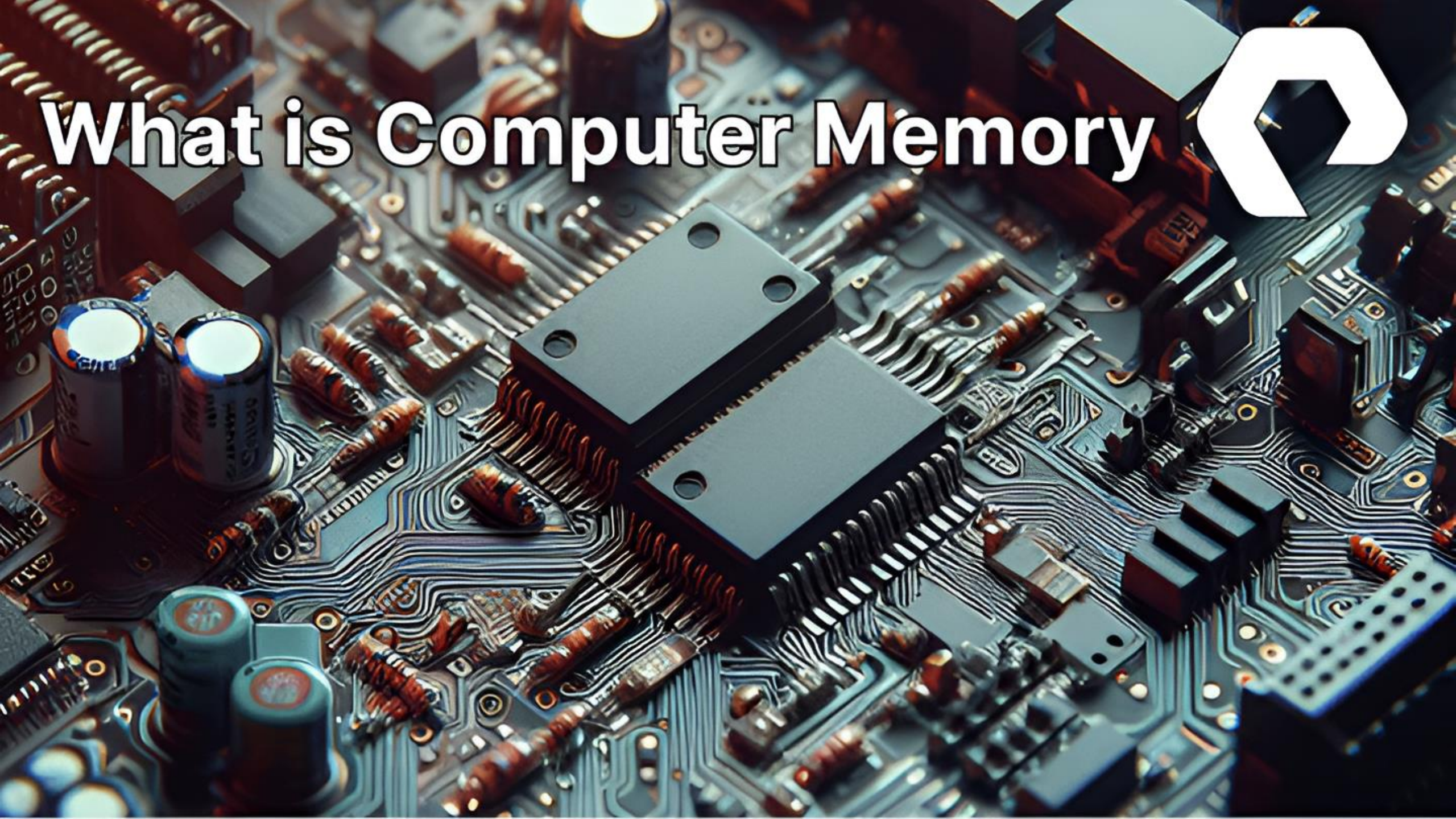
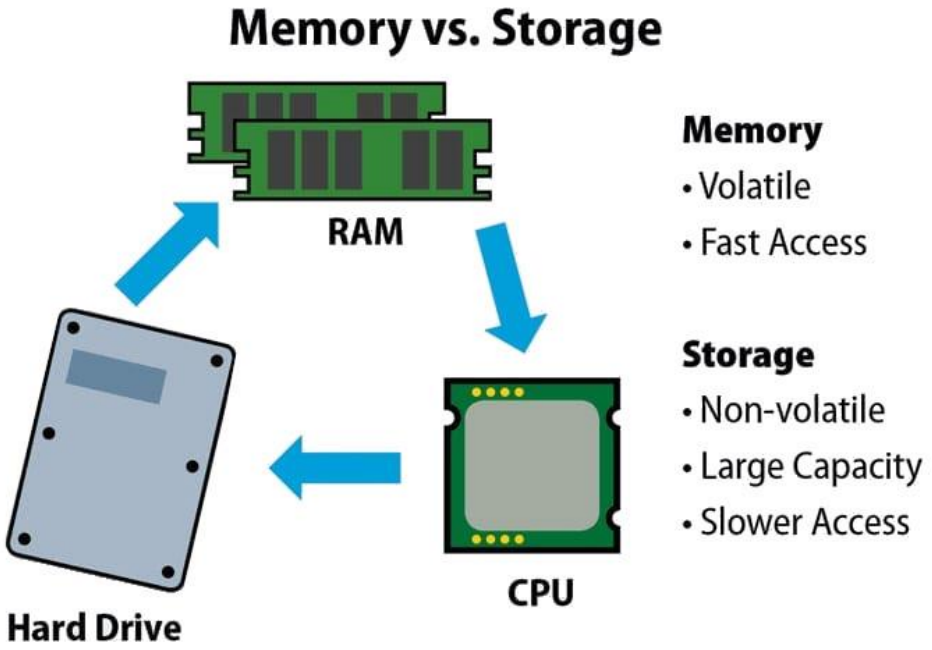
Optimization

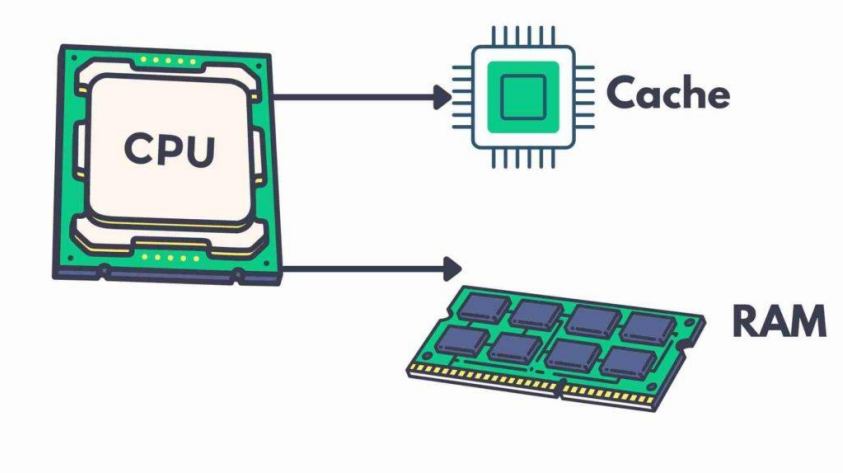
- COMPUTERS MEMORY
- FAST CODE BUT HOW?
- OOP, IS IT ANY GOOD?
- STACK VS HEAP
- MEMORY POOL
- ALGORITHM & DATA STRUCTURE
- LEVEL OF DETAILS (LOD)
- NORMAL & PARALLAX MAPPING
- RENDER TO TEXTURES
- ATLAS TEXTURE VS SPRITE SHEET
- BATCHING
- LIGHT MAPS & PROBES
- OCCLUSION CULLING
- PREFABS
- PROFILING
- COMMON SENSE


What is Computer Memory

A computer has multiple types of memory:

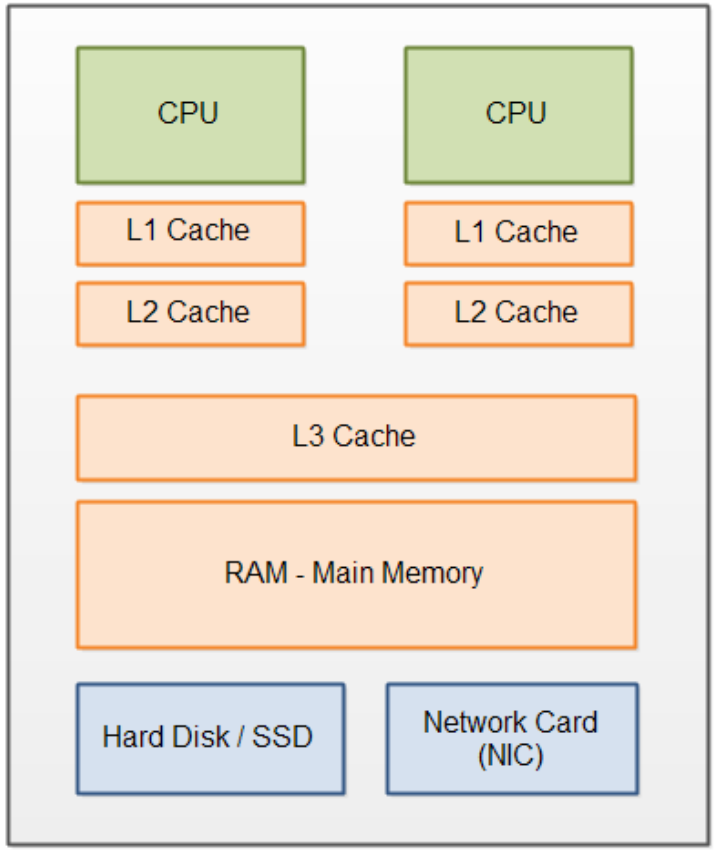
HDD
RAM
Cache





What is memory cache?

