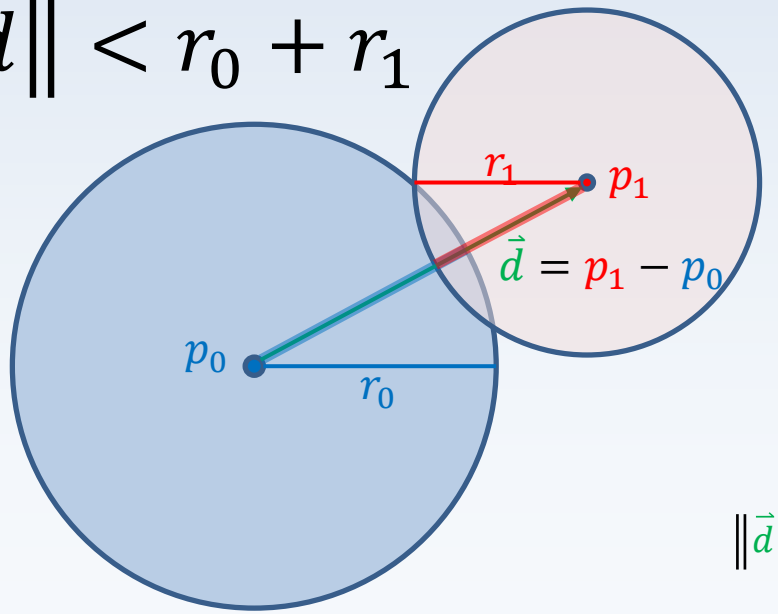
$$\|\vec{d}\| \leq r_0 + r_1$$

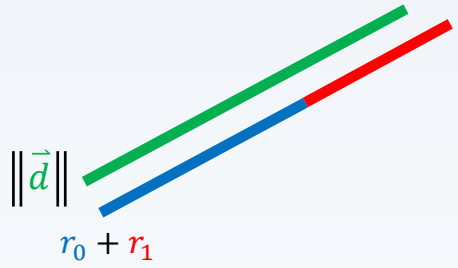


Intro to Collisions

- Bounding volumes & intersection testing:
- ***Circle-circle*** or ***sphere-sphere***
- ***"Intersection" passes*** if

$$\|\vec{d}\| < r_0 + r_1$$





Intro to Collisions

- Bounding volumes & intersection testing:
- ***Circle-circle*** or ***sphere-sphere***
- ***Pro tip***: square roots are the *root* of all evil!!!

hurhur

- Optimize by *squaring both sides*

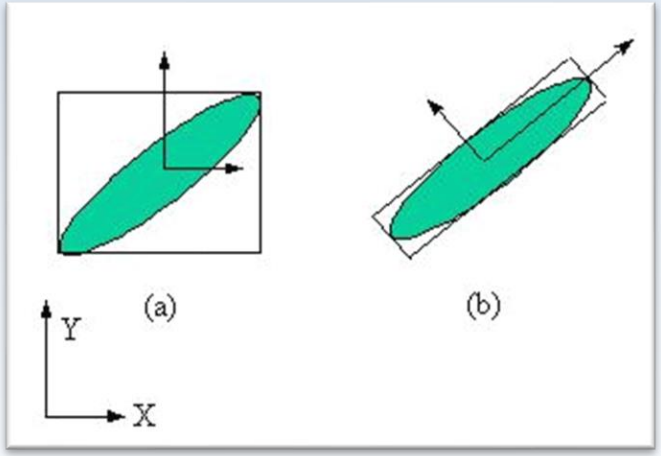
$$\left\|\vec{d}\right\|^2 < (r_0 + r_1)^2$$

- Then we can use math hax:

$$\mathbf{dot}\left(\vec{d}, \vec{d}\right) < (r_0 + r_1)^2$$

Intro to Collisions

- Bounding volumes & intersection testing:
- The next best bet: ***bounding boxes***
- Most common form in game engines…
- ***AABB/AABV***



- Alternatively…
- ***OBB/OBV***

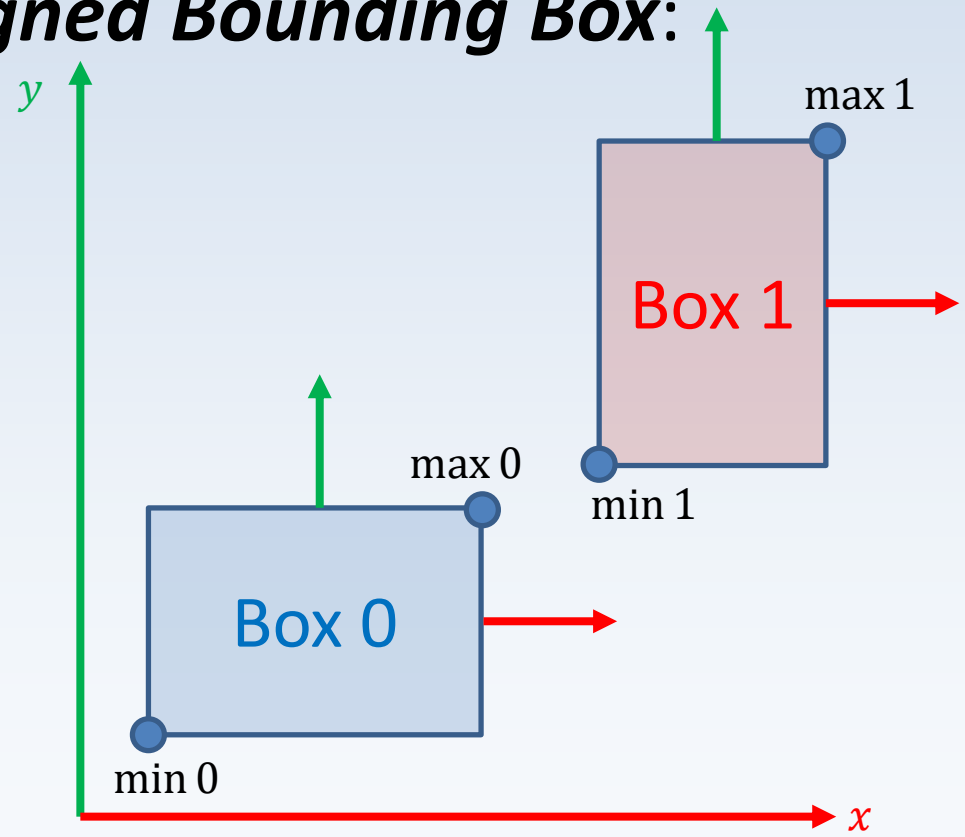
Intro to Collisions

- Bounding volumes & intersection testing:
- The algorithm for box-to-box intersection is also simple:
- Essentially we are projecting the geometry onto the *geometric normals**
- Test for 1-dimensional overlaps of minimum and maximum values in geometry!

