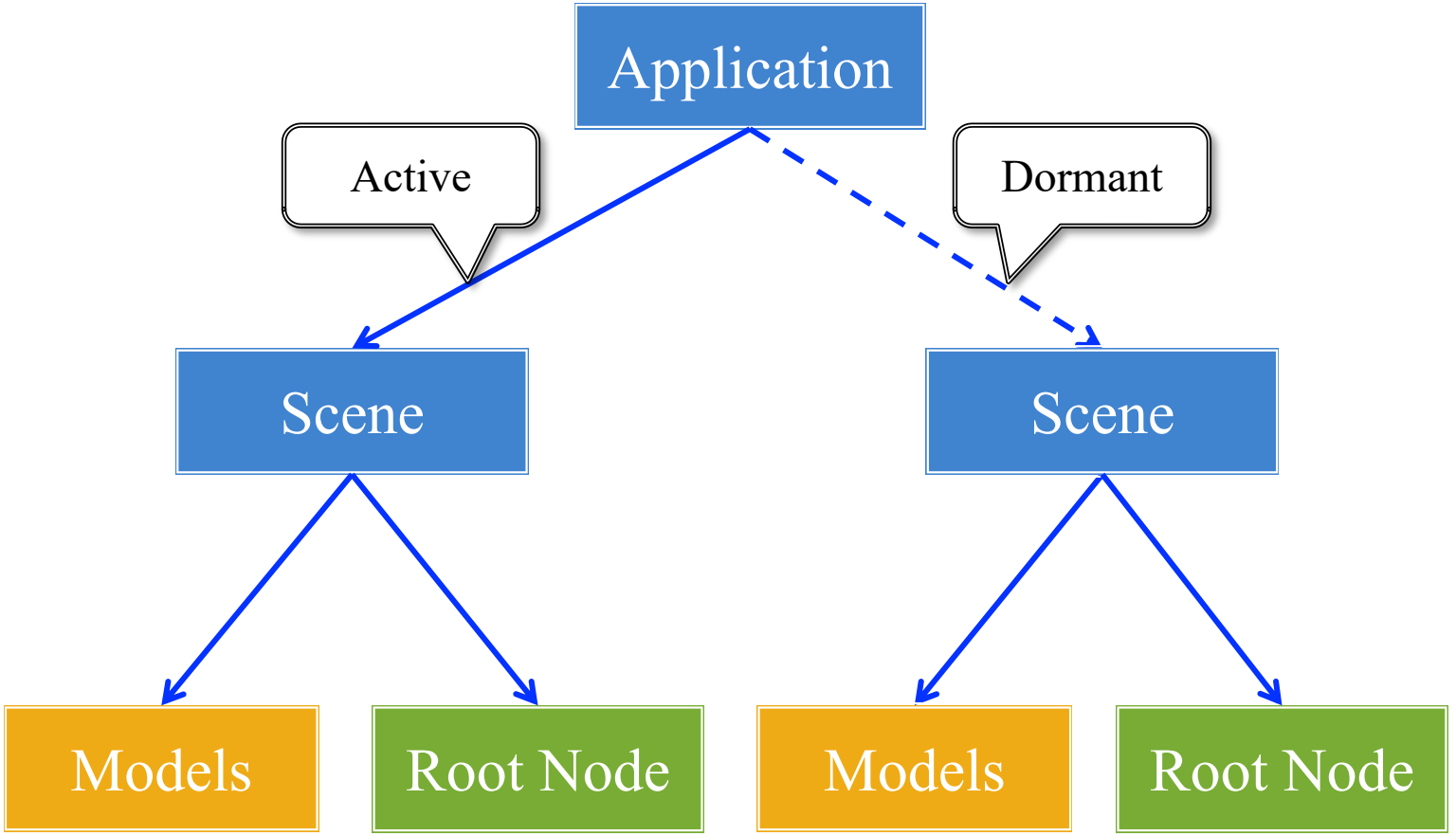
the gamedesigninitiative at cornell university

Lecture 5

# Scene Graphs

# **Recall**: Structure of a CUGL Application



# **Recall**: The Application Class

## `onStartup()`

- **After** backend is initialized
- Loads the game assets
  - Attaches the asset loaders
  - Loads immediate assets
- Starts any global singletons
  - **Example**: `AudioEngine`
- Creates any scenes
  - But does not launch ***yet***
  - Waits for assets to load

## `update()`

- Called each animation frame
- Manages gameplay
  - Converts input to actions
  - Processes NPC behavior
  - Resolves physics
  - Resolves other interactions
- Updates the scene graph
  - Transforms nodes
  - Enables/disables nodes

# Drawing a Scene in CUGL

- Use **render()** method
  - Called after update()
  - Clears screen first
  - Uses clear color attrib
- Draws the scene graph
  - Scene2 uses scene2
  - Scene3 uses scene3
- But you can override it
  - To use a SpriteBatch
  - To use *custom pipelines*

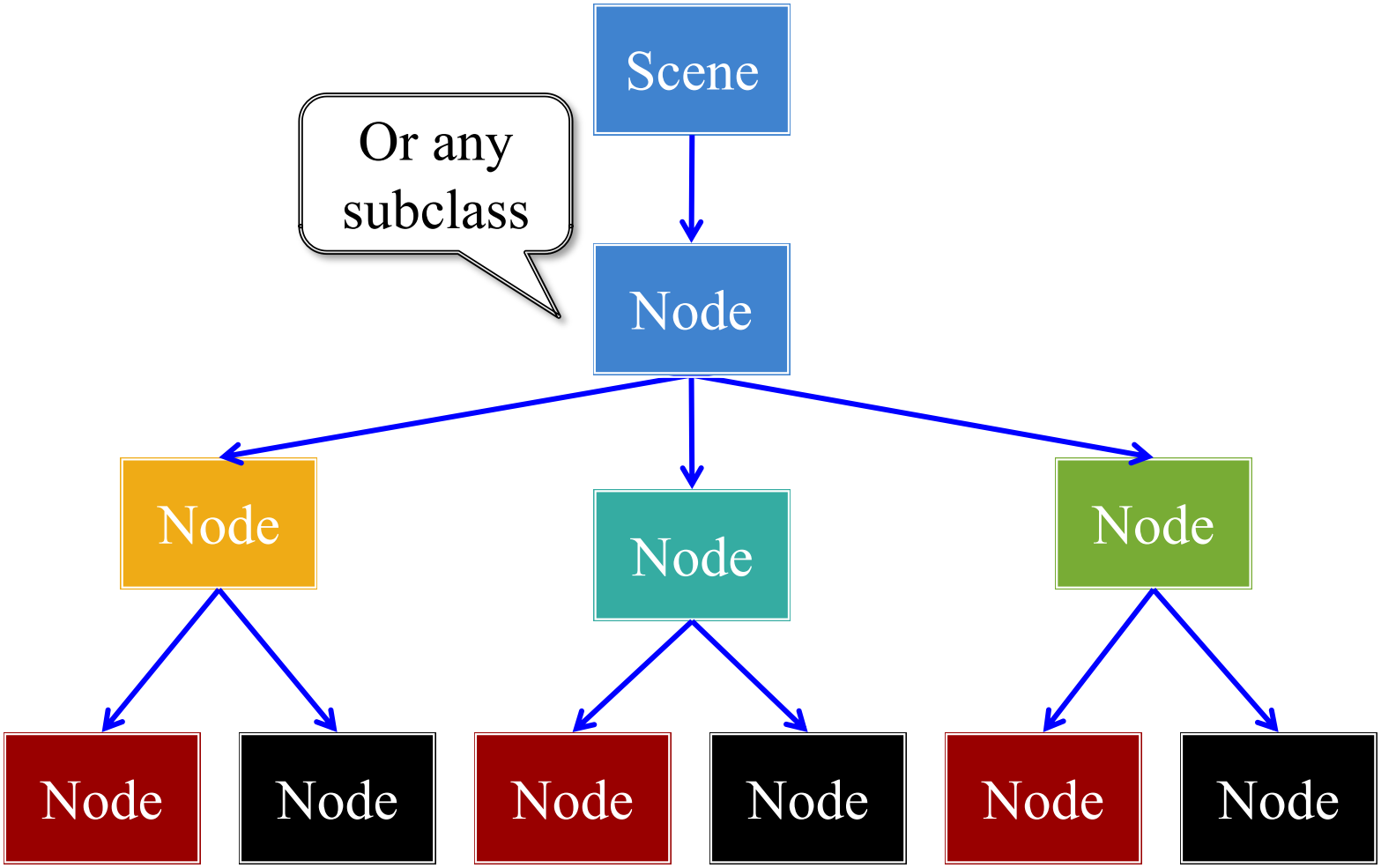
```
void render() {
    _batch->begin();
    _batch->draw(image1,Vec2(10,10));
    _batch->draw(image2,Vec2(50,20));
    _batch->end();
}
```

Attribute of Scene2

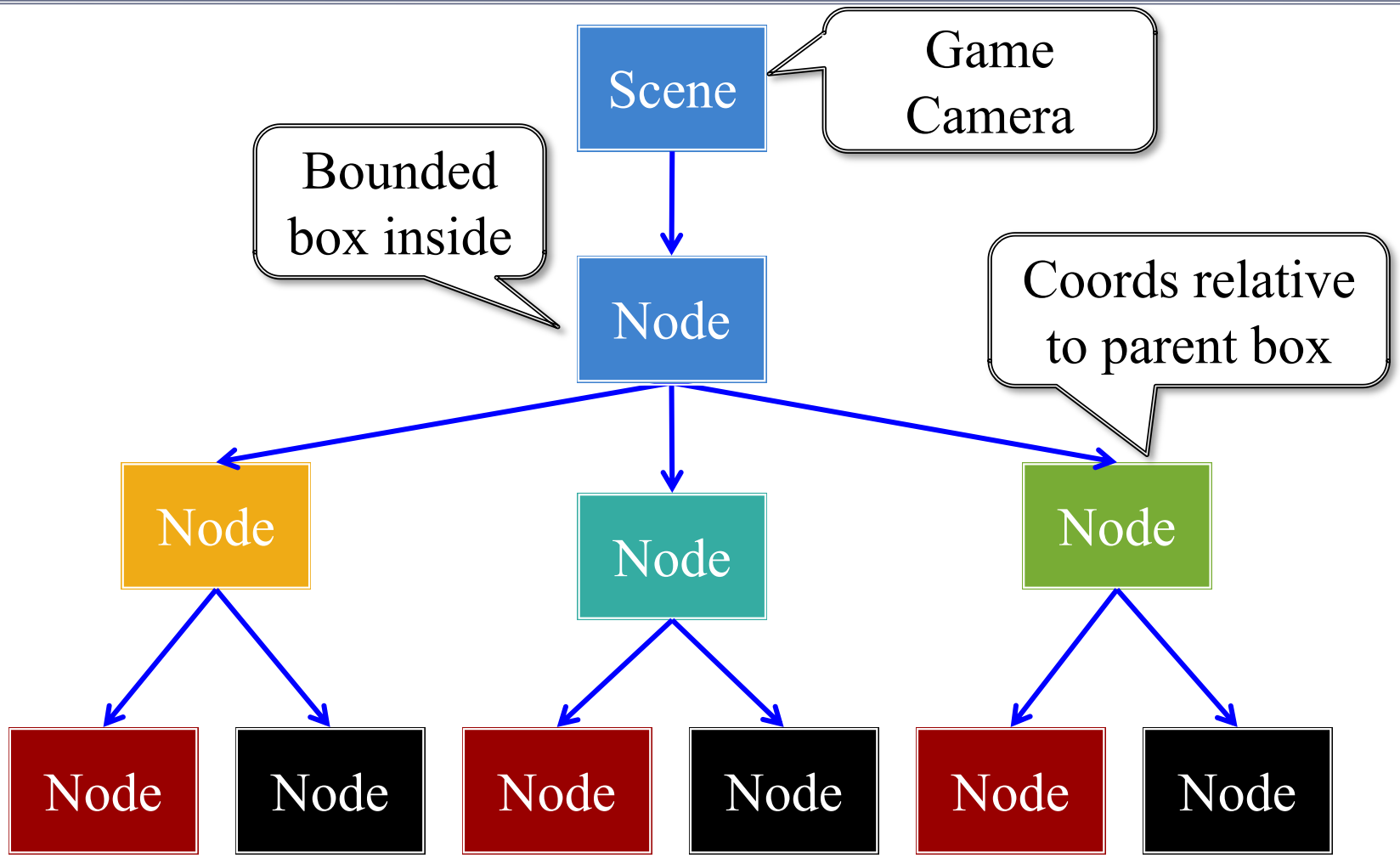
```
void render() {
    _shader->setVertices(0, _vertbuff);
    _shader->setIndices(_indxbuff);

    _shader->begin();
    _shader->pushInt("uType", _type);
    _shader->pushMat4("uPerspective", _matrix);
    _shader->drawIndexed(_icount);
    _shader->end();
}
```