- Bridge between RAM and CPU?
- Performance booster?
- Legacy from the past?



Memory Caches:

L3 *- Fast*
L2 *- Faster*
L1 *- Fastest…*
smallest …
closest to CPU

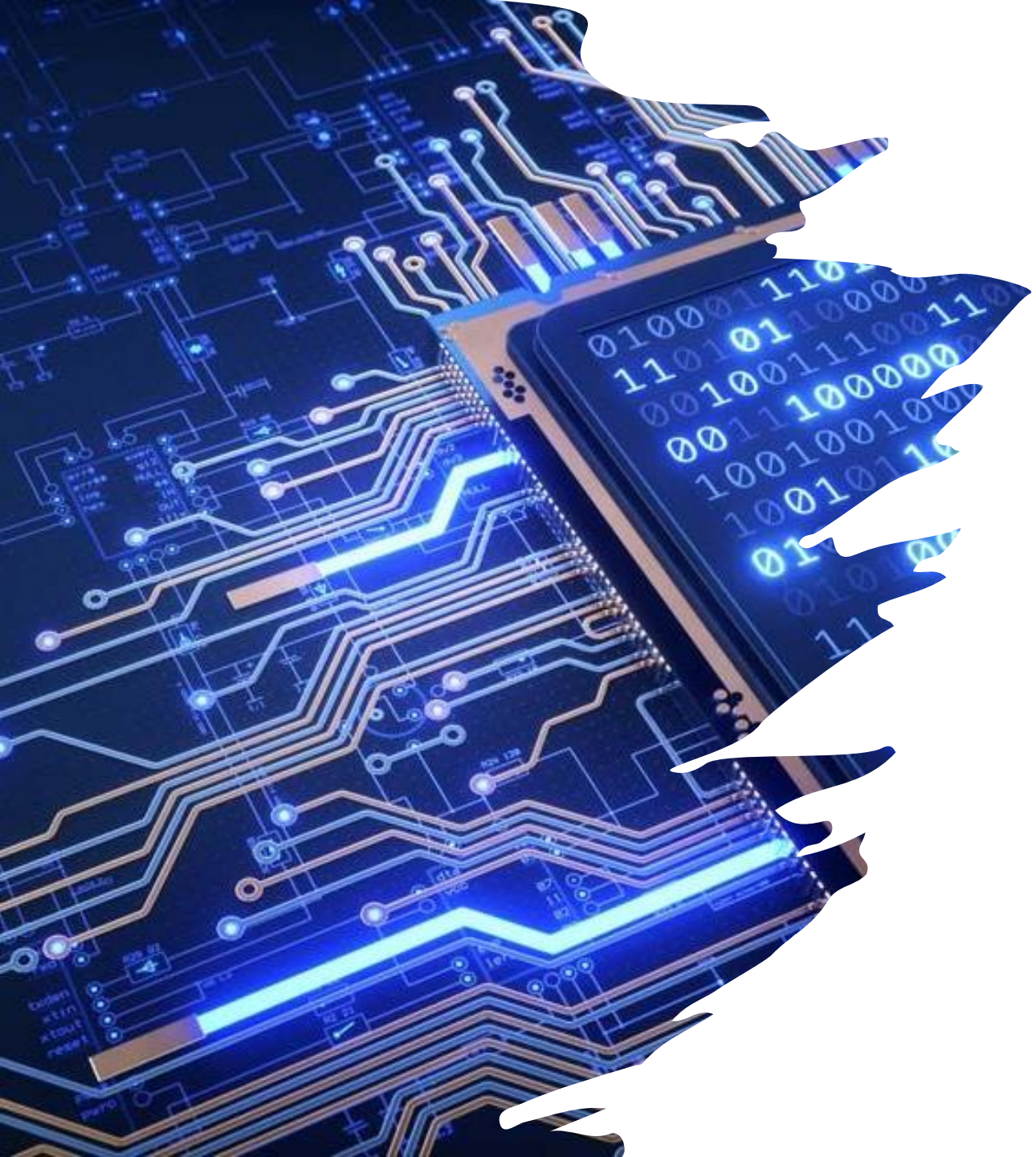
Hierarchal memory!

10-100 times faster than RAM!

How does it work?

Accessing a chunk of data in RAM:

- OS will find entry point in RAM
  This will require a lookup in virtual memory.
- OS will copy data to a memory cache
  If entire chunk of data is too large,
  only parts will be copied.
  This is a slow and daunting task.
- Our program can now read and write to copied data.
- OS will update RAM when it is ready

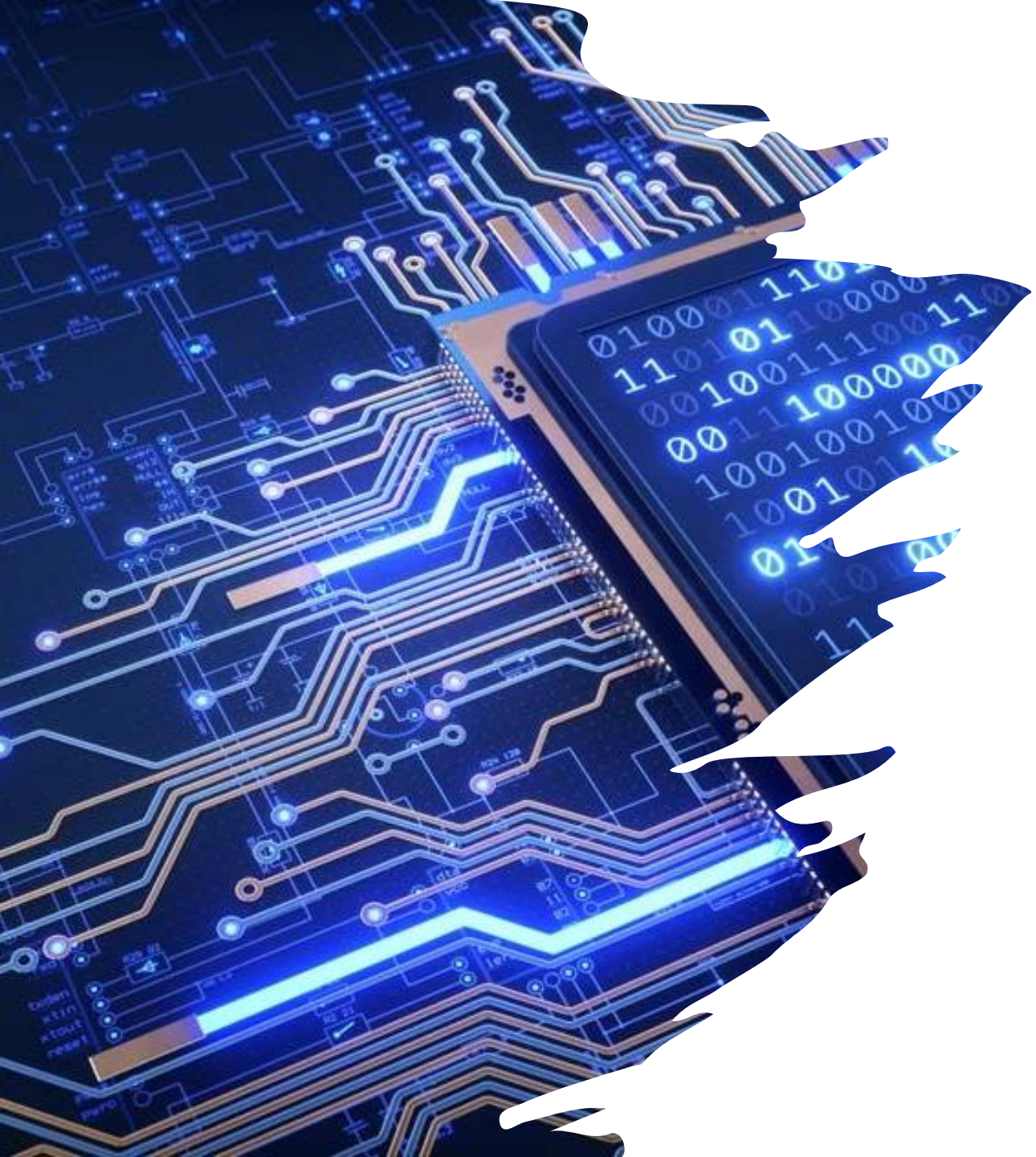


Fast code but how?

```
int [,] numbers = { { 1, 4, 2 }, { 3, 6, 8 } };

for (int i = 0; i < numbers.GetLength(0); i++)
{
    for (int j = 0; j < numbers.GetLength(1); j++)
    {
        Console.WriteLine(numbers[i, j]);
    }
}
```

How many arrays are there in this example?