*re. normals: we are not in the domain of graphics right now… but it's a similar concept

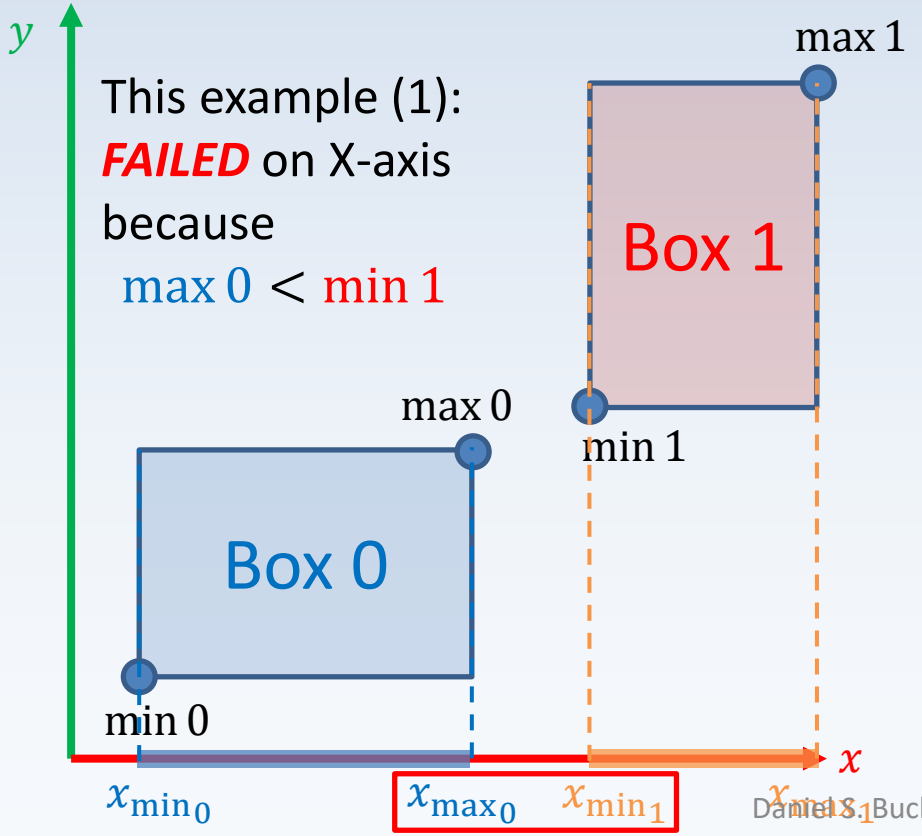
Intro to Collisions

- Bounding volumes & intersection testing:
- ***Axis-Aligned Bounding Box***:



Intro to Collisions

- Bounding volumes & intersection testing:
- Box-to-box (AABB):



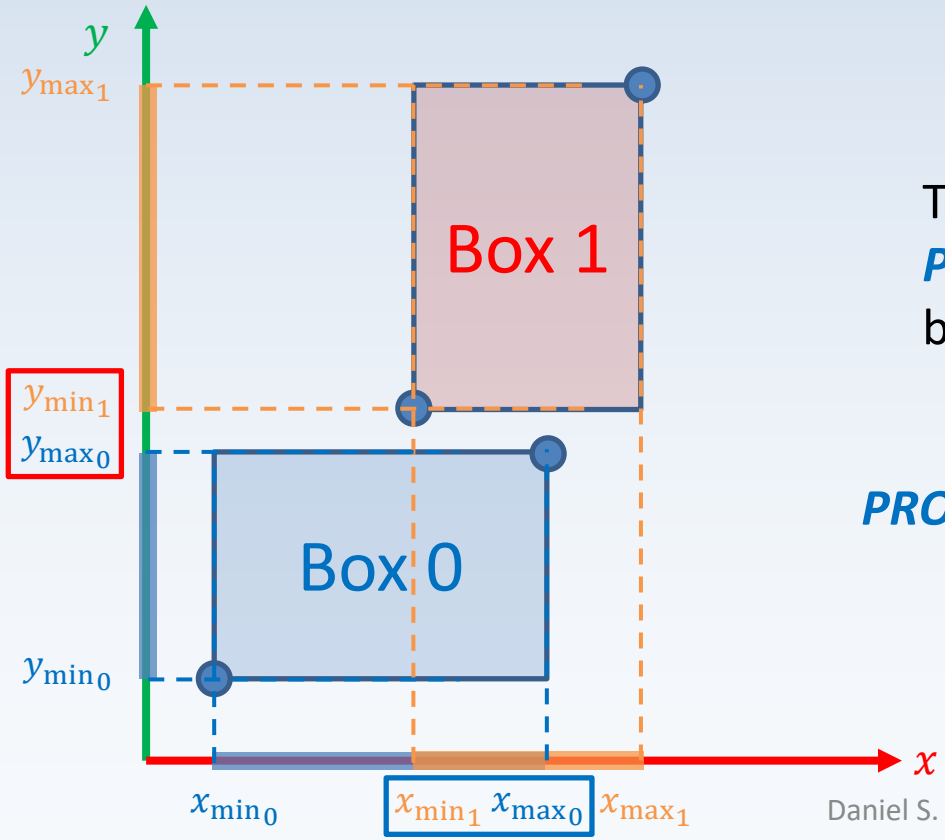
The algorithm* ***PASSES*** if this statement is ***TRUE*** for ***ALL DIMENSIONS***:

$$\max 0 \geq \min 1$$
$$\boldsymbol{AND}$$
$$\max 1 \geq \min 0$$

*Collision detection magic formula

Intro to Collisions

- Bounding volumes & intersection testing:
- Box-to-box (AABB):



PASS if ***TRUE*** (***ALL DIMENSIONS)***:

$$\max 0 \geq \min 1 \quad \textit{AND} \quad \max 1 \geq \min 0$$

This example (2):

Potential pass

because pass on X-axis

$$\max 0 \geq \min 1$$

PROCEED TO TEST Y-AXIS

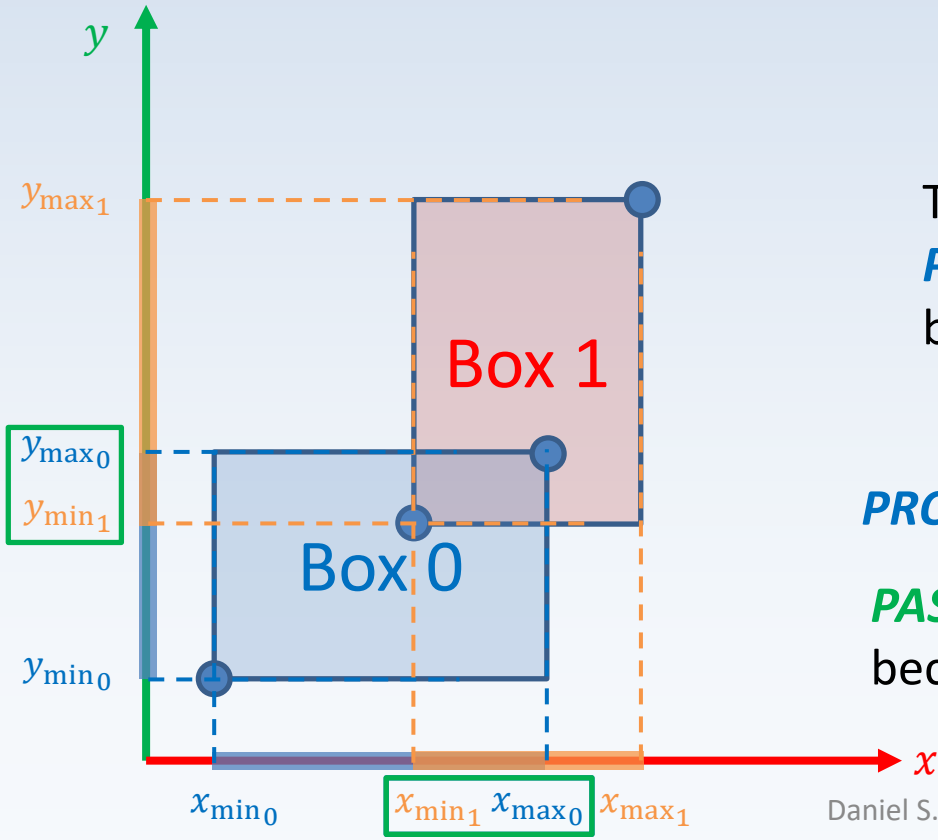
FAILED on Y-axis

because

$$\max 0 < \min 1$$

Intro to Collisions

- Bounding volumes & intersection testing:
- Box-to-box (AABB):