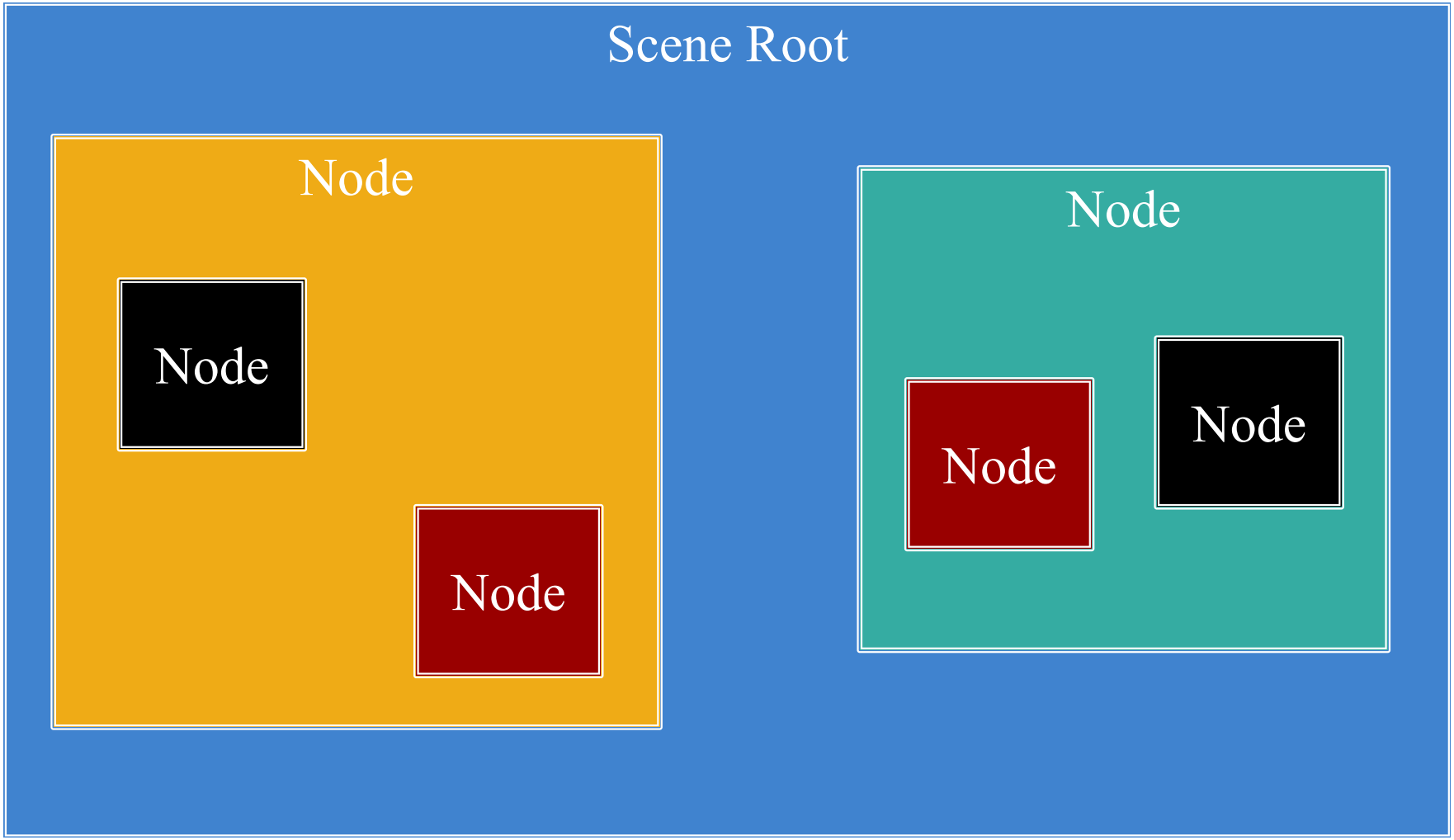
# The Scene Graph



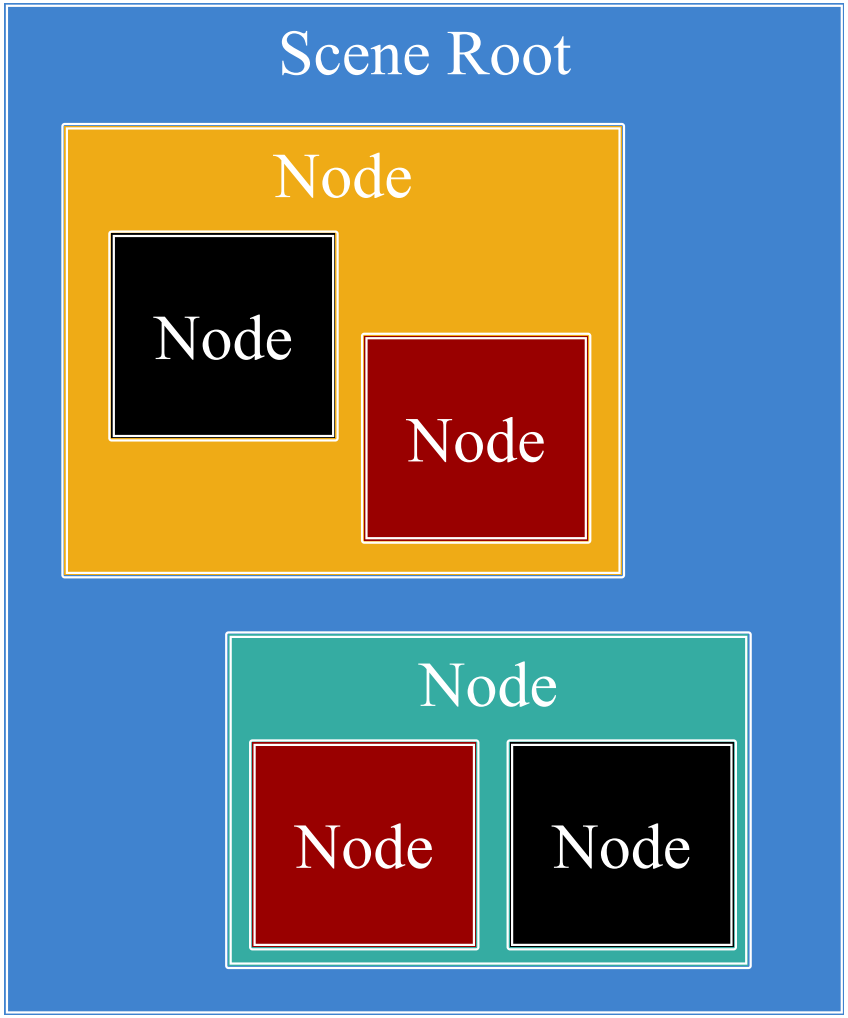
# The Scene Graph

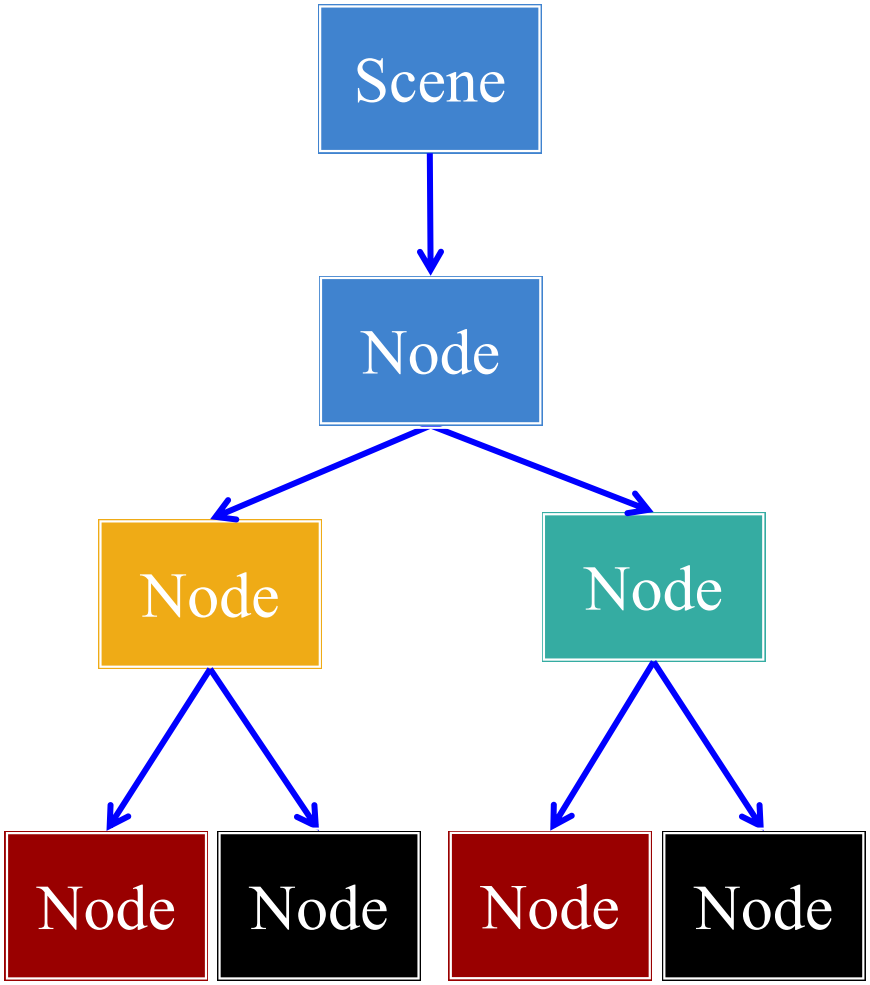


# Each Node is a Coordinate System

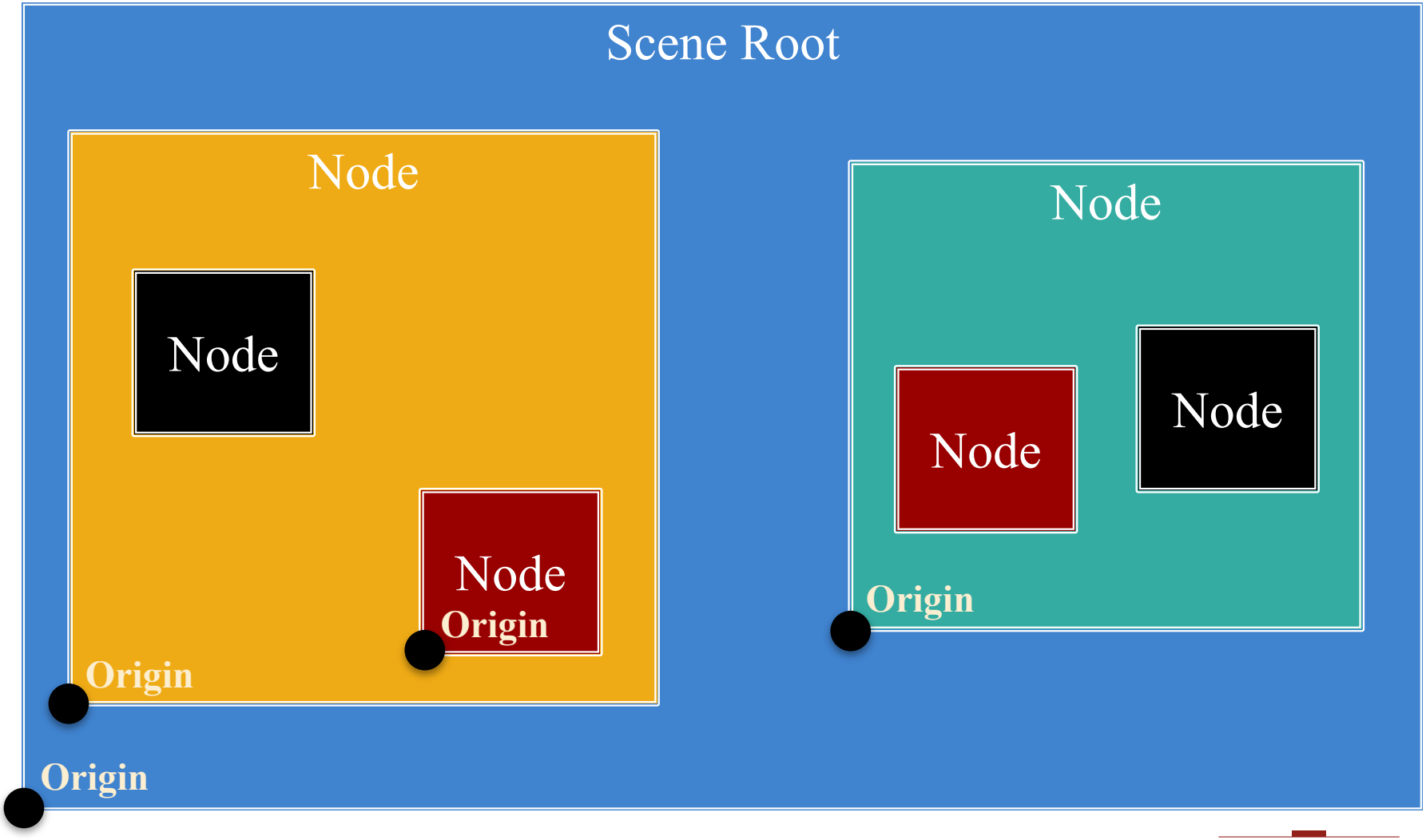


# Each Node is a Coordinate System



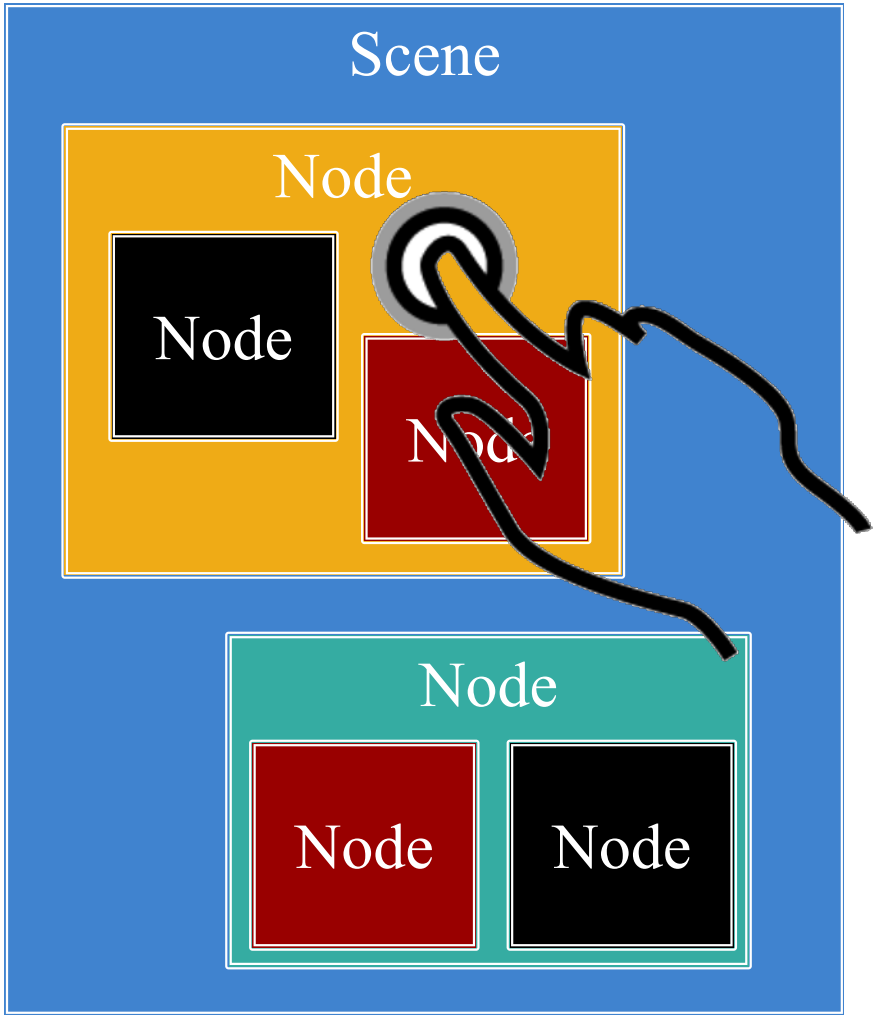



# Each Node is a Coordinate System

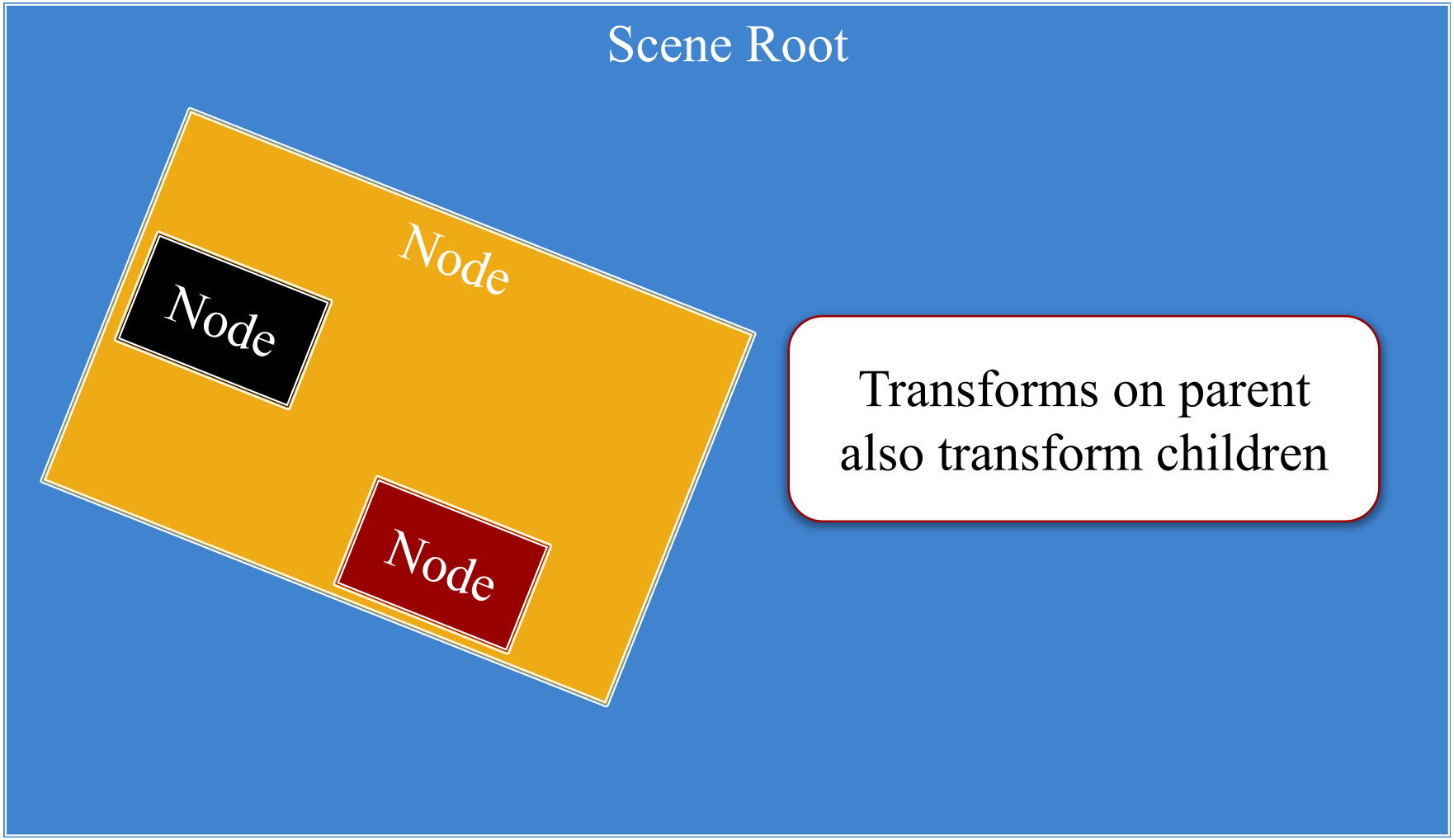


