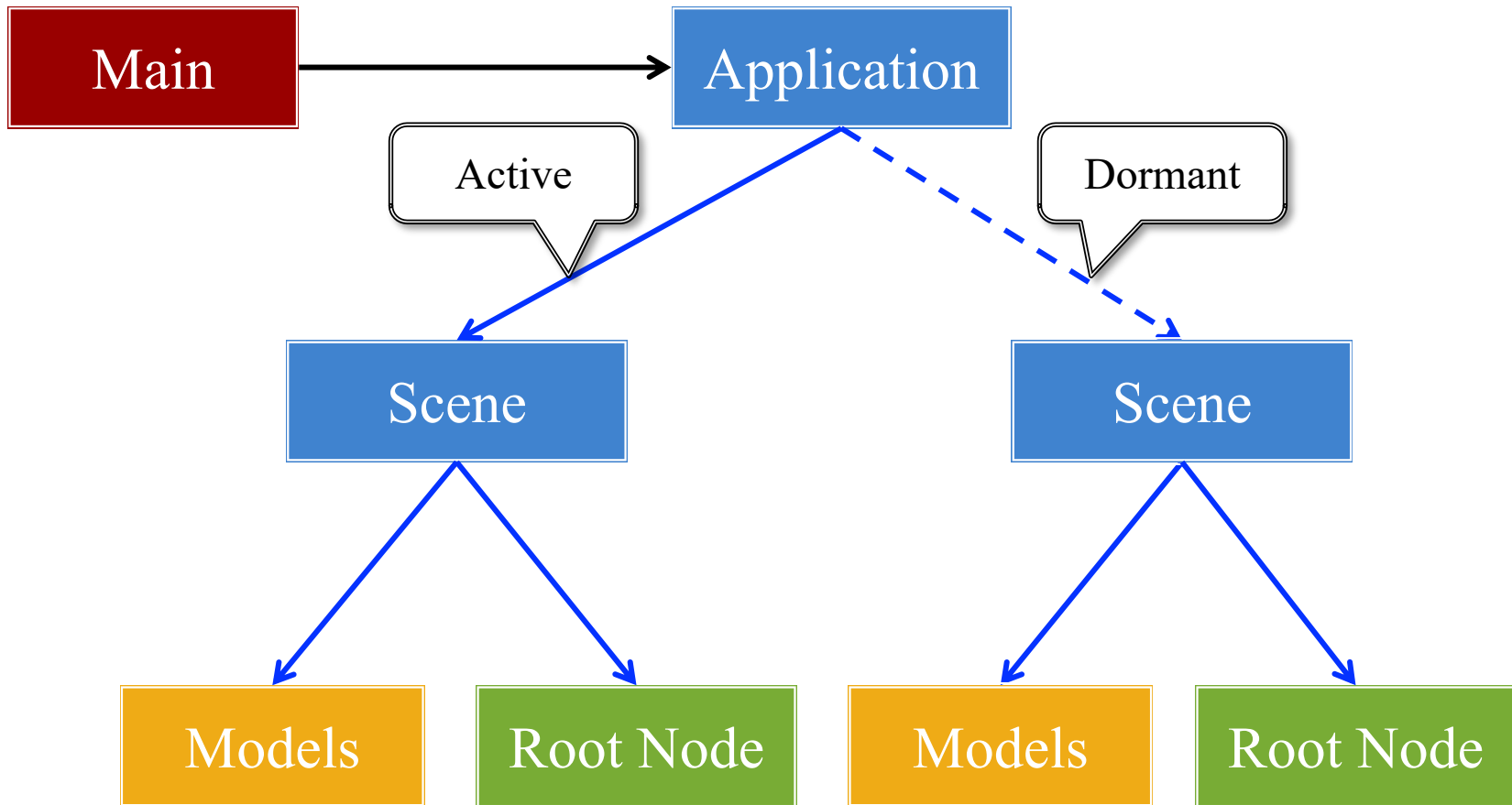


Lecture 7

Scene Graphs

Recall: Structure of a CUGL Application



Recall: The Application Class

onStartup()

- Handles the game assets
 - Attaches the asset loaders
 - Loads immediate assets
- Starts any global singletons
 - **Example:** AudioChannels
- Creates any player modes
 - But does not launch *yet*
 - Waits for assets to load
 - Like [GDXRoot](#) in 3152

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

Recall: The Application Class

onStartup()

- Handles the game assets
 - Attaches the asset loaders
 - Loads immediate assets
- Sets up the scene graph
 - Sets up the scene graph
- Cleans up
 - Cleans up
- Checks for any player modes
 - But does not launch *yet*
 - Waits for assets to load
 - Like `GDXRoot` in 3152

onShutdown()
cleans this up

update()

- Called each animation frame
- Manages gameplay
 - Converts input to actions
 - Updates the scene graph
 - Observes other interactions
- Updates the scene graph
 - Transforms nodes
 - Enables/disables nodes

Does not draw!
Handled separately

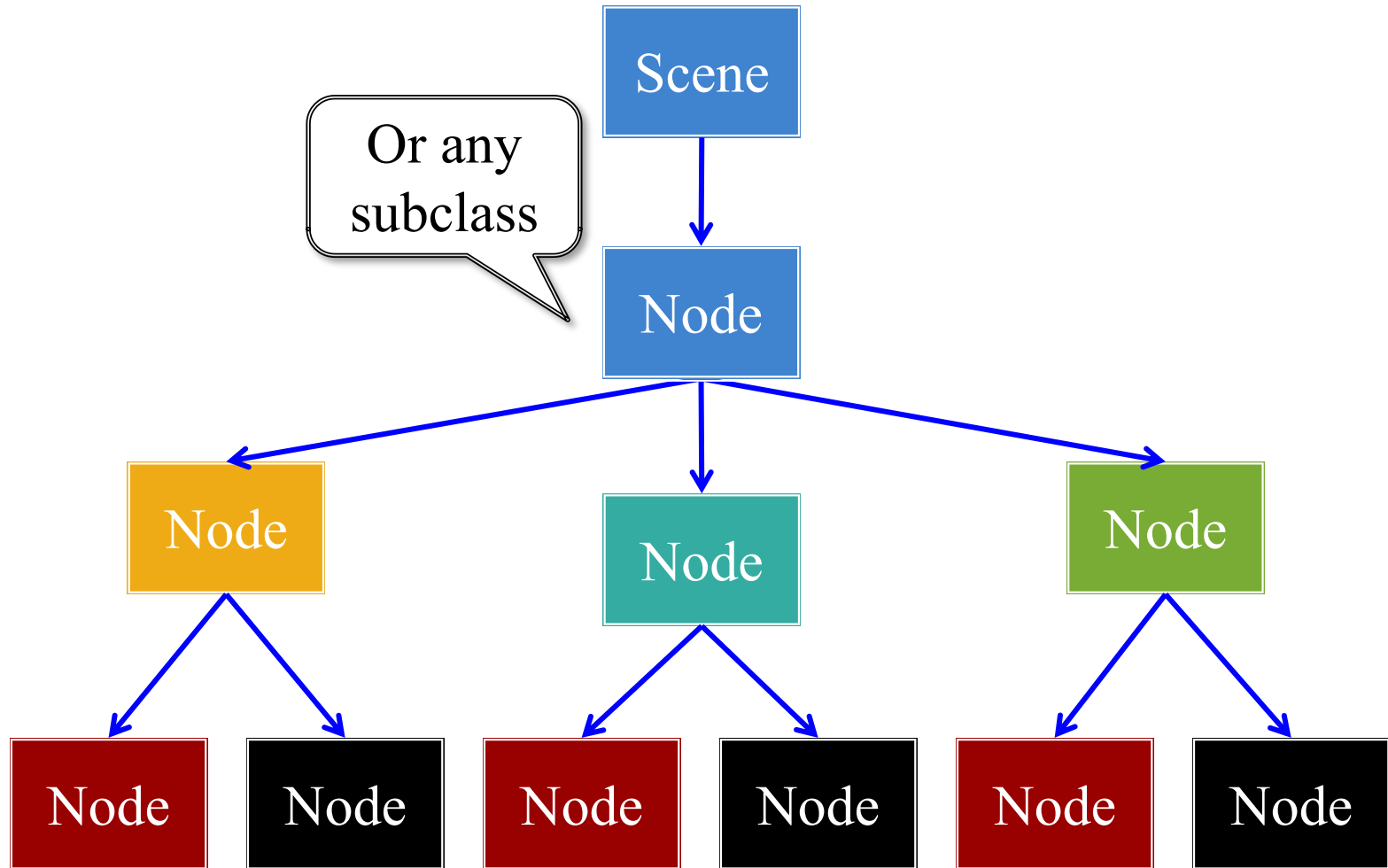
Drawing in CUGL

- Use the `draw()` method
 - Called after `update()`
 - Clears screen first
 - Uses clear color field
- Can use any OpenGL
 - Included in `CUBase.h`
 - Best to use OpenGL ES (subset of OpenGL)
- Or use a `SpriteBatch`
 - *Mostly* like in 3152

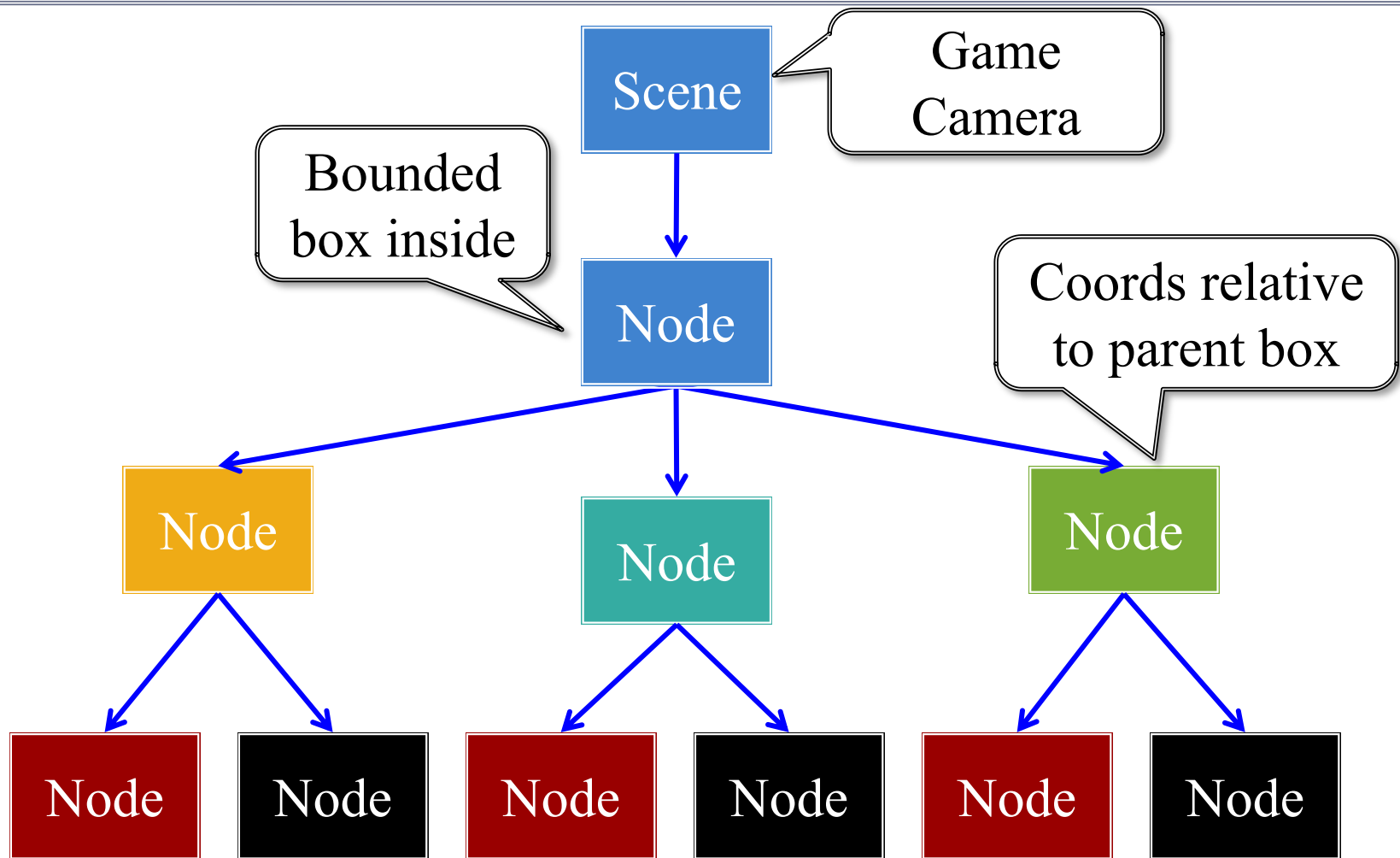
```
void draw() {  
    glEnableVertexAttribArray(0);  
    glBindBuffer(GL_ARRAY_BUFFER,  
                vertexbuffer);  
    glVertexAttribPointer(0, 3, GL_FLOAT,  
                          GL_FALSE, 0, (void*)0 );  
    glDrawArrays(GL_TRIANGLES, 0, 3);  
    glDisableVertexAttribArray(0);  
}
```

