

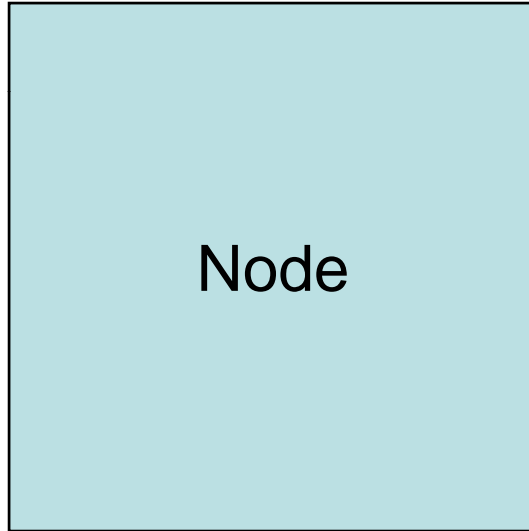
# CS345/DIGM465: Computer Game Development

Now you are thinking with nodes: X3D + ECMAScript

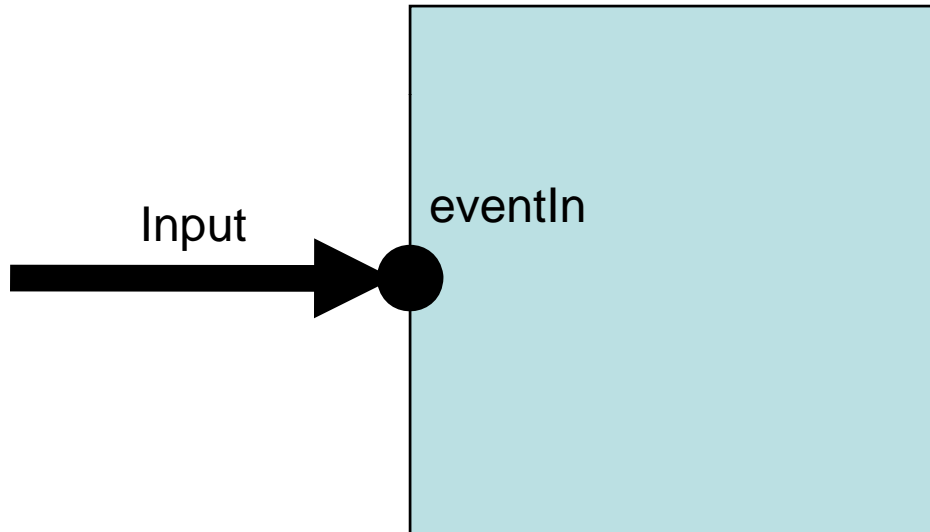


Will Muto  
Digital Media  
Drexel University

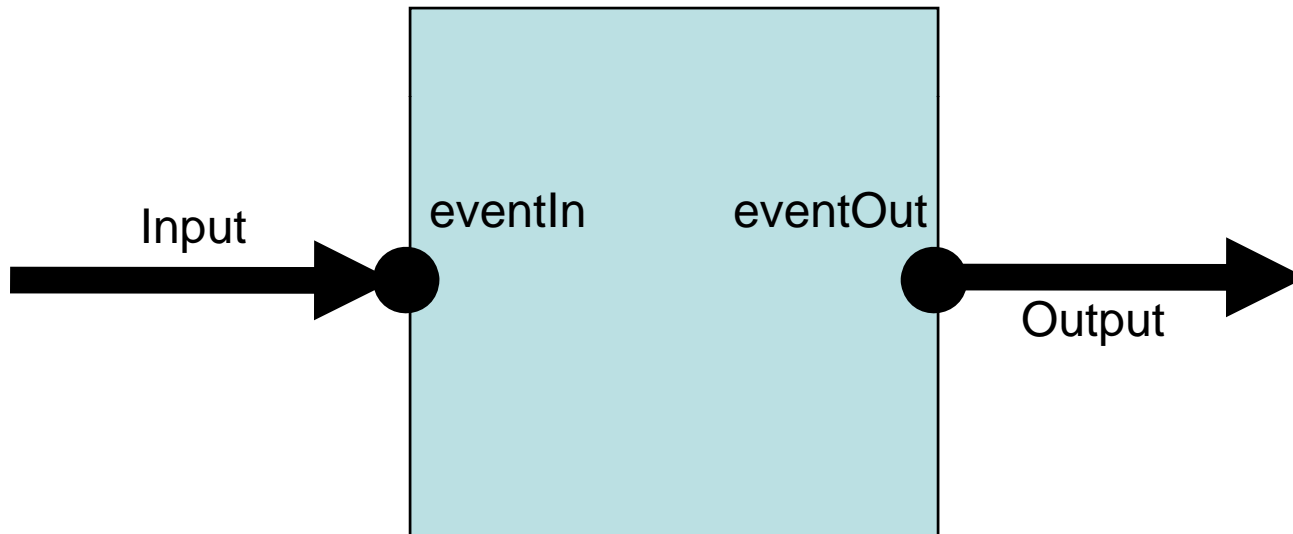
# Nodes



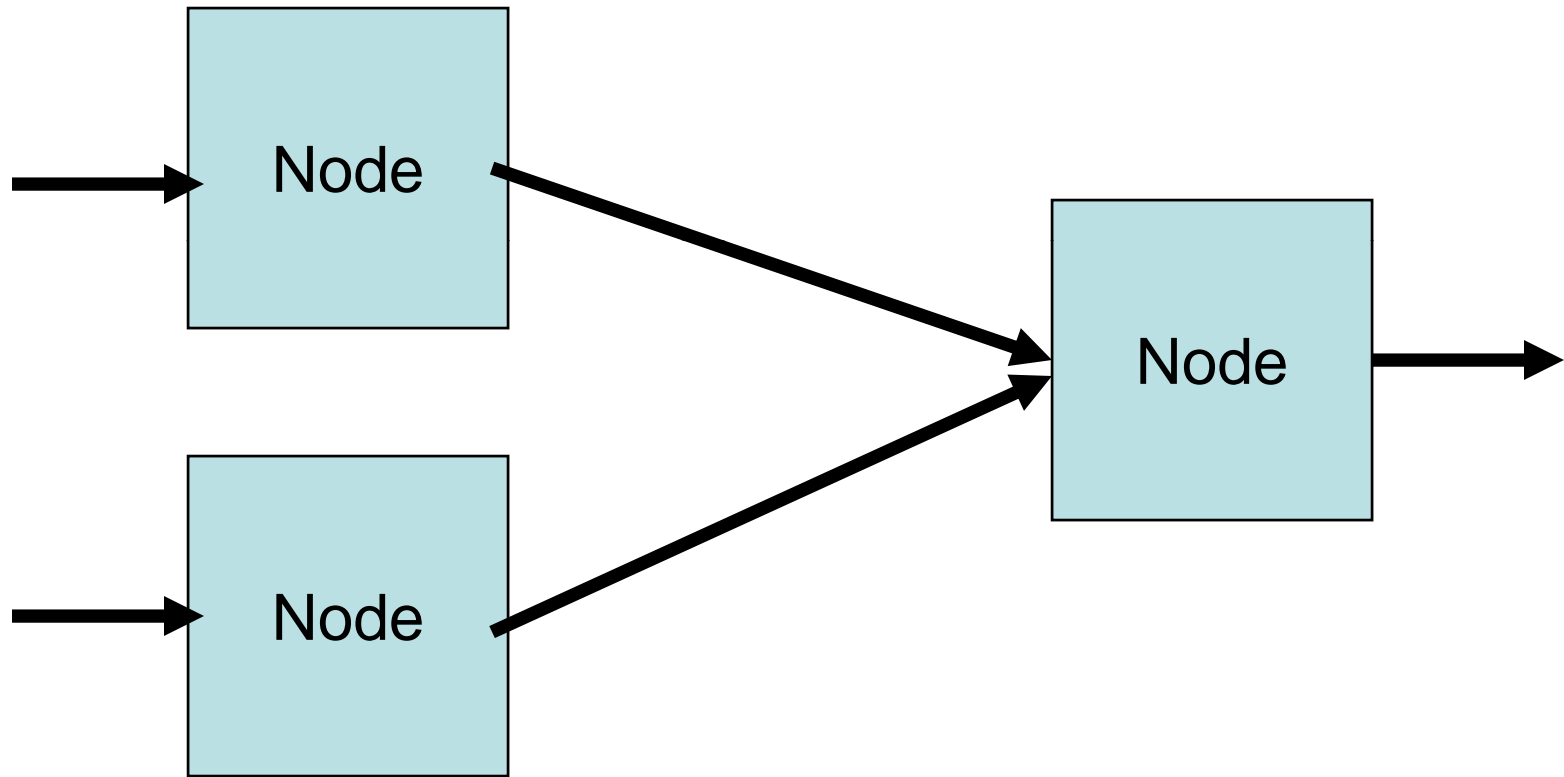
# Nodes



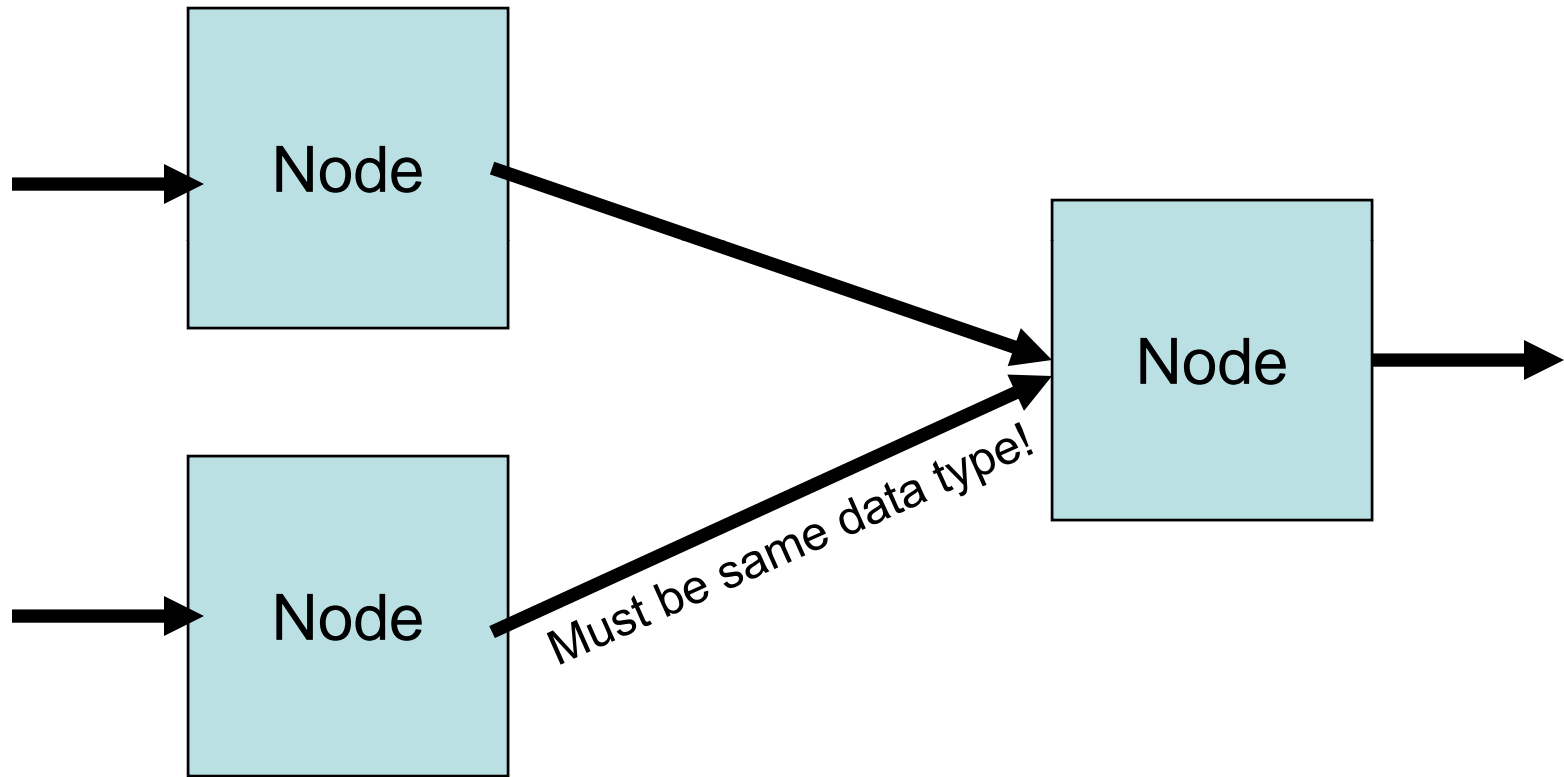
# Nodes



# Nodes

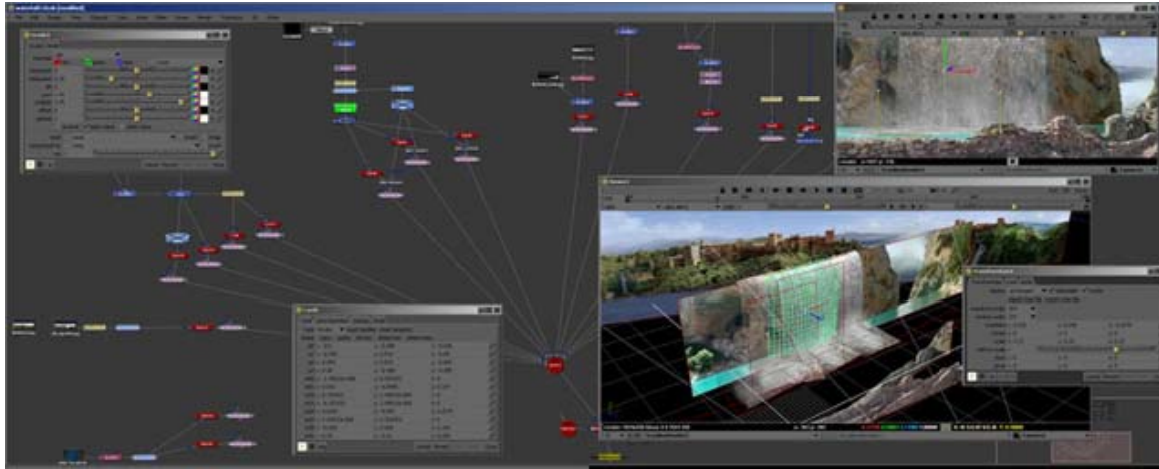


# Nodes

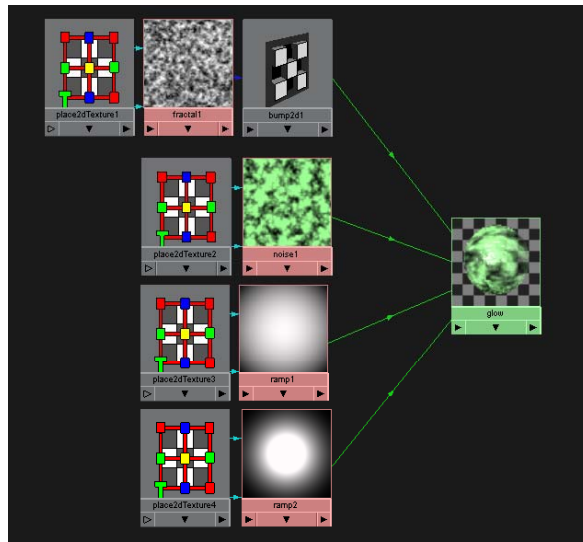


# Nodes

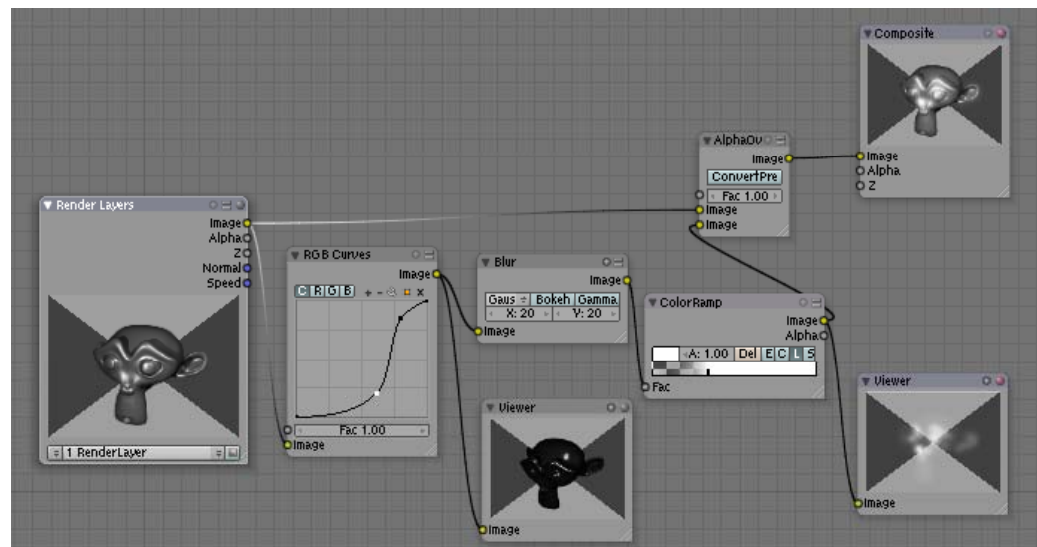
The Foundry's Nuke



Maya Hypershade



Blender Compositor



# UnrealKismet

UnrealKismet: Main\_Sequence

The image displays a complex Kismet graph for a game sequence. The graph is organized into several functional sections:

- Initialize Board (Delete non-player pucks):** This section uses a 'ForEach' loop to iterate through a list of pucks. It checks if a puck is a 'Player Puck' and, if not, triggers a 'Delete' action.
