Physics Systems

OR

Physics in Applications

- Physics is used to simulate how objects move ‘naturally’, using our real world’s physics as a frame of reference.
- Used in games to create and explore unique worlds and spaces otherwise not possible in our real world.
- Used in applications to run simulations to predict real world outcomes.

Writing your own Physics

- Custom game physics be used to create a different world and feeling that could lead to fun.
  - i.e. A world with multiple gravitational centers??
- Simulating physics from the real world will give a more relatable and realistic effect to your game.

Physics System Components

What's in a typical physics system?

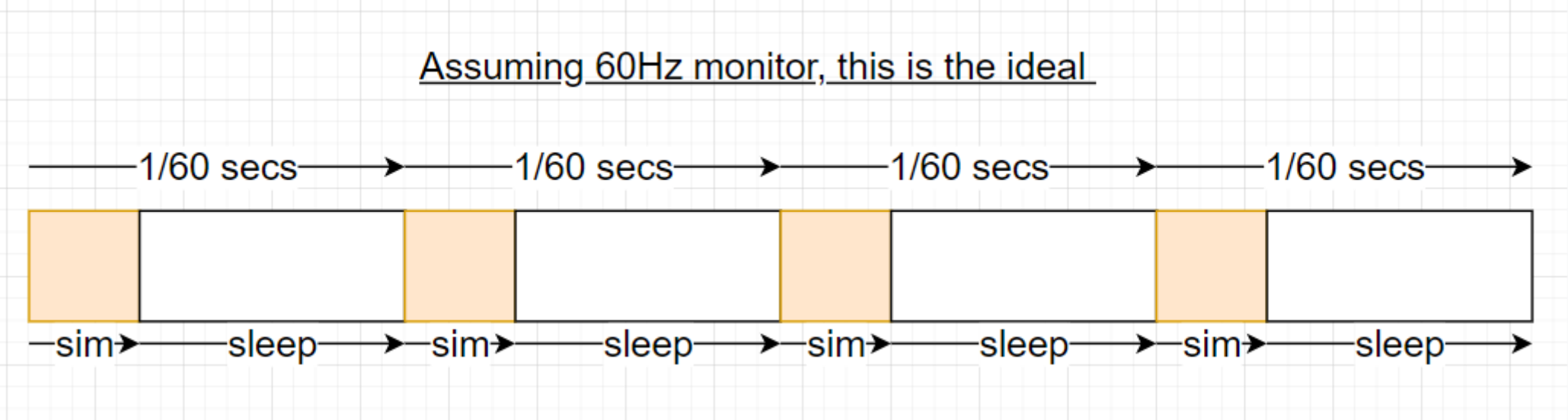
Integration

- Integrations are ways to update your physics bodies over time.
- Euler's Integration
  1. acceleration = Force/mass
  2. velocity += acceleration * deltaTime
  3. position += velocity * deltaTime
- Verlet Integration
- RK5 Integration

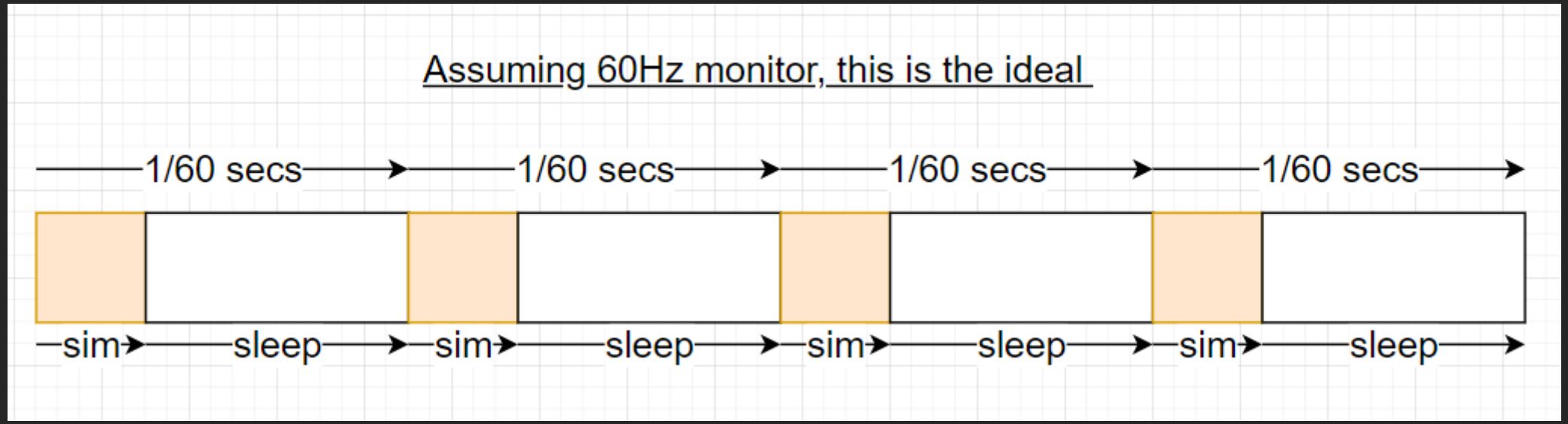
Time Step

- The time interval between two frames is represented by delta time.
- If the game lags, delta time will be bigger.
- Ideally used in every part of the game that is time sensitive (animation, gameplay, physics).
- Typical graphics implementation will try to have delta time match some factor of the monitor's refresh rate (to prevent screen tearing)

Time Step


Assuming 60Hz monitor, this is the ideal
1/60 secs
1/60 secs
1/60 secs
1/60 secs
sim
sleep
sim
sleep
sim
sleep
sim
sleep

Time Step



In case this is not clear yet

Your entire application relies on code related to graphics code at some point.


DT
LIGHTYEAR
DT EVERYWHERE

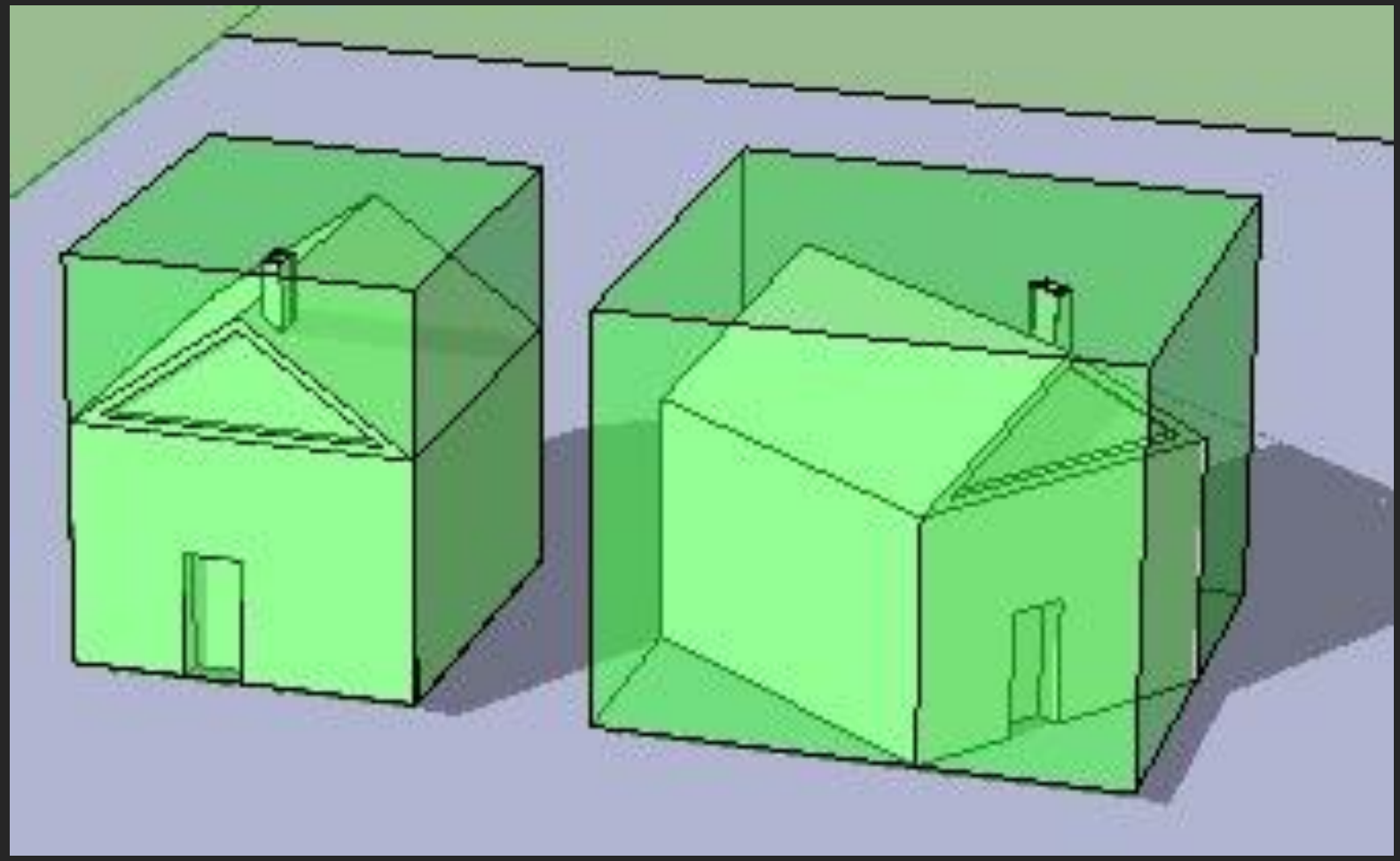
Time Step issues

- If the game lags, delta time will be bigger.
- If delta time is bigger, you might need to resolve more collisions → more lag → more deltaTime → GG ☹
- What about VSYNC, where your app matches its frame rate with the monitor's refresh rate?

