Multiplayer Game Programming

Chapter 9

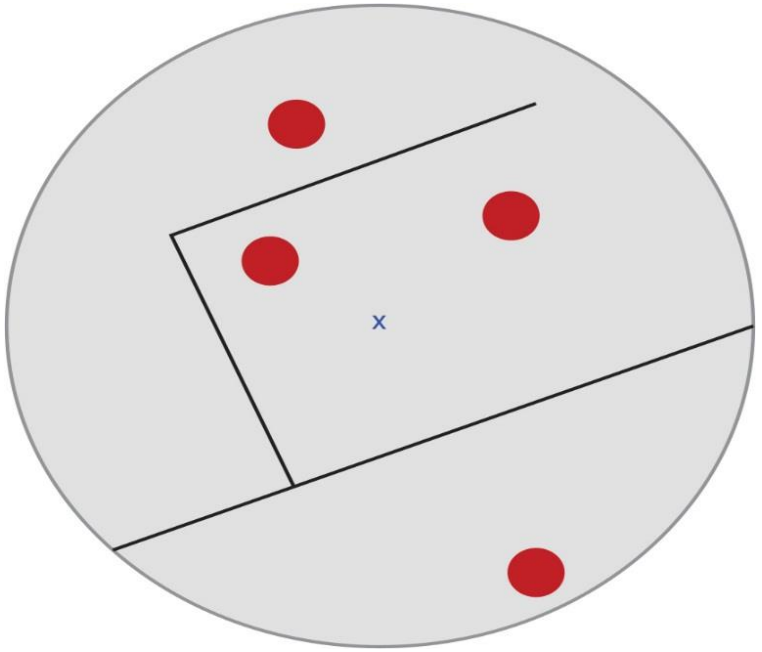
Server Architecture / Scalability

Chapter 9 / 11
Objectives

- **Object scope and relevancy**
  - How to determine which objects are important to a particular client
- **Server partitioning and instancing**
  - What are some methods to reduce the load on any one particular server?
- **Prioritization and frequency**
  - How to prioritize how important and how frequently certain game objects should be replicated
- **Unity Game Networking**
  - How does unity communicate data?
  - How to send information from a Client to a Server and Server to Client?

Object Scope/Relevancy

- An object is *in scope* or *relevant* if a client should be informed about updates to the object.
- A simple approach is to just use distance; closer objects are relevant.
- This can cause issues:
  - A sniper rifle may have a much larger relevancy range than a pistol.
  - Some areas of the level might have a high concentration of players.
  - Equal priority is assigned to objects in front of and behind the player.
  - Objects behind walls are relevant.