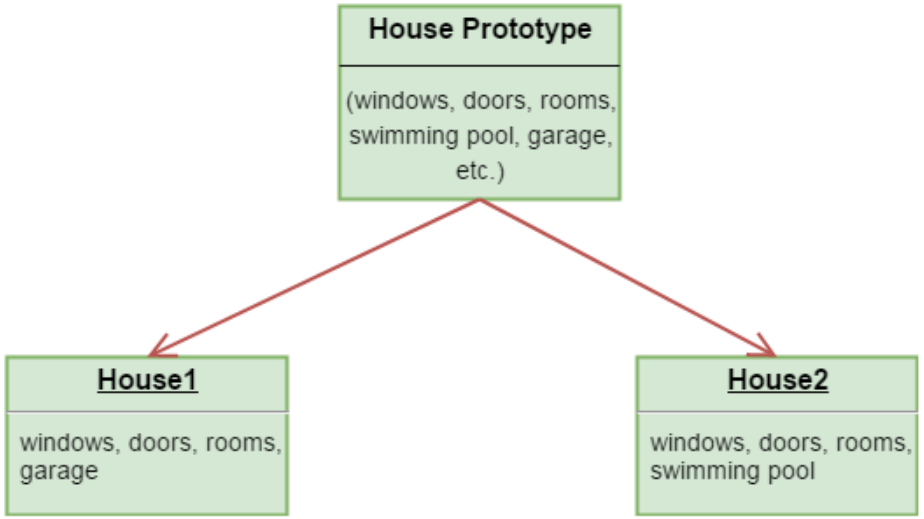


Fast code but how?

```
int [] numbers = { 1, 4, 2, 3, 6, 8 };

for (int i = 0; i < numbers.GetLength(0); i++)
{
    Console.WriteLine(numbers[i]);
}
```

Less code, easier to read, better performance!



OOP, is it any good?

- Pros:

- Easy to work with
- Easy to visualize
- Easy to architect
- Easy to maintain
- …

OOP, is it any good?

Example:

- We have created a city building game
- All buildings, including all interior, is defined in a complex hierarchy of classes
- Write an algorithm which will visit all buildings to check if they are locked.

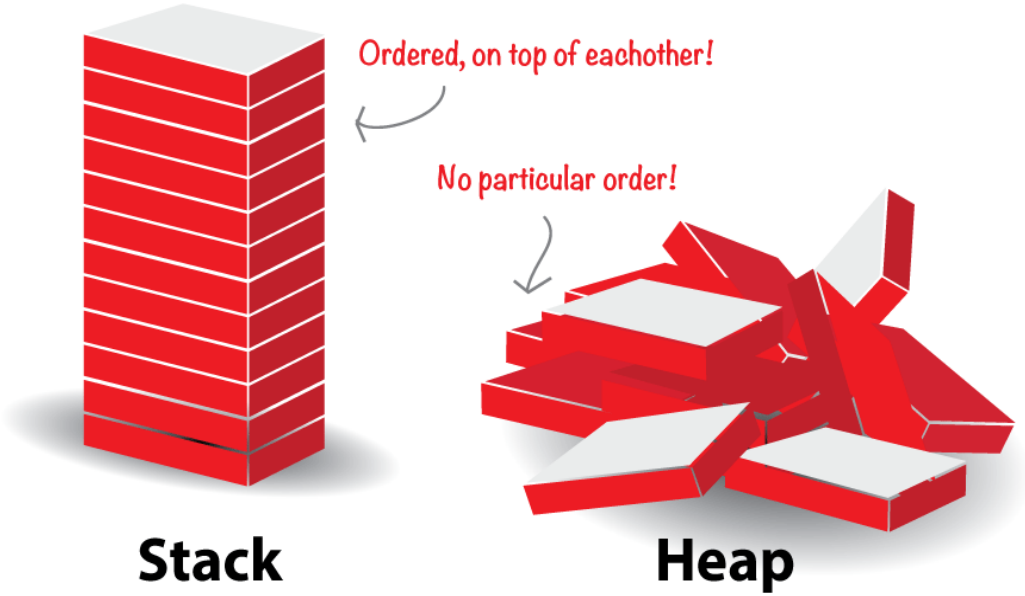
Any performance issues?

OOP, is it any good?

Answer:

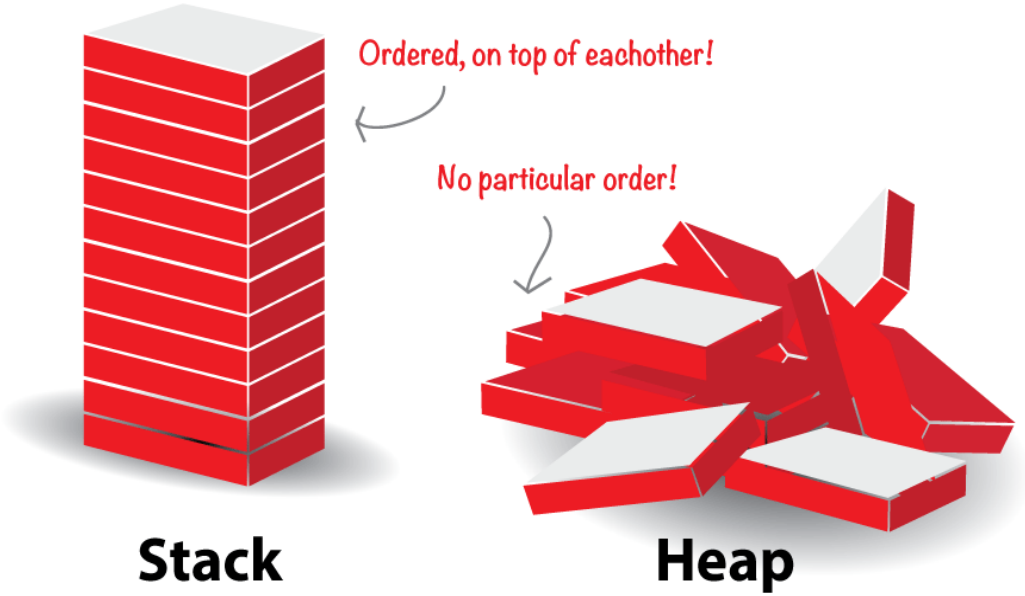
- If a building is built up by multiple classes, we will need to cache a lot of data, slow performance.
- We only need a Boolean to check if a building is accessible or not
- If only we had one array of Booleans for all buildings to check if they were locked or not

Enable Data Driven Design for Unity!



Stack vs Heap

WHY DO WE NEED TO KNOW THE DIFFERENCE?



Stack vs Heap

WHY DO WE NEED TO KNOW THE DIFFERENCE?