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

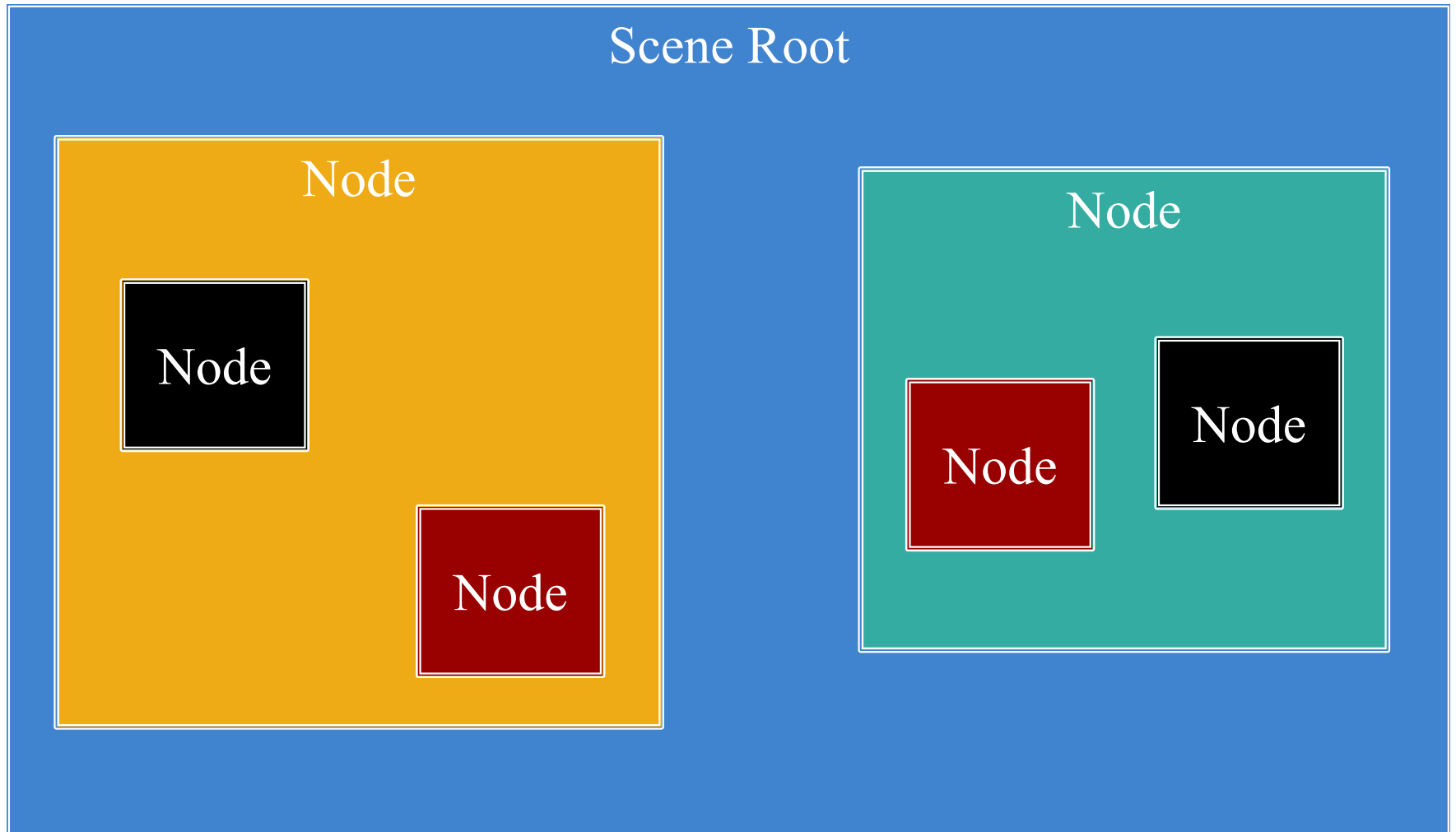
The Scene Graph



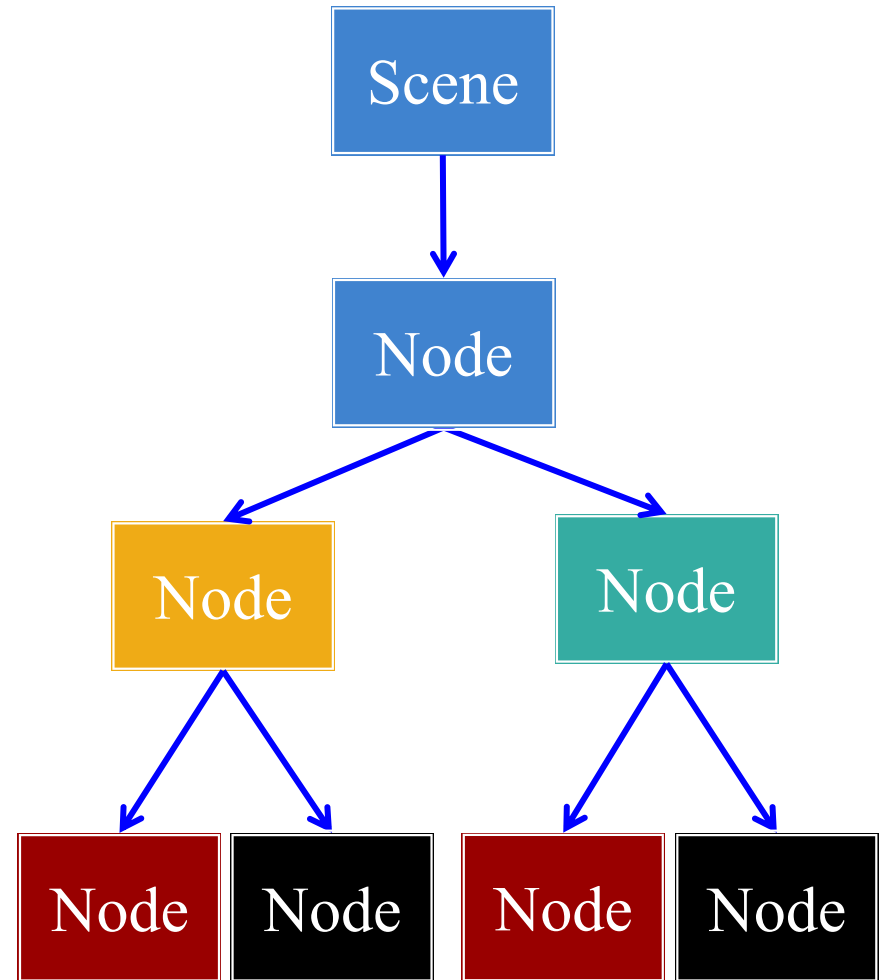
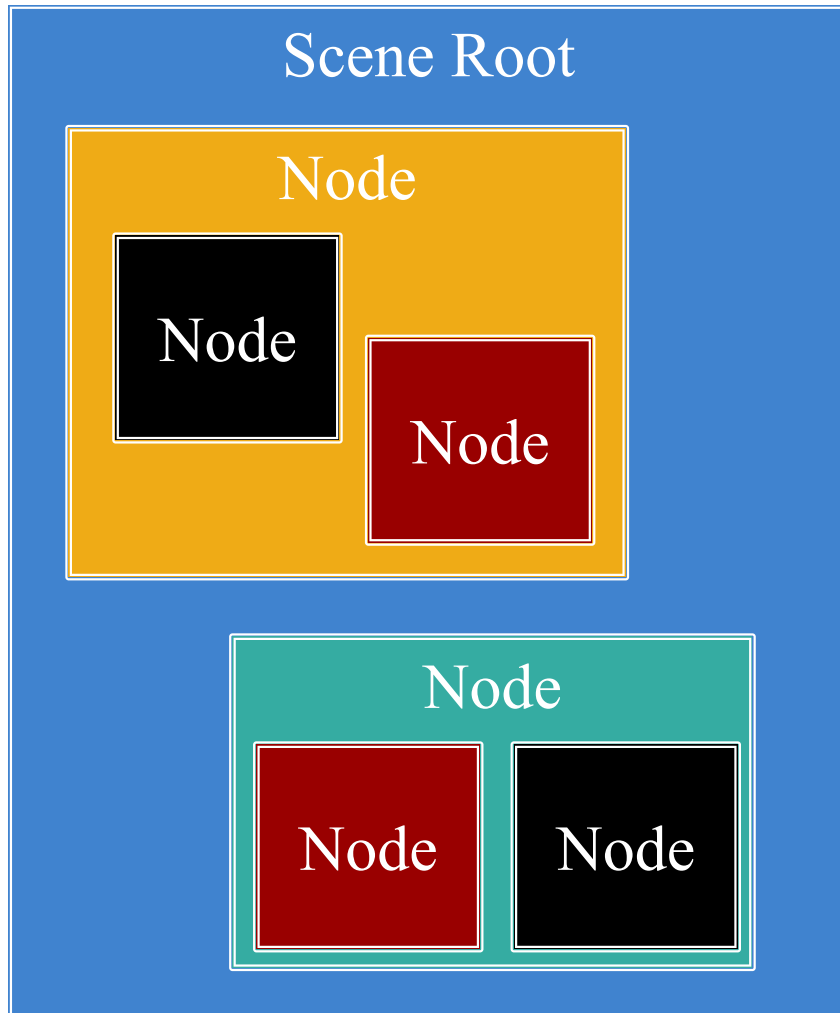
The Scene Graph