# Motivation: Touch Interfaces

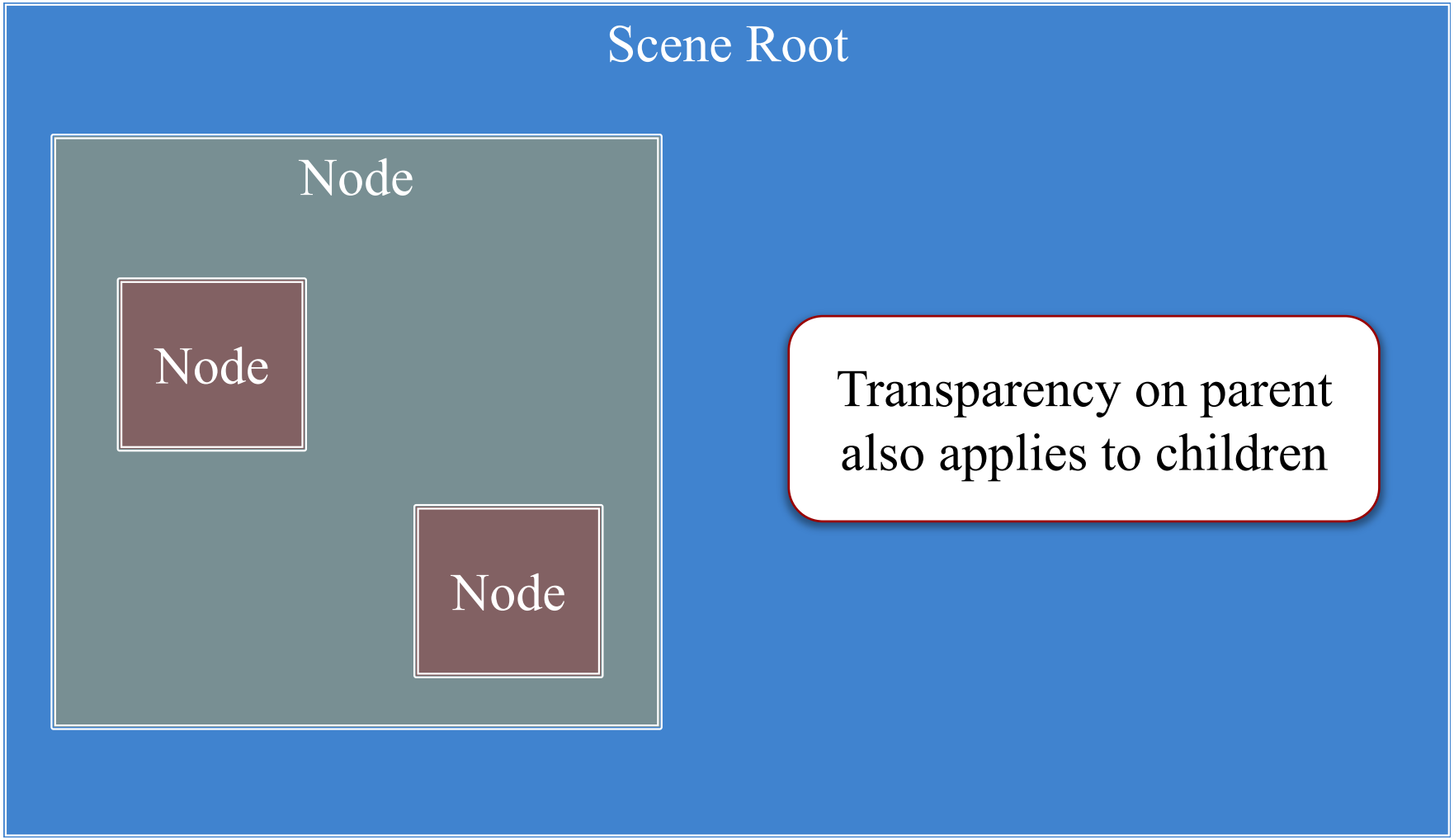
- Touch handler requires
  - Which object touched
  - Location inside object
- Scene graph is a *search tree*
  - Check if touch is in parent
  - … then check each child
  - Faster than linear search
- But limit this to a **search**
  - No input control in node
  - Use polling over callbacks