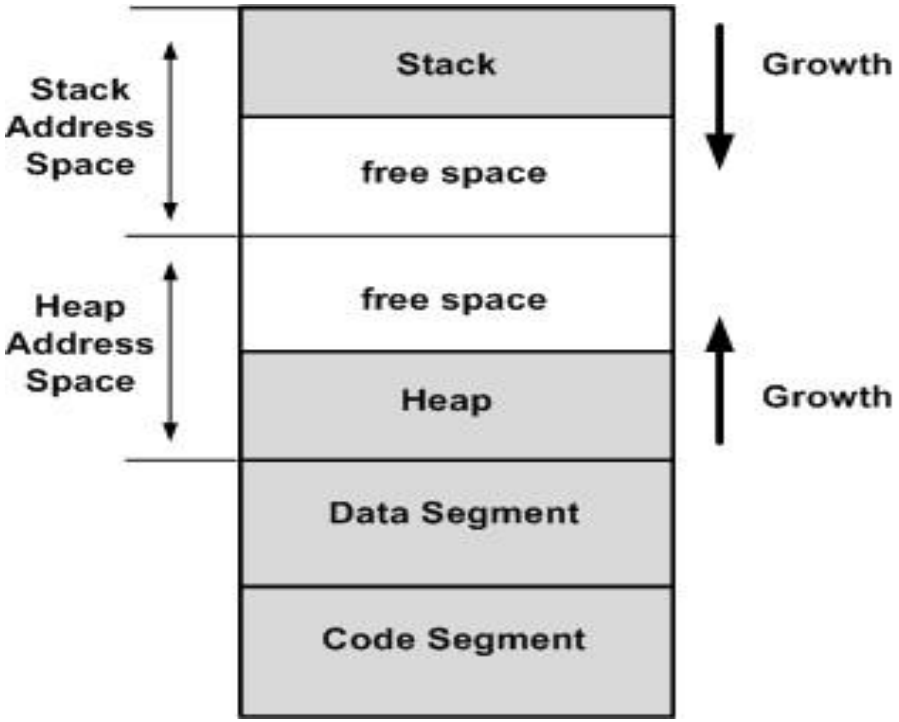
Because we use memory all the time!

Stack

- Local variables
- Memory is stacked (ordered)
- Fast memory allocation
- Automatically removed
- No risk of memory leak
- Limited memory size

Heap

- Global variables
- Memory stored randomly?
- Slow memory allocation
- Not removed automatically
- Risk of memory leak
- Unlimited memory size

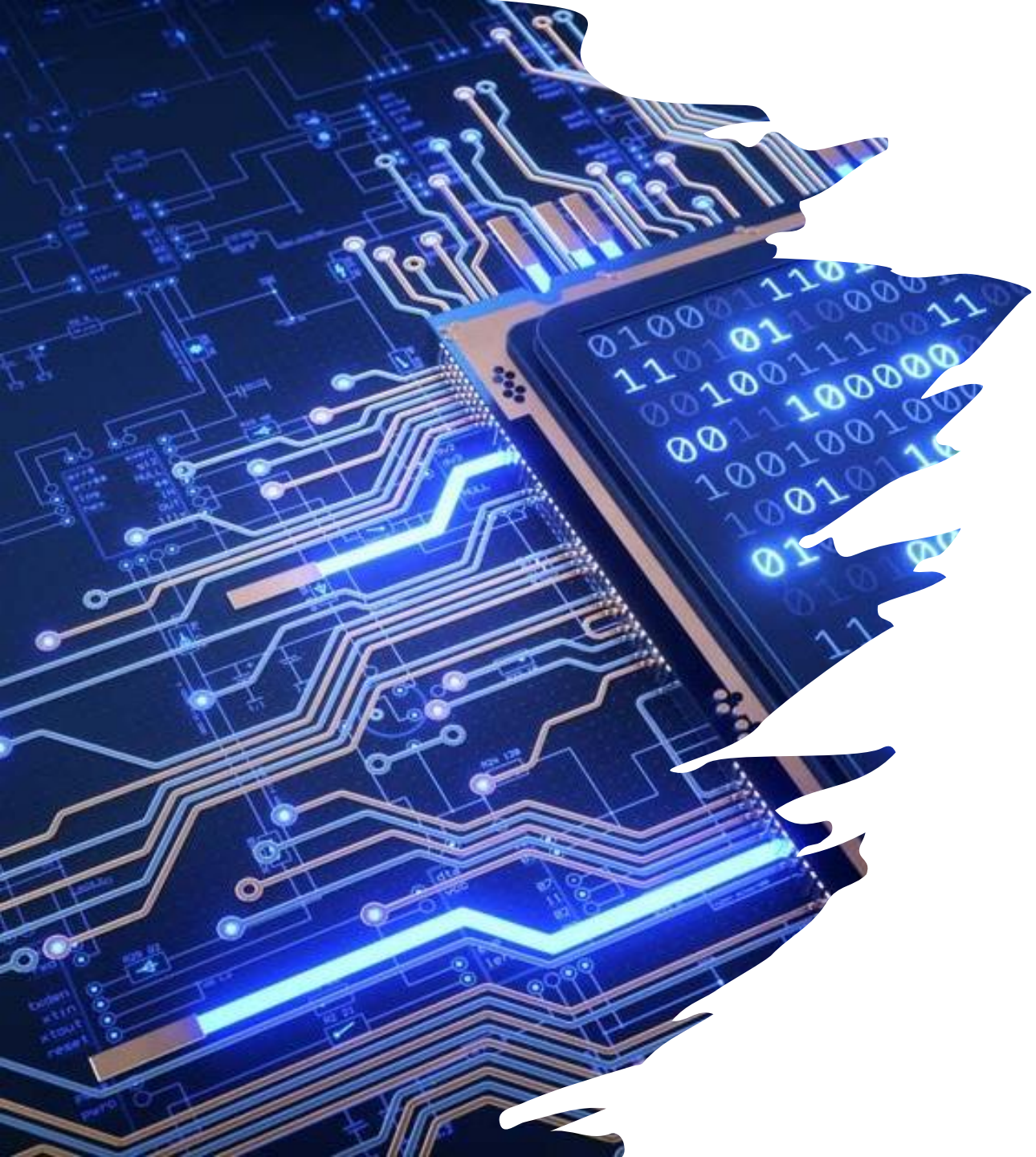


Stack vs Heap

ALWAYS USE **STACK** IF YOU CAN GET AWAY WITH IT… IT'S FAST!

Knowledge of how computer memory works is key for fast code!


A fast CPU will not help you, if you don’t have an efficient memory management!

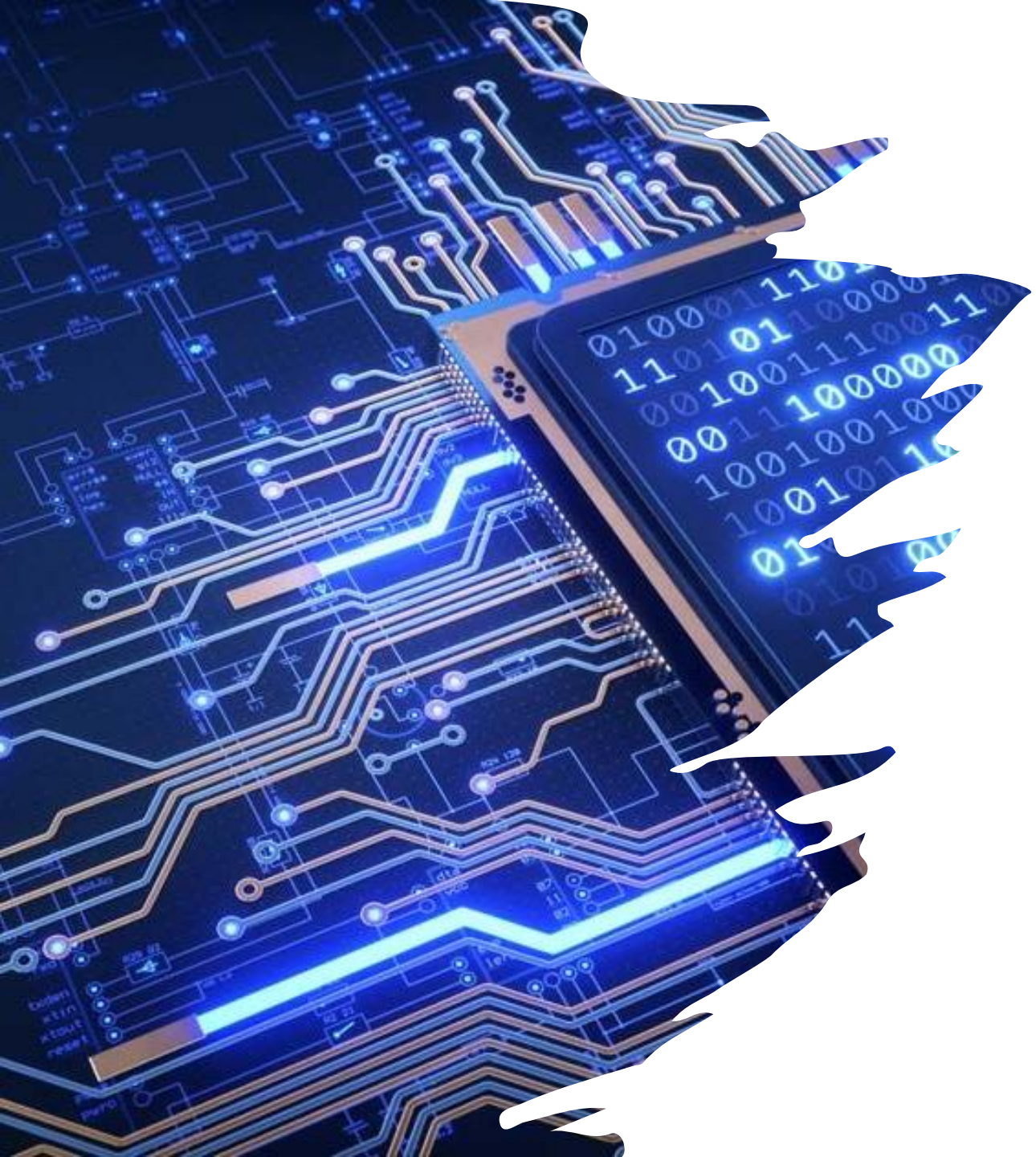


What memory do we use?

```
int [] numbers = { 1, 4, 2, 3, 6, 8 };

for (int i = 0; i < numbers.GetLength(0); i++)
{
    Console.WriteLine(numbers[i]);
}
```

Stack or Heap?