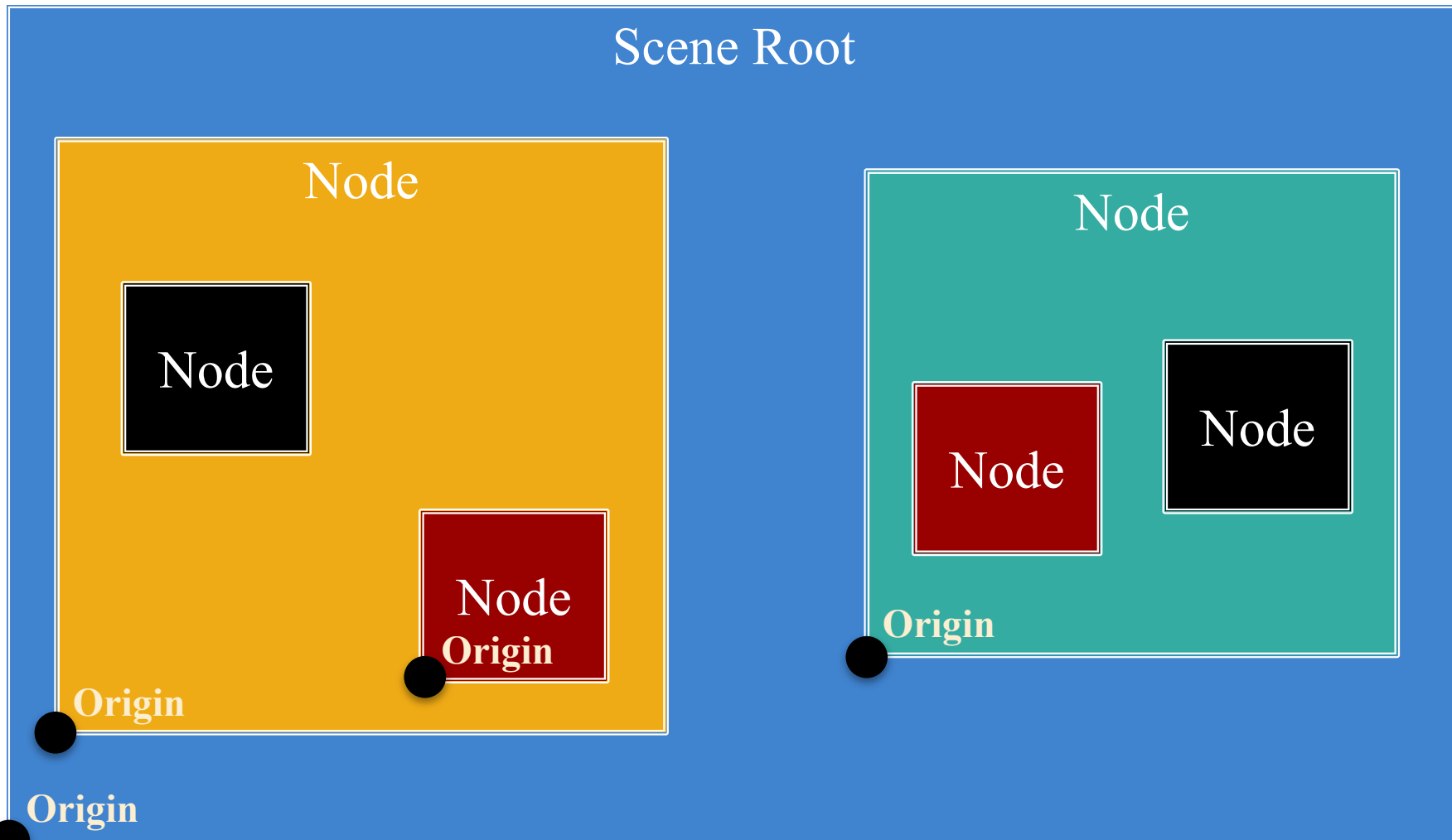
Each Node is a Coordinate System



Each Node is a Coordinate System

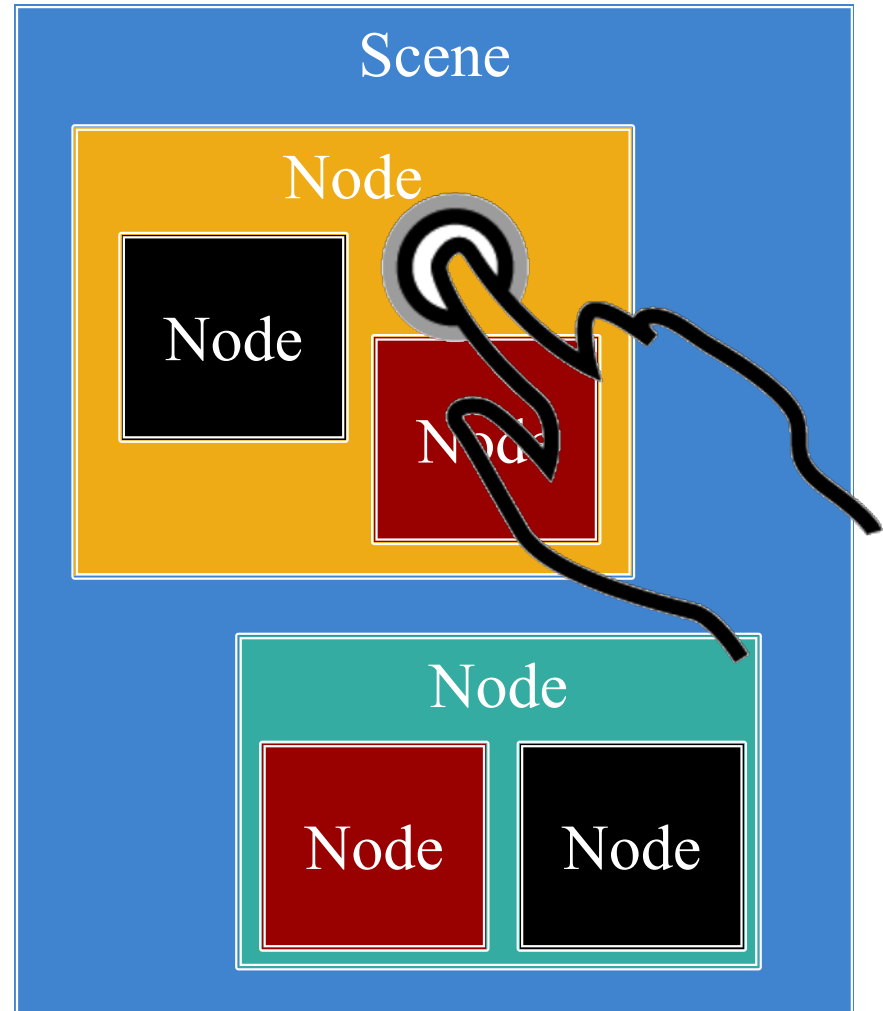


Each Node is a Coordinate System

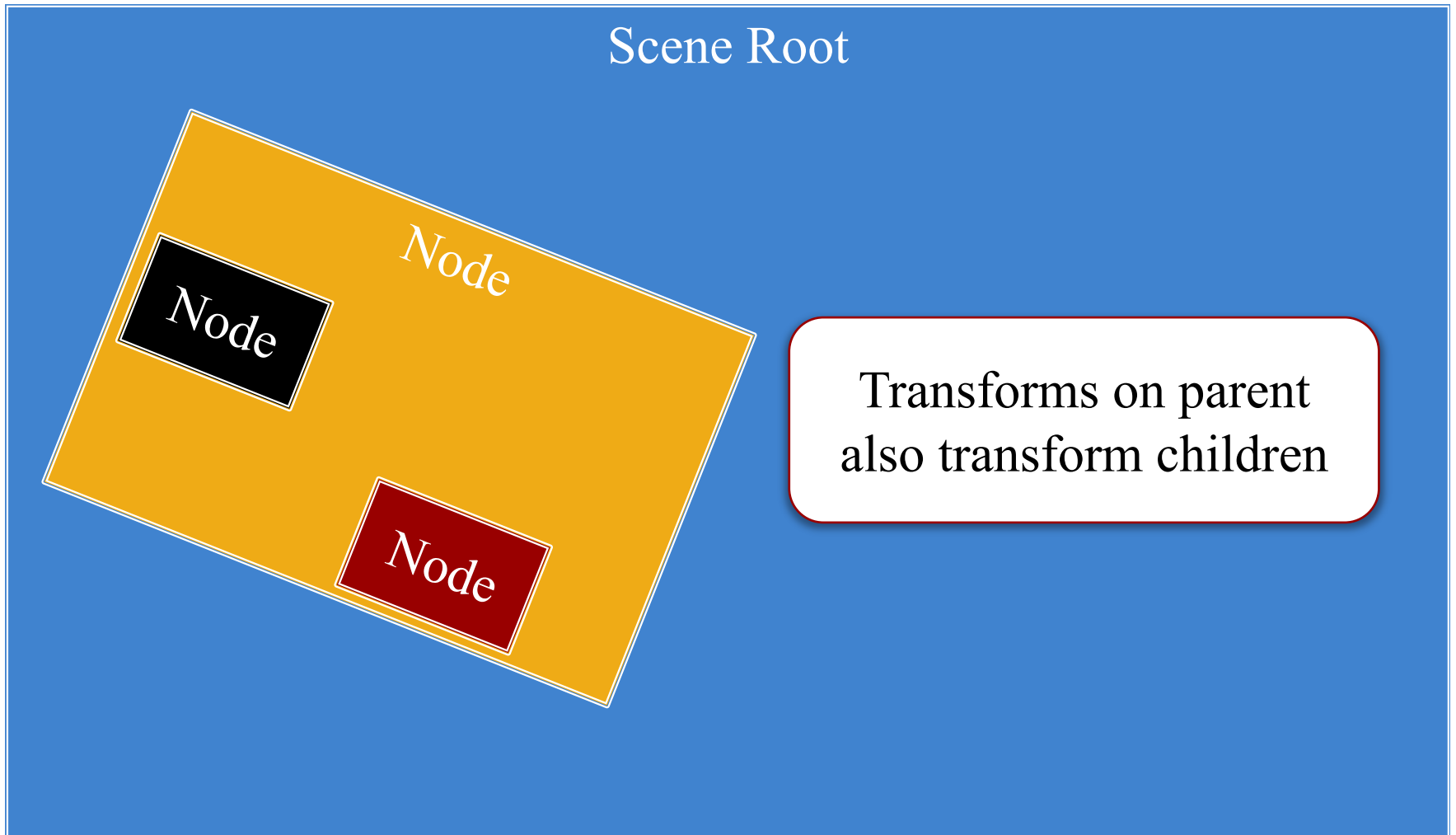


Motivation: Touch Interfaces

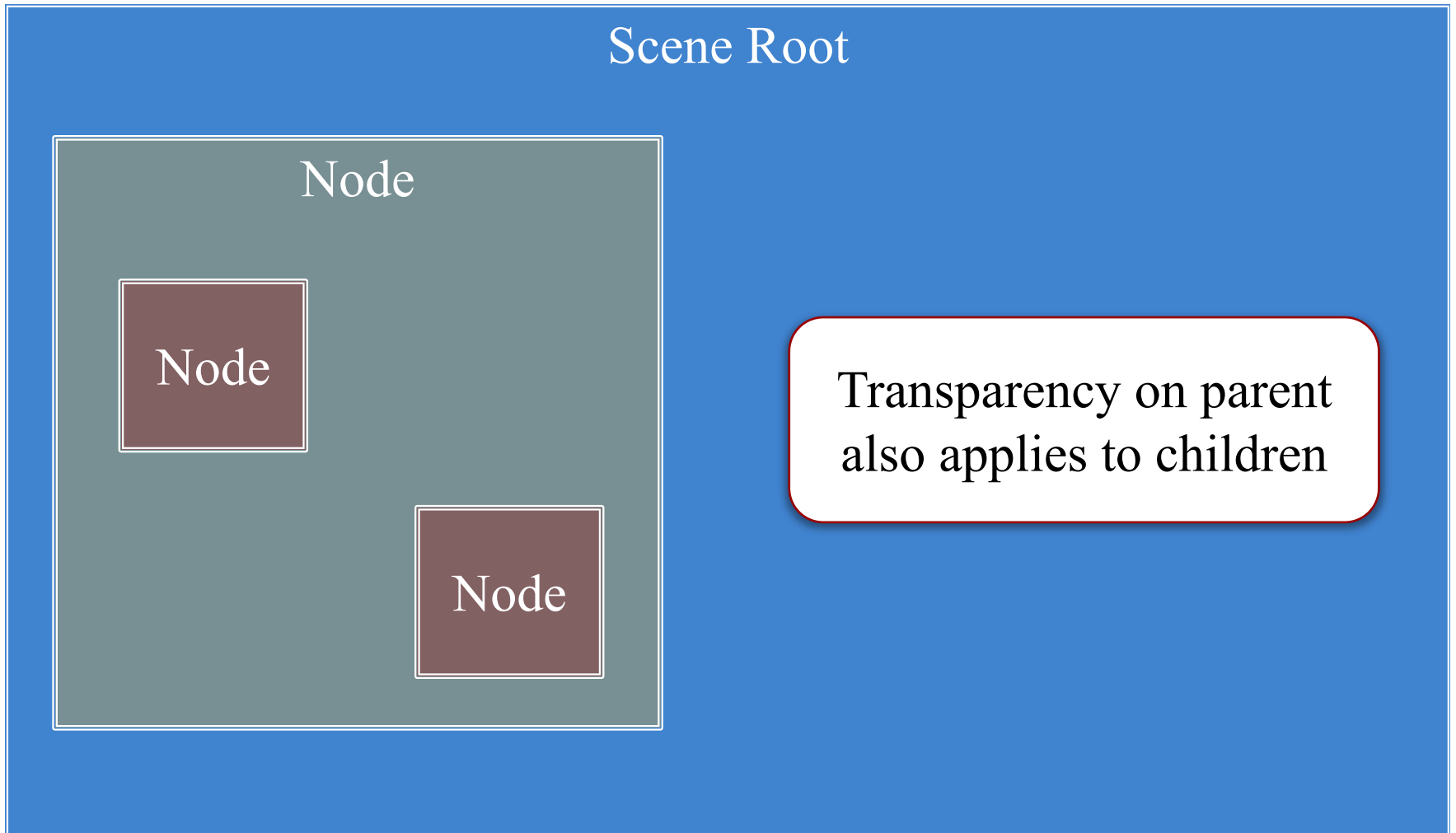
- Touch handler requires
 - Which object touched
 - Location inside object
- Scene graph is a *search tree*
 - Check if touch is in parent
 - ... then check each child
 - Faster than linear search
- But limit this to a **search**
 - No input control in node
 - Use polling over callbacks



