From Art to Engine with Model I/O

Session 610

Nick Porcino, Game Technologies Engineer
Nicholas Blasingame, Game Technologies Engineer

Model I/O

Apple's toolkit for building pipelines

Import and export 3D assets

Geometry, materials, lighting, cameras, voxels

Data format conversions

Processing tools

Model I/O

What's new?

NEW

Improved Importers

Skinned Character Animation

Blend Shapes

Transform Stacks

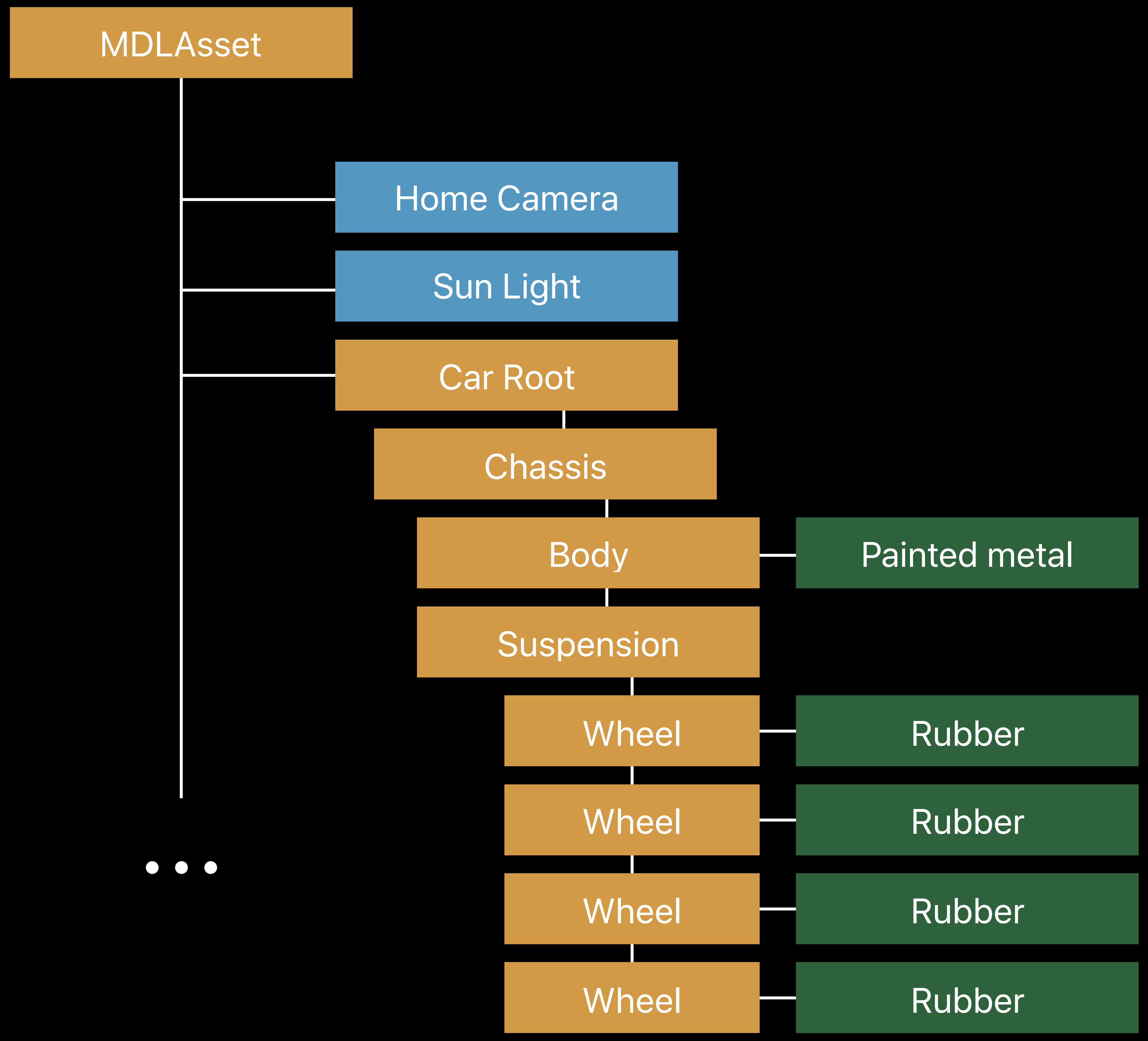
Model I/O

Intuitive Asset Traversal

Format independent graph

Logical

Consistent



Art Assets

Models, materials, animations

Textures

Scenes composed of many files

From Art to Engine

Art asset is like source code

Compiled for an engine

From Art to Engine

UI based tools

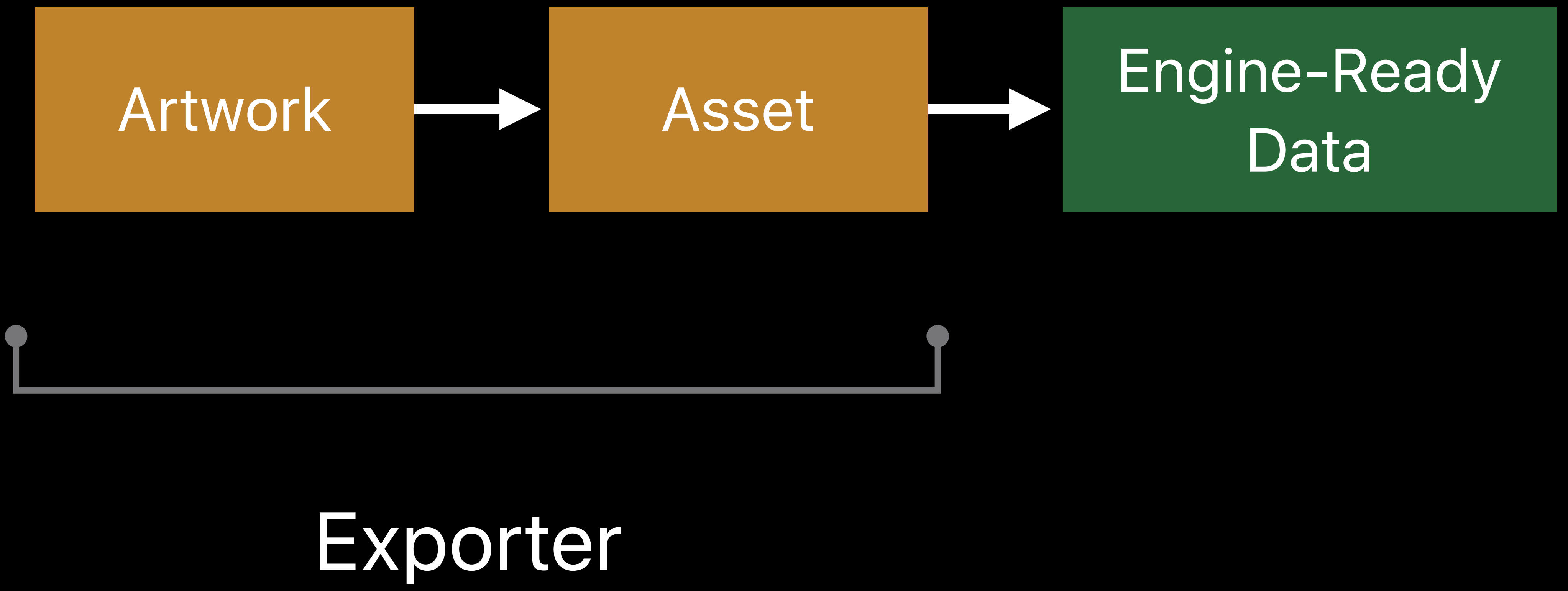
Easy the first few times

Overwhelming during revision

How can the work be scaled?

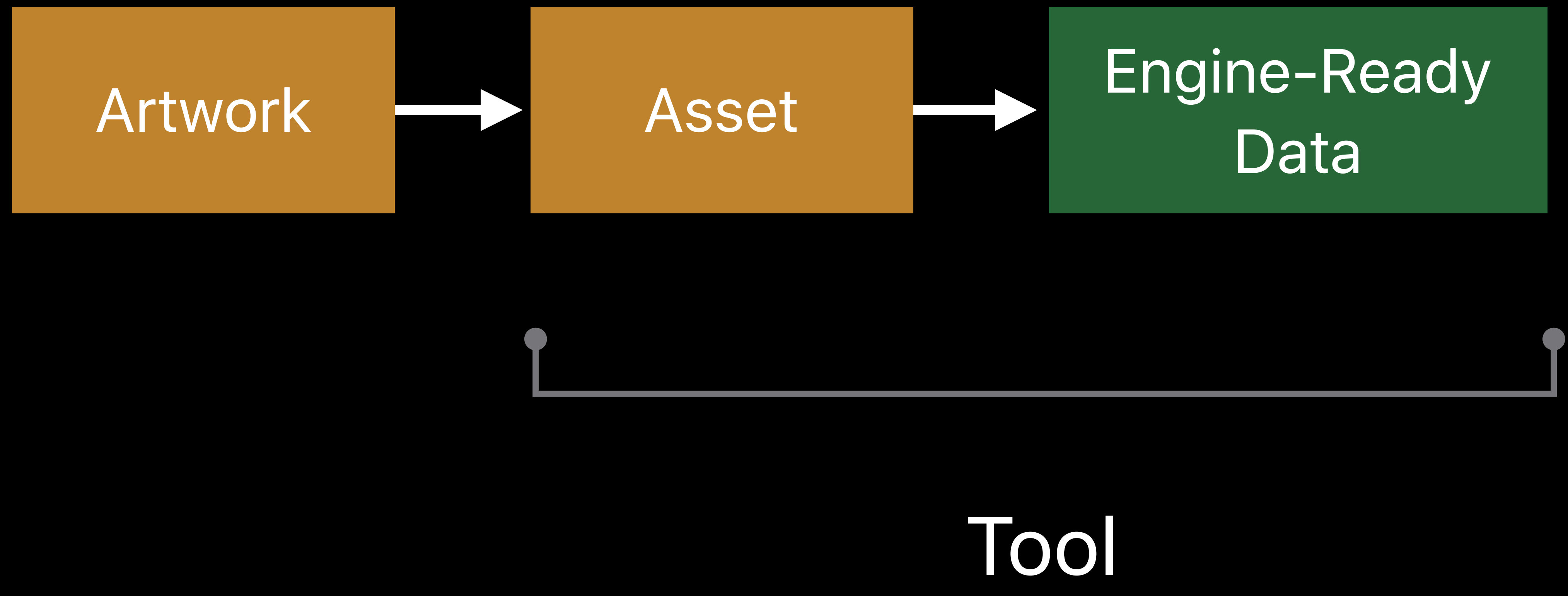
Introducing the Pipeline

Export the art



Introducing the Pipeline

Transform the asset



Introducing the Pipeline

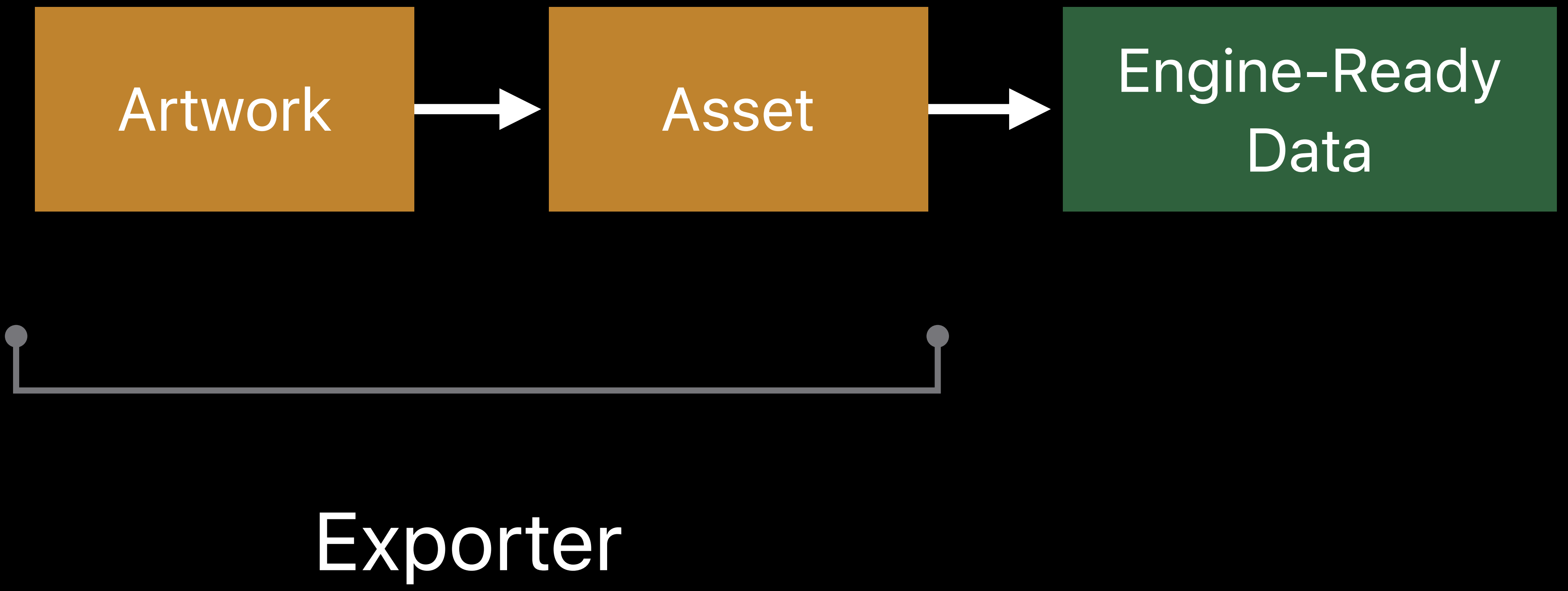
Load engine-ready data

$$\begin{tikzcd}\text{Artwork} \arrow[r] & \text{Asset} \arrow[r] & \text{Engine-Ready Data}\end{tikzcd}$$

Engine

Introducing the Pipeline

Export the art

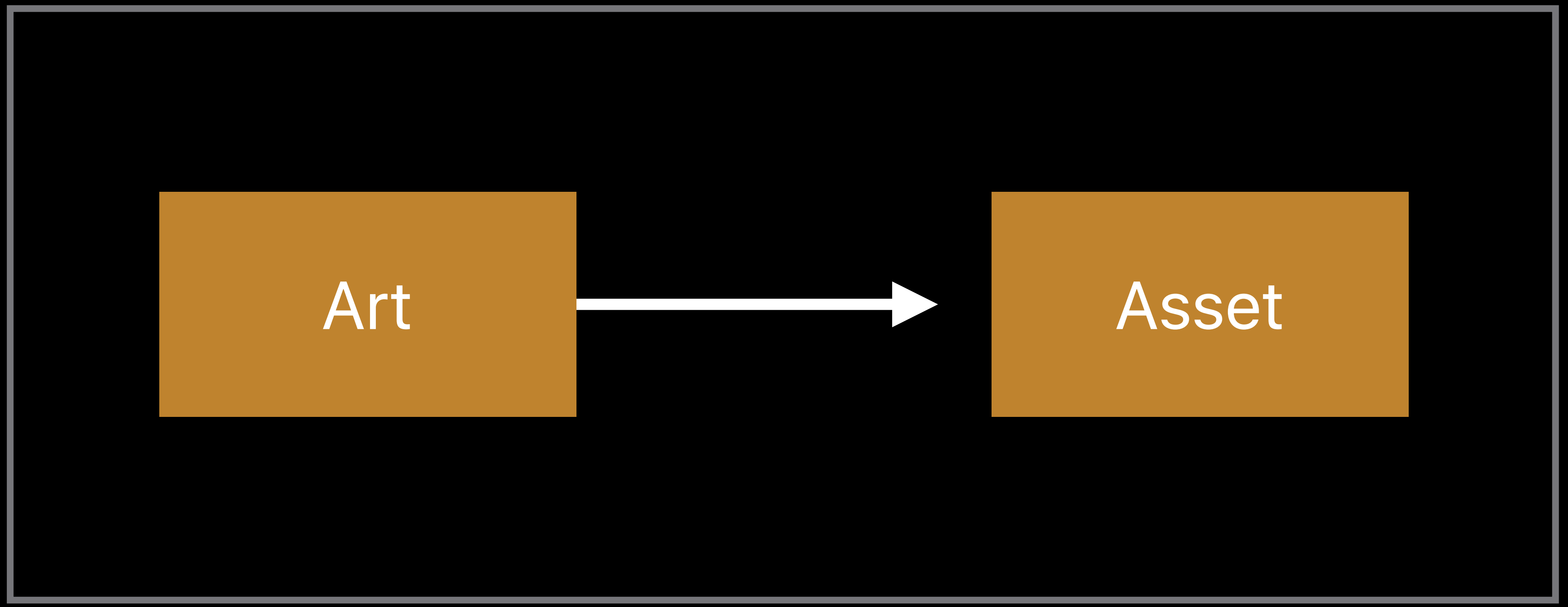


Introducing the Pipeline

Exporter

Maya

- Asset Exporter
- Complex hierarchies of files
- Export script in the sample

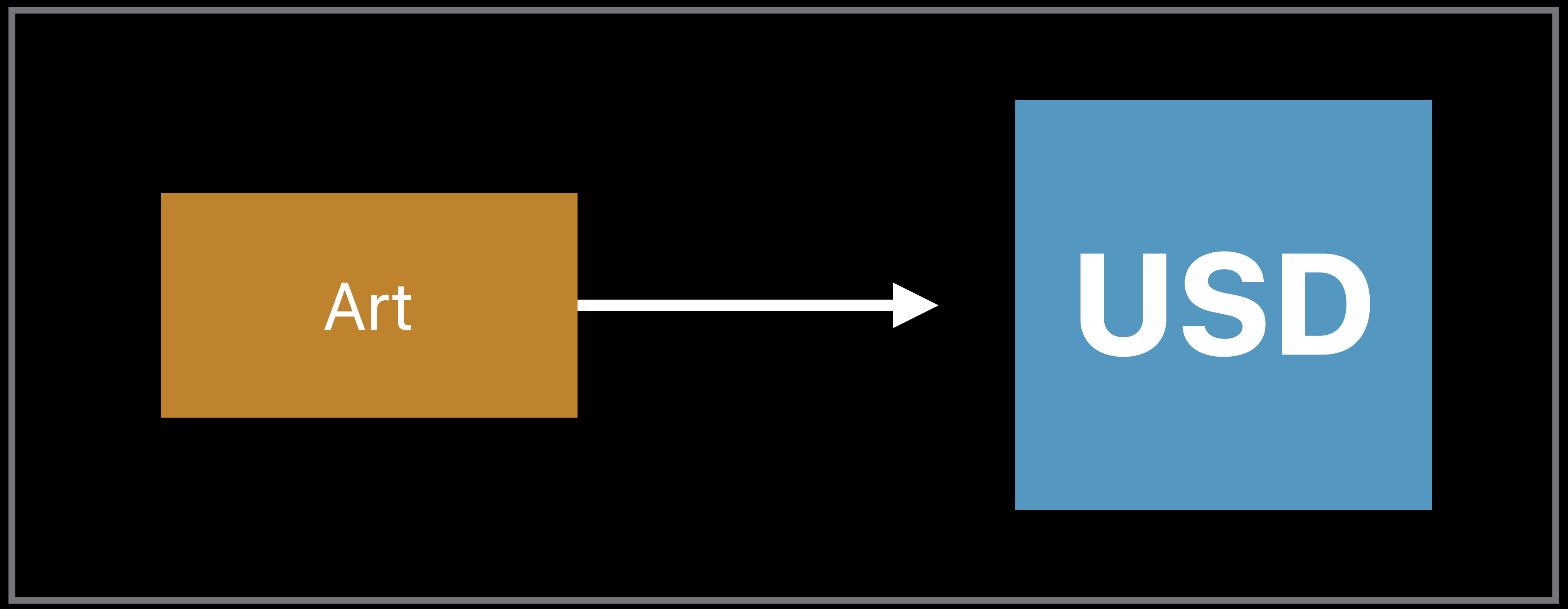


Introducing the Pipeline

Exporter

Maya

- Asset Exporter
- Complex hierarchies of files
- Export script in the sample

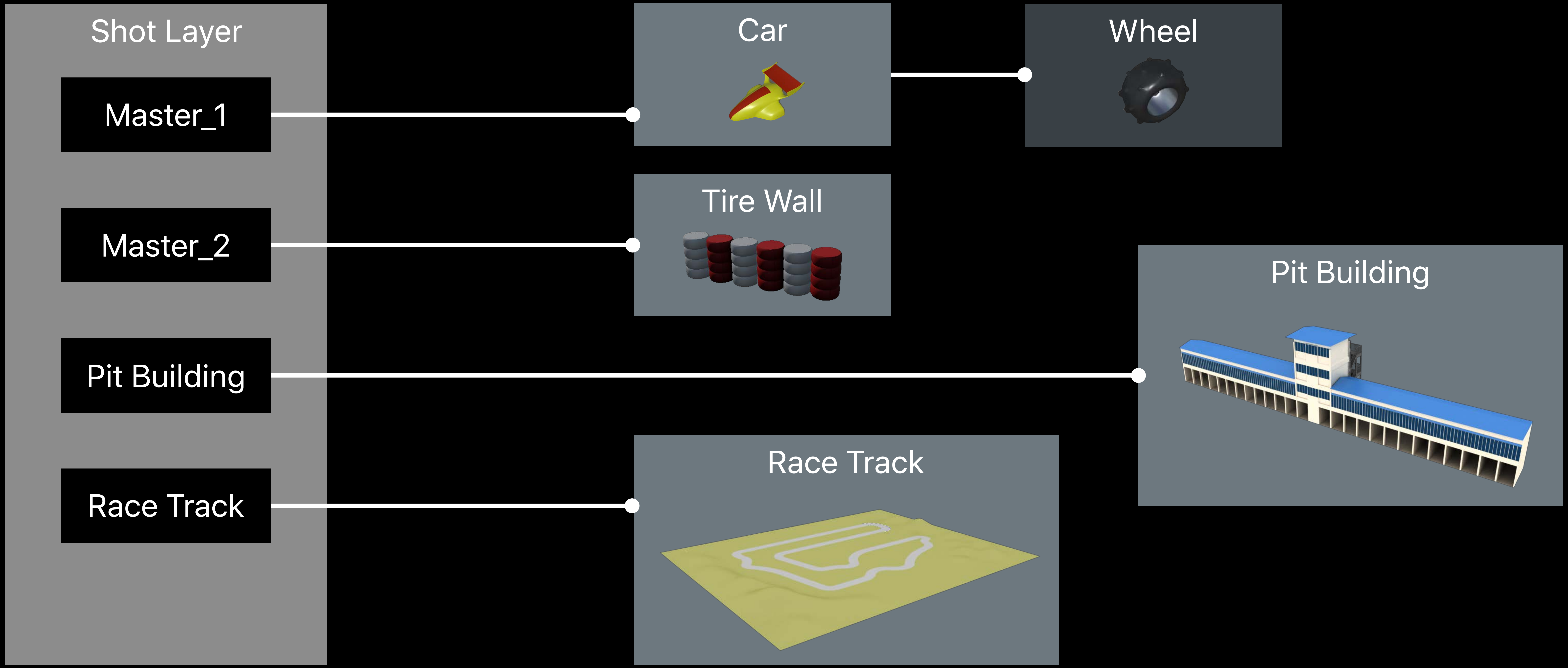


Universal Scene Description

http://openusd.org

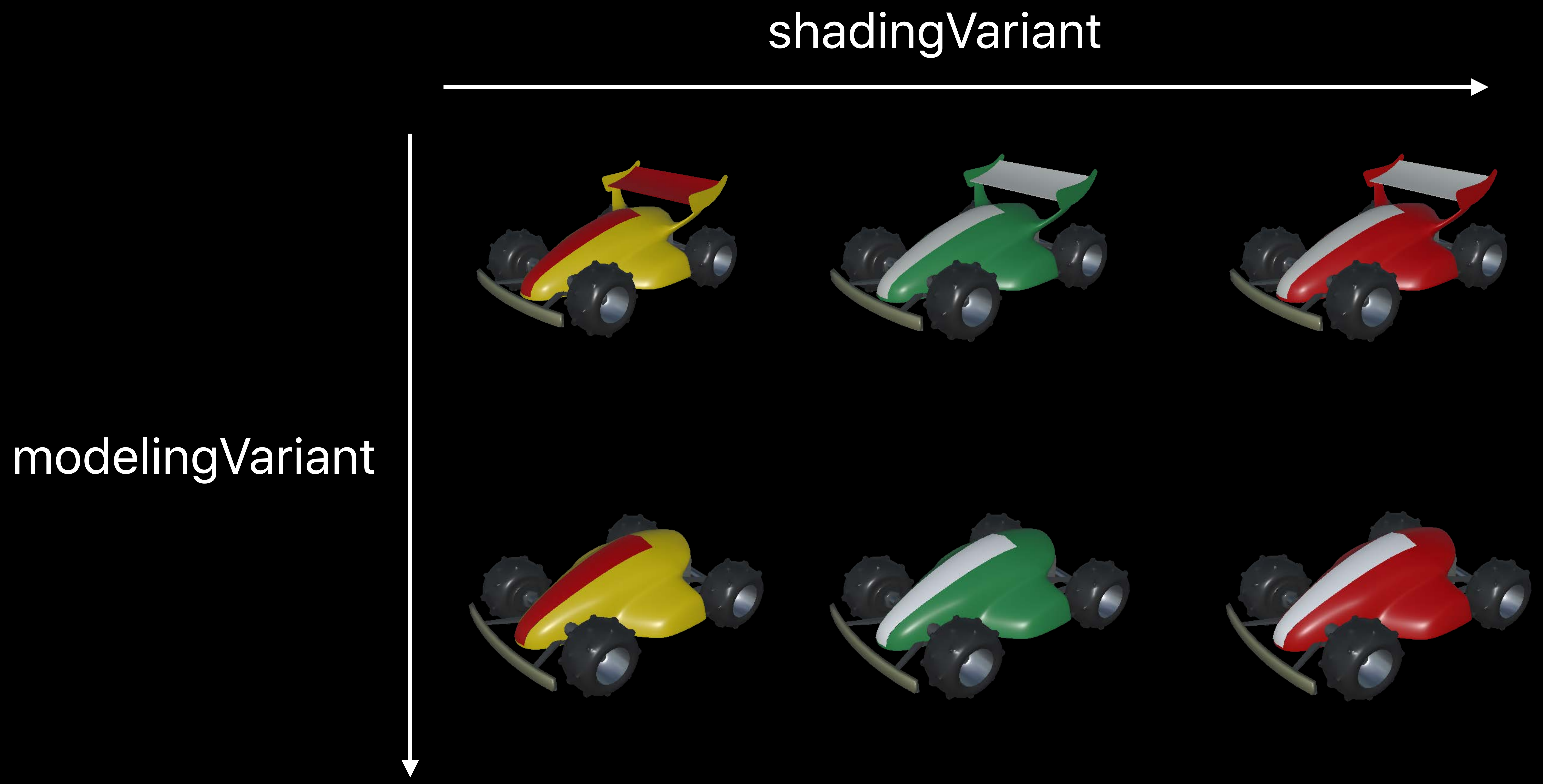
Universal Scene Description

Composition



Universal Scene Description

Classes, Variations, and Overrides



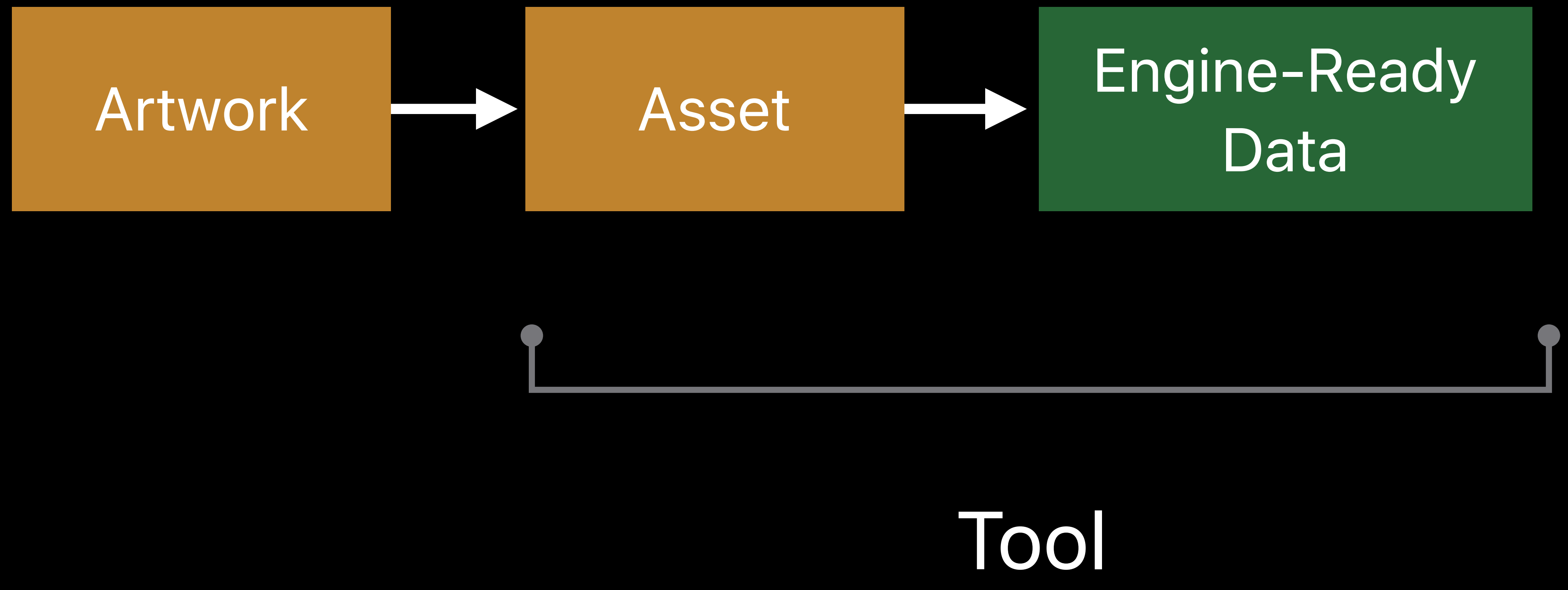
Universal Scene Description

Powerful Text Format, Fast Binary

```
over "World"
{
    over "anim"
    {
        over "chars"
        {
            def "Car"
            (
                add references = @chars/car.usd@</Car>
            )
            {
                color3f displayColor = (0.9, 0, 0)
            }
        }
    }
}
```

