Object Pool Manager Approach

Object Pool Manager Approach

Implementing A Pool Manager

We'll create a Struct to define the information we need to be able to create an object pool. This will be for a 'primary' component in the gameobject at the root of the prefab

```
[System.Serializable]
1 reference
public struct Pool
{
    public int poolSize;
    public GameObject prefab;
    public string componentType;
}
```

Object Pool Manager Approach

Implementing A Pool Manager

We'll create a Struct to define the information we need to be able to create an object pool. This will be for a 'primary' component in the gameobject at the root of the prefab

```
[System.Serializable]
1 reference
public struct Pool
{
    public int poolSize;
    public GameObject prefab;
    public string componentType;
}
```

The **poolsize** will be how many components we want to create in the pool for this prefab

Object Pool Manager Approach

Implementing A Pool Manager

We'll create a Struct to define the information we need to be able to create an object pool. This will be for a 'primary' component in the gameobject at the root of the prefab

```
[System.Serializable]
1 reference
public struct Pool
{
    public int poolSize;
    public GameObject prefab;
    public string componentType;
}
```

The **prefab** will be the 'template' that contains the root gameobject, that the component is attached to, that we want to create in the pool

Object Pool Manager Approach

Implementing A Pool Manager

We'll create a Struct to define the information we need to be able to create an object pool. This will be for a 'primary' component in the gameobject at the root of the prefab

```
[System.Serializable]
1 reference
public struct Pool
{
    public int poolSize;
    public GameObject prefab;
    public string componentType;
}
```

The **componentType** will be the primary component type on the prefab gameobject that we want to store in the pool for future manipulation

Object Pool Manager Approach

Implementing A Pool Manager

We'll create a Struct to define the information we need to be able to create an object pool. This will be for a 'primary' component in the gameobject at the root of the prefab

```
[System.Serializable]
1 reference
public struct Pool
{
    public int poolSize;
    public GameObject prefab;
    public string componentType;
}
```

Object Pool Manager Approach

Implementing A Pool Manager

```
public class PoolManager : SingletonMonobehaviour<PoolManager>

    [SerializeField] private Pool[] poolArray = null;
```

In the `PoolManager` class itself, we'll use the `Pool` struct to define a member variable called `poolArray` which will be a serialised field of type `Pool[]` (an array of `Pool` objects).

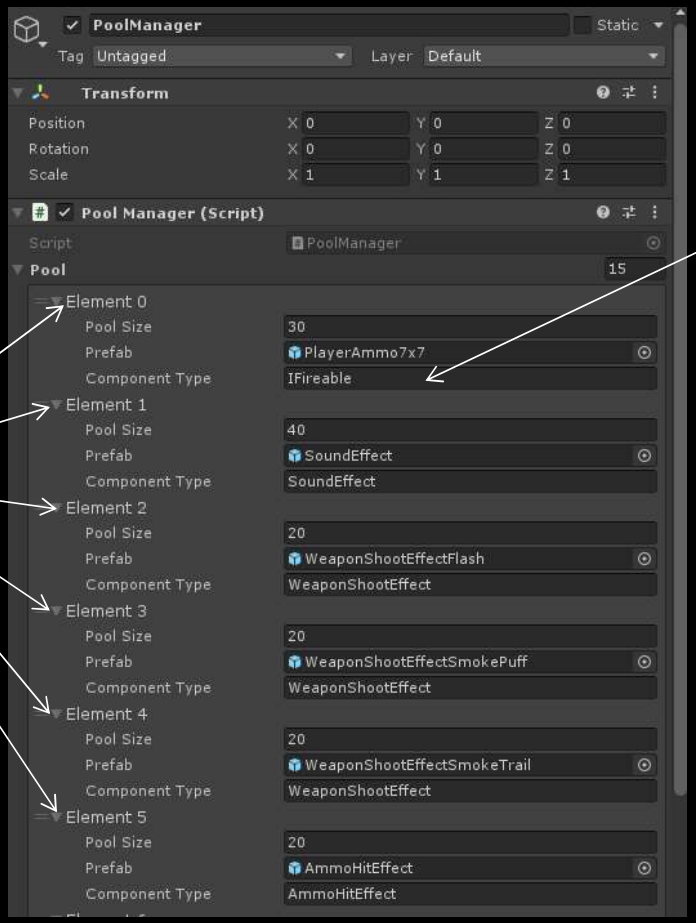
This will allow the user to easily define multiple object pools in the inspector based on different prefabs.

```
[System.Serializable]
1 reference
public struct Pool
{
    public int poolSize;
    public GameObject prefab;
    public string componentType;
}
```