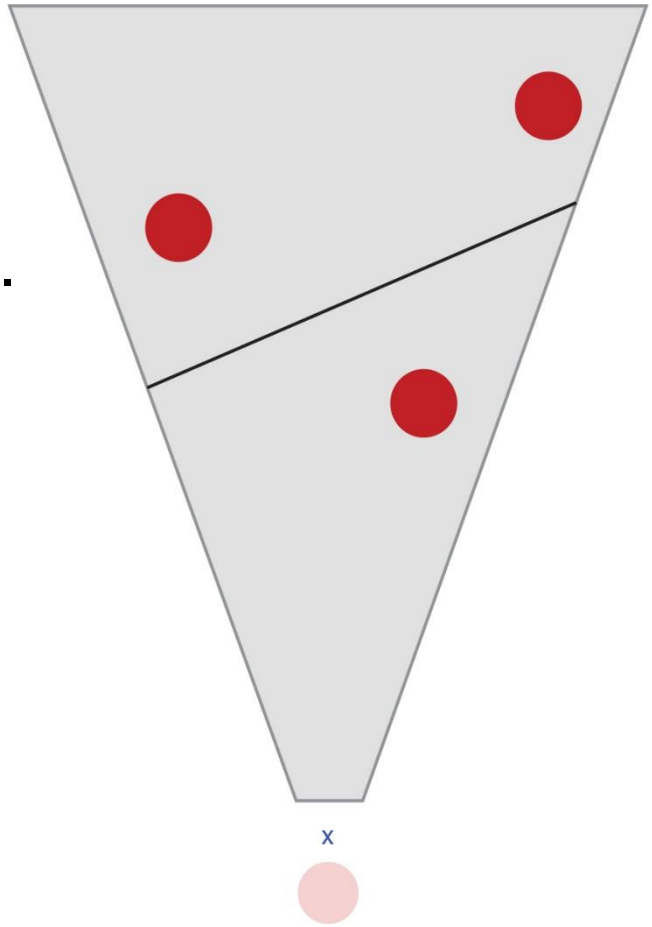


Static Zones

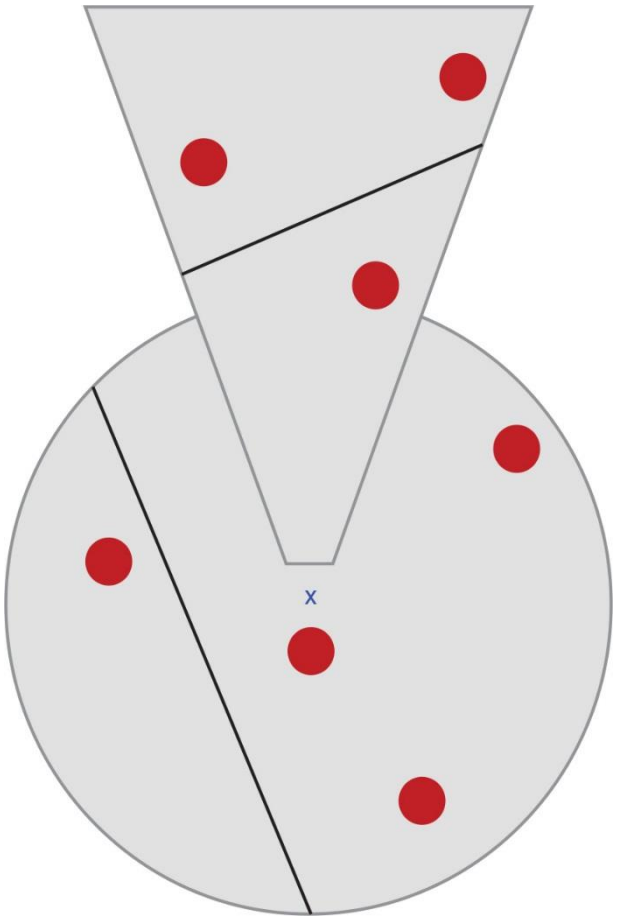
- Idea: Break the world up into static zones.
- Only objects in the same static zone as the player are relevant.
- This is commonly used in shared-world games such a MMORPGs.
- Transitions can be handled by a loading screen, or a streaming approach can be used.
- Can fail in the event that too many players are in one static zone.

Using the View Frustum

- The **view frustum** is used by the perspective projection to determine what to show onscreen.
- Can also use it for object relevancy.
- Problems:
  - Objects behind the player are completely ignored; if the player quickly turns 180 degrees, there may be latency for objects to scope in.
  - Objects behind walls still are relevant.

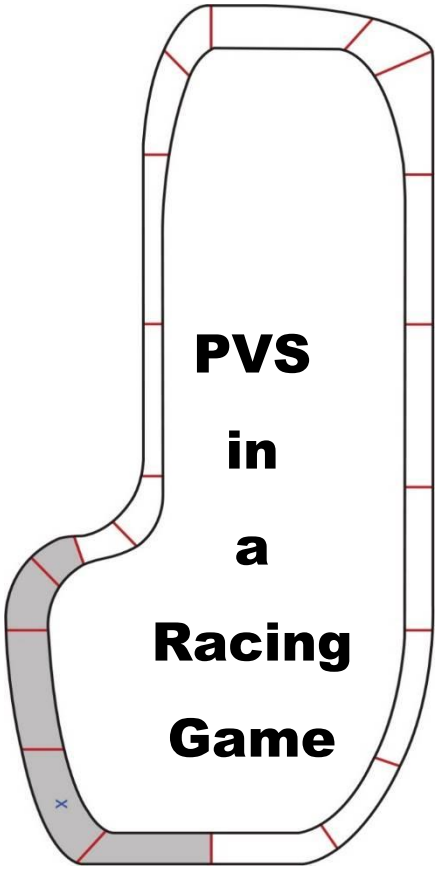


A Combination?



- Combine the radial approach with the view frustum approach.
- Still have wall issues, but….
- Objects in front of the player are given more priority.
- Objects around the player are still relevant.