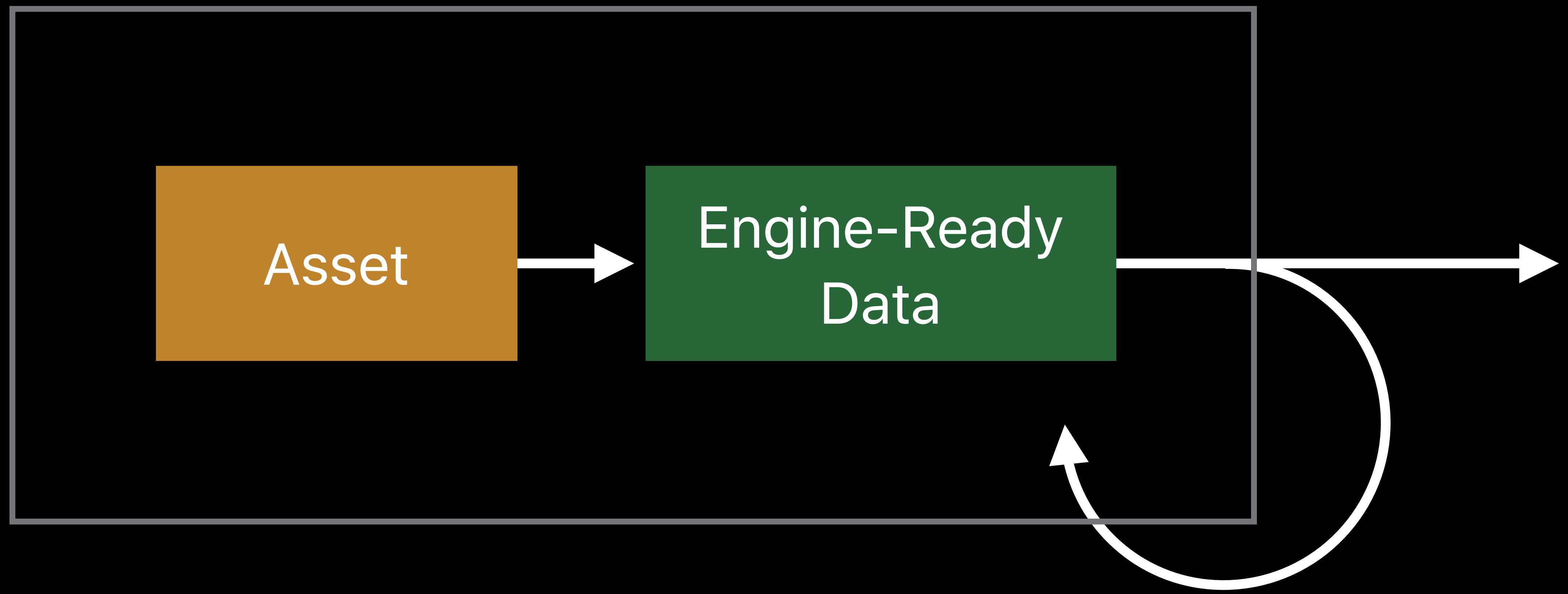
Introducing the Pipeline

Scriptable command line tool



Introducing the Pipeline

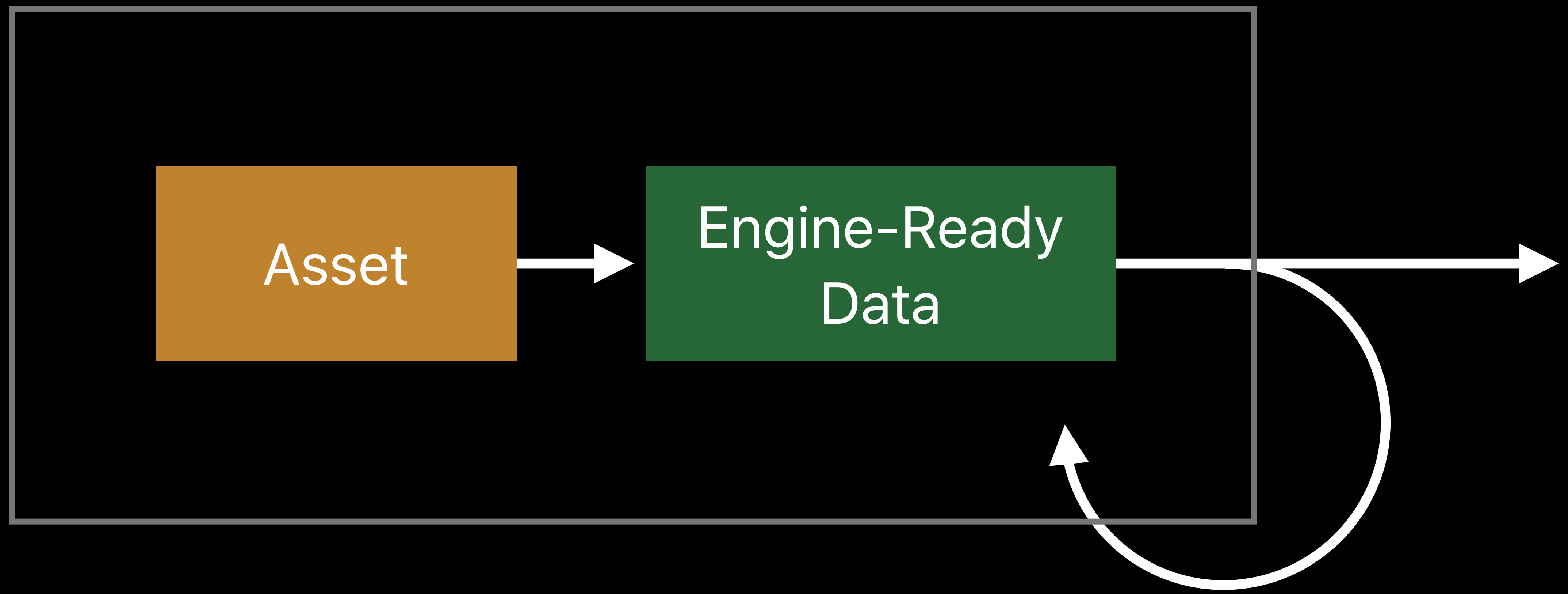
Scriptable command line tool



Introducing the Pipeline

Scriptable command line tool

Repeatable

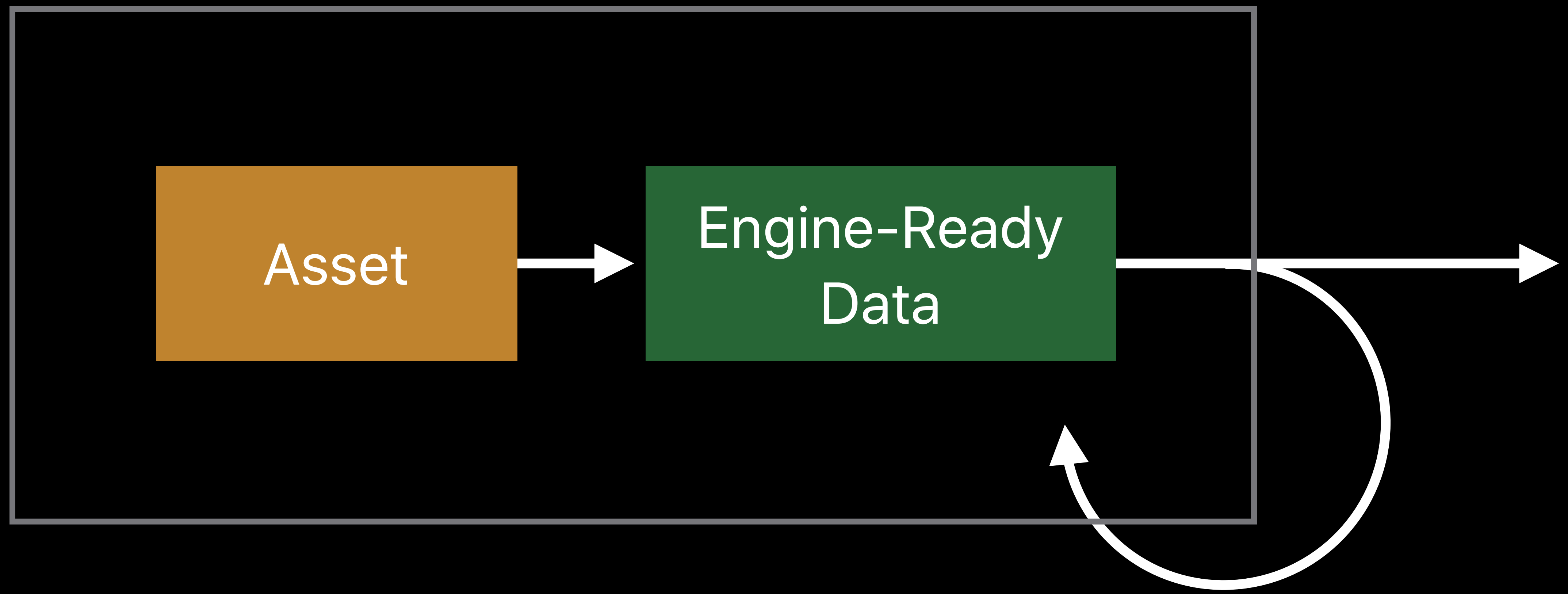


Introducing the Pipeline

Scriptable command line tool

Repeatable

Consistent



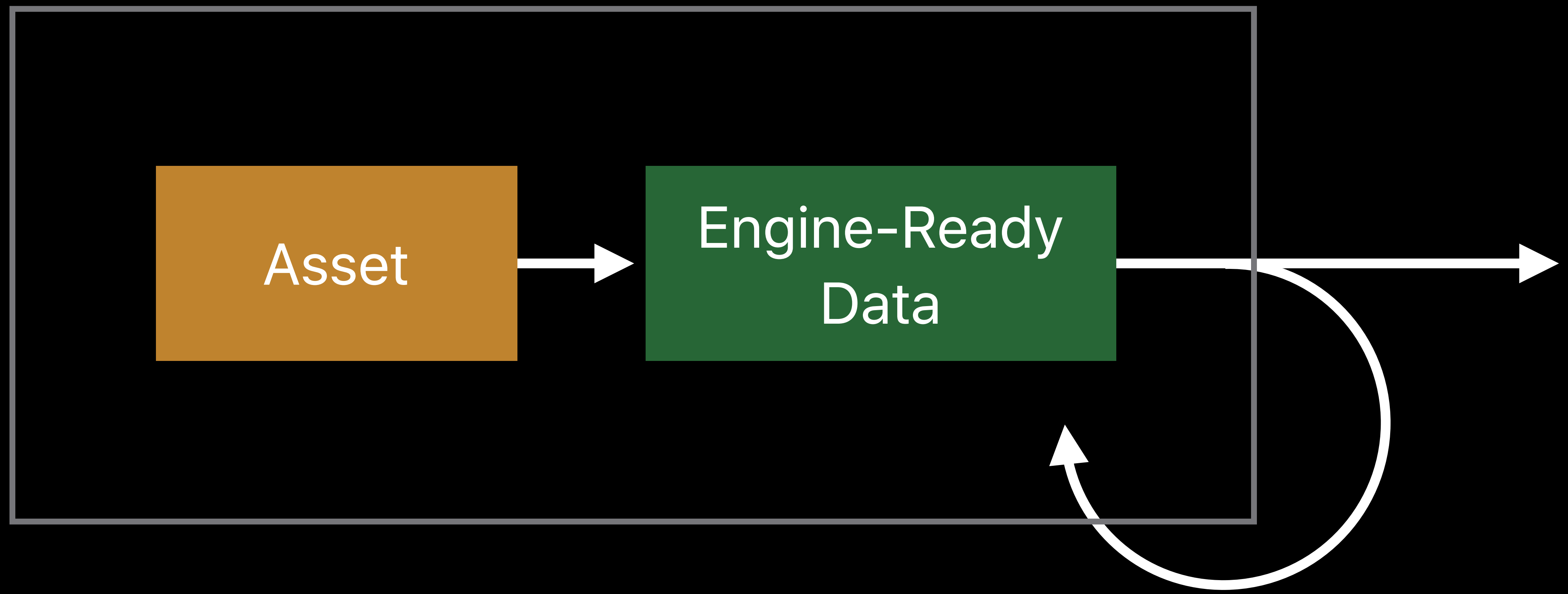
Introducing the Pipeline

Scriptable command line tool

Repeatable

Consistent

Scriptable



Introducing the Pipeline

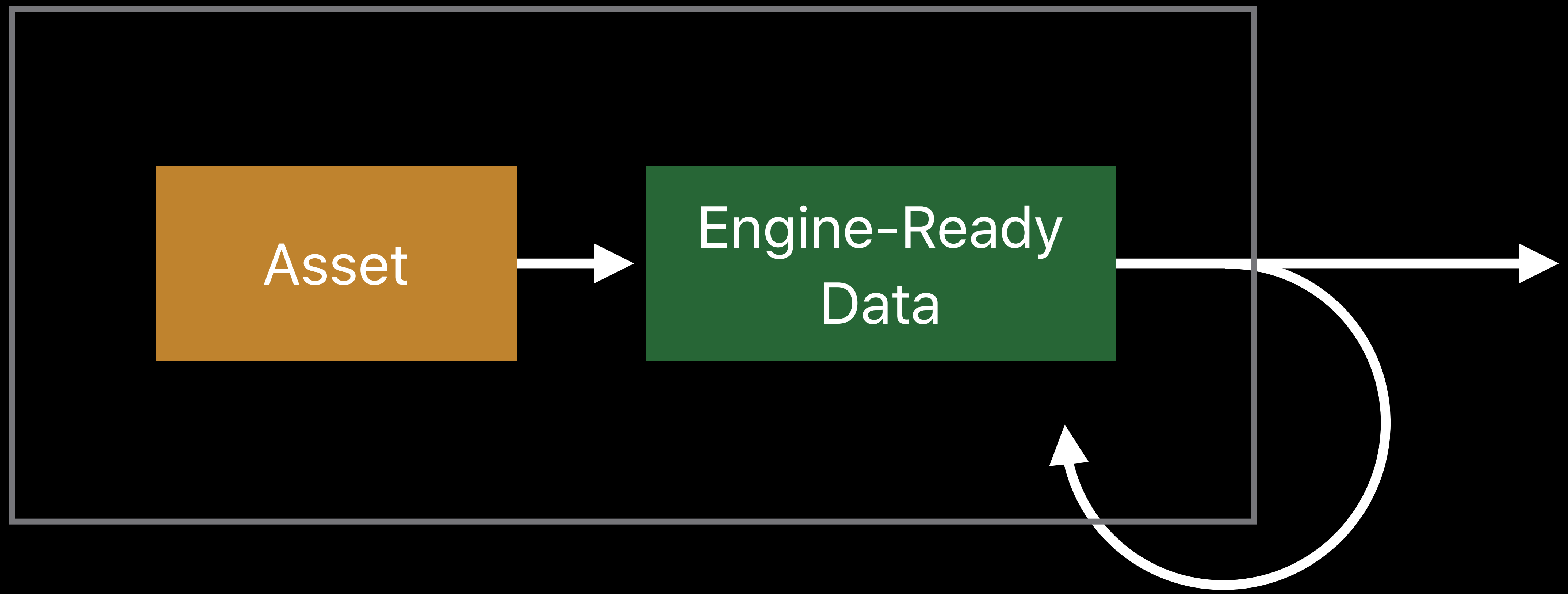
Scriptable command line tool

Repeatable

Consistent

Scriptable

Scalable



Introducing the Pipeline

Scriptable command line tool

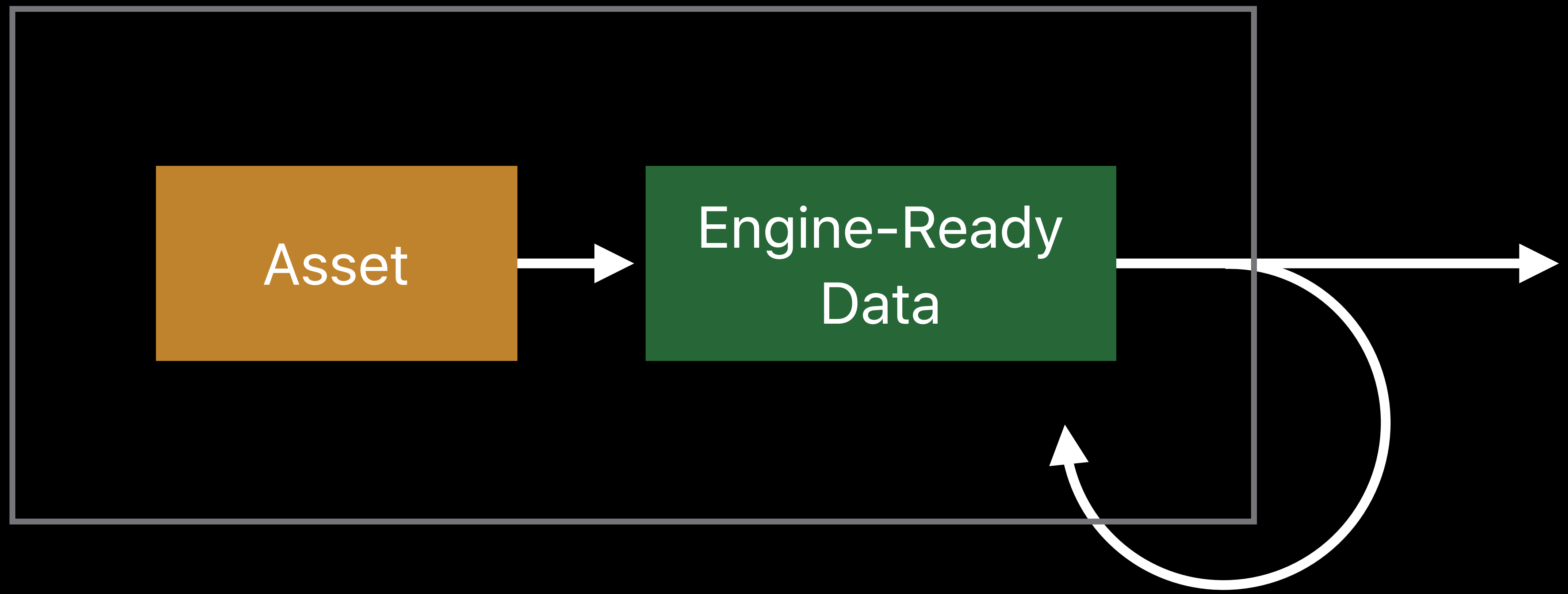
Repeatable

Consistent

Scriptable

Scalable

Composable

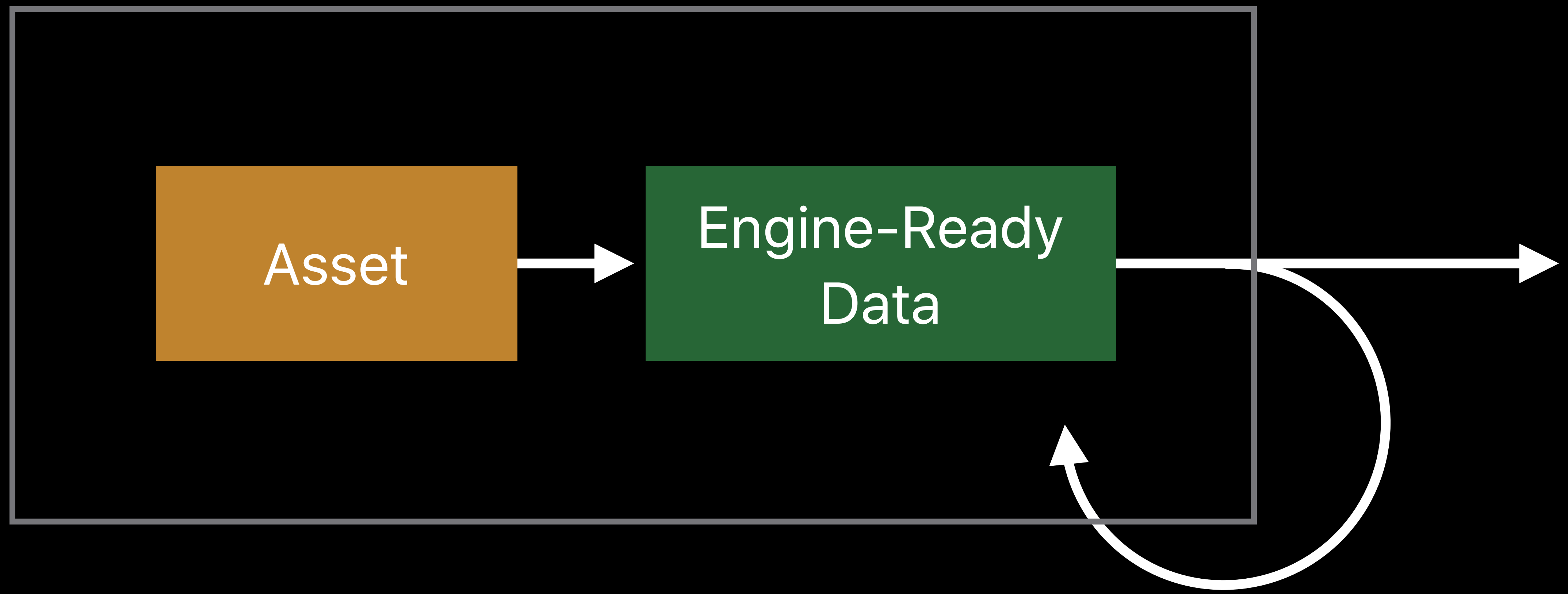


Introducing the Pipeline

The sample

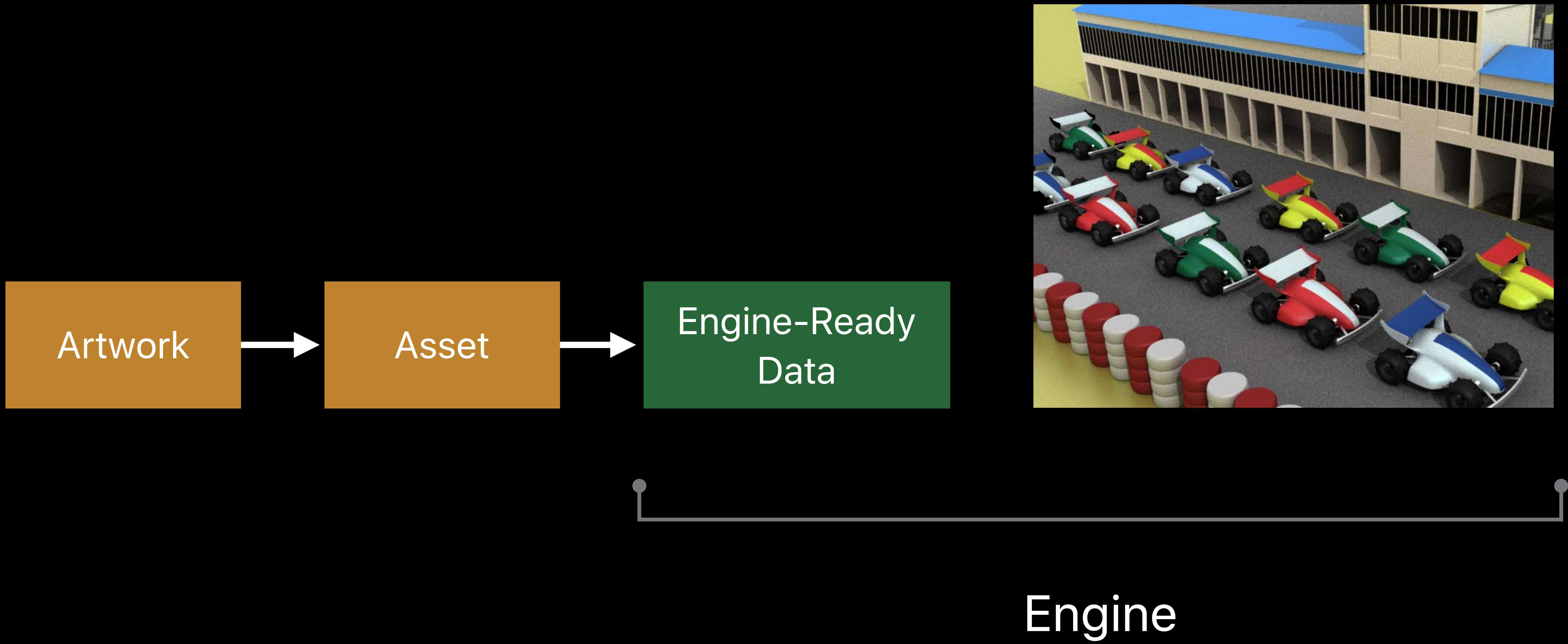
Demonstrates principles

- Simplified data
- Uncompressed
- Good jumping-off point



Introducing the Pipeline

Game engine



Game Engine

Simple renderer

Single-pass forward renderer

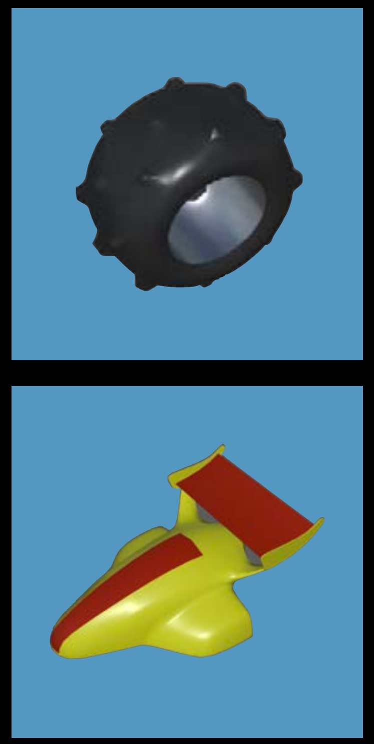
Physically-based shader

Mesh instancing

Skinned and animated meshes

Multiple materials

Render Loop



Set Transform Buffer

Set Skinning Data

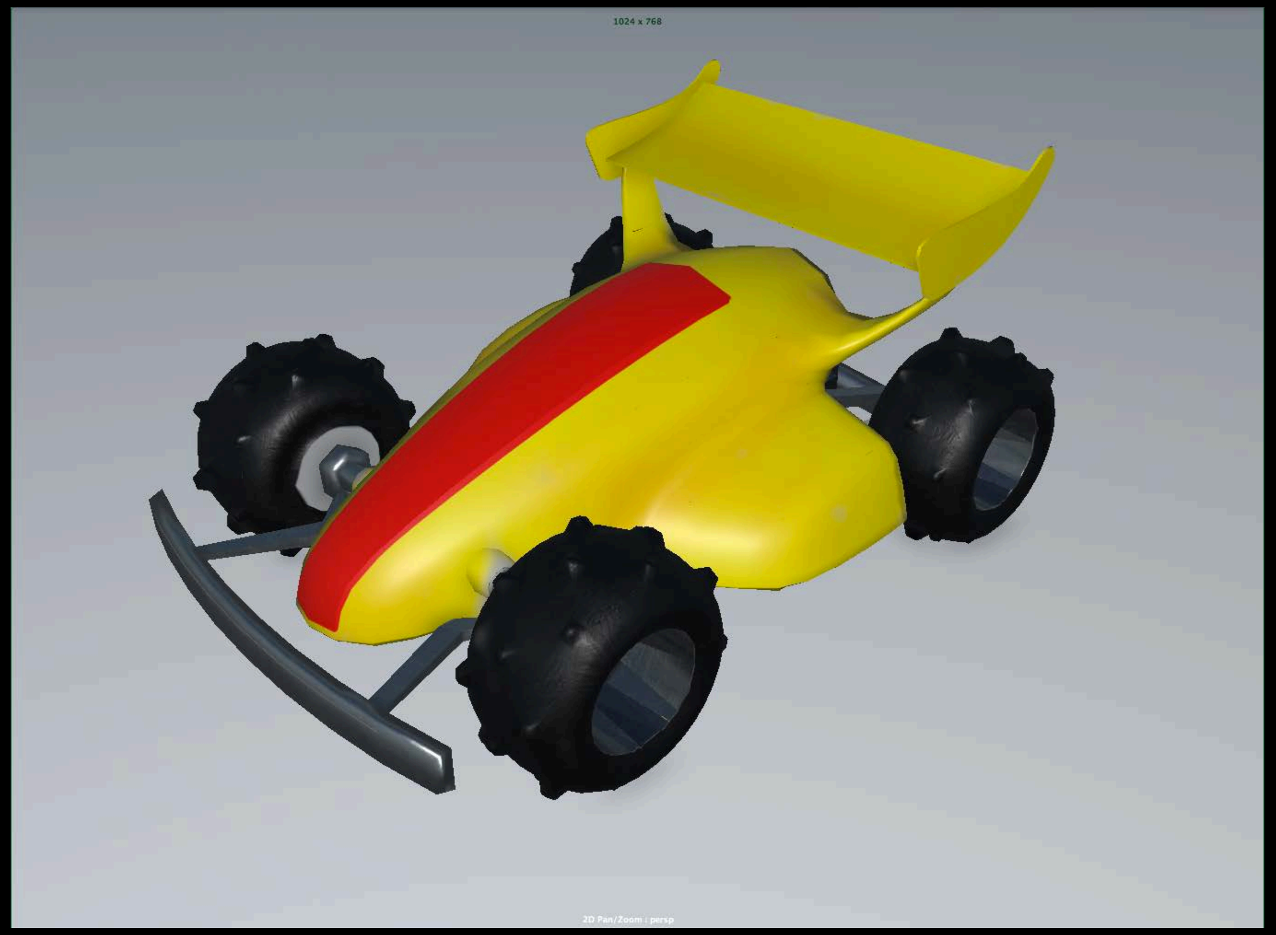
Set Vertex Buffer

Set Pipeline State

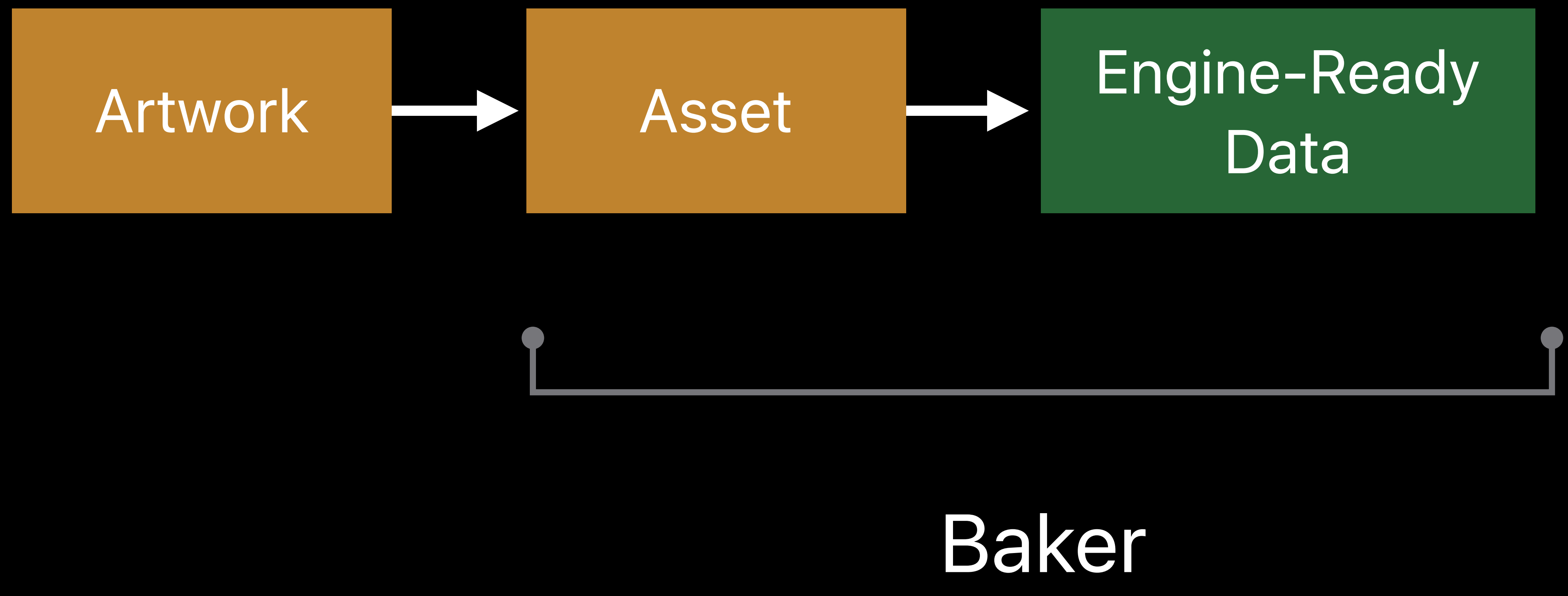
Set Material Uniforms

Set Fragment Textures

Draw Indexed Primitive



The Pipeline



Baking Operations

1. Geometry + Transforms
2. Texture Paths + Materials
3. Instancing Data
4. Transform Animation
5. Skinning + Character Animation

Baking Operations

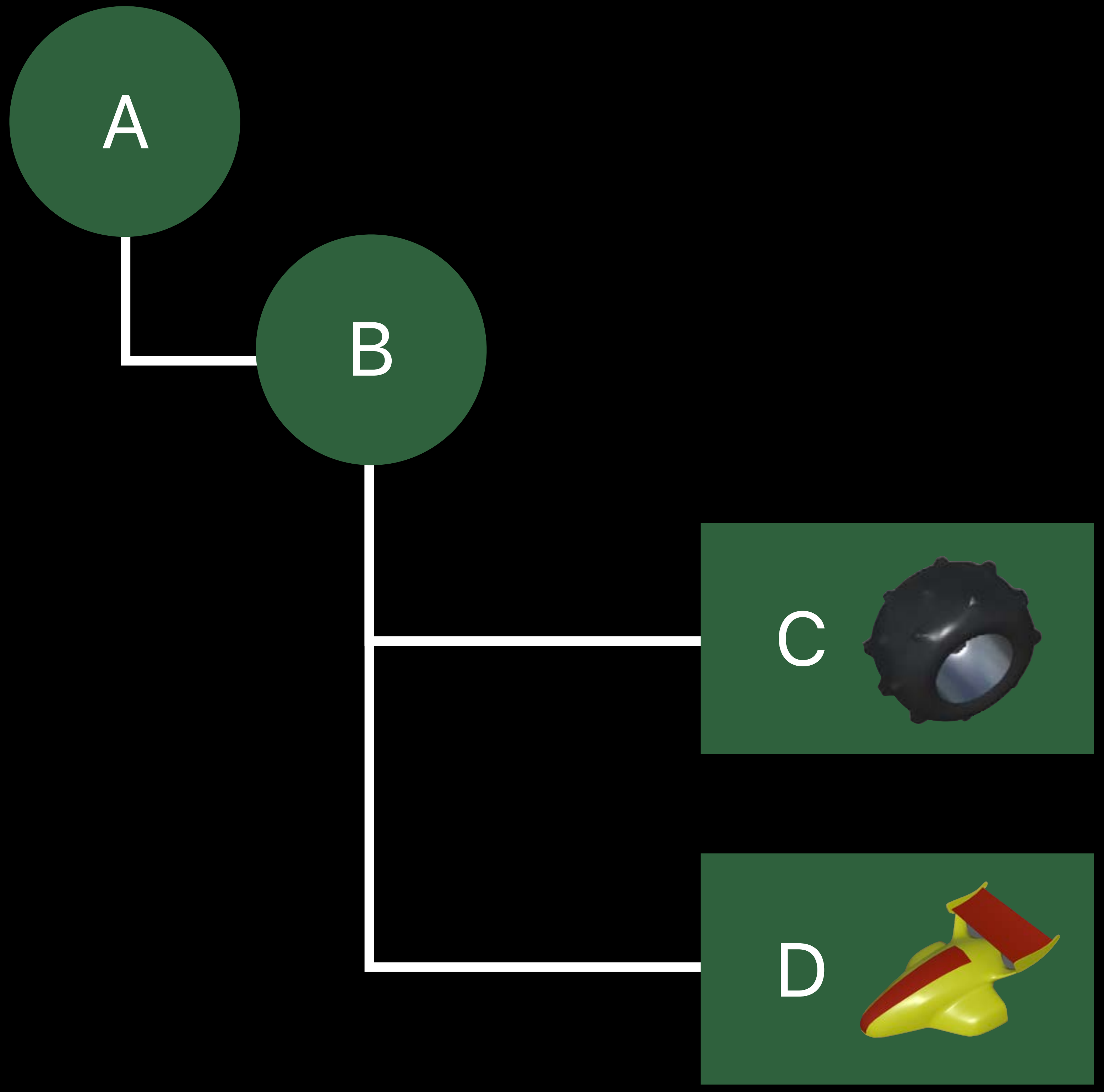
1. Geometry + Transforms
2. Texture Paths + Materials
3. Instancing Data
4. Transform Animation
5. Skinning + Character Animation