- Rotate Out Blank and Driver In:** This section involves a series of 'Set Actor Location' and 'Set Actor Rotation' nodes, likely used to reposition and reorient the puck components.
- WHO'S TURN IS IT?:** A logic brick that determines the current player's turn, possibly by checking the puck's state or a game timer.
- Opening Help Screen:** A sequence of nodes that triggers the opening of a help screen, likely involving a 'Play' node and a 'Set Visibility' node.
- Repeating current player indicator:** A logic brick that updates the current player indicator, possibly by setting a variable or a UI element.
- PUCK BEHAVIOR:** A central logic brick that manages the puck's movement and state, including a 'Set Actor Location' node and a 'Set Actor Rotation' node.
- Toggle Realtime or Turn Based:** A logic brick that allows the game to be played in either real-time or turn-based mode, likely using a 'Toggle' node and a 'Set Actor Location' node.

**Properties Panel:**

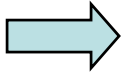
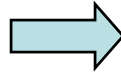
bEnabled	<input checked="" type="checkbox"/>
bPlayerOnly	<input type="checkbox"/>
MaxTriggerCount	1
ObjComment	
Priority	0
ReTriggerDelay	0.000000

**Component List:**

- Main\_Sequence
- Add\_Subtract\_X
- Blank\_Definition\_Swap
- DRIVER\_PUCK
- Retexture\_Driver
- TrackComponents
- TrackComponents\_0
- TrackComponentsB
- TrackComponentB\_1

Object: None

# What is X3D?

- X3D is a “royalty free, open standard for real-time 3D communication”
- Created by the Web3D Consortium ([web3d.org](http://web3d.org))
- The current iteration of VRML (Virtual Reality Modeling Language)
  - VRML 1  VRML97  X3D
- Explored through a browser plug-in or third-party viewer

# What is X3D?

- X3D = VRML + XML

```
DEF redbox Transform {
  translation 0 0 -.29569
  children [
    DEF Box5 Shape {
      appearance Appearance {
        material DEF Red Material {
          ambientIntensity 0.200
          shininess 0.200
          diffuseColor 1 0 0
        }
      }
      geometry DEF GeoBox5 Box {
        size 1 1 1
      }
    }
  ]
}
```