Time Step possible solutions

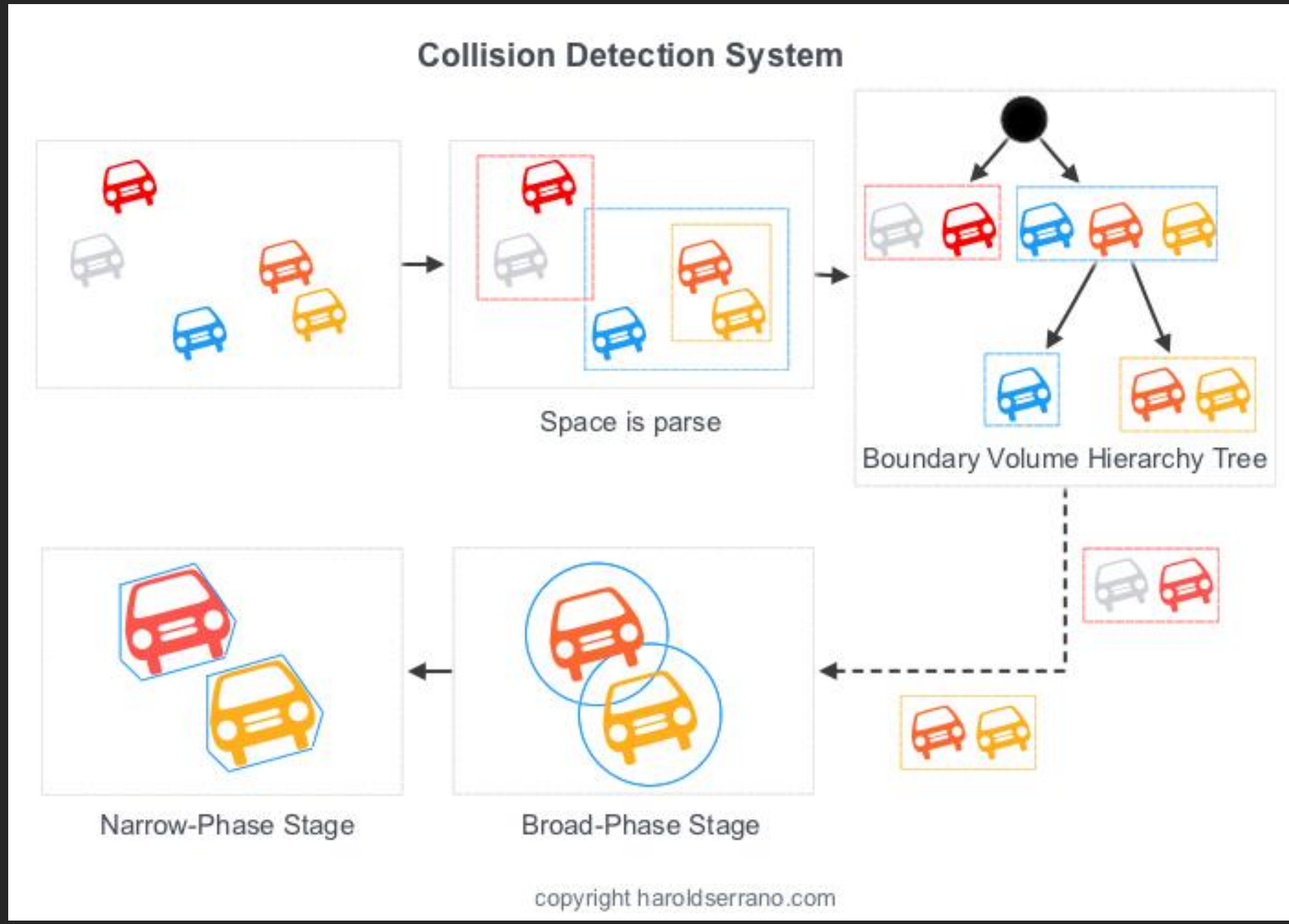
- There are several ideas to handle this, and which to pick depends on your app.
  - Fixed delta time.
  - Variable delta time based on previous frame
    - if you missed 60Hz mark, you drop to 30Hz.
  - Capping the delta time.
  - Have separate delta time for simulations.
- https://gafferongames.com/post/fix_your_timestep/

Collision Detection



Collision Detection

- Collision Detection is split into two phases.
- Narrow-Phase – How to check if pairs of individual bodies are colliding with each other?
  - Raycasting, Bounding Volumes, etc.
- Broad-Phase – How to avoid redundant Narrow Phase computation by eliminating checks for pairs that are far from each other?
  - Spatial Partition, Sweep and Prune, etc.


Collision Detection System
Space is parse
Boundary Volume Hierarchy Tree
Narrow-Phase Stage
Broad-Phase Stage
copyright haroldserrano.com

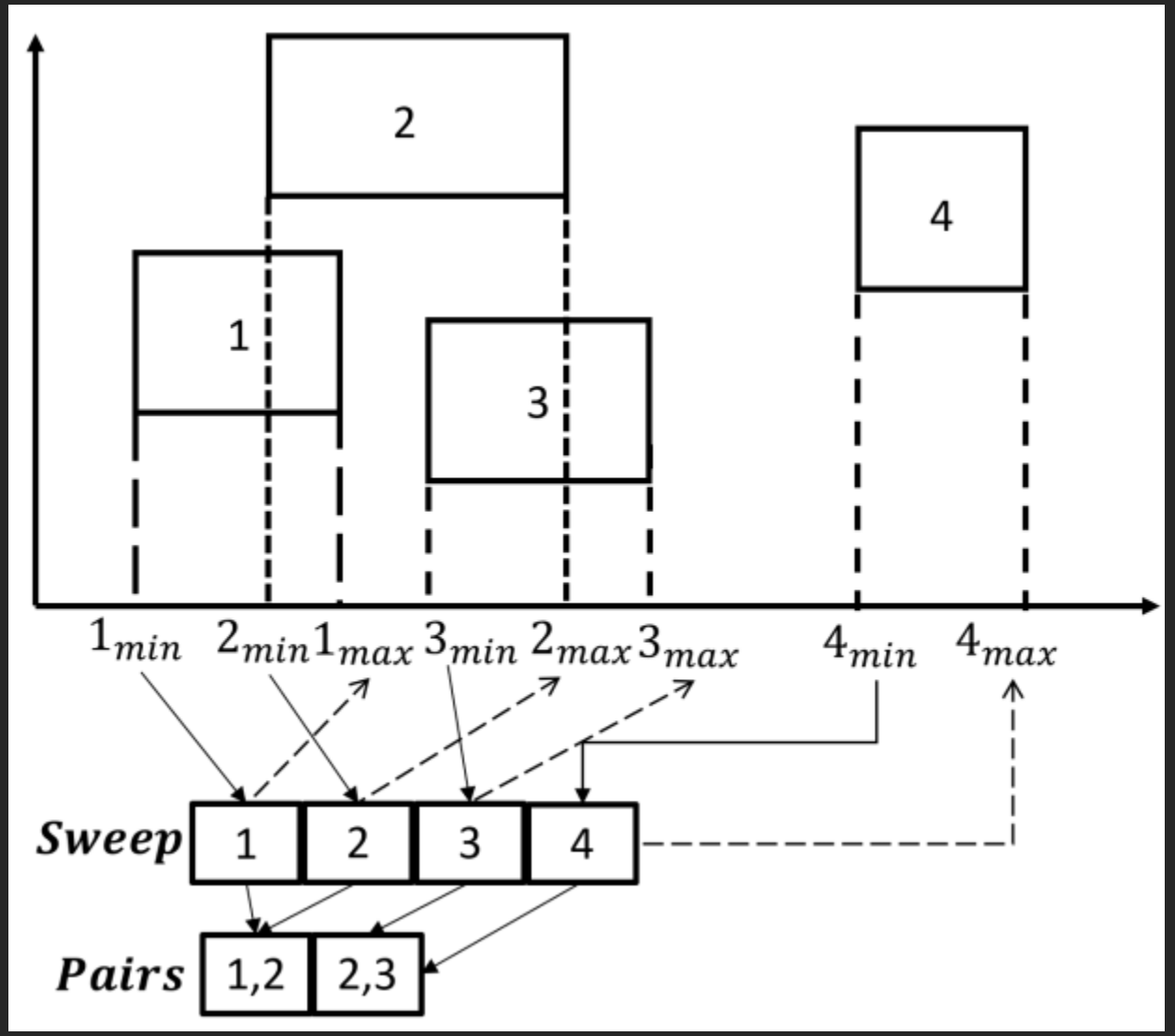
Broad Phase

- Usually considered when there are so many collision checks in one frame that we need to reduce them.
  - Don't assume! Profile first!
- The general idea:
  - Divide space into subspaces
  - Figure out what objects are within which subspace
  - Do collision only for objects within a subspace (or with nearby subspaces)