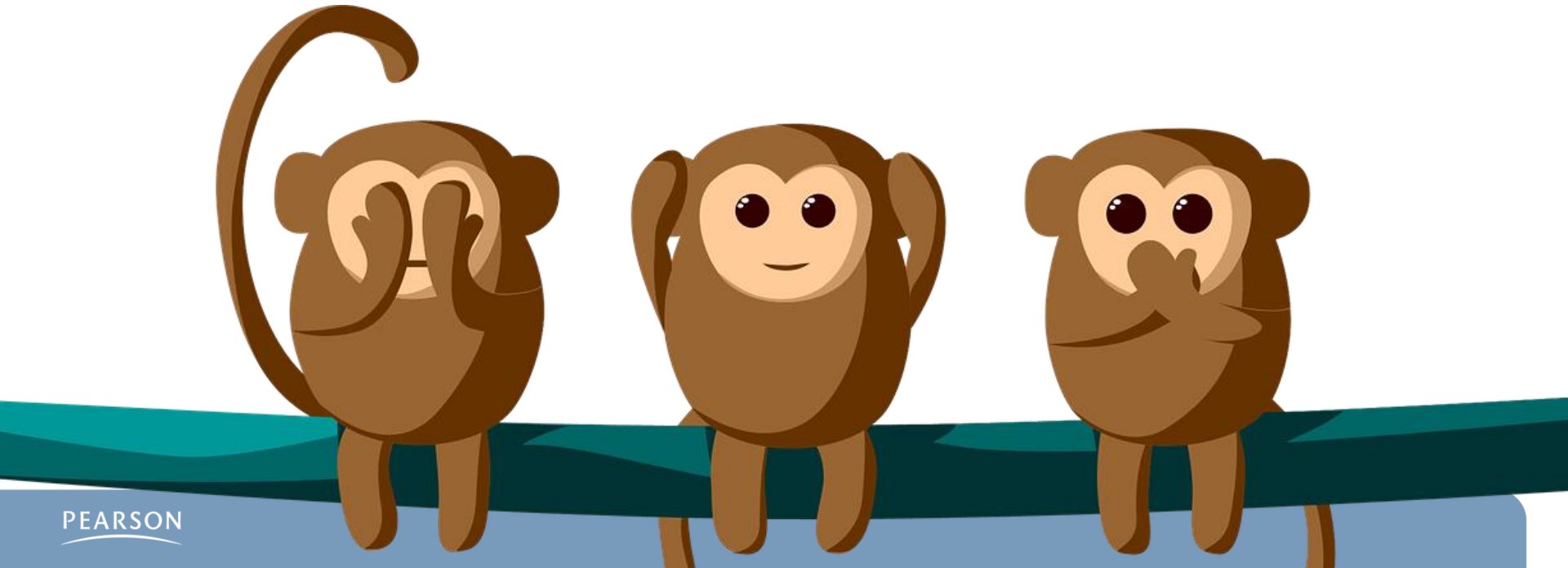
Potentially Visible Sets



- **Potentially visible sets** (Pass) divides the game world into several relatively small regions.
- From each region, determine what is the set of other regions that are visible.
- When in region A, only objects in the regions that are visible from A are relevant.
- Great for games with discrete segments, such as a racing game.