Geometry + Transform

A simple scene graph

Transform Hierarchy

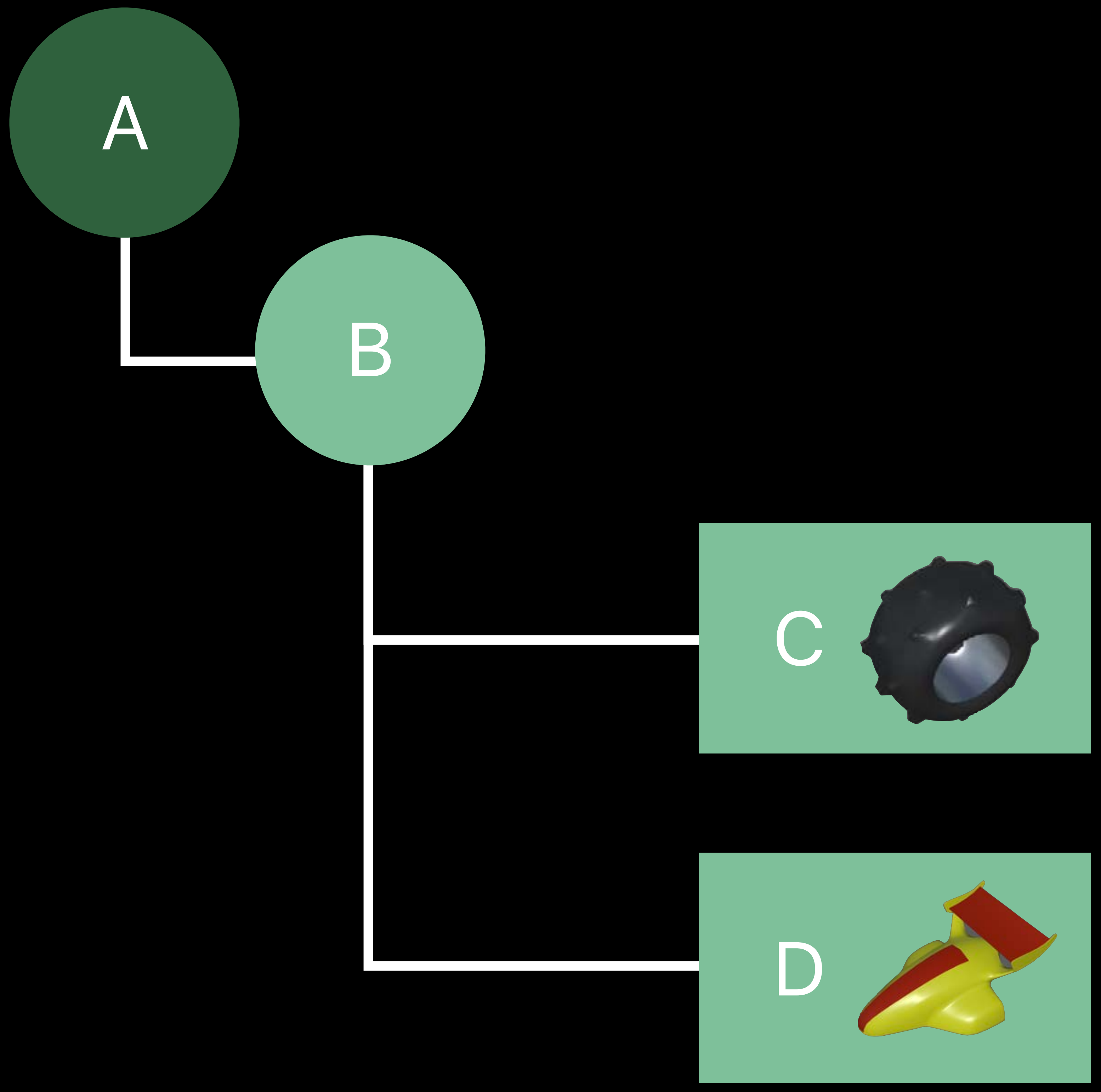
- Tree of nodes
- Meshes and transform objects



Geometry + Transform

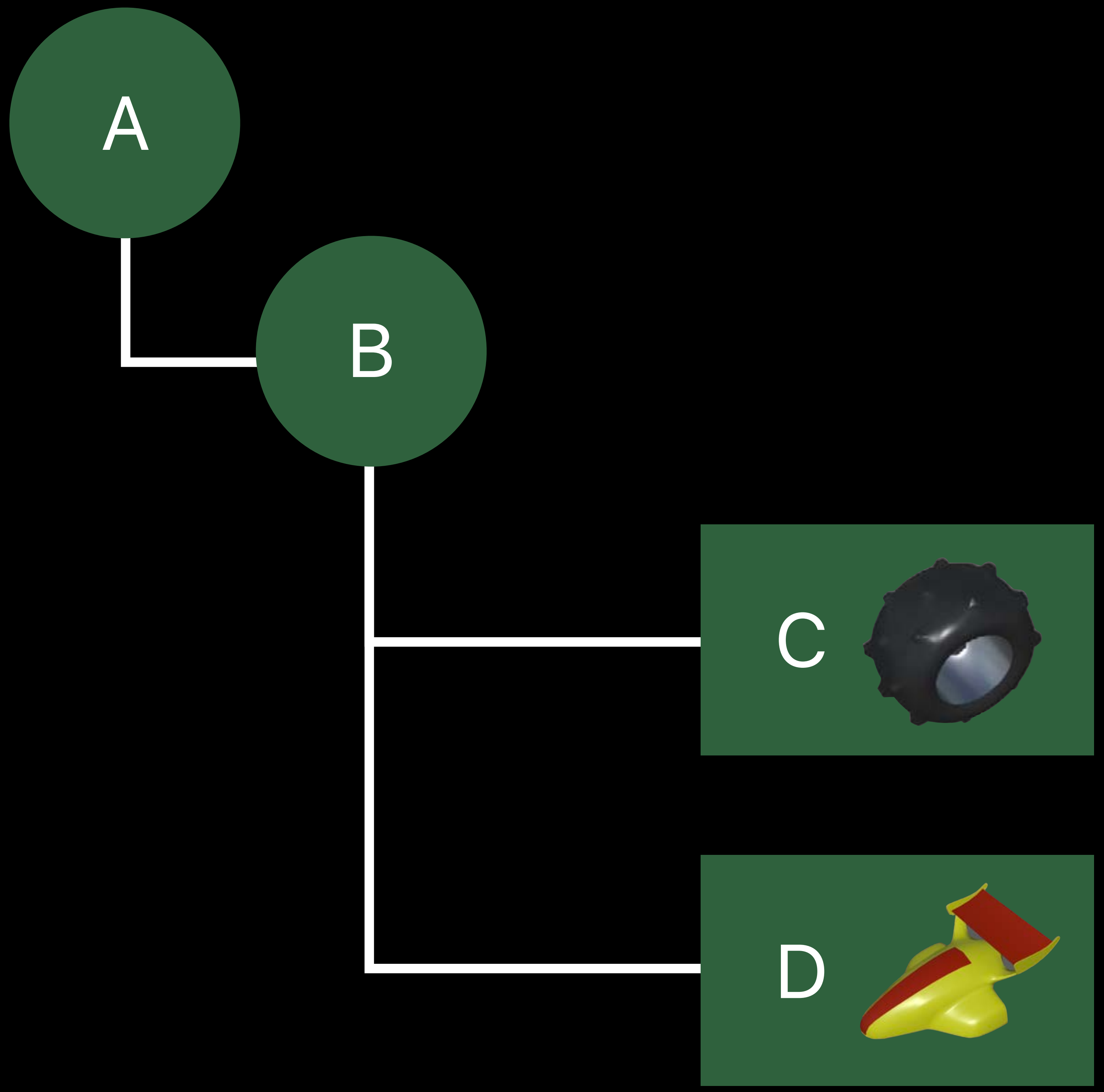
Transform hierarchy

Moving the parent will move the children

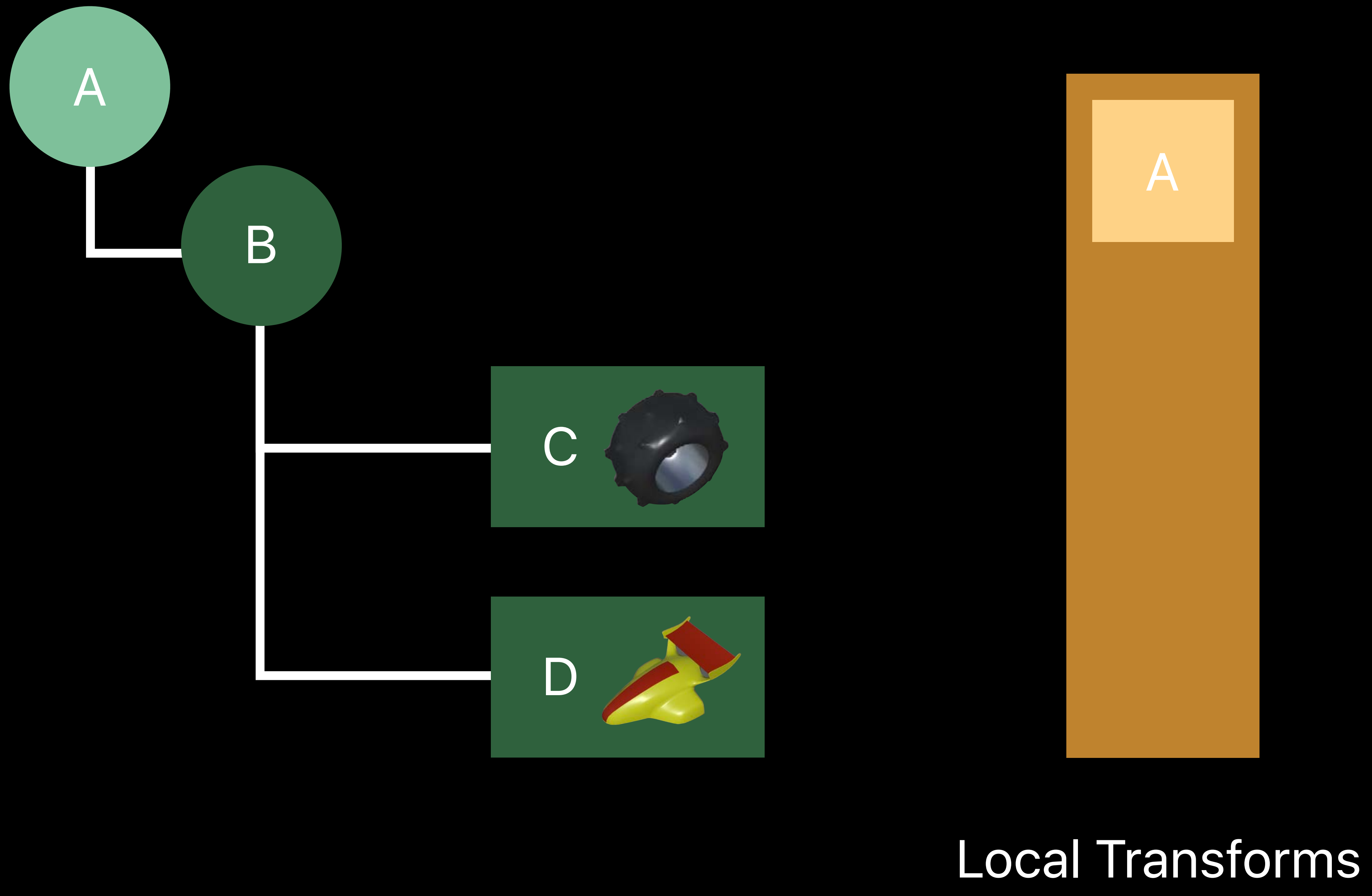


Geometry + Transform

Compactly encode transform hierarchy

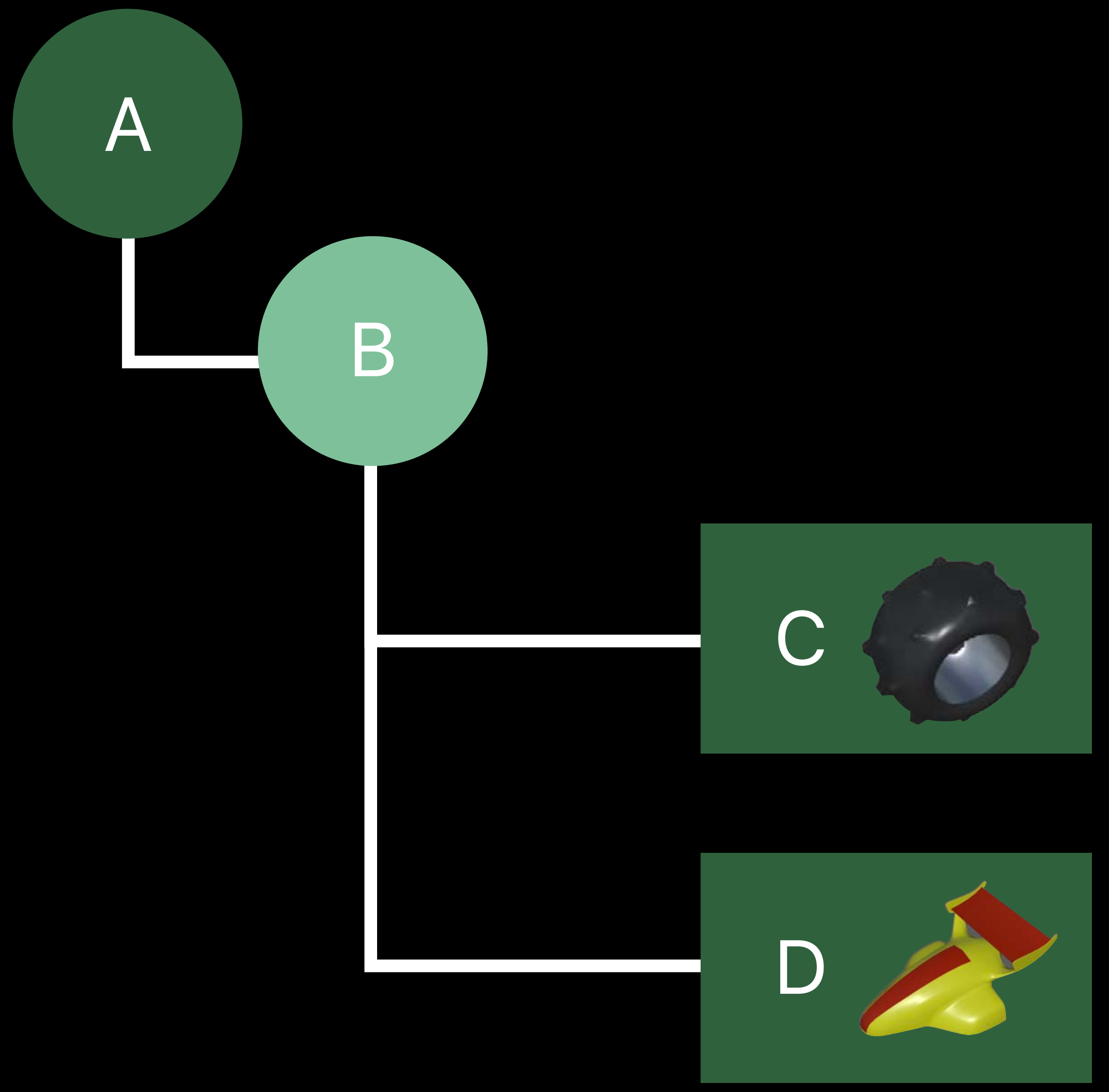


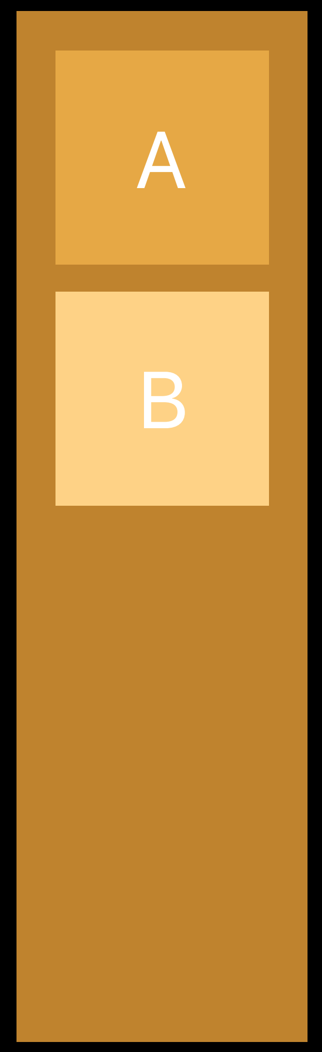
Geometry + Transform
Array of local transforms