PASS if ***TRUE*** (***ALL DIMENSIONS)***:

$$\max 0 \geq \min 1$$
AND
$$\max 1 \geq \min 0$$

This example (3):

Potential pass

because pass on X-axis

$$\max 0 \geq \min 1$$

PROCEED TO TEST Y-AXIS

PASS!!! ☺

because pass on both axes!

$$\max 0 \geq \min 1$$

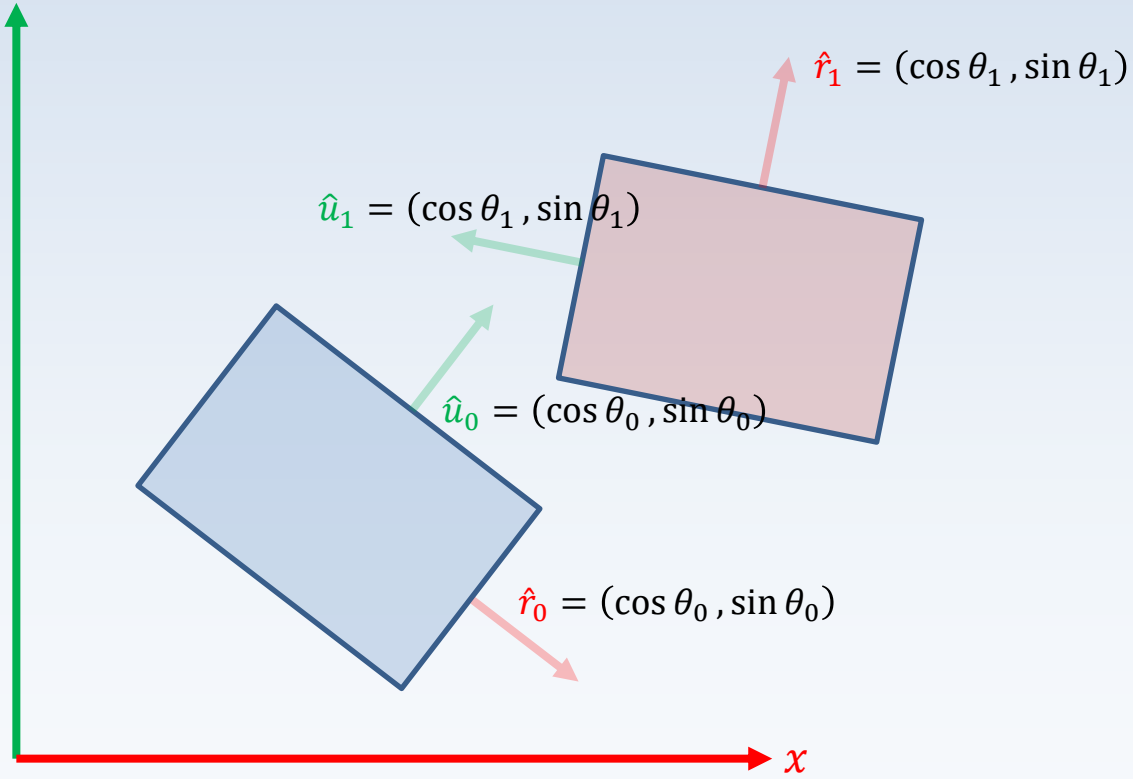
Intro to Collisions

- Complexity introduced with rotating boxes:
- ***Object Bounding Box***:

Normal calculation:

$\hat{r} = (+\cos\theta\,, +\sin\theta)$

$\hat{u} = (-\sin\theta\,, +\cos\theta)$



Intro to Collisions

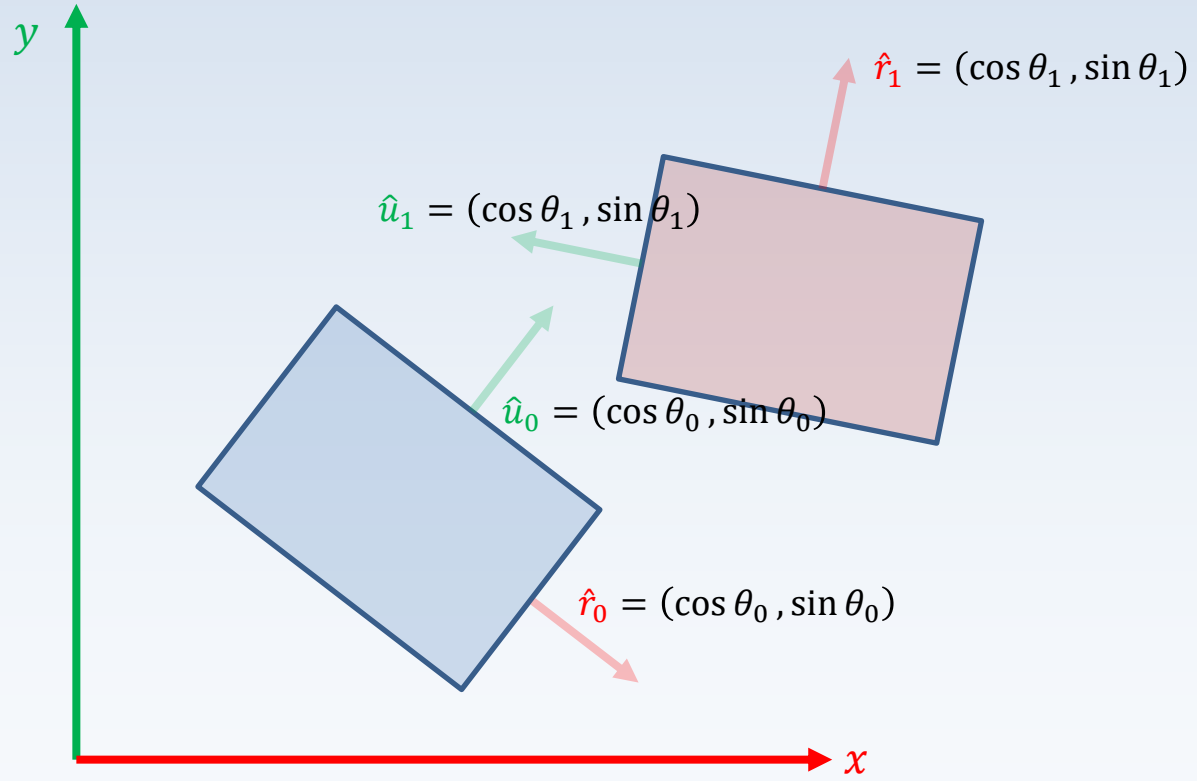
- ***Object Bounding Box***: algorithm makes use of a special (familiar) formula…