Object Pool Manager Approach

Populate Object Pools In The Inspector

So as we progress through the game build and we create Prefabs that we want to be part of the Object Pool, we can just add them to the PoolManager in the inspector as this example shows

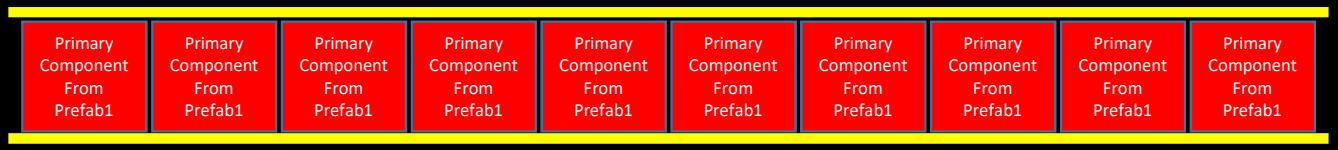


We can also specify 'interfaces' if we want, rather than specific component types. This is beneficial if we have different Ammo prefabs, for example that have different types of Ammo component (e.g. Ammo and AmmoPattern), but they all support the 'IFireable' interface. By doing this, the ammo firing code can just utilise the 'IFireable' interface methods, and not worry about which particular type of Ammo it is.

Object Pool Manager Approach

PoolManager – Creating The Object Pools

Each Object Pool will be implemented as a Queue of Components



Object Pool Queue (Red = Disabled , Green = Enabled)

If required, the gameObject that the component is attached to can be easily accessed using `component.gameObject`. Other components attached to the gameobject can be accessed if required using GetComponent<> e.g. `component.gameObject.GetComponent<>`

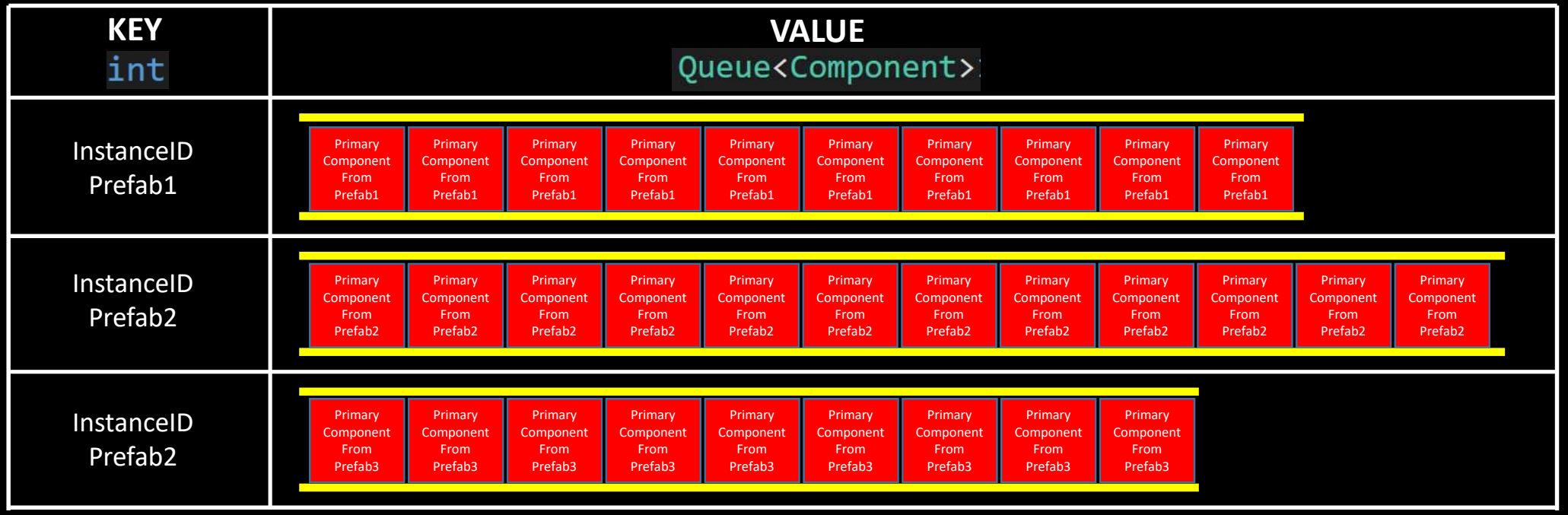
Object Pool Manager Approach

PoolManager – Creating The Object Pools

All the Object Pool Queues will be held in a Dictionary that has the Prefab InstanceID integer value as the key, and the Queue of Components as the value