Geometry + Transform

Array of local transforms

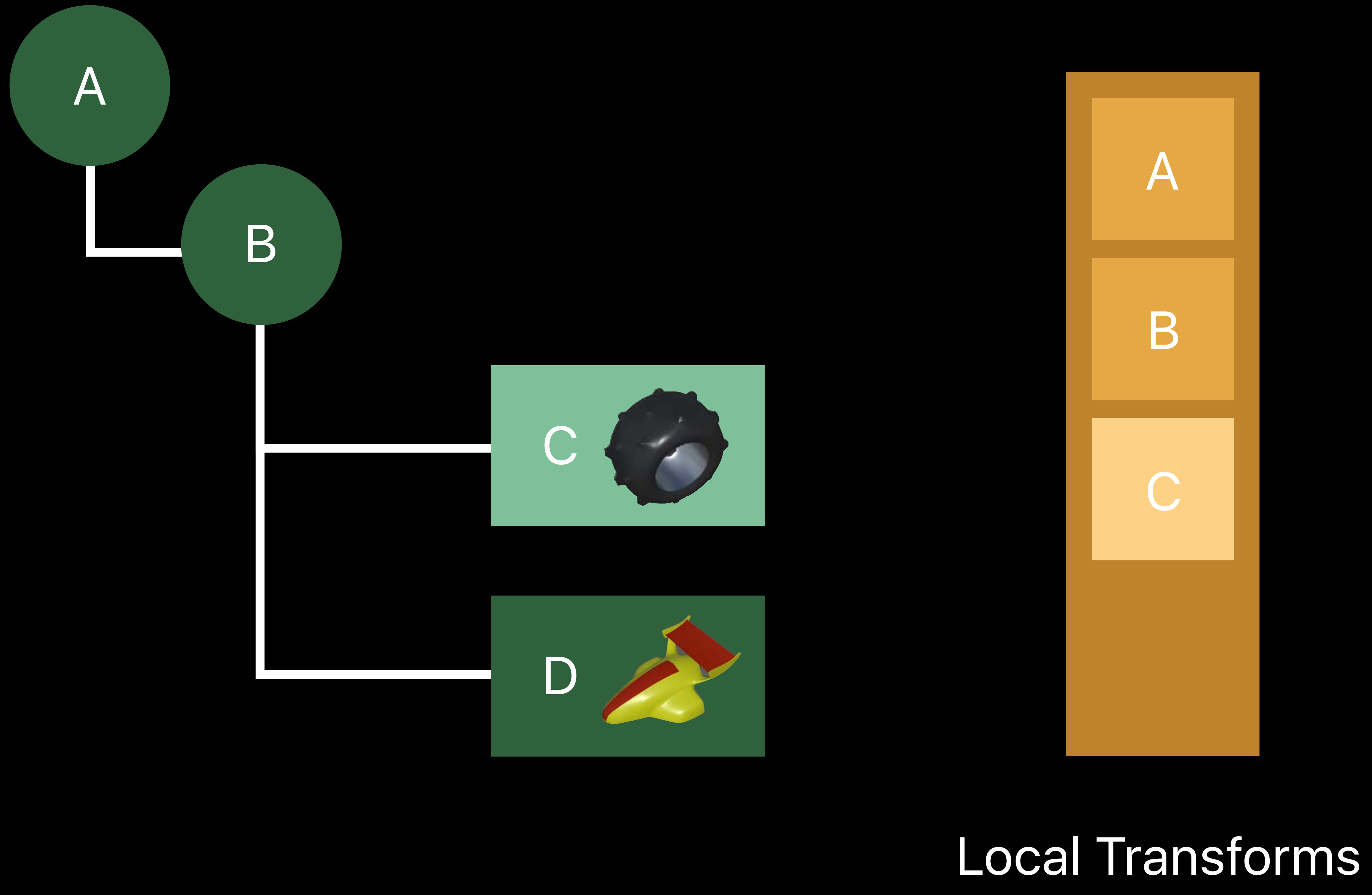




Local Transforms

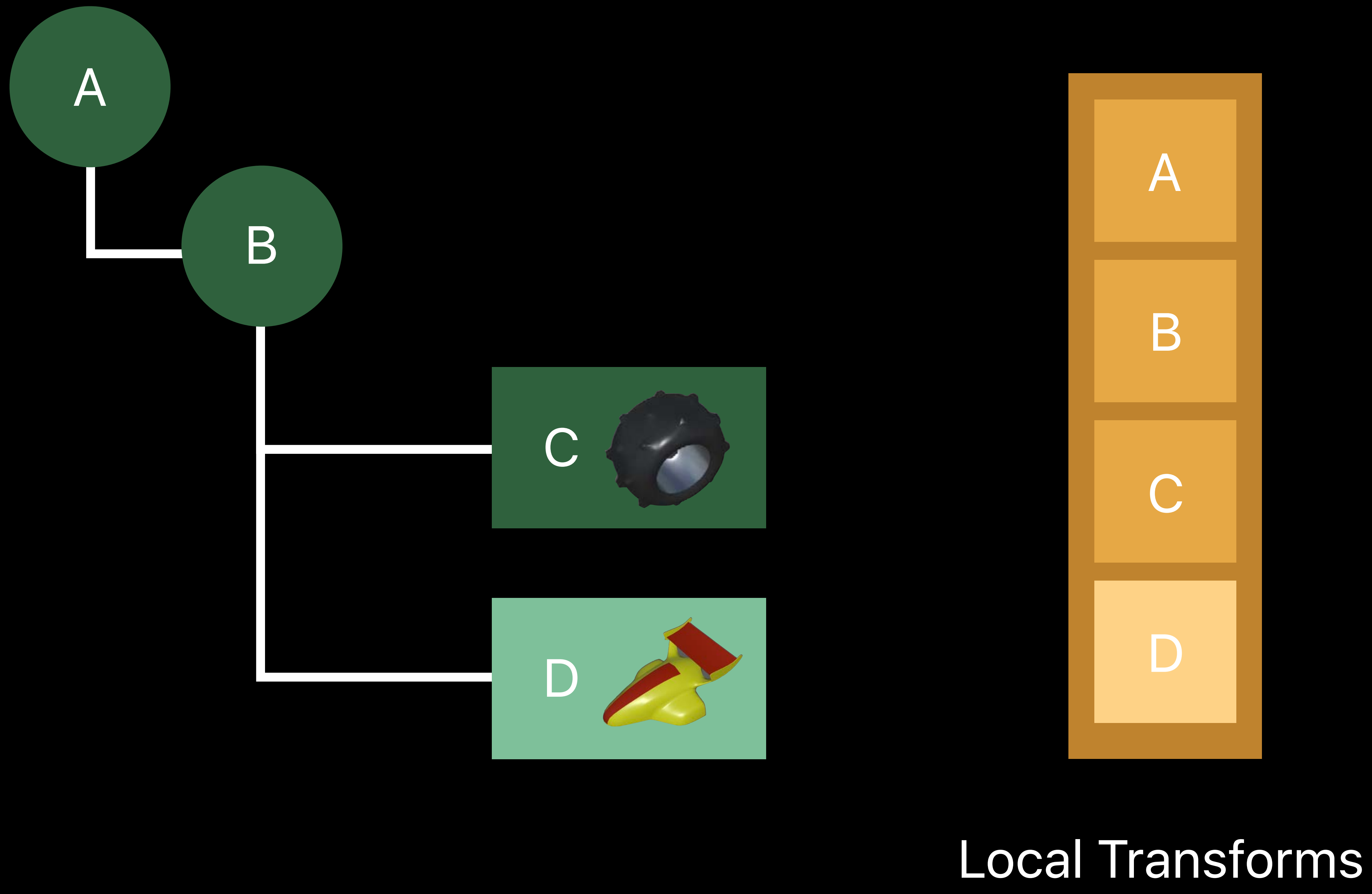
Geometry + Transform

Array of local transforms



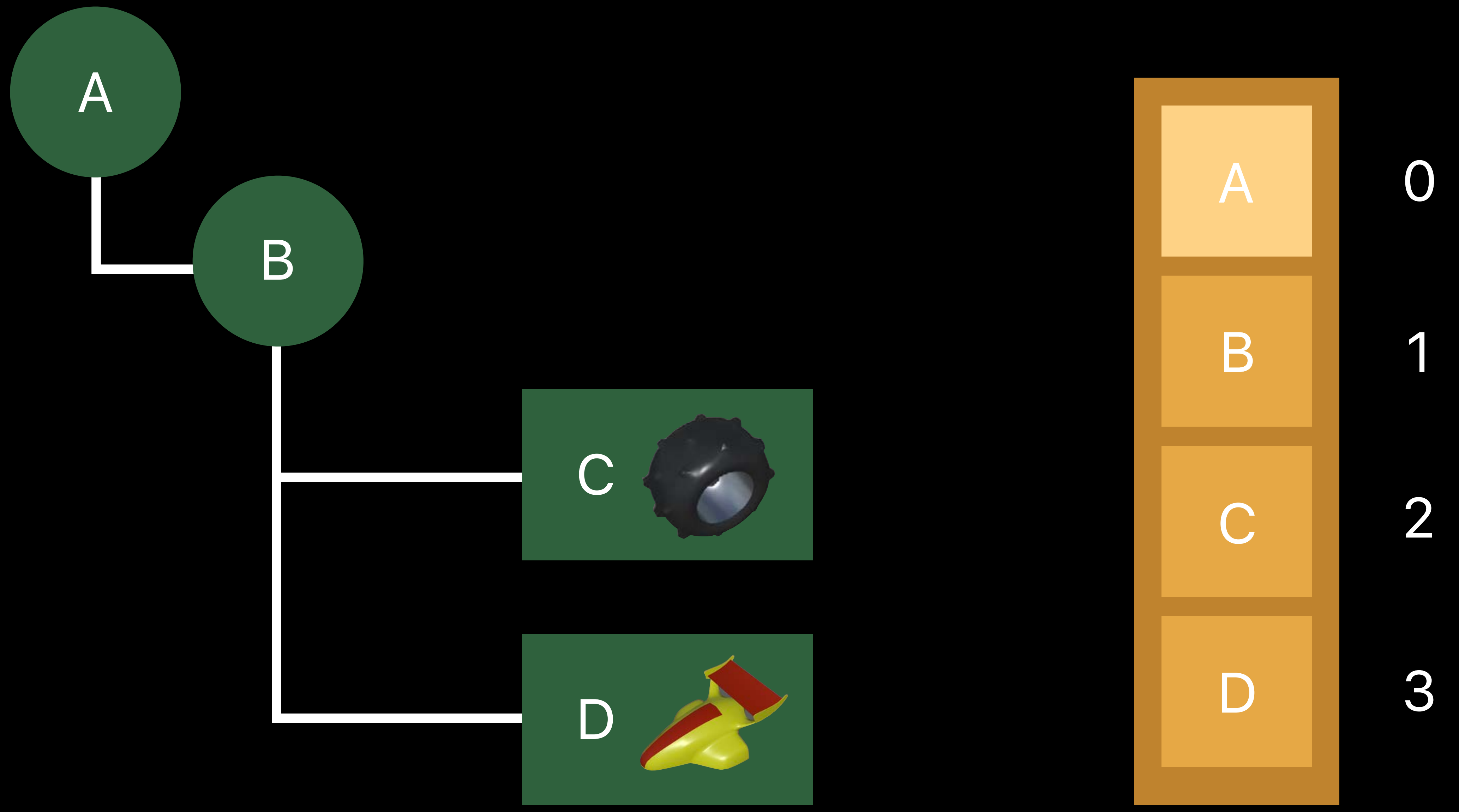
Geometry + Transform

Array of local transforms



Geometry + Transform

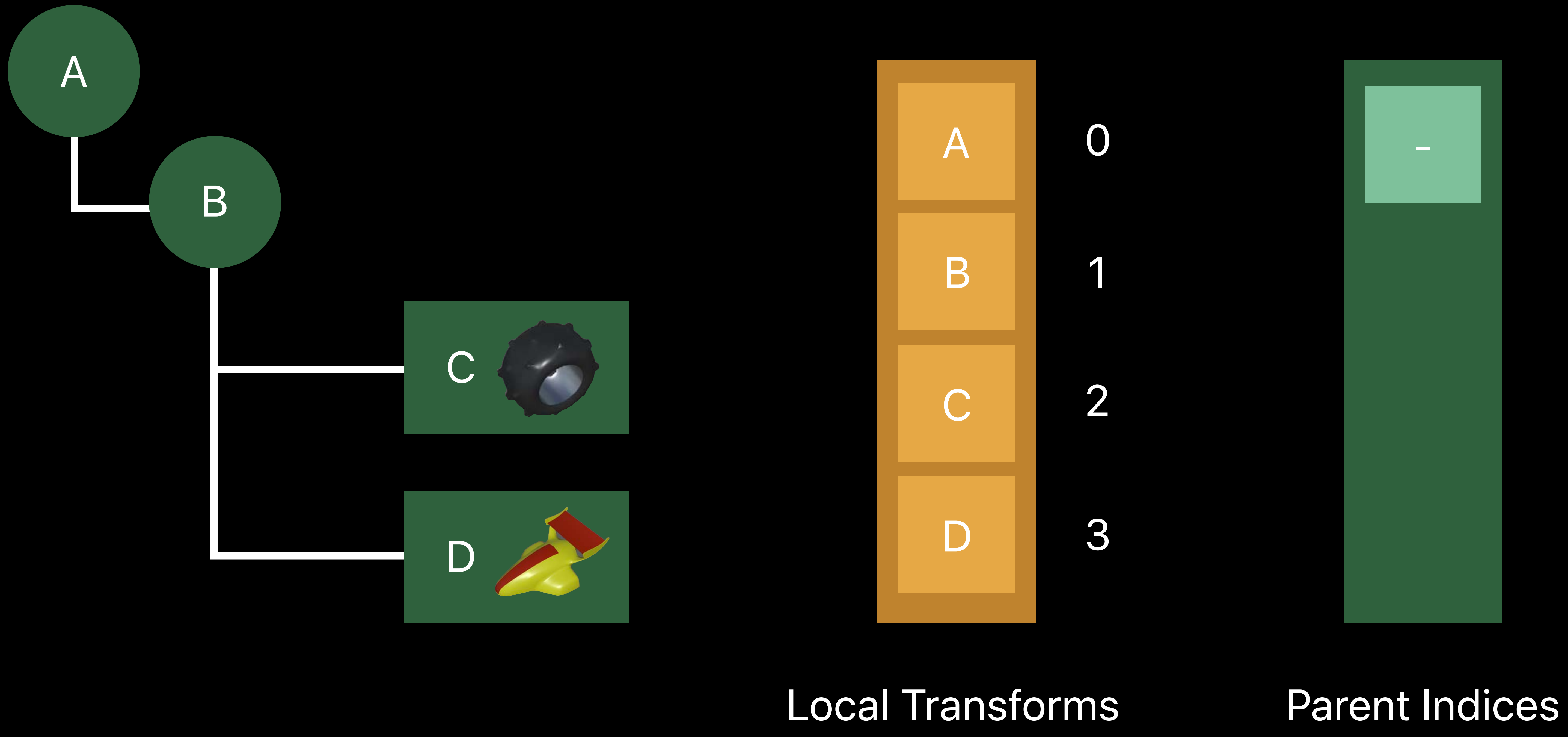
Assign indices



Local Transforms

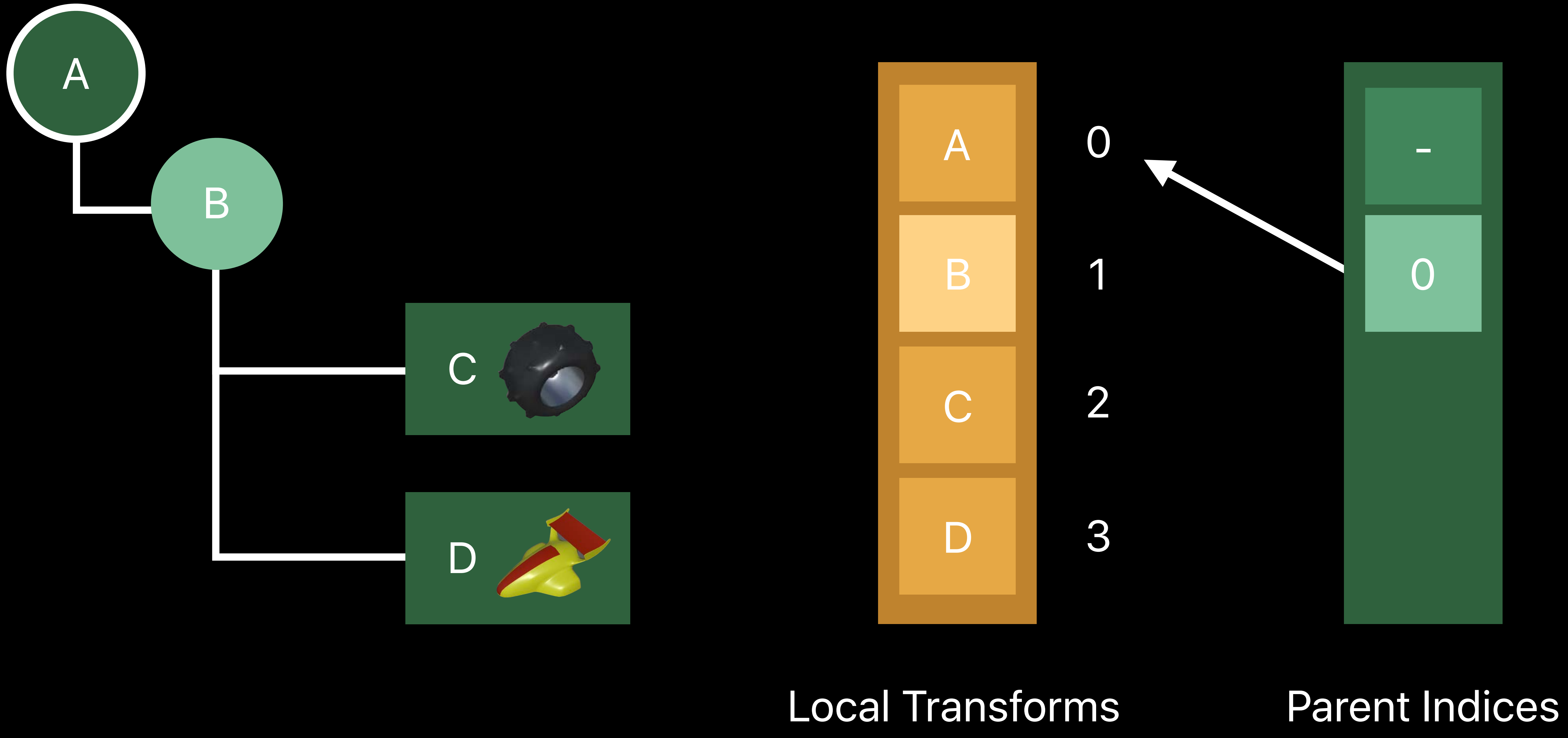
Geometry + Transform

Array of parent indices



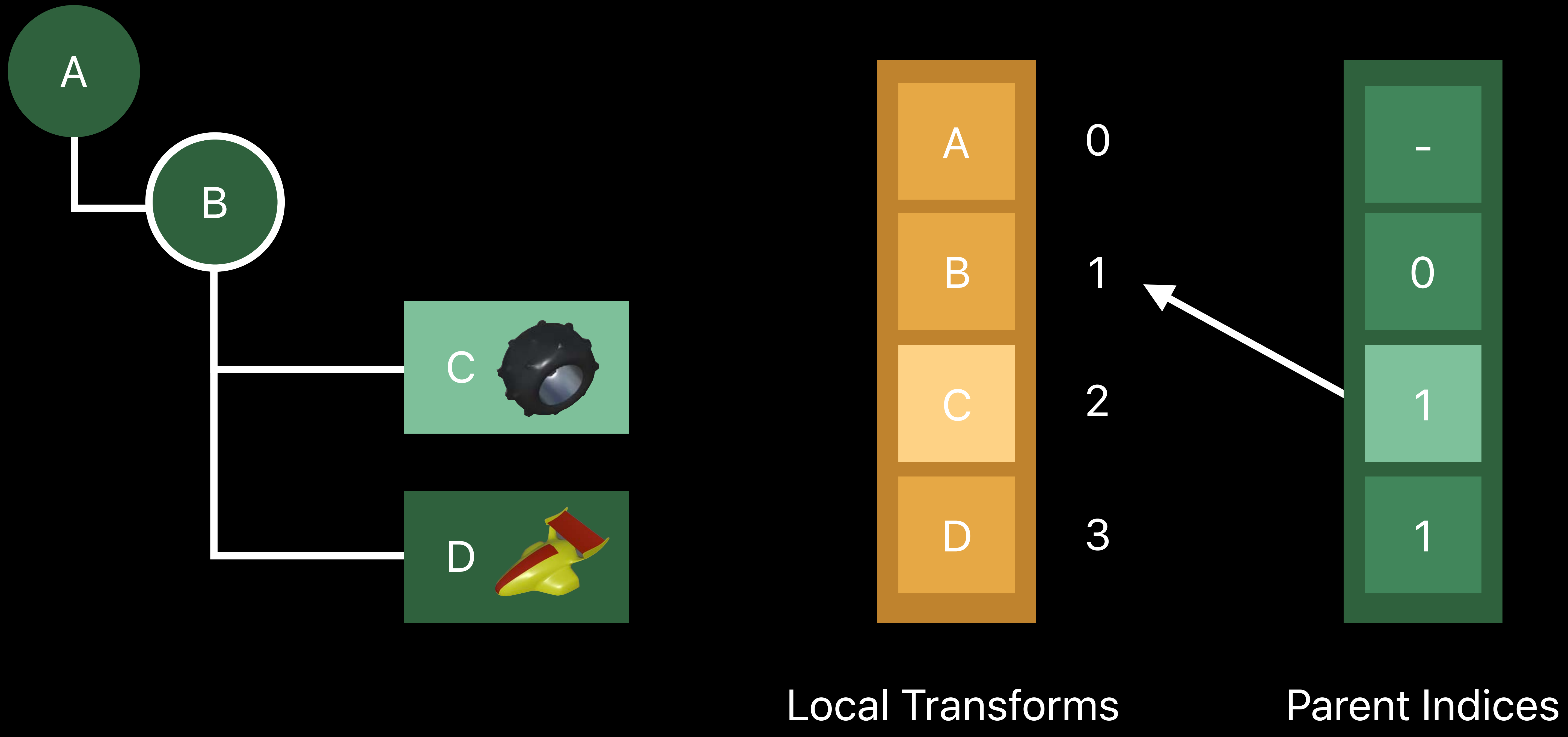
Geometry + Transform

Array of parent indices



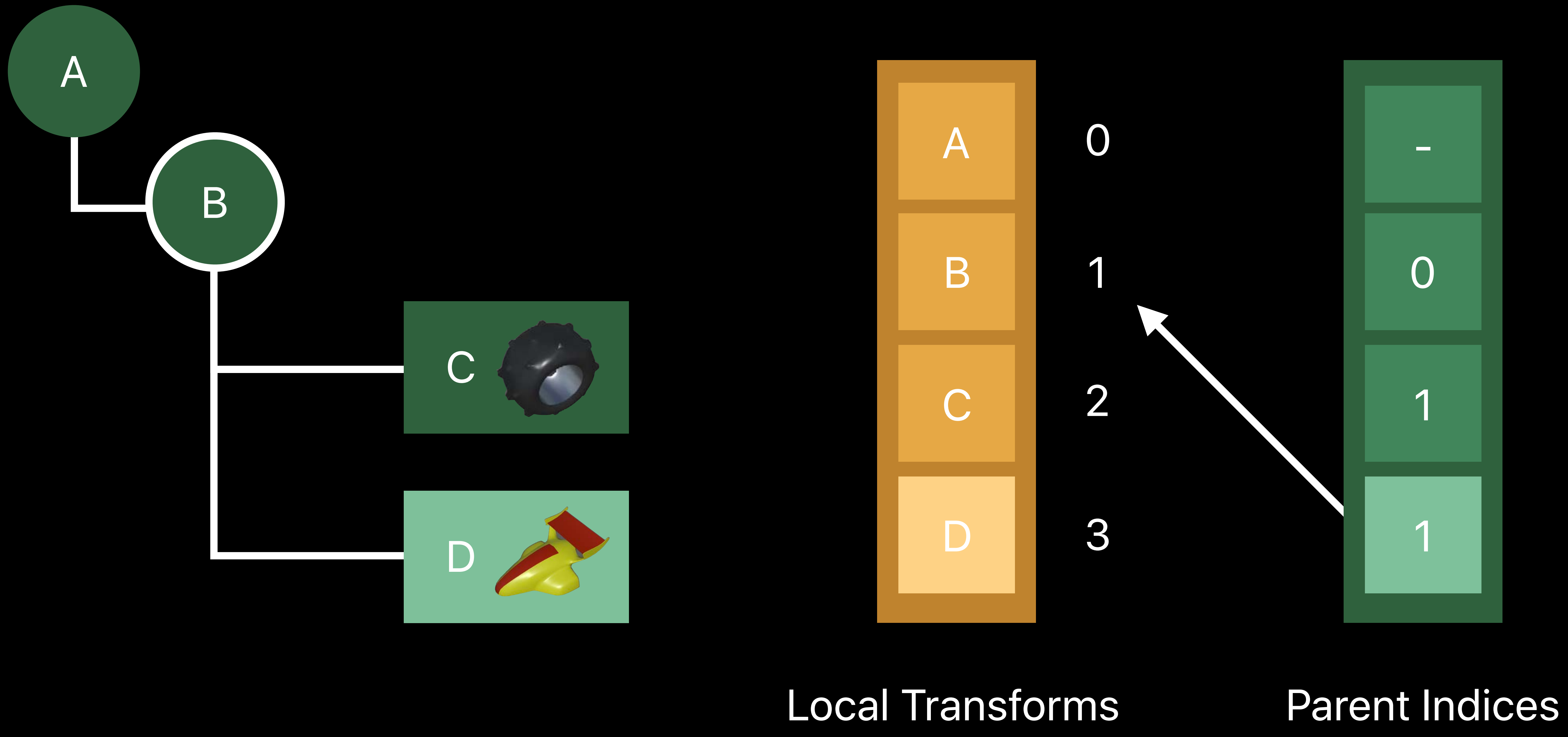
Geometry + Transform

Array of parent indices



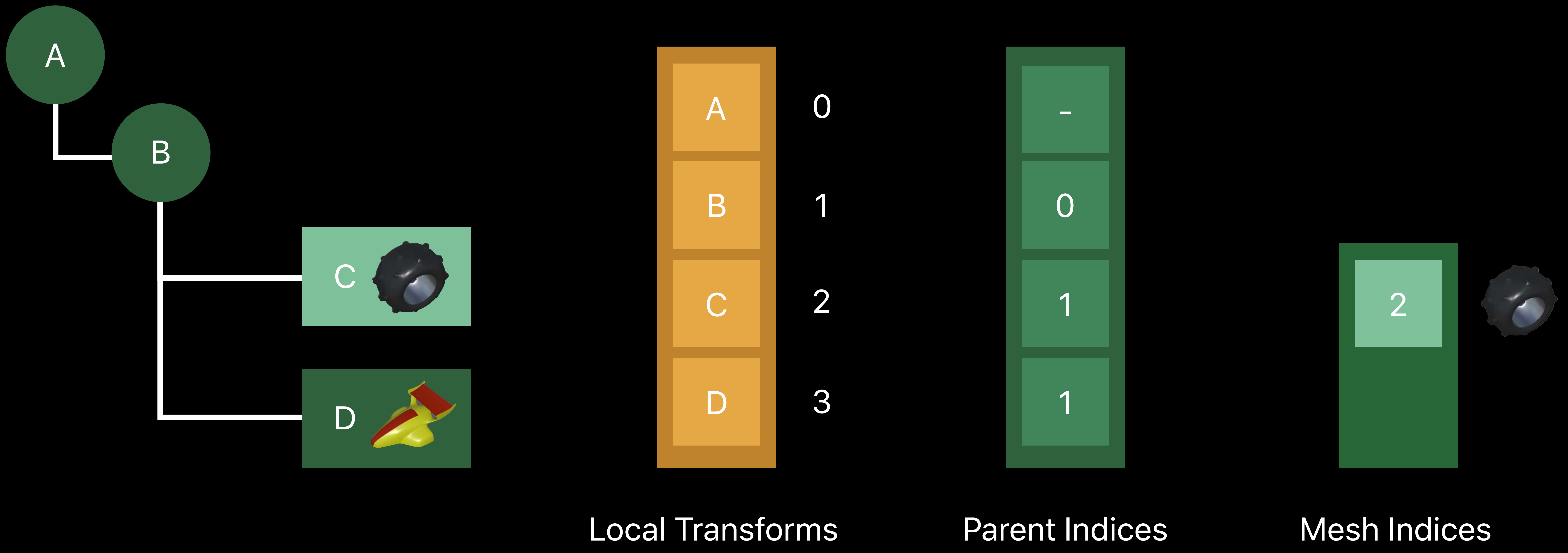
Geometry + Transform

Array of parent indices



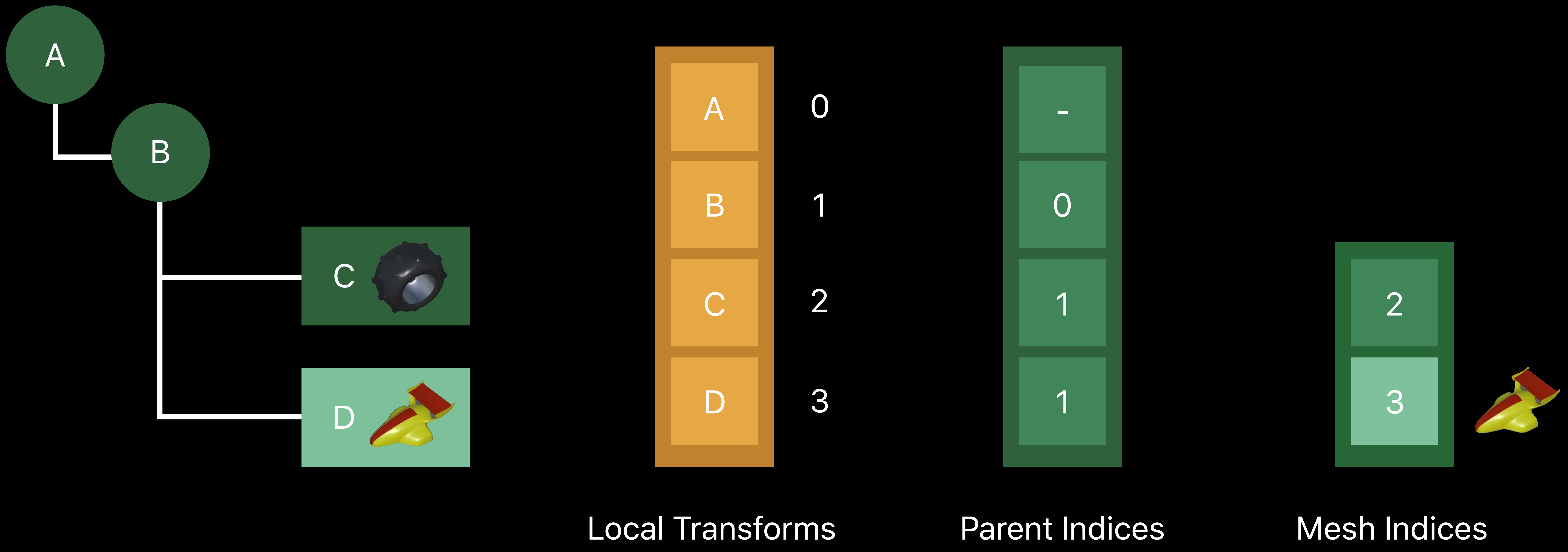
Geometry + Transform

Array of mesh indices



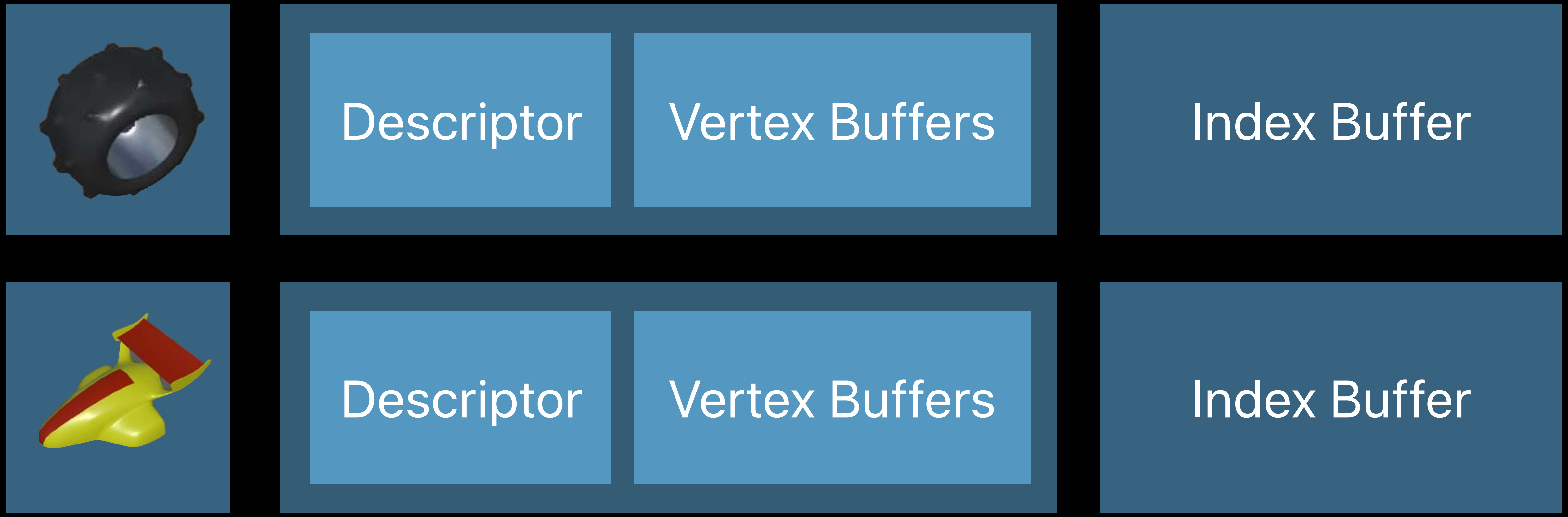
Geometry + Transform

Array of mesh indices



Geometry + Transform

Mesh data



Geometry + Transform

Vertex buffers

```swift
//for every mdlObject in MDLAsset:
if let mesh = mdlObject as? MDLMesh {
  vertexDescriptors.append(mesh.vertexDescriptor)
  for vertexBuffer in mesh.vertexBuffers {
    let vertexBufferData = Data(bytes: vertexBuffer.map().bytes,
                                count: vertexBuffer.length)
    ...
  }
  for submesh in mesh.submeshes! {
    if let indexBuffer = (submesh as? MDLSubmesh)?.indexBuffer {
      let indexBufferData = Data(bytes: indexBuffer.map().bytes,
                                 count: indexBuffer.length)
      ...
    }
  }
}
```

Geometry + Transform

Vertex buffers

```
//for every mdlObject in MDLAsset:
if let mesh = mdlObject as? MDLMesh {
  vertexDescriptors.append(mesh.vertexDescriptor)
  for vertexBuffer in mesh.vertexBuffers {
    let vertexBufferData = Data(bytes: vertexBuffer.map().bytes,
                                count: vertexBuffer.length)
    ...
  }
  for submesh in mesh.submeshes! {
    if let indexBuffer = (submesh as? MDLSubmesh)?.indexBuffer {
      let indexBufferData = Data(bytes: indexBuffer.map().bytes,
                                 count: indexBuffer.length)
      ...
    }
  }
}
```

Geometry + Transform

Vertex buffers

```
//for every mdlObject in MDLAsset:
if let mesh = mdlObject as? MDLMesh {
  vertexDescriptors.append(mesh.vertexDescriptor)
  for vertexBuffer in mesh.vertexBuffers {
    let vertexBufferData = Data(bytes: vertexBuffer.map().bytes,
                                count: vertexBuffer.length)
    ...
  }
  for submesh in mesh.submeshes! {
    if let indexBuffer = (submesh as? MDLSubmesh)?.indexBuffer {
      let indexBufferData = Data(bytes: indexBuffer.map().bytes,
                                 count: indexBuffer.length)
      ...
    }
  }
}
```

Geometry + Transform

Index buffer

```
//for every mdlObject in MDLAsset:
if let mesh = mdlObject as? MDLMesh {
  vertexDescriptors.append(mesh.vertexDescriptor)
  for vertexBuffer in mesh.vertexBuffers {
    let vertexBufferData = Data(bytes: vertexBuffer.map().bytes,
                                count: vertexBuffer.length)
    ...
  }
  for submesh in mesh.submeshes! {
    if let indexBuffer = (submesh as? MDLSubmesh)?.indexBuffer {
      let indexBufferData = Data(bytes: indexBuffer.map().bytes,
                                 count: indexBuffer.length)
      ...
    }
  }
}
```

Geometry + Transform

Local transform

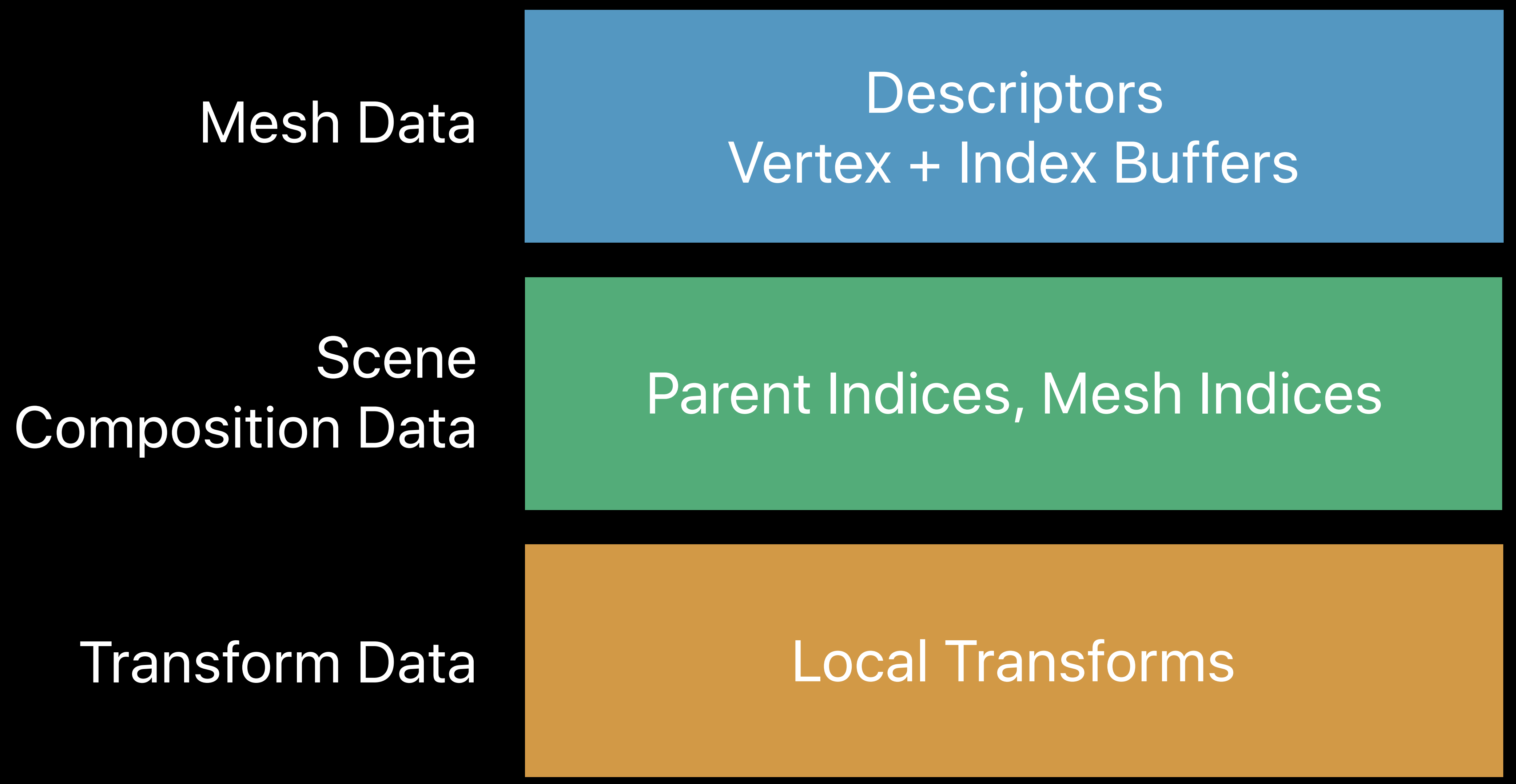
```swift
var localTransforms: [matrix_float4x4] = []
// for every mdlObject in MDLAsset:
if let transform = mdlObject.transform {
   localTransforms.append(transform.matrix)
}
```

Geometry + Transform

Local transform

```
var localTransforms: [matrix_float4x4] = []
// for every mdlObject in MDLAsset:
if let transform = mdlObject.transform {
   localTransforms.append(transform.matrix)
}
```

Geometry + Transform



Baking Operations

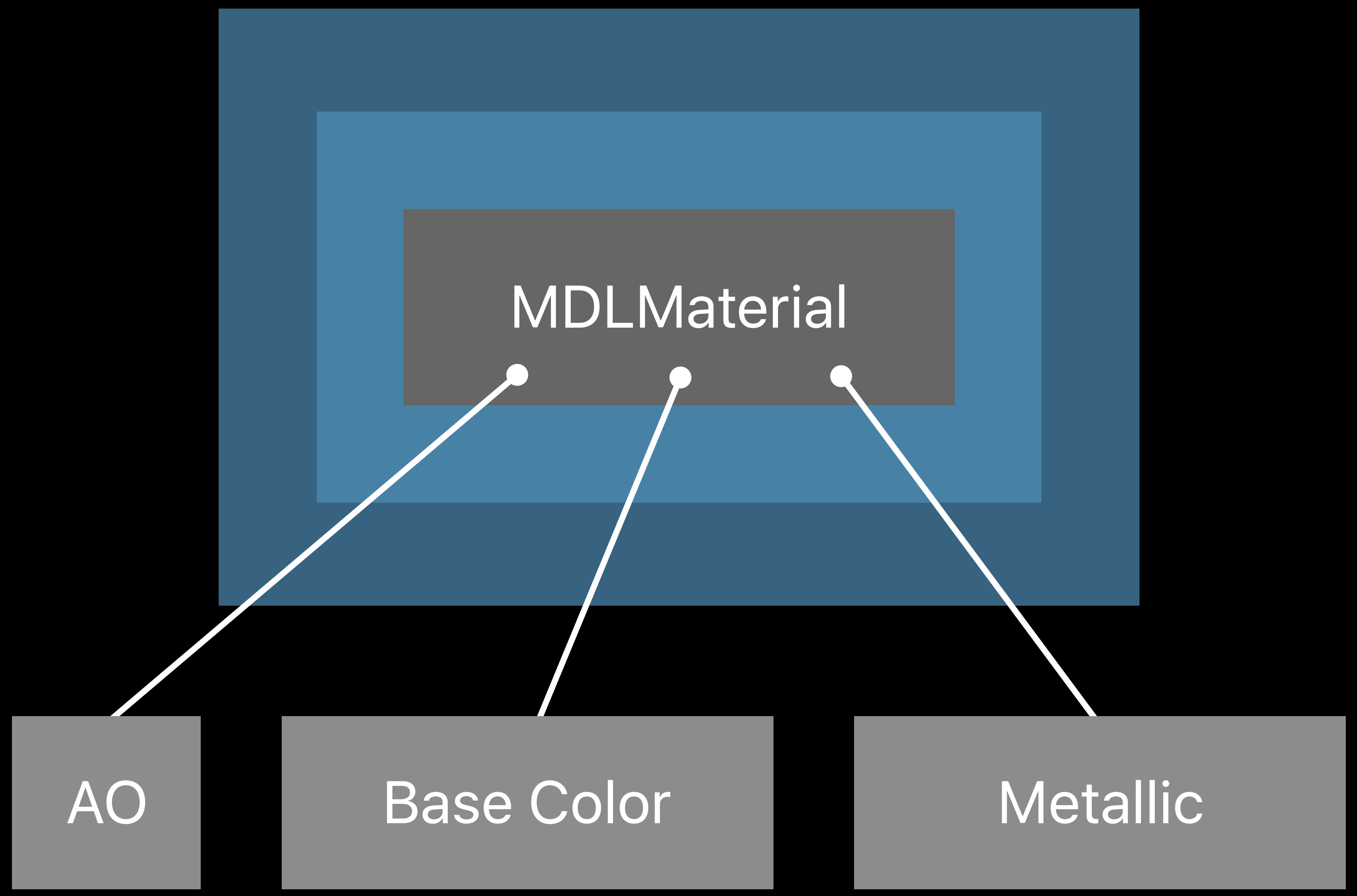
1. Geometry + Transforms
2. Texture Paths + Materials
3. Instancing Data
4. Transform Animation
5. Skinning + Character Animation

Texture Paths + Materials

Materials stored on MDLSubmesh

Fetch properties referenced by shader

Record texture paths and values



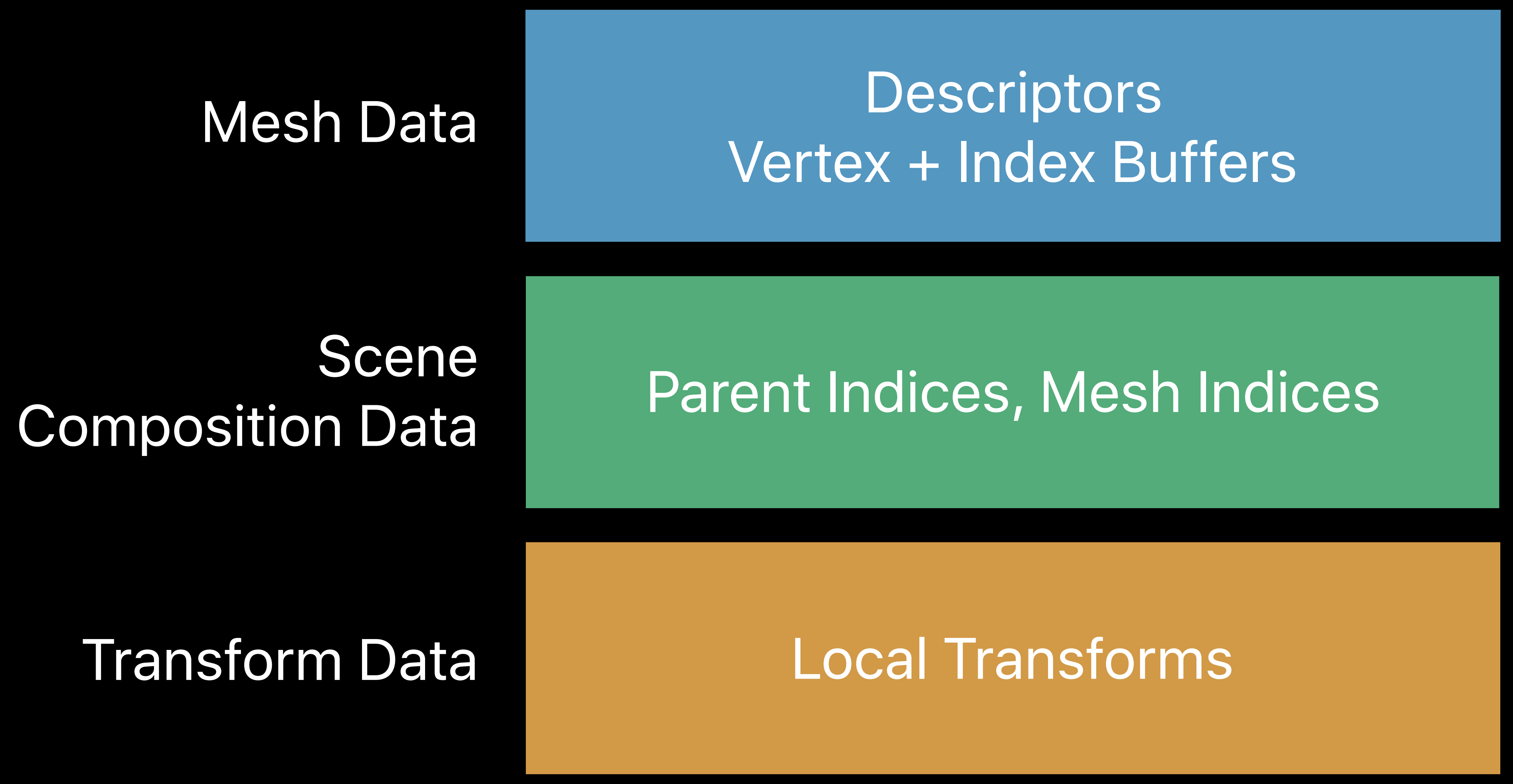
Texture Paths + Materials

```swift
// for every submesh:
if let material = submesh.material {

    for property in material.properties(with:<MDLMaterialSemantic>) {
        if property.type == .string || property.type == .URL {
            // texture
        }
        else if property.type == <MDLMaterialPropertyType> {
            // uniform value
        }
    }
}
```