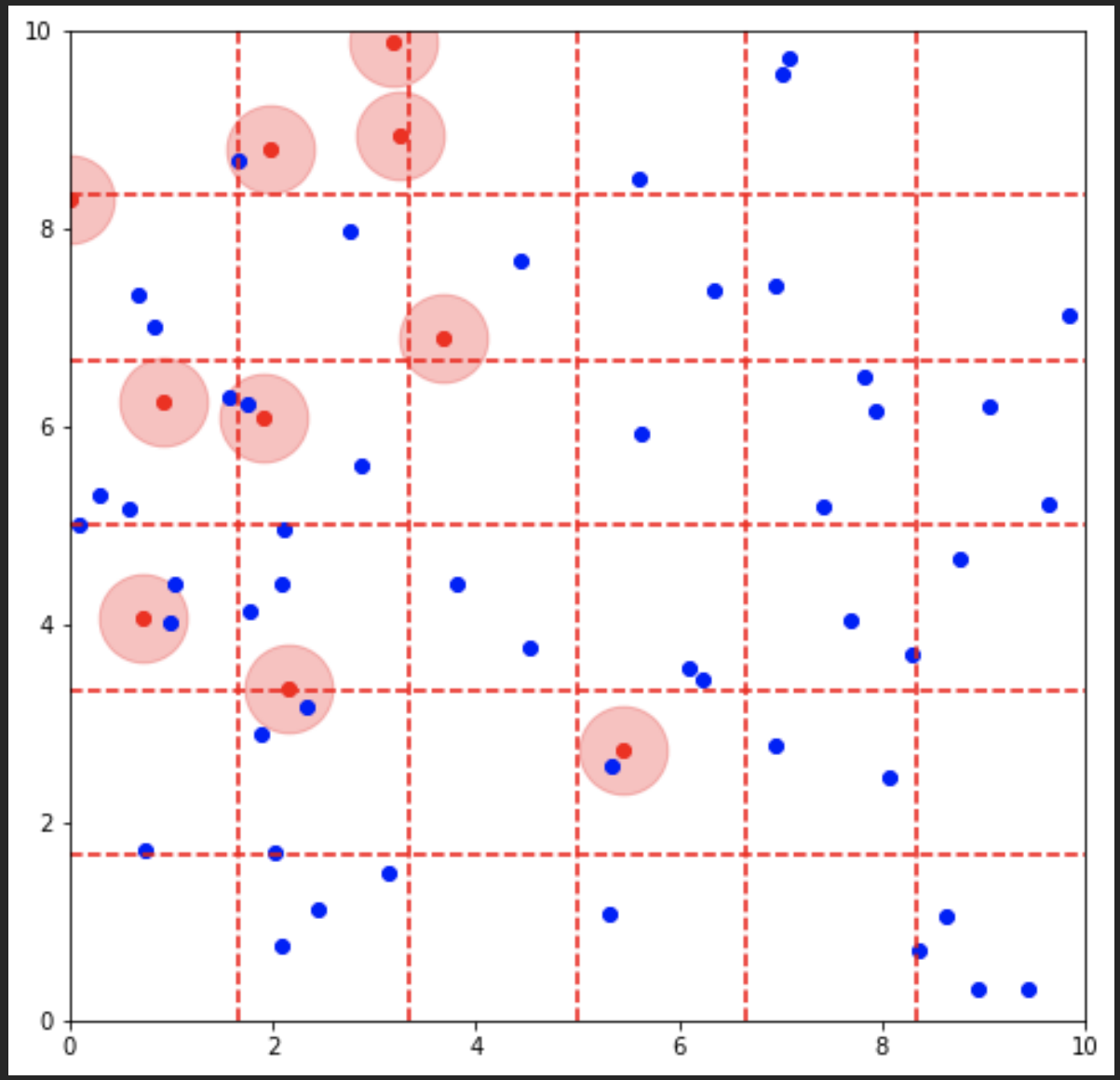
Dividing space into subspaces

- We use spatial partitioning techniques
- There are many techniques available, and many of them are used not just in physics programming, but also other disciplines in computer science!
- All of them have their pros and cons.

