UNIVERSITÀ DEGLI STUDI
DI MILANO

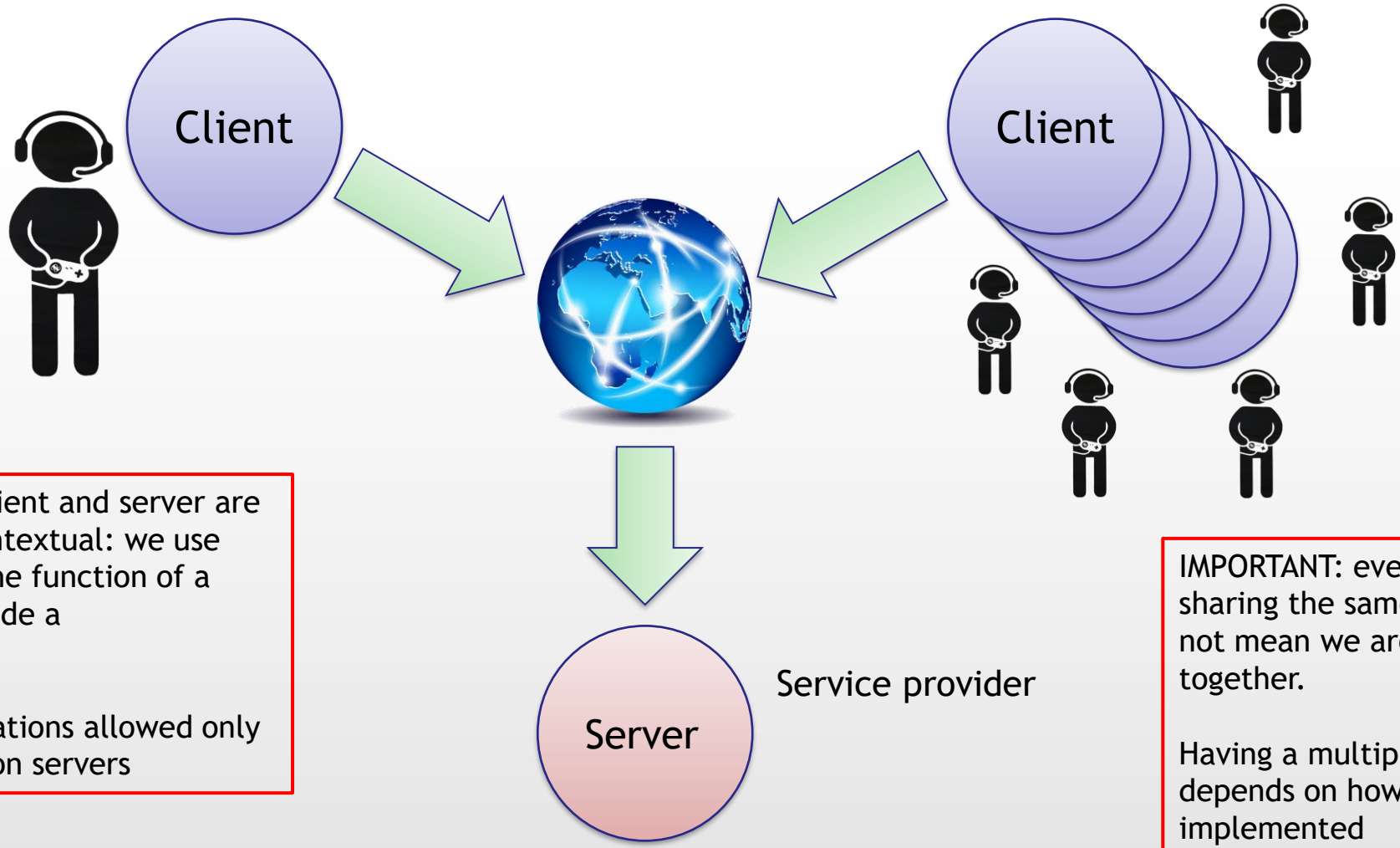
Unity 101 - Networking

Protocols? Where we're going, we don't need protocols

Dario Maggiorini (dario@di.unimi.it)

Playlab fOr inNovation in Games

Game Networking in General



The concepts of client and server are indicative and contextual: we use them to identify the function of a network entity inside a communication.

There are no operations allowed only on clients or only on servers

IMPORTANT: even if we are sharing the same server, it does not mean we are playing together.

Having a multiplayer game depends on how the server is implemented

Unity Networking in Particular

- Is component-based
  - You add components to your scene, and these components will sync their parameters through the network
- Reduce the amount of code involved
  - You can write your own (network) layer … only if strictly required
    - We do not need to design a protocol!
- Internet services are included
  - No need to create and parse messages to/from complex services
    A behavior will be already there for you
    - Matchmaking
    - Relay Servers
    - Advertisement

We Used to Use UNET … But …

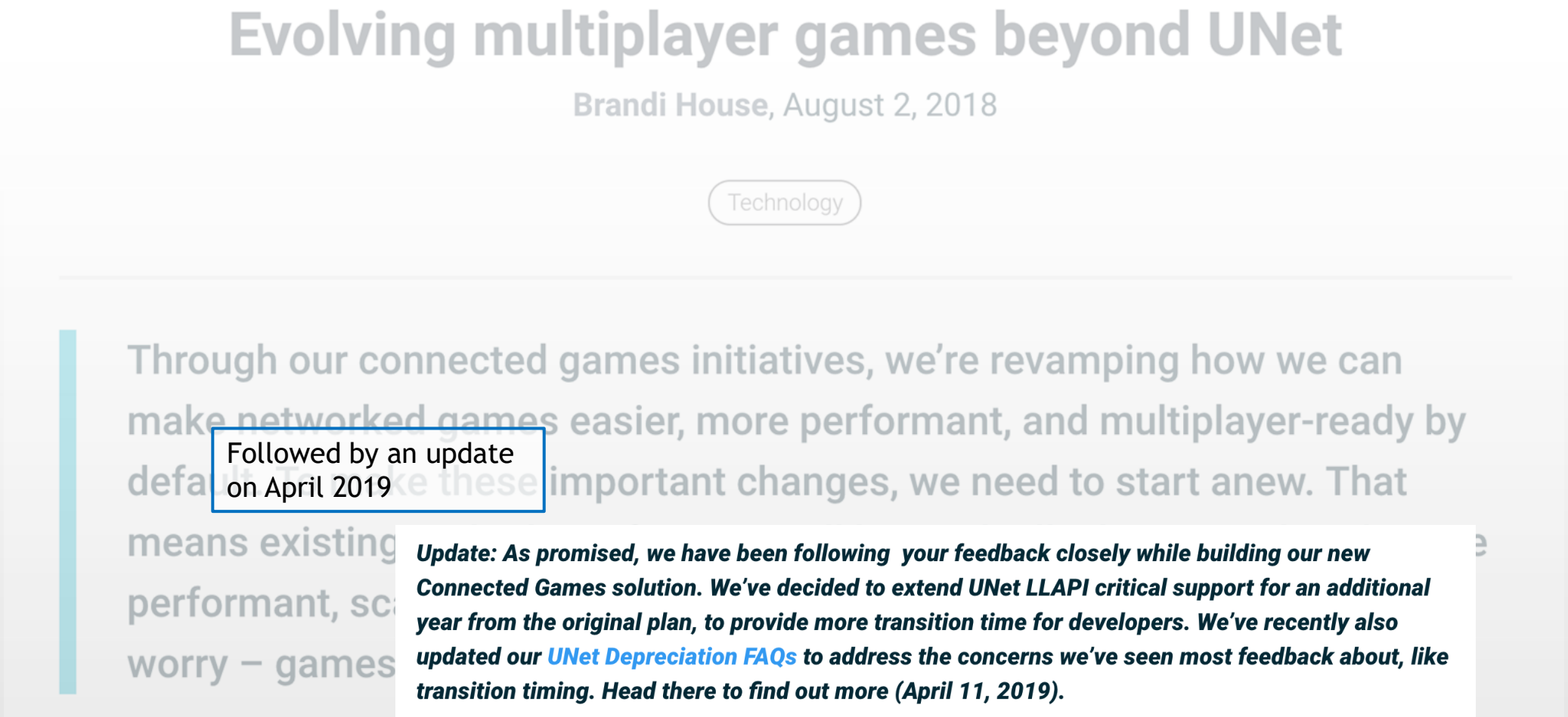
Evolving multiplayer games beyond UNet

Brandi House, August 2, 2018

Technology

> Through our connected games initiatives, we're revamping how we can make networked games easier, more performant, and multiplayer-ready by default. To make these important changes, we need to start anew. That means existing multiplayer features will be gradually deprecated, with more performant, scalable, and secure technologies taking their place. But don't worry – games with impacted features will have plenty of time to react.

We Used to Use UNET … But …



Evolving multiplayer games beyond UNet

Brandi House, August 2, 2018

Technology

Through our connected games initiatives, we're revamping how we can make networked games easier, more performant, and multiplayer-ready by default. To make these important changes, we need to start anew. That means existing … performant, sc… worry – games…

Followed by an update on April 2019

Update: As promised, we have been following your feedback closely while building our new Connected Games solution. We've decided to extend UNet LLAPI critical support for an additional year from the original plan, to provide more transition time for developers. We've recently also updated our UNet Depreciation FAQs to address the concerns we've seen most feedback about, like transition timing. Head there to find out more (April 11, 2019).

Meanwhile, in the Manual for Unity 2019.3 ...

Unity User Manual (2019.3) / Multiplayer and Networking

Multiplayer and Networking

Note: UNet is deprecated, and will be removed from Unity in the future. A new system is under development. For more information and next steps see this blog post and the FAQ.

Still under development????