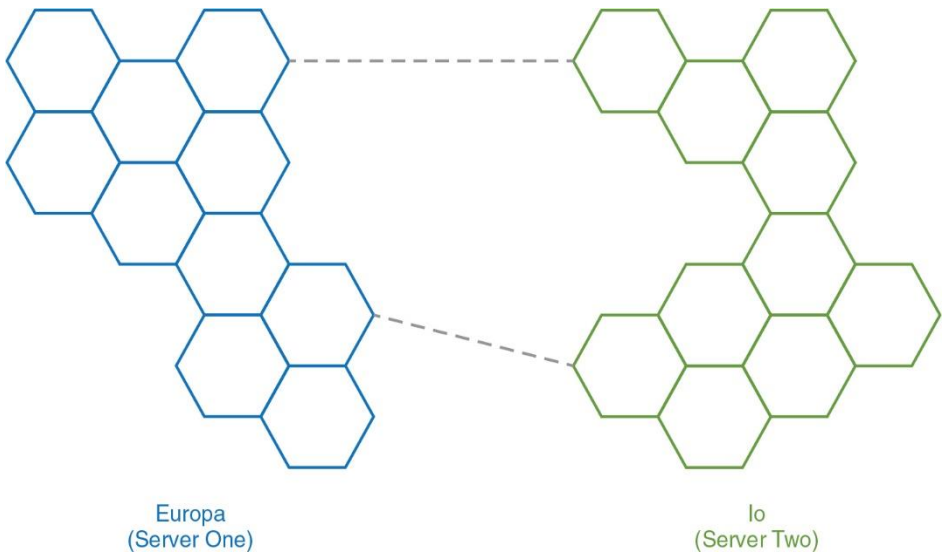
Relevancy When Not Visible

- Sometimes, invisible objects may still be relevant.
- If a player throws a grenade on the other side of the wall, you should still hear the grenade.
- One solution is to have special case replication code for objects that fit this profile.

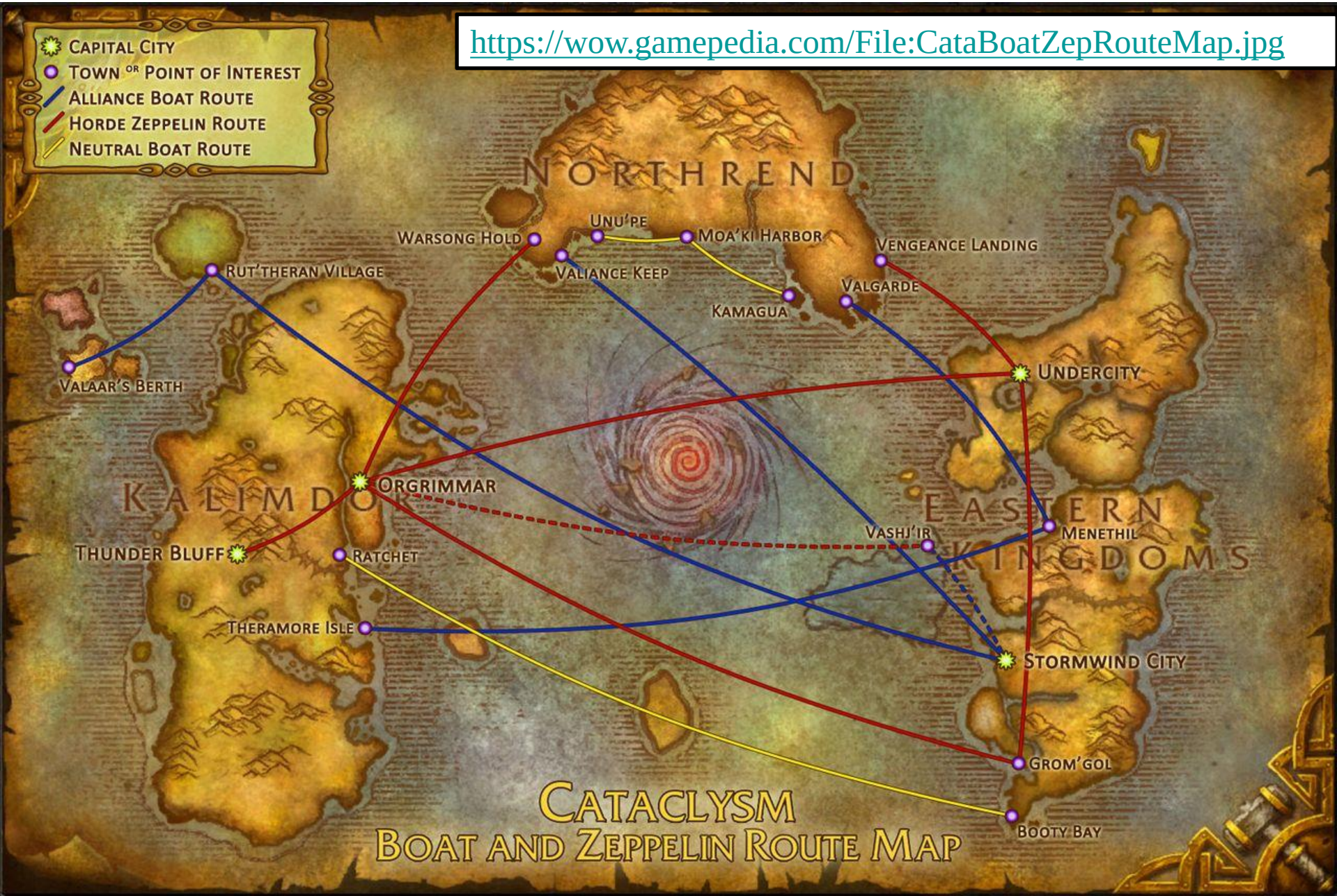


Server Partitioning

- **Server partitioning** or **sharding** involves running multiple server processes simultaneously.
- Most action games use this, because there is a cap for the number of active players.
- MMORPG games also use this to partition their shared worlds; for example, there may be two continents on separate servers.