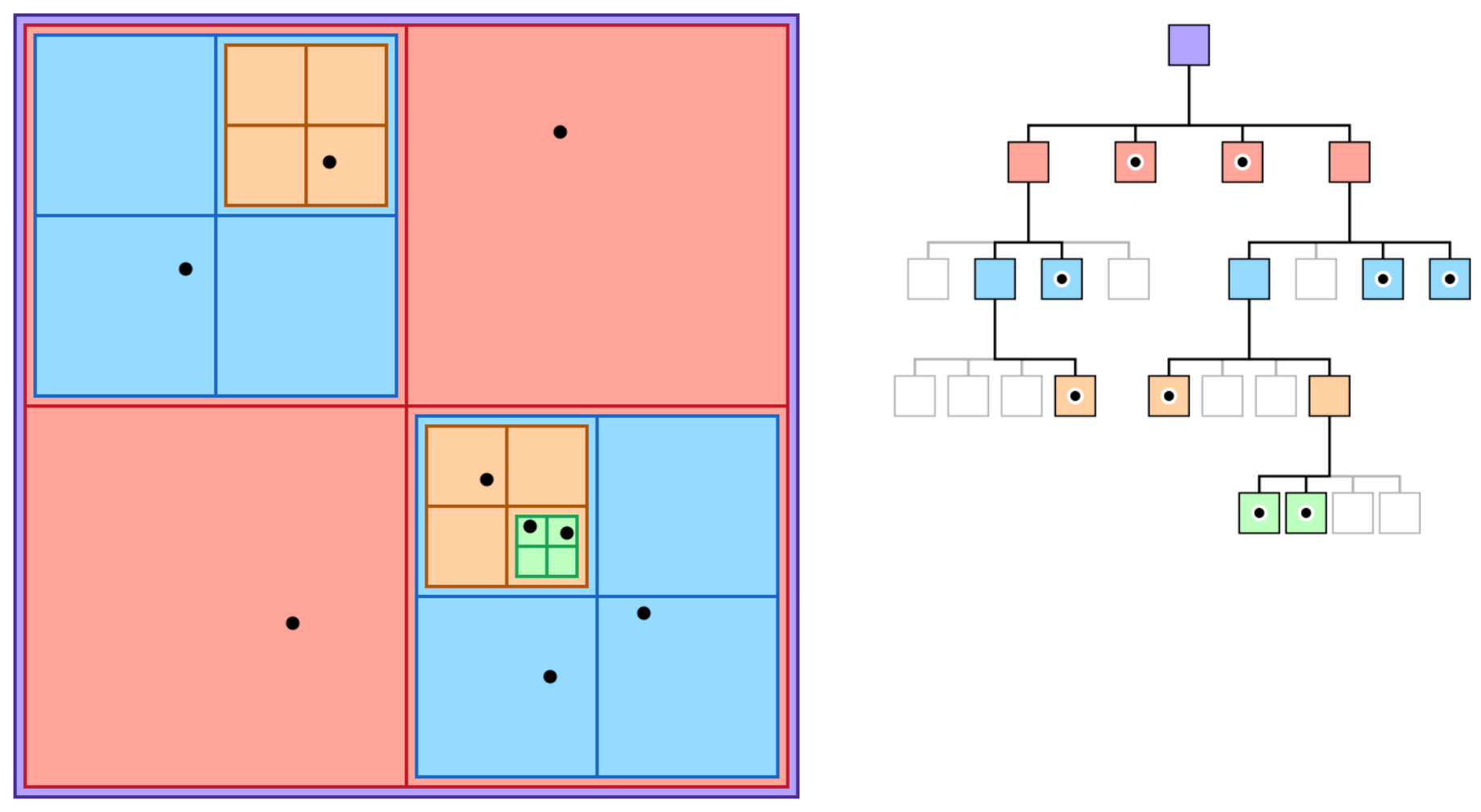
Sweep and Prune



Uniform Grid



Quad Tree



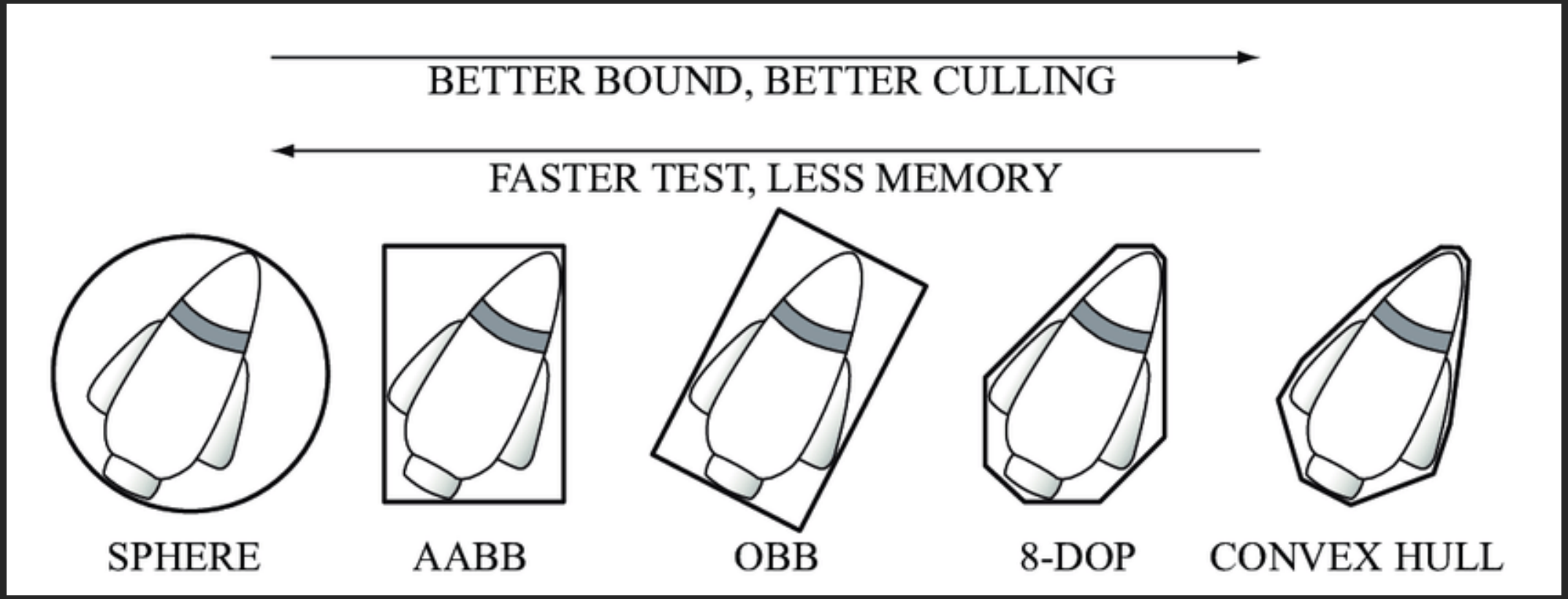
Narrow Phase

- At this point, we will have filtered all the objects that are close to each other.
- This is where you perform all your actual calculations for collision between two or more objects.
- Typically, there are two sub-phases here:
  - Checking for overlap
  - Determining the response

Bounding Volumes

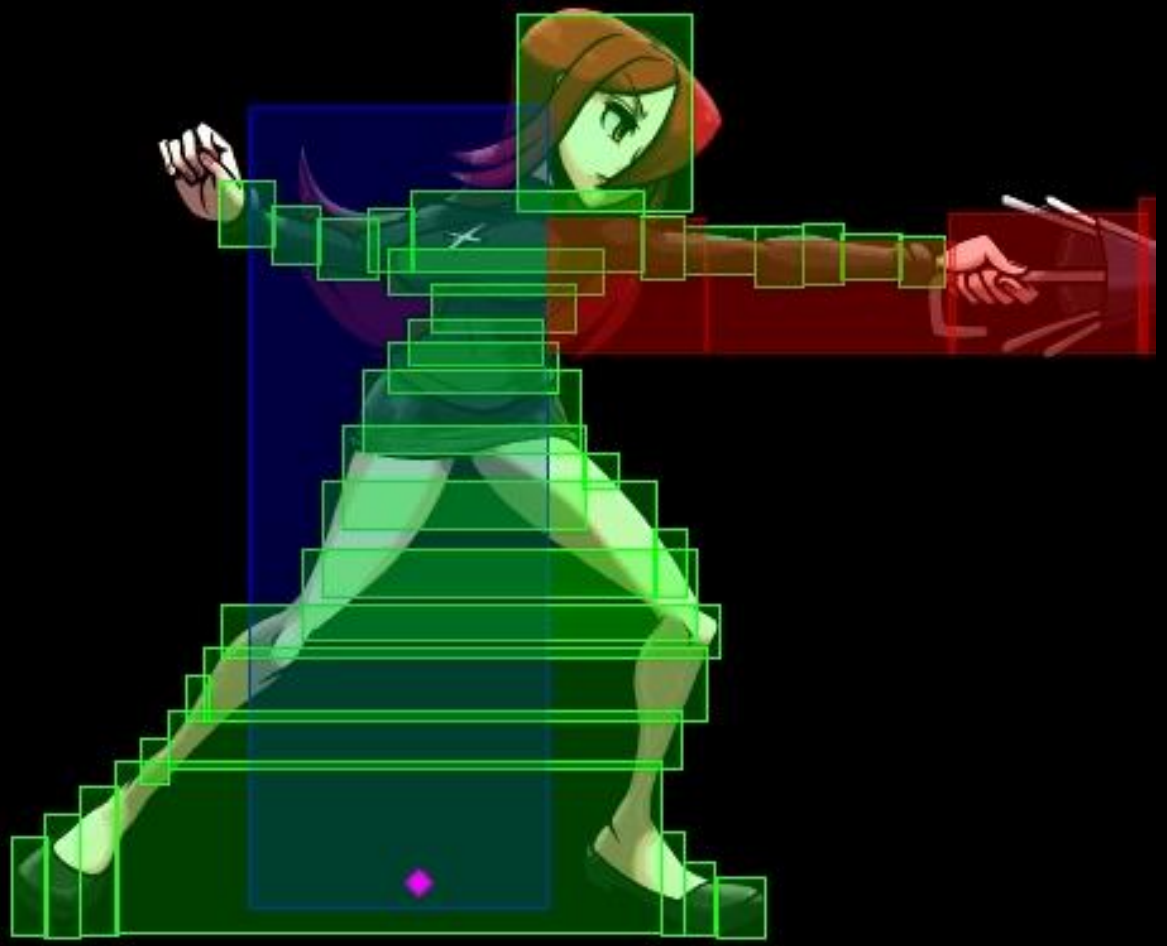
- It is difficult and costly to compute objects colliding with their actual shape. We need a faster alternative.
- Bounding Volumes is an approximation that covers the object.
- We use it to filter checks to optimize our physics systems.
  - Best communicate what type of Bounding Volumes you are going to have early to your engine team!