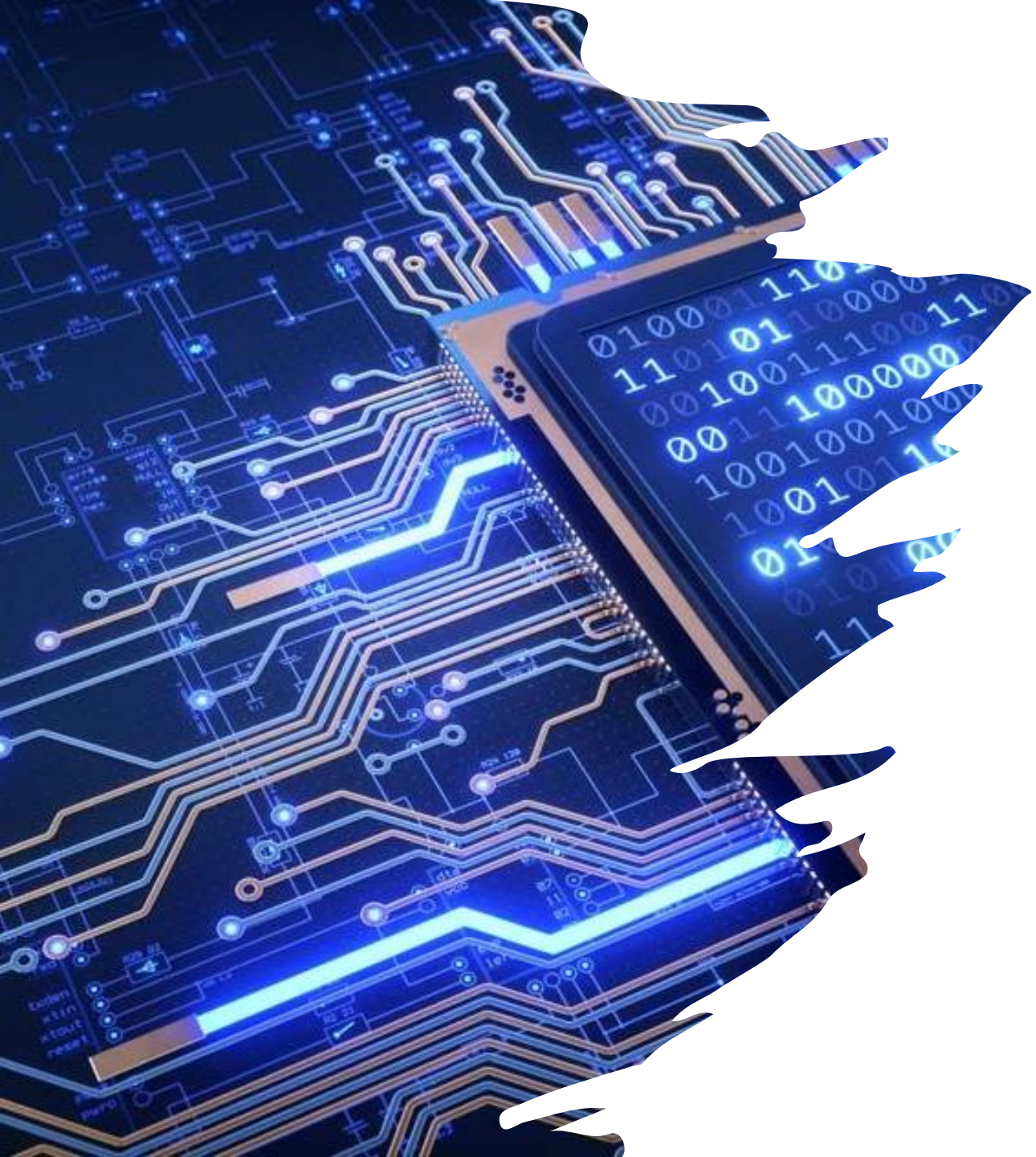


How do we use heap memory?

```
int [] numbers = { 1, 4, 2, 3, 6, 8 };

for (int i = 0; i < numbers.GetLength(0); i++)
{
    Console.WriteLine(numbers[i]);
}
```

Why would we want to use heap memory?

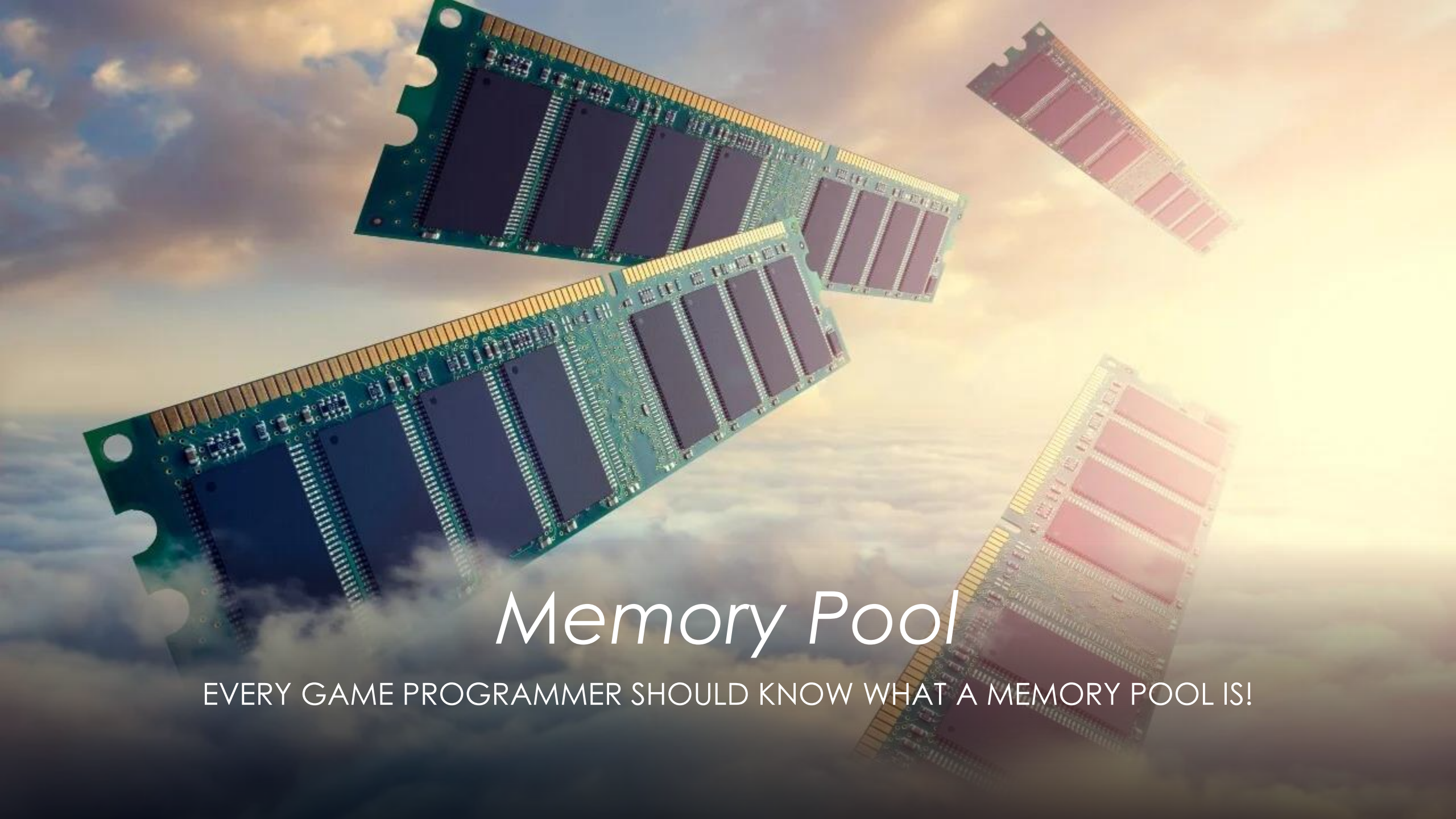


How do we use heap memory?

```
int [] numbers = new int [6];
numbers[0] = 1;
numbers[1] = 4;
…
numbers[7] = 8;

for (int i = 0; i < numbers.GetLength(0); i++)
{
    Console.WriteLine(numbers[i]);
}
```

Heap memory can be used as a memory pool!

