Today

- Scene Transitions
- GameObject Hierarchies and Transformations
- GameObject Component Scripting

Scene Transitions

- Scenes must be added to the **Build Settings**.
- **First scene** (at index 0) in Build Settings is the **starting scene**.
- Starting scenes require no initial transition.
- **Scene names matter** as you will be referencing them in code when doing a transition.

Scene Transitions (Obsolete)

- Transitioning between scenes destroys the previous scene's hierarchy.

```
Application.LoadLevel("SceneName");
```

- Scenes can also be loaded on top of the existing scene.

```
Application.LoadLevelAdditive("SceneName");
```

Scene Transitions

```
SceneManager.LoadScene(name,mode);
SceneManager.LoadScene(index, mode);
```

More information:
https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html

GameObject
Transform

- Fundamental component of a GameObject.
- Defines a GameObject in 3D space.
  - **translation** (position), **rotation** (orientation), and **scale** (size).
- A GameObject without a transform cannot be defined in a scene; therefore, it cannot exist.

GameObject Transform Hierarchy

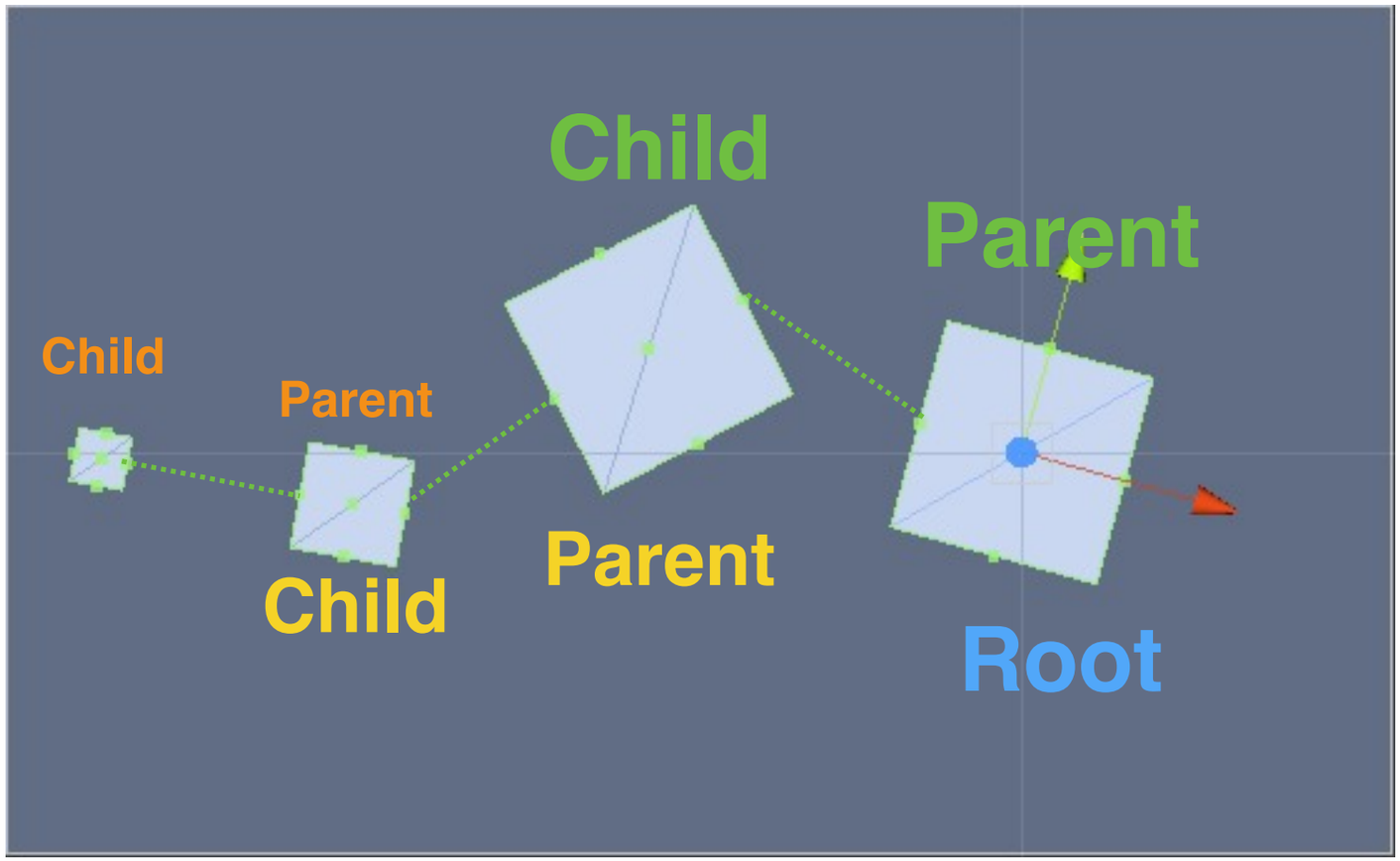
- GameObject Hierarchy == Transform Hierarchy