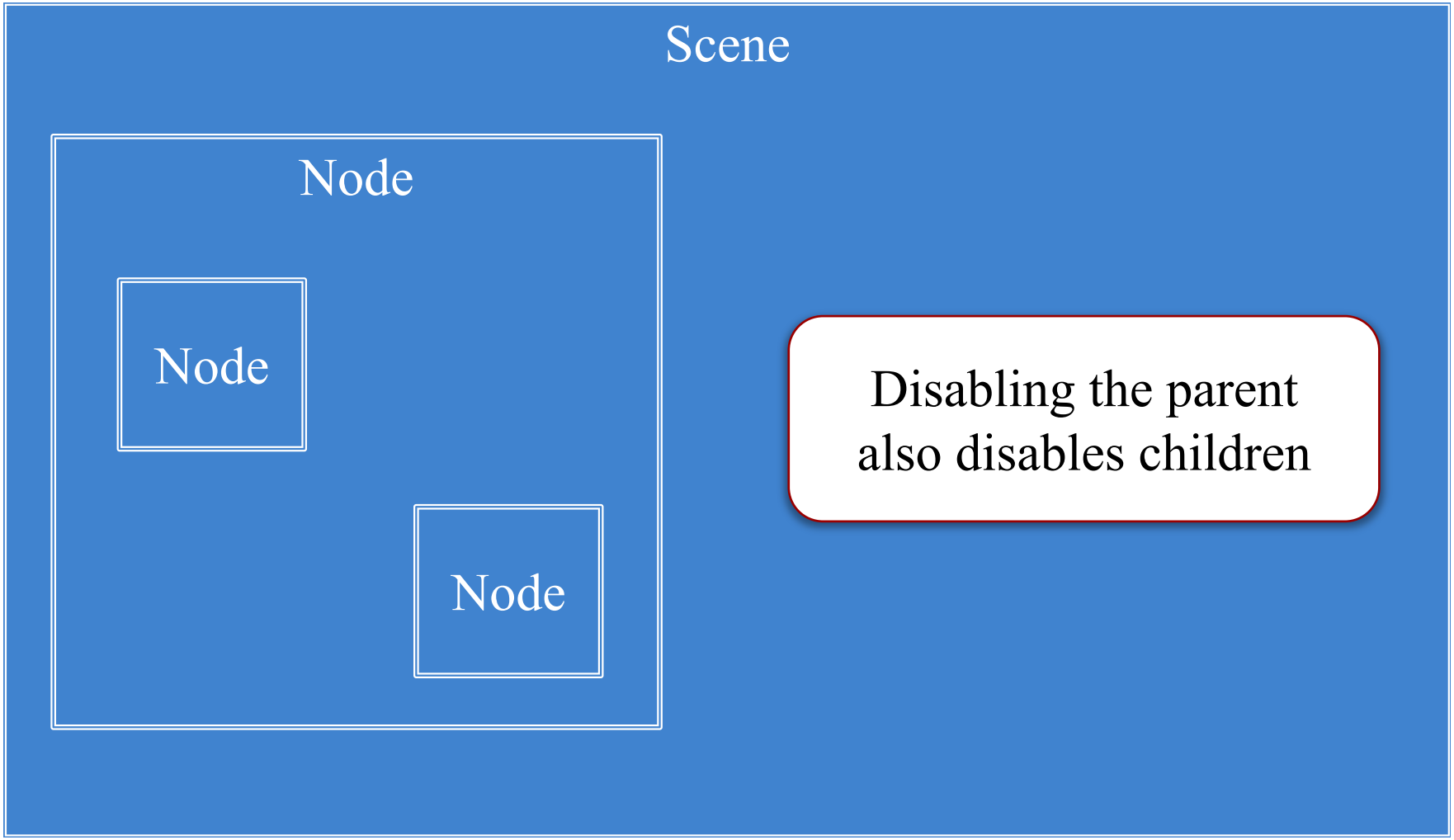
# Settings Pass Down the Graph



# Settings Pass Down the Graph

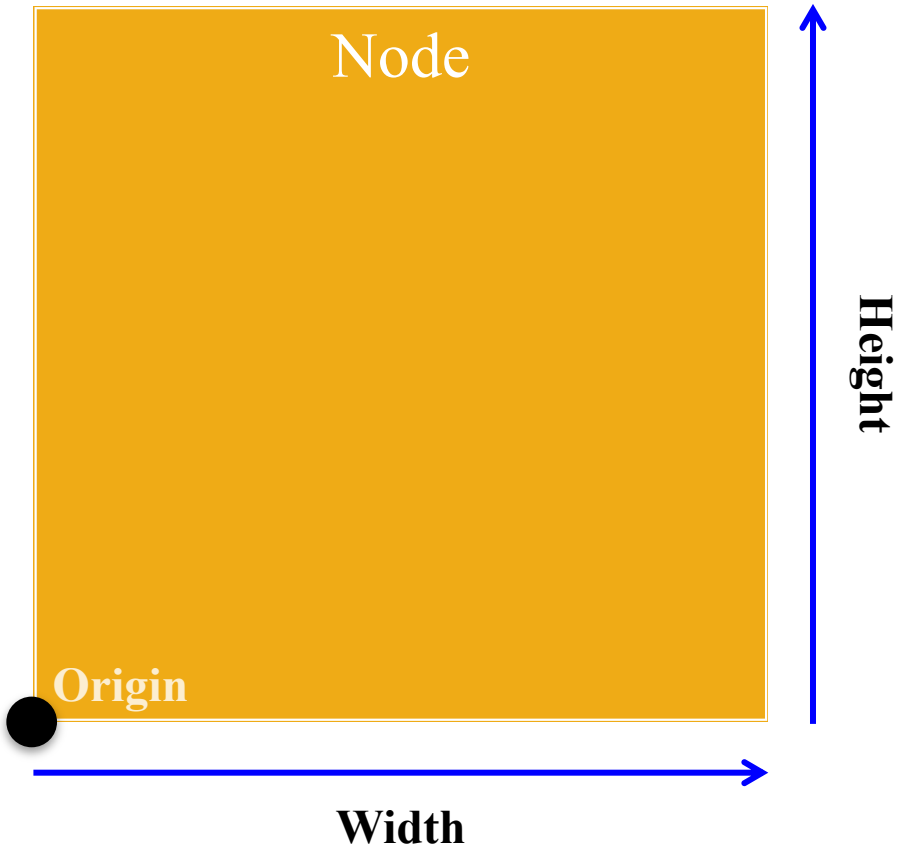


# Settings Pass Down the Graph



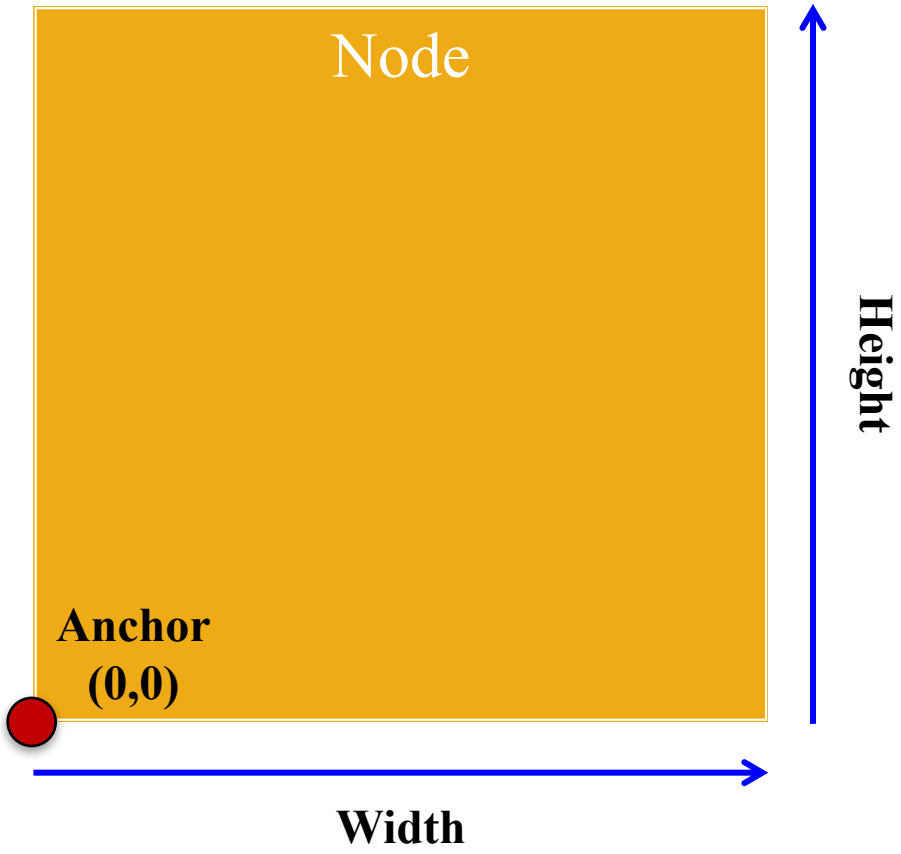
# Anchors and Content

- Nodes have **content size**
  - Width/height of contents
  - Measured in node space
  - But only a guideline: content can be outside
- Nodes have an **anchor**
  - Location in node space
  - *Percentage* of width/height
  - Does not affect the origin
- Both may affect **position**



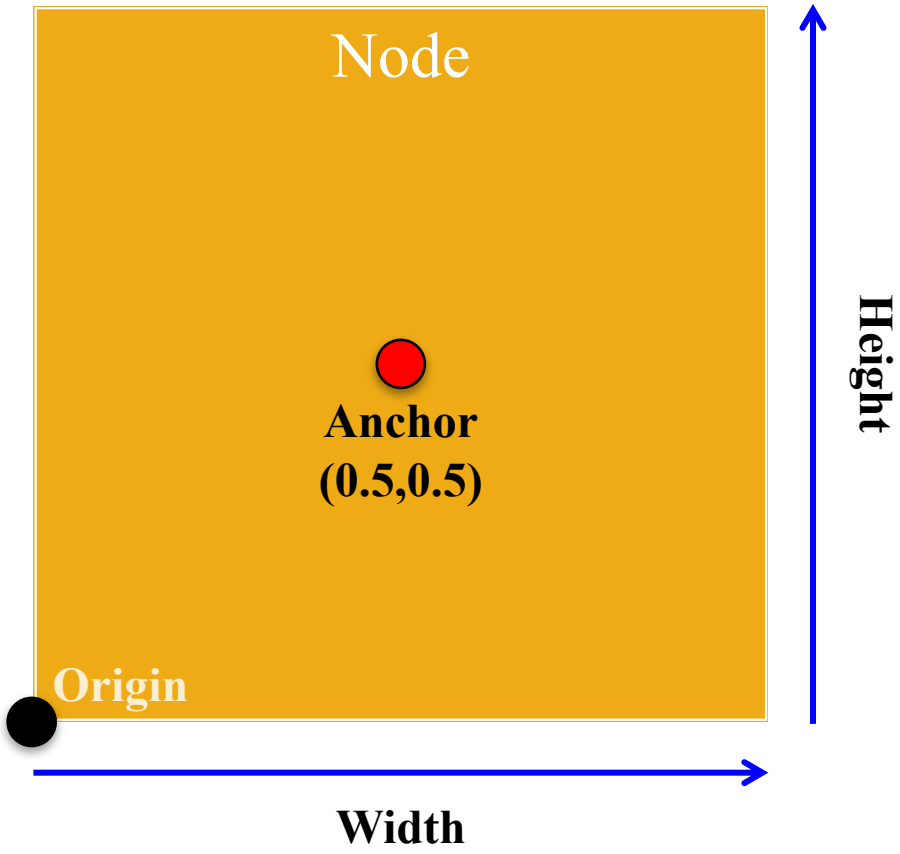
# Anchors and Content

- Nodes have **content size**
  - Width/height of contents
  - Measured in node space
  - But only a guideline: content can be outside
- Nodes have an **anchor**
  - Location in node space
  - *Percentage* of width/height
  - Does not affect the origin
- Both may affect **position**



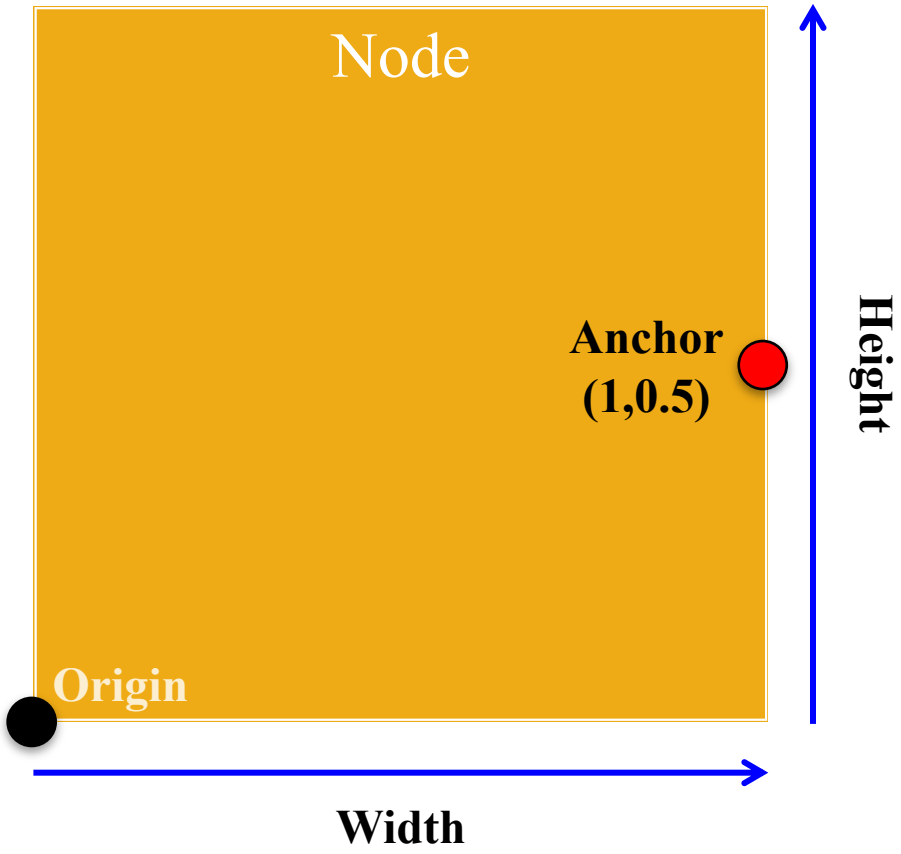
# Anchors and Content

- Nodes have **content size**
  - Width/height of contents
  - Measured in node space
  - But only a guideline: content can be outside
- Nodes have an **anchor**
  - Location in node space
  - *Percentage* of width/height
  - Does not affect the origin
- Both may affect **position**