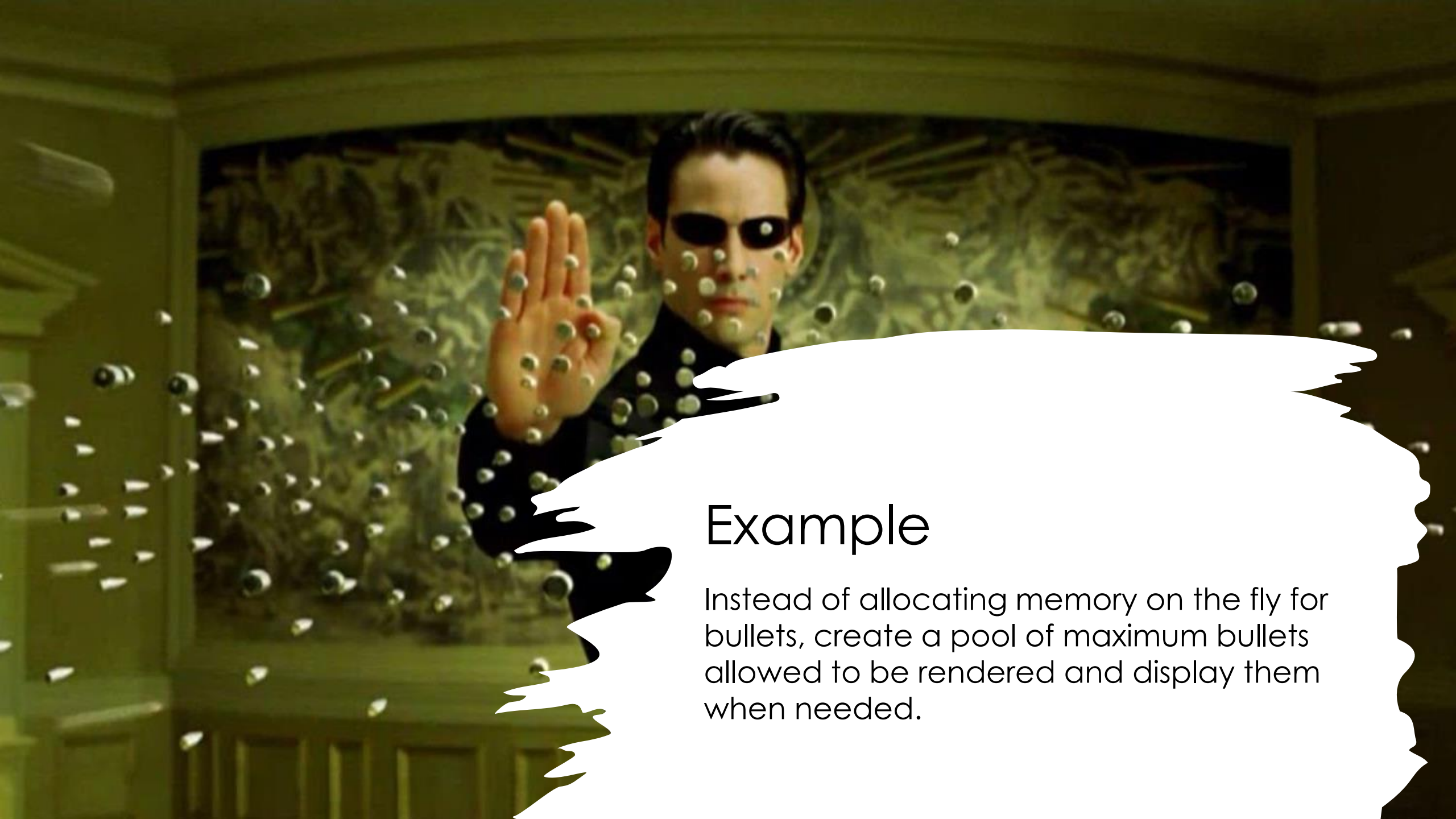


Memory Pool

EVERY GAME PROGRAMMER SHOULD KNOW WHAT A MEMORY POOL IS!

Memory Pool

- It is a common technique used to efficiently allocate and deallocate memory for data structures and objects during program execution.
- Instead of allocating memory separately on the fly, allocate a pool of memory and use it throughout the course of the game.