Texture Paths + Materials

```
// for every submesh:
if let material = submesh.material {

   for property in material.properties(with:<MDLMaterialSemantic>) {
      if property.type == .string || property.type == .URL {
         // texture
      }
      else if property.type == <MDLMaterialPropertyType> {
         // uniform value
      }
   }
}
```

Texture Paths + Materials



Texture Paths + Materials

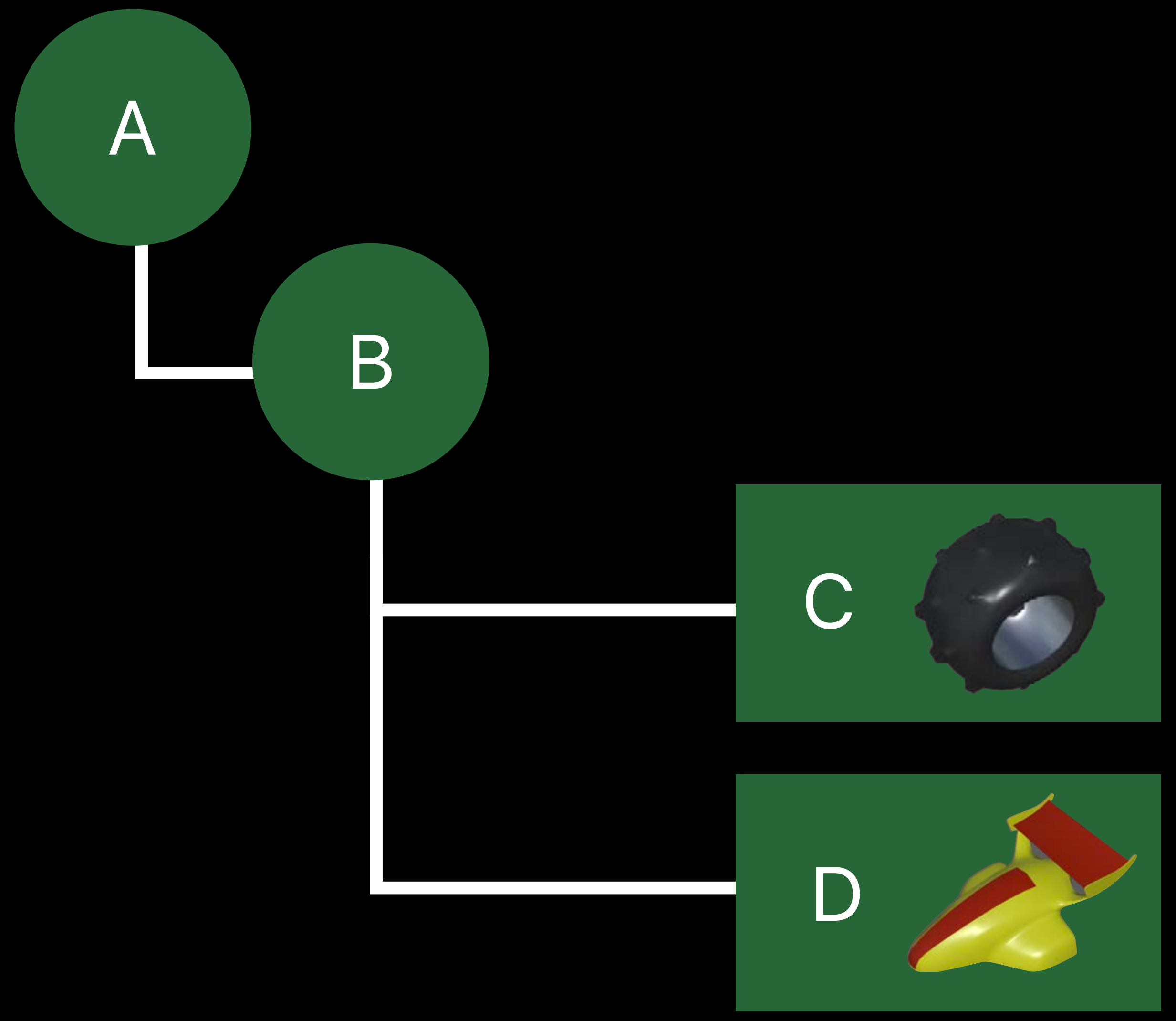
Mesh Data	Descriptors Vertex + Index Buffers
Scene Composition Data	Parent Indices, Mesh Indices
Transform Data	Local Transforms
Material Data	Material Uniforms Texture Paths

Baking Operations

1. Geometry + Transforms
2. Texture Paths + Materials
3. Instancing Data
4. Transform Animation
5. Skinning + Character Animation

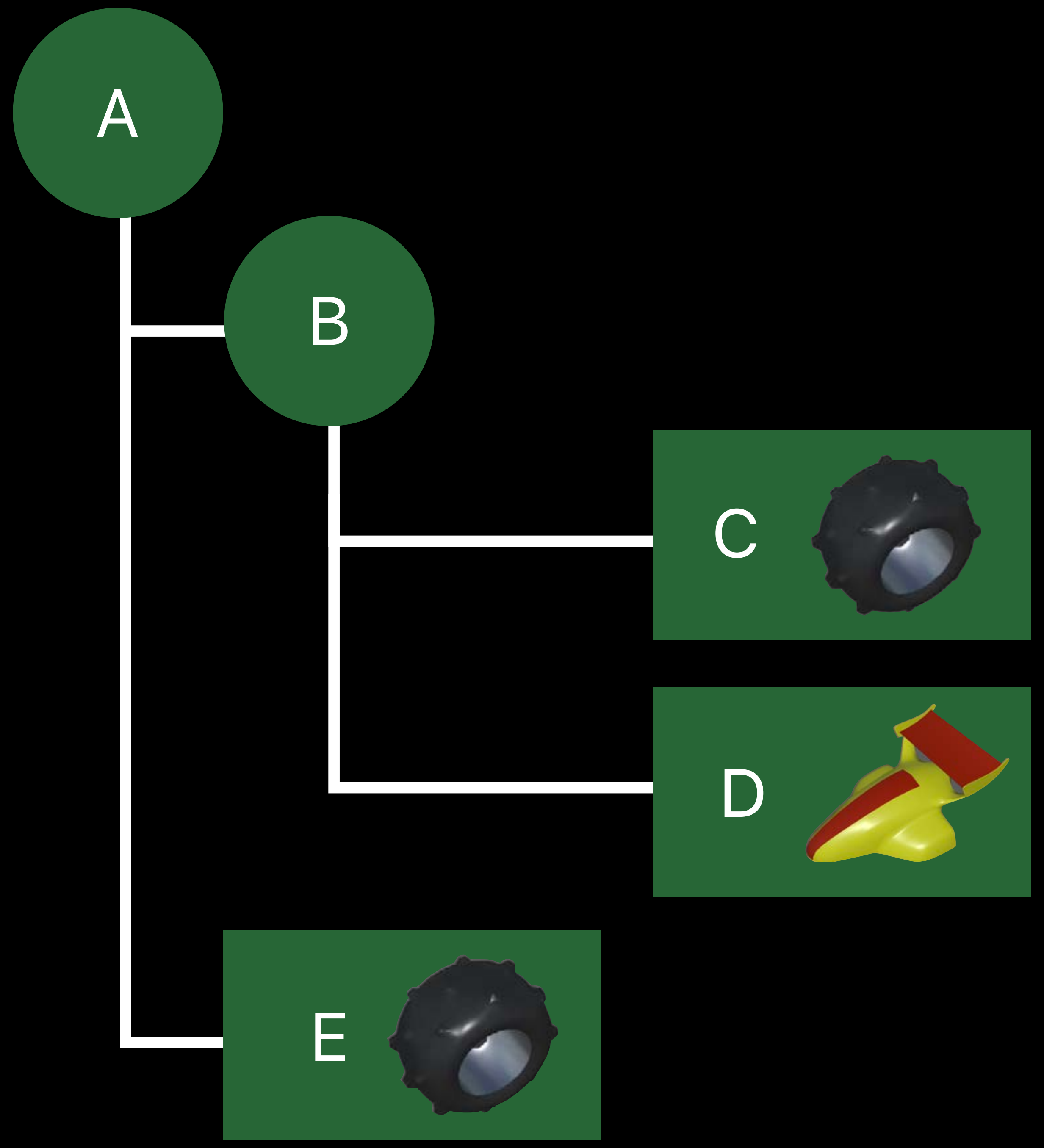
Instancing

A mesh can be used multiple times



Instancing

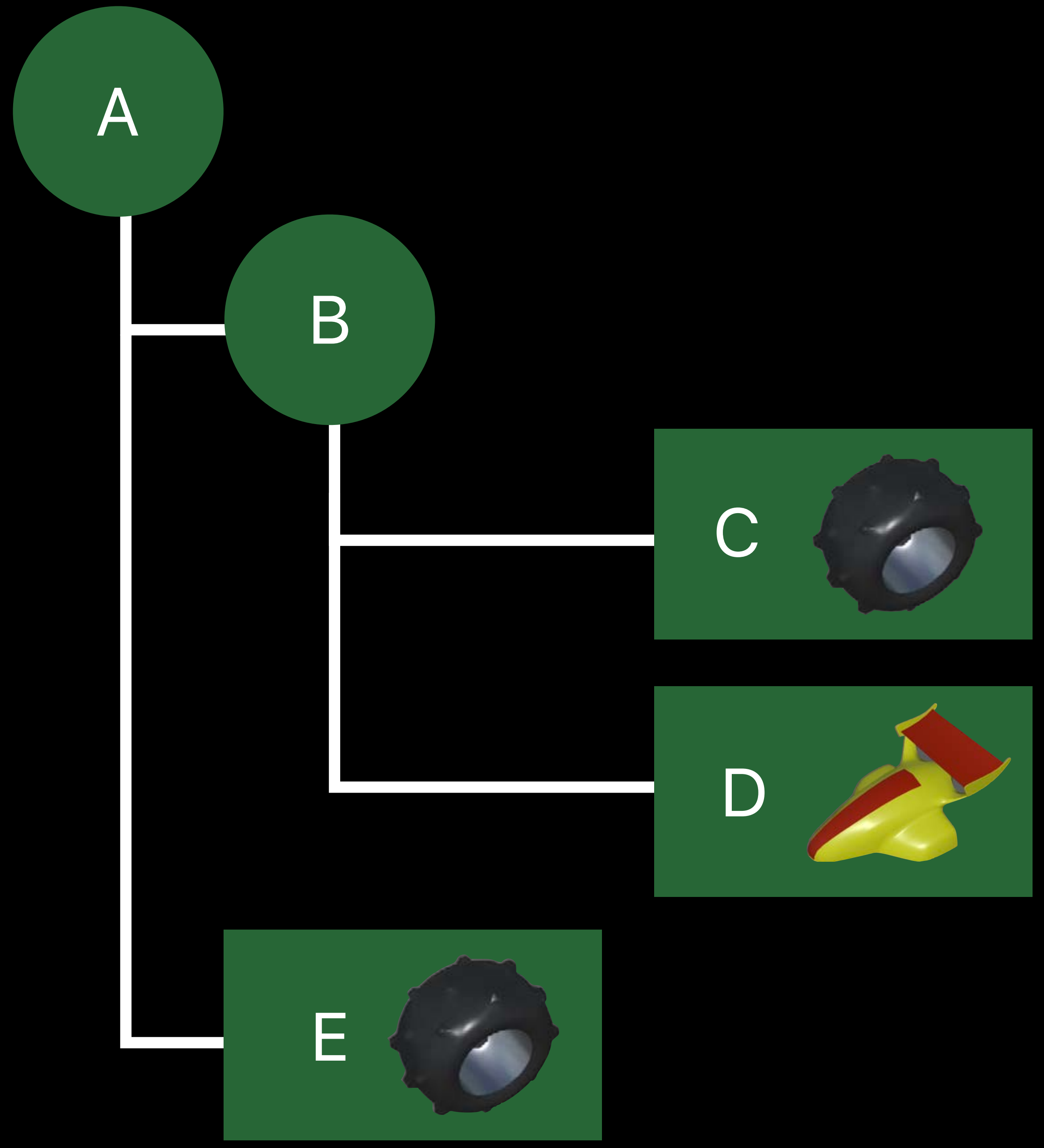
A mesh can be used multiple times



Instancing

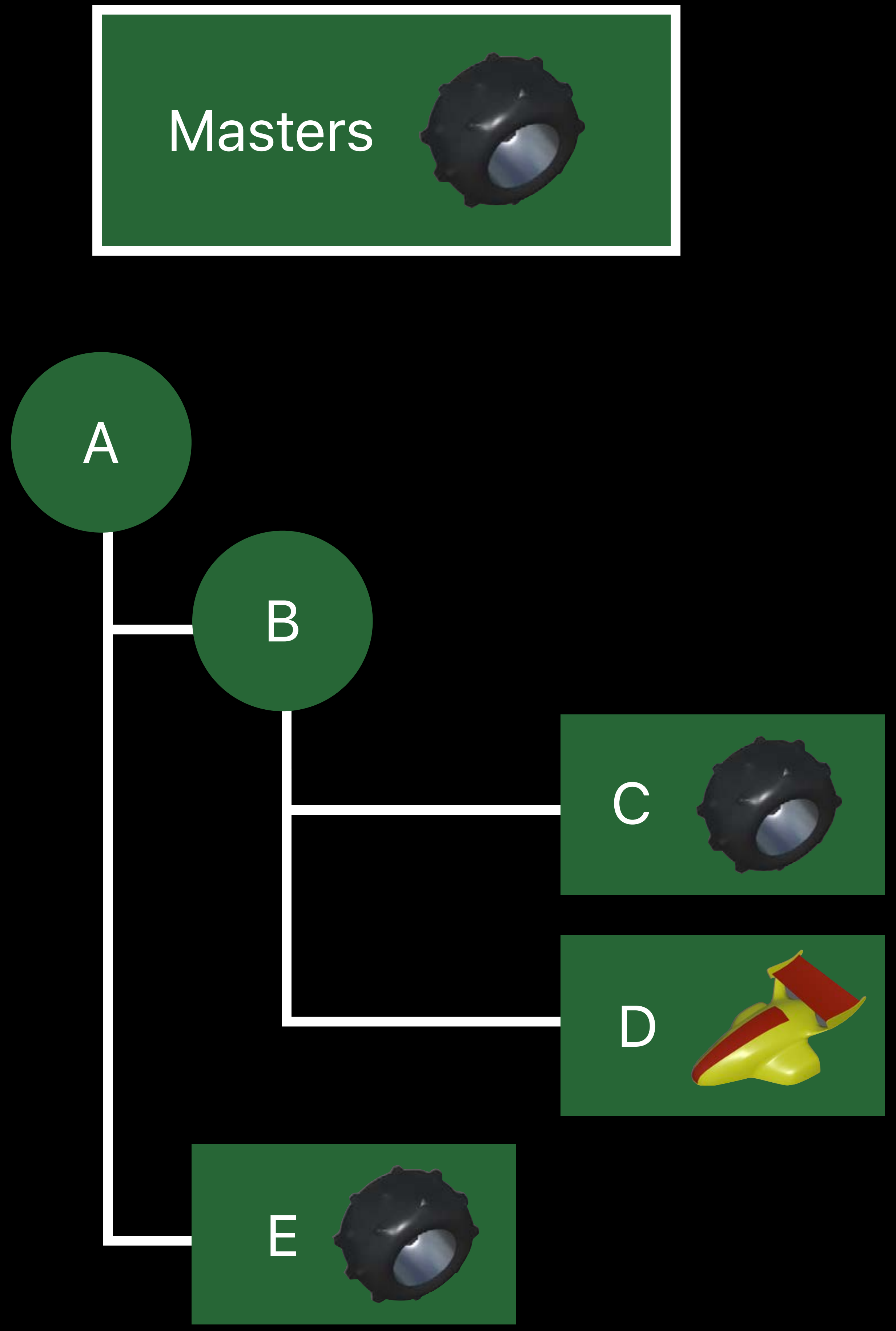
A mesh can be used multiple times

Why store it multiple times?



Instancing
Masters

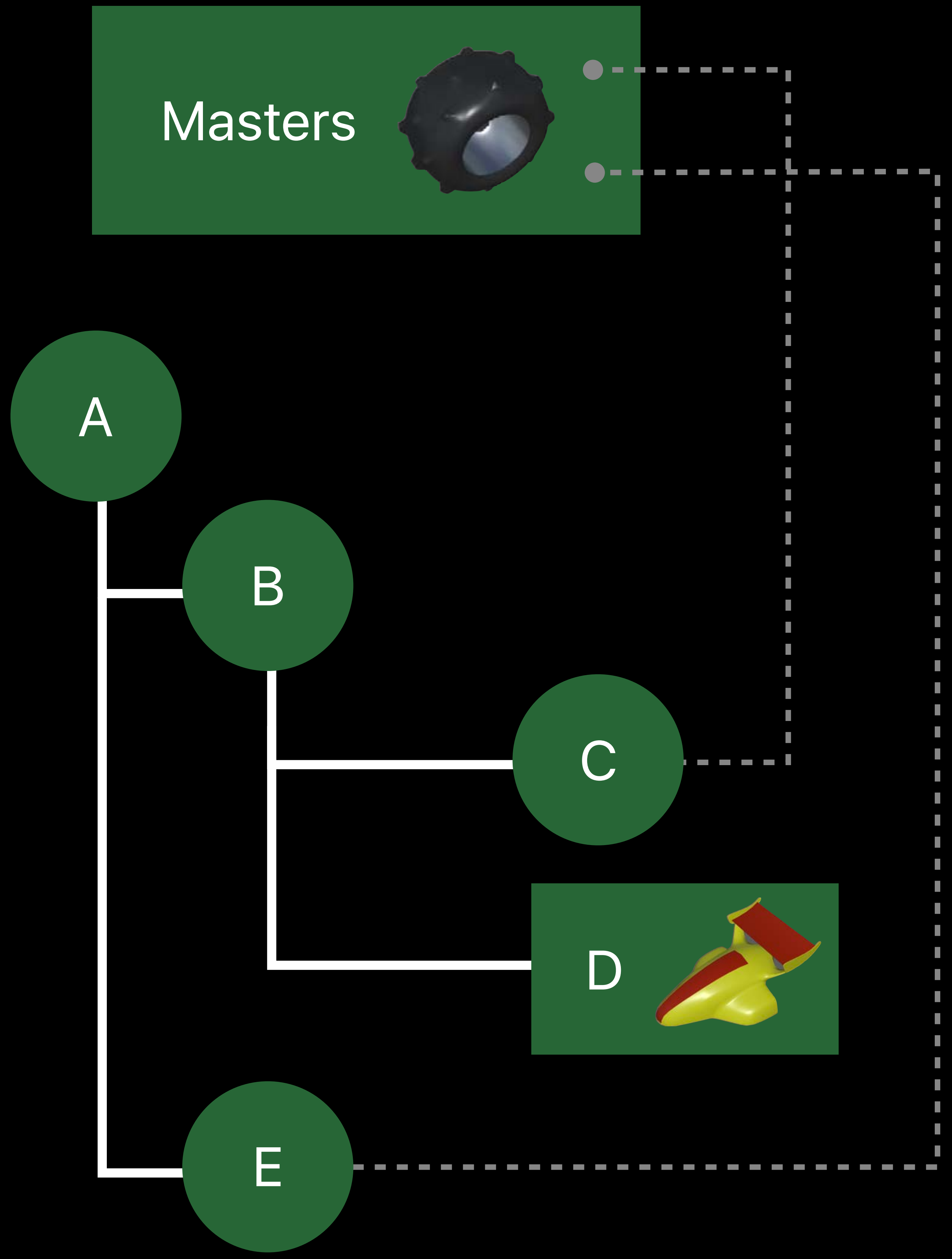
MDLAsset has a masters array



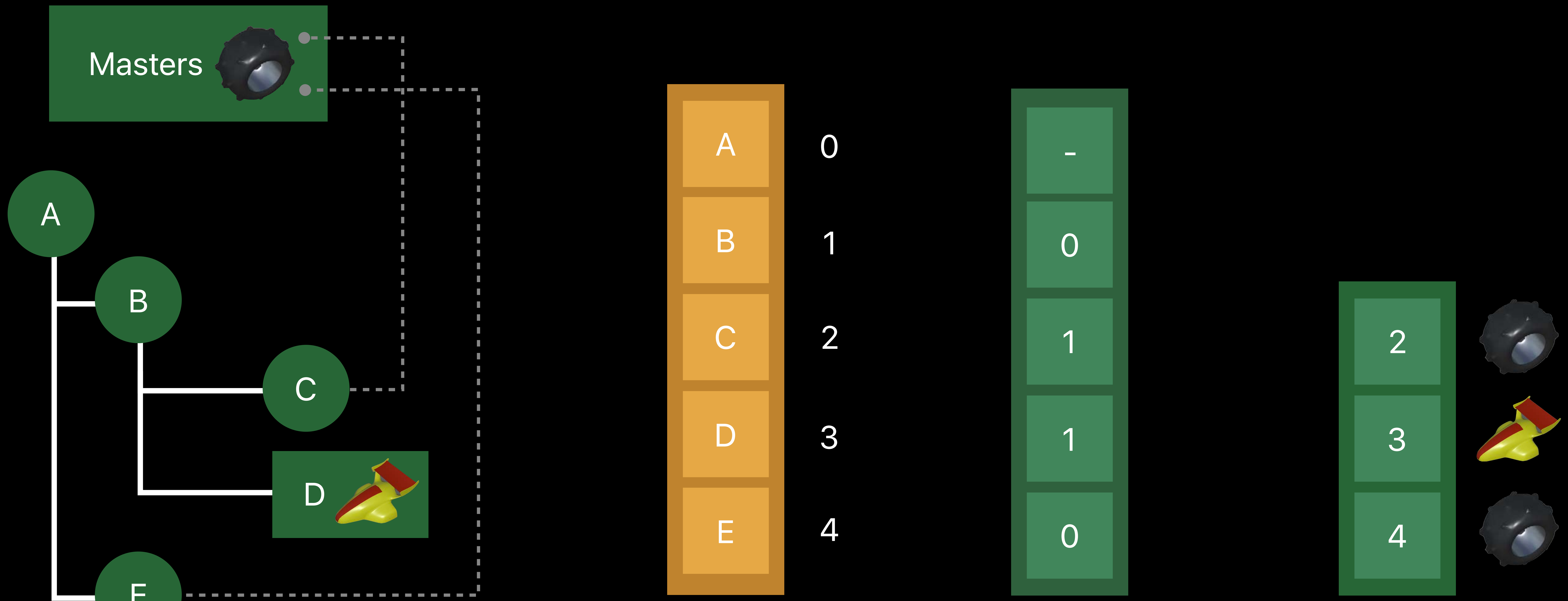
Instancing

Model I/O support

MDLObject instance pointers refer to masters



Instancing

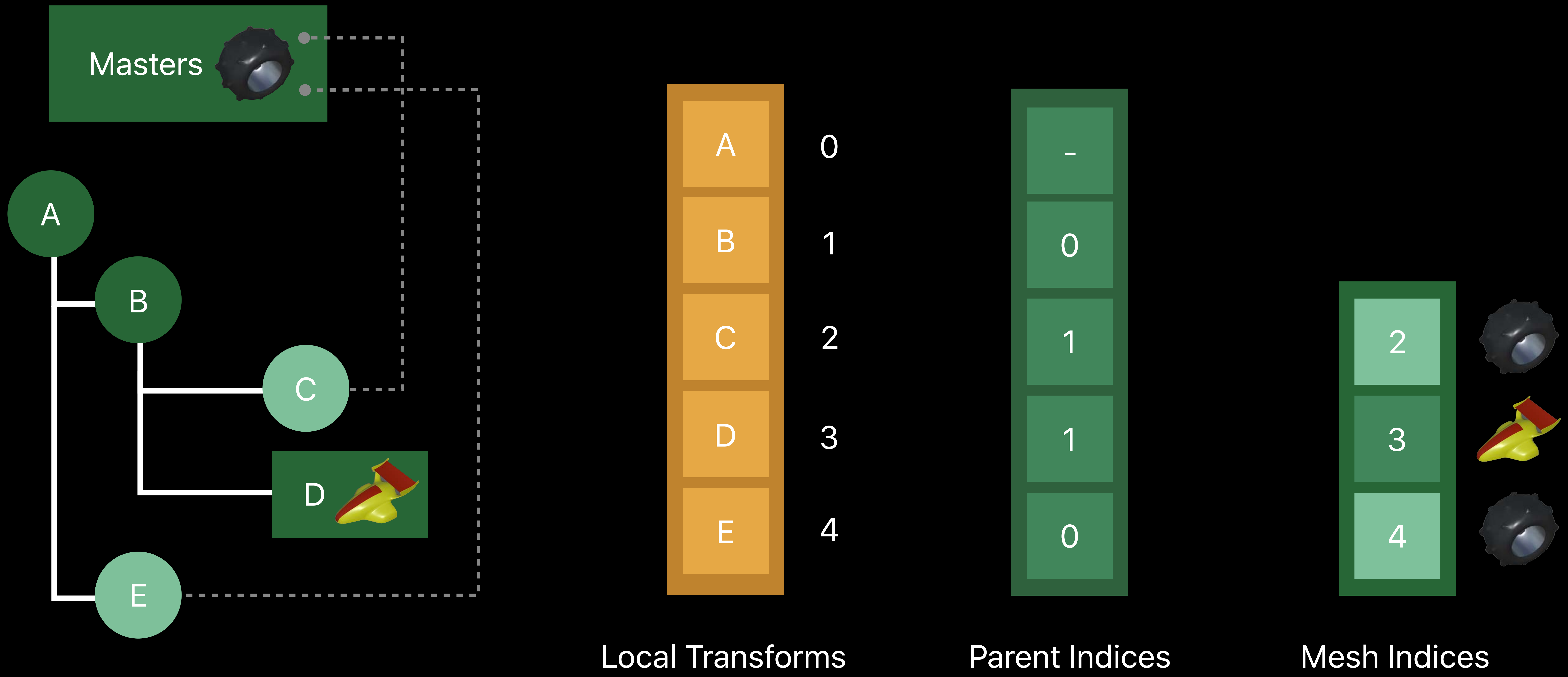


Local Transforms

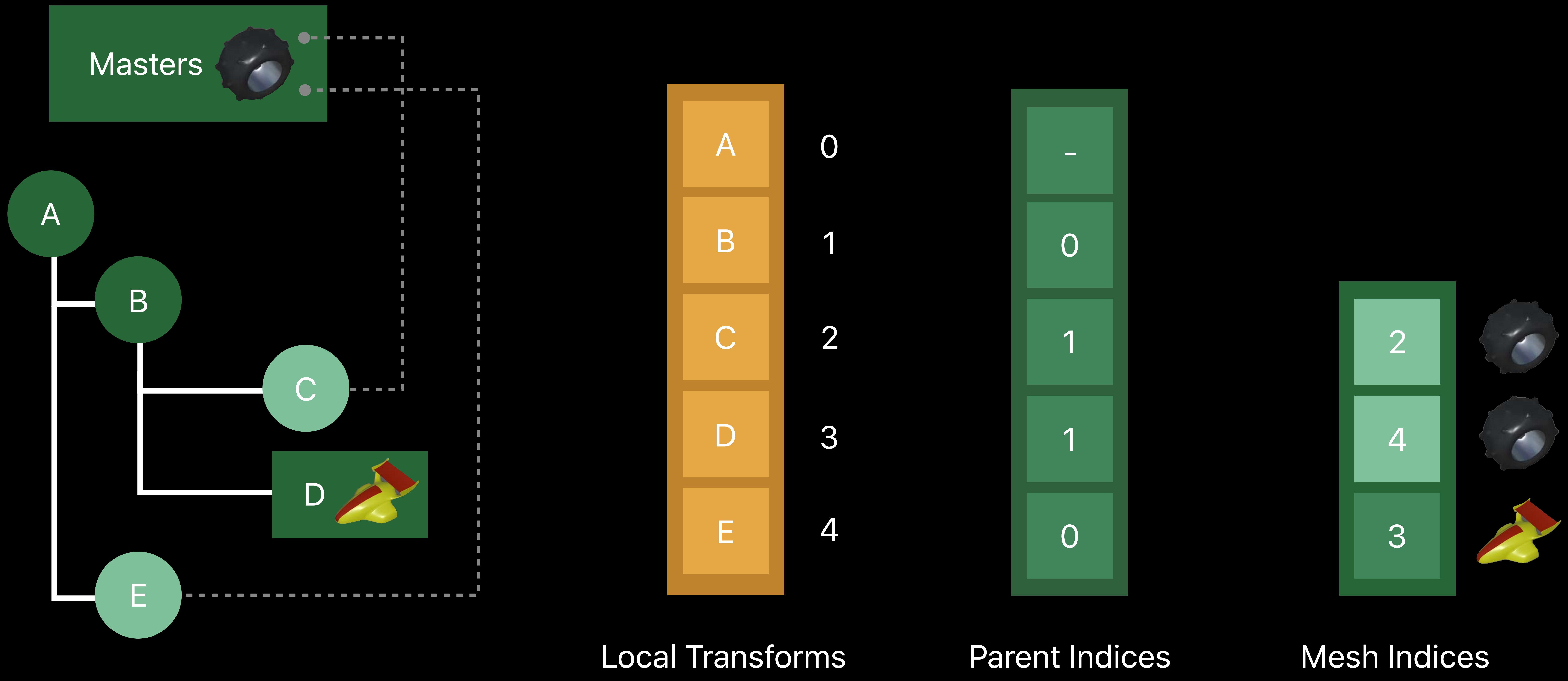
Parent Indices

Mesh Indices

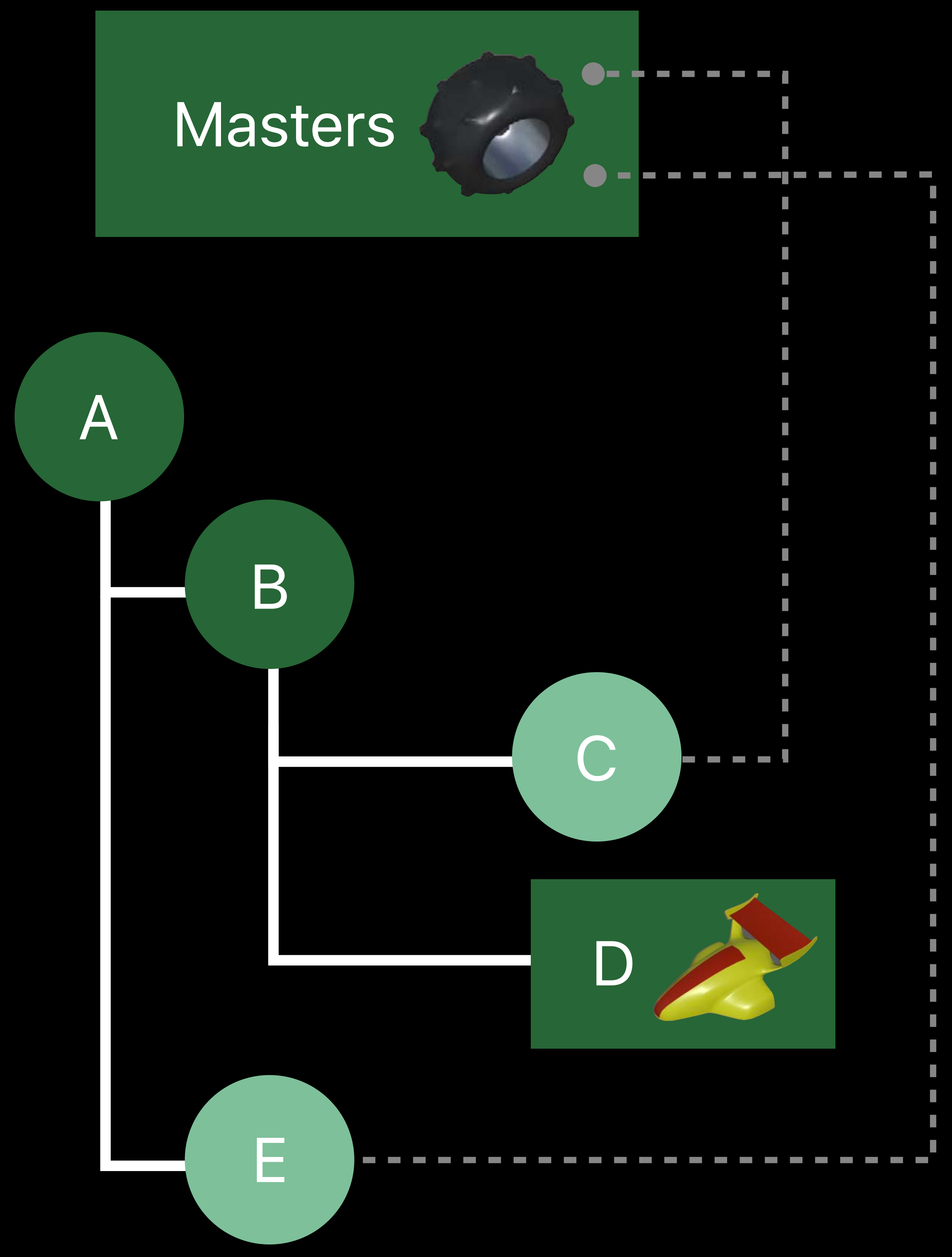
Instancing

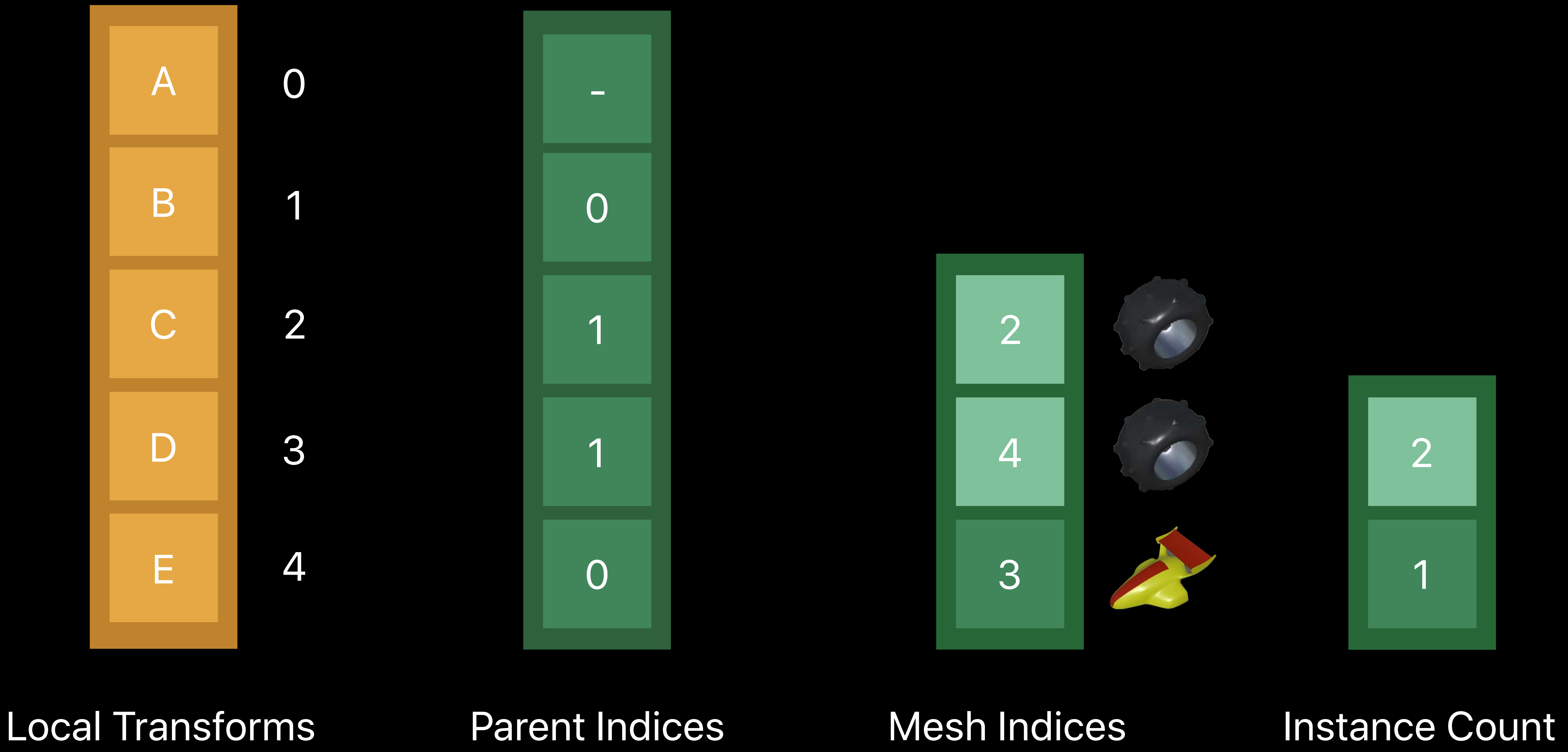

Masters
A
B
C
D
E
A
B
C
D
E
0
1
2
3
4
-
0
1
1
0
2
3
4
Local Transforms
Parent Indices
Mesh Indices