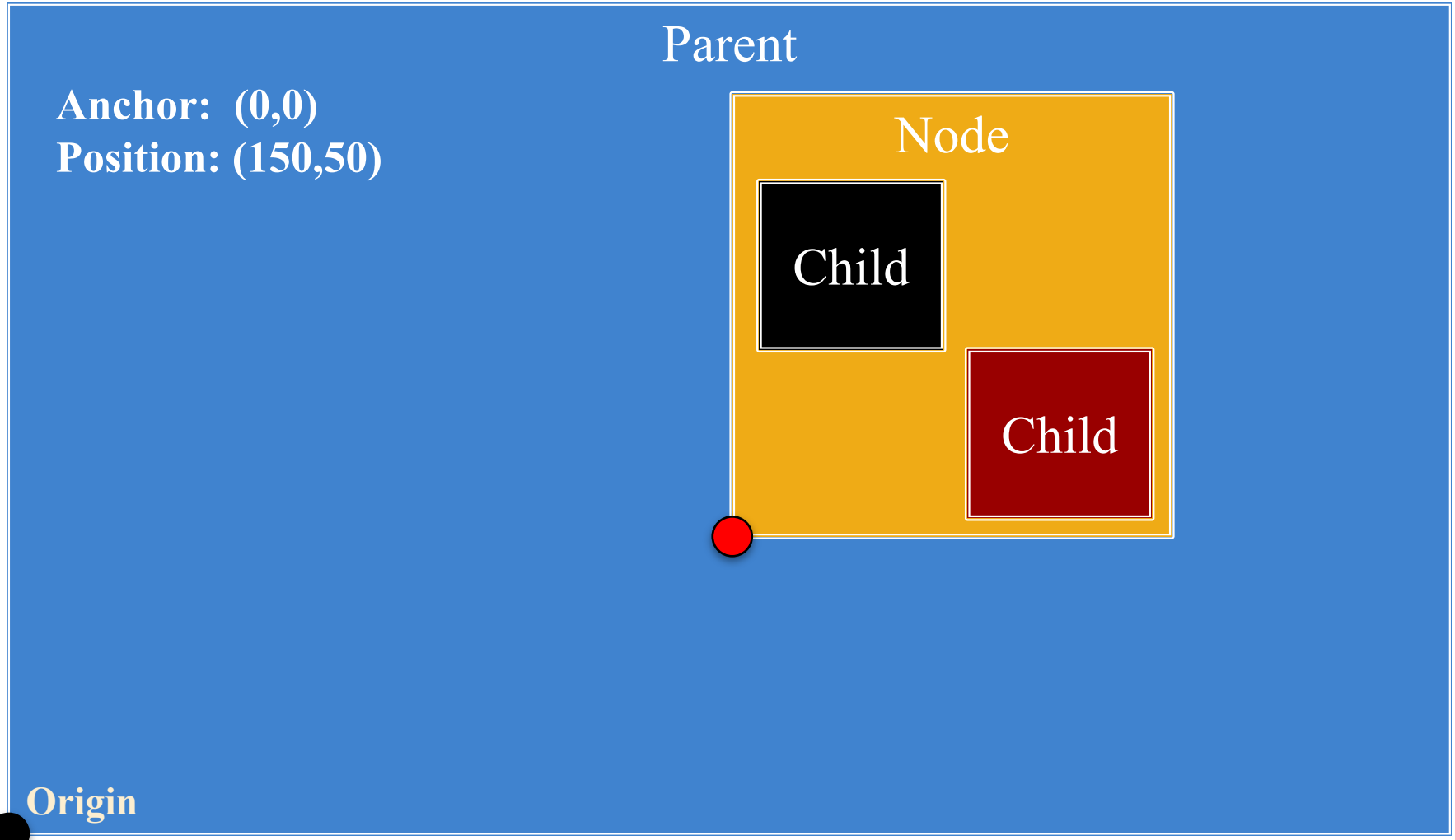


# Anchors and Content

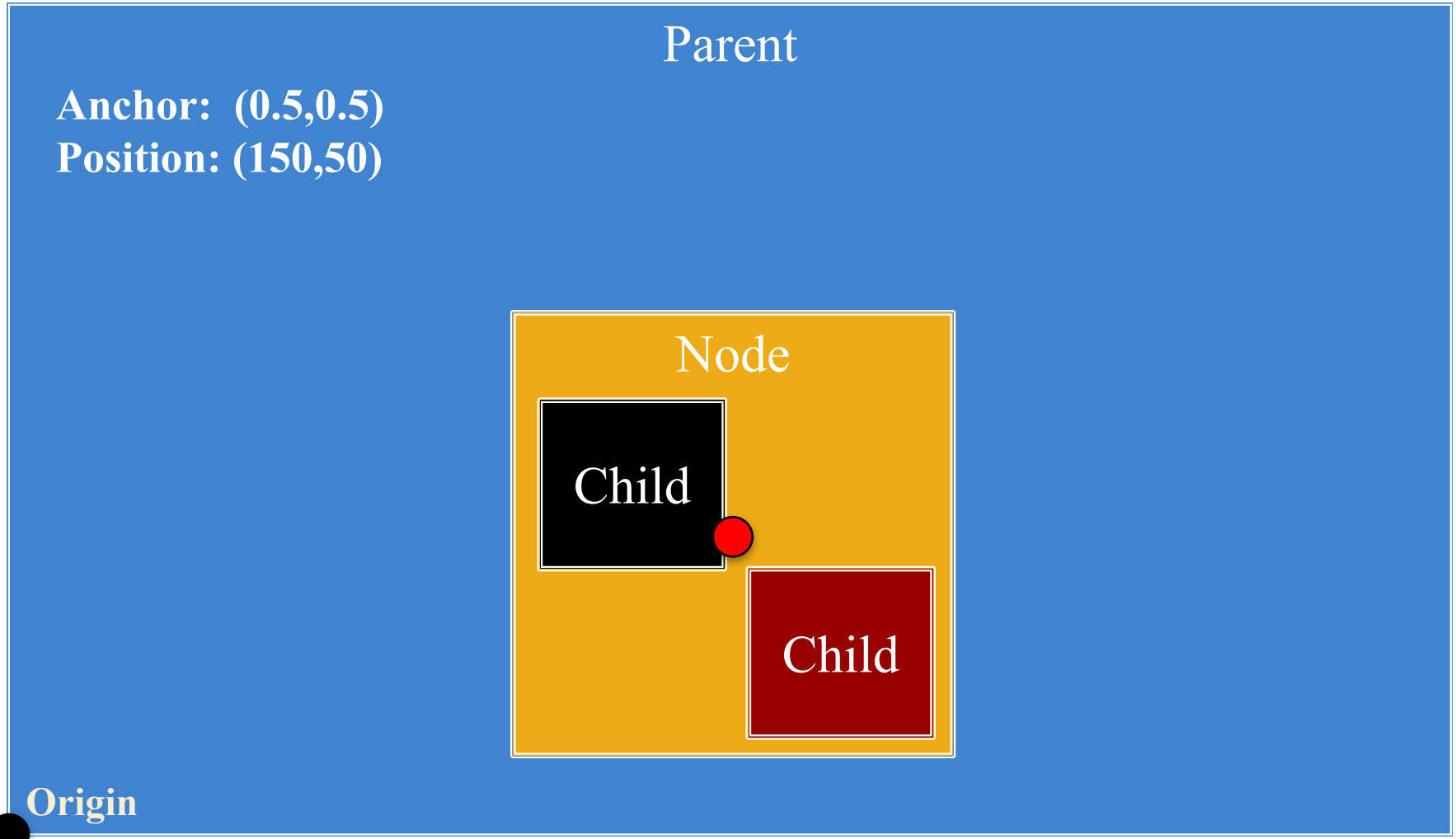
- Nodes have **content size**
  - Width/height of contents
  - Measured in node space
  - But only a guideline: content can be outside
- Nodes have an **anchor**
  - Location in node space
  - *Percentage* of width/height
  - Does not affect the origin
- Both may affect **position**



# Anchor and Position

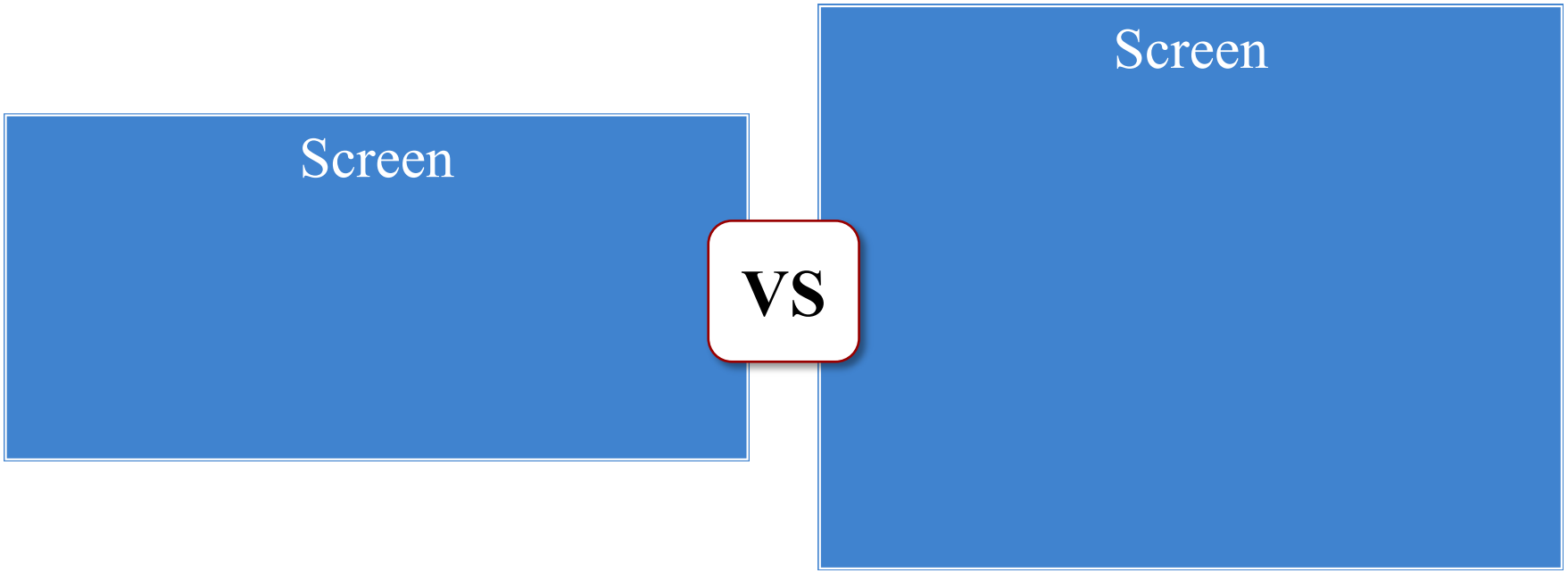


# Anchor and Position



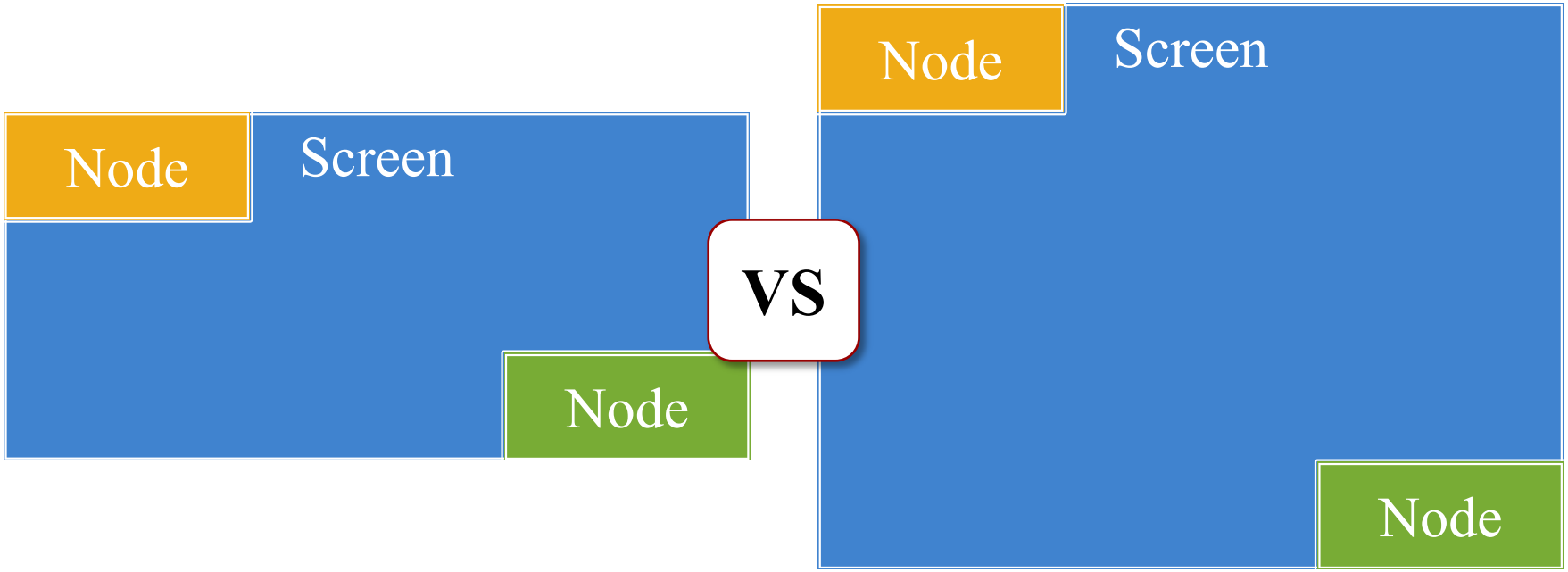
# Layout Managers

- Not all devices have the same aspect ratio
- Sometimes, want placement to adjust to fit

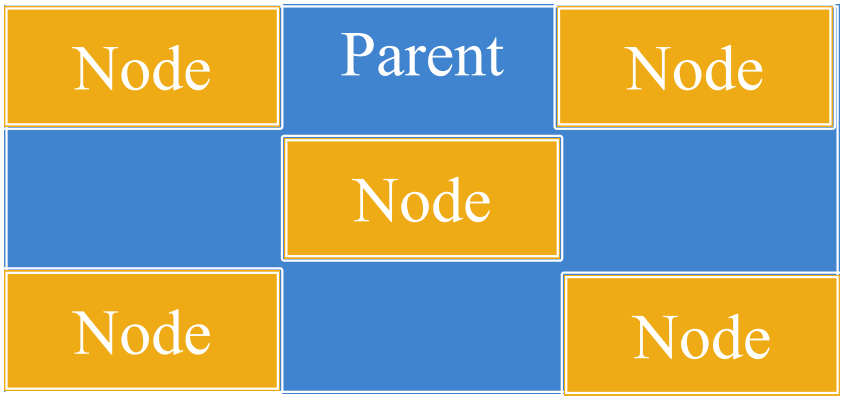


# Layout Managers

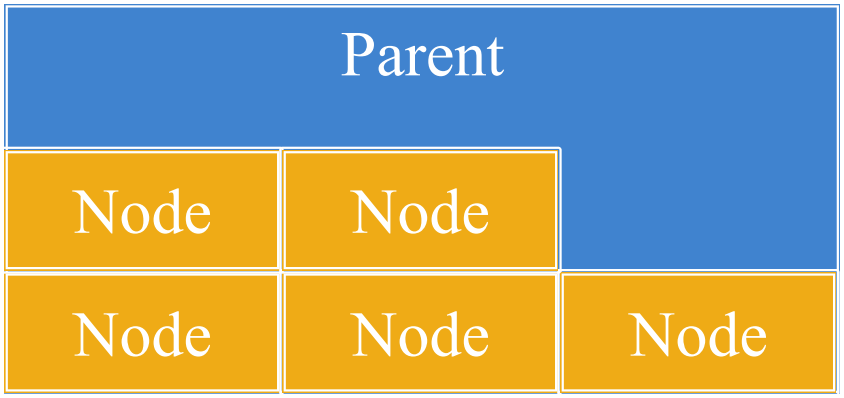
- Not all devices have the same aspect ratio
- Sometimes, want placement to adjust to fit