Bounding Volume Techniques



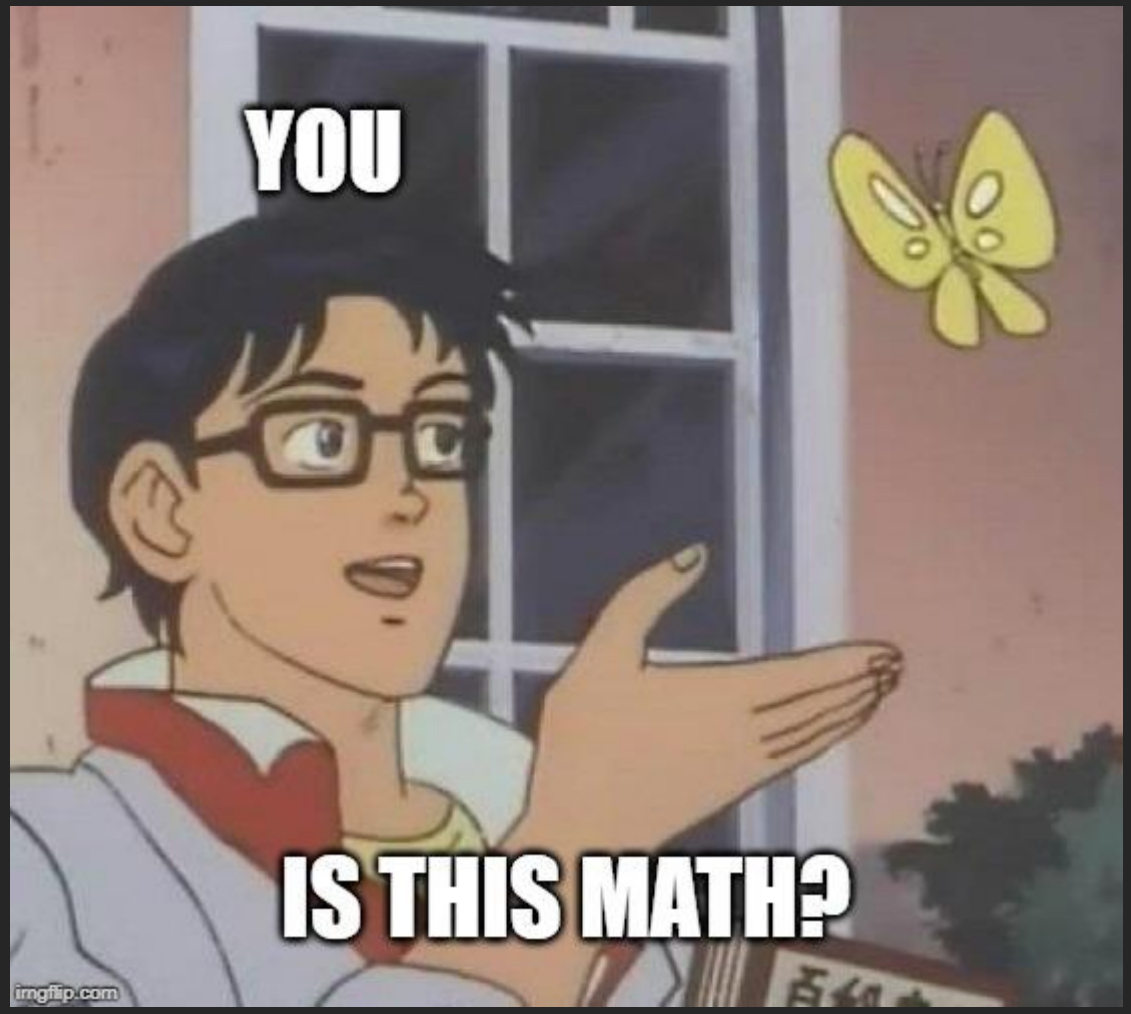
How to choose?

- Typically, if Spheres and/or AABBs are good enough for your game, just use them.
- Only consider the more complex ones if you really REALLY need the added accuracy.
- Always keep in mind that you are catering for each combination of shapes!
  - i.e. Having AABB and Spheres means 3 unique cases to code (AABB vs AABB, Sphere vs Sphere, AABB vs Sphere).