**VRML97**

=

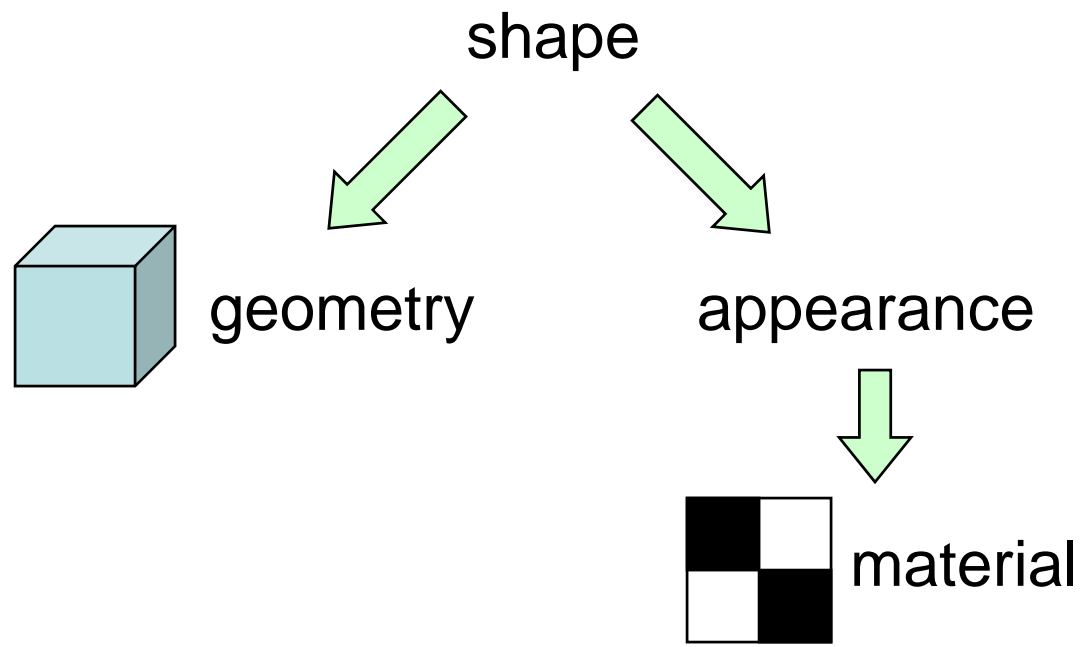
```
<Transform DEF='redbox'
  translation='0 0 -.29569'>
  <Shape DEF='Box5'
    containerField='children'>
    <Appearance
      containerField='appearance'>
      <Material DEF='Red'
        containerField='material'
        ambientIntensity='0.200'
        shininess='0.200'
        diffuseColor='1 0 0' />
      </Appearance>
      <Box DEF='GeoBox5'
        containerField='geometry'
        size='1 1 1' />
    </Shape>
  </Transform>
</Scene>
</X3D>
```

**X3D**

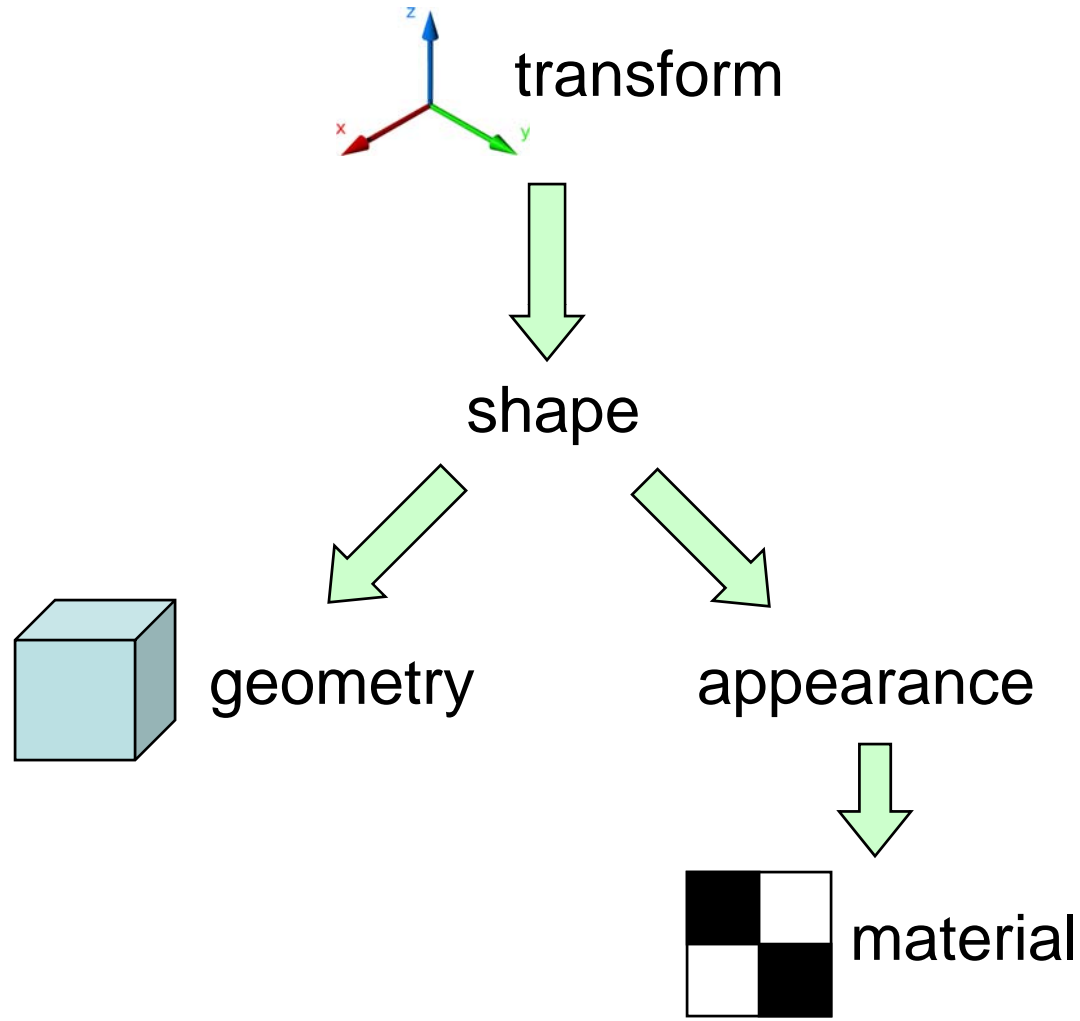
What is all that gobbledygook?

shape

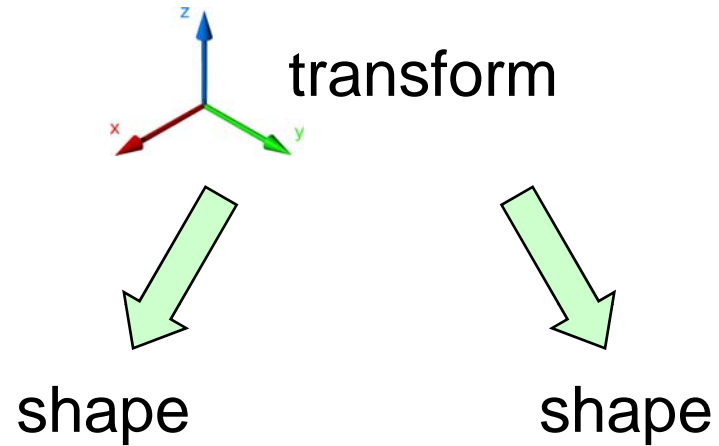
# What is all that gobbledygook?