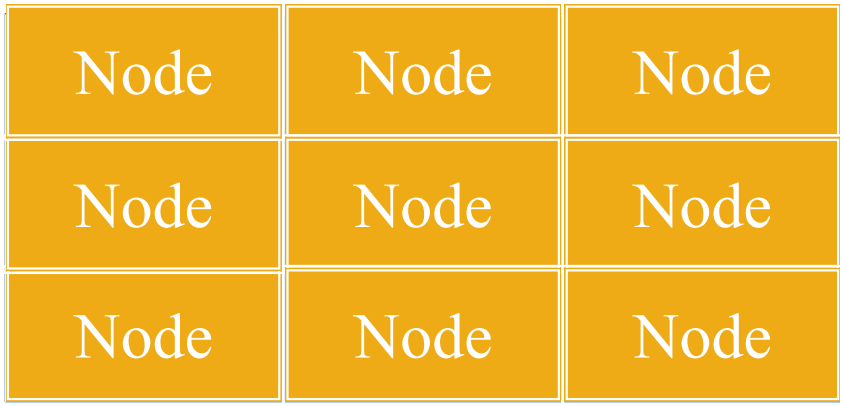
# Layout Managers




**AnchorLayout**

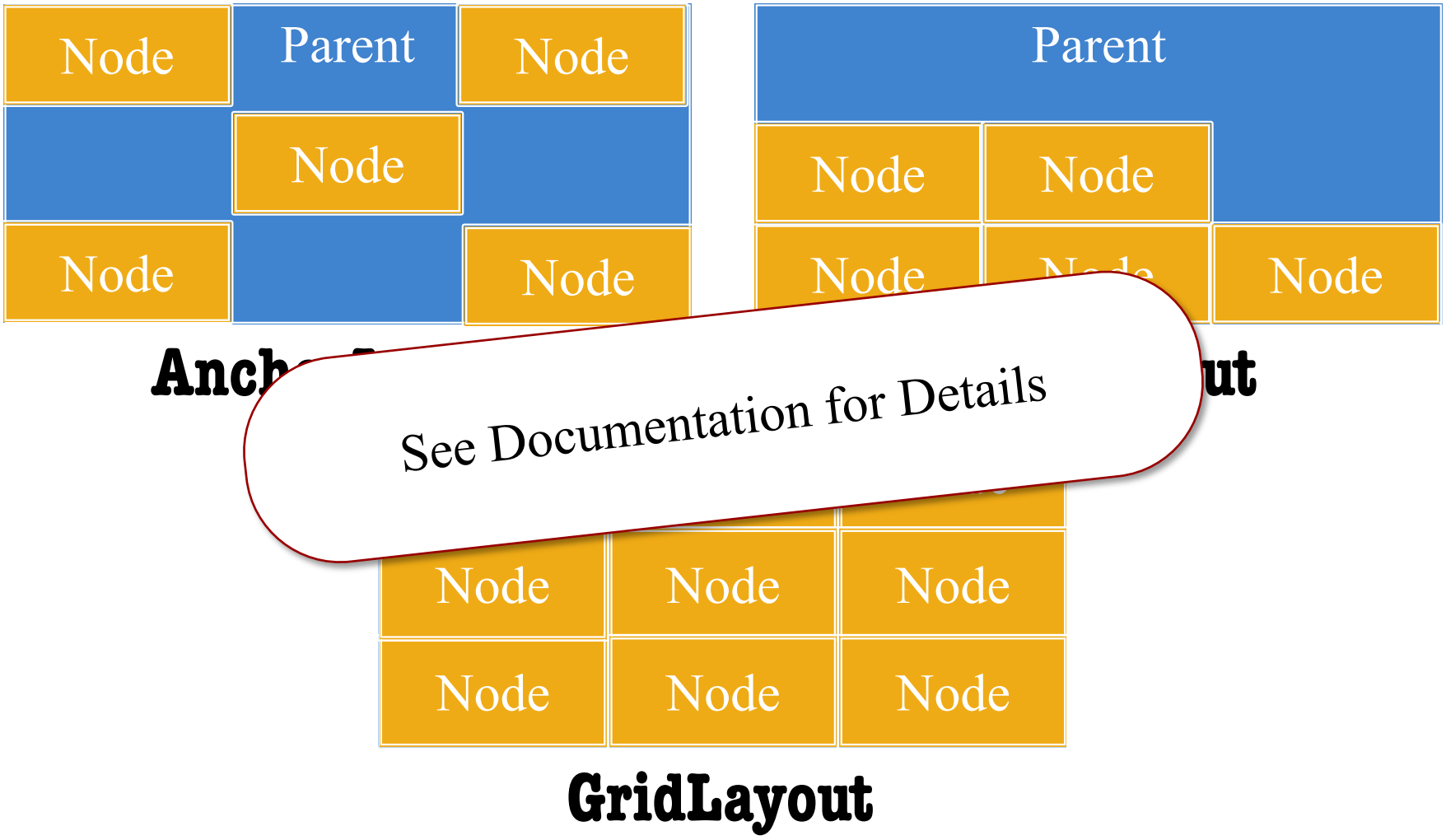



**FlowLayout**




**GridLayout**

# Layout Managers

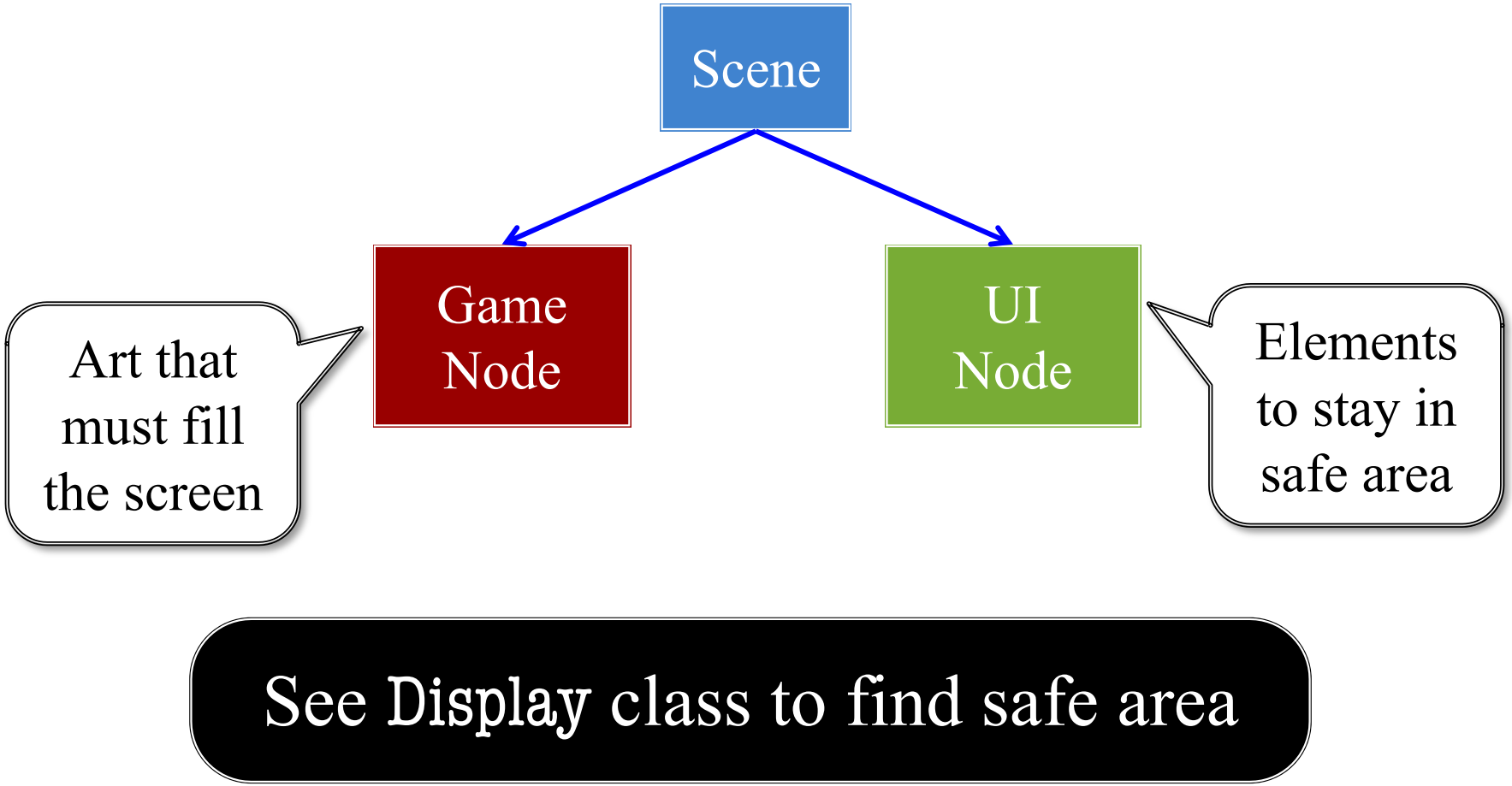


# How to Use a Layout Manager

1. Create a layout manager
2. Assign a relative position to each child
   - **Example**: middle left in an anchor layout
   - Layout manager maps strings to layout
   - Use the "name" string of the child node
3. Attach manager to the parent node
4. Call **`doLayout()`** on the parent

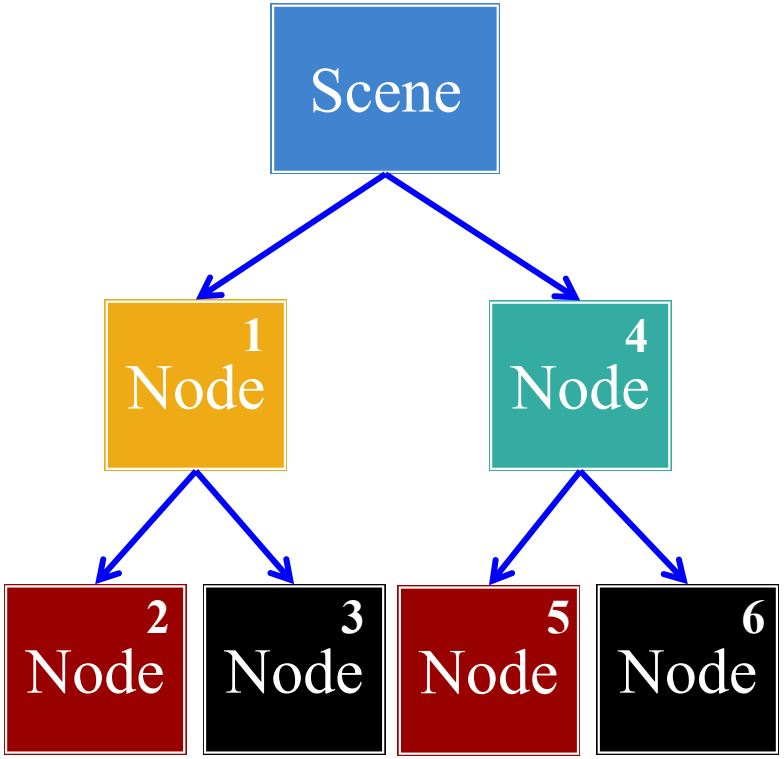
# Safe Area: Modern Phones

# Safe Area: Modern Phones



# Rendering a Scene is Easy

- **scene->render()**
  - Uses SpriteBatch to draw
  - Calls begin()/end() for you
  - Sets the SpriteBatch camera
  - Limits *in-between* drawing
- Uses a **preorder traversal**
  - Draws a parent node first
  - Draws children in order
  - Parent acts as background



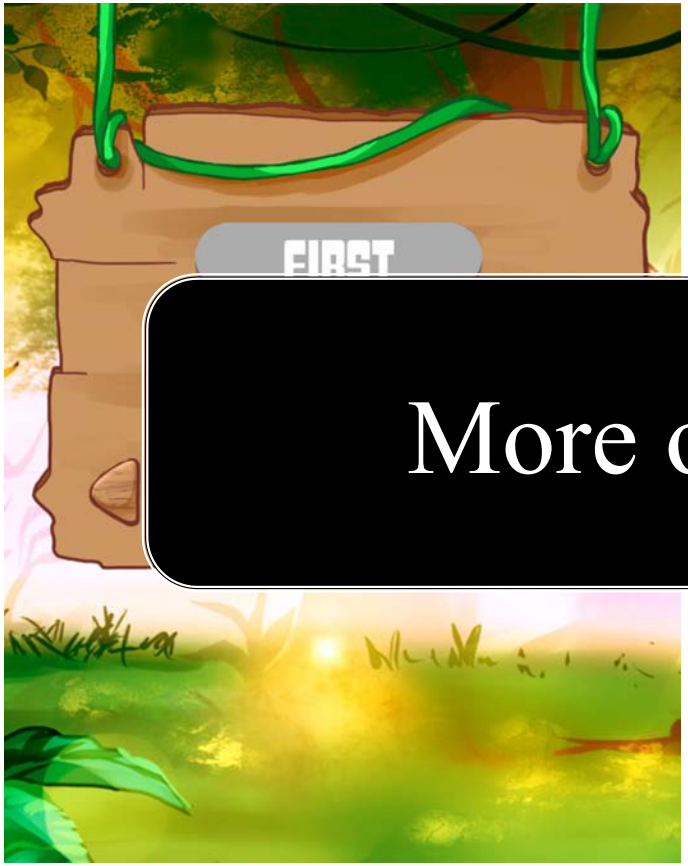
# Is Preorder Traversal Always Good?