Instancing


Masters
A
B
C
D
E
A
B
C
D
E
0
1
2
3
4
-
0
1
1
0
2
4
3
Local Transforms
Parent Indices
Mesh Indices

Instancing


Masters
A
B
C
D
E


A 0
B 1
C 2
D 3
E 4
Local Transforms
-
0
1
1
0
Parent Indices
2
4
3
Mesh Indices
2
1
Instance Count

Instancing

Mesh Data	Descriptors Vertex + Index Buffers
Scene Composition Data	Parent Indices, Mesh Indices
Transform Data	Local Transforms
Material Data	Material Uniforms Texture Paths

Instancing

Mesh Data	Descriptors Vertex + Index Buffers
Scene Composition Data	Parent Indices, Mesh Indices Instance Count
Transform Data	Local Transforms
Material Data	Material Uniforms Texture Paths

Demo

Geometry, Materials, and Instancing

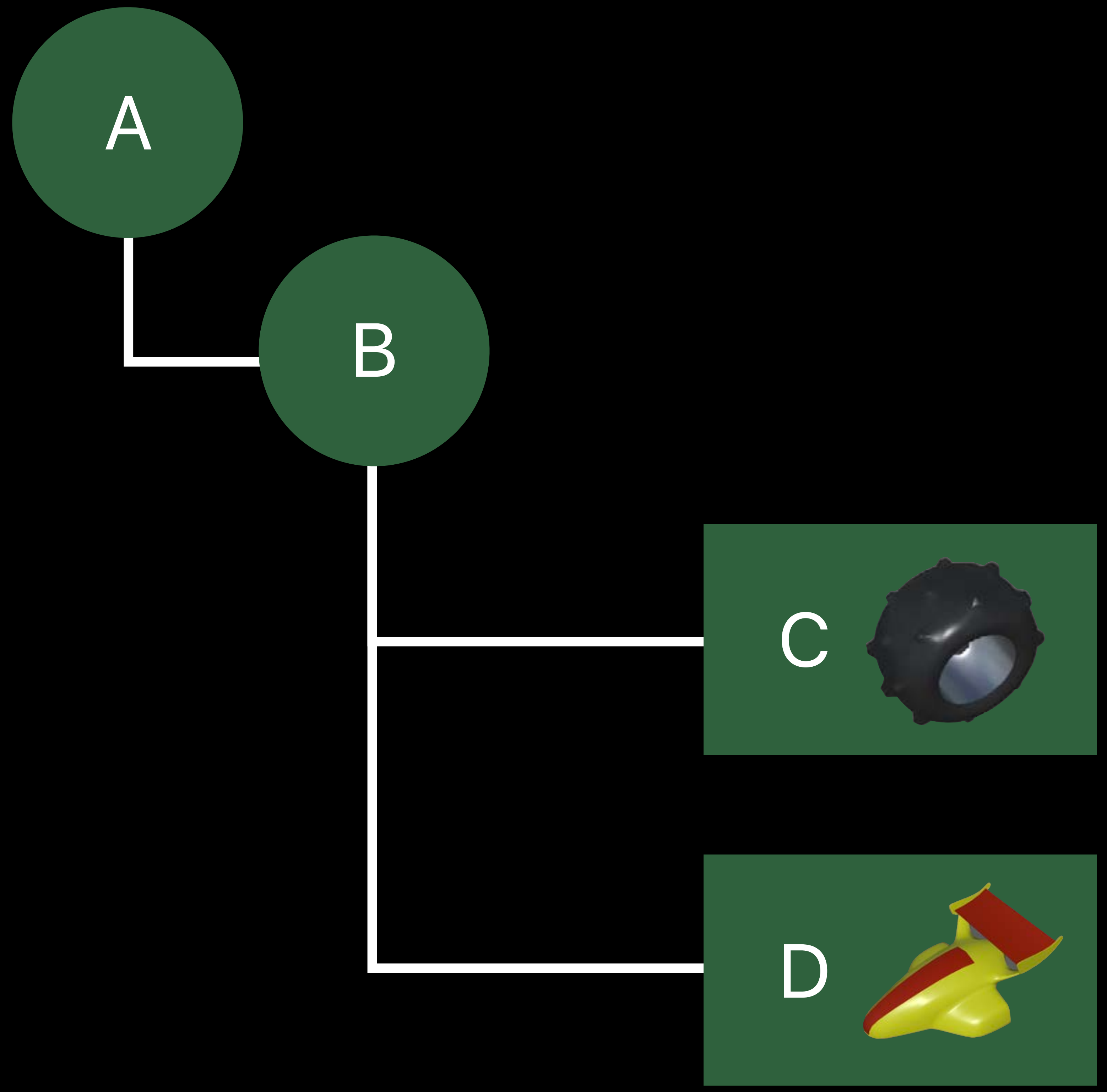
Nicholas Blasingame, Game Technologies Engineer

Baking Operations

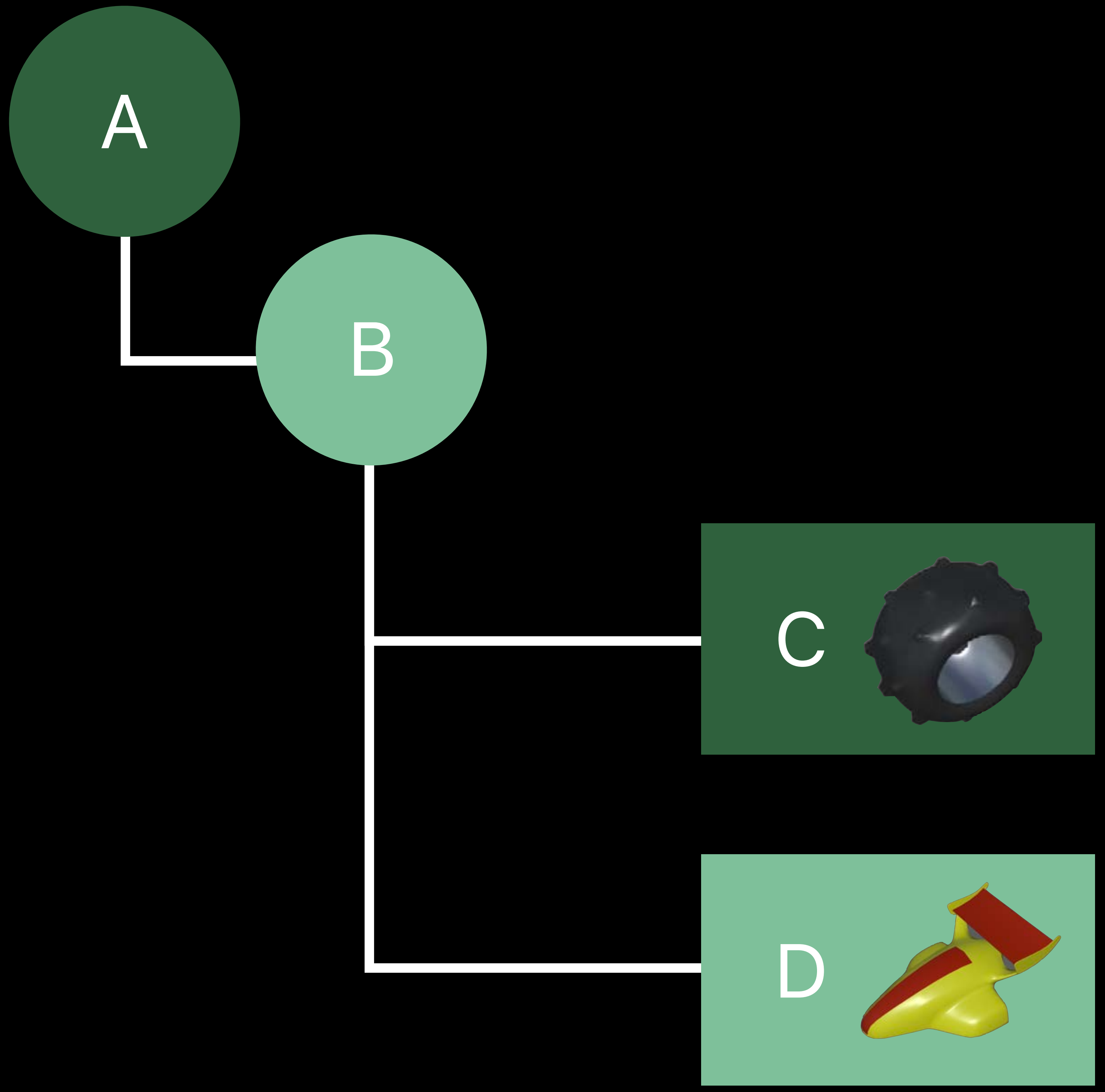
1. Geometry + Transforms
2. Texture Paths + Materials
3. Instancing Data
4. Transform Animation
5. Skinning + Character Animation

Transform Animation

Transforms that vary over time

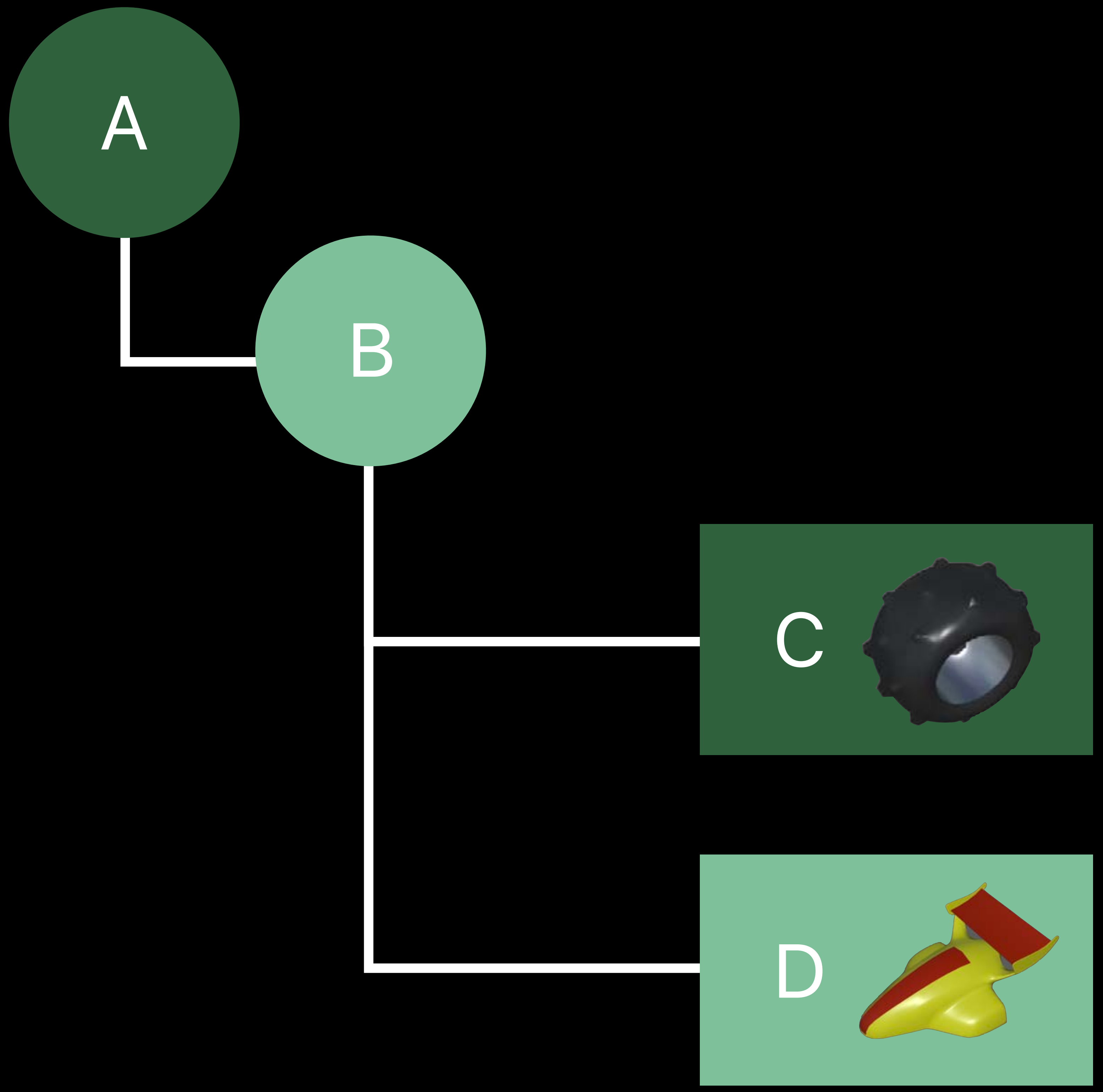


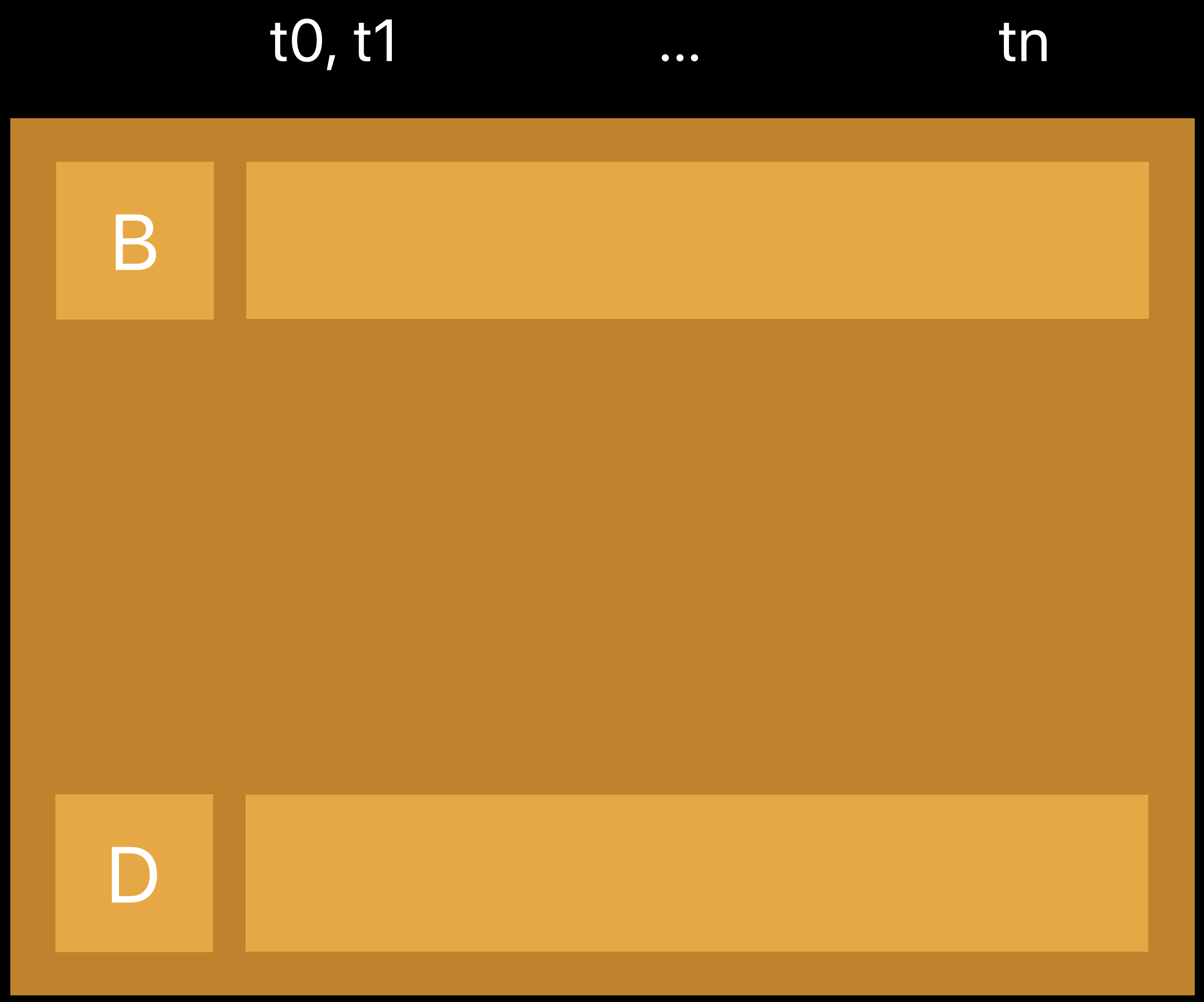
Transform Animation



Transform Animation

Sample the animations





Animated Local Transforms

Transform Animation

```
var localTransforms: [matrix_float4x4] = []
// for every mdlObject in MDLAsset:
if let transform = mdlObject.transform {
   localTransforms.append(transform.matrix)





}
```

Transform Animation

```
var localTransforms: [matrix_float4x4] = []
// for every mdlObject in MDLAsset:
if let transform = mdlObject.transform {
   localTransforms.append(transform.matrix)

   if (transform.keyTimes.count > 1) {
     let sampledXM = sampleTimes.map{ transform.localTransform!(atTime: $0) }
     animatedLocalTransforms.append(sampledXM)
     ...
   }
}
```

Transform Animation

```swift
var localTransforms: [matrix_float4x4] = []
// for every mdlObject in MDLAsset:
if let transform = mdlObject.transform {
   localTransforms.append(transform.matrix)

   if (transform.keyTimes.count > 1) {
     let sampledXM = sampleTimes.map{ transform.localTransform!(atTime: $0) }
     animatedLocalTransforms.append(sampledXM)
     ...
   }
}
```

Transform Animation

Mesh Data	Descriptors Vertex + Index Buffers
Scene Composition Data	Parent Indices, Mesh Indices Instance Count
Transform Data	Local Transforms
Material Data	Material Uniforms Texture Paths

Transform Animation

Mesh Data	Descriptors Vertex + Index Buffers
Scene Composition Data	Parent Indices, Mesh Indices Instance Count
Transform Data	Local Transforms Animated Local Transforms
Material Data	Material Uniforms Texture Paths

Baking Operations

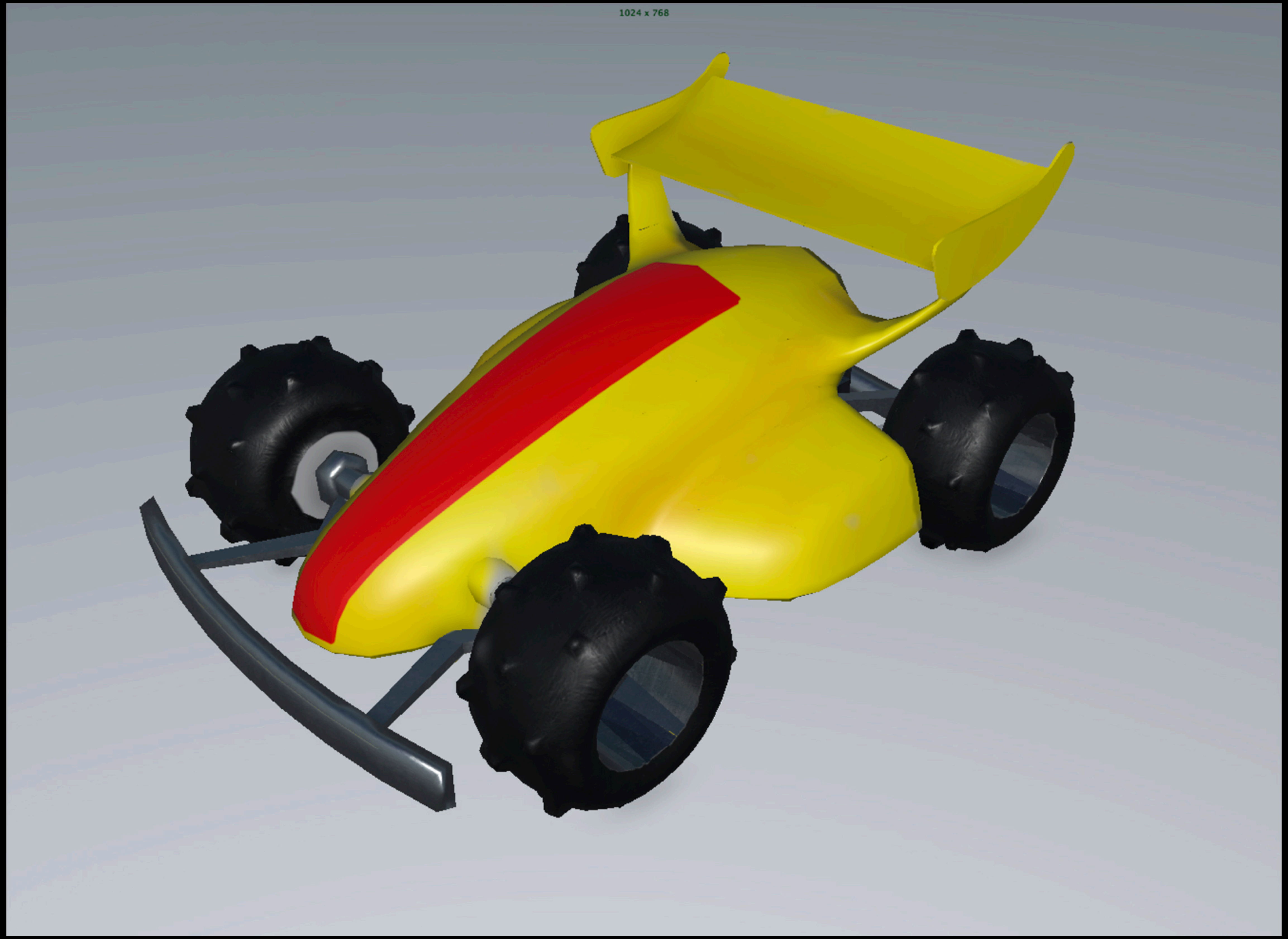
1. Geometry + Transforms
2. Texture Paths + Materials
3. Instancing Data
4. Transform Animation
5. Skinning + Character Animation