WOW Continents



Instancing

- One shared game supporting several simultaneous instances.

- Used to allow several players to independently experience the content, such as a dungeon.
  - Typically accessed through a portal.
  - Loading screens applied.

- Also used as a solution for overcrowding static zones. *Star Wars: The Old Republic* spins off an instance if too many players are in a zone.

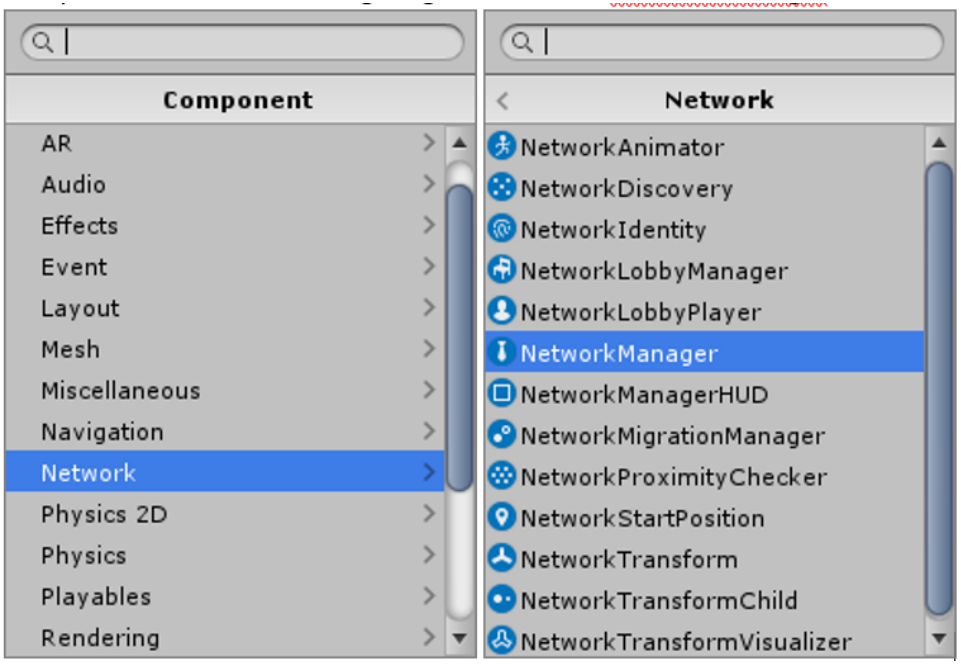
Prioritization and Frequency

- Some objects could be assigned a lower priority.
- If there isn't enough bandwidth, lower-priority objects may have their replication updates deferred.
- Should still have a method to track how long since the last update, to ensure that low-priority objects are at least occasionally updated.

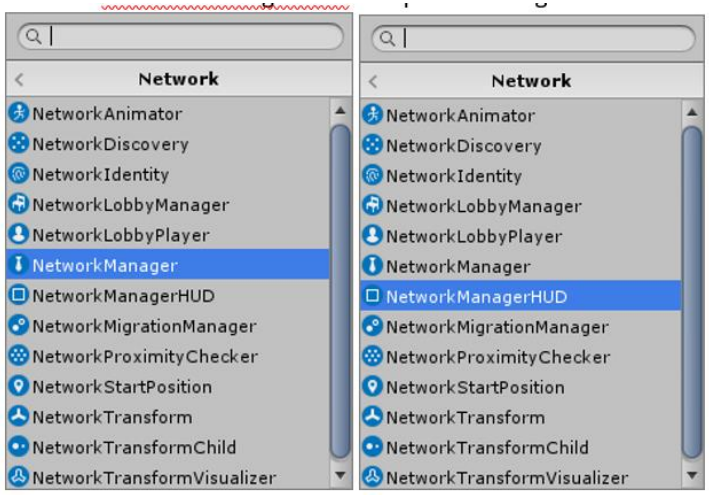
- Very popular for mobile and independent developers
- UNET: As of Unity 5.1, a completely new networking system
  - Marked now as deprecated, Unity is presently working on a complete overhaul after purchasing
- An external options is Photon which has plugins available for unity as well.
  - For this class we will focus on the built in capabilities of Unity