## Good for UI Elements


## Bad For Animation

# Is Preorder Traversal Always Good?

## Good for UI Elements

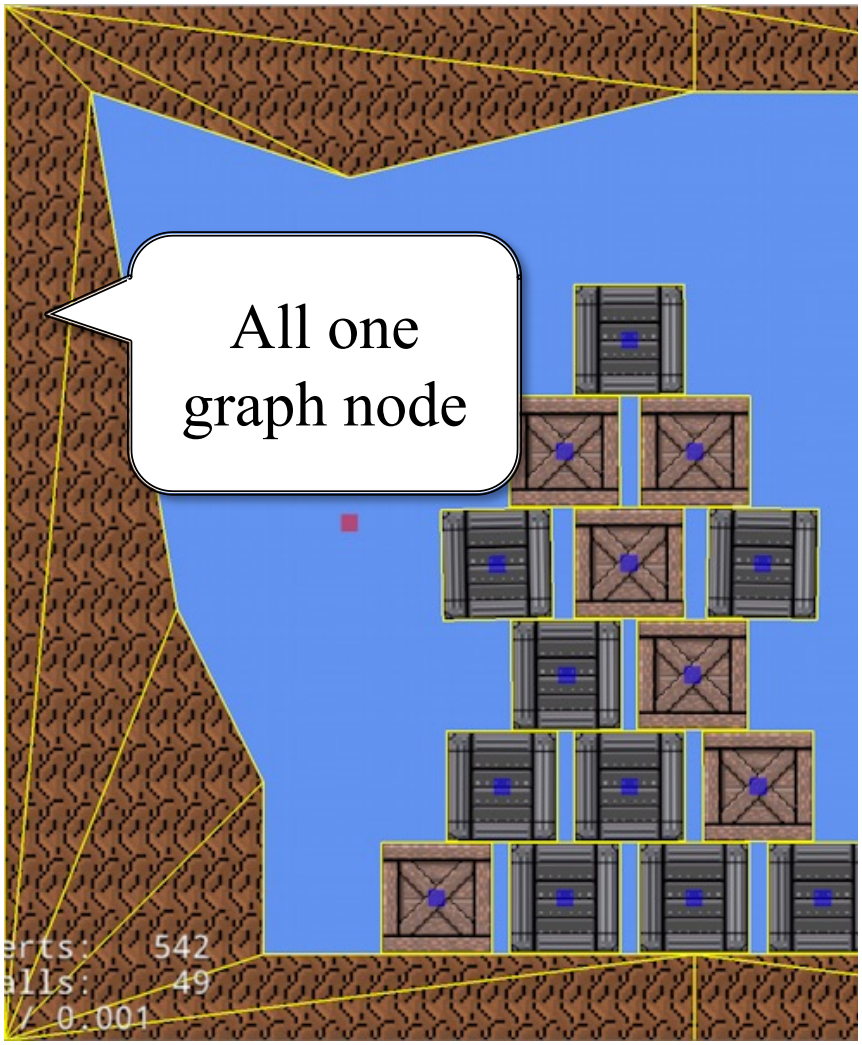



## Bad For Animation

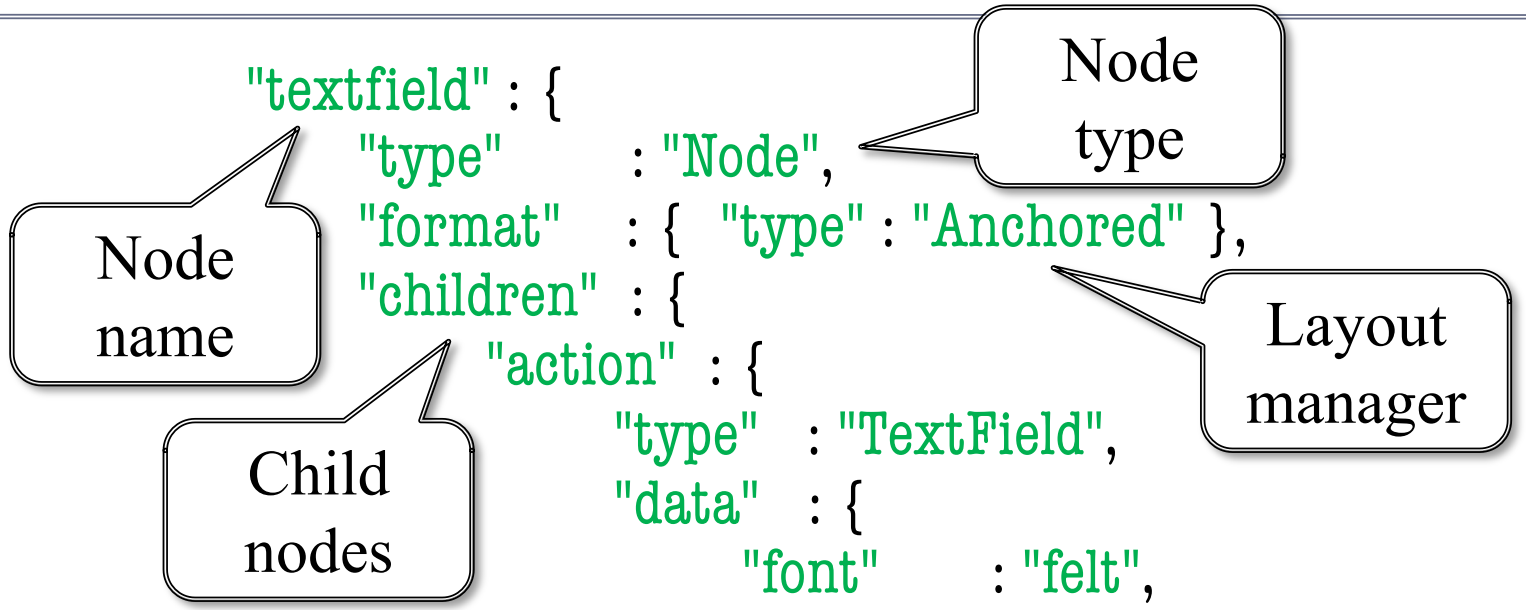
More on this later

# Specialized Nodes

- CUGL has many node types
  - `SpriteNode` (animation)
  - `WireNode` (wireframes)
  - `PolygonNode` (tiled shapes)
  - `PathNode` (lines with width)
  - `NinePatch` (UI elements)
  - `Label` (text)
- Learn them outside of class
  - Read the documentation
  - Play with the demos



# JSON Language for Scene Graphs



# JSON Language for Scene Graphs

```
"textfield" : {
    "type"      : "Node",
    "format"    : { "type" : "Anchored" },
    "children"  : {
        "action" : {
            "type"   : "TextField",
            "data"   : {
                "font"      : "felt",
                "text"      : "Edit me",
                "size"      : [600,80],
                "anchor"    : [0.5,0.5]
            },
            "layout" : {
                "x_anchor" : "center",
                "y_anchor" : "top"
            }
```

Layout manager

Node data

Info for parent layout

# JSON Language for Scene Graphs

```
"textfield" : {
    "type"      : "Node",
    "format"    : {  "type" : "Anchored" },
    "children"  : {
        "action"  : {
            "type"   : "TextField",
            "data"   : {
                "font"      : "felt",
                "text"      : "Edit me",
                "size"      : [600,80],
                "anchor"    : [0.5,0.5]
            },
            "layout" : {
                "x_anchor" : "center",
                "y_anchor" : "top"
            }
```

Each node has

- Type
- Format
- Data
- Children
- Layout

# Using JSON Scene Graphs

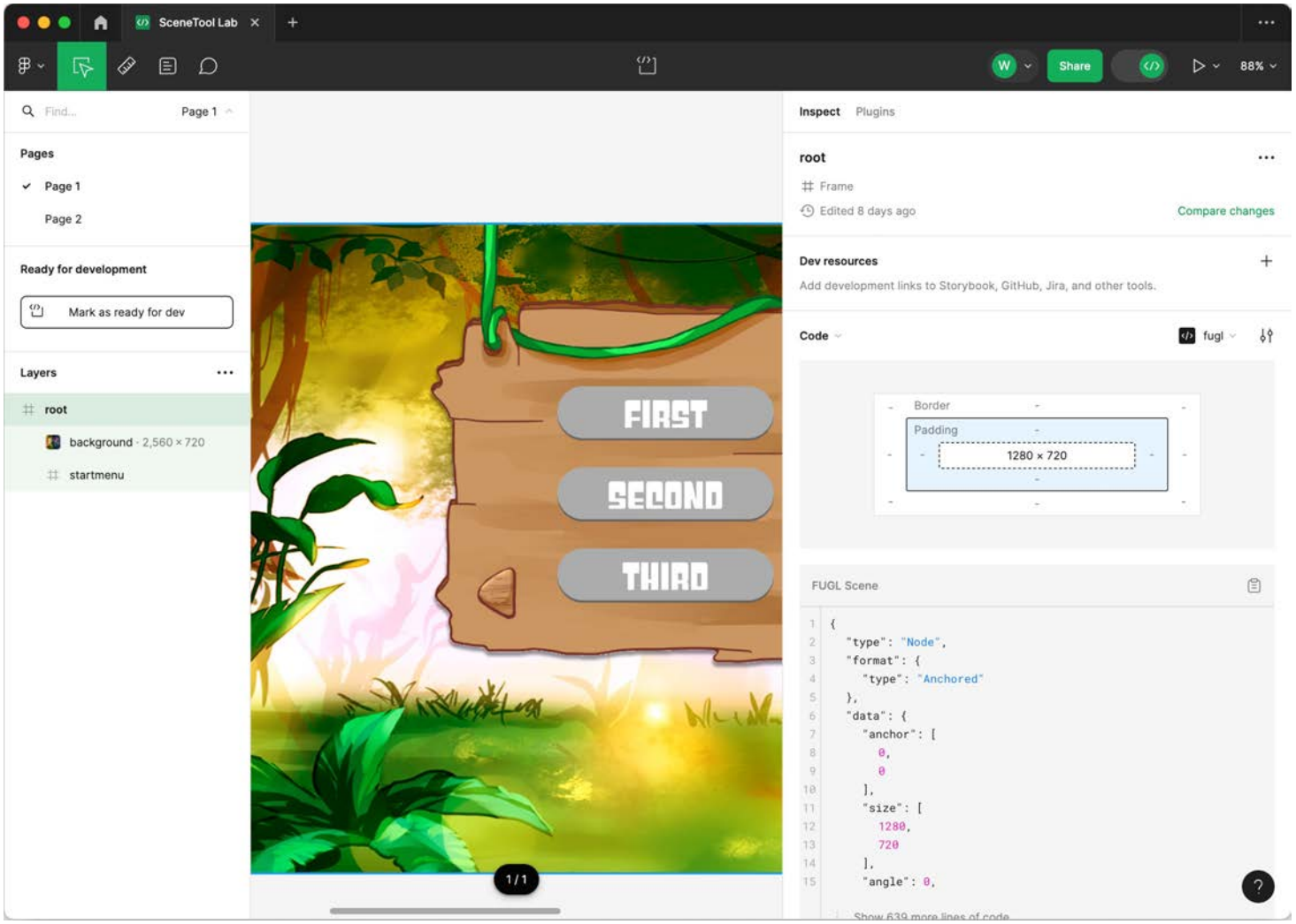
## Advantages

- Designers **do not need C++**
  - Using special tool in lab
  - Tool good for entire semester
- Format is **ideal for mobile**
  - Integrated layout managers
  - Aspect ratio support is easy
- **Integration is simple**
  - Load JSON with asset loader
  - Refer to scene root by name

## Disadvantages

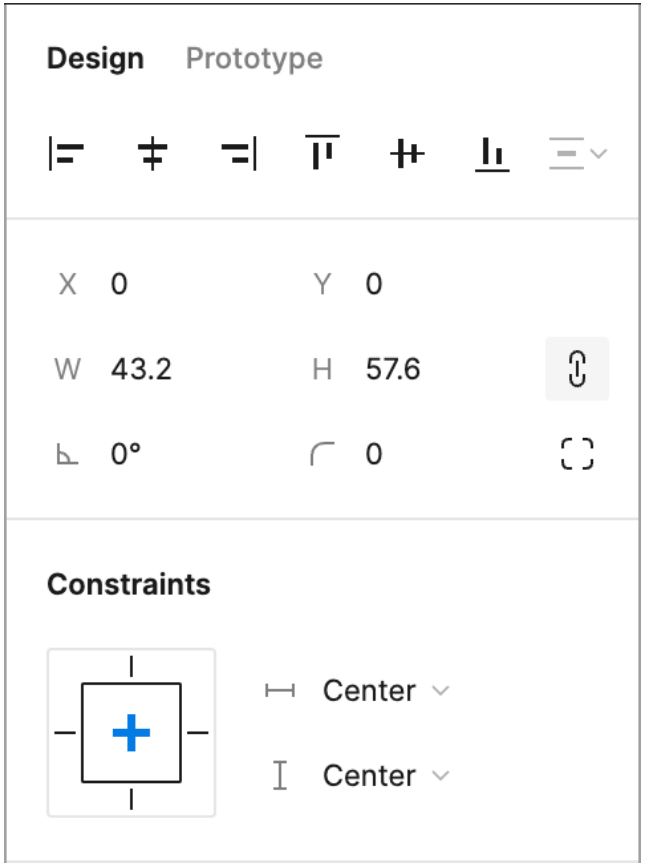
- UI still needs **custom code**
  - Buttons etc. do nothing
  - Essentially need listeners
  - Programmers do manually
- Files can be very **confusing**
  - Format is a tree structure
  - Each tree node is verbose
- **Not a level editor format!**
  - Levels need more info