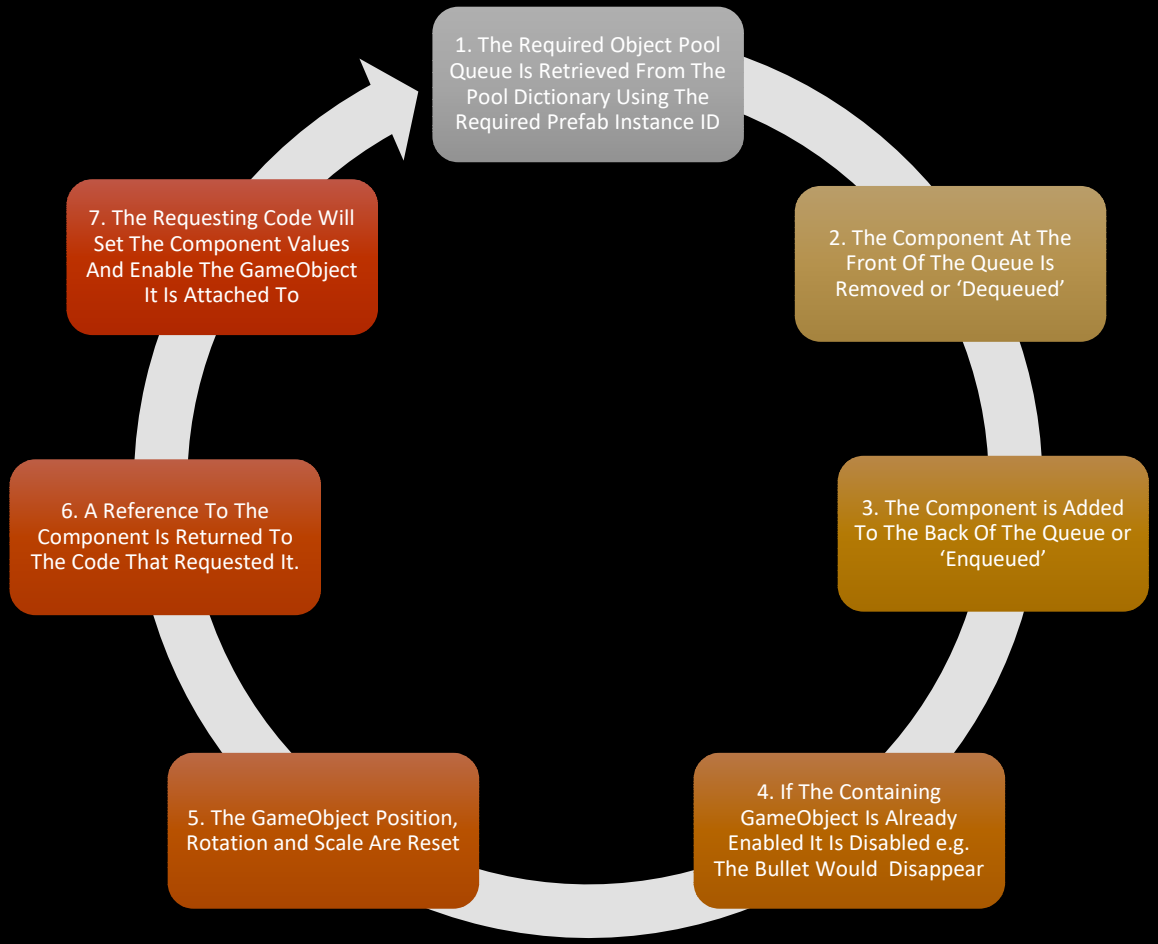
```
private Dictionary<int, Queue<Component>> poolDictionary = new Dictionary<int, Queue<Component>>();
```