But today …

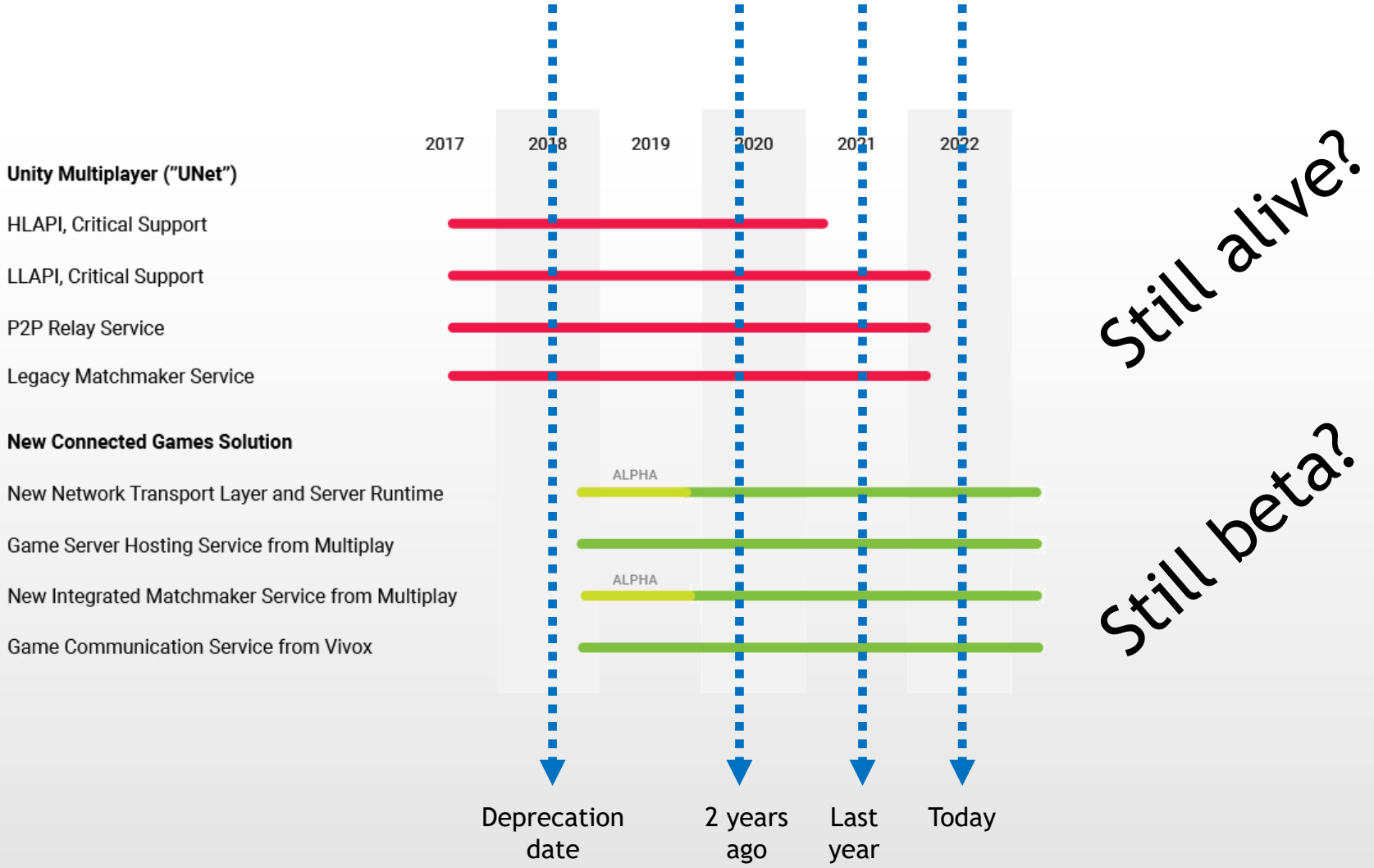
They changed the release tag, name, and a bit of text!

Unity User Manual 2020.3 (LTS) / Multiplayer and Networking

Multiplayer and Networking

Important: UNet is a deprecated solution, and a new Multiplayer and **Networking** Solution (MLAPI) is under development. For more information and next steps see the information on the Unity MLAPI website.

Then, You Head to the FAQs



And then Another Blog Post (Yeah!)

https://blogs.unity3d.com/2019/06/13/navigating-unitys-multiplayer-netcode-transition/

> This blog was last updated, 27th April 2020.

Launch Window

Assuming that you've ended up with a DGS topology, we need to know when you hope for your game to launch, because your options will change based on what will be available. Here's a high-level outline:

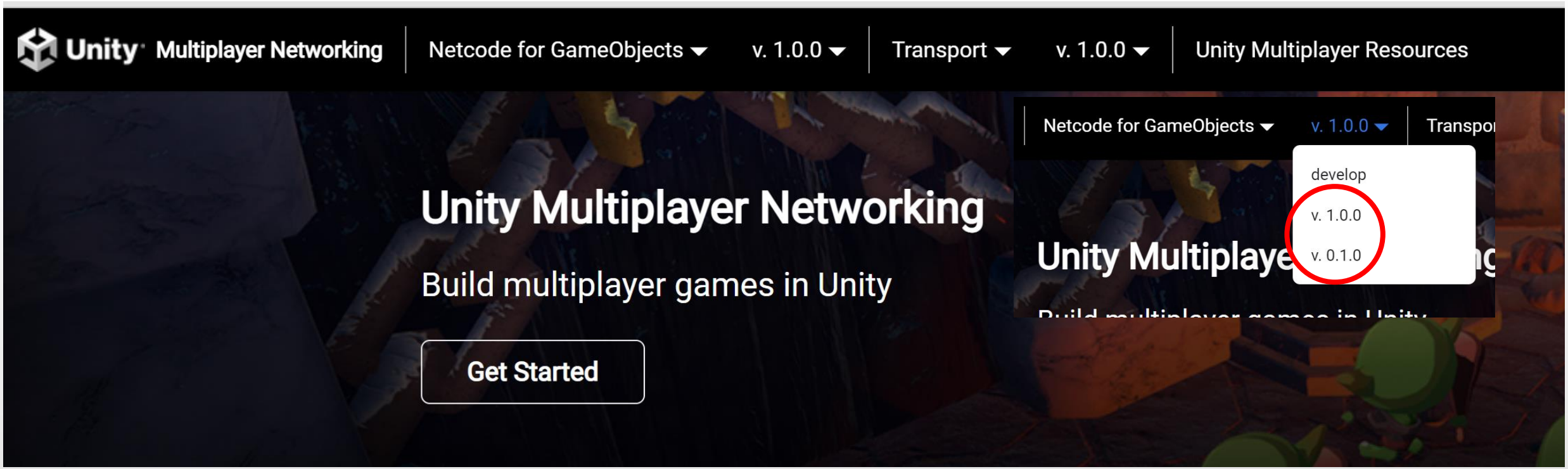
- **Right now: the Unity Transport Package and Reliability** are available in preview, and this can be paired with the new DOTS Netcode preview. . Given the Preview state of these libraries, you could use LLAPI or external libraries instead if you need a more stable API surface immediately.
- **Q2 2021: Production-quality DOTS-Netcode** – we're targeting a production-quality release in early 2021 for DOTS-Netcode and UTP to be sufficiently stable and full featured. By this time, developers should have a fairly stable API surface, tech stack, and better documentation.

So, if you plan to launch before Q2 2021, you should plan to use off-the-shelf (OTS) solutions, such as Forge, Photon or DarkRift 2, that may also work for DGS games (*note: we have not verified the quality of these OTS options*).

Production ready for Q2 2021?
So, it is not a beta, it is definitely a pre-release

But then, in 2022 … surprisingly …

- Release time!



Just Install the Pack …

- What the F… ????

- OK Guys. Now you know another definition for “marketing”

Summary

- Unity networking is in a transition phase

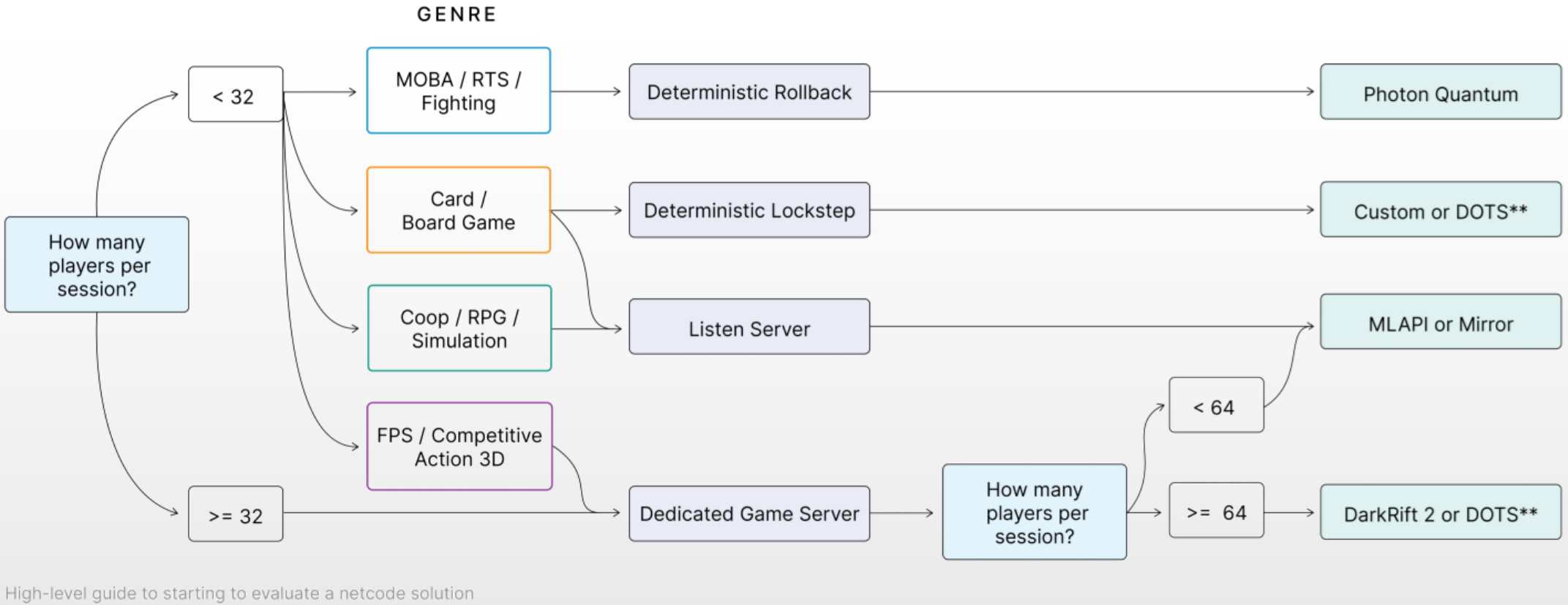
- Legacy networking will be dropped some day
  - Actually, it is still marked to be removed “sometime in the future” but HLAPI package 1.1.1 is available in the package manager window for 2020.3 LTS
- New networking is just out of beta (pre release!)
  - We now have some supported documentation
  - Expect some misbehaviors

- TIP FOR SURVIVAL
  - Use version 2020.3 LTS ... and stick with it!
  - Need P2P gaming? Stick to UNet!
  - … or look for another framework (there are many)
    - Photon (server based, proprietary)
    - Mirror (open source, reproposing UNet)