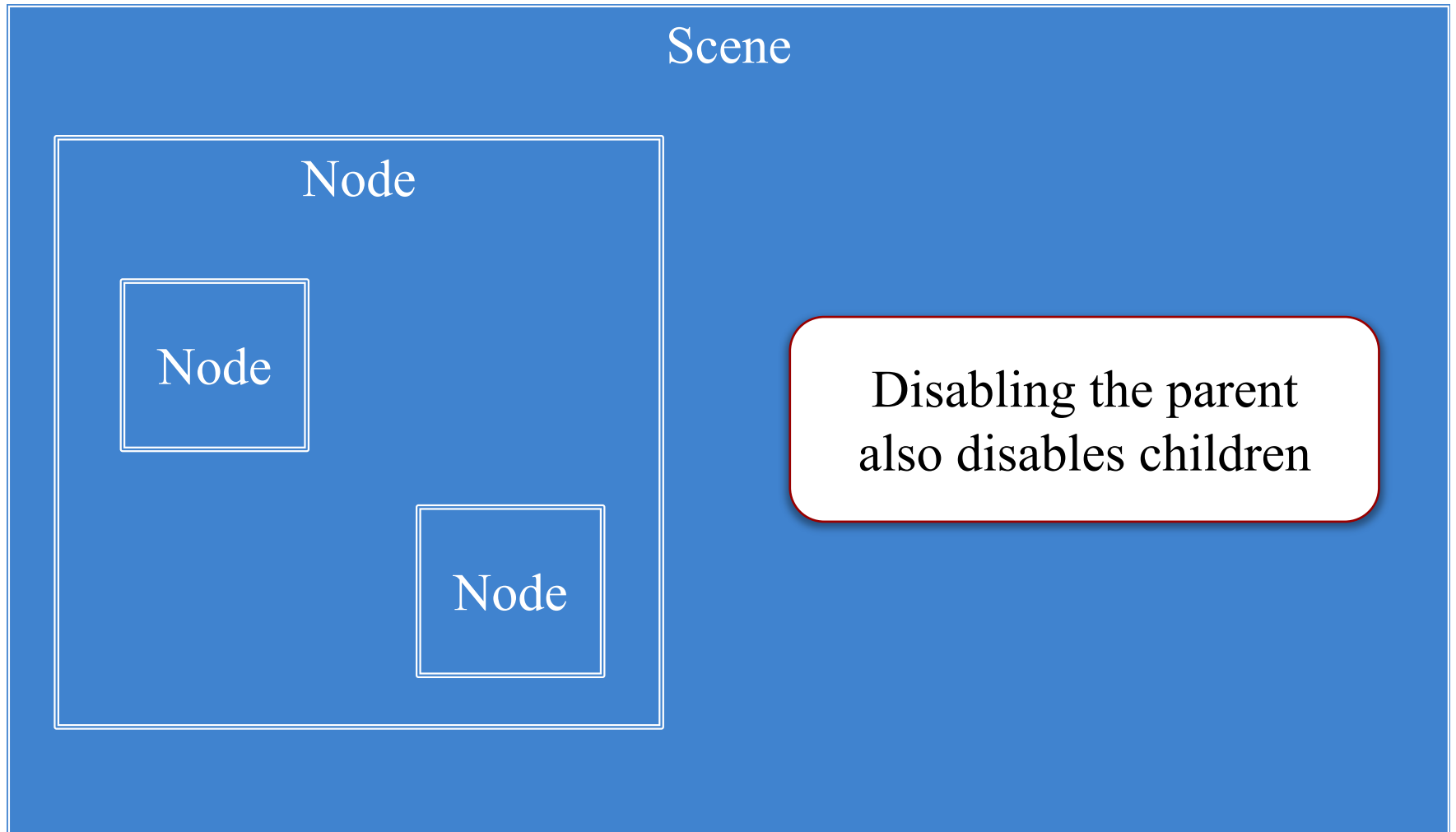
Settings Pass Down the Graph



Settings Pass Down the Graph

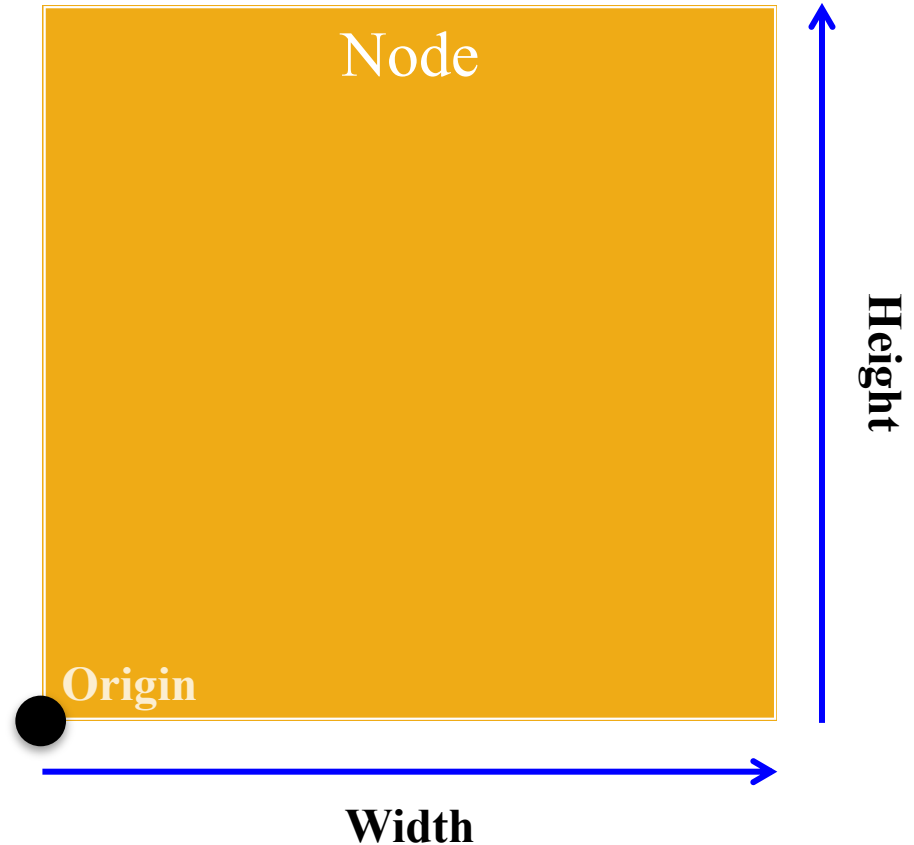


Settings Pass Down the Graph



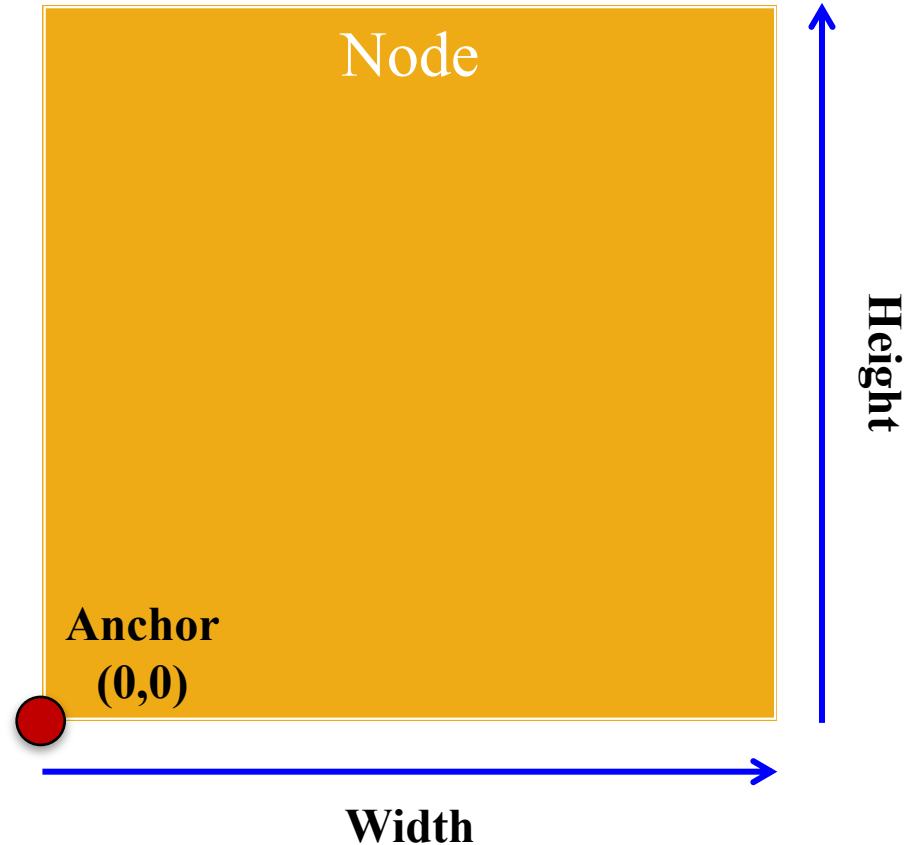
Anchors and Content

- 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**



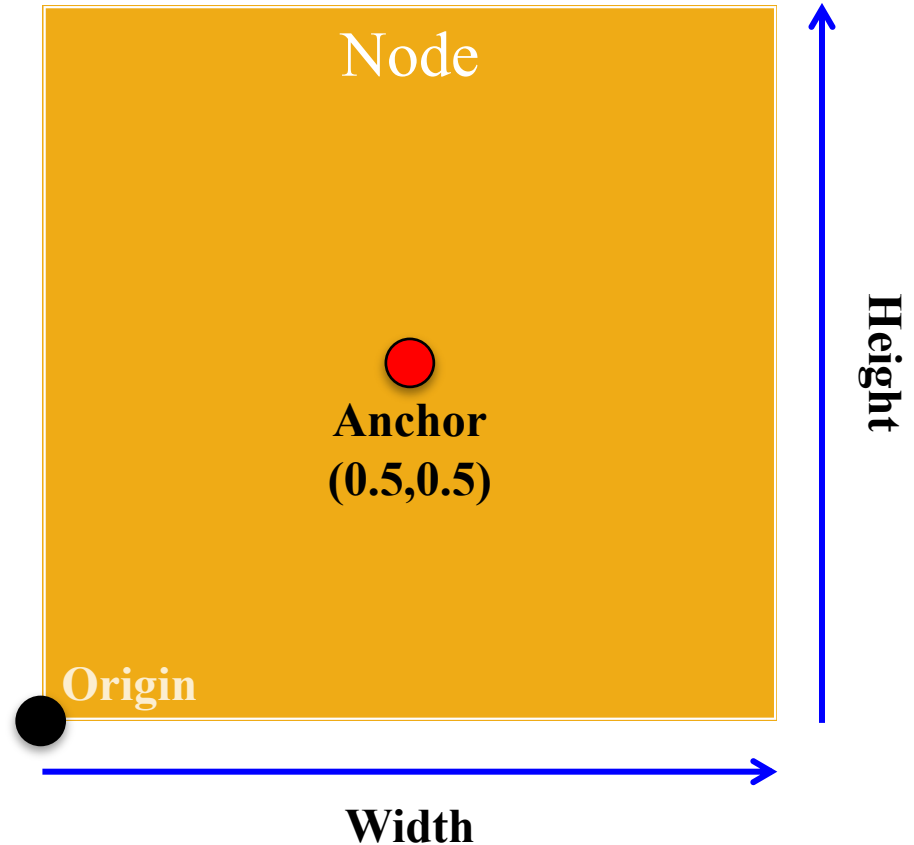
Anchors and Content

- 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**



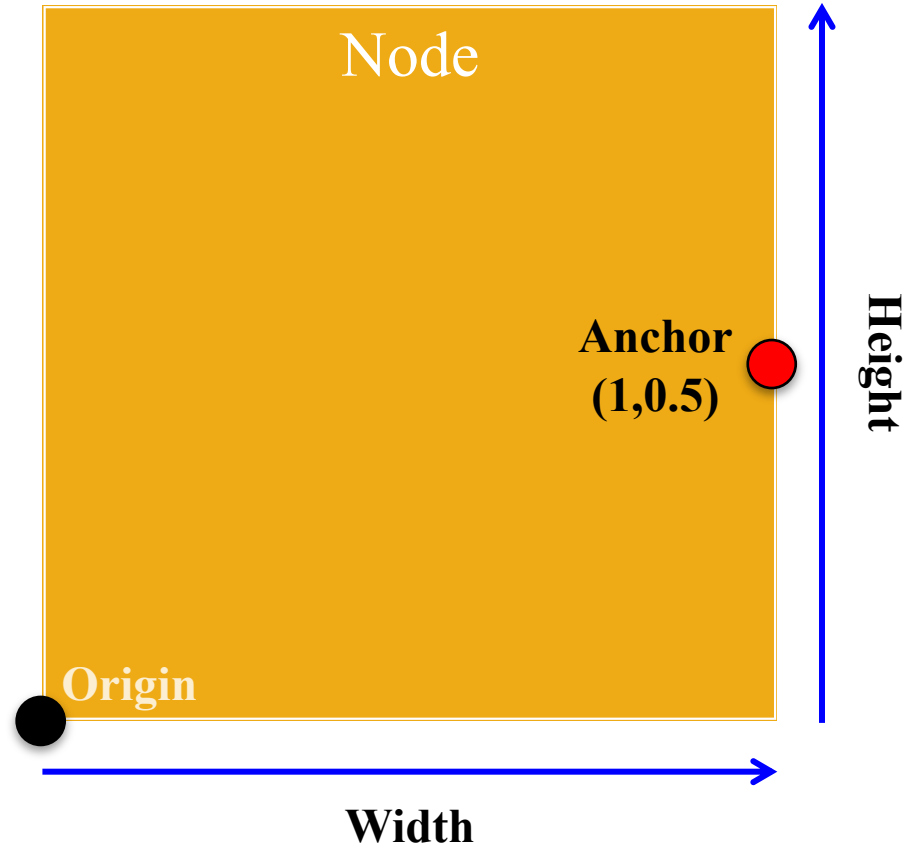
Anchors and Content

- 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**

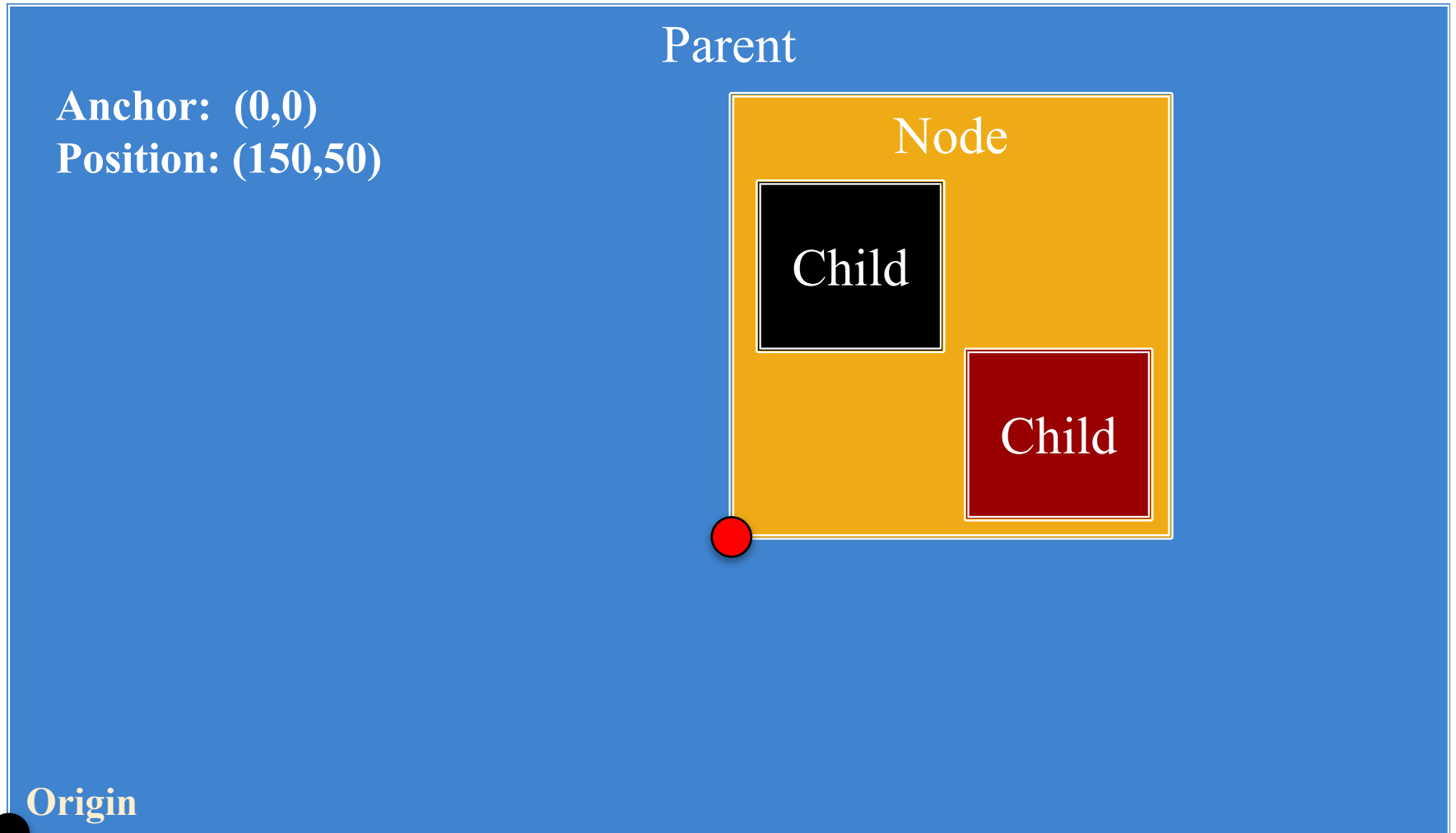


Anchors and Content

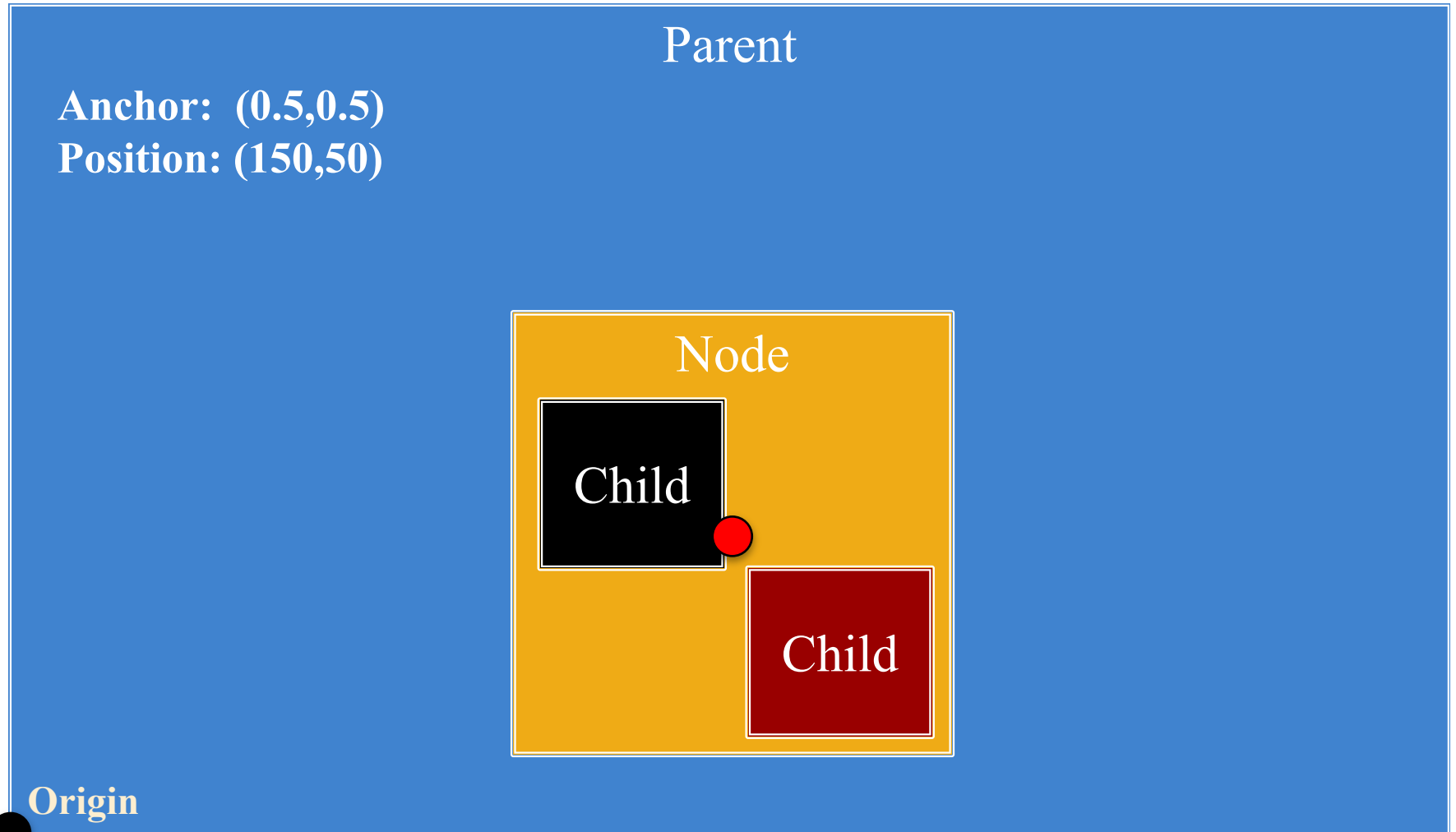