poolDictionary



Object Pool Manager Approach

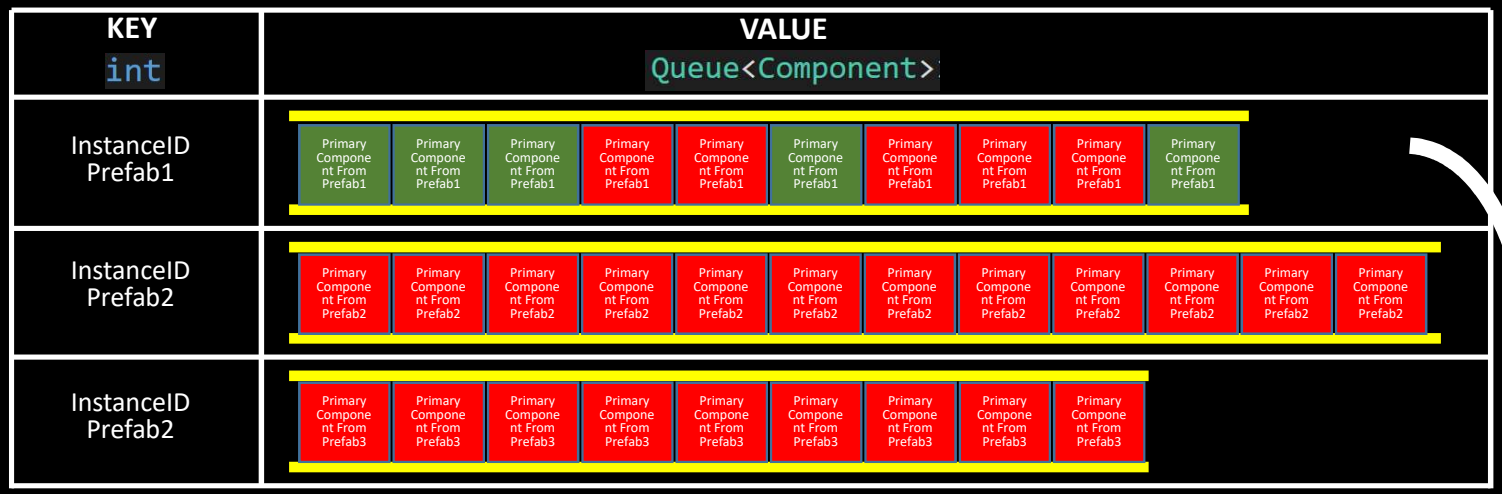
PoolManager – Retrieving Components From The Pool