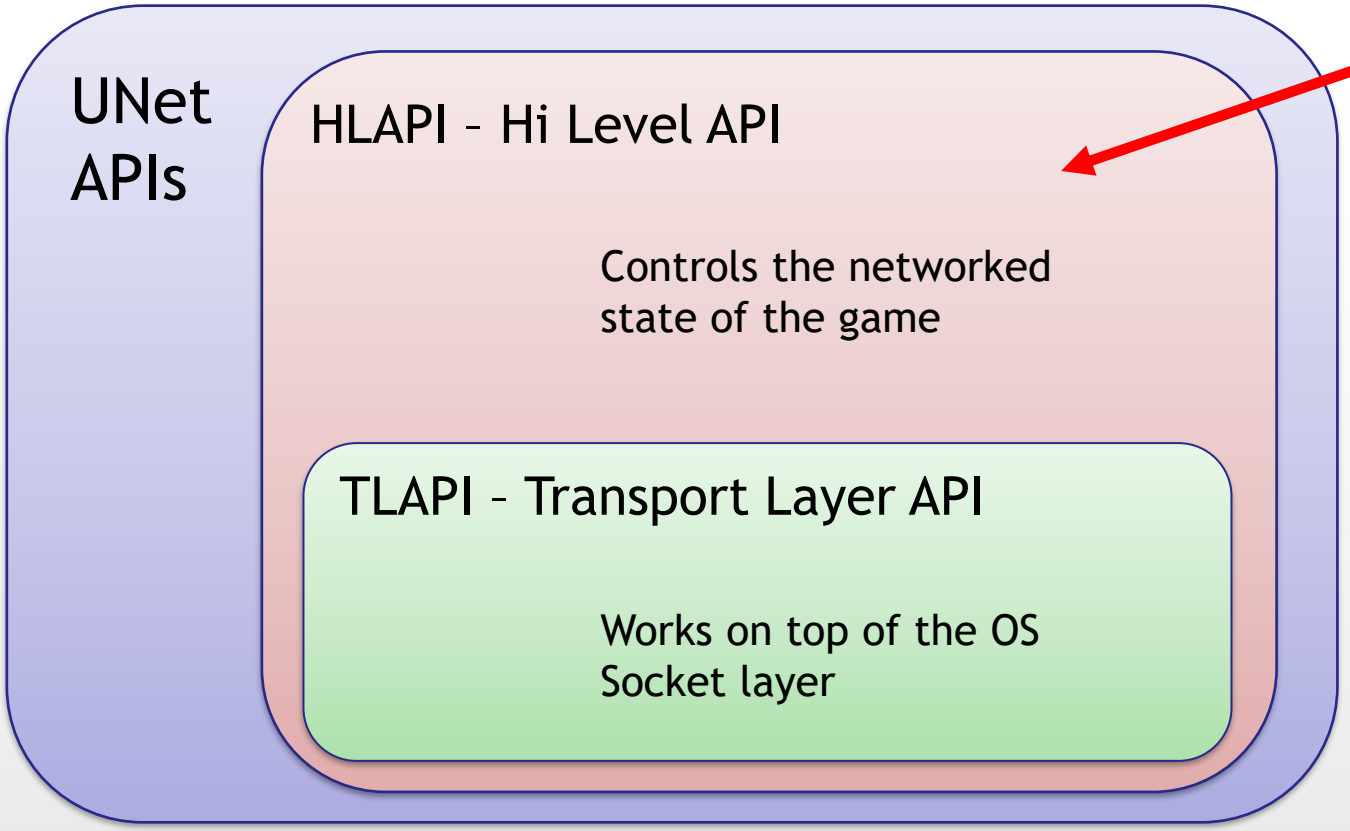
... Because you only have three months

Using Other Frameworks

https://blogs.unity3d.com/2018/08/02/evolving-multiplayer-games-beyond-unet/



UNet Architecture Overview



You can
write
your own

photon

GAMESPARKS

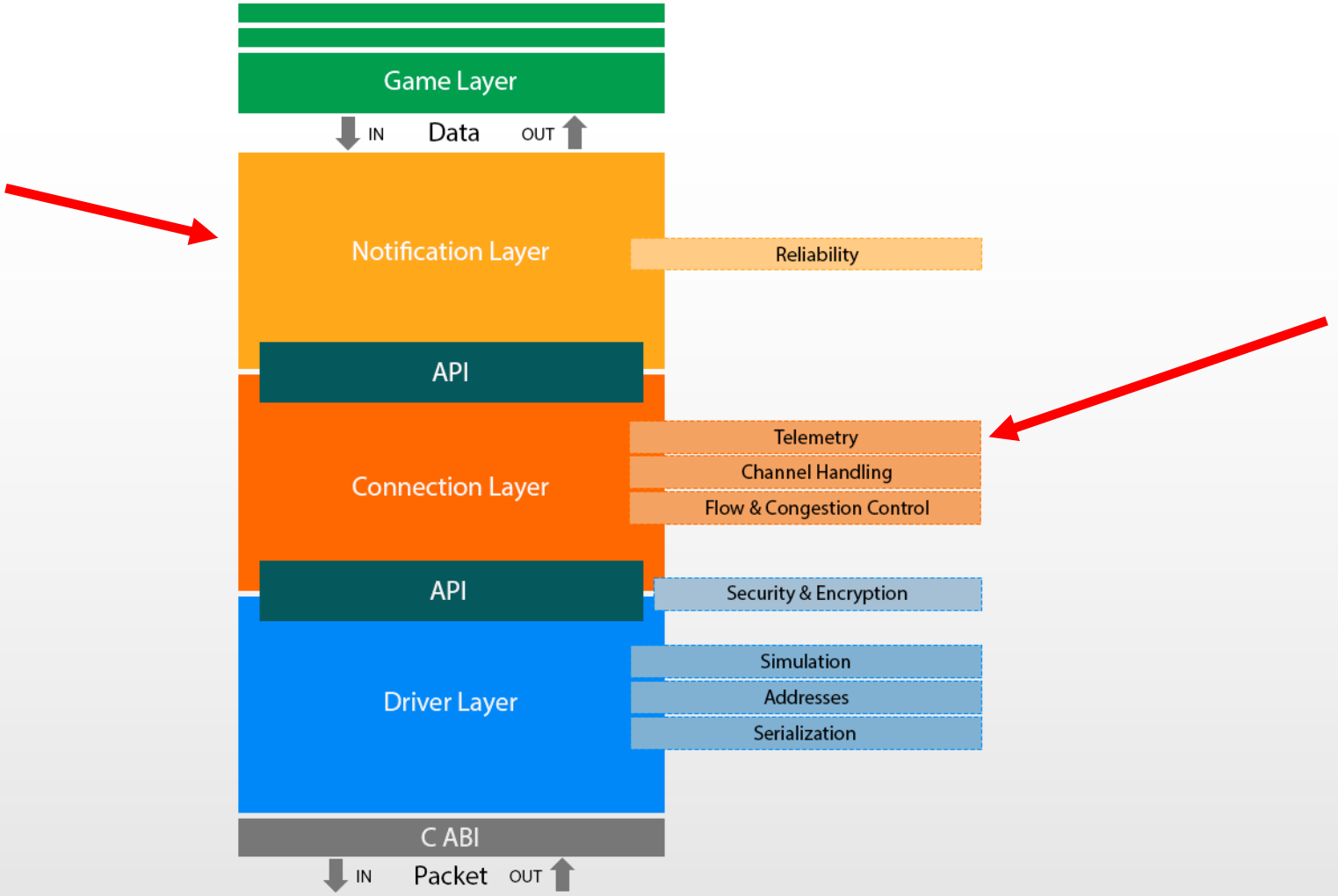
HLAPI

- Controls the networked state of the game
  - Network Manager Component
- Operates client hosted games
  - A client will connect to a server running on the PC of another client
    - P2P architecture
  - P2P architecture **WILL NOT BE UNSUPPORTED BY NetCode**
- Sends and receives stuff from client to server (and back)
  - Commands
  - Messages
  - Serialized data
  - Networked events

Looks like a nice way to force us using their (paid) service and forbid people to play in ad-hoc mode.

Drawn your own conclusions.

NetCore Transport Architecture Overview



Unity Transport APIs

- Optimized UDP protocol
  - A standard UDP protocol with optimized payload (we will send as few packet as possible)
- Multi-channel design to avoid head-of-line blocking
- Support for Quality of Service
  - Application-level QoS does not interact with network infrastructure
  - It is a local packets schedule based on priority over the multiple channels
- Flexible and dynamic network topology
  - Peer-to-peer (not in the way you think!)
  - client-server

We are not going to dig into these

Why UNet is Still a Thing Then?

1. I have trouble trusting NetCode right now

2. It is still working … and not “experimental”

3. It is the one still allowing you to have a direct client-server setup without having a gaming service provider in the middle
   - Mirror anyone?

4. WebGL is supported

Why Talking abour UNet Today?

- NetCode functionalities are THE SAME as UNet with some pieces missing
  - Same logic and workflow
  - Same names for the components (you cannot have both installed)
  - Conversion is a matter of change a few things
    - some syntax
    - utility classes
    - decorators
  - Lobby interface HUD is missing (!)

https://docs-multiplayer.unity3d.com/docs/migration/migratingtonetcode/index.html

Hold your Desperation

- General concepts are valid for both UNet and NetCode
  - That is the part about “how to implement a networked game”

- Specific concepts are mostly similar
  - … if not the same
  - Differences between UNet and NetCode will be outlined

- WARNING
  This topic is generally hard to digest even without Unity Technologies getting in the middle
  Just take your time

Rules of the Game

1. Each GameObject has a representation on every connected clients
   - Exception: you may define a field of existence based on proximity for sake of performances, or to implement a field of vision/fog of war

2. Each GameObject belongs to a client which is authoritative for that object
   - By default, server has authority on everything which is not a player (or directly controlled by a player)

3. All clients can modify directly only GameObjects for which they have authority
   - This prevents cheating by implementing strict access control

4. Modifications made on each client are pushed to the server and then propagated back to all other clients
   - And this is what we call a network update

How to Apply the Rules

- All clients are using the same assets
  - Maps, objects, shaders, animations ... everything!

- We define as ”networked” only those objects that contribute to the shared experience
  - All other objects will be local and behave in the same way just because all assets are the same

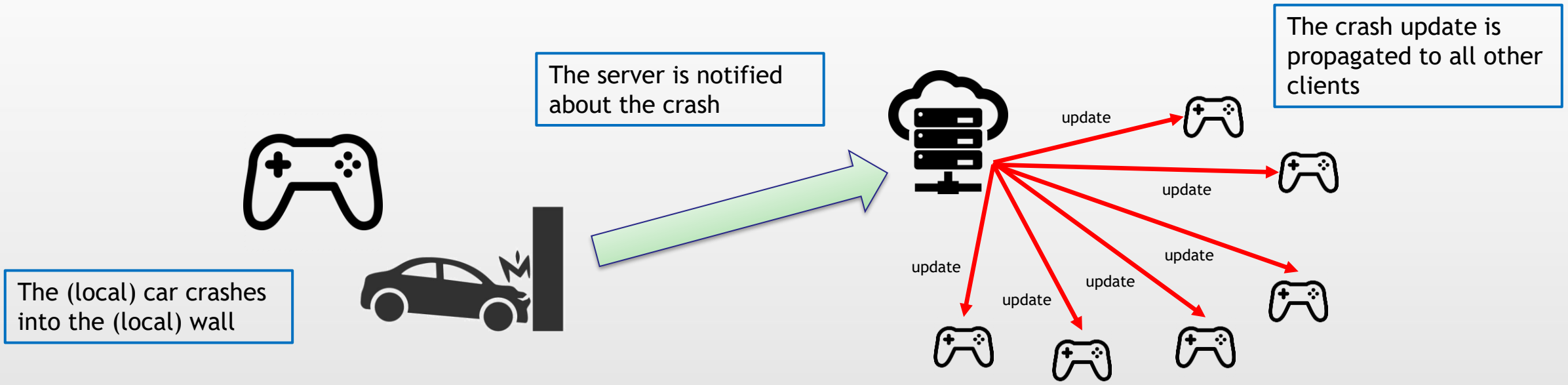
The (local) player is driving the (local) car

In a race, the local car is the only object needing to be networked. Other player need to interact only with my car

How to Apply the Rules

- All clients are using the same assets
  - Maps, objects, shaders, animations ... everything!

- We define as ”networked” only those objects that contribute to the shared experience
  - All other objects will be local and behave in the same way just because all assets are the same



How to Apply the Rules

- All clients are using the same assets
  - Maps, objects, shaders, animations ... everything!

- We define as ”networked” only those objects that contribute to the shared experience
  - All other objects will be local and behave in the same way just because all assets are the same

When you see me crashing into a wall, I actually crashed into MY wall, but you think I crashed into YOUR wall just because they are in the same position!