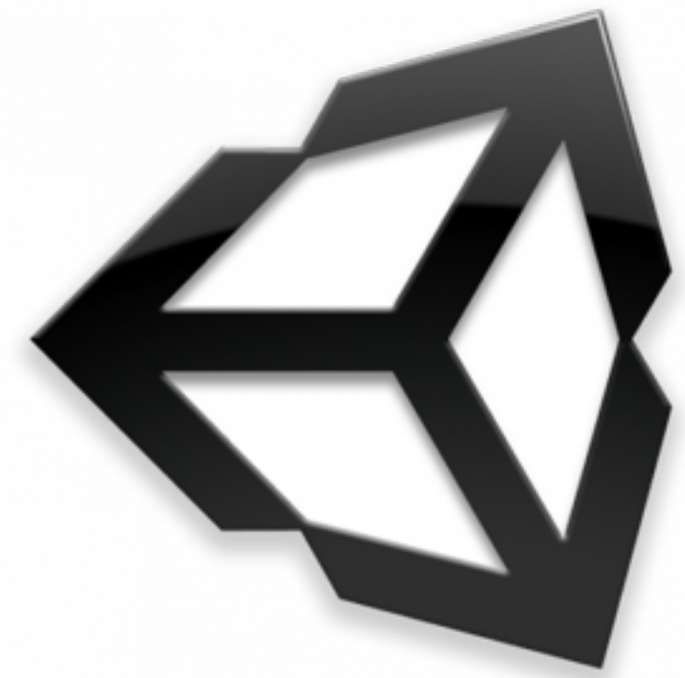
- Defines a **relative transform space** between GameObjects inside a hierarchy.

- Enables child GameObjects to **inherit** their parent's relative transforms (a.k.a. **parent-child** relationship).

- This inheritance does not affect the individual relative spaces of game objects; instead, it **affects global transformations** in world-space.

Hierarchy Example





Root & Container GameObjects

- Top-level (root) **containers** of other GameObjects.

- They define game object concepts
    - E.g. *Table* is a root game object containing *TableTop* and four *TableLeg* child game objects.

- They allow child game objects to be transformed all at once.
    - E.g. moving the *Table* moves all of its pieces.