Topology

- The Unity NetworkManager can run in three modes:
  - As a client
  - As a dedicated server
  - As a combined “host” (both server and client)
- In essence, supports both a dedicated server or listen server.
- A NetworkManager is a component available to be added to a GameObject.
  - You should only have 1 single NetworkManager.
  - You can configure the NetworkManager to a specific IP and port



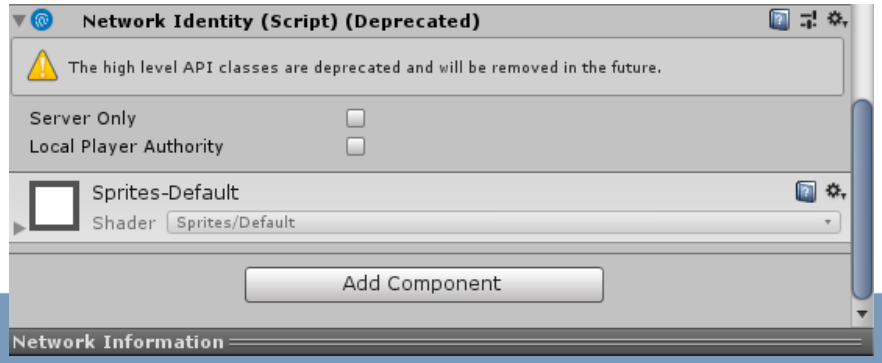
unity Network Manager HUD



- In Addition to the Network Manager a NetworkManagerHUD component exists to interface with the NetworkManager and allow for a quick UI for testing purposes.
- This provides acces for users to choose to Host a game or join an existing game as a Client.
- Unity also has it's own built in Match Maker which can be enable through this menu.

Network Manager

- The **NetworkManager** script requires a PlayerPrefab assigned to it to function.
  - Every time a player joins the game, a new PlayerPrefab is spawned.
  - Any Prefab can be placed into the PlayerPrefab slot so long as it has a **NetworkIdentity** component attached to it.
- **NetworkIdentity** keeps track of an ID value for each player and object added to the game.
  - This is incremented automatically on the server and the clients are informed of their ID through this component.



NetworkBehaviour

- The **NetworkBehaviour** object can replace the Monobehaviour inheritance of Gameobject used for networking.
  - NetworkBehaviour is based off of MonoBehaviour, so you will retain all properties and functionality of a MonoBehaviour in addition to new networking properties / functionality.
  - Requires:

```
using UnityEngine.Networking;
```

```
//The abstract keyword enables you to create classes and cl
public abstract class MovingObject : NetworkBehaviour
{
```

  - Provides access to things variables which assist in determining if the function is being running on a client, a server, or someone with authority.
    - isServer can be used to determine if this is a server execution.
      isLocalPlayer can tell if the function being executed is the current local player.

```
if (isServer)
```

```
if (!isLocalPlayer)
```

Commands

- Actions sent from **client to server** are called **commands**.
  - **Commands** are marked with the **`[Command]`** attribute, and the function name **must begin with `Cmd`**.

```
[Command]
void CmdNetworkMove(int x, int y)
{
    AttemptMove<Wall>(x, y);
}
```

Requires:

```
using UnityEngine.Networking;
```

unity Remote Procedure Calls

- Actions sent from **server to client** are called client **RPCs**.
  - **Client RPCs** are marked with the **`[ClientRpc]`** attribute and the function name **must begin with `Rpc`**.

```
[ClientRpc]
void RpcUpdateMovablePosition(Vector3 newPos)
{
    rb2D.MovePosition(newPos);
}
```

Requires:

```
using UnityEngine.Networking;
```

Summary

- Determining which objects to replicate and when needs to be determined on a game by game basis.
- Level design and object importance are large considerations for information sharing.
  - Some tricks we can implement are:
    - Frustum culling
    - Radial culling
    - Static Zones / PVS
    - Instances
    - Sharding
    - Prioritization
- Unity has built in functionality to handle communication between clients for a multiplayer game.
  - NetworkManager / NetworkManagerHUD
  - NetworkBehaviour
  - RPCs
  - Commands