Skinned Character Animation

Skinned Character Animation

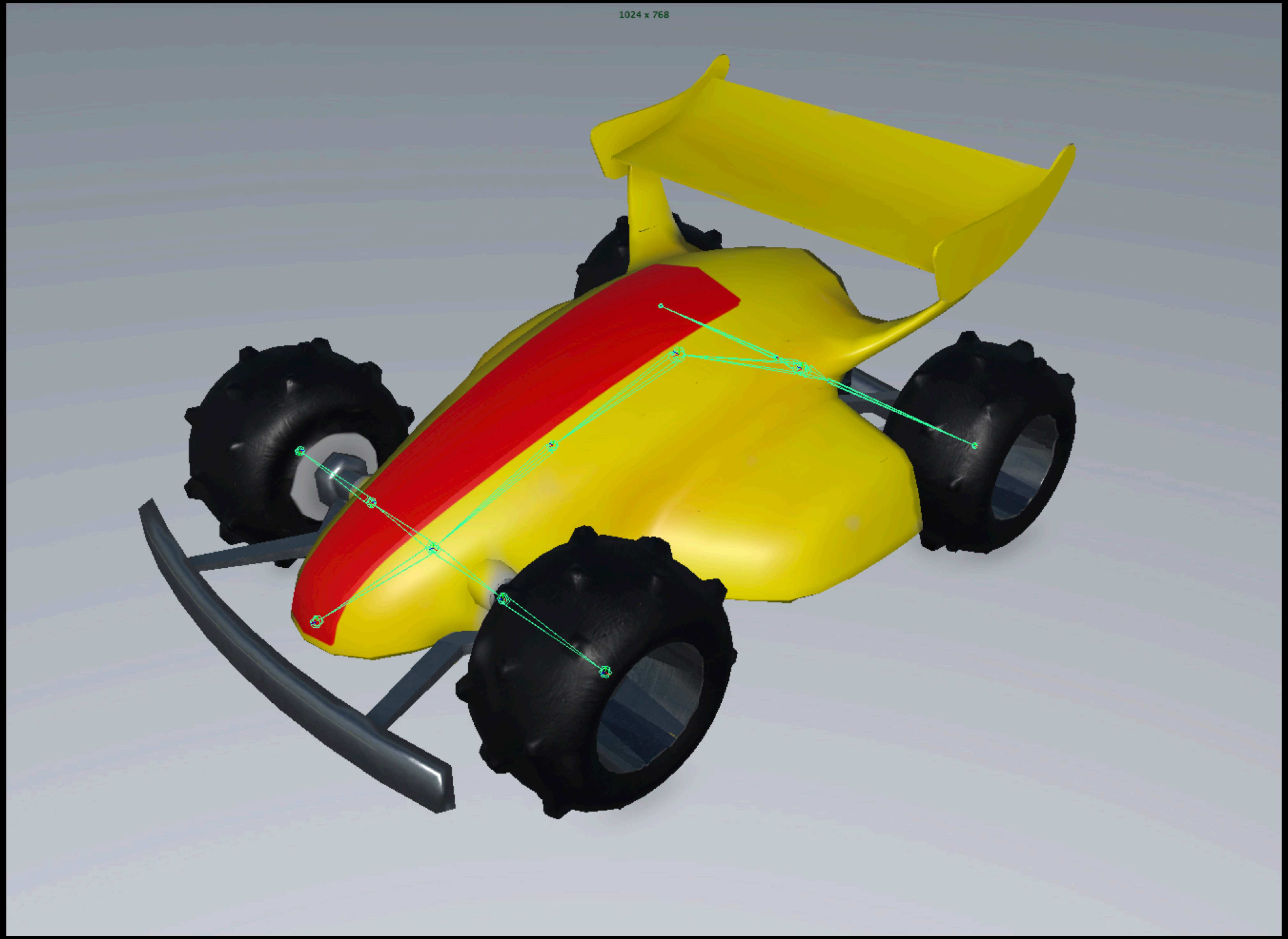
Skinned Character Animation

Mesh has geometry



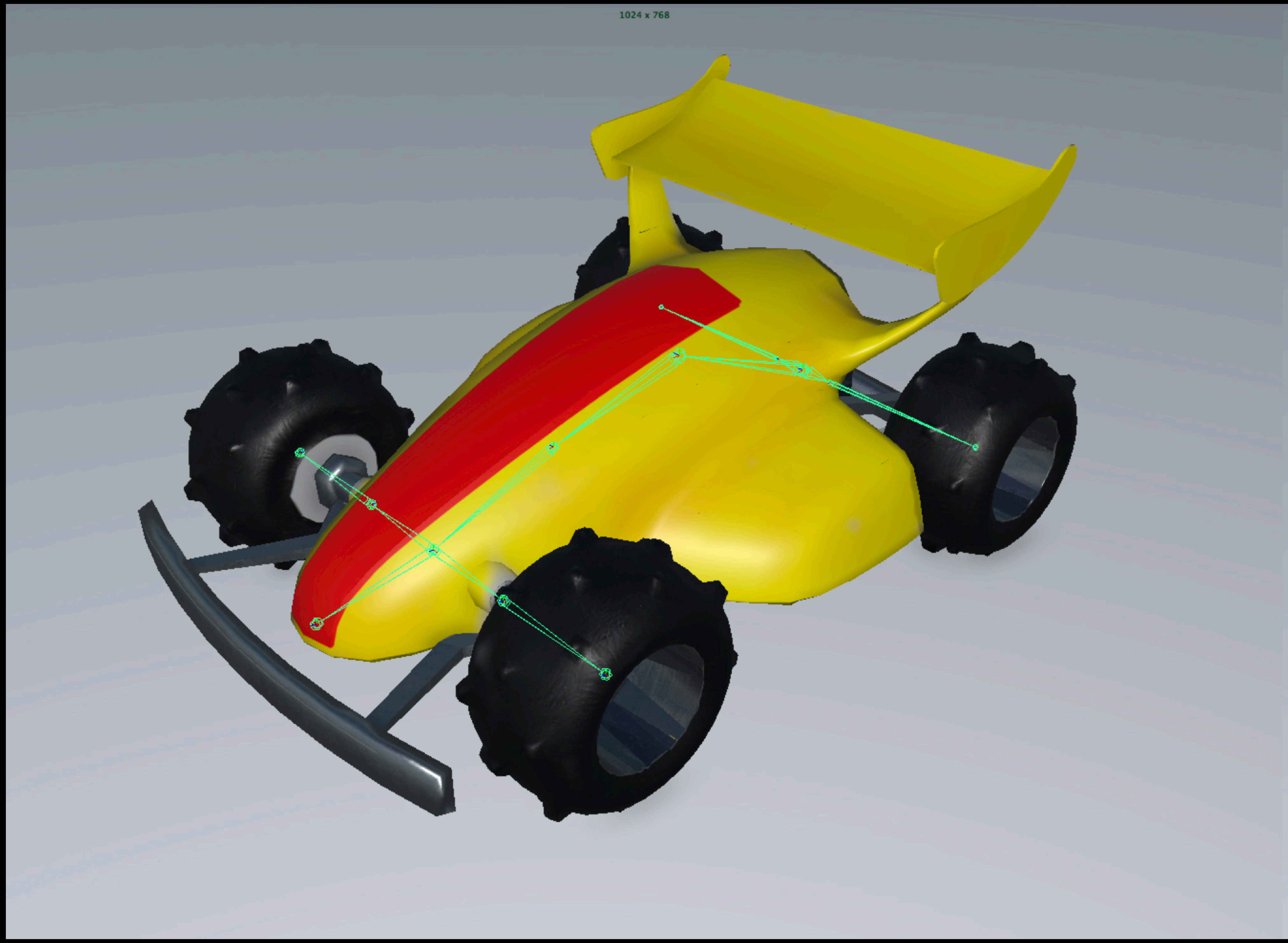
Skinned Character Animation

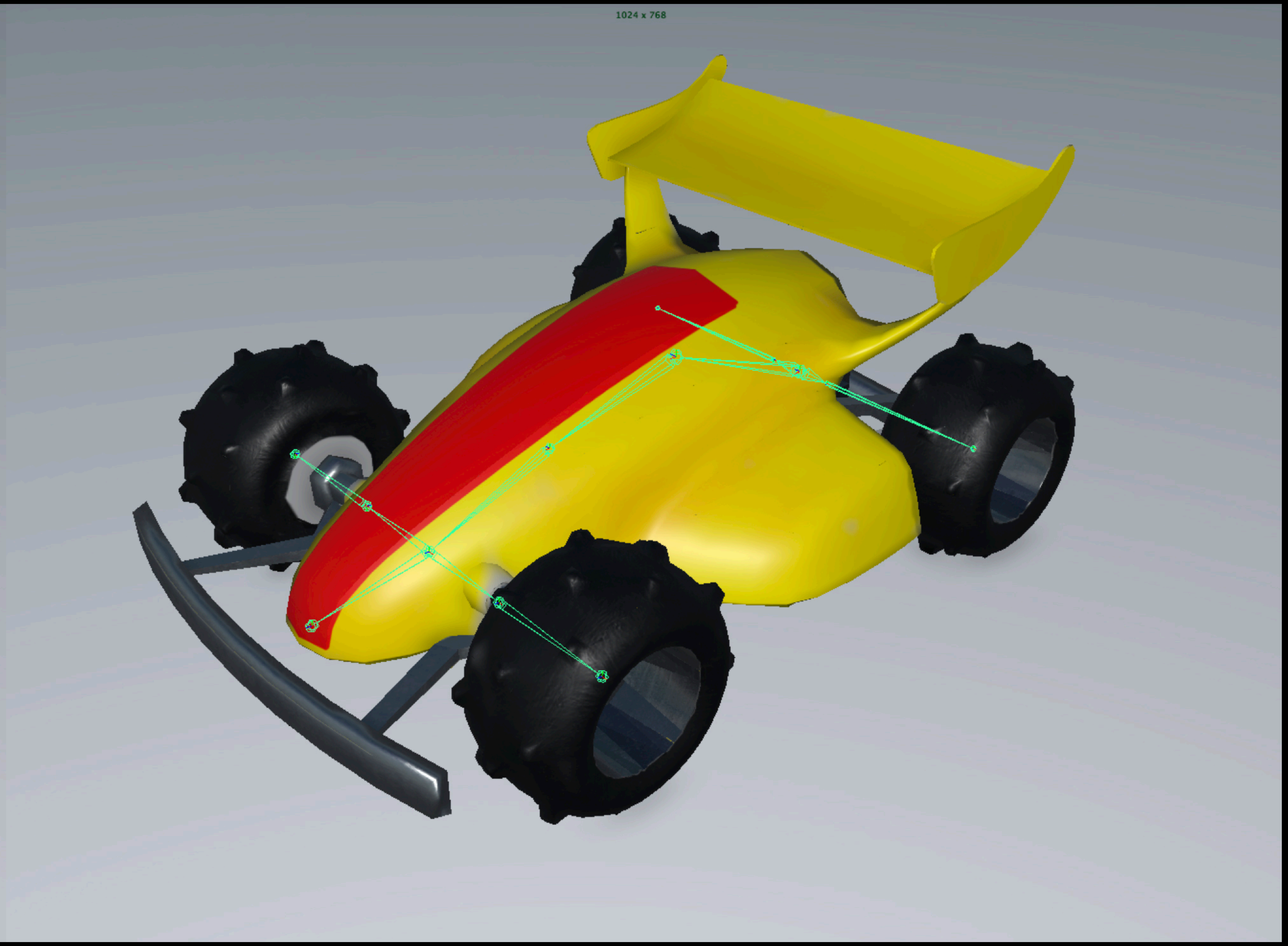
Embedded skeleton



Skinned Character Animation

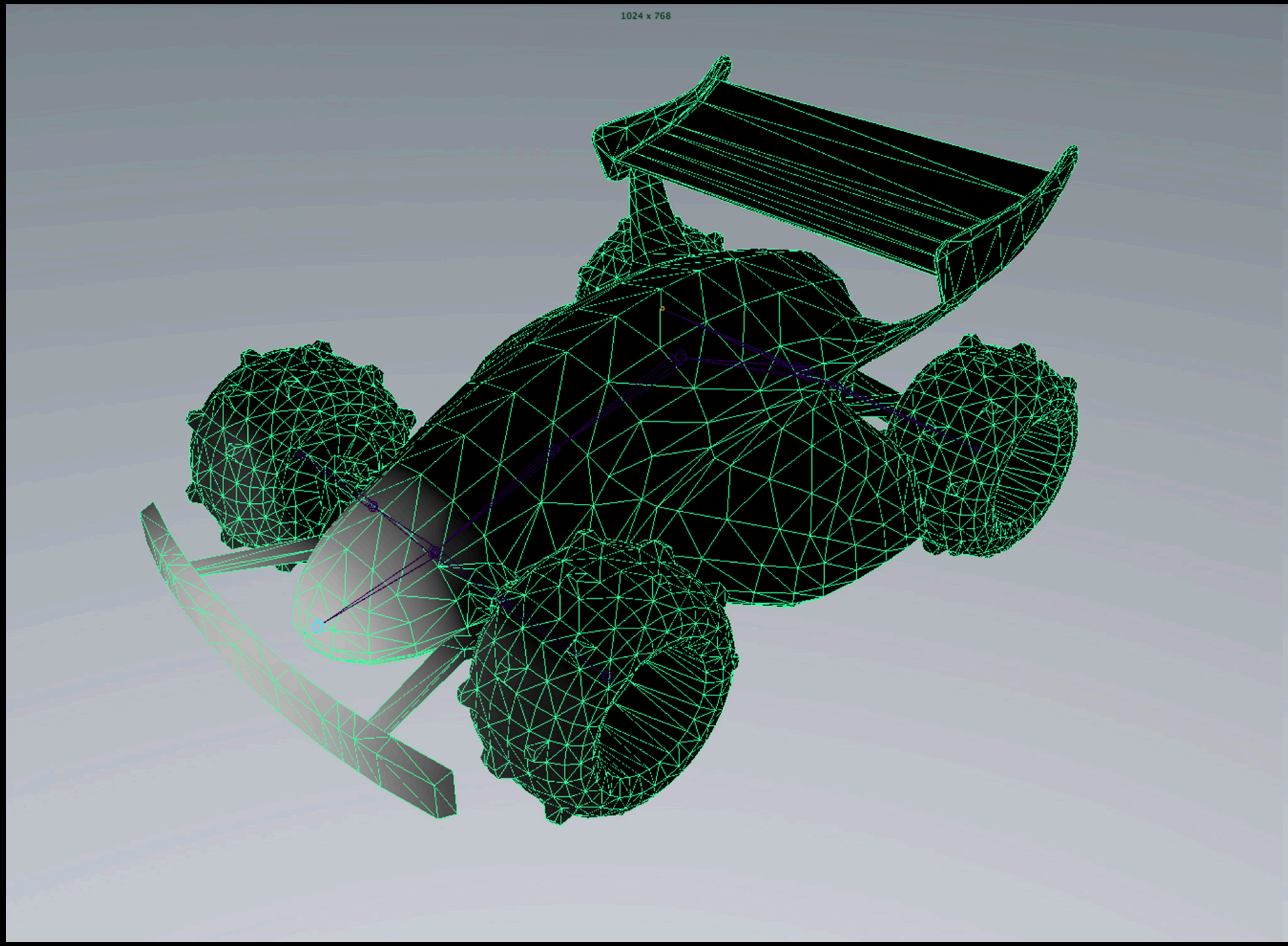
Vertex weighting to joints

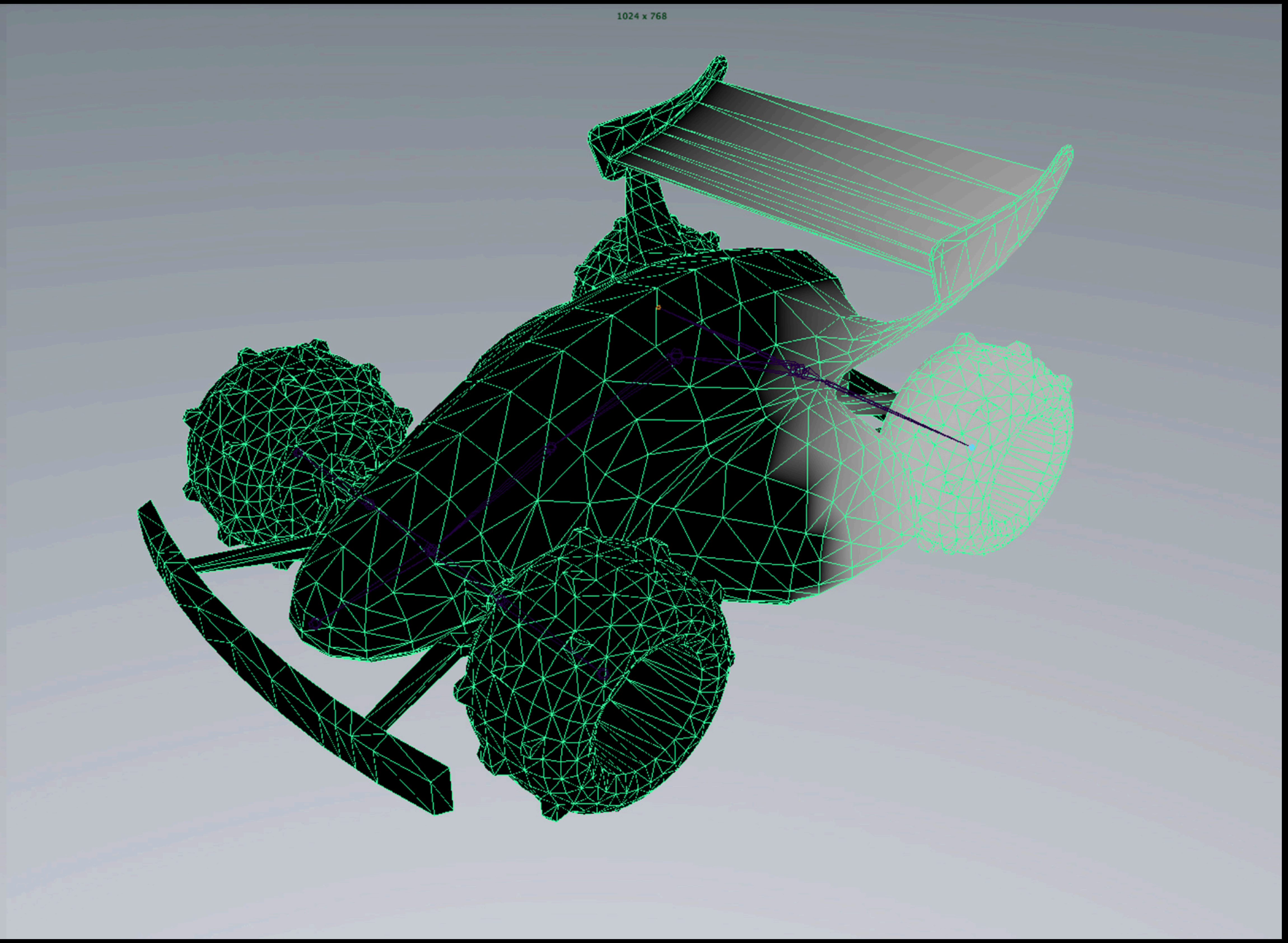




Skinned Character Animation

Vertex weighting to joints





Skinned Character Animation

Vertex blending in shader

```
{
  ...
  float4 position = vertex.pos;
  packed_uchar4 jIdx = vertex.jointIndices;
  packed_float4 w = vertex.jointWeights;

  float4 skinnedPosition = w[0] * (palette[jIdx[0]] * modelPosition) +
                           w[1] * (palette[jIdx[1]] * modelPosition) +
                           w[2] * (palette[jIdx[2]] * modelPosition) +
                           w[3] * (palette[jIdx[3]] * modelPosition);


  ...
}
```

Skinned Character Animation

Input vertex attributes

```
{
  ...
  float4 position = vertex.pos;
  packed_uchar4 jIdx = vertex.jointIndices;
  packed_float4 w = vertex.jointWeights;

  float4 skinnedPosition = w[0] * (palette[jIdx[0]] * modelPosition) +
                           w[1] * (palette[jIdx[1]] * modelPosition) +
                           w[2] * (palette[jIdx[2]] * modelPosition) +
                           w[3] * (palette[jIdx[3]] * modelPosition);


  ...
}
```

Skinned Character Animation

Per vertex joint indices

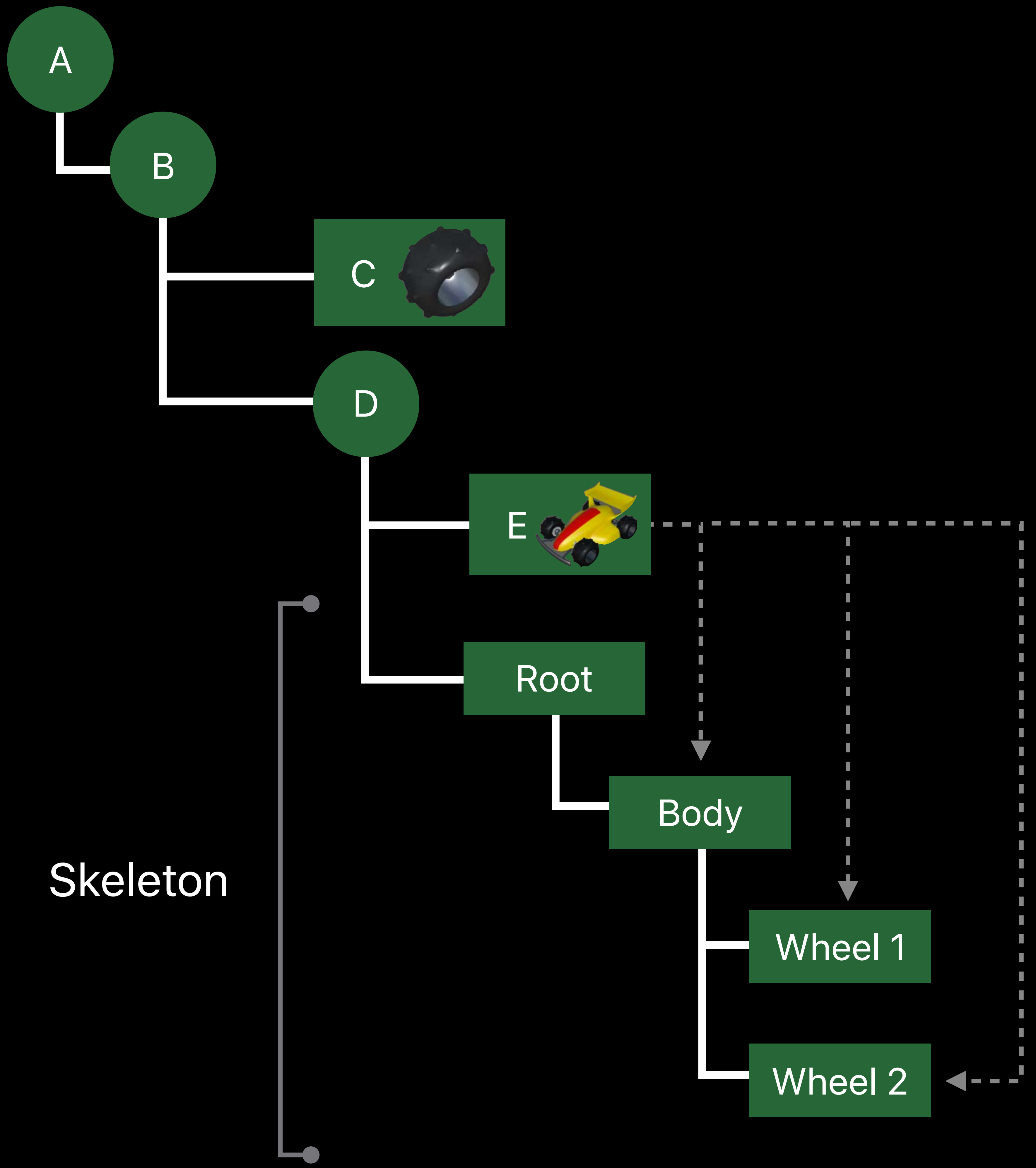
```
{
  ...
  float4 position = vertex.pos;
  packed_uchar4 jIdx = vertex.jointIndices;
  packed_float4 w = vertex.jointWeights;

  float4 skinnedPosition = w[0] * (palette[jIdx[0]] * modelPosition) +
                           w[1] * (palette[jIdx[1]] * modelPosition) +
                           w[2] * (palette[jIdx[2]] * modelPosition) +
                           w[3] * (palette[jIdx[3]] * modelPosition);


  ...
}
```

Skinned Character Animation

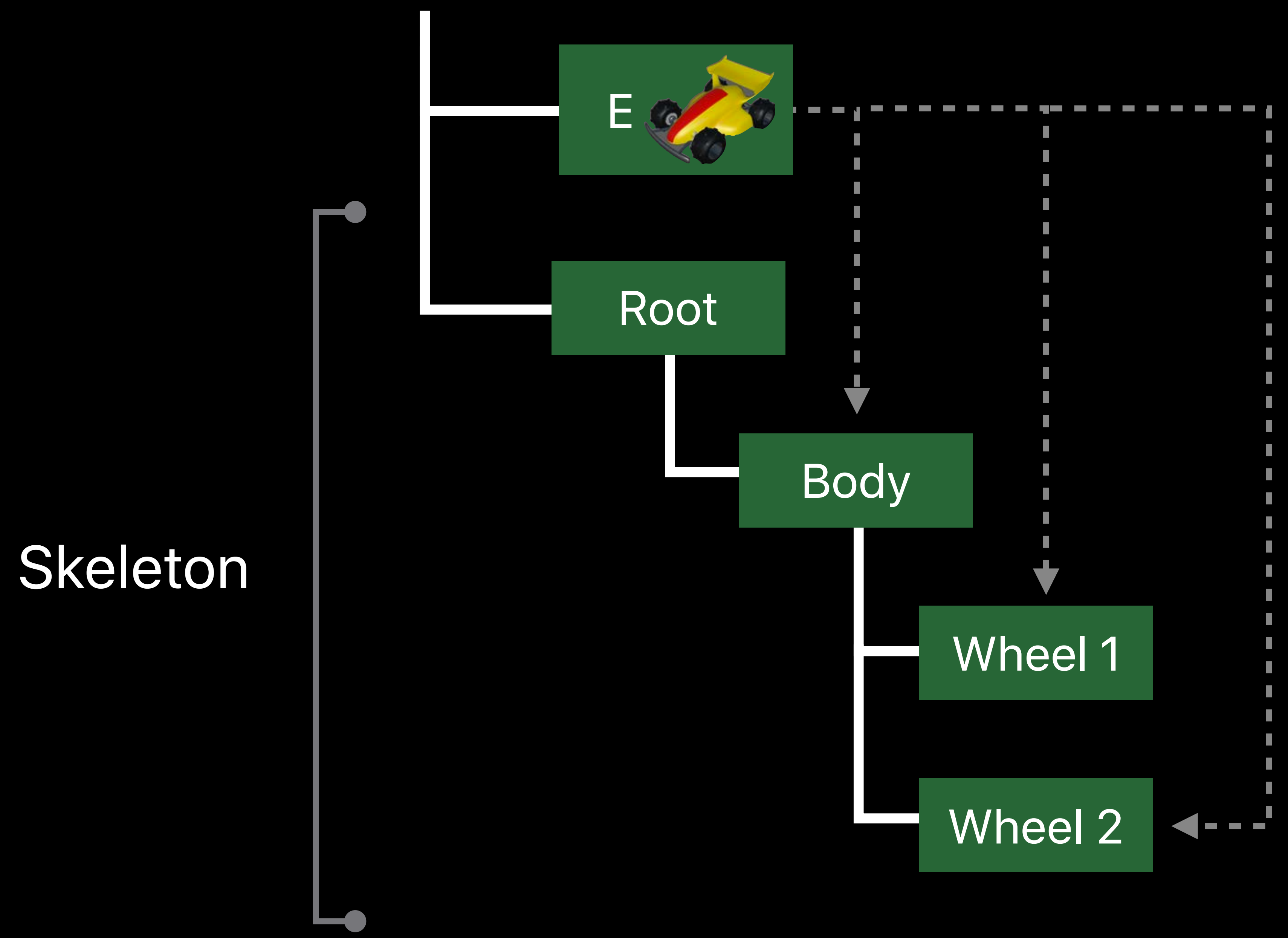
Indexed bones for weight each vertex

```
{
  ...
  float4 position = vertex.pos;
  packed_uchar4 jIdx = vertex.jointIndices;
  packed_float4 w = vertex.jointWeights;

  float4 skinnedPosition = w[0] * (palette[jIdx[0]] * modelPosition) +
                           w[1] * (palette[jIdx[1]] * modelPosition) +
                           w[2] * (palette[jIdx[2]] * modelPosition) +
                           w[3] * (palette[jIdx[3]] * modelPosition);

  ...
}
```