Root & Container GameObjects

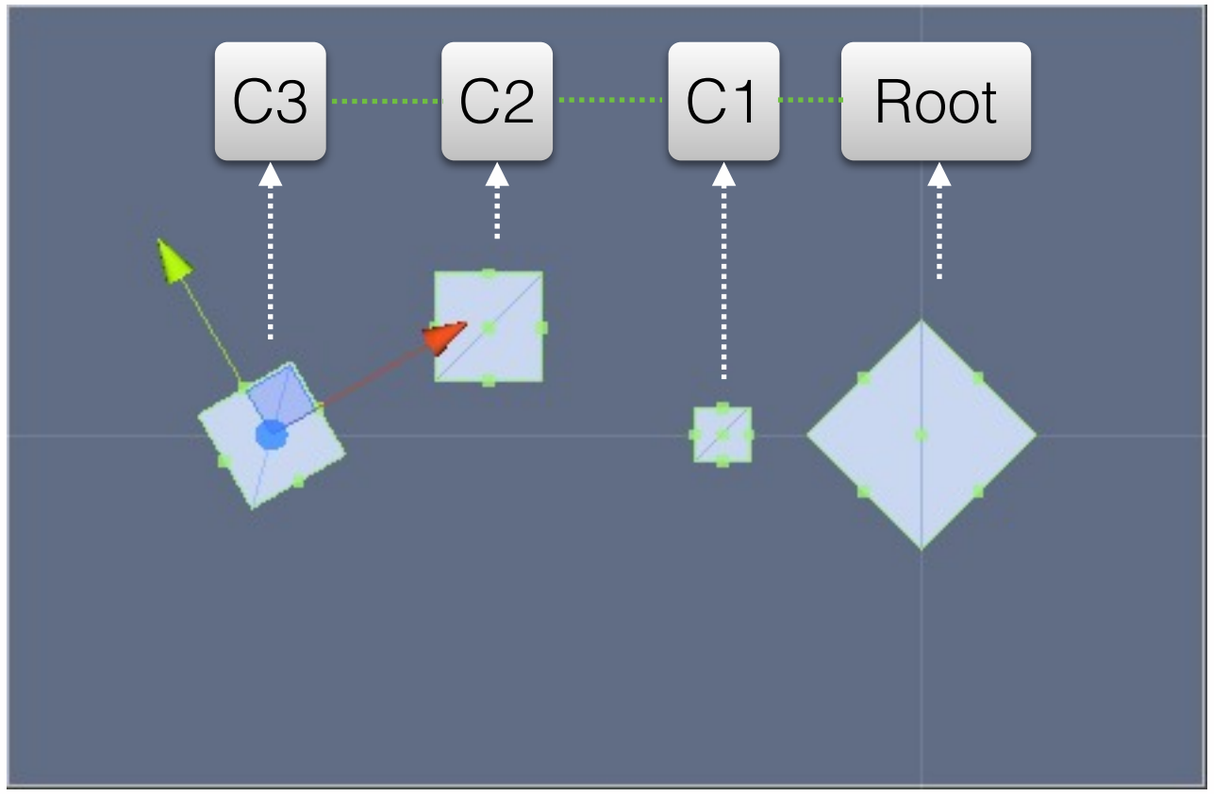
- Often used purely as “container” game objects.
- Empty GameObjects with no mesh geometry.
- Similar to folders on the hard drive.

Root & Container GameObjects

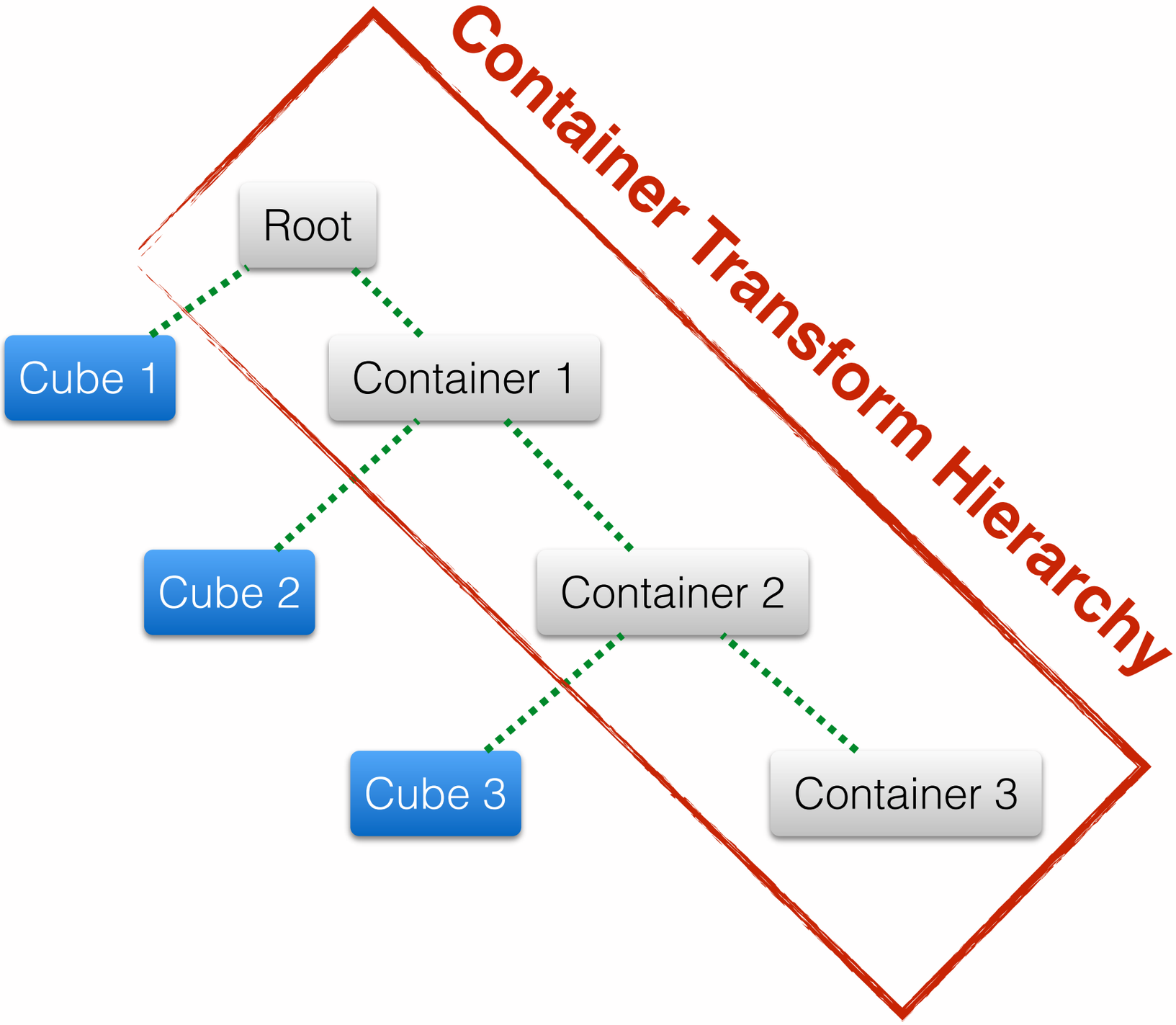
- Create a container by selecting *GameObject* > *Create Empty* from the application menu.