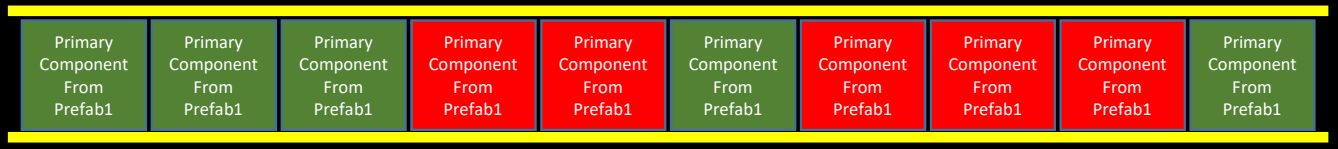
Object Pool Manager Approach

PoolManager – Retrieving Components From The Pool

poolDictionary



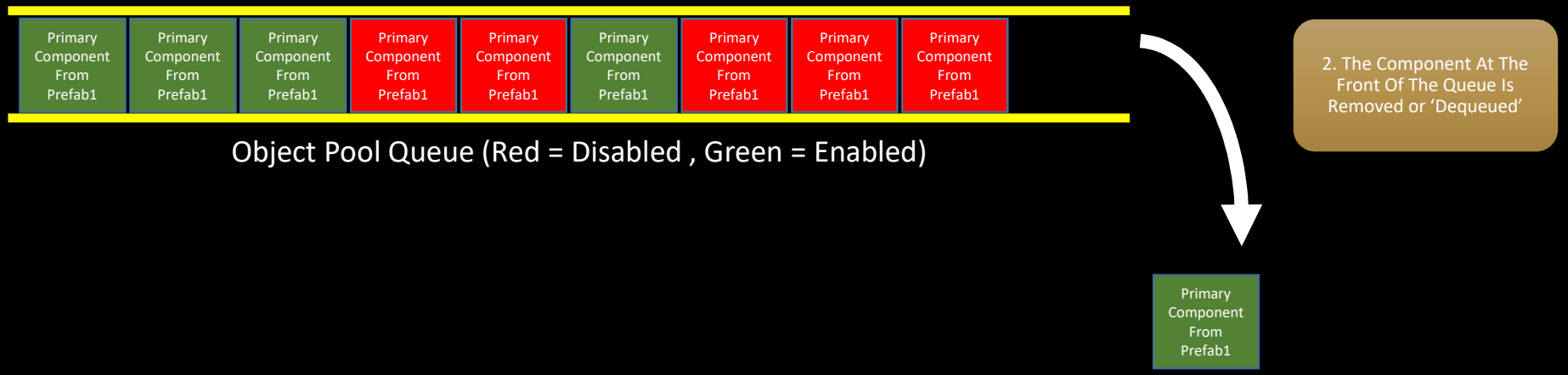
1. The Required Object Pool Queue Is Retrieved From The Pool Dictionary Using The Required Prefab Instance ID



Object Pool Queue (Red = Disabled , Green = Enabled)