- Nodes have **content size**
 - Width/height of contents
 - Measured in node space
 - But only a guideline: content can be outside
- Nodes have an **anchor**
 - Location in node space
 - *Percentage* of width/height
 - Does not affect the origin
- Both may affect **position**



Anchor and Position

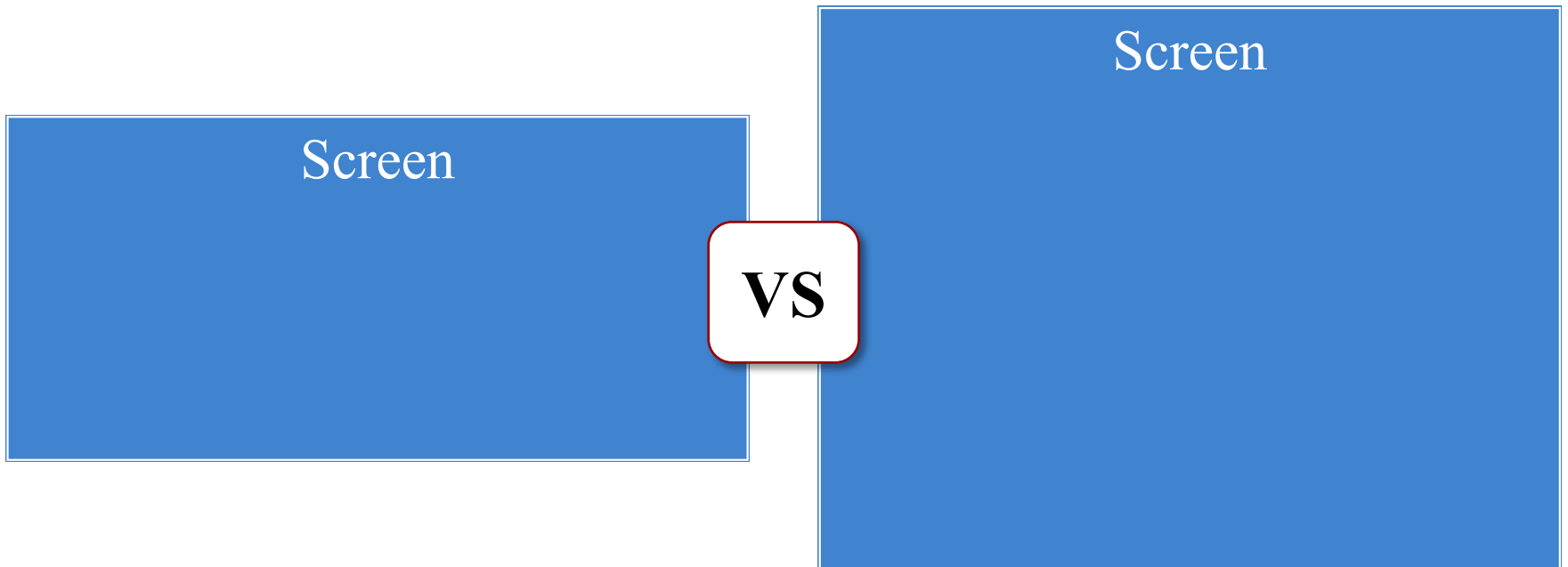


Anchor and Position



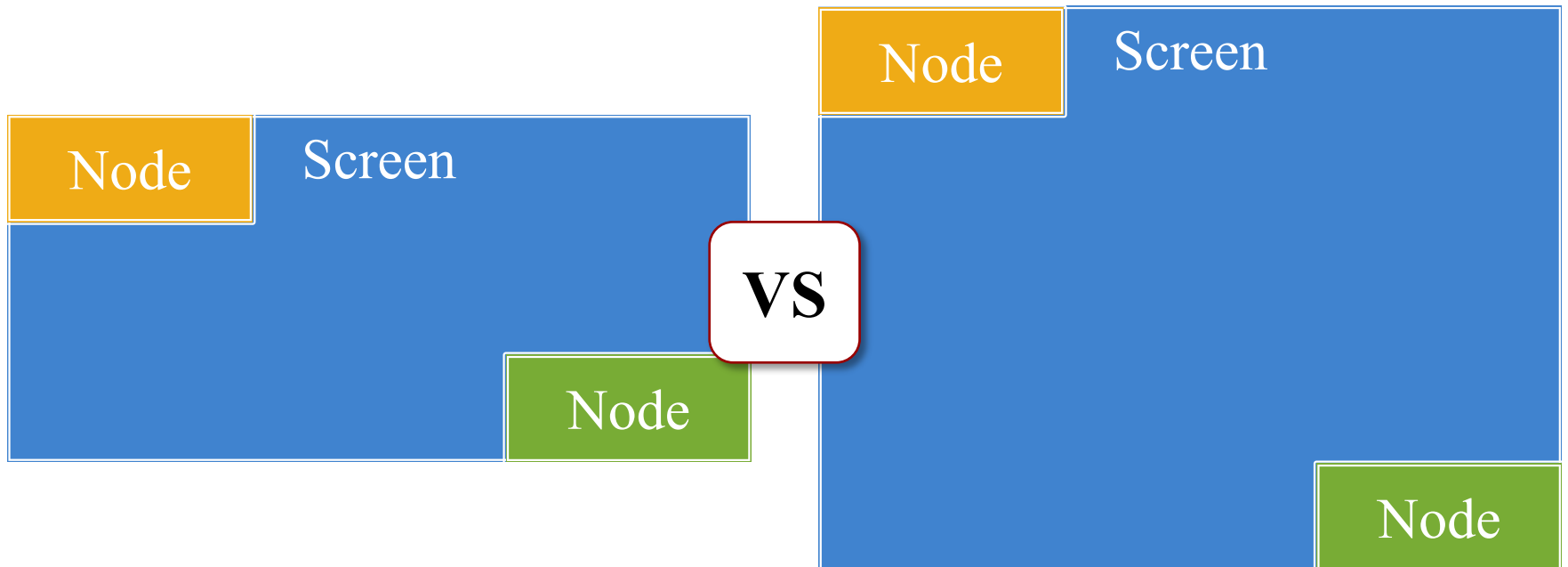
Layout Managers

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

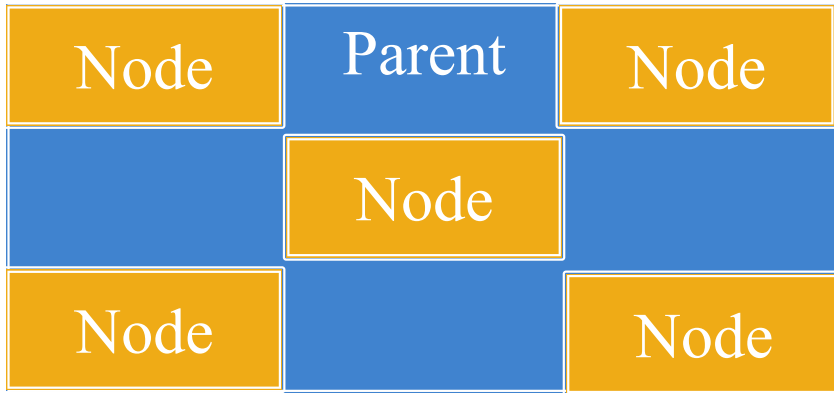


Layout Managers

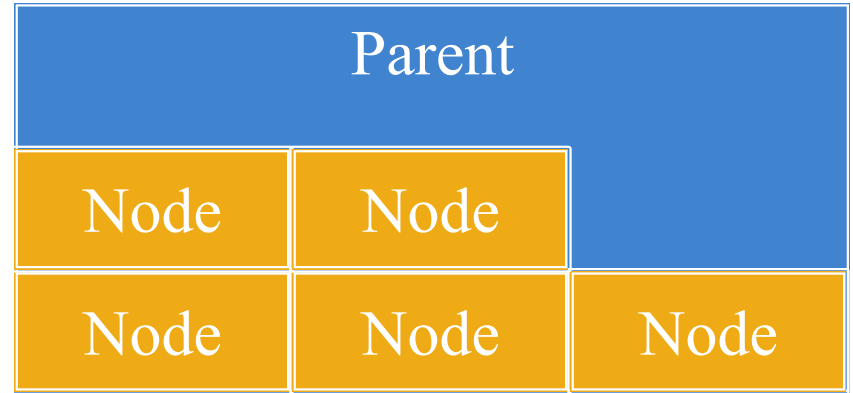
- Not all devices have the same aspect ratio
- Sometimes, want placement to adjust to fit