All I tell you are my shape and position while crashing

Something Happened Over the Network

- When something (e.g., a collision) happens, the host with authority will take care to manage it
- Local effects will propagate thanks to UNet

My cannon fires

I tell the server I sunk the ship

Something Happened Over the Network

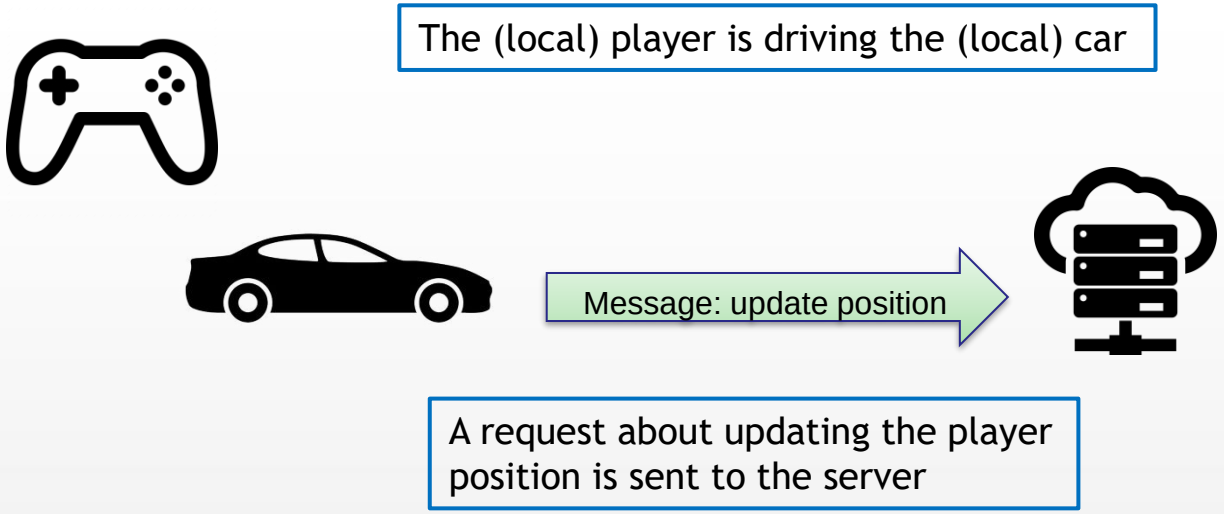
- When something (e.g., a collision) happens, the host with authority will take care to manage it
- Local effects will propagate thanks to Unet

My cannon fails

I tell the server I sunk the ship anyway

- Never, EVER, for any reason let a decision on the client!
  - Requires more resources
  - Increases lag
  - Cheating is possible

Back to the Car Crash Example, but Cheat-Proof

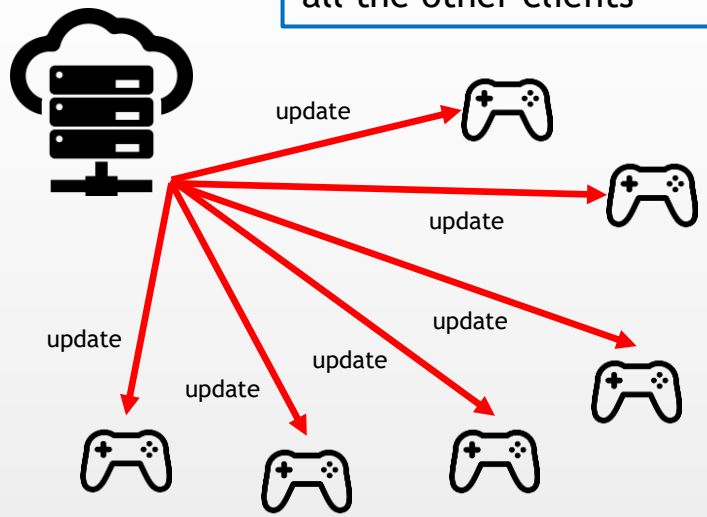


Back to the Car Crash Example, but Cheat-Proof

The server is in charge to compute the result of the update, and notify the result

And, of course, to update all the other clients

Reply: you crashed



To achieve this, a client must NOT be authoritative for the avatar (the car).
The server must be authoritative for the player avatar, while the player should be authoritative for another (invisible) gameobject used to tell the server what the player want to do.
This invisible object will then adjust to the avatar, moved in a “legal” position by the server

Back to the Car Crash Example, but Cheat-Proof

The server is in charge to compute the result of the update, and notify the result

And, of course, to update all the other clients

Reply: you crashed

update

update

update

update

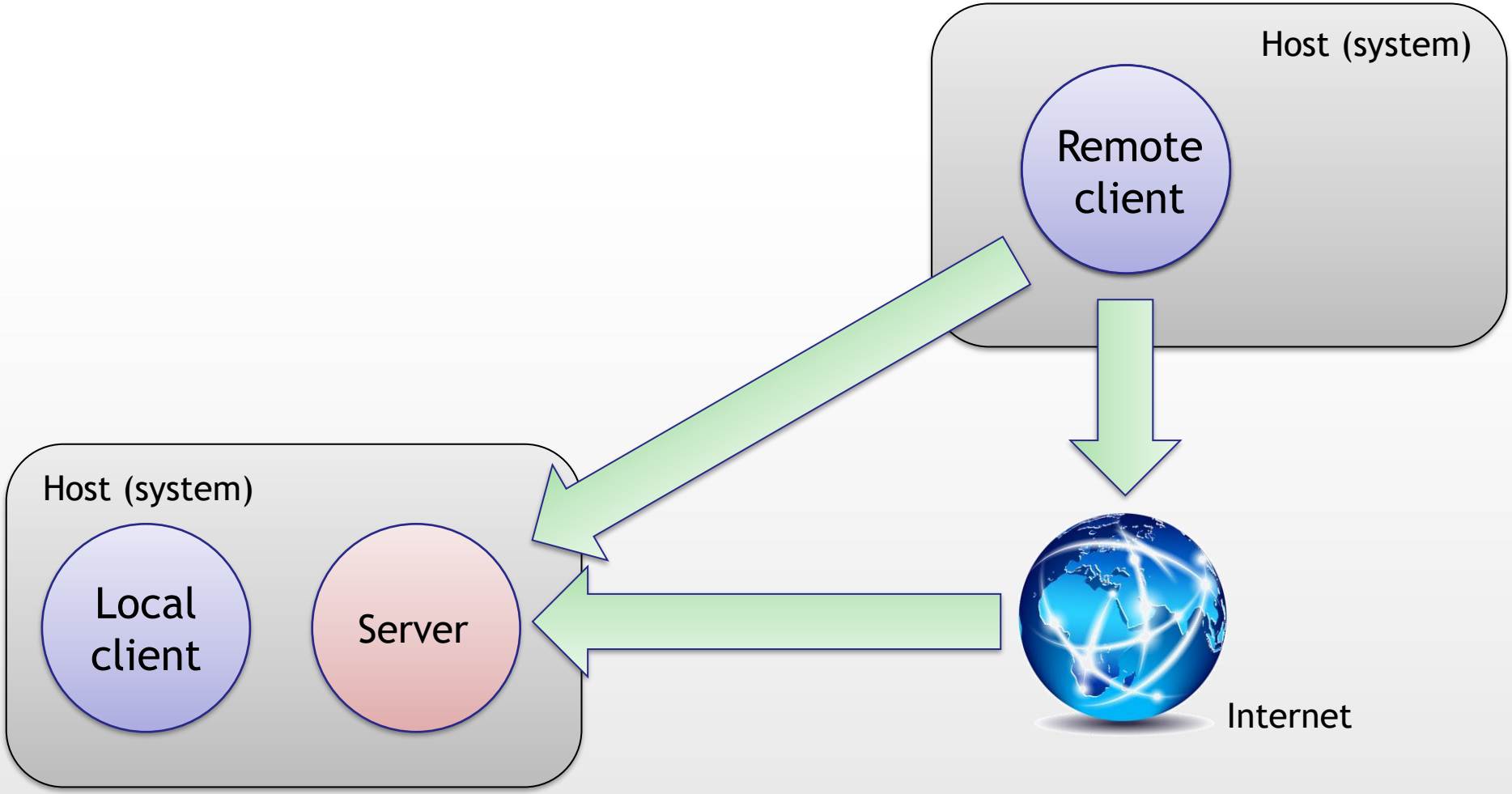
Let's get practical.
Albeit cheat-proof, this approach is much more complicated to implement and difficult to debug.

For your prototype you have three months, and no one really cares to cheat in your game.

Take the shortest route, and you will be fine

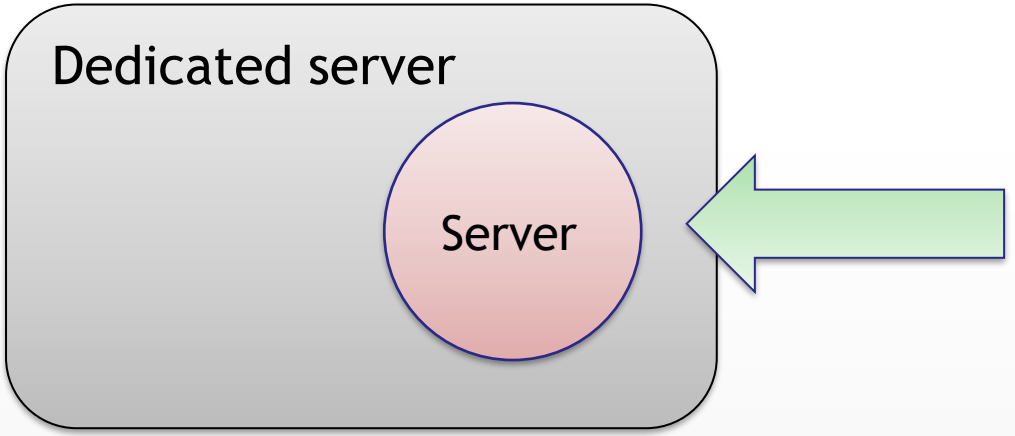
To achieve this, a client must NOT be authoritative for the avatar (the car).
The server must be authoritative for the player avatar, while the player should be authoritative for another (invisible) gameobject used to tell the server what the player want to do.
This invisible object will then adjust to the avatar, moved in a "legal" position by the server

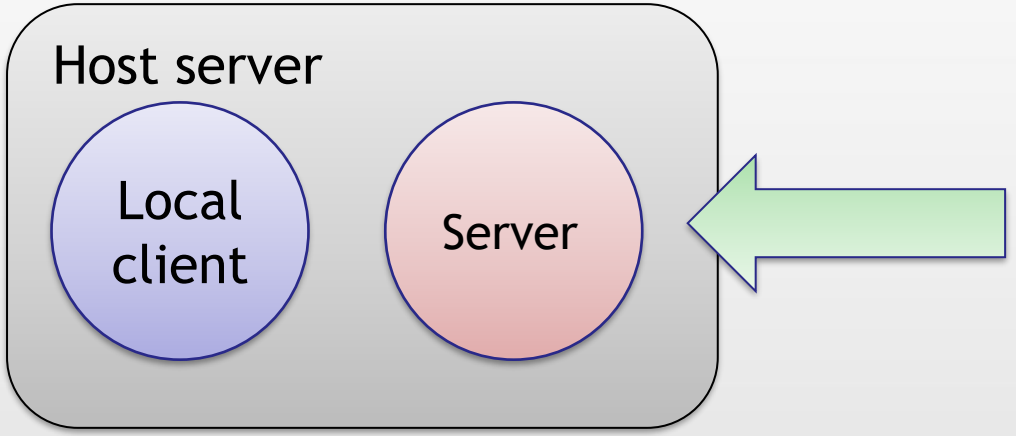
Naming Stuff (UNet Legacy)



Naming Stuff (UNet Legacy)

- It is not required to have a local client
- UNet defines two kinds of server-side host systems
  1. Dedicated server
     When no local client is allowed
  2. Host server
     When a local client can be present





What Do We Need to Sync Over the Network?