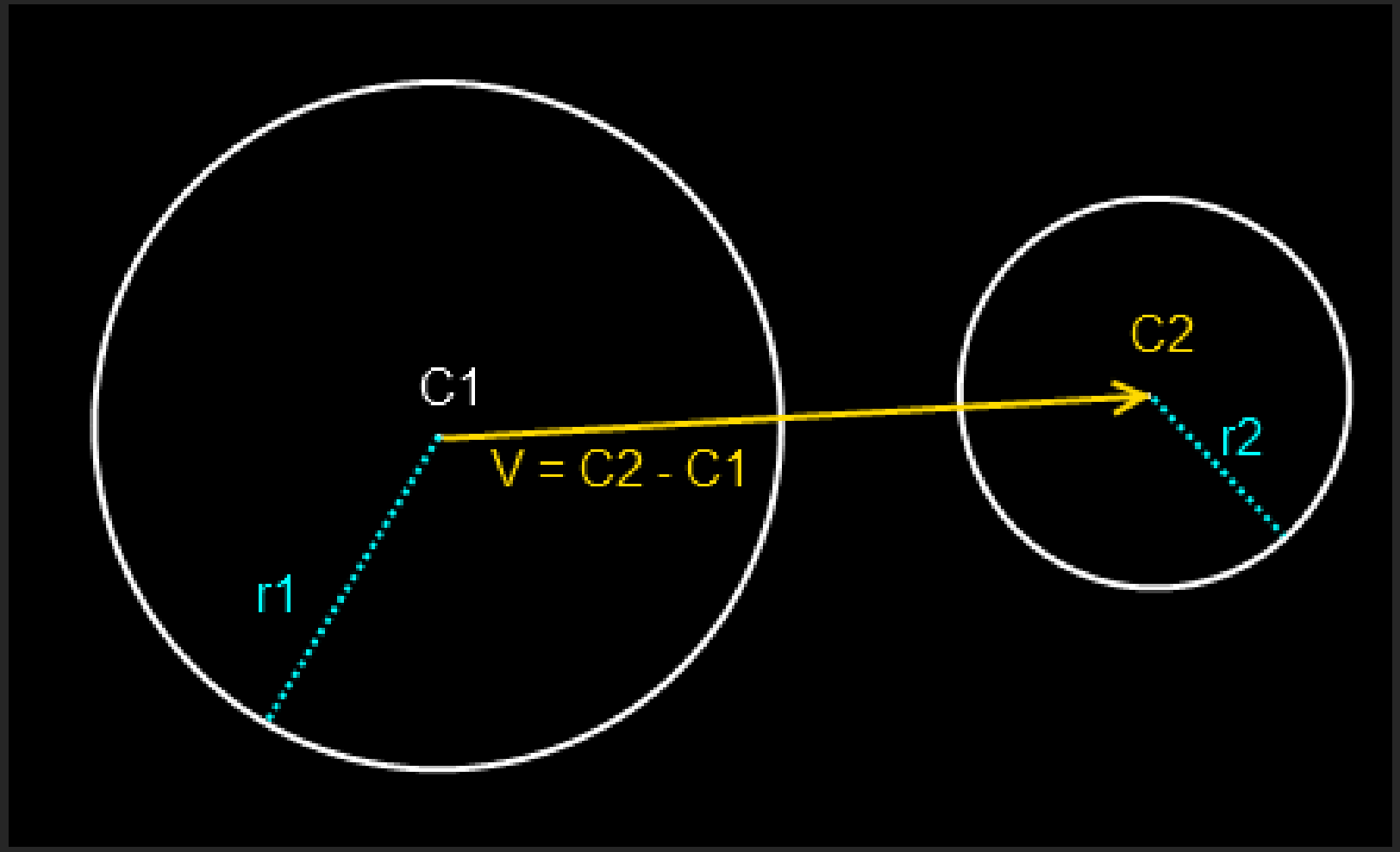



DOCTOR DOOM
WESKER
3RD LORD
5TH SCOUT
2 HITS!
181frm
PAUSE

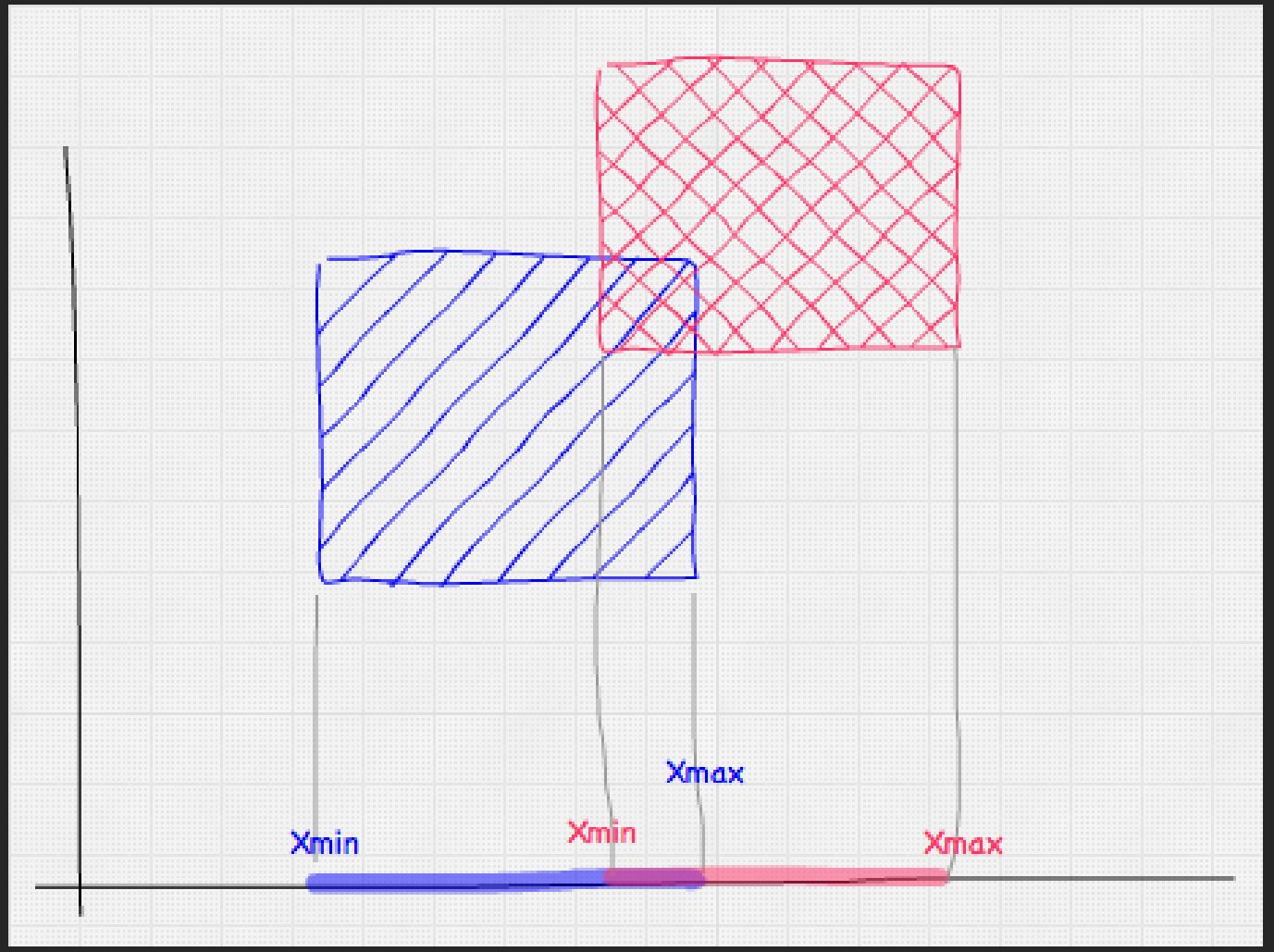
Checking for overlaps



Sphere vs Sphere

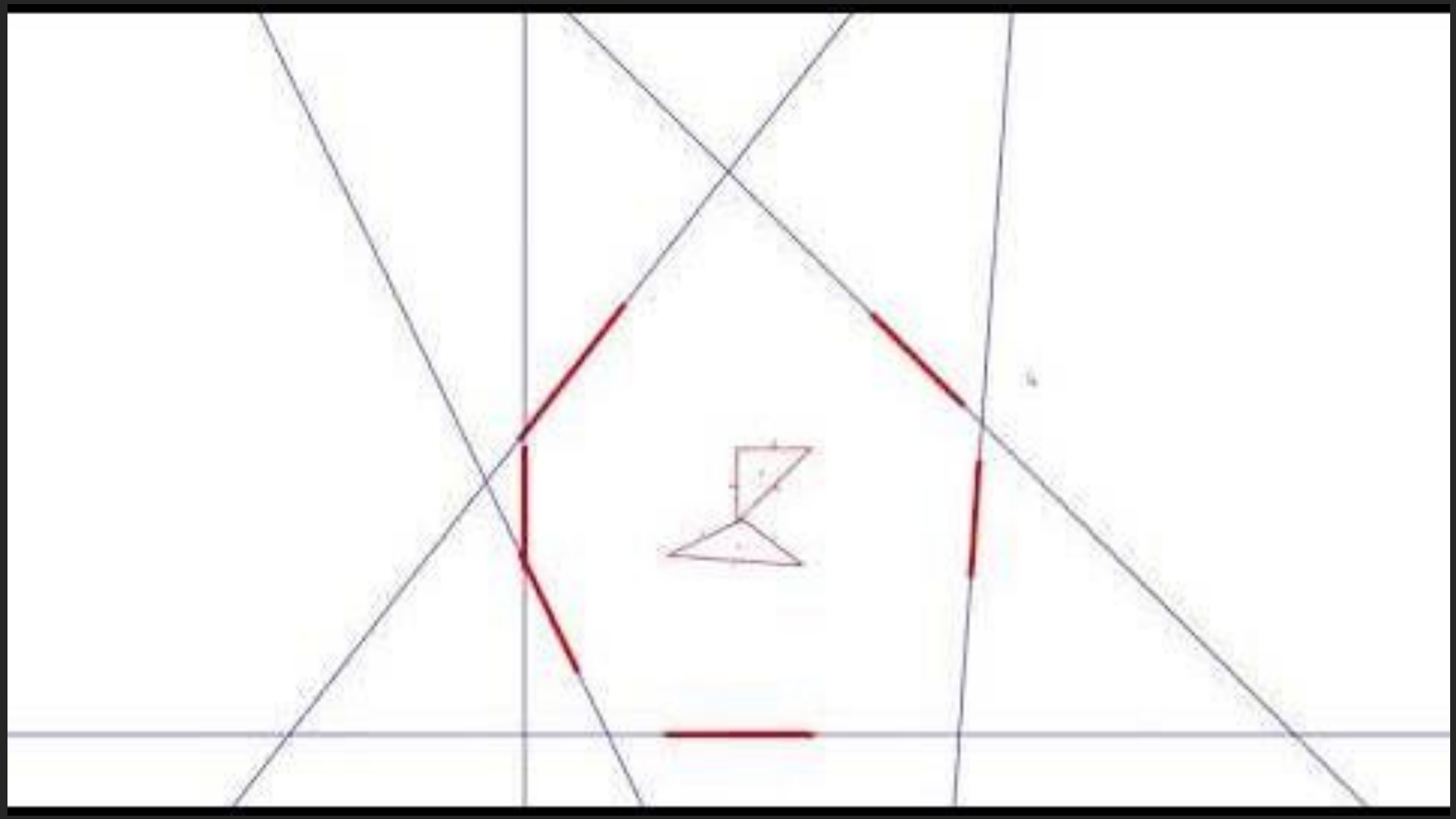


AABB vs AABB



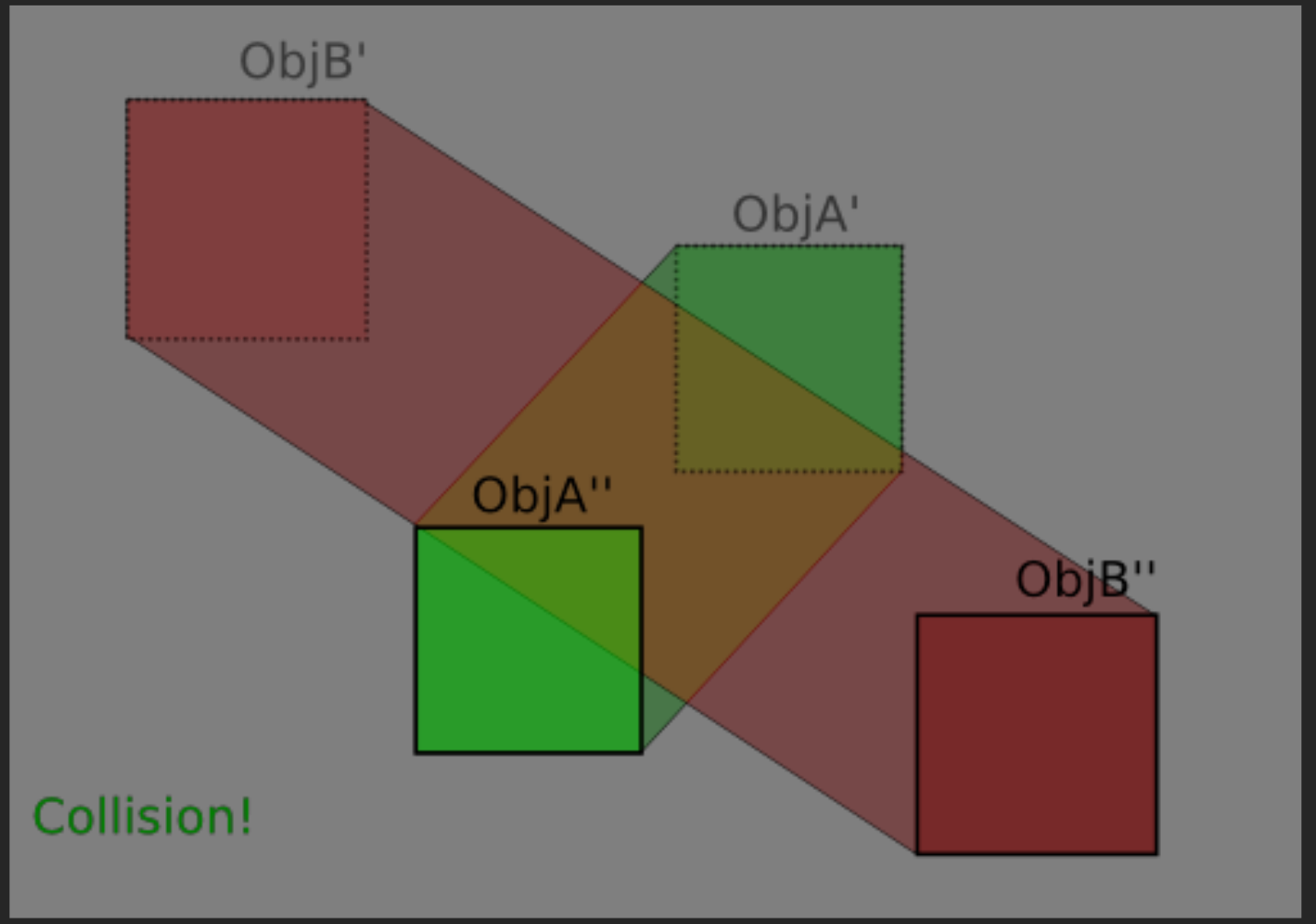
AABB vs Sphere?
OBB vs OBB?
AABB vs OBB?
Sphere vs OBB?
Convex Hull…?