Projection formula:
"*a* onto *b*"

$$\text{proj}_{\vec{b}}\vec{a} = \frac{\vec{a}\cdot\vec{b}}{\|\vec{b}\|^2}\vec{b}$$

$$\text{proj}_{\hat{b}}\vec{a} = \frac{\vec{a}\cdot\hat{\boldsymbol{b}}}{\|\hat{\boldsymbol{b}}\|^2}\hat{\boldsymbol{b}}$$

$$\mathbf{proj}_{\hat{\boldsymbol{b}}}\vec{\boldsymbol{a}} = (\vec{\boldsymbol{a}}\cdot\hat{\boldsymbol{b}})\hat{\boldsymbol{b}}$$



Intro to Collisions

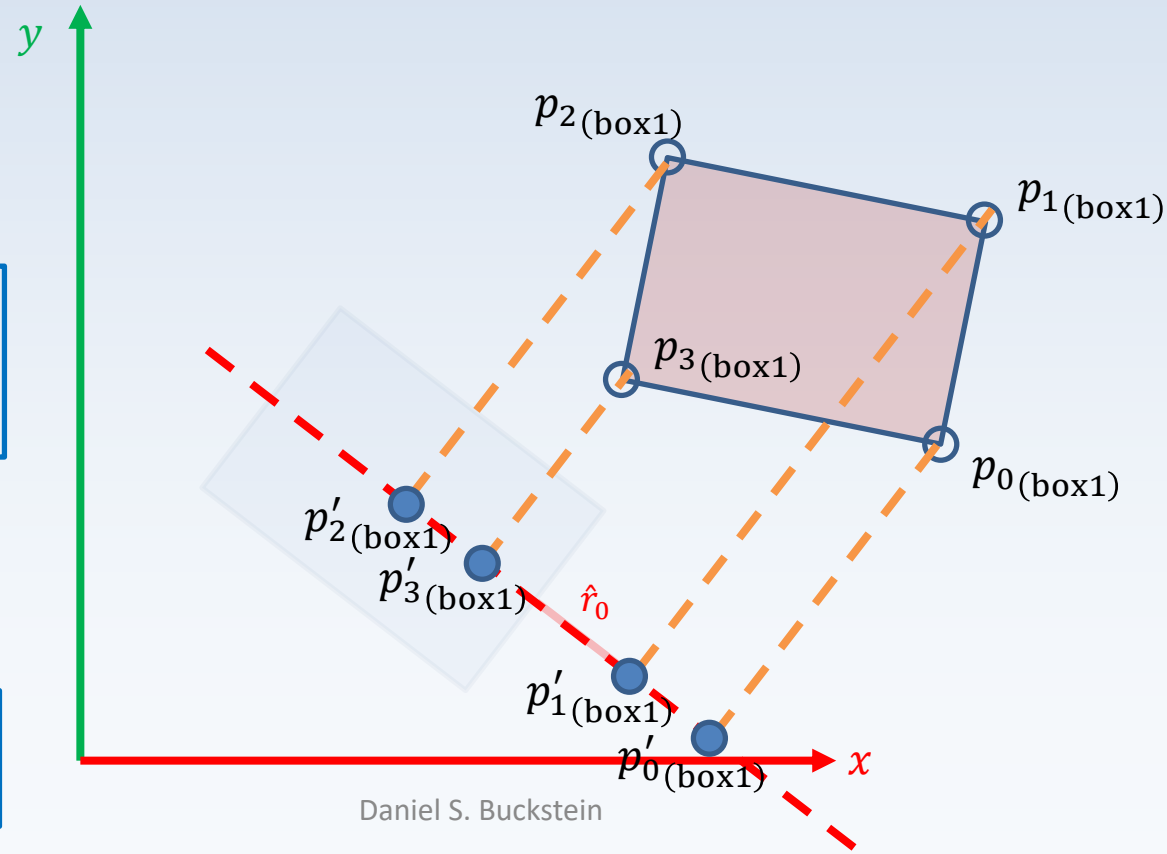
- ***Object Bounding Box***: Collision test is ***2 steps for each edge normal*** until failure occurs:

Step 1:
project all vertices onto normal

$$\boldsymbol{p'} = \mathbf{proj}_{\hat{\boldsymbol{n}}}\boldsymbol{p} = (\boldsymbol{p} \cdot \hat{\boldsymbol{n}})\hat{\boldsymbol{n}}$$

(repurposed formula from general formula below)

$$\mathbf{proj}_{\hat{\boldsymbol{b}}}\vec{\boldsymbol{a}} = (\vec{\boldsymbol{a}} \cdot \hat{\boldsymbol{b}})\hat{\boldsymbol{b}}$$

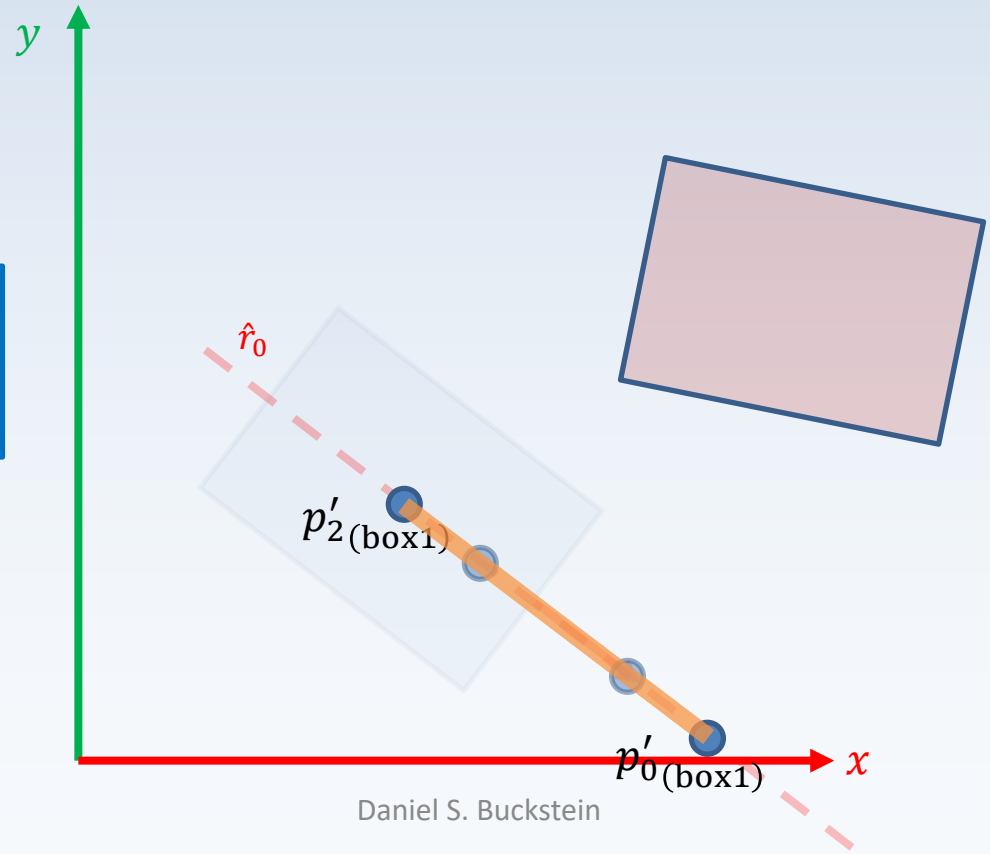


Intro to Collisions

- ***Object Bounding Box***: *During* projection step, keep track of ***min and max values***