- Gameobjects position and rotation
  - They need to move around, right?
  - NetworkTransform component will take care of this

- Animation state for animated (rigged) gameobjects
  - Mind, “just the kind of animation” (running, walking, sleeping)
  - NetworkAnimator component will take care of this

- Generic values
  - Elapsed time, energy left, free resources
  - You will tell the core to take care of this
    - Use [SyncVar] decoration in your scripts or SyncList* classes
    - NetCode uses a NetworkVariable<*> generic in place of [SyncVar]
    - NecCode uses NetworkList<*> generic in place of SyncList* classes

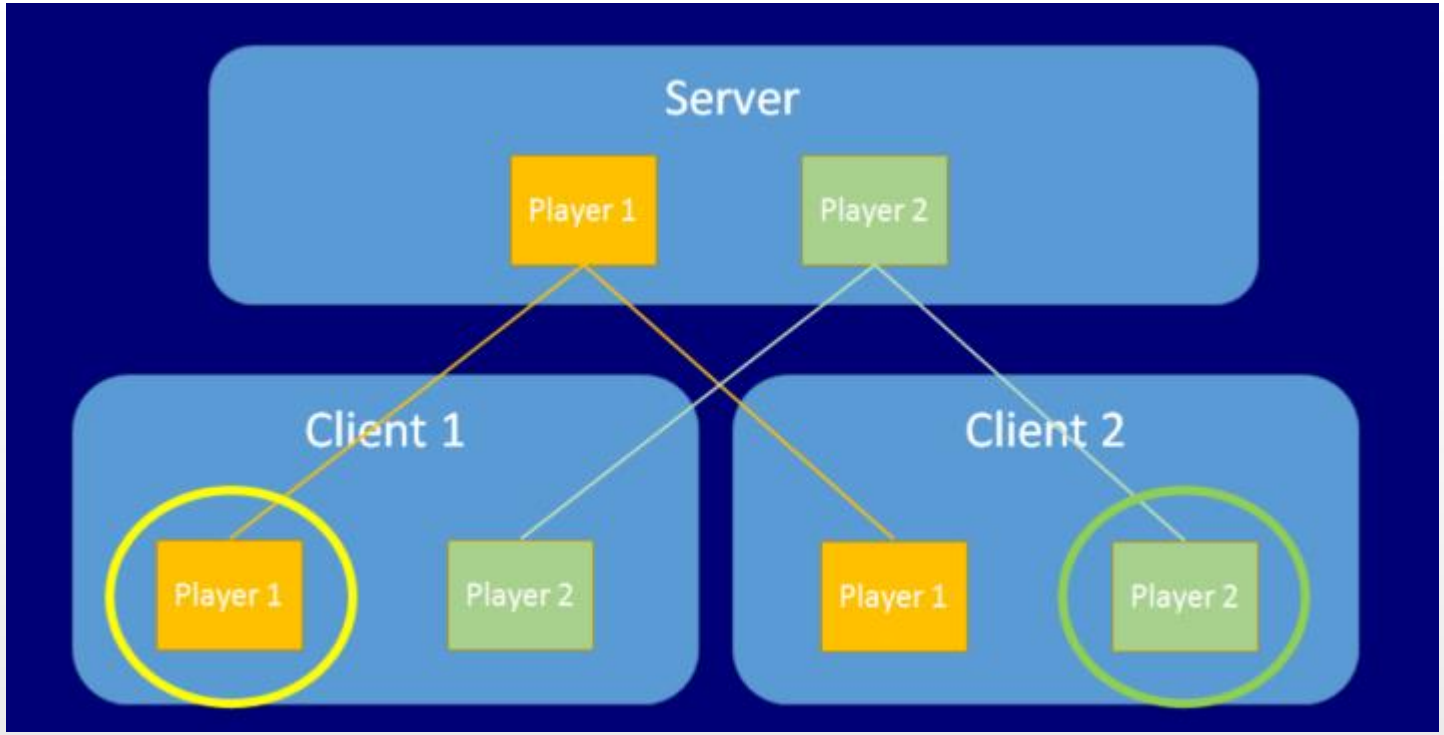
You do not need to know the exact position of every limb; it is enough to see someone walking. No one will know we are not in the exact same position.
Animations, remember, are shared assets!

Player and Non-Player GameObjects

- A player gameobject is an object representing the player
  - When created, it will become the “local player”
  - The NetworkManager will bound the connection to the local player
  - All messages coming from the server will be routed through the local player
    - If the gameobject representing the player is deallocated, the connection is dropped
  - Every client MUST have its own (possibly single) local player

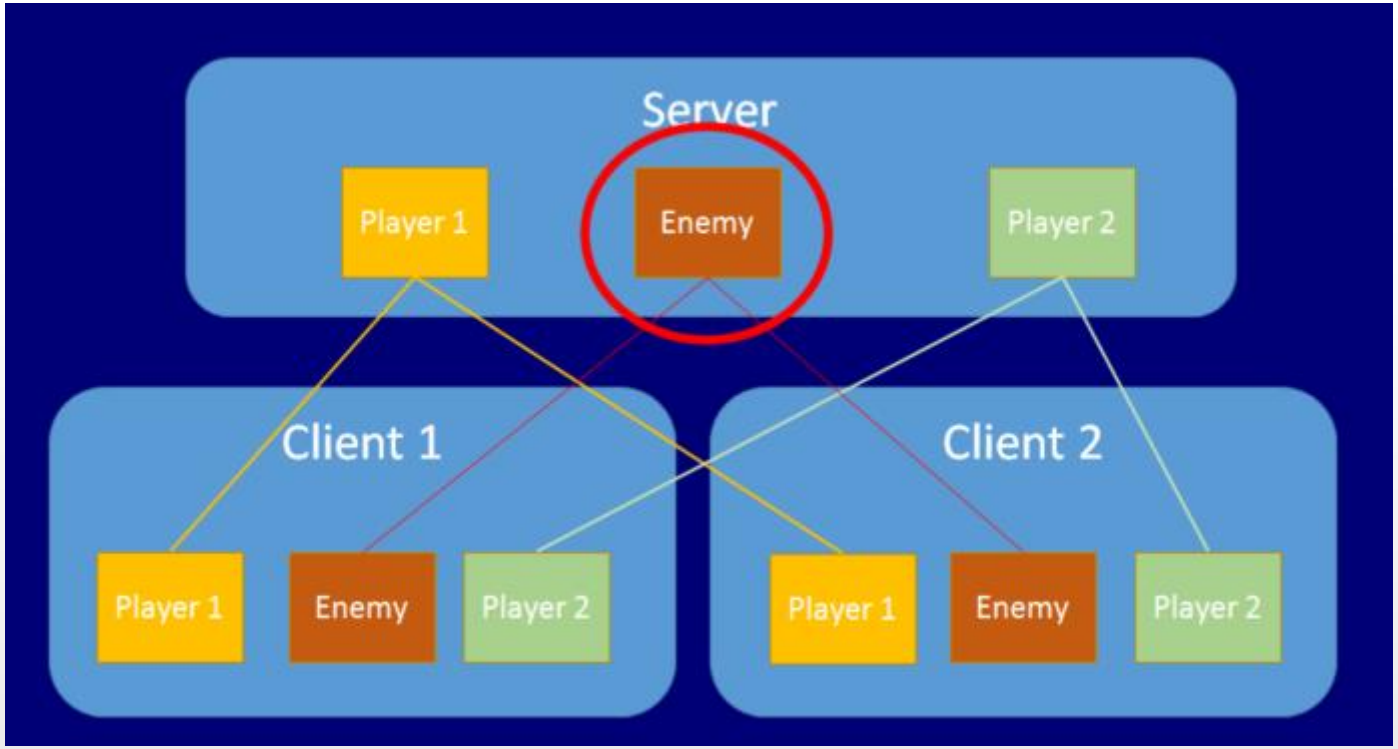
- Non-Player gameobjects are all other networked gameobjects in your scene

Players and Local Players on a Dedicated Server



Each client is authoritative (maneuvers) its own player object (inside circles) and update the information for other player objects accordingly to the messages coming from the server

Players and Non-Players on a Dedicated Server



The server is authoritative for NPCs and UNet will keep them synched on the clients

Spawning Gameobjects

- In single player you Instantiate() an object

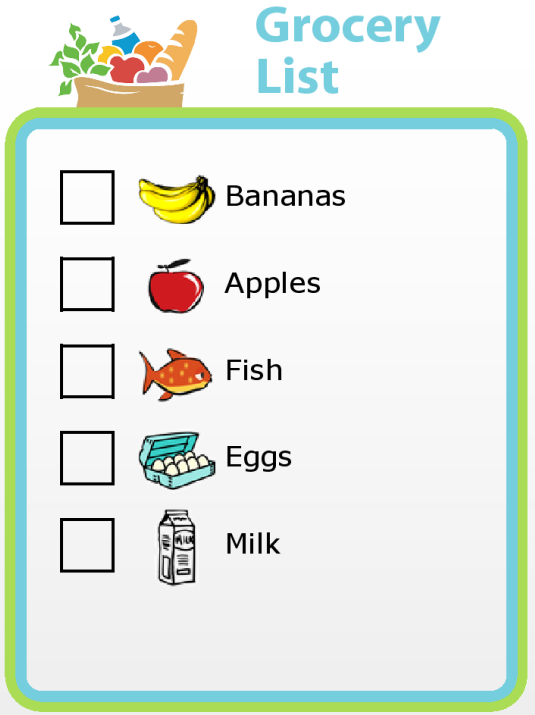
- In multiplayer you Spawn() an (already instantiated) networked object
  - You ask the server to replicate it on every client
  - Remember to put it in the “spawnable list”
  - Spawned object **MUST** have a NetworkIdentity component
  - NetCode changed the name to NetworkObject

- Spawn() is a static method of the NetworkServer class
  - NetCode moved Spawn() to NetworkObject (still static)

How to Set Up a Game (Quickly)

- Here is a minimal shopping list

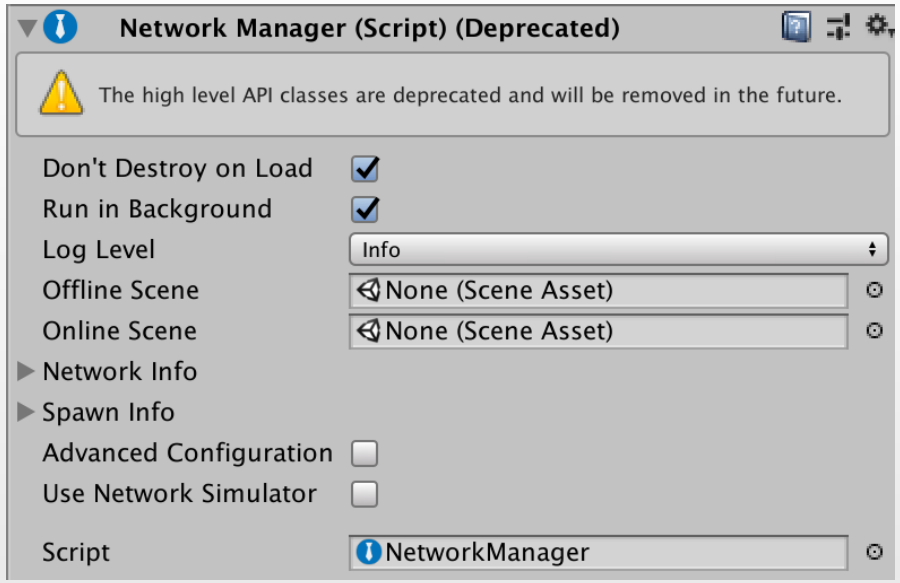
1. A network Manager
   - To coordinate things
2. A (visual) interface to a lobby
   - To find and join games
3. Networked scripts
   - Designed for multiplayer