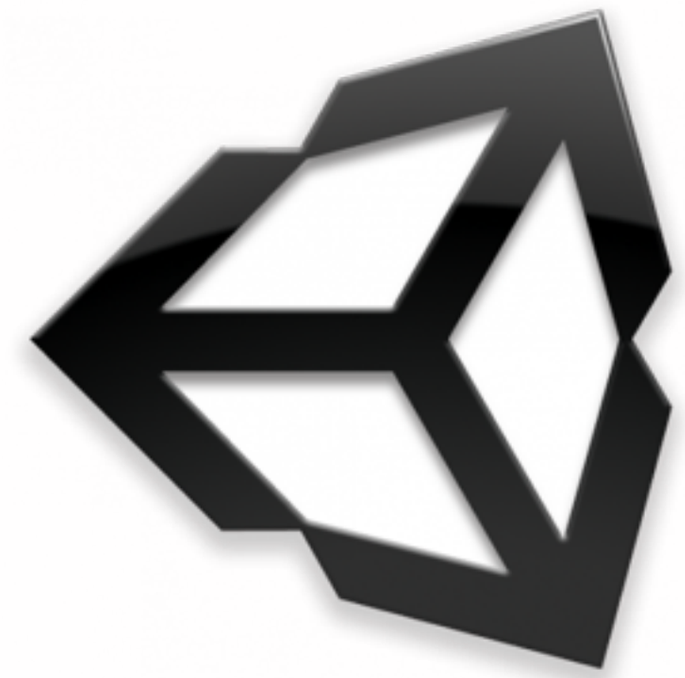
- Define its hierarchy by adding child game objects to it.

Root & Container Example



Root & Container Example





Unity API

- Creating an empty game object:

```
new GameObject(“Game object name”);
```

Unity API

- Creating an empty game object and saving a reference to it:

```
GameObject myNewObject =

    new GameObject(“Game object name”);
```

Unity API

- Instantiating an existing game object (cloning):

```
GameObject myGameObject = new GameObject();

GameObject.Instantiate(myGameObject);
```

Unity API

- How do we instantiate a **non-existing** existing game object from a **prefab**?