Step 1:
project all vertices
onto normal

$$\boldsymbol{p'} = \mathbf{proj}_{\hat{\boldsymbol{n}}}\boldsymbol{p} = (\boldsymbol{p} \cdot \hat{\boldsymbol{n}})\hat{\boldsymbol{n}}$$

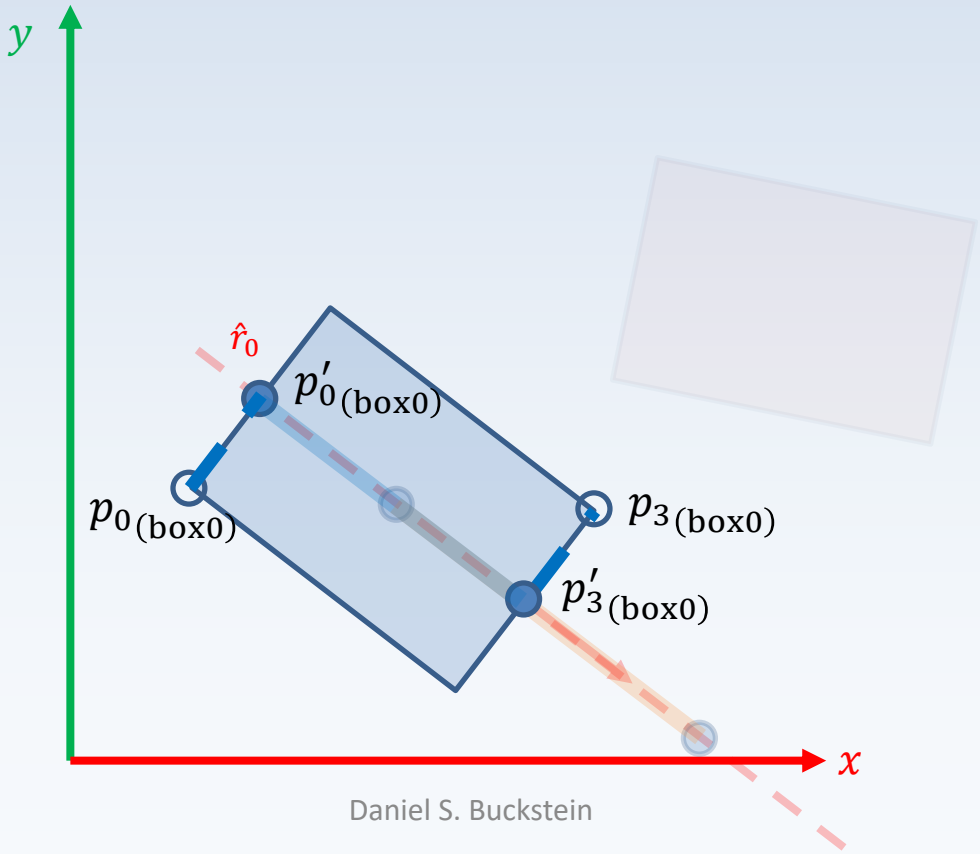


Intro to Collisions

- ***Object Bounding Box***: Need to project vertices from both shapes…

Step 1:
project all vertices onto normal

$$\boldsymbol{p'} = \mathbf{proj}_{\hat{\boldsymbol{n}}}\boldsymbol{p} = (\boldsymbol{p} \cdot \hat{\boldsymbol{n}})\hat{\boldsymbol{n}}$$



Intro to Collisions

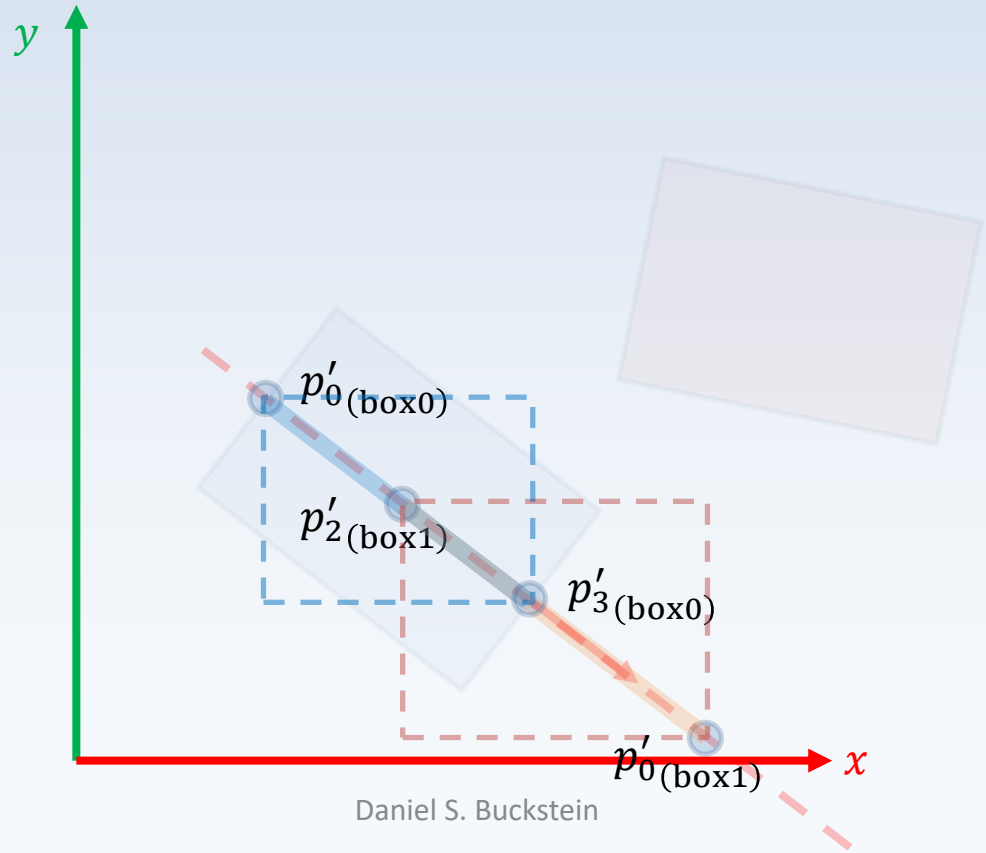
- ***Object Bounding Box***: We are left with two sets of ***minimums and maximums***…

Step 2:

Any guesses???