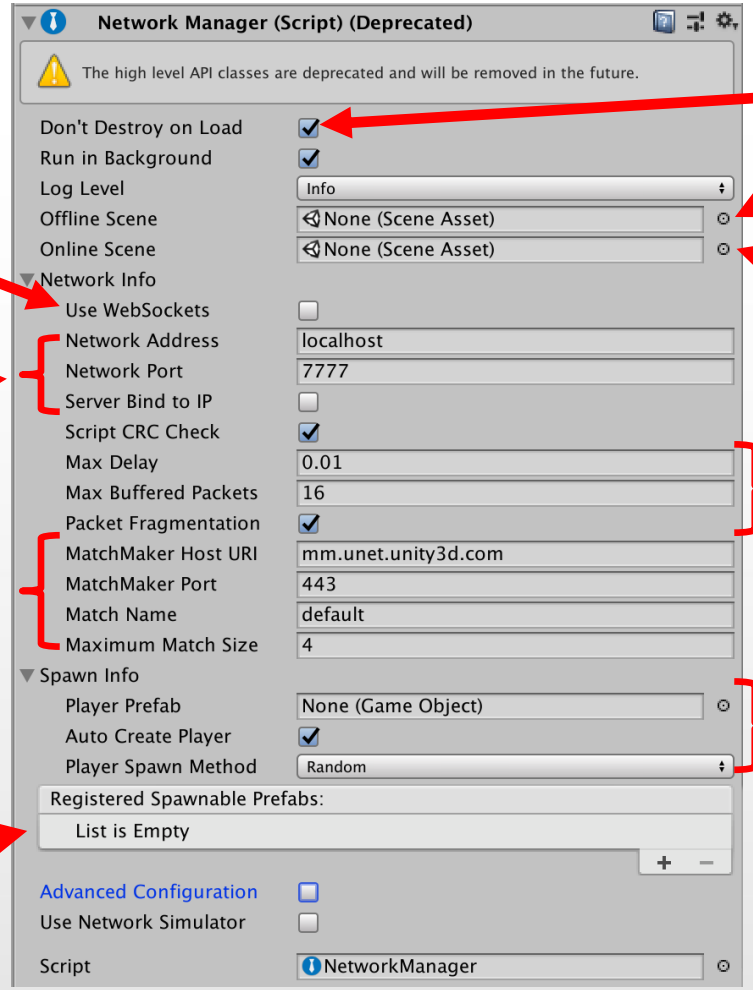
Network Manager (It is a Component!)

- Put it somewhere, in any gameobject
  WHICH IS NOT NETWORKED
- YOU MUST HAVE ONE
- YOU CAN HAVE ONLY ONE

NOTE
While extending NetworkManager is a recommended pattern, it is discouraged in NetCode



Anatomy of the NetworkManager Component



Check this if you have web clients

Server network configuration

Matchmaking service from UT

List of prefabs that can be spawned on clients

Keep this between scene changes

Scene with menu

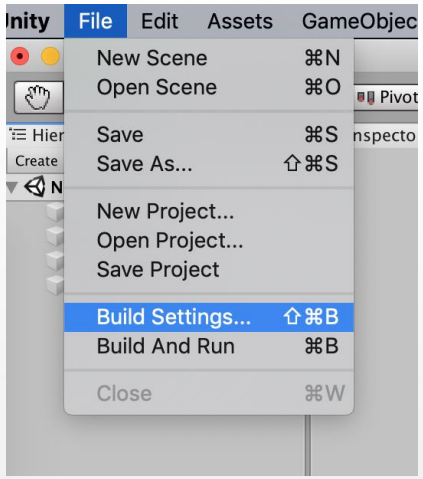
Scene with level

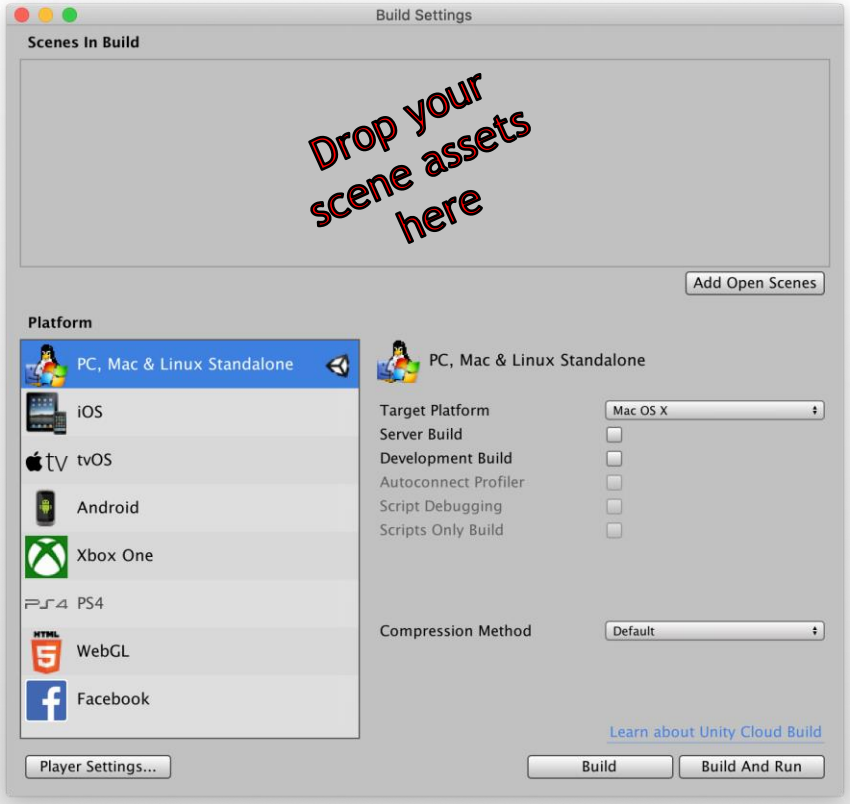
Network QoS

How to represent the player and where to create it. Just put one or more objects with a NetworkStartPosition component inside the online scene

Tip About Setting the Scenes

- To be accepted in the NetworkManager, a scene must be known in the build settings (order DOES matter)





You do not need to click “build”.
Just drag and drop your scenes and close the window.
Remember that, after building, unity will always run the first scene in the list

Using the Network Manager

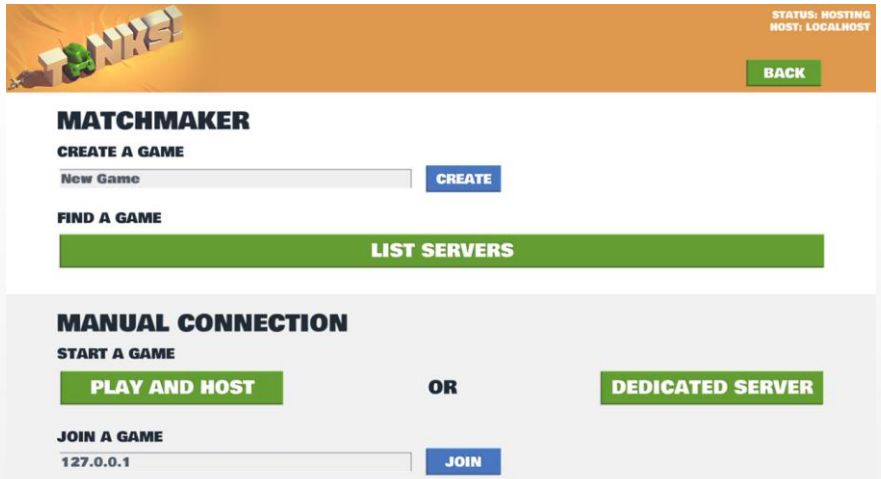
- NetworkManager by itself is doing NOTHING
- You must drive it in some way
  - This is why we need an interface

```
NetworkManager.StartClient() // work as a client
NetworkManager.StartServer() // work as a server
NetworkManager.StartHost()   // work as both client and server
```

- TIP: since there is only one, you can reach it easily using NetworkManager.singleton
- Documentation on all above is kind of

Interface to the Lobby

- You need an interface to the lobby to drive the network manager
  - Something like this would be perfect



- Luckily enough, Unity is providing a basic one out of the box
  - Just add a NetworkManagerHUD Component to the same object where the NetworkManager is (otherwise, it will create another NetworkManager)
  - NetCode is NOT implementing this as of today

Sort of ....

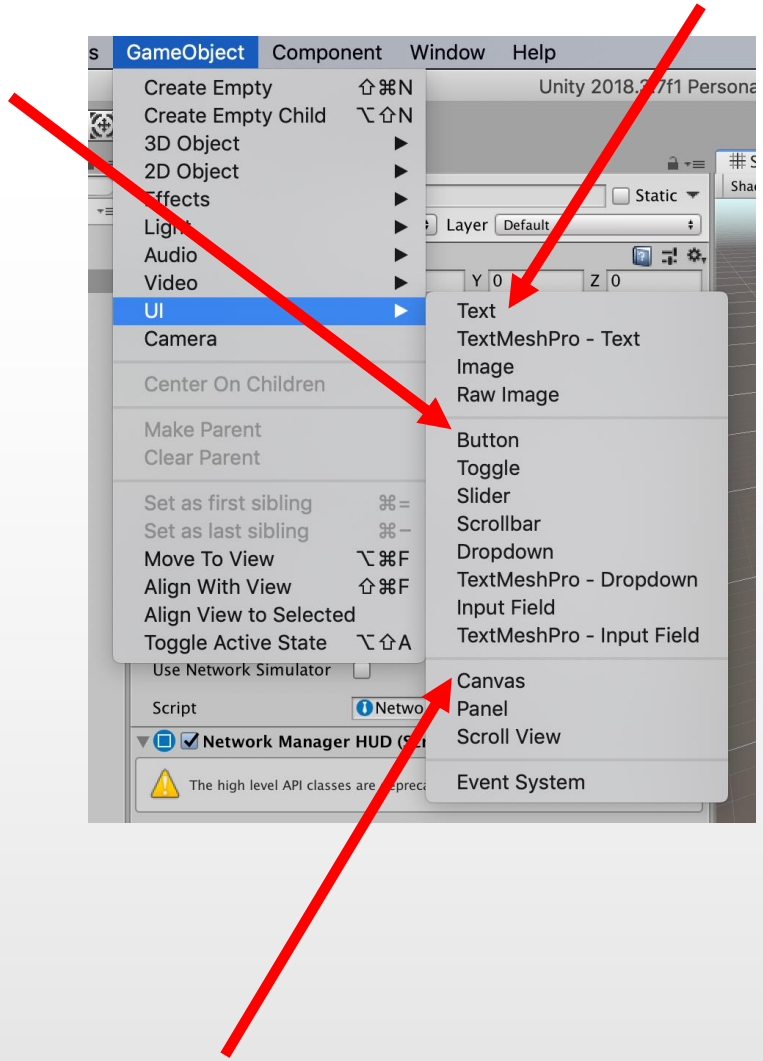

Game
Display 1
1920x1080
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmos
LAN Host(H)
LAN Client(C)
localhost
LAN Server Only(S)
Enable Match Maker (M)
FullHD screen