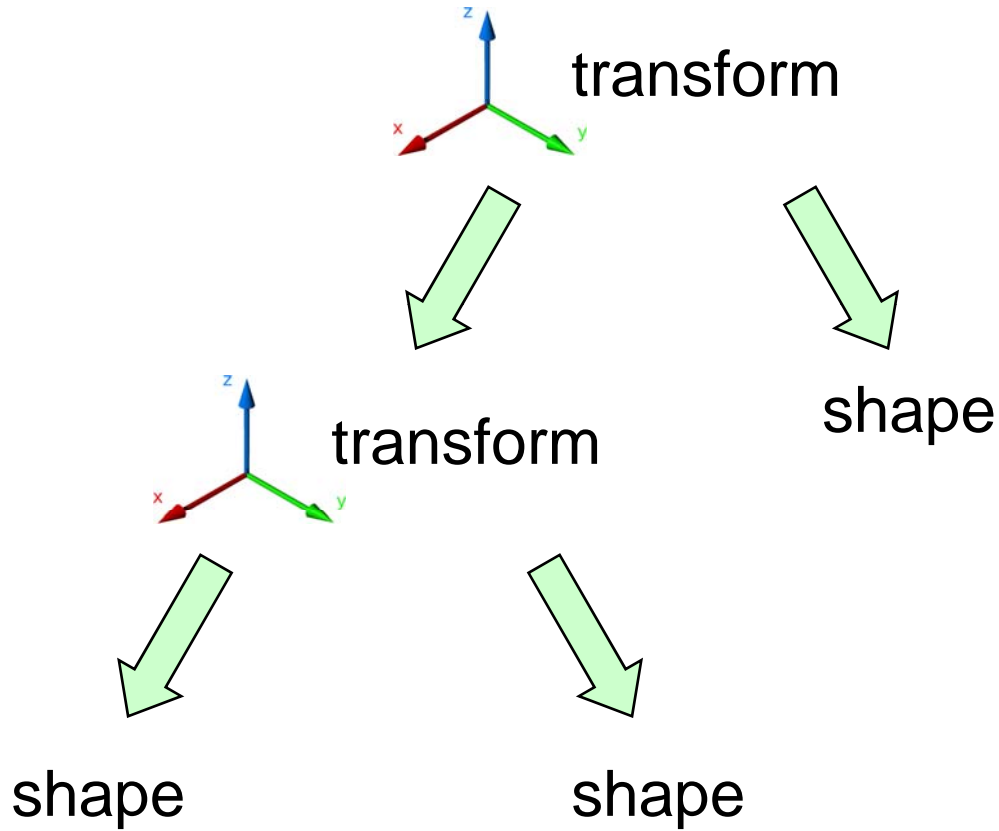
# What is all that gobbledygook?



# What is all that gobbledygook?



# What is all that gobbledygook?



*Ad nauseum...*

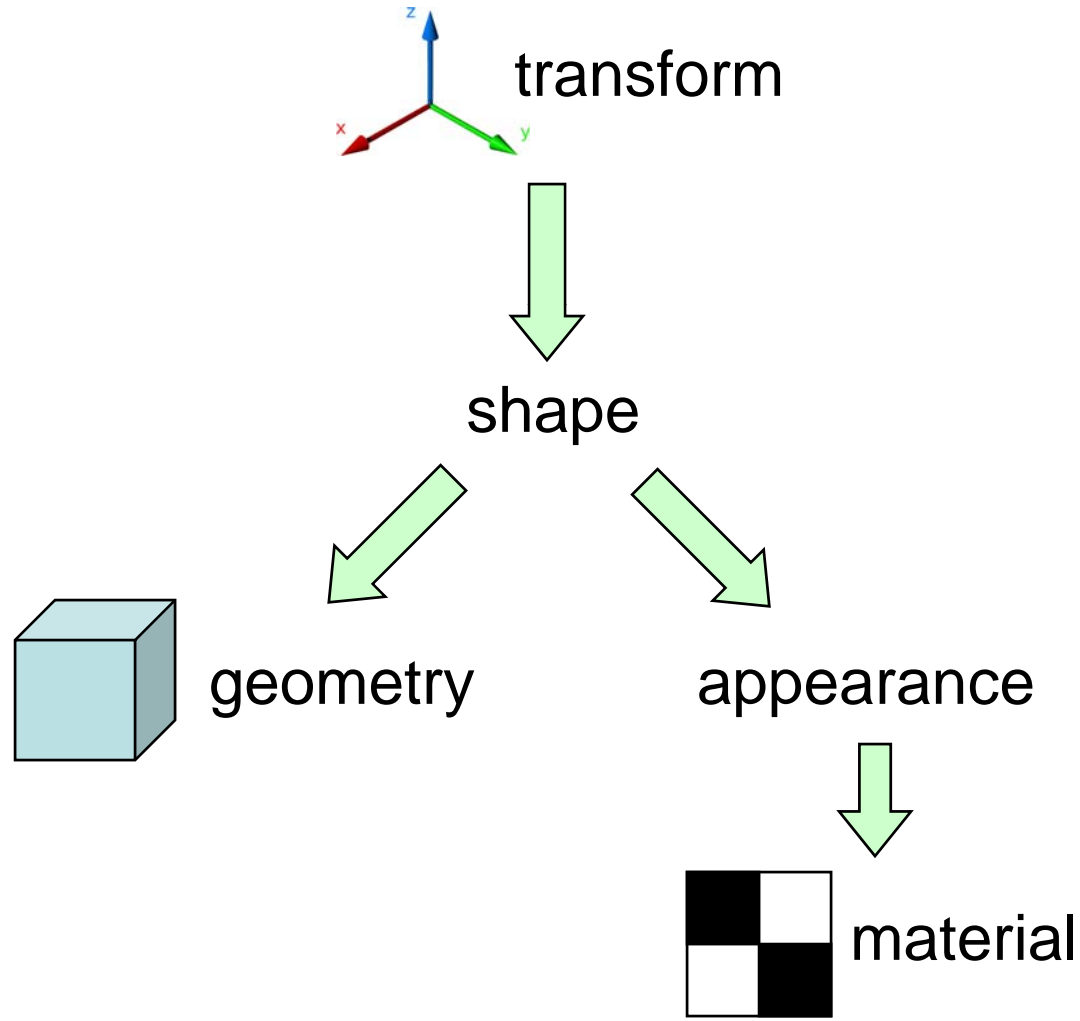
# What is all that gobbledygook?

Like all good documents, include metadata!

```
#X3D V3.0 utf8
```

```
WorldInfo {  
  title "Hello World"  
  info "A small sample world."  
}
```

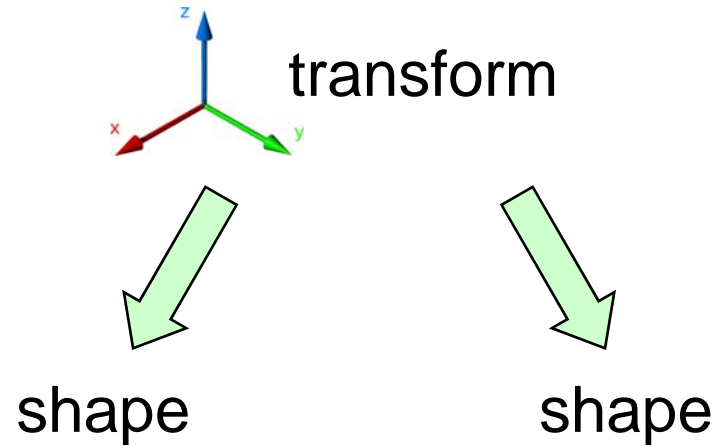
# What is all that gobbledygook?