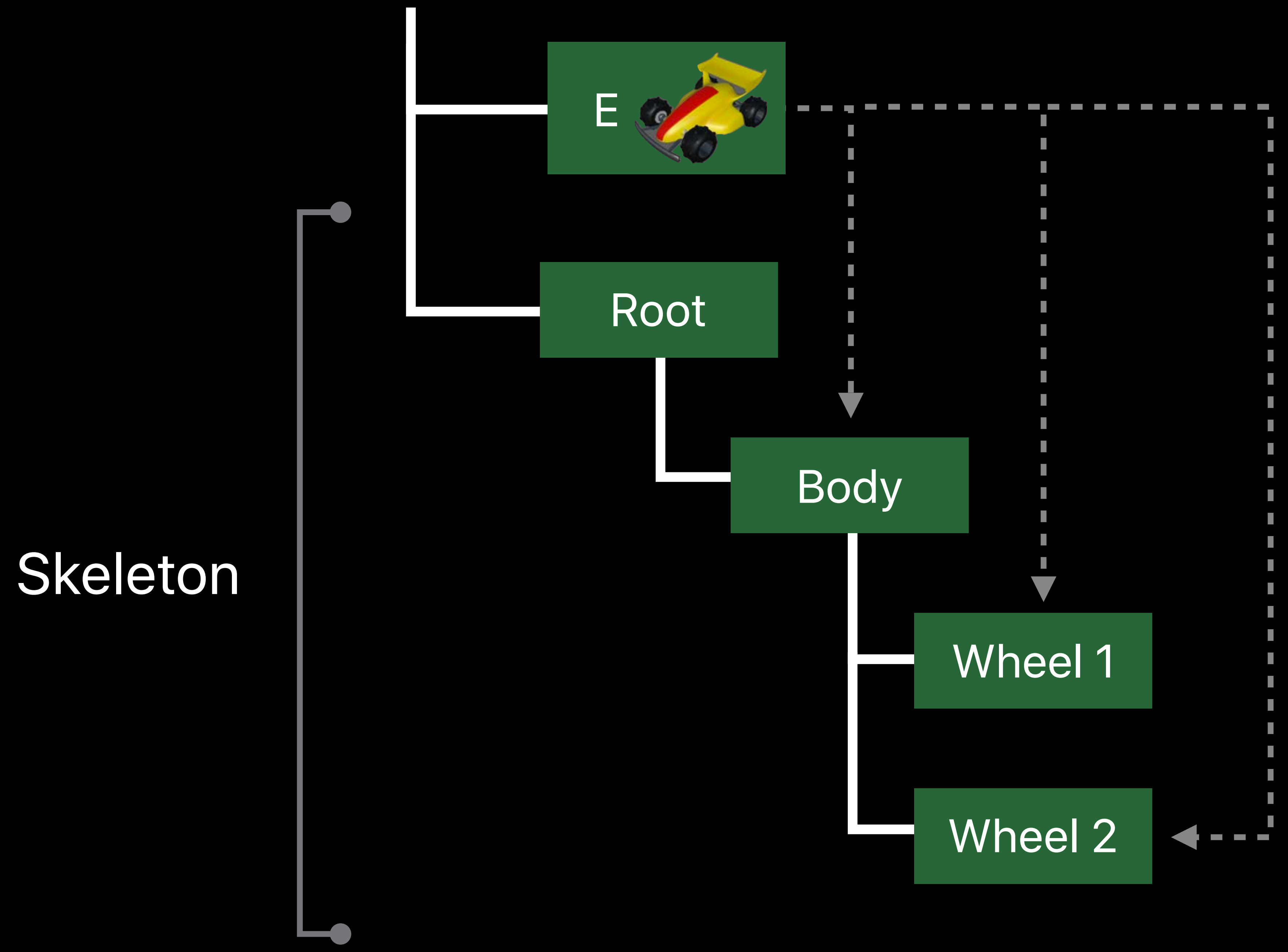
Skinned Character Animation



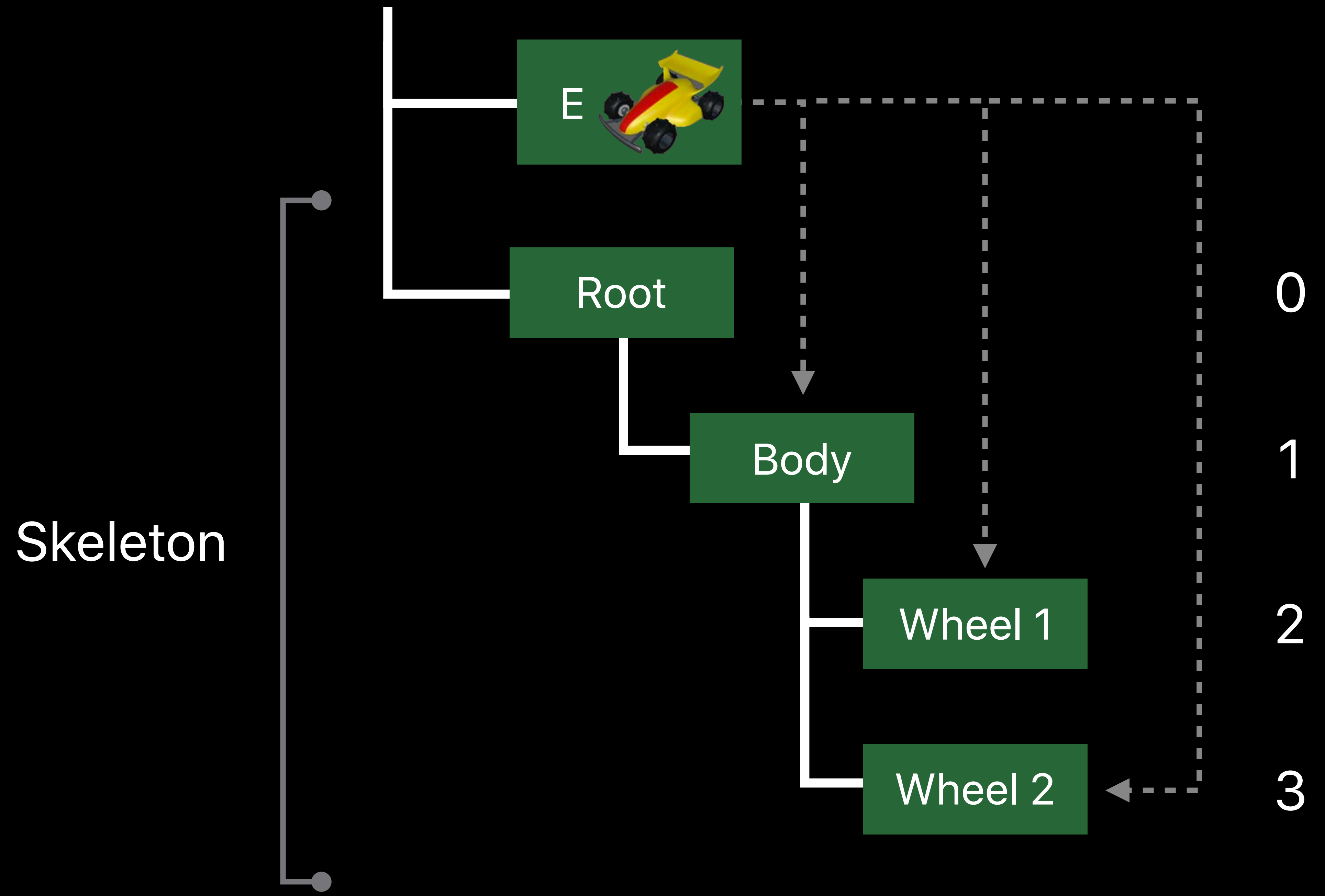
Skinned Character Animation



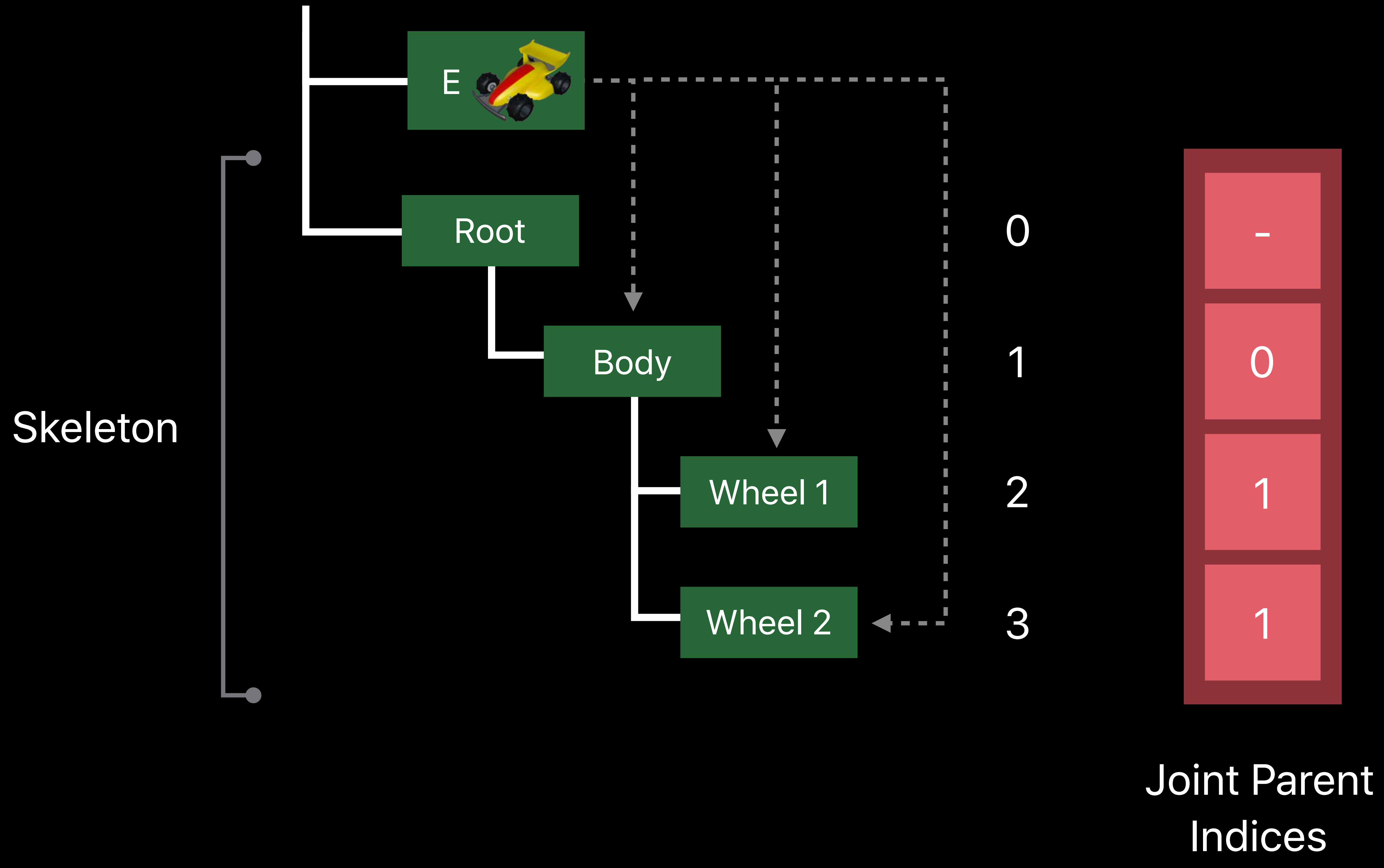
Skinned Character Animation



Skinned Character Animation

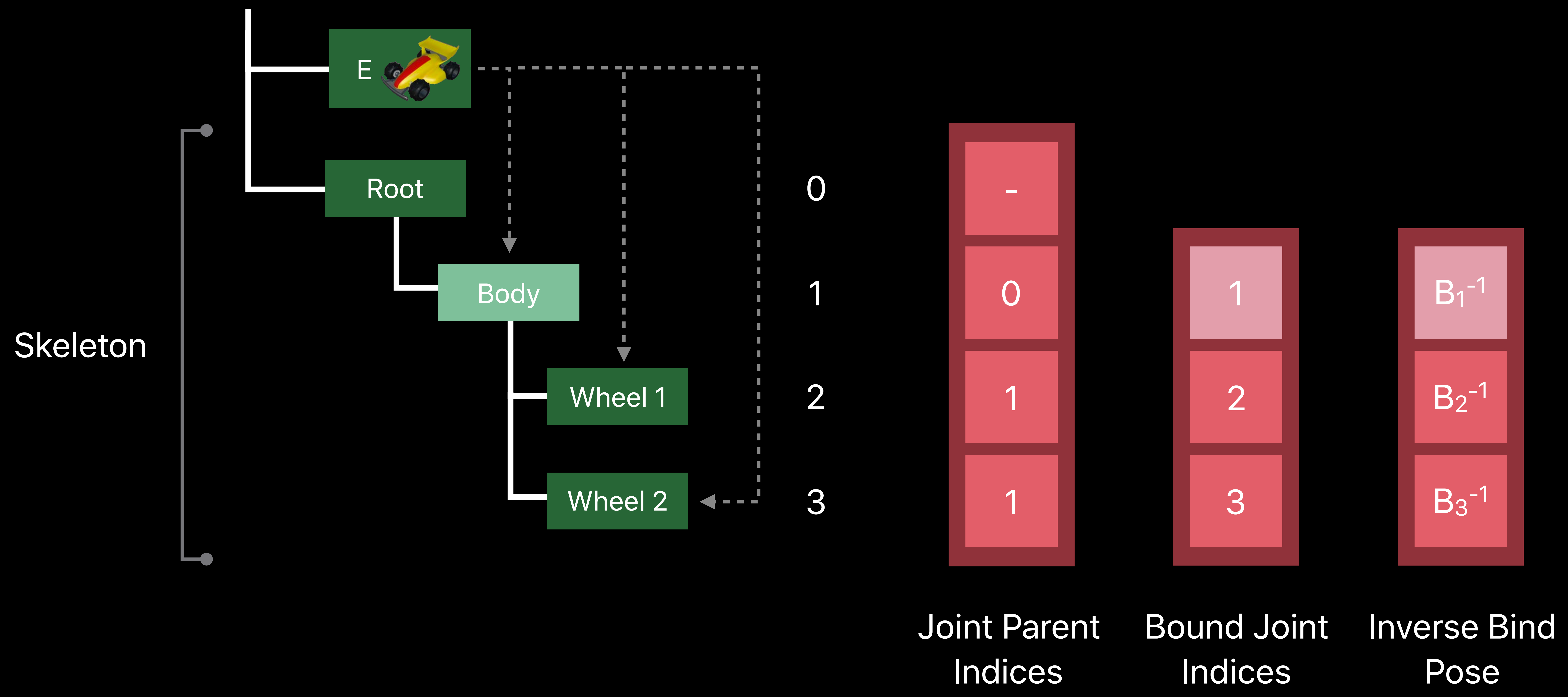


Skinned Character Animation



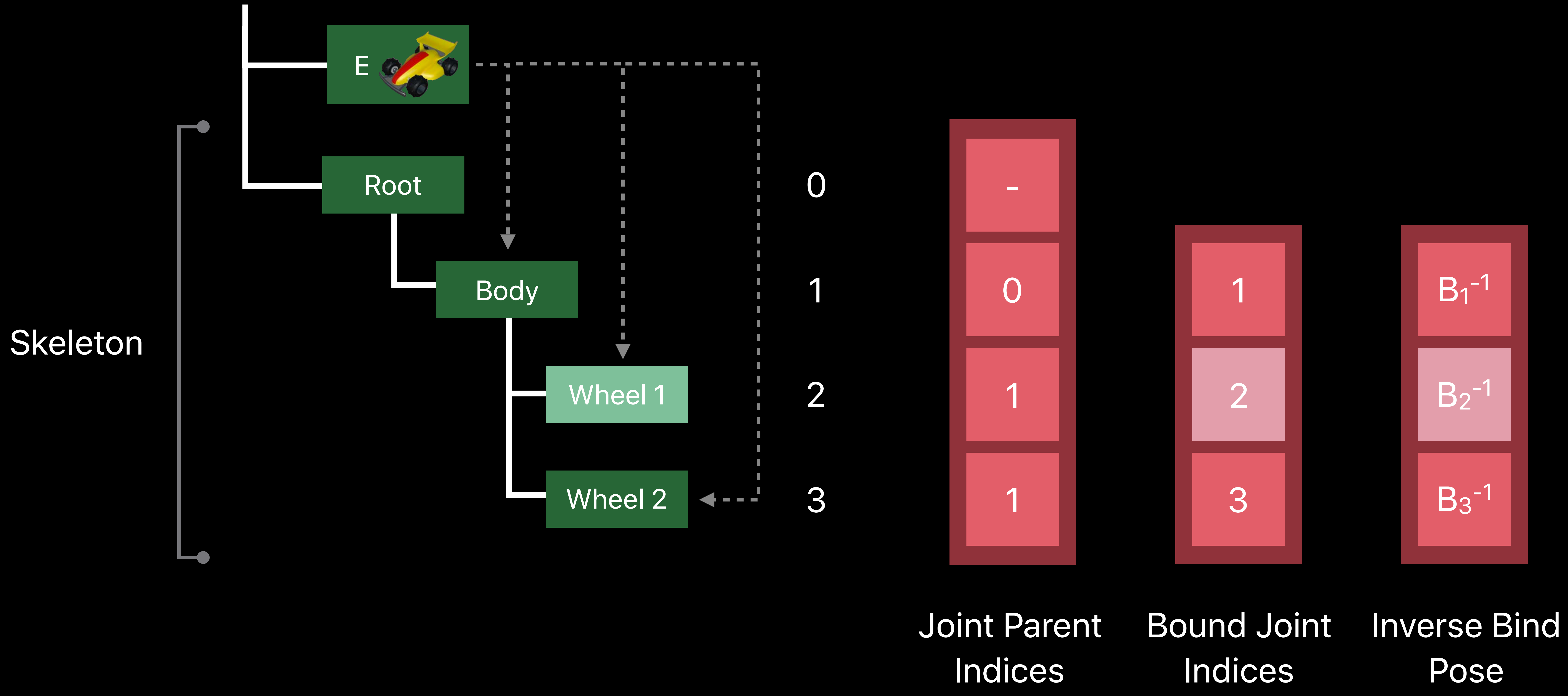
Skinned Character Animation

Bind Poses + Joint Indices



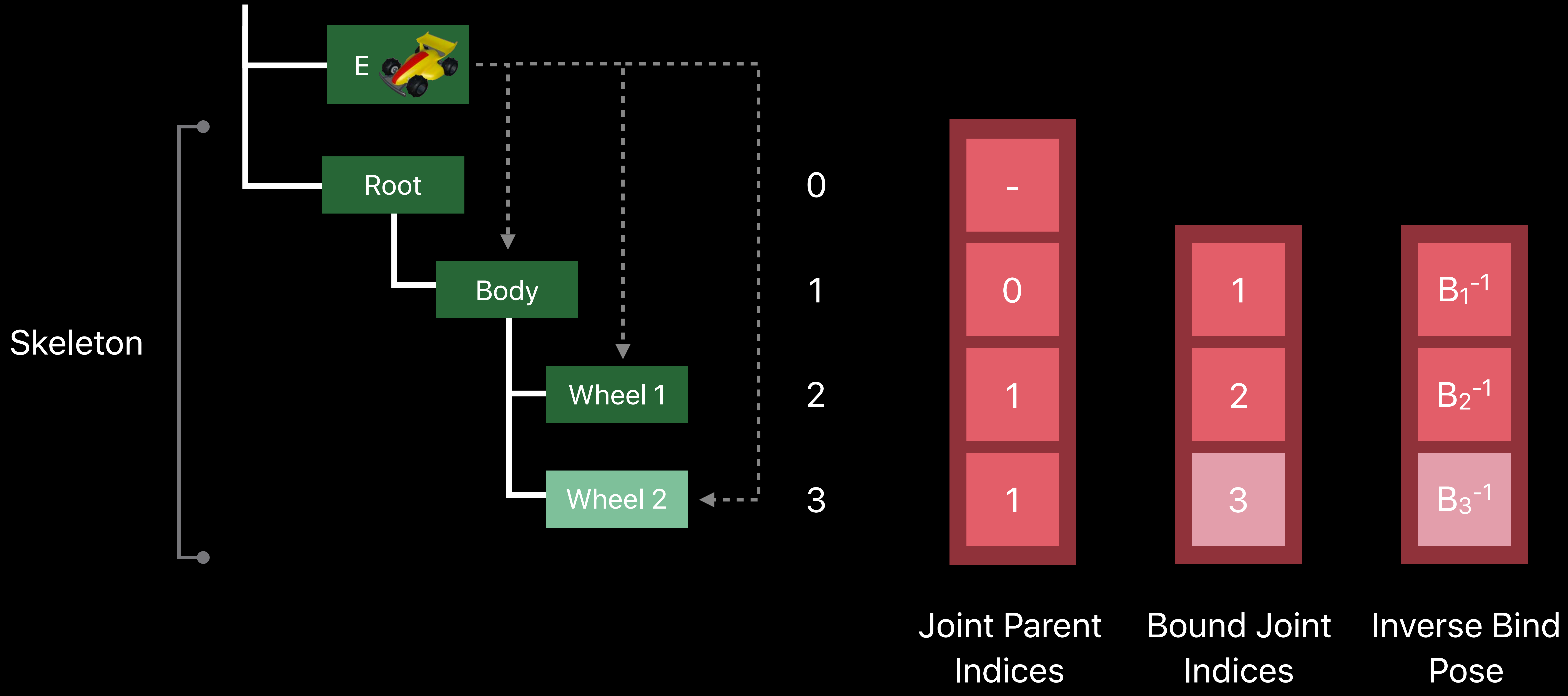
Skinned Character Animation

Bind Poses + Joint Indices



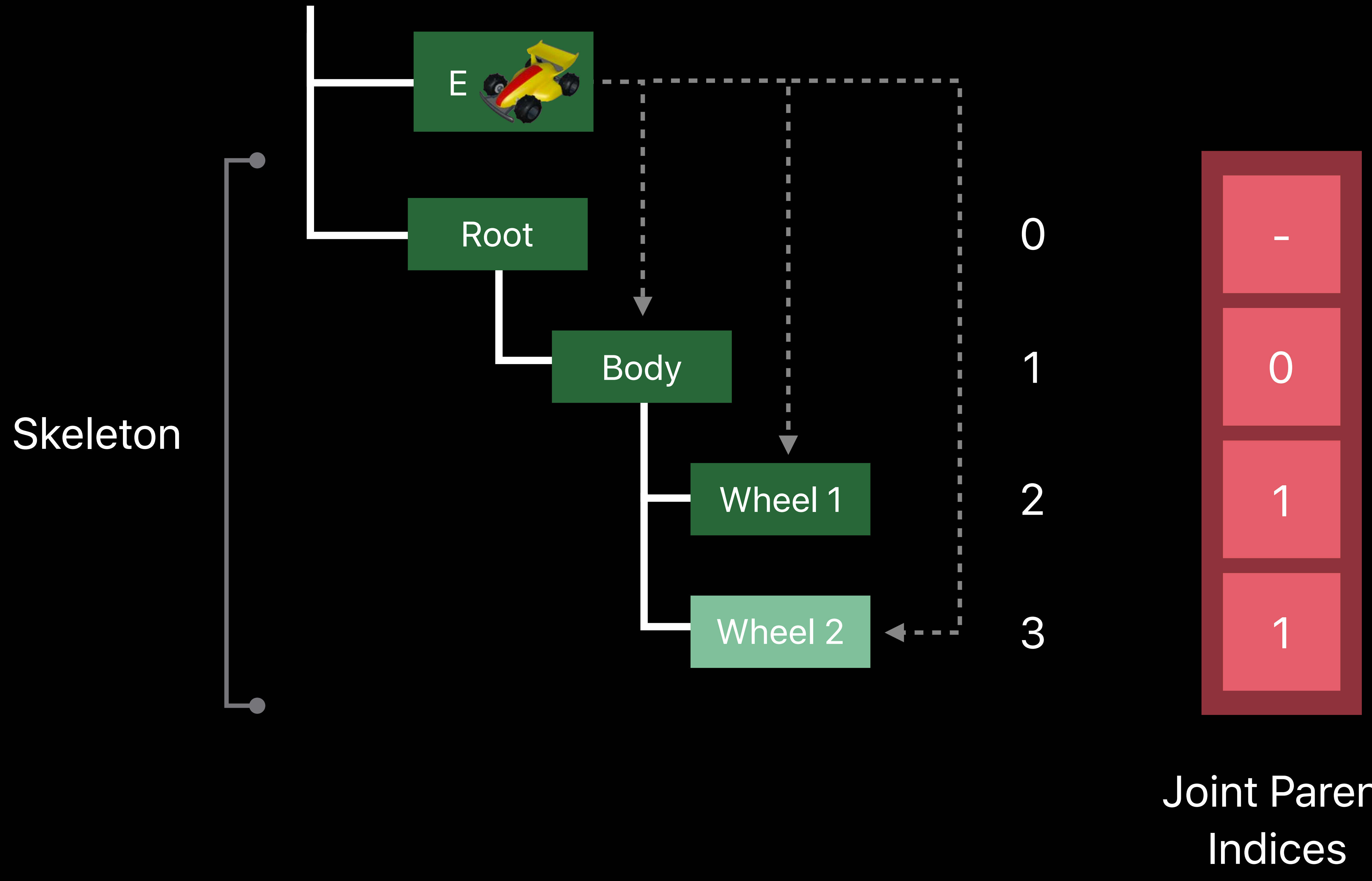
Skinned Character Animation

Bind Poses + Joint Indices



Skinned Character Animation

Time sample animation



Joint Parent Indices	Bound Joint Indices	Inverse Bind Pose
-		
0	1	B_1^{-1}
1	2	B_2^{-1}
1	3	B_3^{-1}

Animation Clip

Root	$t_0 \ldots t_N$
Body	$t_0 \ldots t_N$
W1	$t_0 \ldots t_N$
W2	$t_0 \ldots t_N$

Skinned Character Animation

```
// for every mdlObject in MDLAsset:
if let mesh = mdlObject as? MDLMesh {
  if let skin =
    object.componentConforming(to:MDLSkinDeformerComponent.self)
                                       as? MDLSkinDeformerComponent {

   let inverseBindTransforms = skin.jointBindTransforms().map{simd_inverse($0)}

    ...
}
```

Skinned Character Animation

```
// for every mdlObject in MDLAsset:
if let mesh = mdlObject as? MDLMesh {
  if let skin =
    object.componentConforming(to:MDLSkinDeformerComponent.self)
                                       as? MDLSkinDeformerComponent {

  let inverseBindTransforms = skin.jointBindTransforms().map{simd_inverse($0)}

    ...
}
```

Skinned Character Animation

Mesh Data	Descriptors Vertex + Index Buffers
Scene Composition Data	Parent Indices, Mesh Indices Instance Count
Transform Data	Local Transforms Animated Local Transforms
Material Data	Material Uniforms Texture Paths

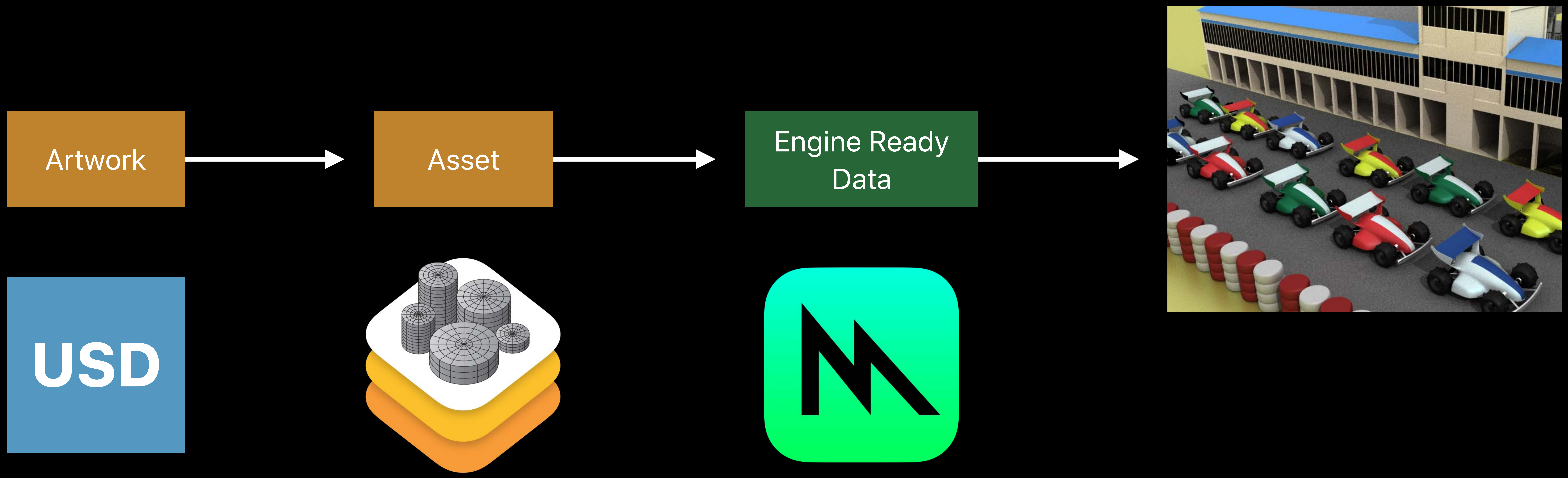
Skinned Character Animation

Mesh Data	Descriptors Vertex + Index Buffers
Scene Composition Data	Parent Indices, Mesh Indices Instance Count
Transform Data	Local Transforms Animated Local Transforms Animation Clips
Material Data	Material Uniforms Texture Paths
Skinning Data	Inverse Bind Transforms Joint to Palette Mapping Skeleton Parent Indices

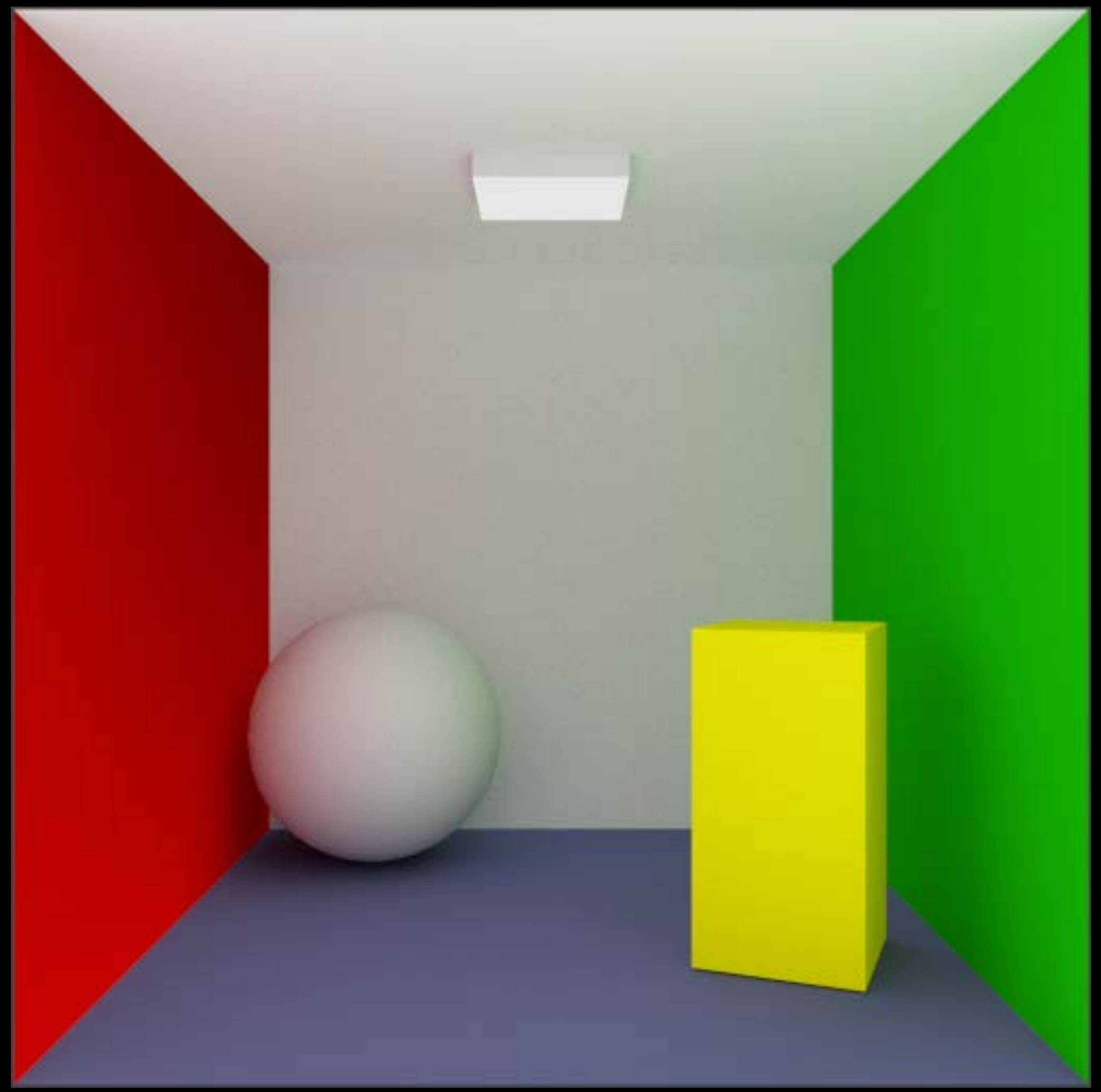
Demo

Instancing and Characters

Recap

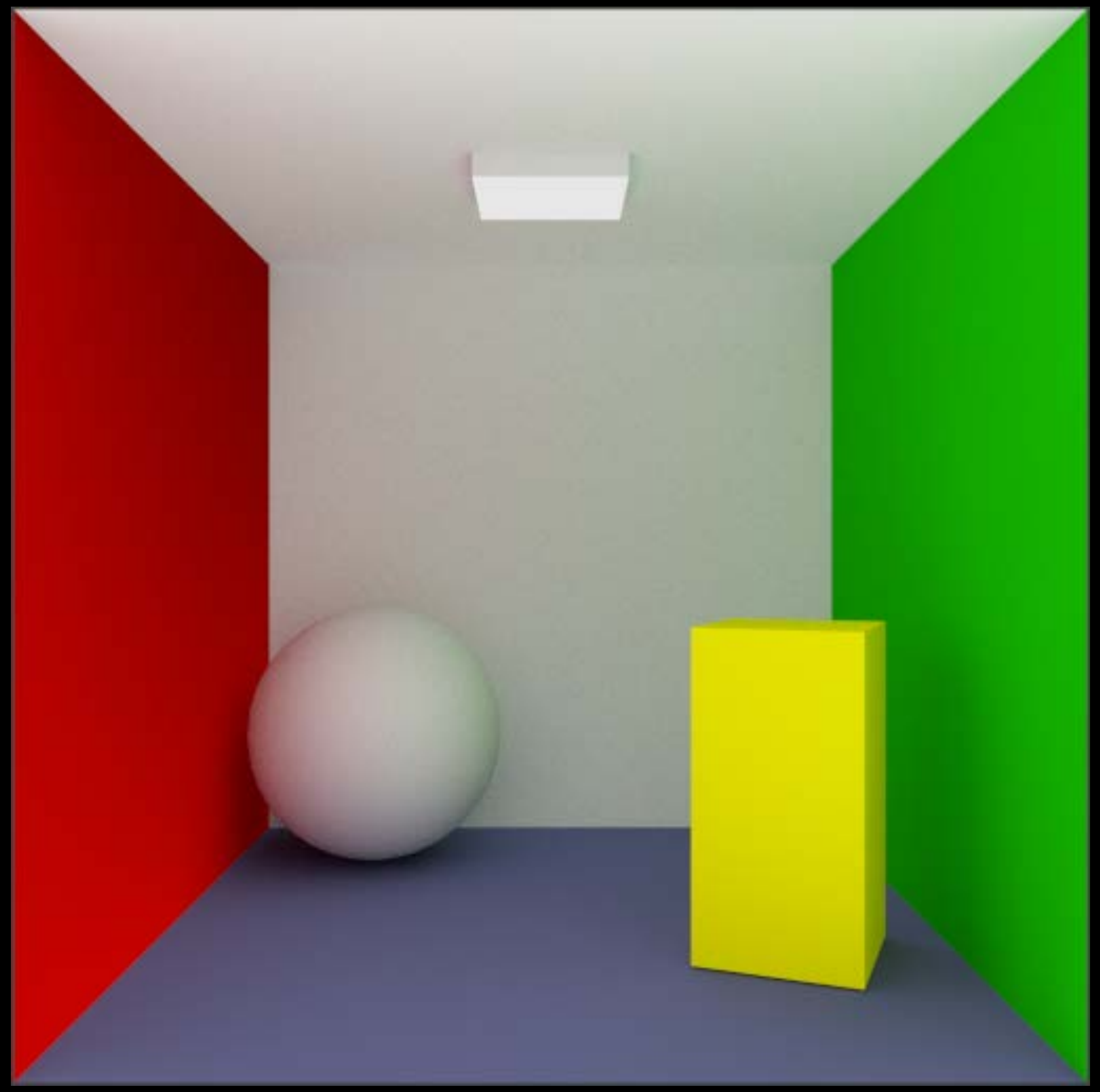


Enhancements



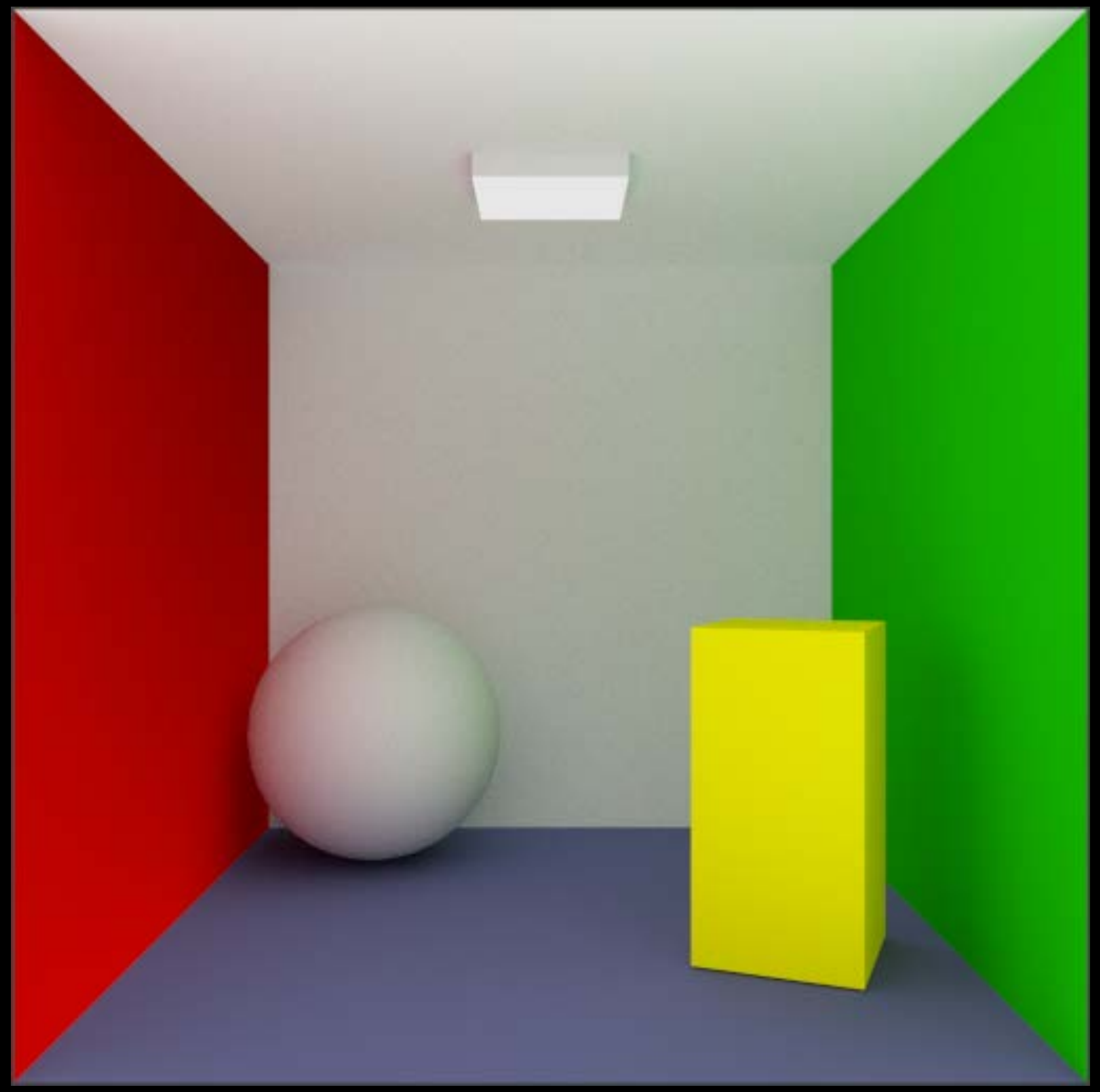
Light Mapping

Enhancements


Light Mapping

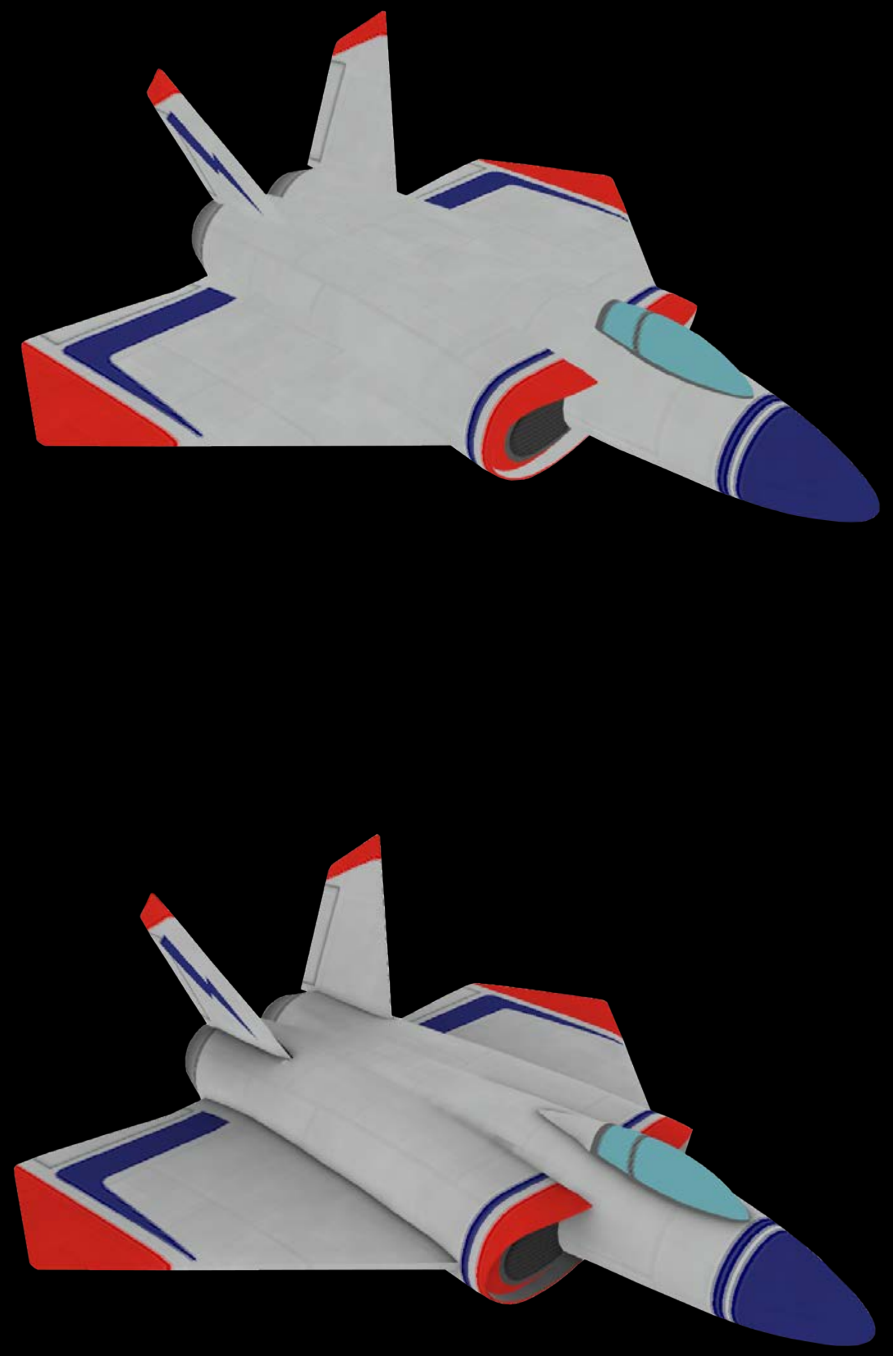
UV Unwrapping

Enhancements



Light Mapping

UV Unwrapping



Ambient Occlusion

Enhancements

Image-based lighting

More Information

https://developer.apple.com/wwdc17/610

Related Information

Managing 3D Assets with Model I/O	WWDC 2015
Introducing Metal 2	WWDC 2017
What’s New in SceneKit	WWDC 2017

WWDC17