Tip About Using the Scenes

- When first started, Unity will stay on the current scene; while on connection and disconnection the online and offline scenes will be loaded respectively
- If the NetworkManager and the HUD are in a third scene the HUD will work, but what the player will see beyond the UI will be different at startup and after disconnection
- It is a good idea, as a beginner, to start with only two scenes: one for online and one for offline. Then, put the NetworkManager and the HUD in the offline scene

Creating Your Own Lobby Interface

1. Create a UI canvas
2. Add a button for each action
3. Link buttons callback to NetworkManager functionalities



We already discussed how to setup a UI

Building a Networked Game

- We already know about NetworkManager and NetworkManagerHUD
- Every networked gameobject MUST have a NetworkIdentity (NetworkObject) component to be recognized
  - Check the localPlayerAuthority flag if it must be controlled by the player on a client
- Every networked object moving around MUST have a NetworkTransform component to be able to sync correctly
  - Remember to set the authority in the network transform!
  - For the player avatar you want to set Local Player Authority
- [your] Components operating on networked objects MUST extend NetworkBehaviour and check the isLocalPlayer field on Update
  - Because you need to update only your avatar; other avatars will be updated by the NetworkTransform (or similar) components
  - NetCode changed the name to IsLocalPlayer (uppercase I)

Very Important About “isLocalPlayer”

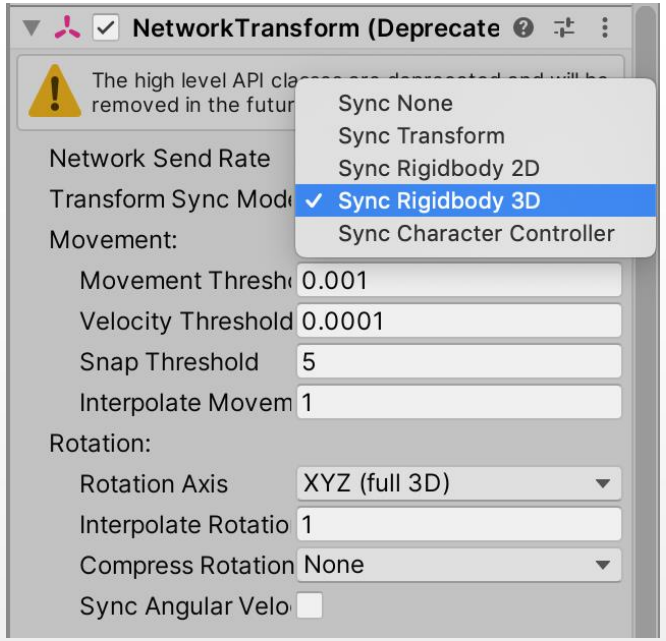
- The field isLocalPlayer is a facility for you, and not for the system

- You can update an object for which you have no authority and no error will be raised, but the parameters will NOT sync
  - So, you will see only your local copy changing

Not Caring About *isLocalPlayer*

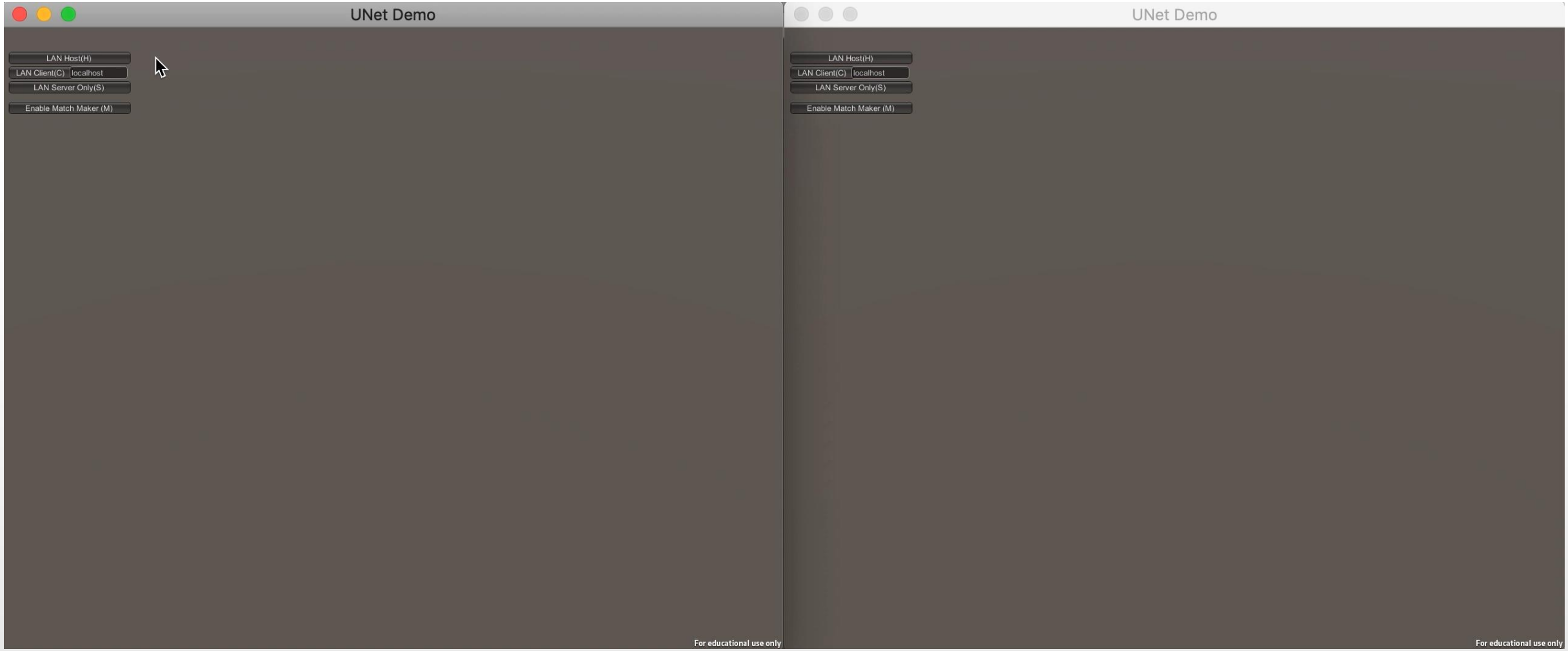
Beware of NetworkTransform and Physics

- The cheat-proof approach we saw before may have a lot of sense in the case when physics is involved

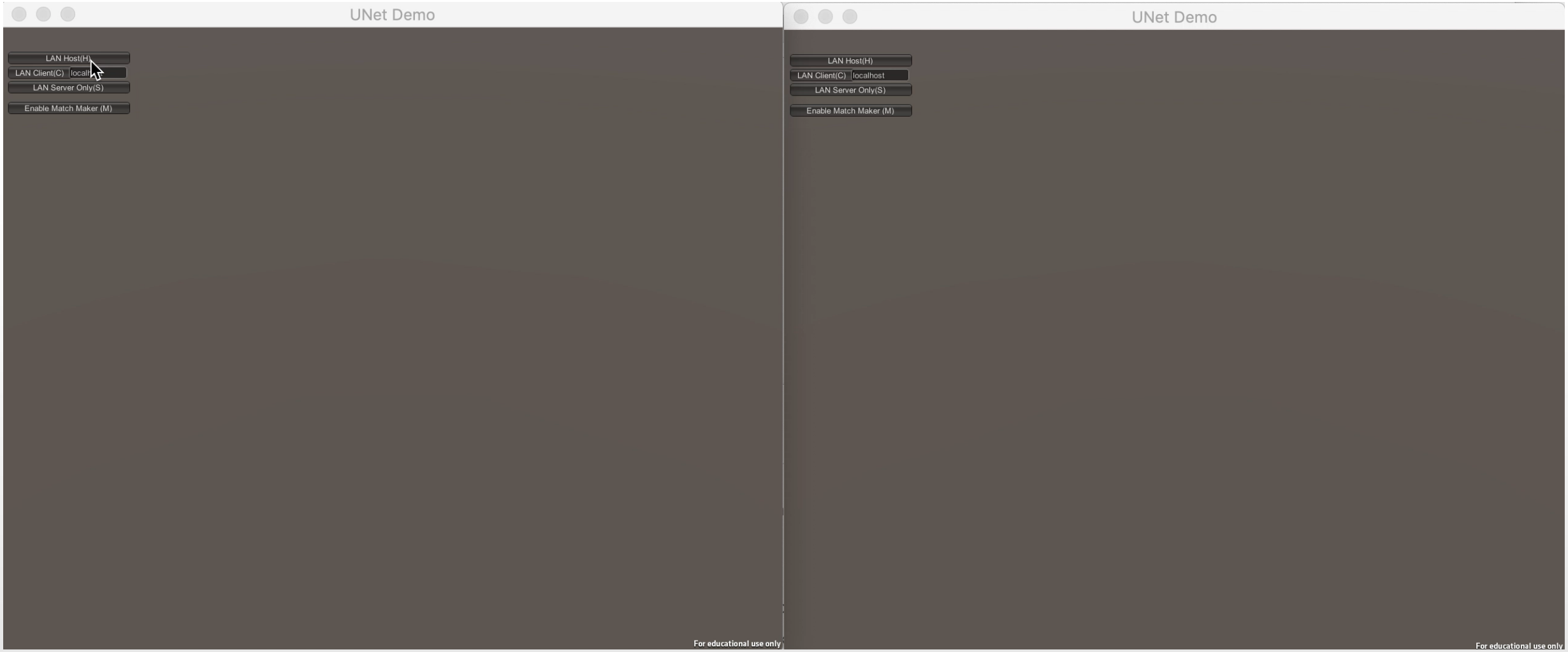


- The NetworkTransform can select what to sync
  - It will auto-select RiggidBody 3D (2D) based on the presence of other components
- Syncing rigid bodies might create unstable results on the remote side, such as vibrations, jumps, rolls, or missed collisions
- In this case, much better to use an invisible gameobject on the clients, ask the server to perform all the physics calculation, and sync back the geometric transform
- Otherwise, you can force the use of ”sync transform” despite the presence of the RigidBody and set the network send rate parameter to the maximum
  - In this case you will greatly increase the network usage and collisions will be stuttering anyway
- NetCode NetworkTransform is NOT able to sync rigid bodies

NetworkTransform Syncing RigidBody 3D



NetworkTransform Syncing Transform



Back to the Cannon

- We need to let the player fire a cannonball, but we do not want, for the sake of the game, to let him decide the result of the cannonball landing

- "Ask" the server to create (and take care of) the cannonball
  - When collision happens, damage and results will be observed and calculated by the server; then, the clients will be informed
  - If something must explode, each client will make it explode

- We need a way to send requests across the network
  - The client will issue a Command to the server to ask the execution of a method
    - E.g., "fire a cannonball"
  - The server will issue a ClientRPC (Remote Procedure Call) to all clients to ask an execution of a method
    - E.g., "make this ship explode"

Commands vs ClientRPCs

- A command is a request from a client to the server
  - Add a [Command] decoration to a method
  - When invoked (by the client) the method will be executed by the server
  - A command MUST have a “Cmd” prefix in its name
    - Not doing so will generate a non-blocking error at compile time. Nevertheless, the command will not work
  - NetCode uses [ServerRPC] in place of [Command]