Other techniques: SAT



https://www.youtube.com/watch?v=EzD7FY62A20

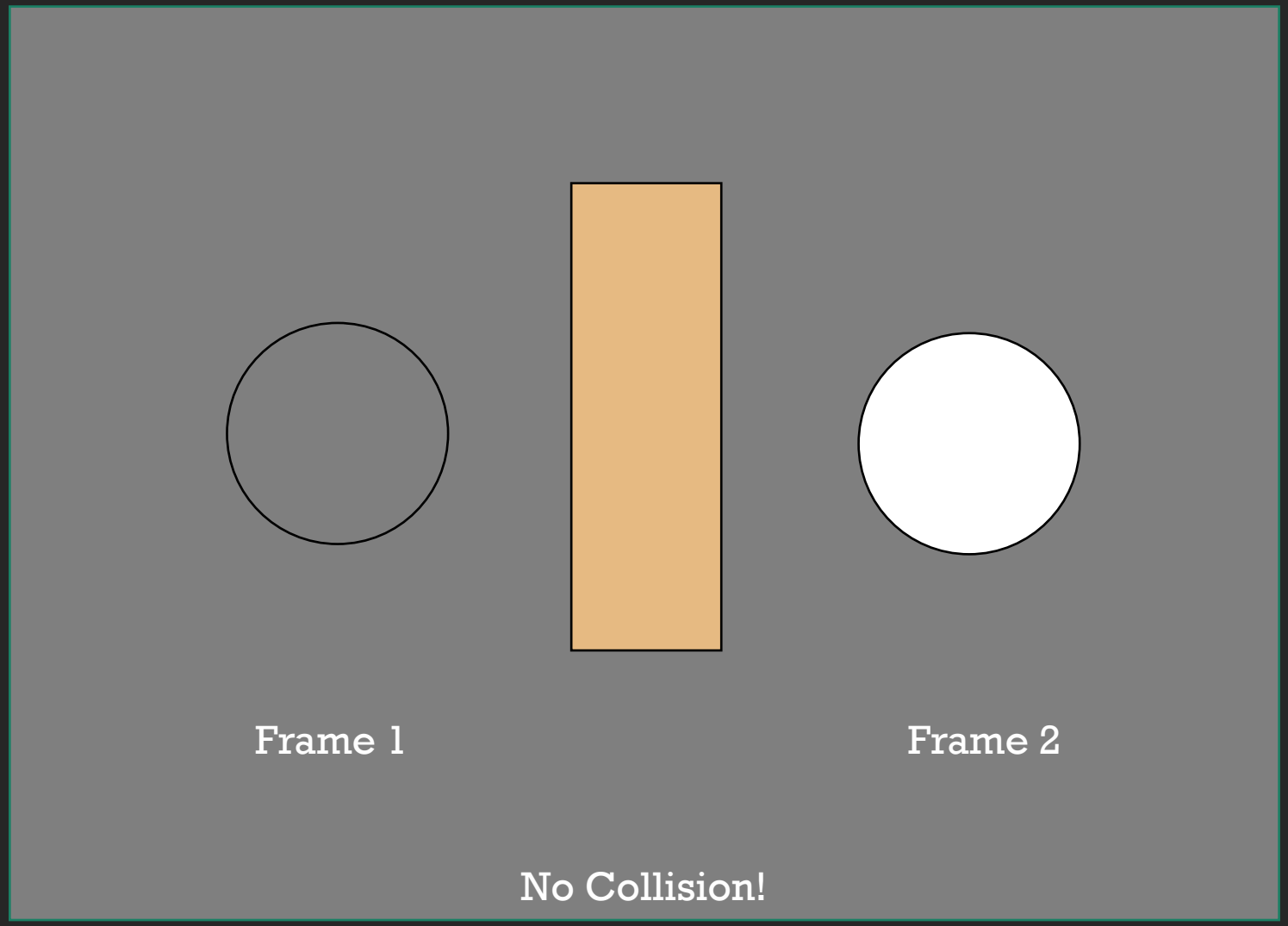
Fast Moving Objects



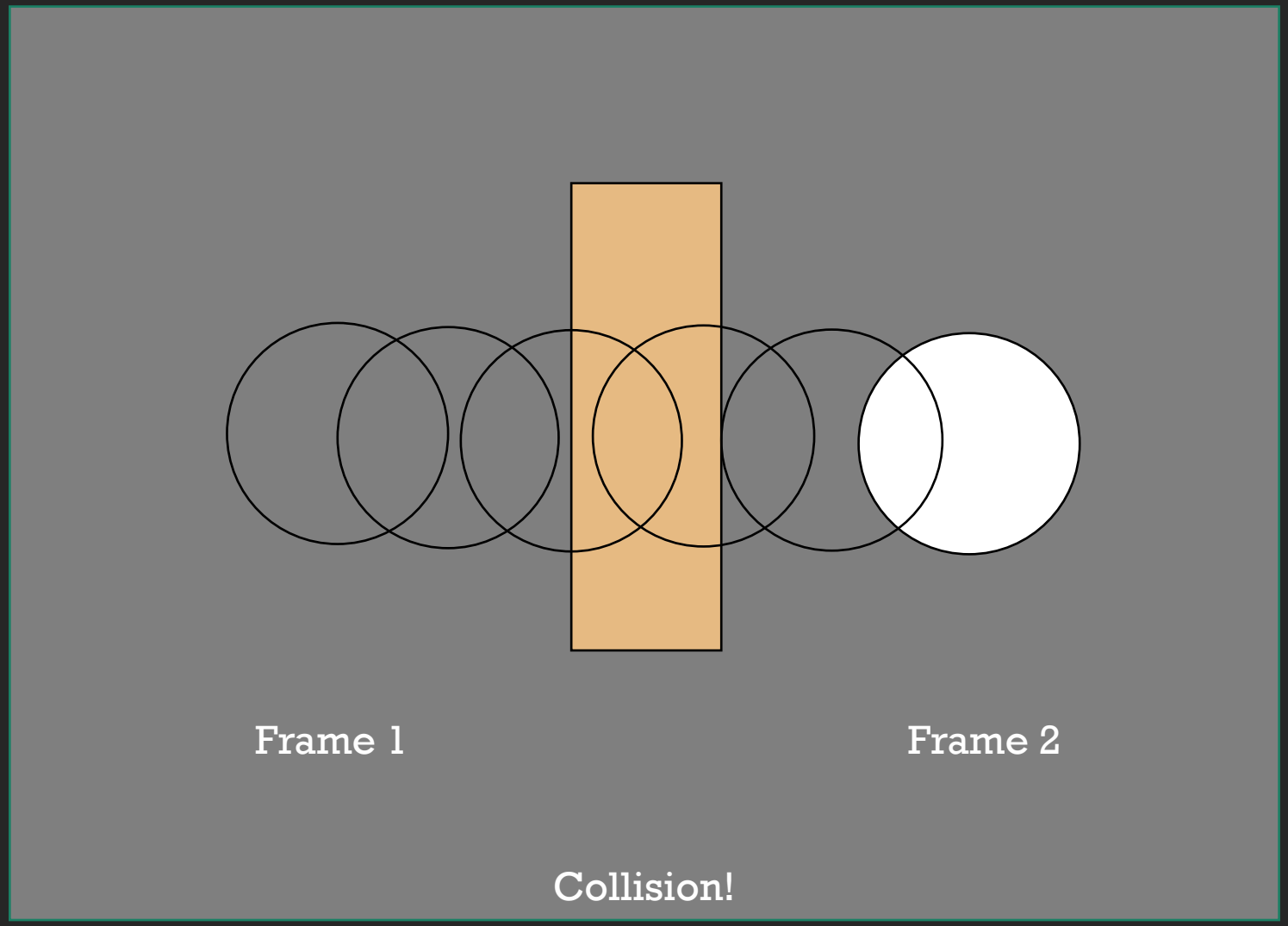
Fast Moving Objects

- If your game has many objects that moves so fast, sometimes they ‘bypass’ collision.
- You might not have to deal with this, depending on your product.
  - Capping the velocity of all your objects.
  - Capping delta time.
- If you must have fast-moving objects, then you might need to implement ‘CCD’.