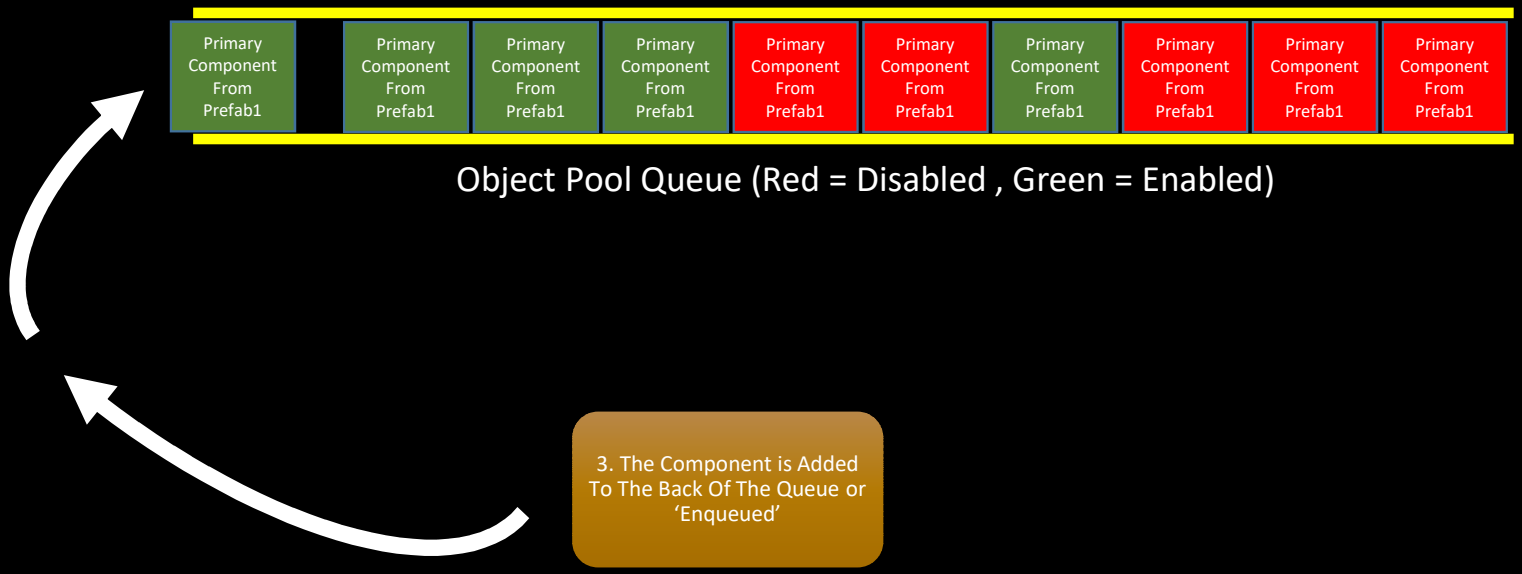
Object Pool Manager Approach

PoolManager – Retrieving Components From The Pool



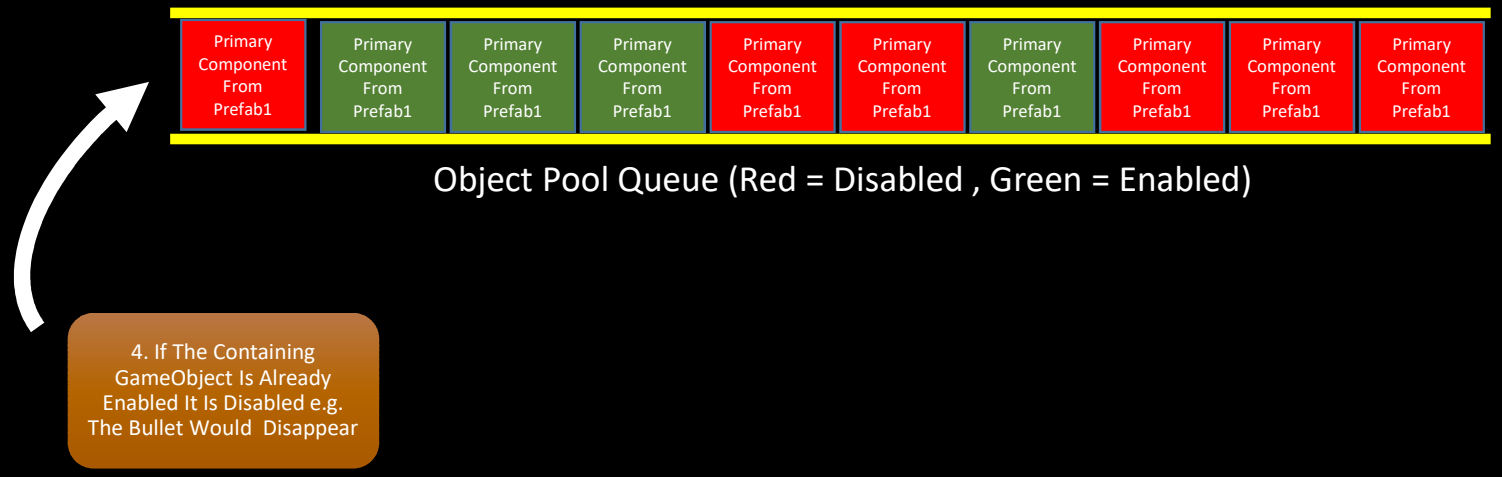
Object Pool Manager Approach

PoolManager – Retrieving Components From The Pool



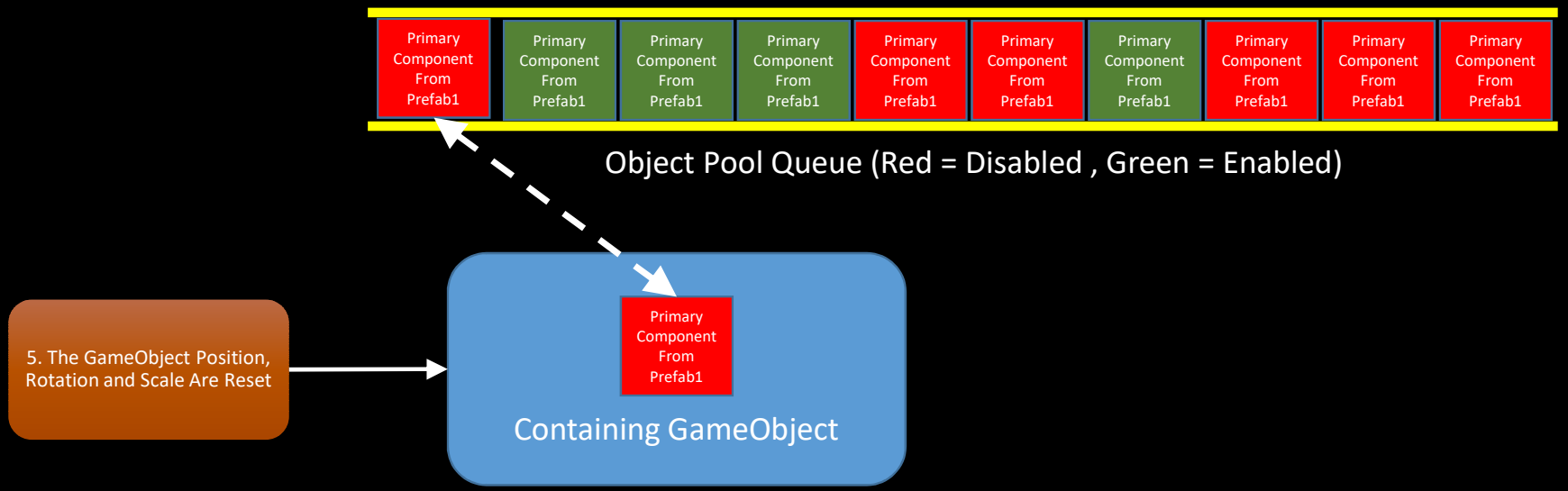
Object Pool Manager Approach

PoolManager – Retrieving Components From The Pool