- A ClientRPC is a request from the server to ALL clients
  - Add a [ClientRpc] decoration to a method
  - When invoked (by the server) the method will be executed by all clients
  - A command MUST have a “Rpc” prefix in its name
  - ClientRPC are broadcast, but also contextual
  - NetCode uses [ClientRPC] in place of [ClientRpc] (another big change)

On this last point, I believe the Unity documentation is not very clear

- In both cases, NetCode changet Cmd and Rpc from prefix to postfix

ClientRPC and Context

- Keep in mind that the RPC method is bound to the object where the NetworkBehaviour is attached to
  - So, when the RPC is called, it will be executed by all clients, but on the same object

- With RPCs you can do, with one call:
  “hello everybody, please raise gameobject X by one meter”

- With RPCs you CANNOT do, in one call:
  “hello everybody, please raise yourself (your avatar) by one meter”

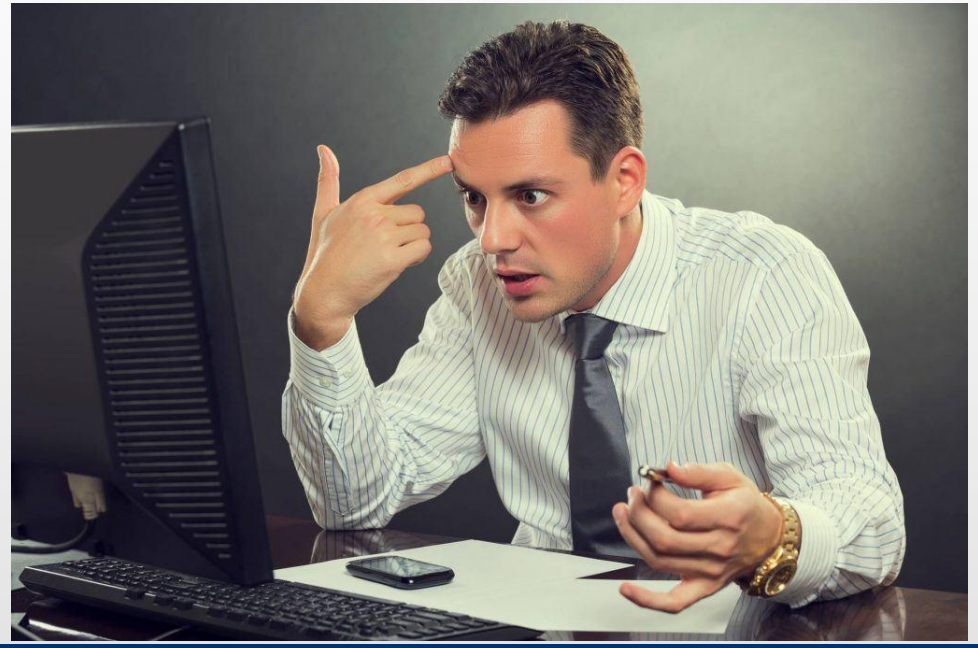
- Commands are used to tell the server what **a single client** want to do and ClientRPCs are used by the server to tell **all the clients** what actually happened

Running Your Networked Project

- No way to have two unity instances running on the same codebase
  - At least, if you do care about consistency ... and you DO!

- Your only option is as follows:
  - Build the project
  - Run it as an application
  - Run another instance in the editor
  - ... and debug!

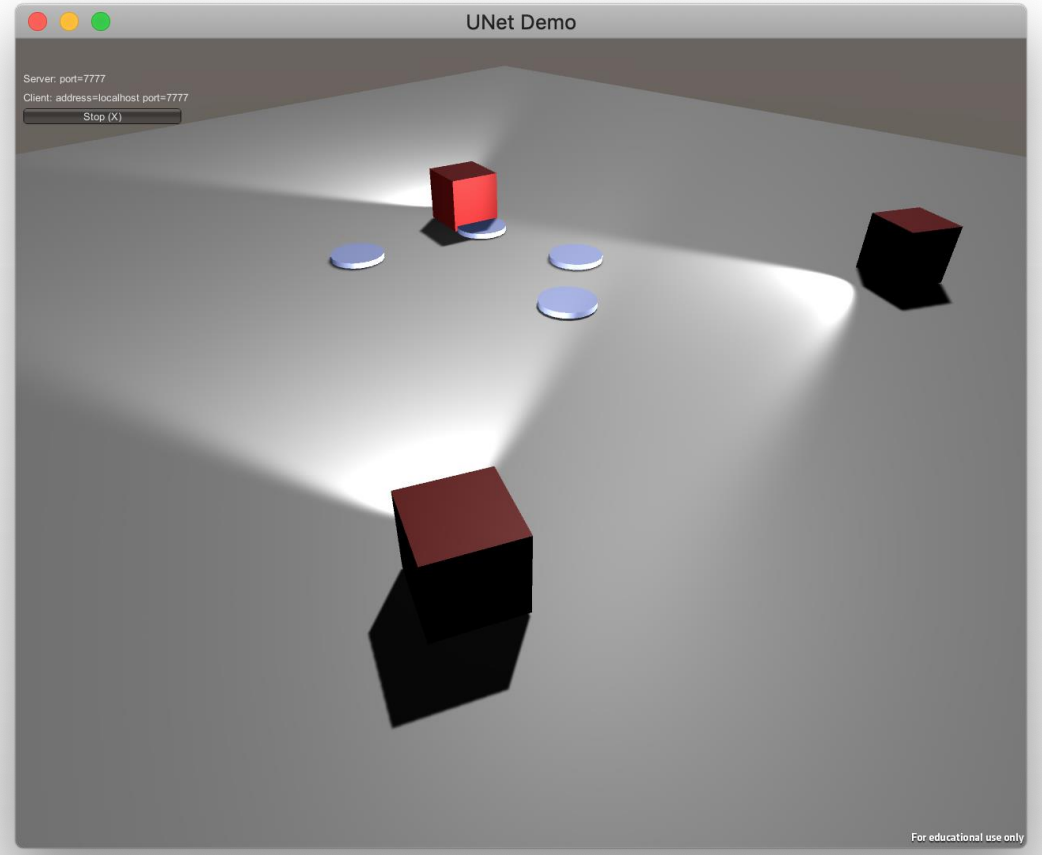
If your game is stuttering and you do not know why, try raising the Network Sending Rate parameter in the NetworkTransform components giving you troubles



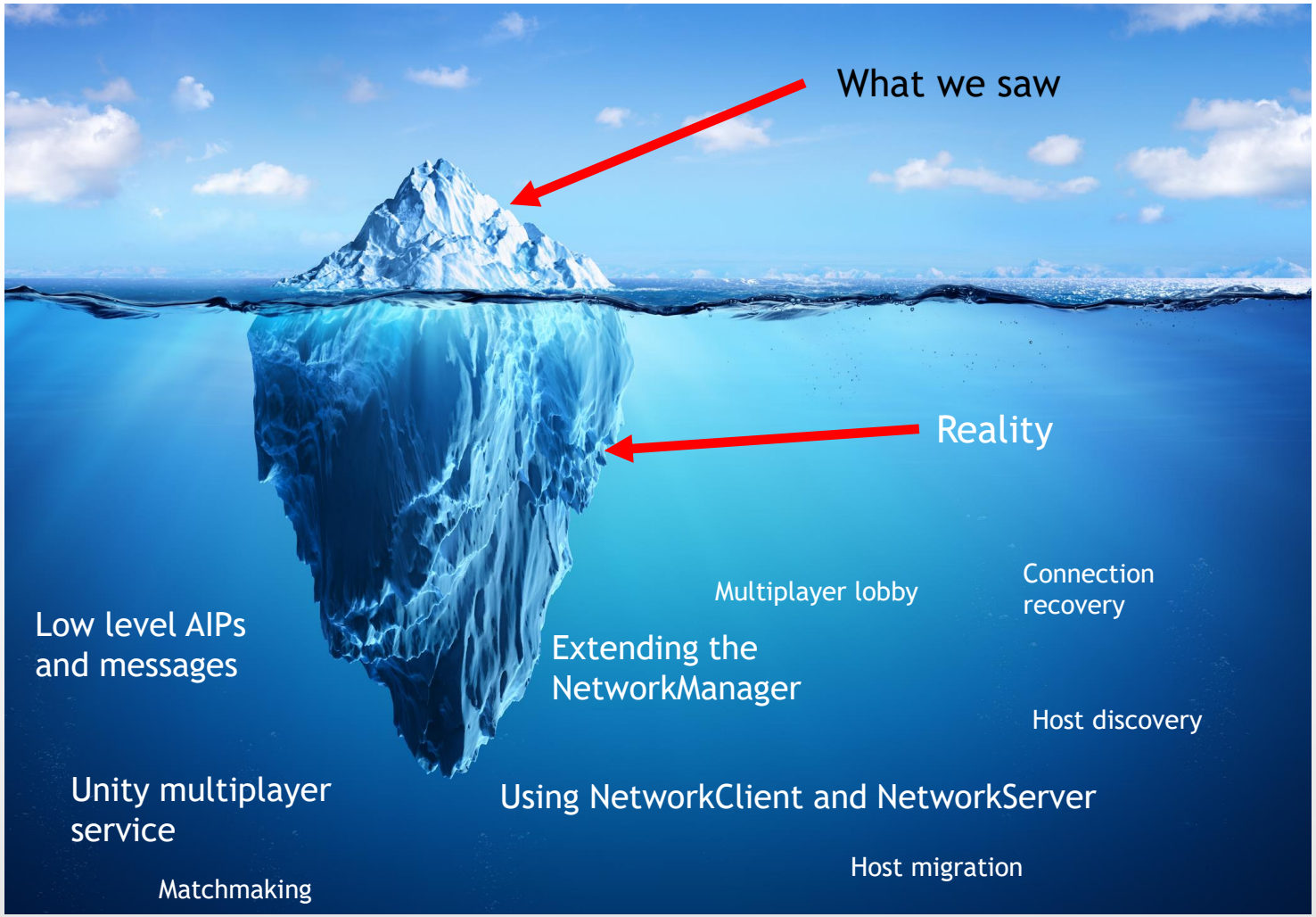
A Quick Example

- In this example there is a minimal P2P system
  - The offline scene holds the connection menu
  - The online scene holds the level
- Each player will control an avatar made of a red cube and a headlight
- The player can move around and lay objects on the ground by pressing the space bar
  - The lay is a command to the server to spawn the object
- While laying, the player has no control over the avatar. The avatar will go straight for a fixed amount of time
  - The movement while laying is the result of an RPC to the client issued in response to the lay command

A Quick Example



But, as Usual ...


What we saw
Reality
Low level AIPs and messages
Multiplayer lobby
Connection recovery
Extending the NetworkManager
Host discovery
Unity multiplayer service
Using NetworkClient and NetworkServer
Host migration
Matchmaking

Where to Look Now

- https://docs.unity3d.com/Manual/UNet.html
- https://docs-multiplayer.unity3d.com/
  - Root of all evil documentation (long and boring)

- https://docs.unity3d.com/Manual/UNetReference.html
- https://docs-multiplayer.unity3d.com/docs/getting-started/about
  - Components reference

- https://docs.unity3d.com/Manual/UNetClassesReference.html
  - Classes reference

Happy Hacking

Dario Maggiorini (dario@di.unimi.it)