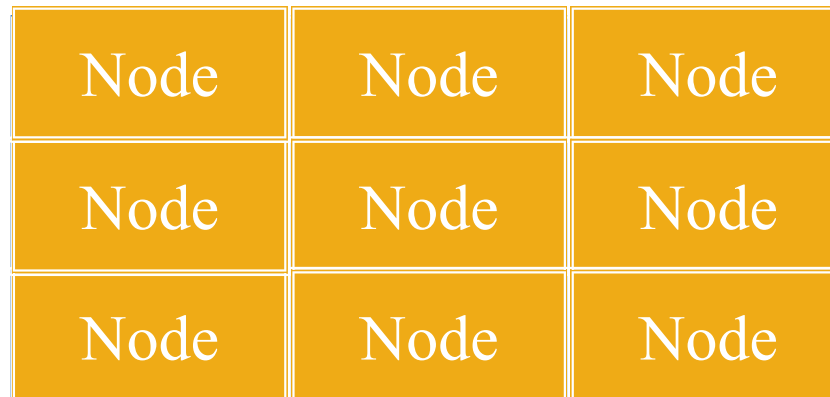
Layout Managers



AnchorLayout

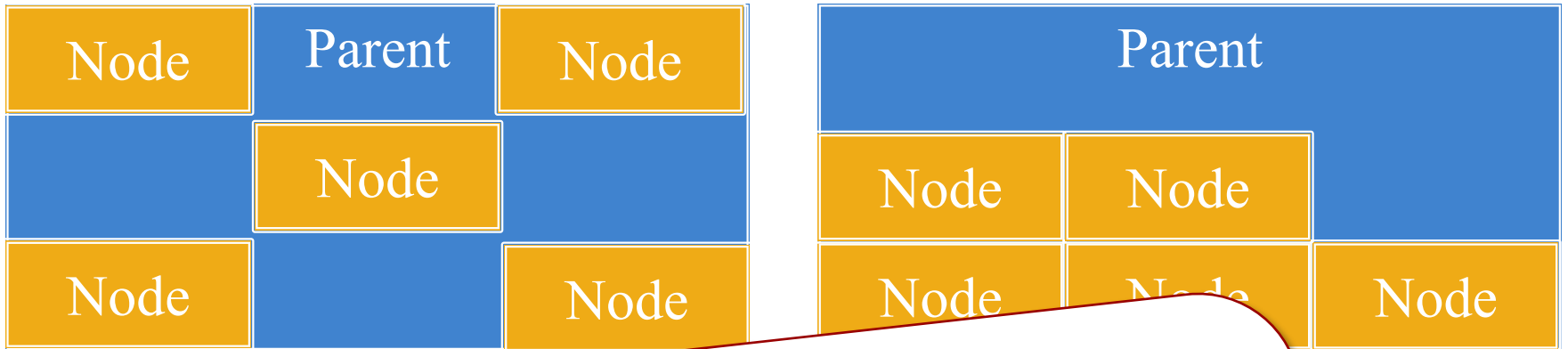


FlowLayout



GridLayout

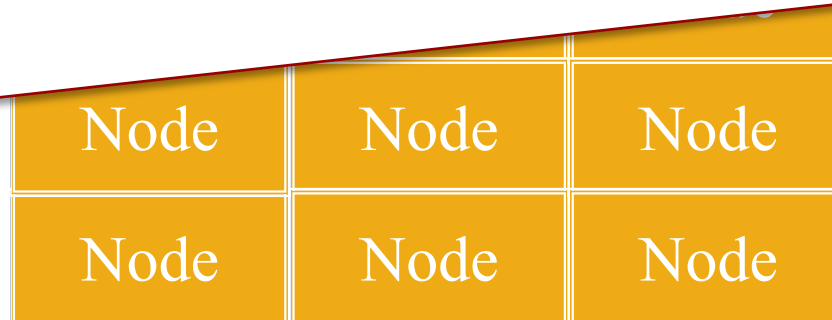
Layout Managers



AnchorLayout

BorderLayout

See Documentation for Details



GridLayout

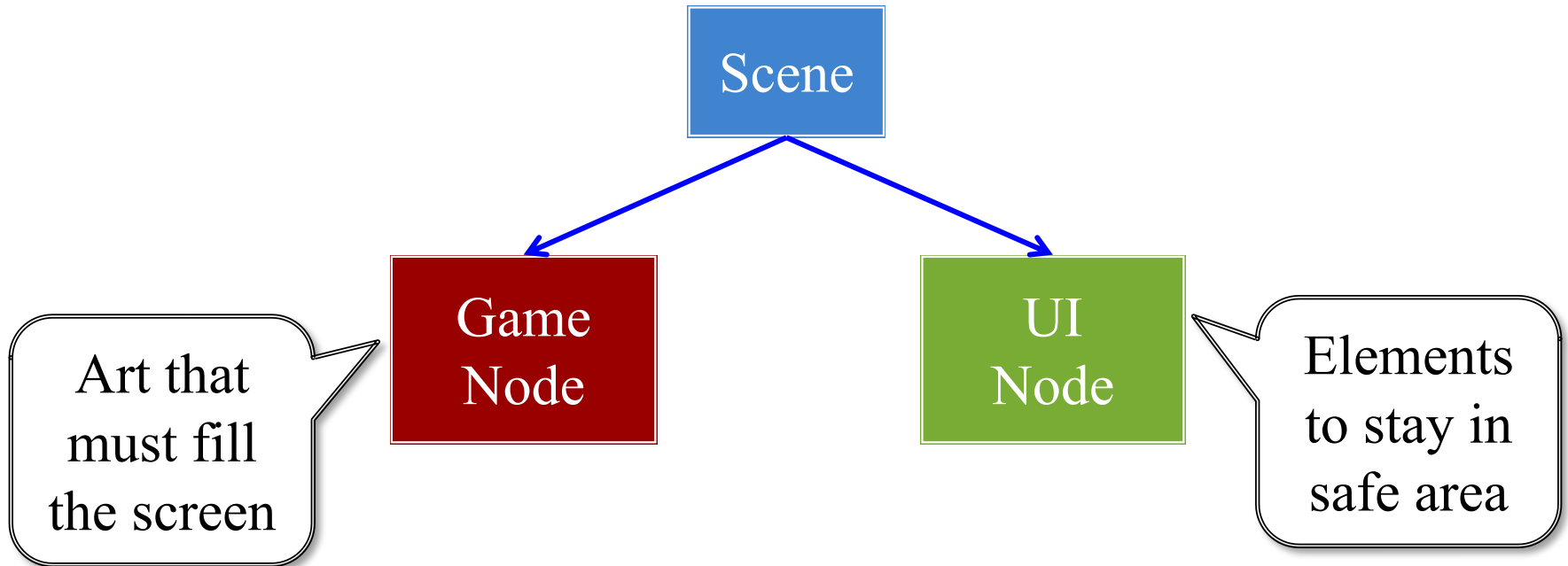
How to Use a Layout Manager

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



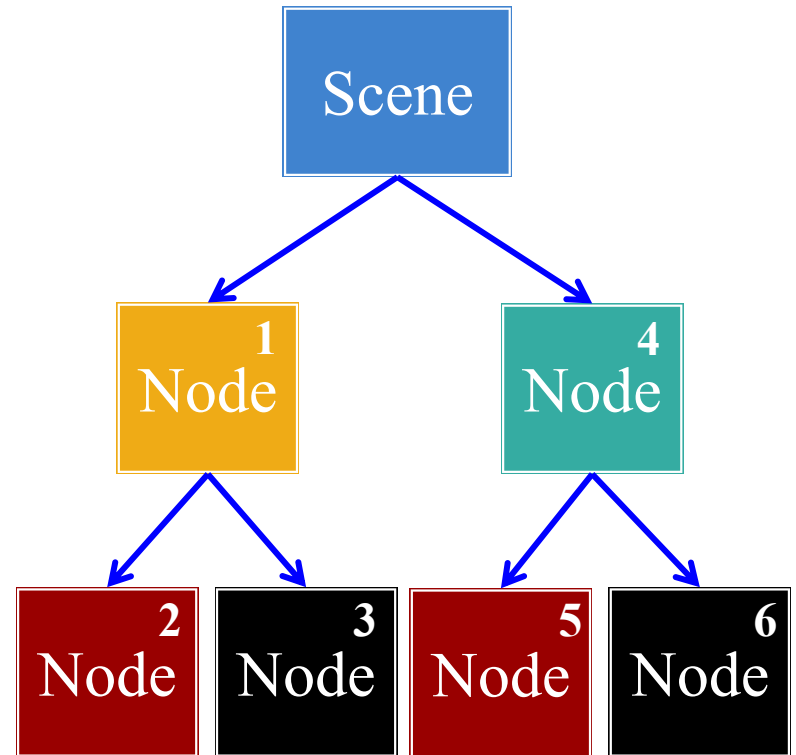
Safe Area: Modern Phones



See Display class to find safe area

Rendering a Scene is Easy

- **scene->render(batch)**
 - 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

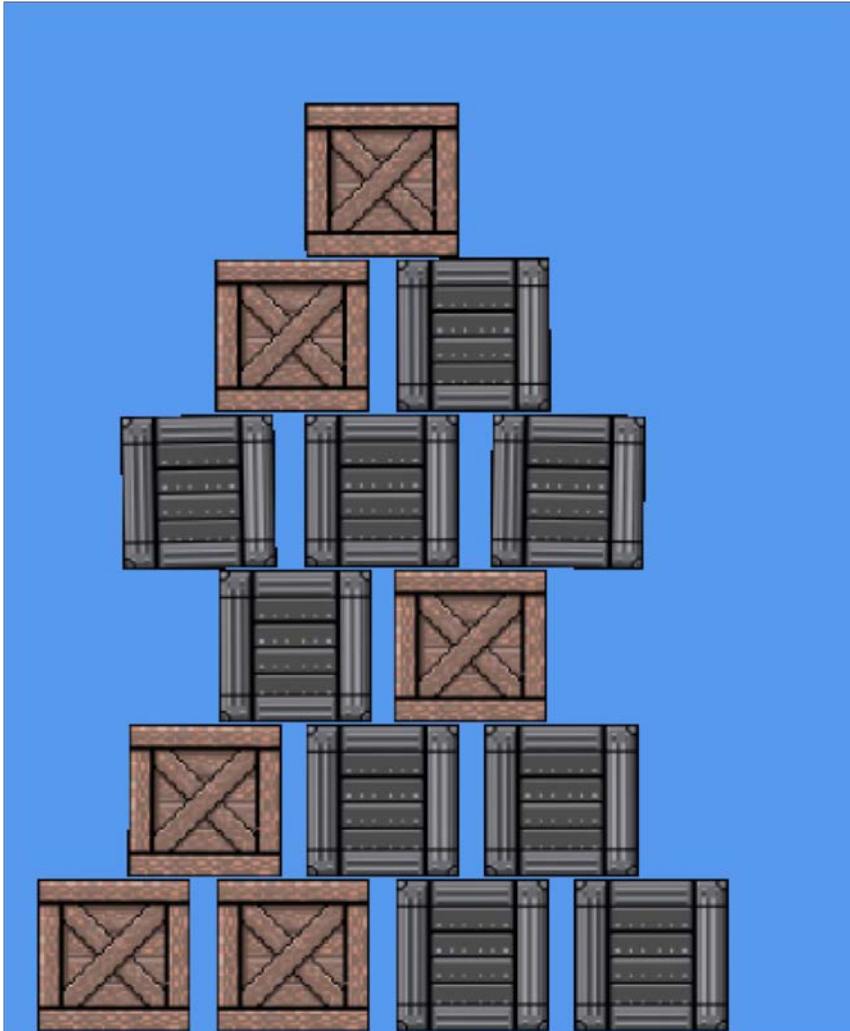


How Does a SpriteBatch Work?



- Sprites = textured **triangles**
 - Gather all sprite vertices
 - Make one list of triangles
 - Send them to GPU at once
- But stall on texture change
 - Reorder data on texture
 - Draw texture all at once
 - Limits texture switches
 - Safe if there is **no overlap**
- Hence the name!

How Does a SpriteBatch Work?