Unity API

```
public class MyScript : MonoBehaviour
{
    public GameObject somePrefab;

    void Start ()
    {
        // Create a game object from ‘somePrefab’.
        GameObject.Instantiate(somePrefab);
    }
}
```

Unity API

- How do we save a reference to the instantiated game object?

Unity API

- Instantiating an existing game object (cloning) and saving a **reference** to it:

```
// Save a ‘clone’ game object reference.
GameObject clone =
   (GameObject) GameObject.Instantiate(myGameObject);

// Save a ‘clone’ game object reference.
GameObject clone =
   GameObject.Instantiate(myGameObject) as GameObject;
```

Unity API

- Instantiating a **non-existing** game object from a **prefab** and saving a **reference** to it:

```
public class MyScript : MonoBehaviour
{
    public GameObject treePrefab;
    private GameObject myTree;

    void Start ()
    {
        this.myTree = (GameObject)
                GameObject.Instantiate(somePrefab);
    }
}
```

Unity API

- Accessing game object's Transform values:

```
// Local position?

// Global position?

// Rotation?

// Scale?
```

Unity API

- Accessing game object's Transform values:

```
gameObject.transform.localPosition;
```

gameObject.transform.position (global)

```
gameObject.transform.eulerAngles;
```

```
gameObject.transform.localScale;
```

Unity API

```
GameObject myObject = new GameObject();

myObject.transform.localPosition;

myObject.transform.position (global)

myObject.transform.eulerAngles;

myObject.transform.localScale;
```

Unity API

- Setting a game object as a child of another.

```
GameObject go1 = new GameObject(“Child”);

GameObject go2 = new GameObject(“Parent”);

go1.transform.parent = go2.transform;
```

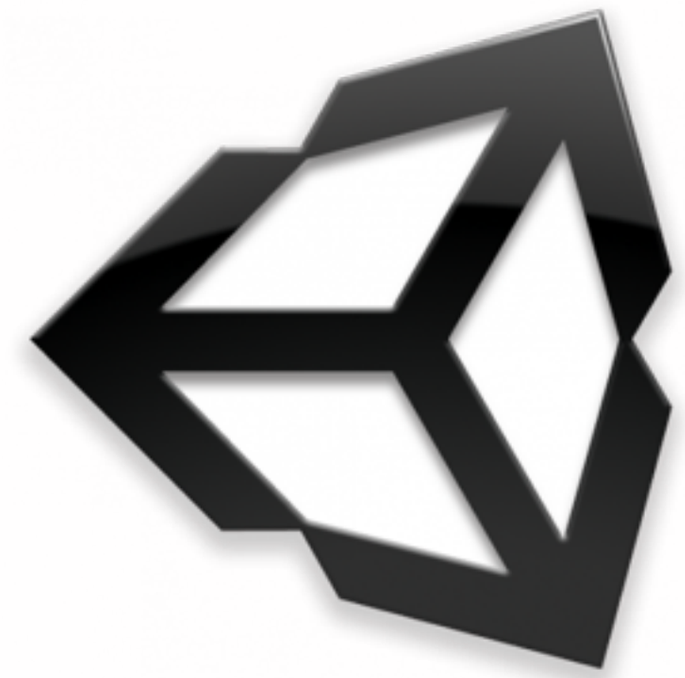
Unity API

- Setting a game object as a child of another.

```
GameObject go1 = new GameObject(“Child”);

GameObject go2 = new GameObject(“Parent”);

go1.transform.parent = go2.transform;
```



Unity API

```
public class RotationController : MonoBehaviour
{
    public float speed = 0;

    void Update ()
    {
        // Rotate my game object each frame
        // by ‘speed’ amount of degrees.
        this.gameObject.transform.Rotate(
        new Vector3(0, speed, 0);
    }
}
```

Unity API

- To **get** a script from a game object:

```
RotationController rc =
gameObject.GetComponent<RotationController>();

// We now have a reference ‘rc’ to a script
// of type RotationController.
// Let’s use it to set the speed of rotation.
rc.speed = 15;
```

Unity API

- To **add** a script to a game object:

```
this.gameObject.AddComponent<ScriptType>();

// Add RotationController script to my game object.
this.gameObject.AddComponent<RotationController>();

// Add RotationController script to my game object
// and save its reference inside ‘rc’.
RotationController rc =
   this.gameObject.AddComponent<RotationController>();

rc.speed = 15;
```