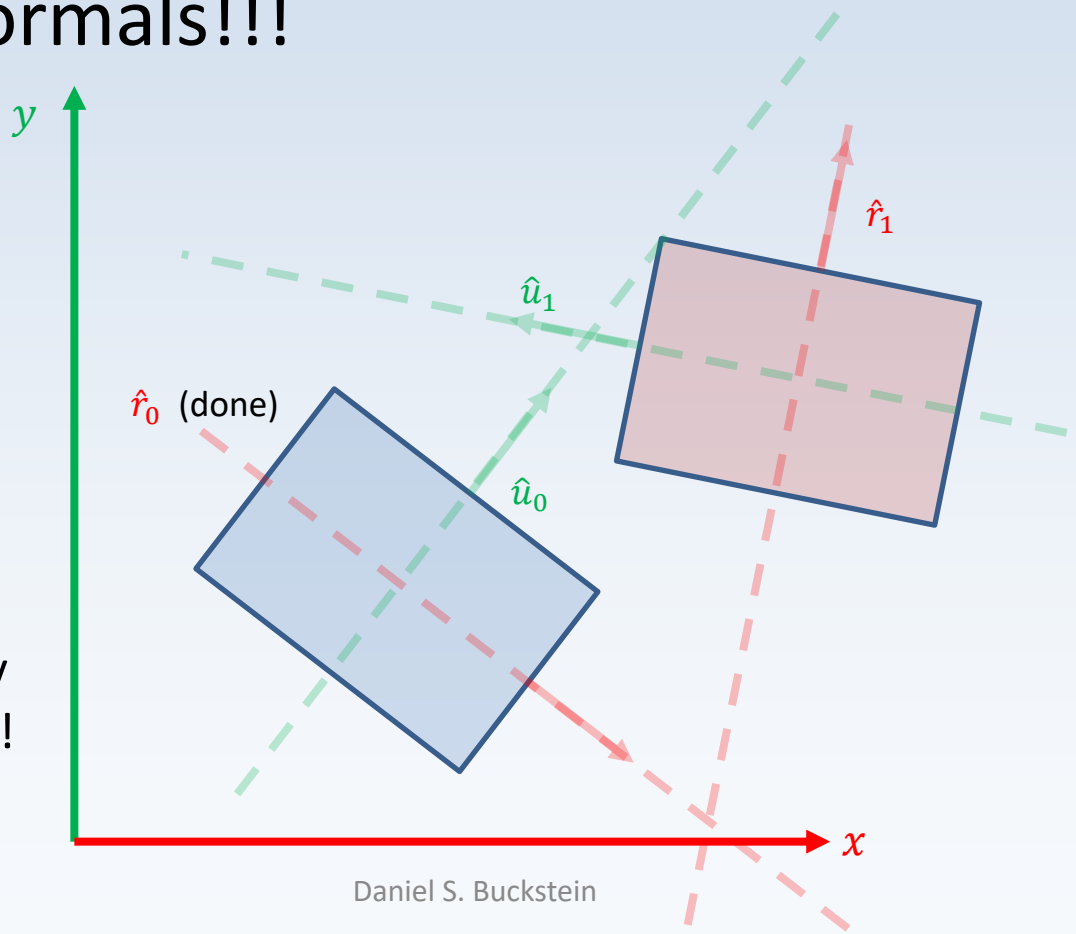
Perhaps AABB test?

☺



Intro to Collisions

- ***Object Bounding Box***: repeat 2 steps for each of the normals!!!



Algorithm ***FAILS*** if any part of any test fails!!!

Intro to Collisions

- ***AABB data structure*** (same idea in 2D & 3D):
- Need to know min and max values:

```
struct AABB
{
    vec3 position;  // where do we get this info??? hmmmm...
    vec3 minCorner, maxCorner;
};
```

- Update corners (in case object ***moved***):

```
void updateCorners(AABB *aabb);
```

Collision Detection & Response

- ***Collision response***:
- Our focus for now: ***AABB (axis-aligned, same algorithm for 2D and 3D)***

- Detection is easy…
- …response is ***generally*** not easy
- Let's modify our algorithm and create a data structure to respond to collisions

Collision Detection & Response

- ***Collision response***:
- Assume we have a raw AABB collision test (and a structure for AABB)

```
bool TestAABB (AABB box0, AABB box1);
```

- Returns pass/fail only!

Collision Detection & Response

- ***Collision response***:
- Handling requires a bit more than *bool*…

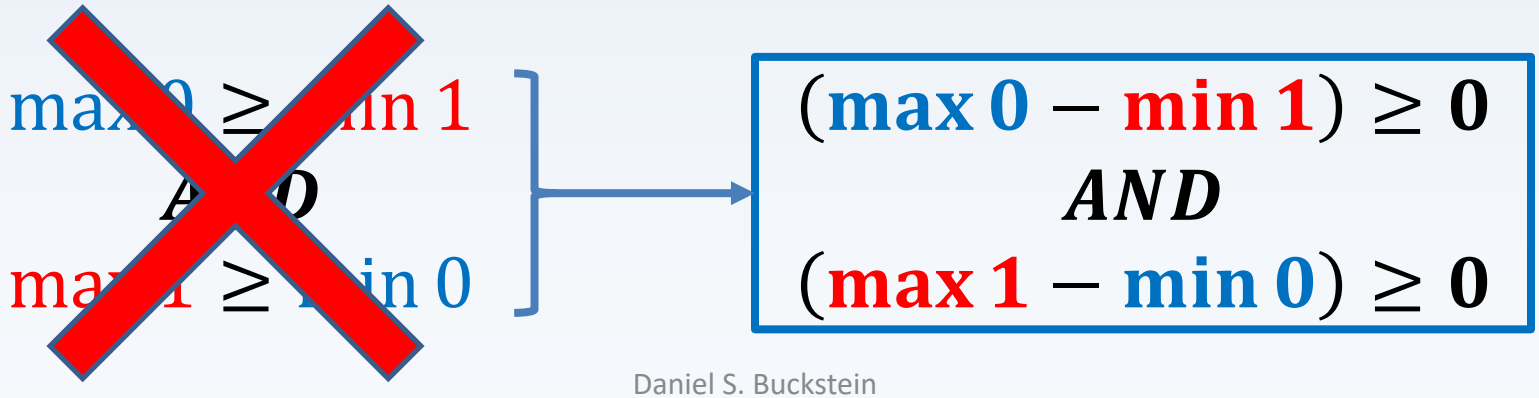
```
struct Collision {
    bool status;    // store result in struct
};

Collision TestAABB (AABB box0, AABB box1);
```

(returns a collision descriptor!!!)

Collision Detection & Response

- ***Collision response***:
- The critical reason why AABB test works:
- ***The overlap on each axis.***
- ***We need to be storing this***
- Revisit the pass condition (for all axes):



Collision Detection & Response

- ***Collision response***:
- Need our collision descriptor to be a little more complex…
- Test function should store the overlap as well

```
struct Collision {
    bool status;      // result of test
    vec2 overlap;     // actual overlap
};
```

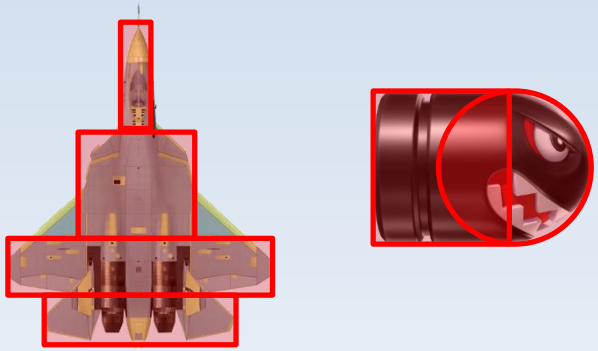
Collision Detection & Response

- Problem with bounding boxes: airplane vs. bullet

- Look at all the empty space!
- Collision will not be very precise… good for a *preliminary collision check*
- …but what would we do for a precise collision

Collision Detection & Response

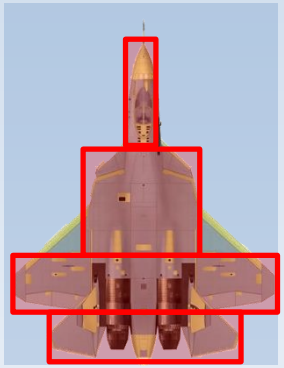
- Complex objects: **multiple bounding volumes**

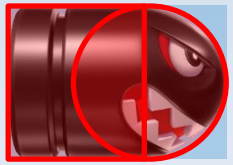


- All OBBs still follow object's transform
- But they each have their own sub-transform relative to the whole object!