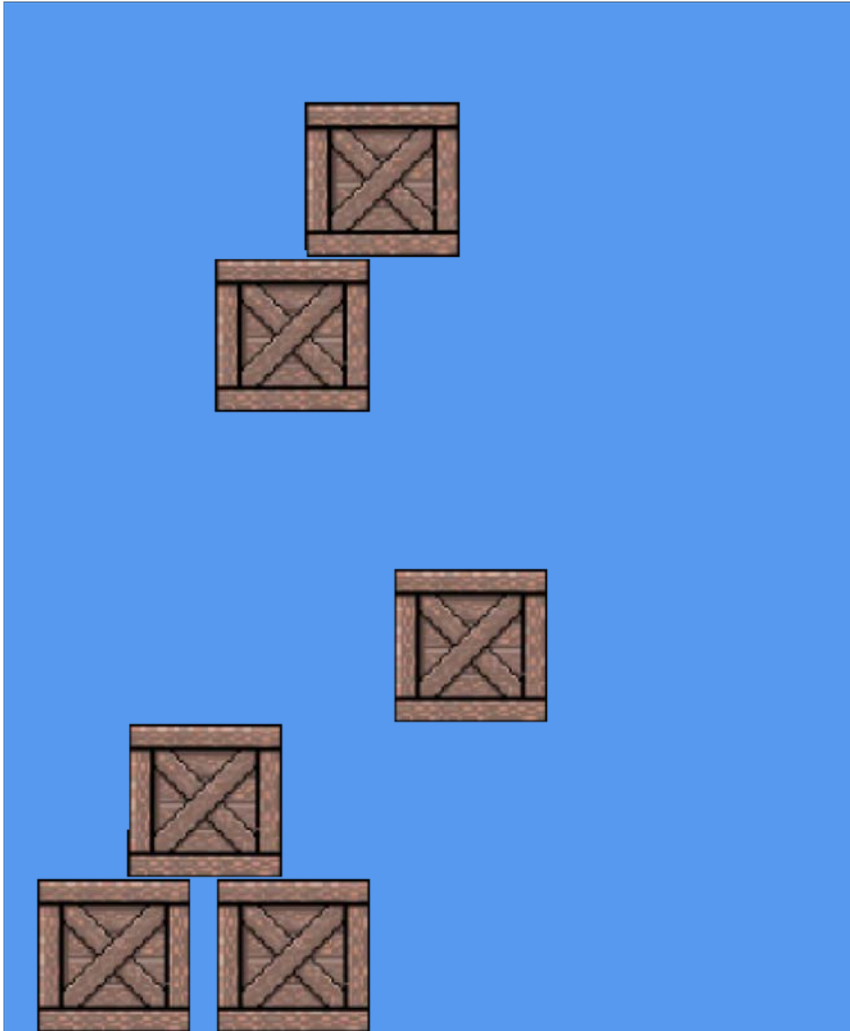
- Sprites = textured **triangles**
 - Gather all sprite vertices
 - Make one list of triangles
 - Send them to GPU at once
- But stall on texture change
 - Reorder data on texture
 - Draw texture all at once
 - Limits texture switches
 - Safe if there is **no overlap**
- Hence the name!

How Does a SpriteBatch Work?



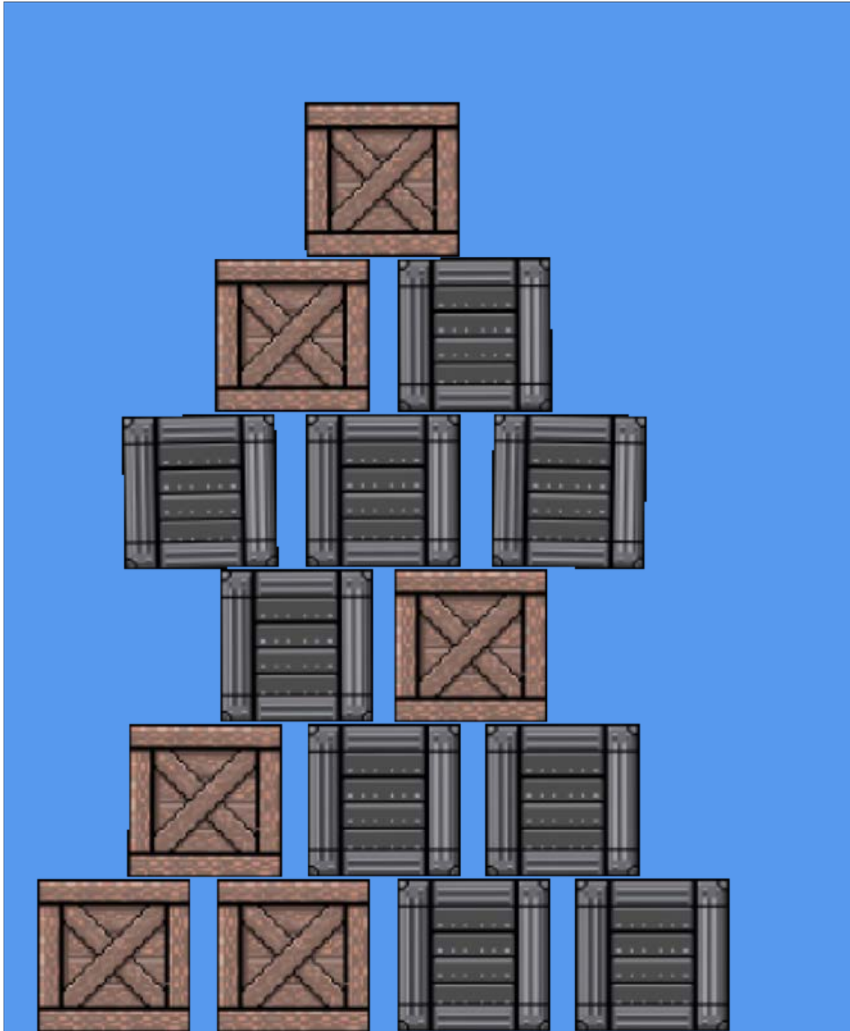
- Sprites = textured **triangles**
 - Gather all sprite vertices
 - Make one list of triangles
 - Send them to GPU at once
- But stall on texture change
 - Reorder data on texture
 - Draw texture all at once
 - Limits texture switches
 - Safe if there is **no overlap**
- Hence the name!

How Does a SpriteBatch Work?



- Sprites = textured **triangles**
 - Gather all sprite vertices
 - Make one list of triangles
 - Send them to GPU at once
- But stall on texture change
 - Reorder data on texture
 - Draw texture all at once
 - Limits texture switches
 - Safe if there is **no overlap**
- Hence the name!

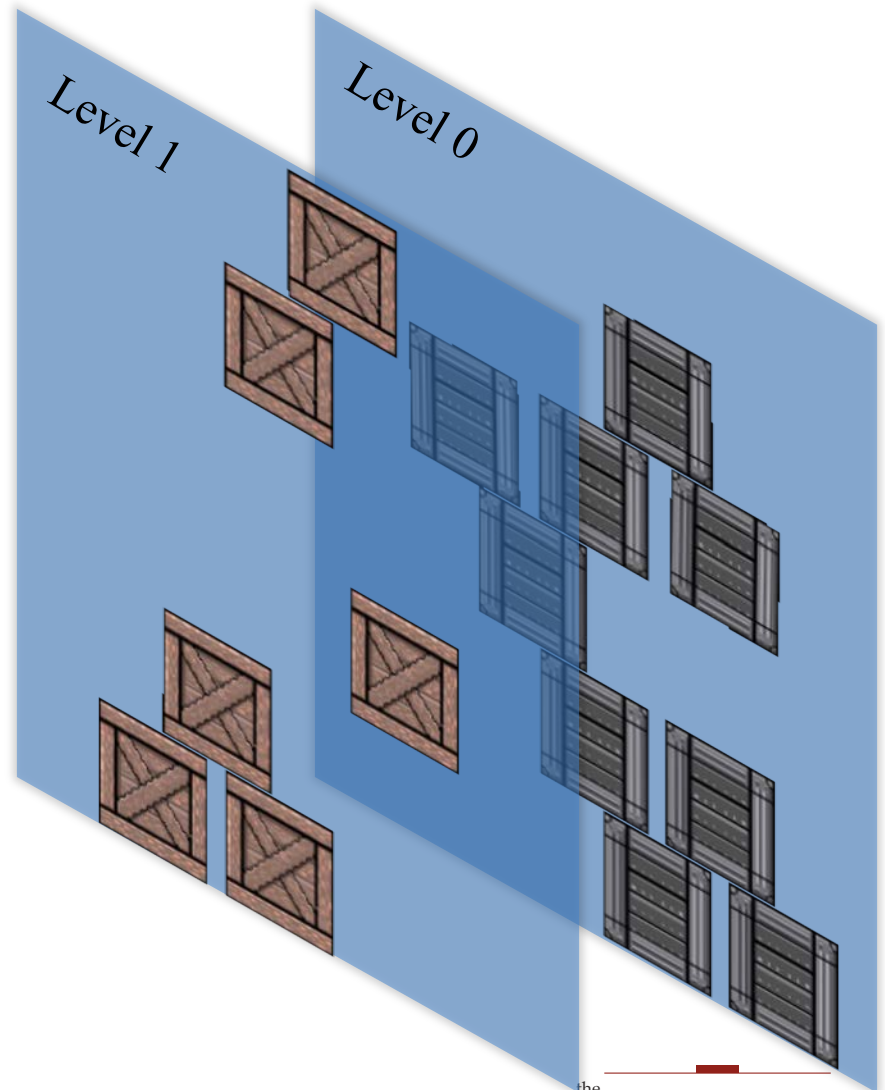
How Does a SpriteBatch Work?



- Sprites = textured **triangles**
 - Gather all sprite vertices
 - Make one list of triangles
 - Send them to GPU at once
- But stall on texture change
 - Reorder data on texture
 - Draw texture all at once
 - Limits texture switches
 - Safe if there is **no overlap**
- Hence the name!

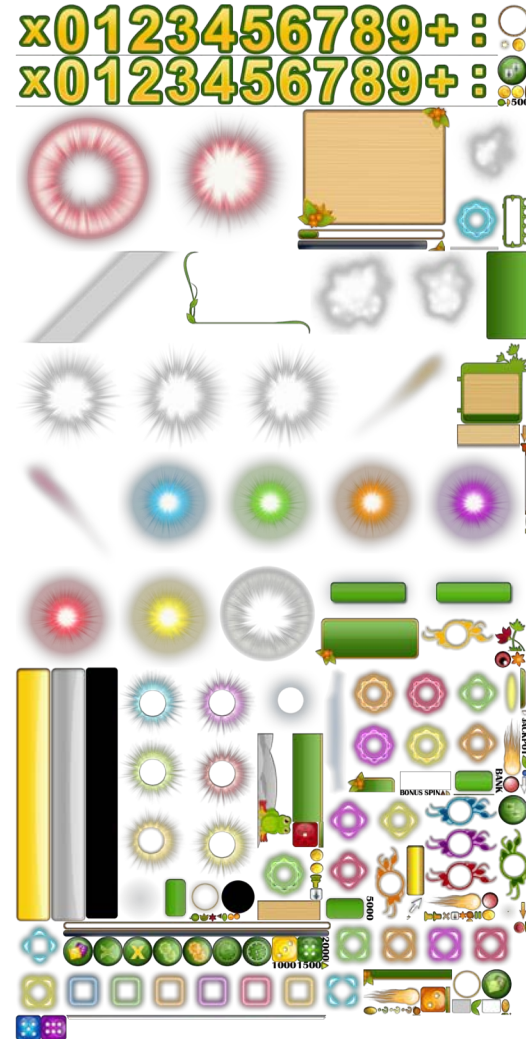
Optimizing Performance: zOrder

- SpriteBatch defers to Scene
 - Scene determines order
 - Reordering a scene is bad
- Give *hints* to the Scene
 - Give each child a z-value
 - Ties are permitted
 - `sortZOrder()` sorts tree
 - Can make this automatic
- Controls **texture switching**
 - One texture = one z-value
 - Reduces it to one draw call



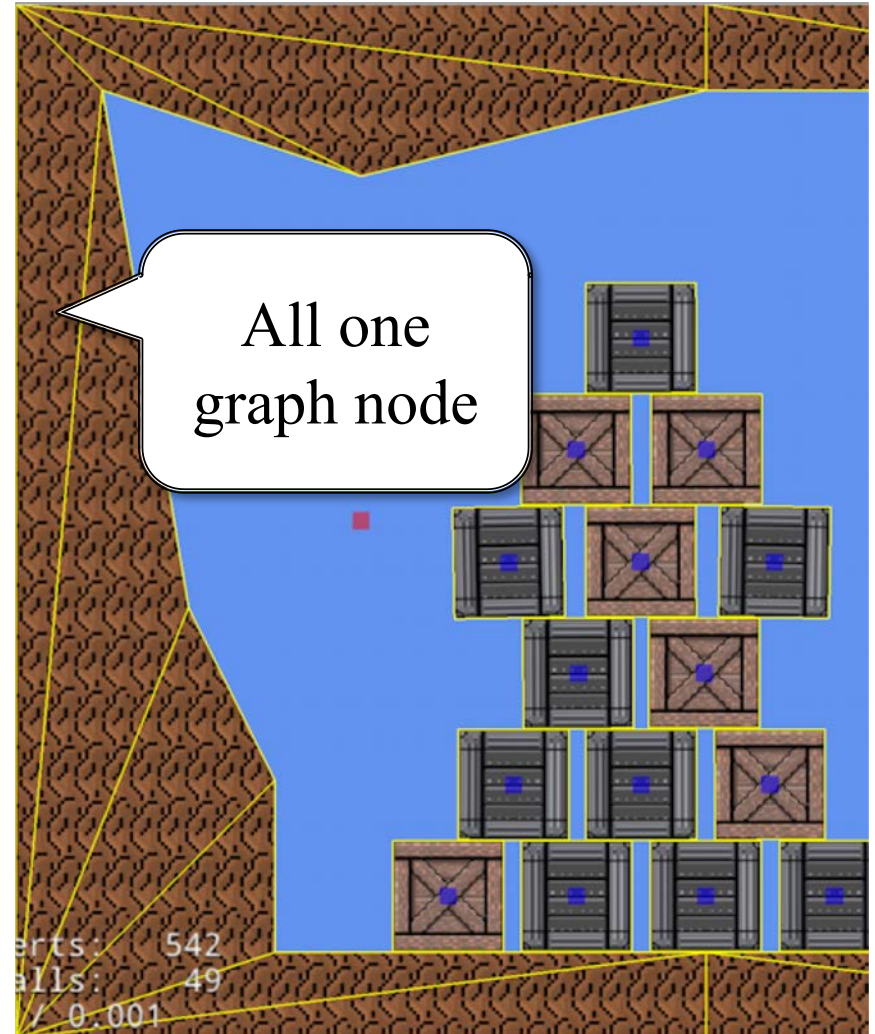
Optimizing Performance: Atlases

- **Idea:** Never switch textures
 - Film strip is many images
 - We can draw part of texture
 - One texture for everything?
 - Called a **texture atlas**
- Disadvantages?
 - Cannot tile textures
 - Can be tricky to pack
- Ideal for **interface design**
 - Images for UX widgets
 - Often small and compact