# What is all that gobbledygook?

```
Transform {
    translation 10 0 0
    rotation 1 0 0 0
    DEF redbox Shape {
        appearance Appearance {
            material DEF Red Material {
                ambientIntensity 0.200
                shininess 0.200
                diffuseColor 1 0 0
            }
        }
        geometry DEF GeoBox2 Box {
            size 1 1 1
        }
    }
}
```

# What is all that gobbledygook?



# What is all that gobbledygook?

```
Transform {  
  translation 10 0 0  
  rotation 1 0 0 0  
  children [  
    DEF redbox Shape {...}  
    DEF bluesphere Shape {...}  
  ]  
}
```

# What else can I create?

## Geometry

- Box *length width height*
- Cone *radius height sidesCreated bottomCreated*
- Cylinder *bottomCreated height radius sidesCreated topCreated*
- ElevationGrid **#uniform grid of varying height**
- Extrusion
- IndexedFaceSet **#all of your geometry will probably be this**
- Sphere *radius*
- Text *string[]*

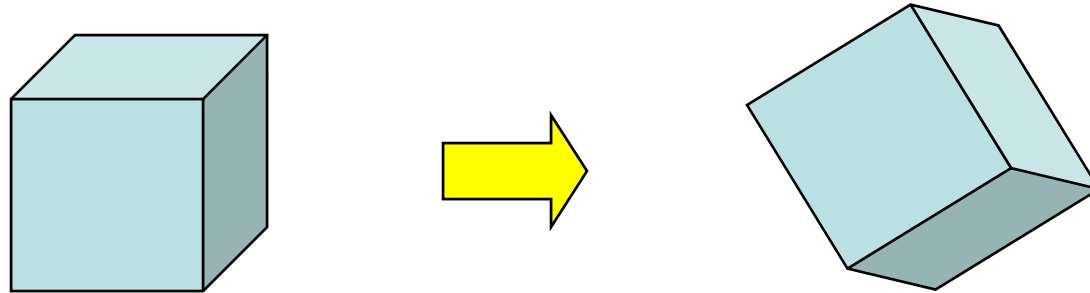
# What else can I create?

## Lights

- `DirectionalLight` *ambientIntensity color direction intensity*
- `PointLight` *attenuation color intensity location radius ...*
- `SpotLight` *attenuation beamWidth color direction intensity ...*

# What about interaction?

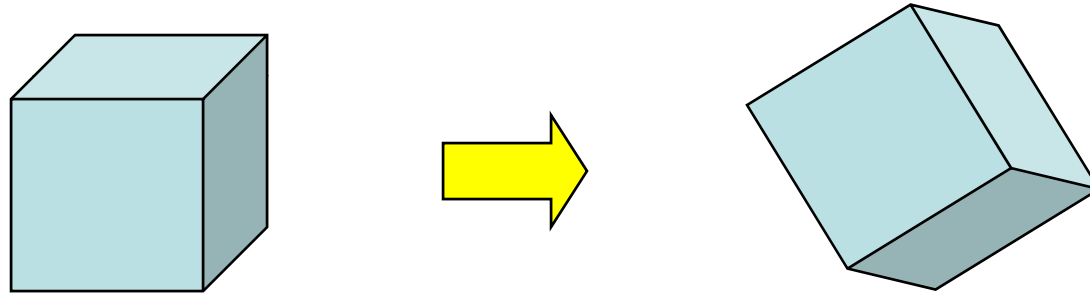
Say I want to click and drag a cube to rotate it.



The cube's rotation is already set with its transform. How can it change at runtime?

# What about interaction?

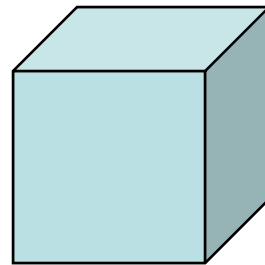
Say I want to click and drag a cube to rotate it.



The cube's rotation is already set with its transform. How can it change at runtime?

Answer: routes.

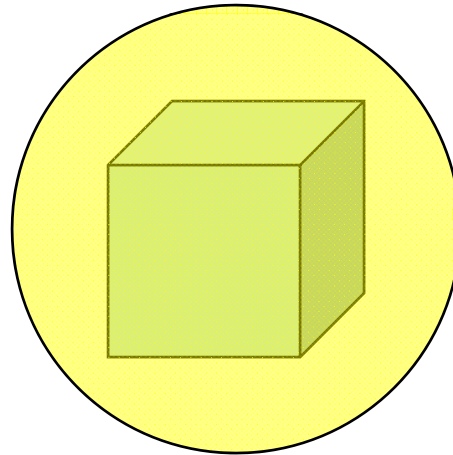
# What about interaction?



Cube

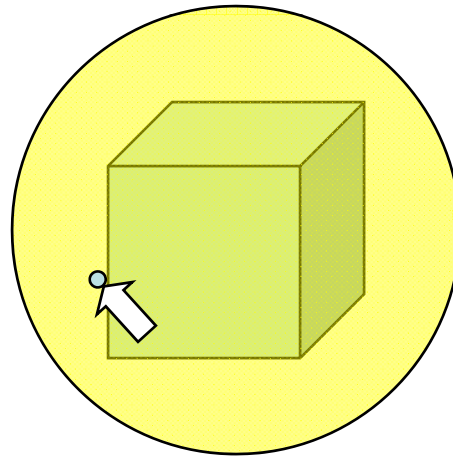
# What about interaction?

SphereSensor



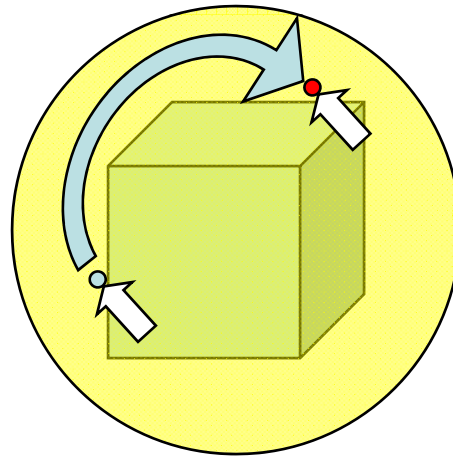
# What about interaction?