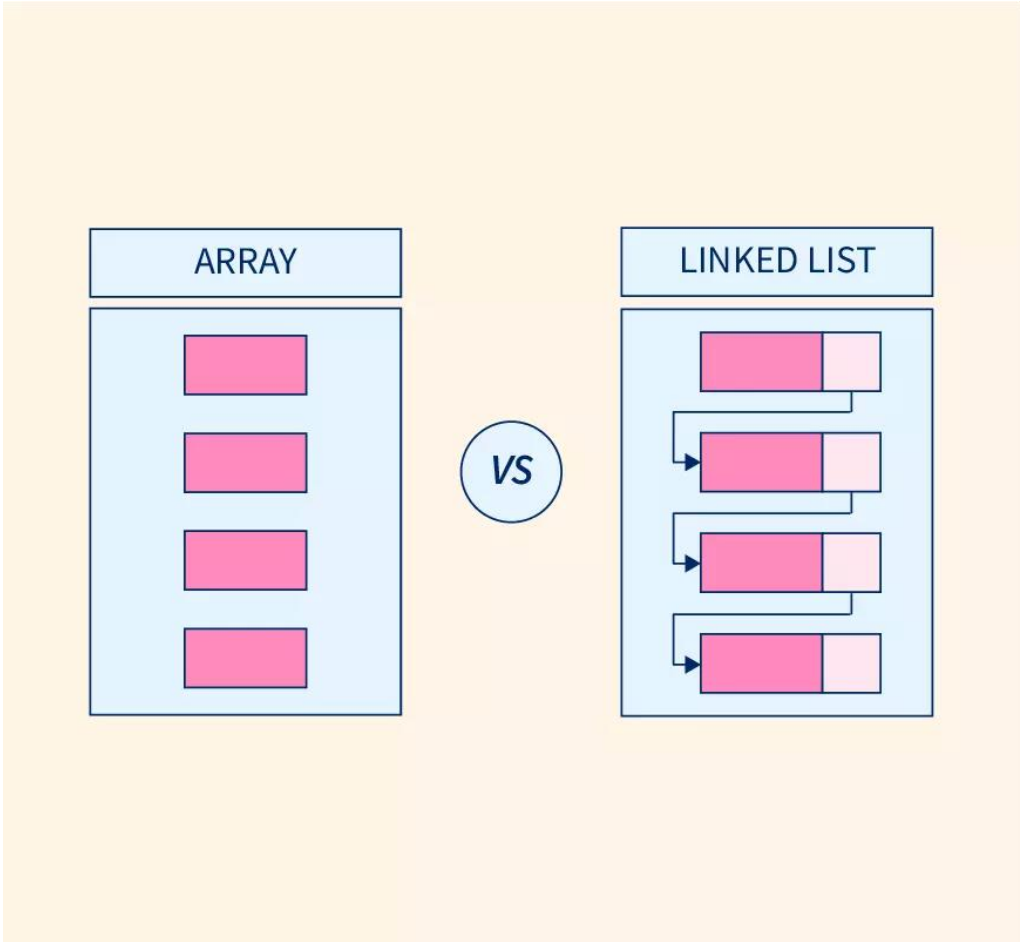


Example

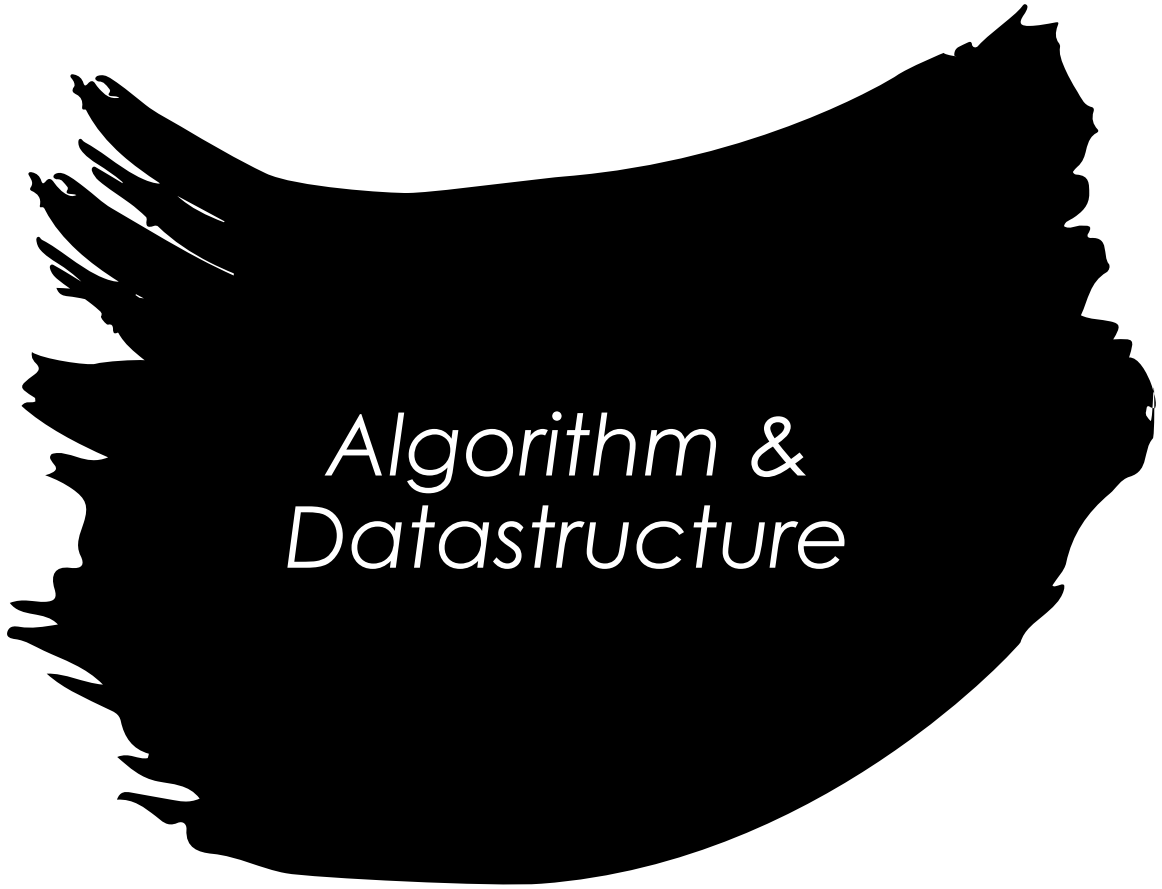
Instead of allocating memory on the fly for bullets, create a pool of maximum bullets allowed to be rendered and display them when needed.

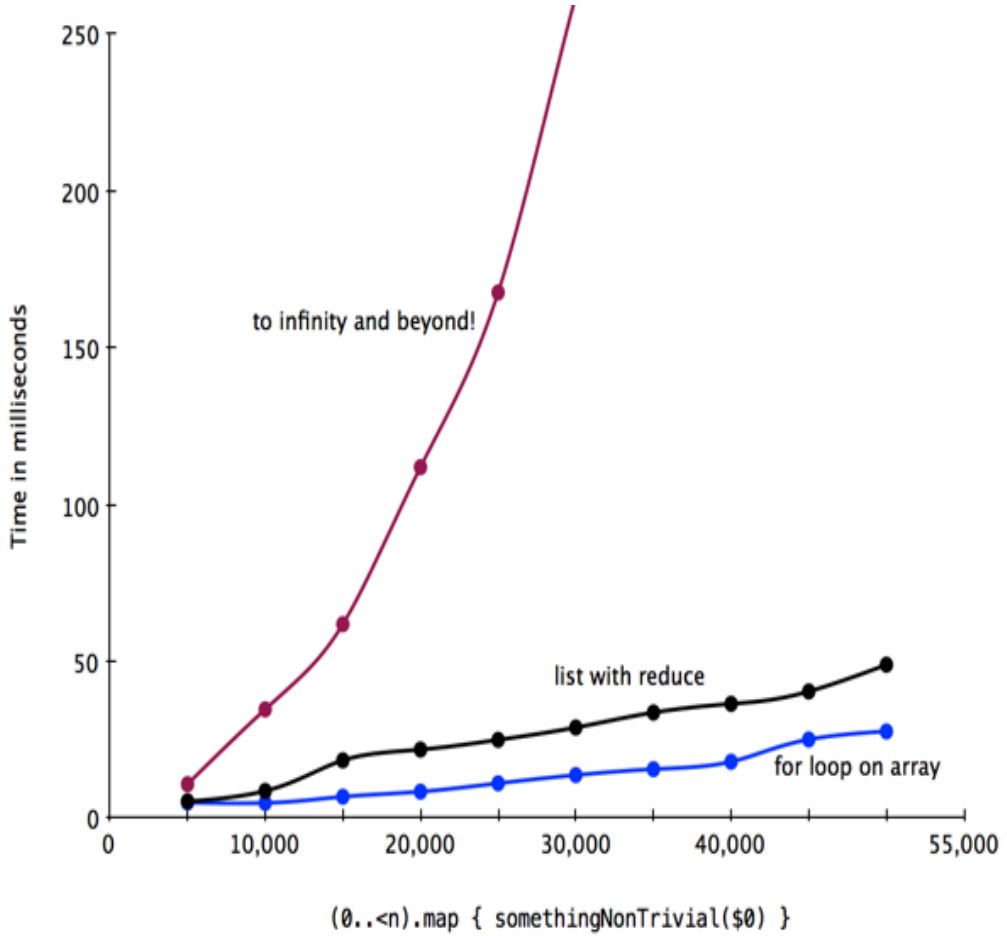

Most games have
a limited number of
ragdolls available
due to memory
pool

Algorithm & Datastructure



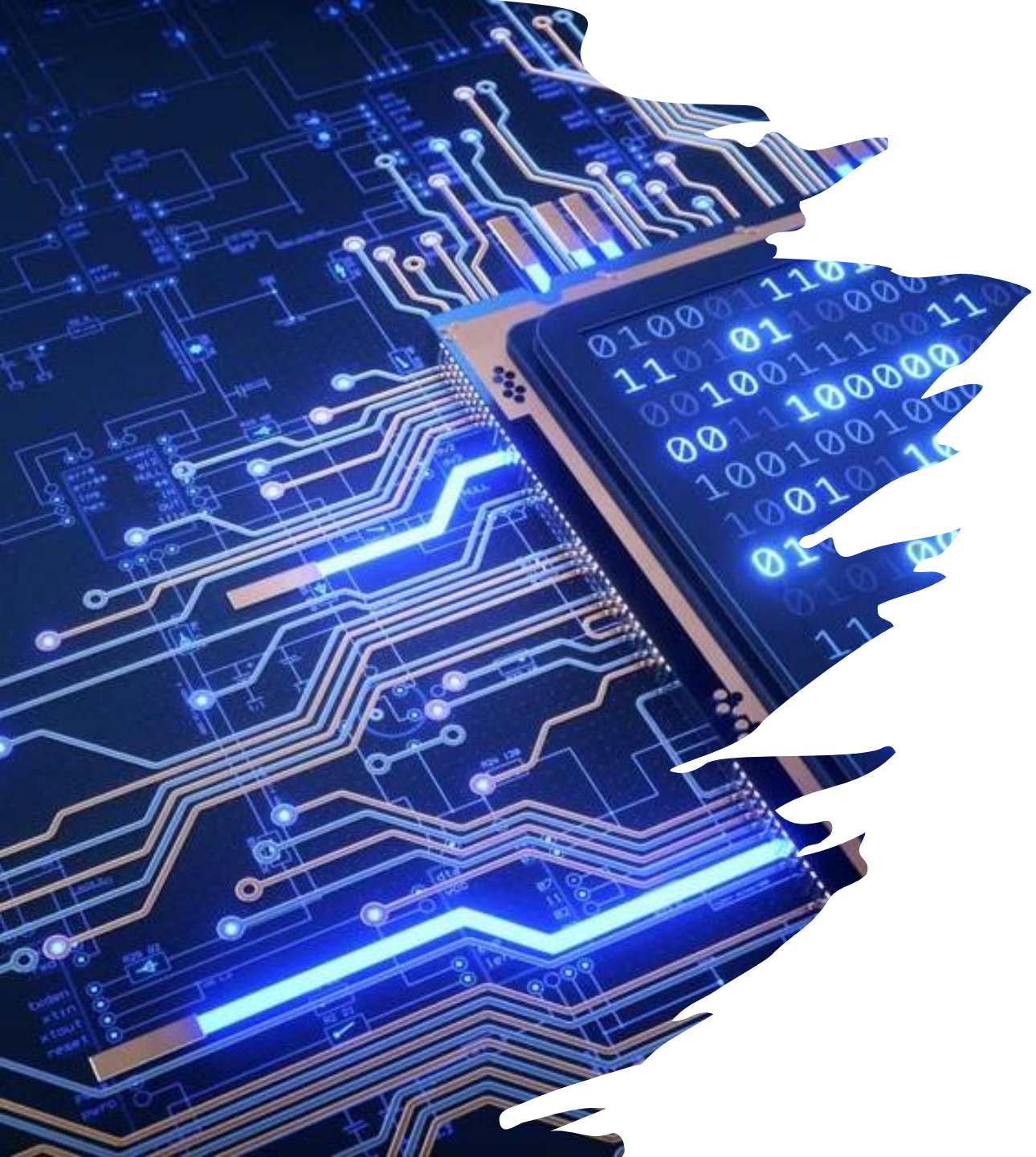
- Key to great performance!