Collision Detection & Response

- Complex objects: **multiple bounding volumes**

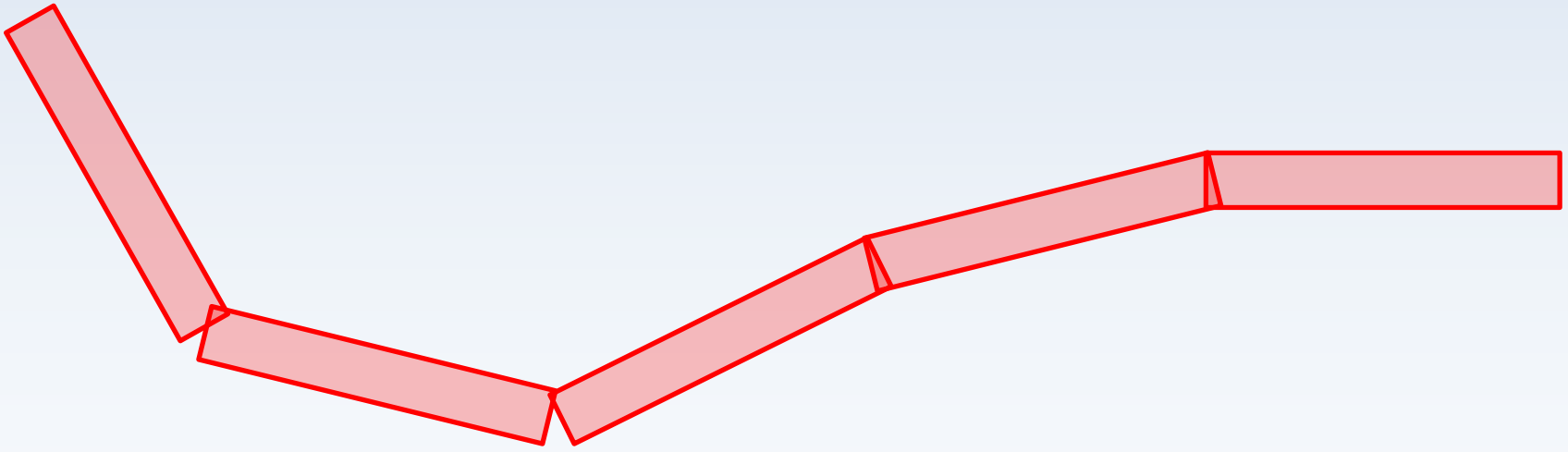




- Increase precision by adding sub-bounding volumes and super-bounding volumes
- ***Collision layers***

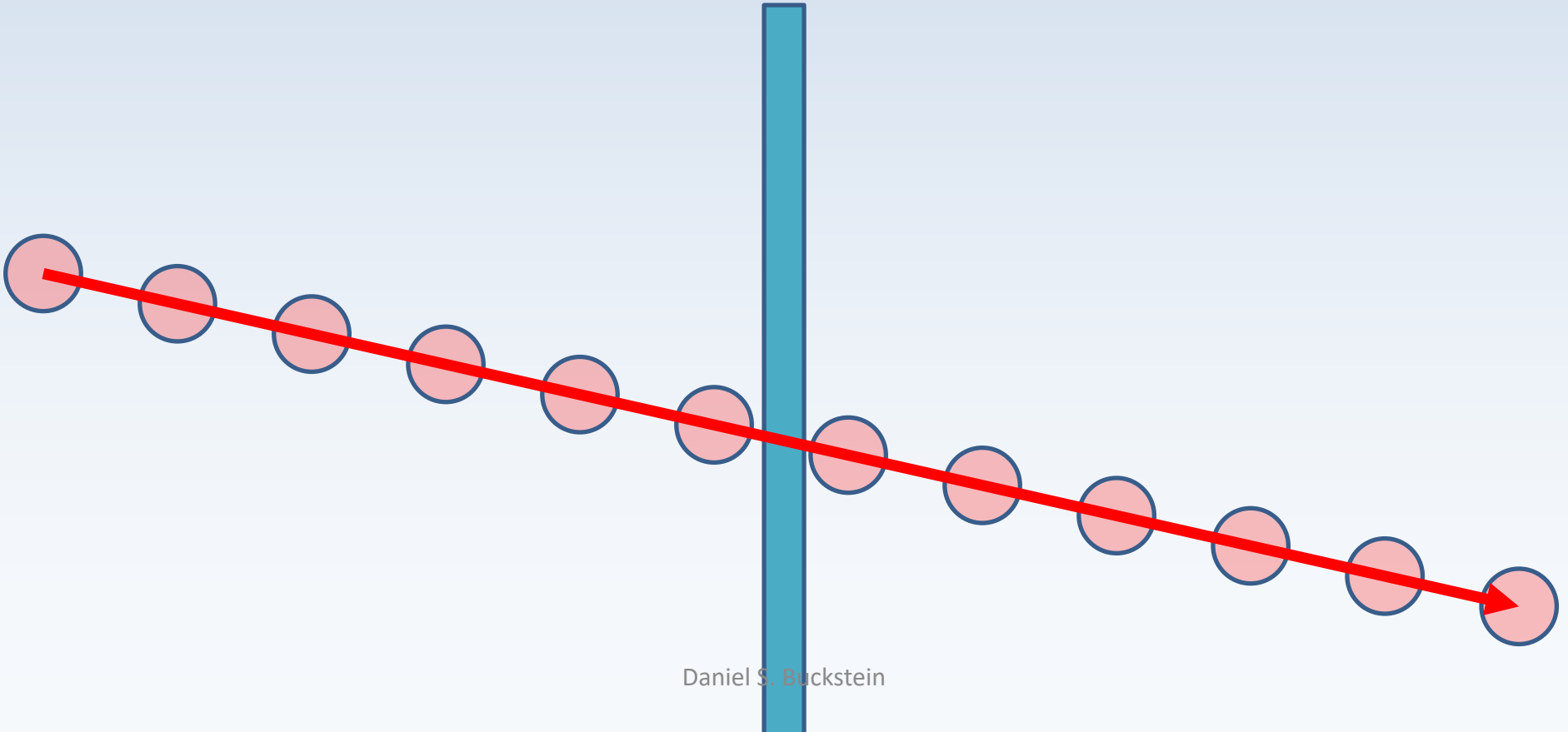
Collision Detection & Response

- How do games handle complex maps with lots of angled walls n’ stuff???
- Just position static OBBs everywhere!