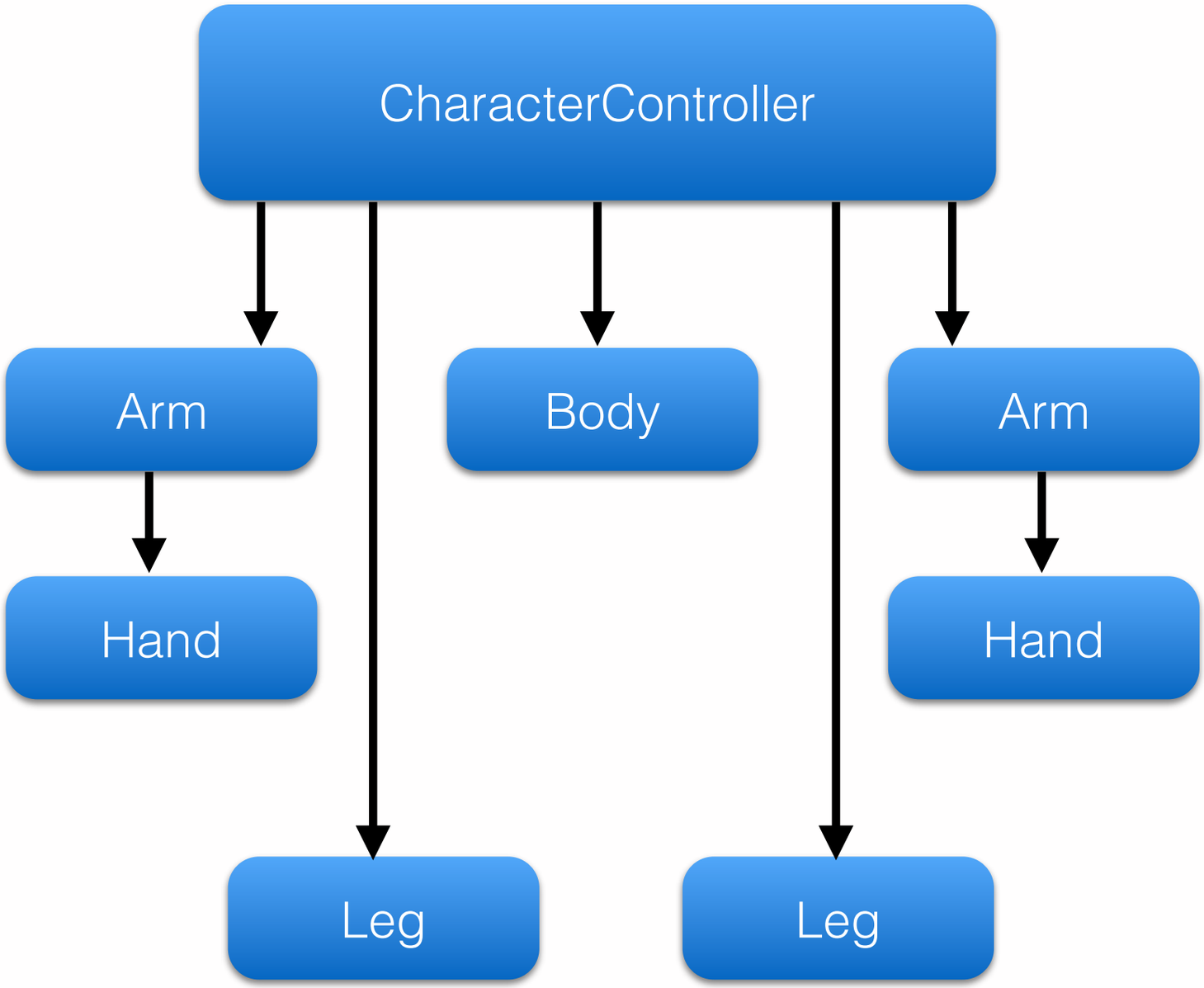
Unity API

- To **get** a script from a game object:

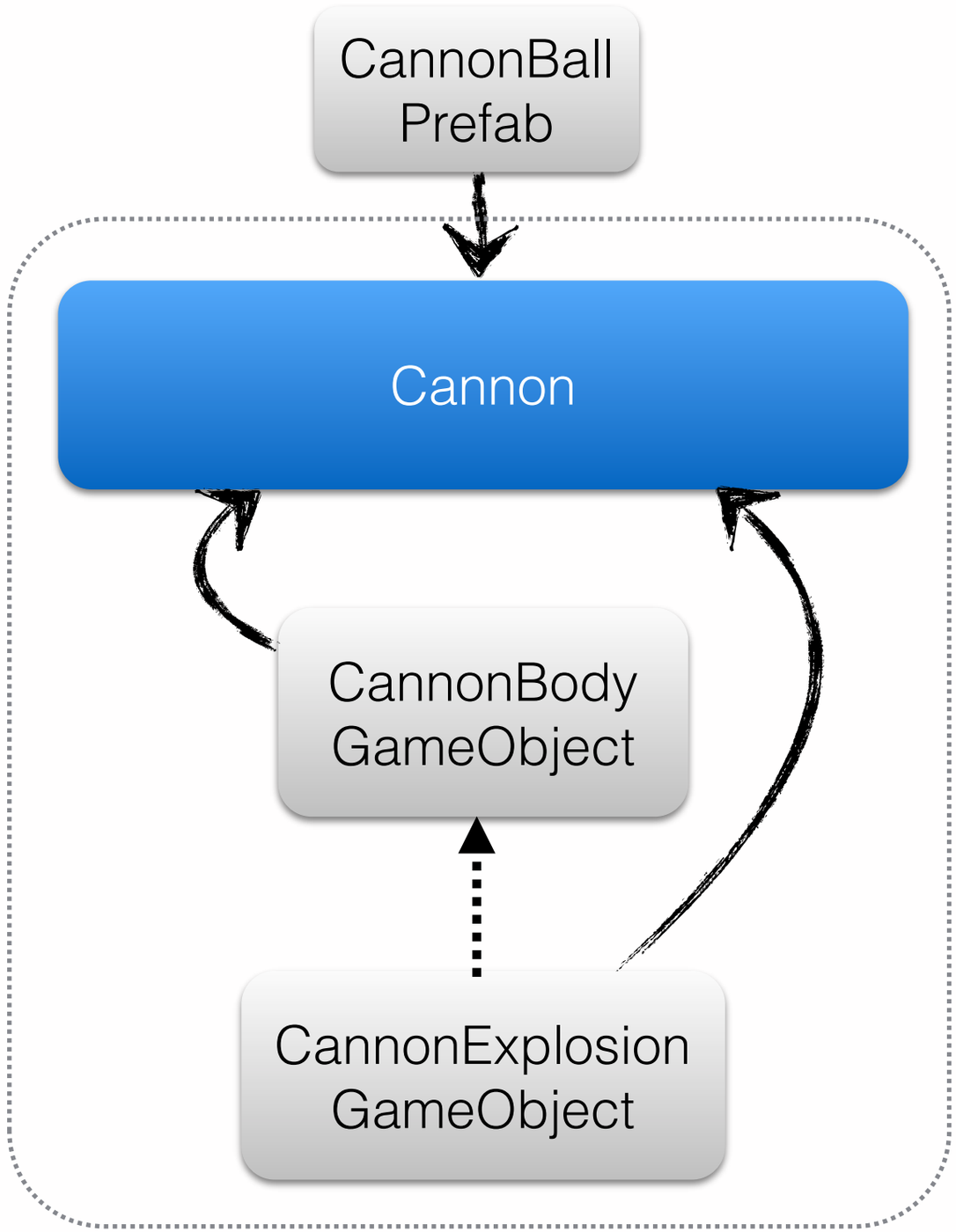
```
RotationController rc =
gameObject.GetComponent<RotationController>();

if (script == null)
{
  Debug.LogError("Could not find script!");
}
```

Script Hierarchy via Transform Hierarchy



Single Script



Script Components

- Can have references between each other, either via:
  - **Sibling relationship** in one GameObject.
  - **Hierarchical relationship** between parent-child game objects.
  - **Indirect relationship** — referenced from another hierarchy in the scene.
- Scripts can be **called**, **added and retrieved** from game object, as well as **removed**, **activated** and **deactivated** at runtime by other scripts.