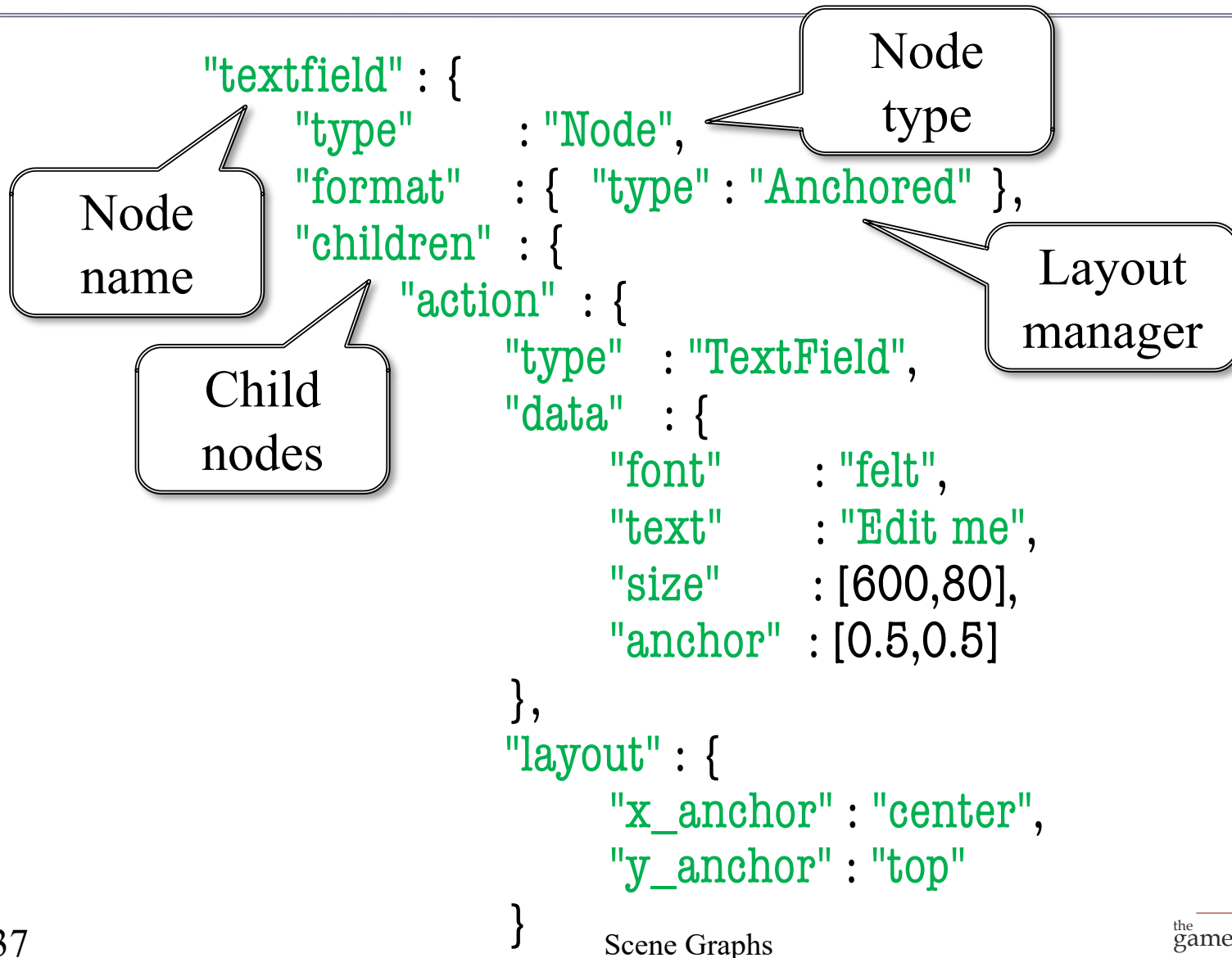


Specialized Nodes

- CUGL has many node types
 - [AnimationNode](#) (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

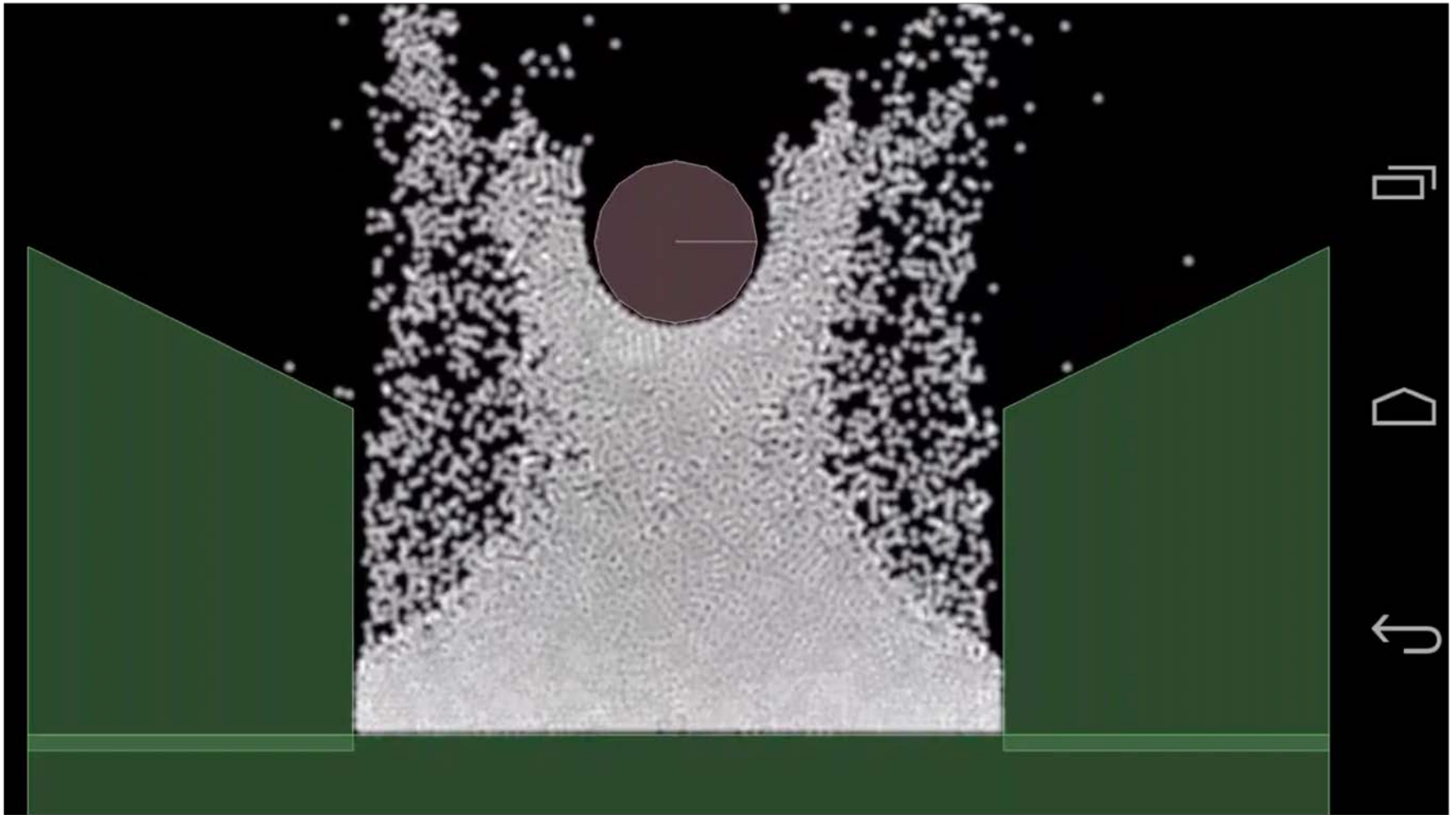
JSON Language for Scene Graphs

```
"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"  
    }  
  }  
}
```

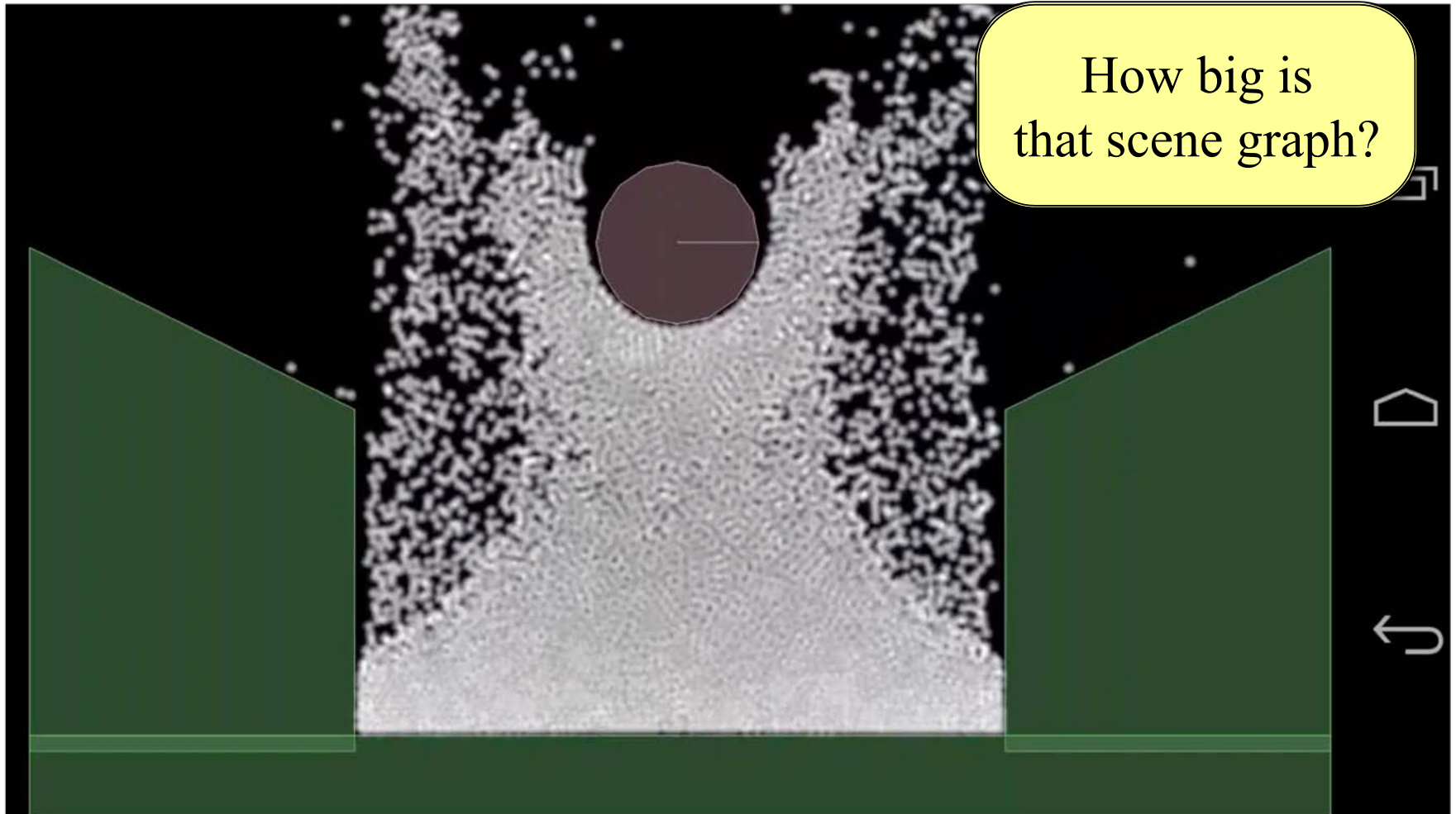
Each node has

- Type
- Format
- Data
- Children
- Layout

The Problem: **Physics**



The 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 draw() Method

```
void CustomNode::draw(const std::shared_ptr<SpriteBatch>& batch,
                    const Mat4& transform, Color4 tint) {

    if (!_rendered) {
        generateRenderData();
    }

    batch->setColor(tint);
    batch->setTexture(_texture);
    batch->setBlendEquation(_blendEquation);
    batch->setBlendFunc(_srcFactor, _dstFactor);

    batch->fill(_vertices, _vertsized, 0,
              _indices, _indxsize, 0,
              transform);
}
```

The draw() Method

```
void CustomNode::draw(const std::shared_ptr<SpriteBatch>& batch,  
                     const Mat4& transform, Color4 tint) {
```

```
    if (!_rendered) {  
        generateRenderData();  
    }
```

Computed from
parent (+camera)

Computed from
parent (+scene)

```
    batch->setColor(tint);  
    batch->setTexture(_texture);  
    batch->setBlendEquation(_blendEquation);  
    batch->setBlendFunc(_srcFactor, _dstFactor);
```

```
    batch->fill(_vertices, _vertsiz, 0,  
              _indices, _indxsize, 0,  
              transform);
```

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)
- New JSON language makes design easier