SphereSensor

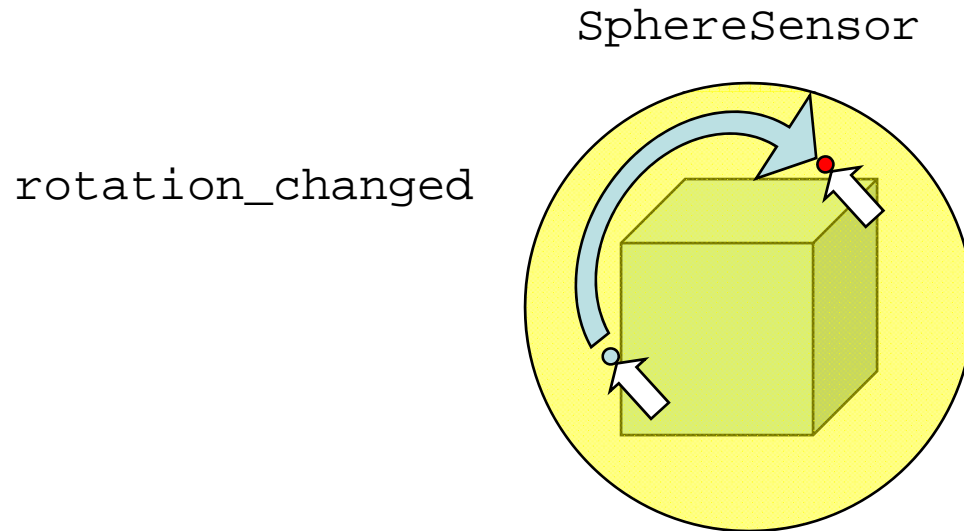


# What about interaction?

SphereSensor

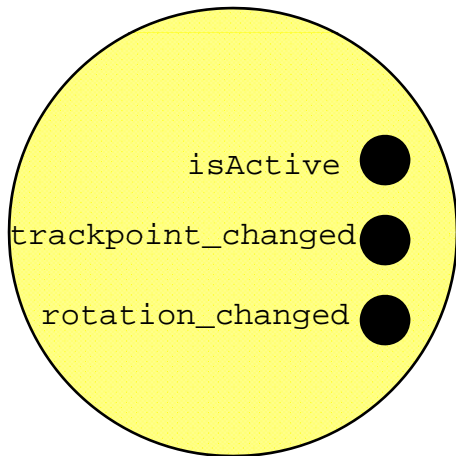


# What about interaction?

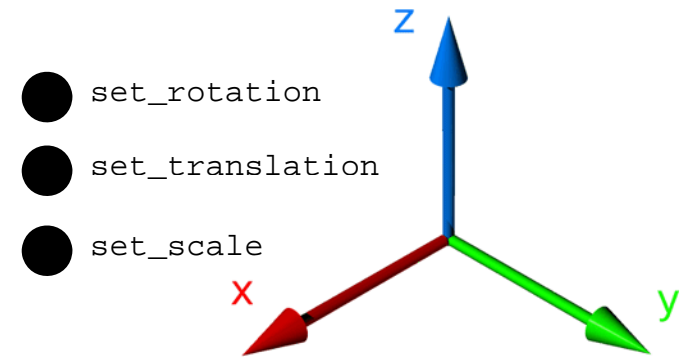


# What about interaction?

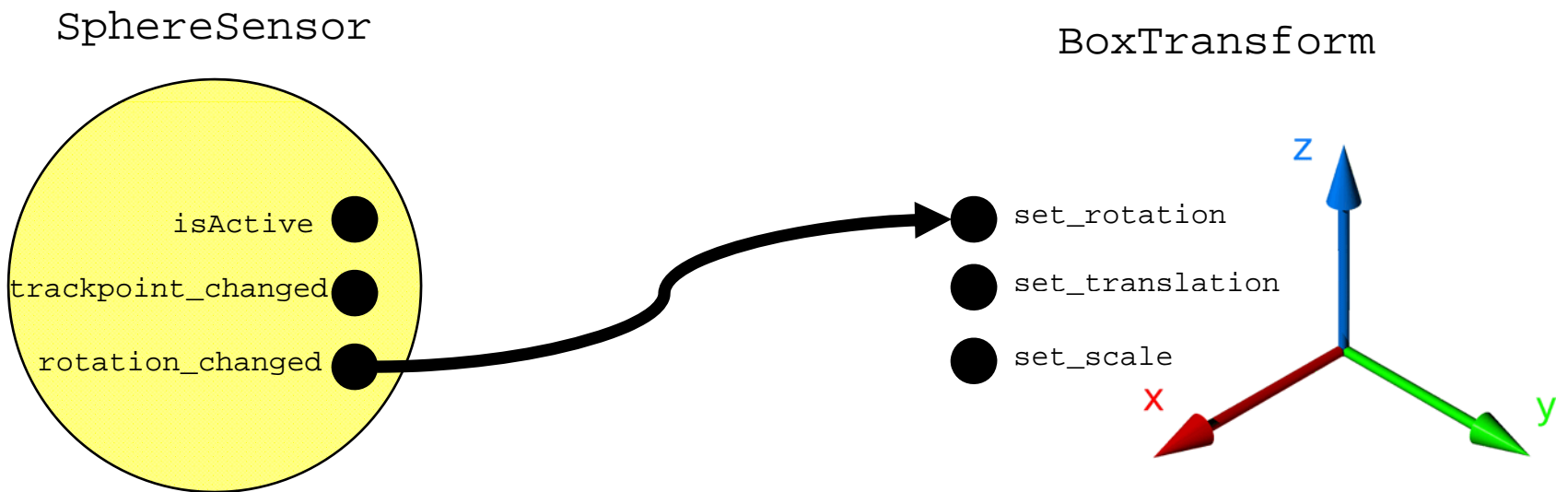
SphereSensor



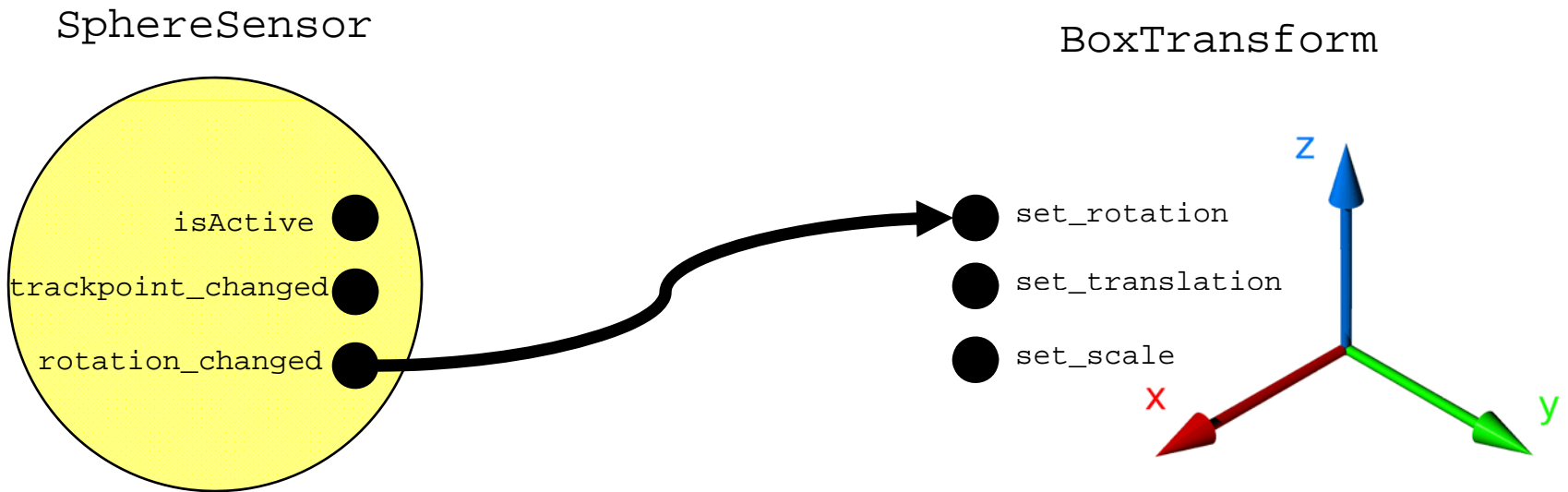
BoxTransform



# What about interaction?



# What about interaction?



ROUTE `SensorName.rotation_changed` TO `BoxTransform.set_rotation`

# What about interaction?

There are several types of sensors:

- SphereSensor (spherical motion around an origin)
- CylinderSensor (circular motion around Y-axis of cylinder)
- PlaneSensor (maps point to a 2D translation on a plane)
- ProximitySensor (sends event on enter/exit/move within area)
- TouchSensor (Detects touch on geometry at a point)
- VisibilitySensor (Detects visibility changes of box region)