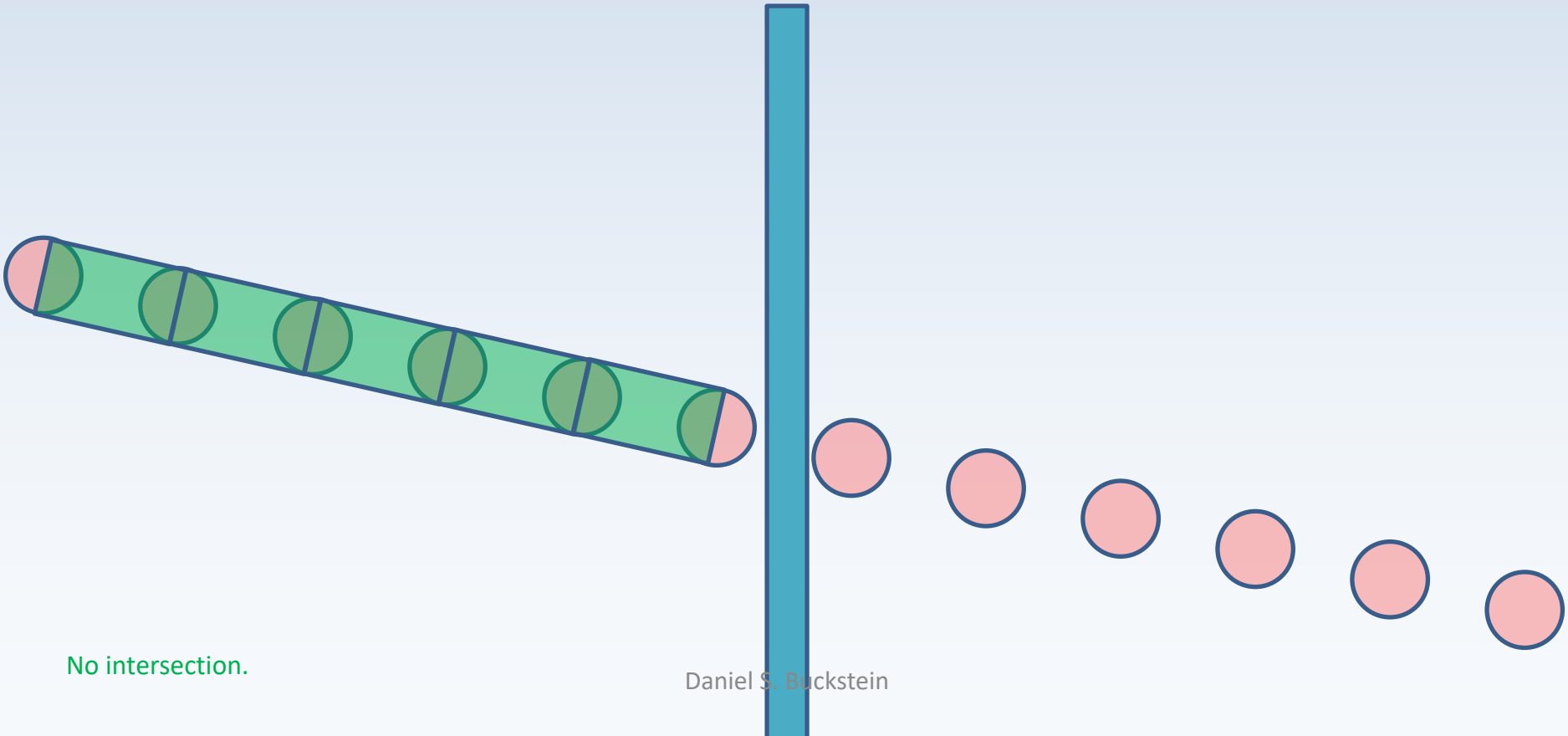
Collision Detection & Response

- What is this event called???



Collision Detection & Response

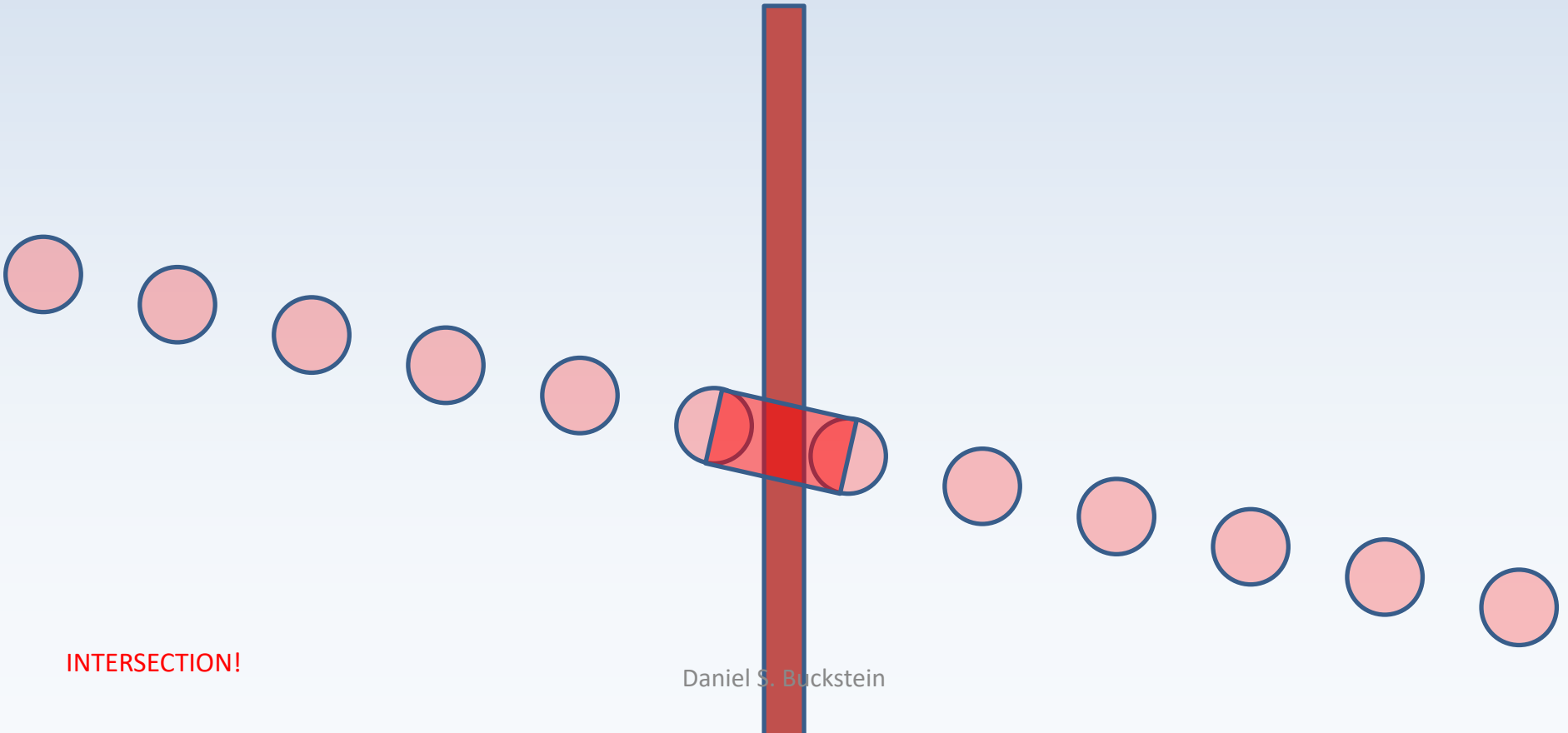
- How might we solve it???



No intersection.

Collision Detection & Response

- How might we solve it???



INTERSECTION!

The end.

- Questions? Comments? Concerns?