Discrete Collision Detection



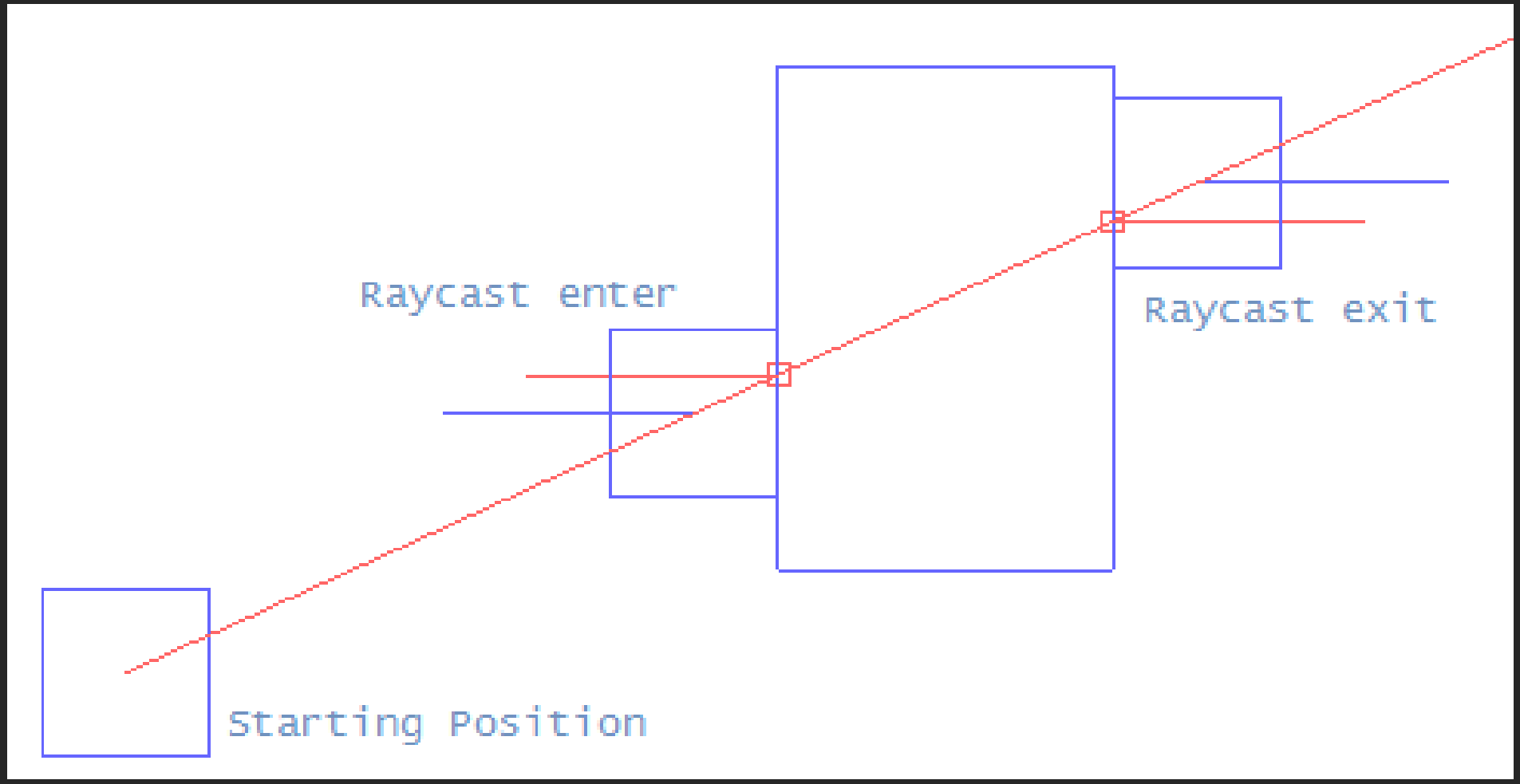
Continuous Collision Detection (CCD)



CCD

- Usually implemented as special case.
- Types of CCDs:
  - Sweep based CCDs
  - Speculative CCDs
- Don't forget to think about how this will fit with the rest of your system
  - esp if you have spatial partitioning implemented already!
- Good references:
  - https://docs.cocos.com/creator/manual/en/physics/physics-ccd.html
  - https://docs.unity3d.com/Manual/ContinuousCollisionDetection.html

Raycasting



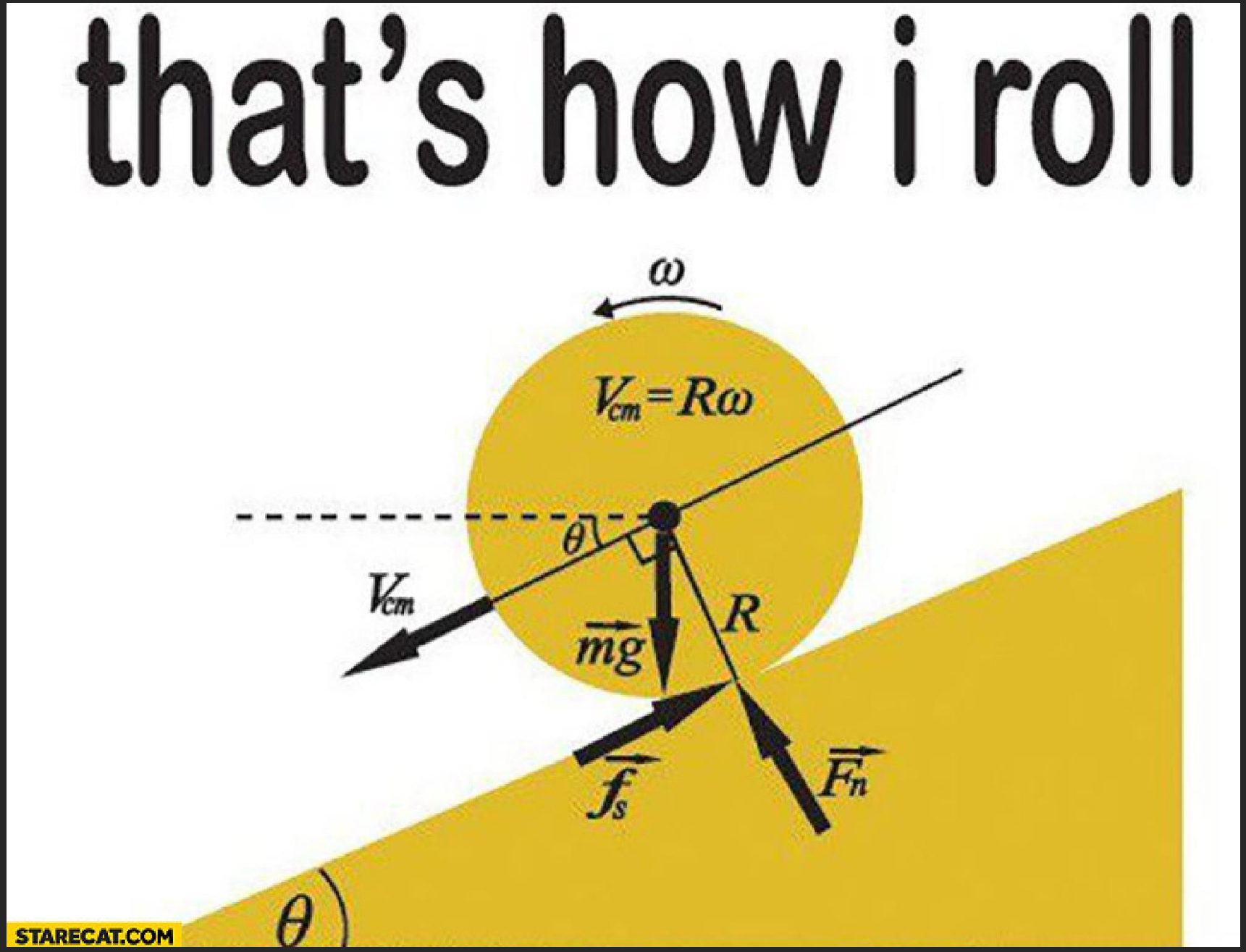
Raycasting

- It is yet another form of collision detection.
- Requires an origin position and a direction
- Gives you the points of intersection of any surfaces (and thus objects) it collides in that direction.
- Also useful in Graphics.

Collision Resolution

- This part deals with how objects respond when they collide with each other.
  - Hooke's law, conservation of momentum, friction, bounciness, etc…
- This is also where you resolve gameplay specific stuff!

that's how i roll



Things we didn't cover

- Bounciness/Friction/Restitution
- Rotational Physics
- Stacking boxes
- Voxel Physics