# A little trickier...

You can also track the passage of time using a TimeSensor



`cycleTime`



`fraction_changed`



`isActive`

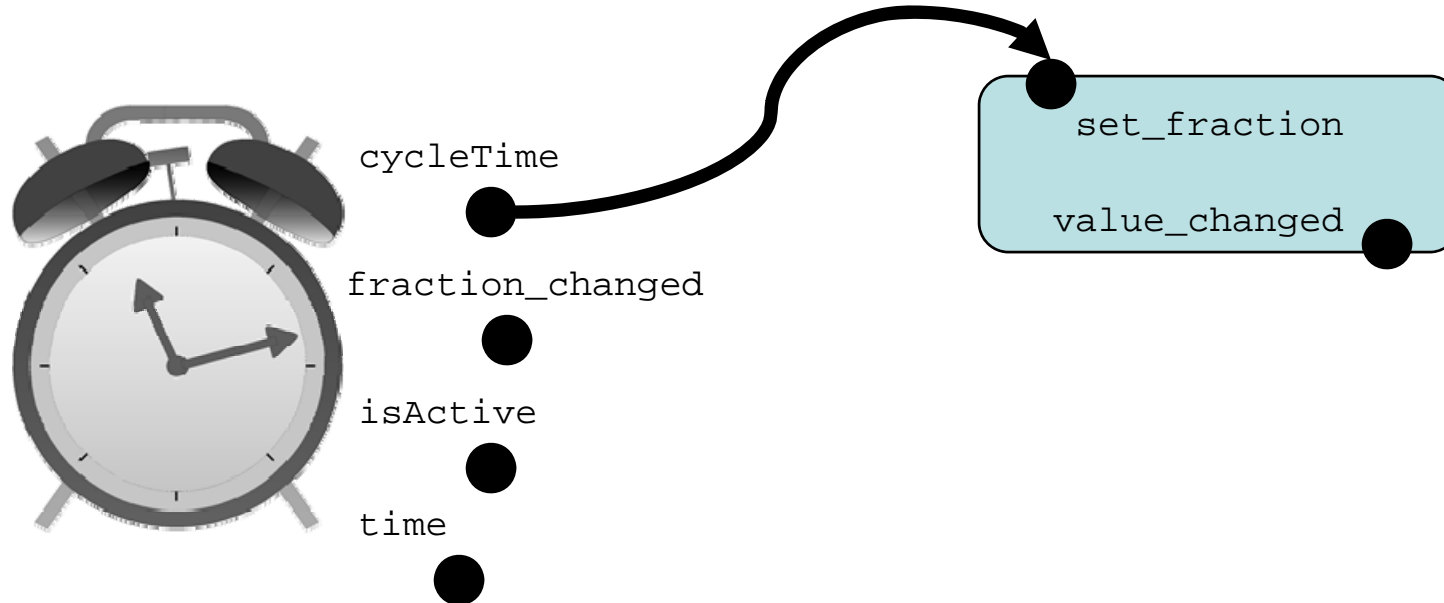


`time`



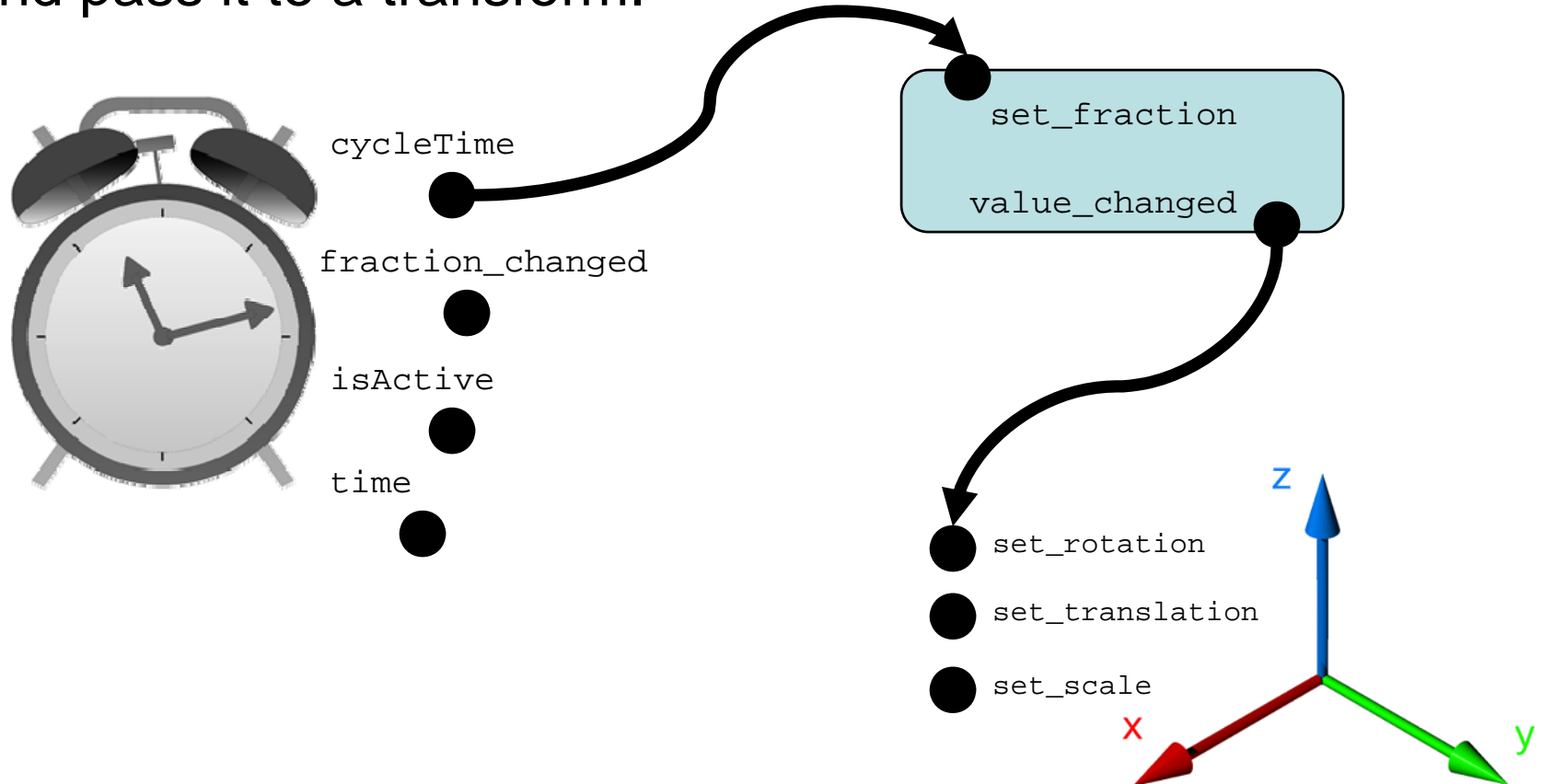
# A little trickier...

You can also track the passage of time using a TimeSensor...  
Convert it to a usable type using an interpolator...



# A little trickier...

You can also track the passage of time using a TimeSensor...  
Convert it to a usable type using an interpolator...  
And pass it to a transform.



# What is ECMAScript?

It's Javascript.