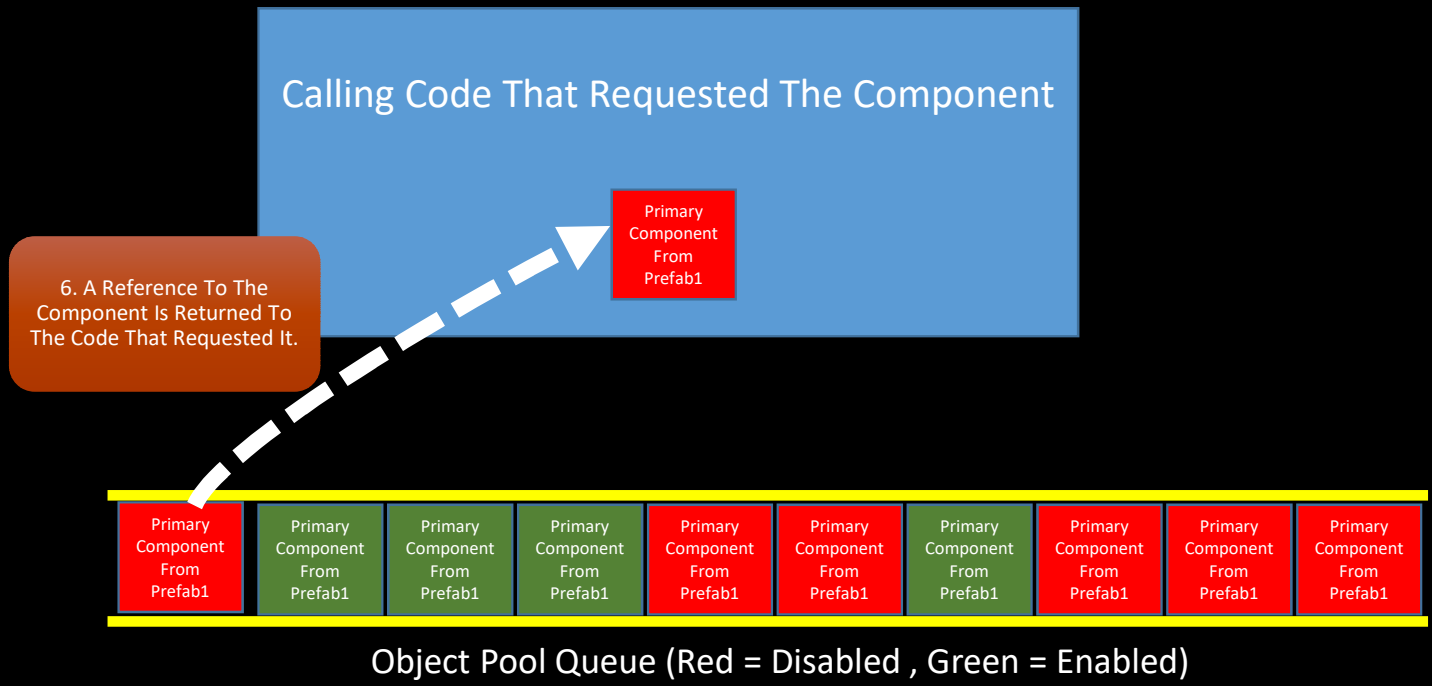
Object Pool Manager Approach

PoolManager – Retrieving Components From The Pool



Object Pool Manager Approach

PoolManager – Retrieving Components From The Pool



Object Pool Queue (Red = Disabled , Green = Enabled)

Object Pool Manager Approach

PoolManager – Retrieving Components From The Pool