# Solution: The Figma Plugin

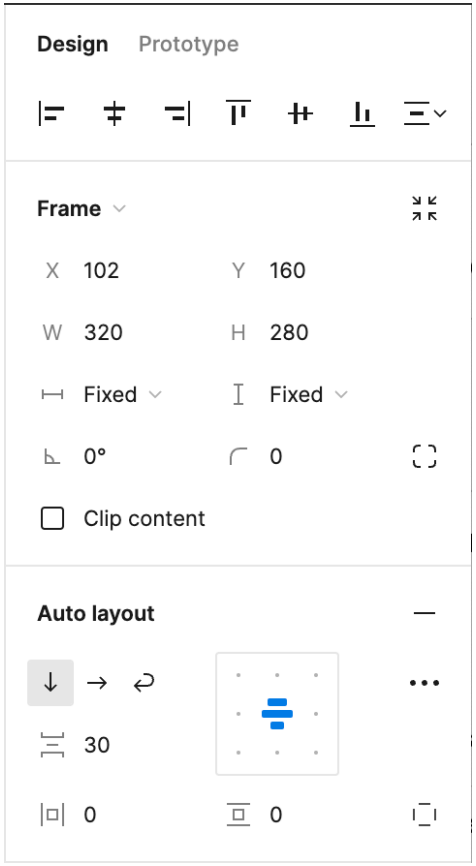


# Plugin Manages Anchors **and** Layouts

## AnchorLayout

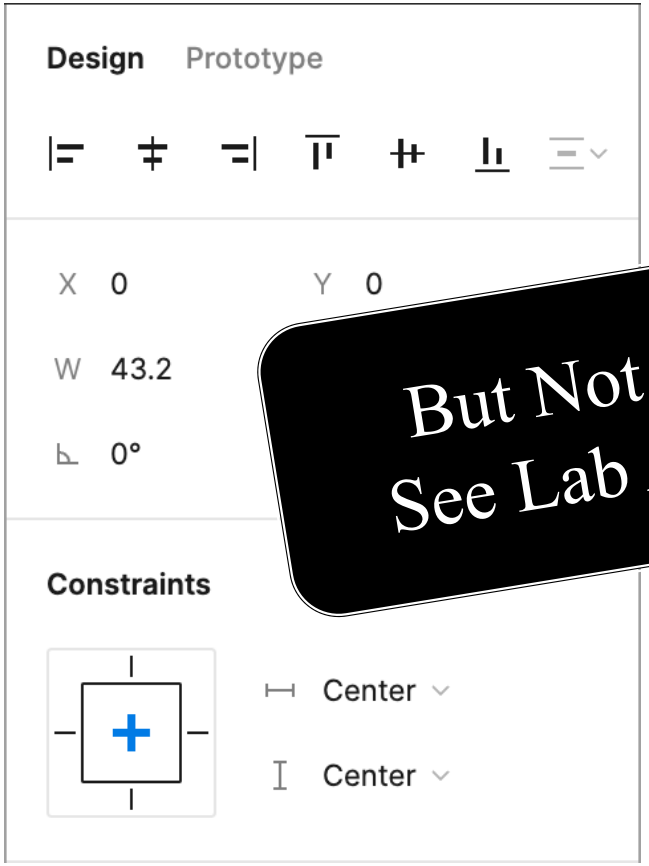



## FloatLayout

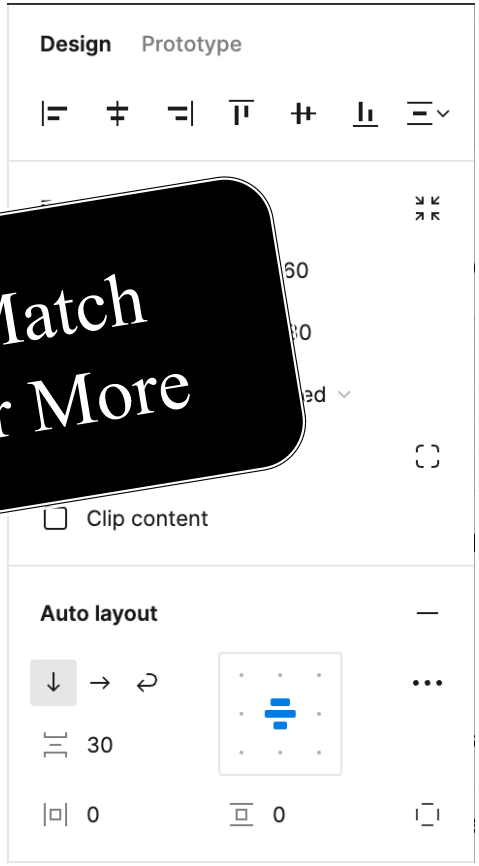


# Plugin Manages Anchors **and** Layouts

## AnchorLayout




## FloatLayout




But Not an Exact Match
See Lab Activity for More

# Widgets: JSON Templates

## Widget

```
"variables" : {
   "image" : ["children","up","data","texture"]
},
"contents"  : {
   "type"   : "Button",
   "data"   : {
      "upnode"   : "up", "visible"  : false,
      "anchor"   : [0.5,0.5], "scale"    : 0.8
   },
   "children" : {
      "up"       : {
         "type"   : "Image",
         "data"   : { "texture"  : "play"}
   }}}
```

Widget is a subtree

## JSON

```
"widgets": {
   "mybutton" : "widgets/mybutton.json",
},
"scene2s": {
   "thescene" : {
   "type"     : "Node",
   "format" : { "type" : "An
   "children"  : {
      "button"  : {
         "type"   : "Widget",
         "data"   : {
            "key" : "mybutton",
            "variables" : { "image":"altplay" }
         },
         "layout" : { "x_anchor" : "center" }
   }}}
```

Replace w/ subtree

# Widgets: JSON Templates

## Widget

```
"variables" : {
   "image" : ["children","up","data","texture"]
},
"contents"  : {
   "type"   : "Bu
   "data"   : {
      "upnode"   : "up", "visible"  : false,
      "anchor"   : [0.5,0.5], "scale"    : 0.8
   },
   "children" : {
      "up"      : {
         "type"   : "Image",
         "data"   : { "texture"  : "play"}
   }}}
```

Full path to value to change

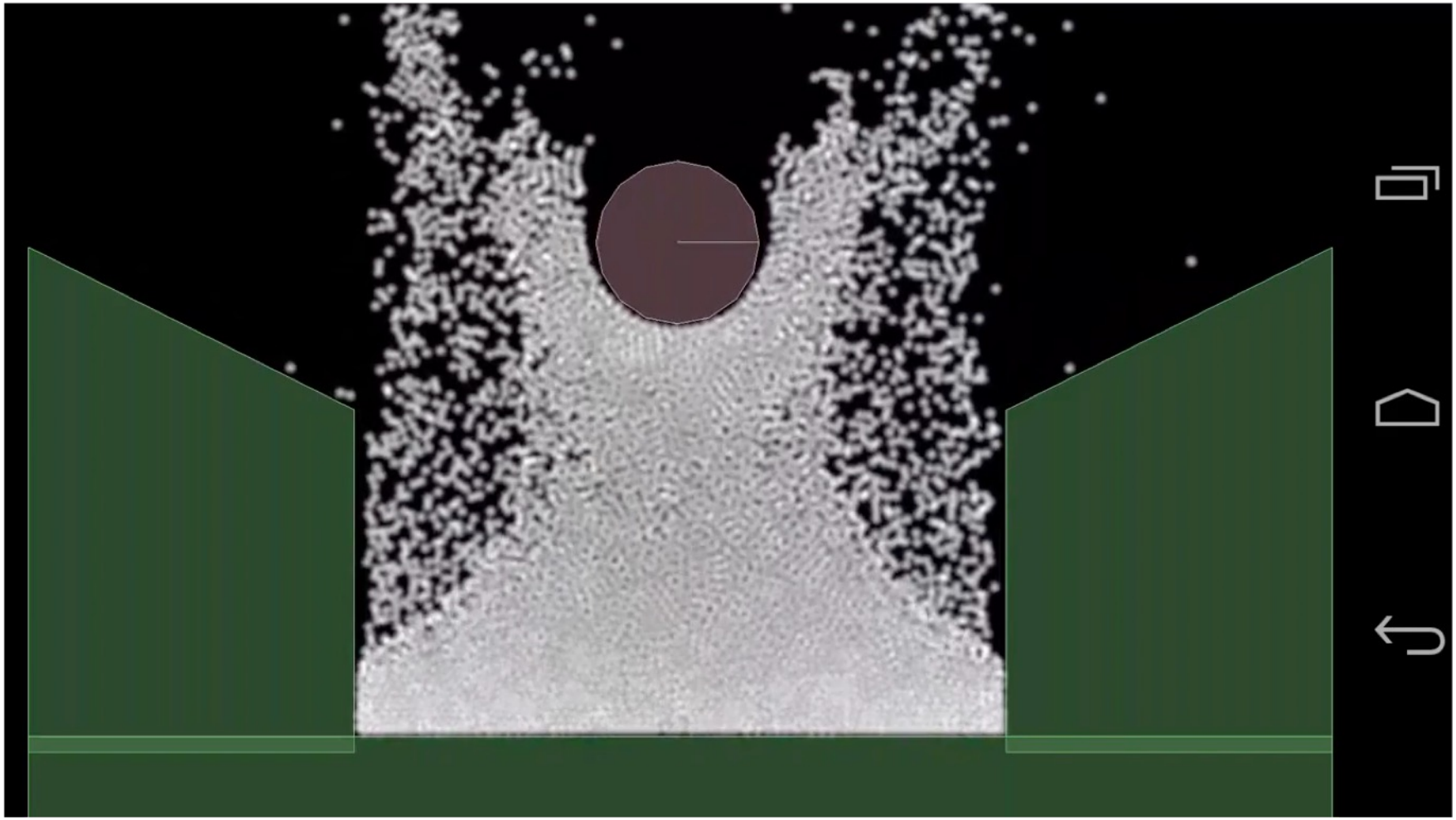
Provide the layout

## JSON

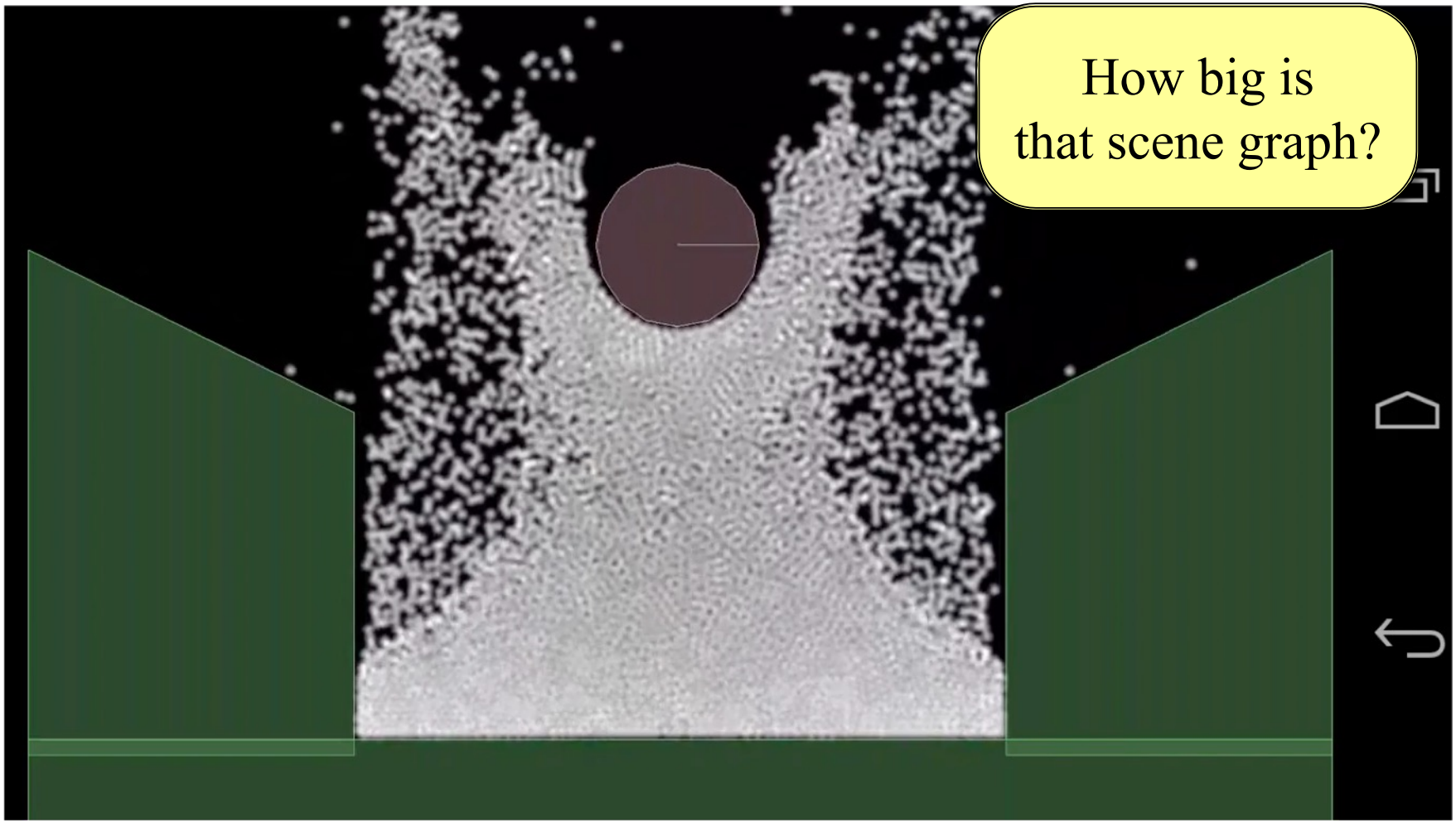
```
"widgets": {
   "mybutton" : "widgets/mybutton.json",
},
"scene2s": {
   "thescene" : {
   "type"     : "Node",
   "format" : { "type" : "Anchored" },
   "children"  : {
      "button"  : {
         "type"   : "W
         "data"   : {
            "key" : "mybutton",
            "variables" : { "image":"altplay" }
         },
         "layout" : { "x_anchor" : "center" }
      }}}
```

Change the variable

# One Last Problem: **Physics**



# One Last Problem: **Physics**



# Defining Custom Nodes

## draw()

- Overridden to render node
  - Only node, not children
  - The `render` method (do not touch) handles children
- Drawing data is **cached**
  - The vertex positions
  - The vertex colors
  - The texture coordinates
- Cache passed to `SpriteBatch`

## generateRenderData()

- Overridden to update cache
  - Change vertex positions
  - Change vertex colors
  - Change texture coordinates
- Only needed for **reshaping**
  - Transforms for movement
  - Called infrequently
- Optimizes the render pass

# The PolygonNode **draw()** Method

```
void draw(const std::shared_ptr<SpriteBatch>& batch,
          const Affine2& transform, Color4 tint) {

    if ( !_rendered ) {
        generateRenderData();
    }

    batch->setColor(tint);

    batch->setBlendState( _blend );
    batch->drawMesh( _mesh, transform );
}
```

# The PolygonNode **draw()** Method

```
void draw(const std::shared_ptr<SpriteBatch>& batch,
          const Affine2& transform, Color4 tint) {

    if ( !_rendered ) {
        generateRende
    }

    batch->setColor(tint);

    batch->setBlendState( _blend );
    batch->drawMesh( _mesh, transform );
}
```

Computed from parent (+camera)

Computed from parent (+scene)

The Render Data

# Summary

- CUGL tries to leverage ideas from 3152
  - Top level class works like the classic GDXRoot
  - Design architecture to switch between modes
  - Use `SpriteBatch` class to draw textures in 2D.
- New idea is using **scene graphs** to draw
  - Tree of nodes with relative coordinate systems
  - Makes touch input easier to process
  - Also helps with animation (later)
- JSON language makes design easier