It is important to use the correct tool for the job!

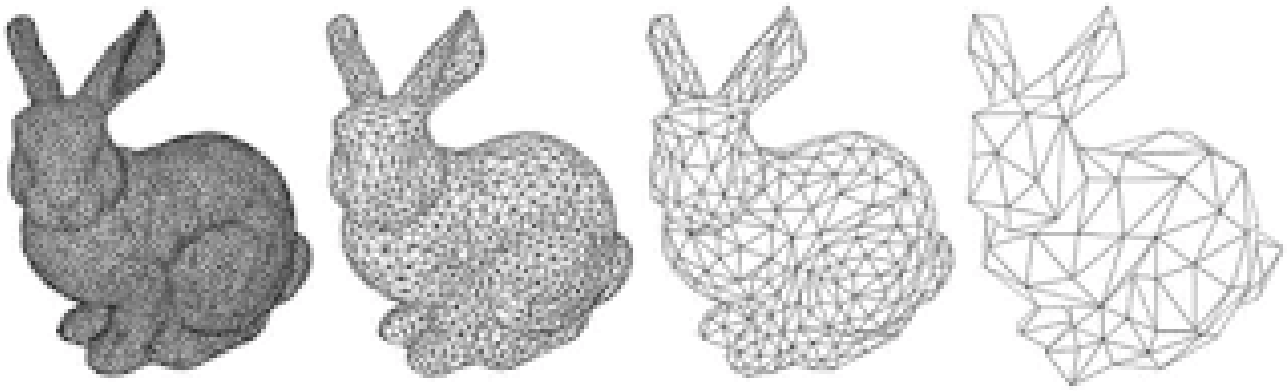

For fast performance, learn to think like a computer!



Fast code but how?

```
void foo(bool something)
{
    if (check() && something)
    {
        Console.WriteLine(“Something”);
    }
}
```

Does the order in an if-statement matter?

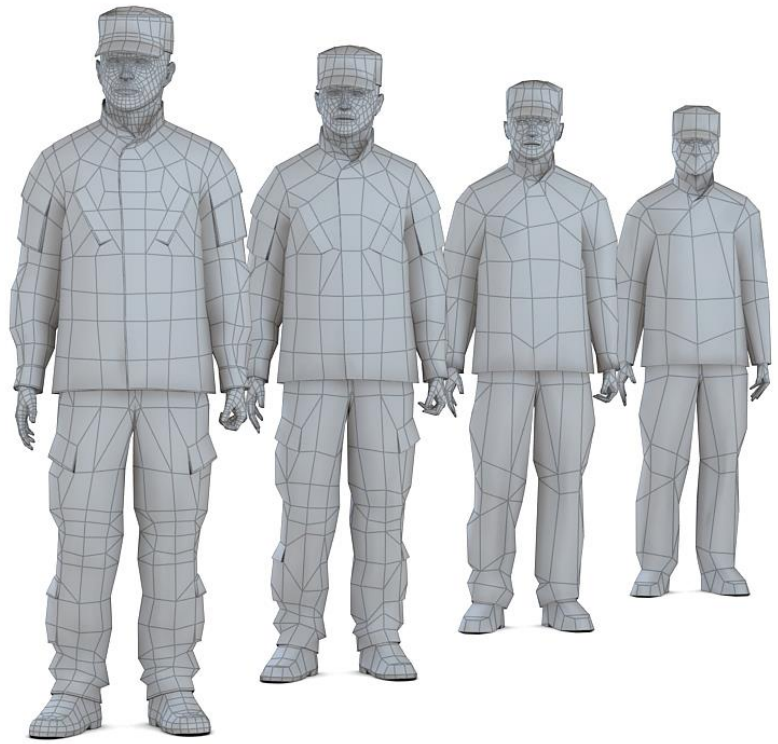


Level of Details (LOD)

WHY SHOULD I USE IT?

In computer graphics, LOD refers to the complexity of 3D models and textures

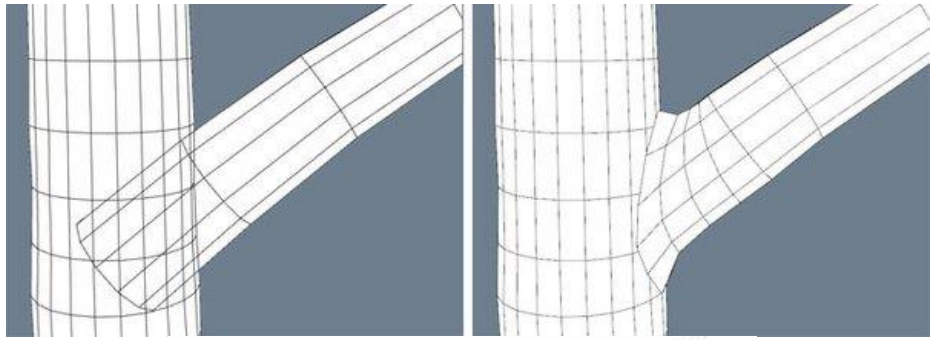
- Details are less important as they get further away from the camera
- *LOD can reduce poly-count and texture resolution for objects further away to boost performance*
- A perfect system allows a user to customize graphics settings to run on any system

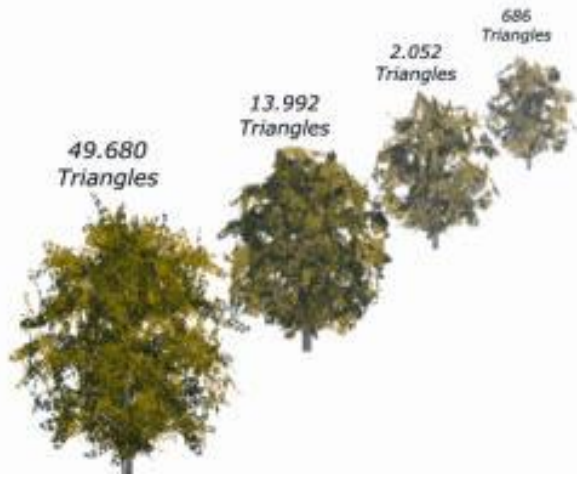


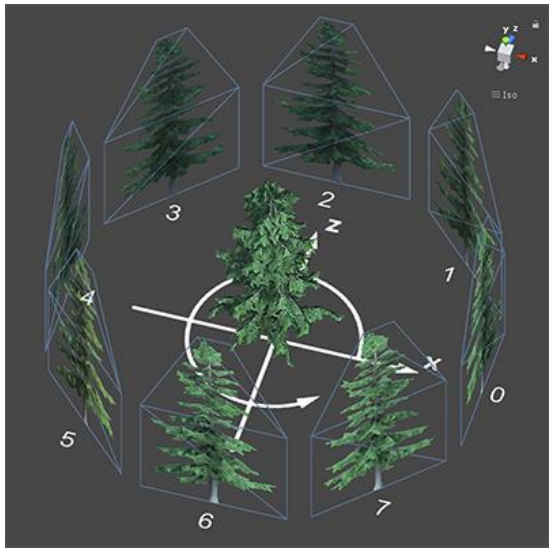
LOD & Billboards

Rendering of trees and vegetation is very performance heavy

It is a common trick to use LOD, where the lowest level of a vegetation is a rendered to a billboard

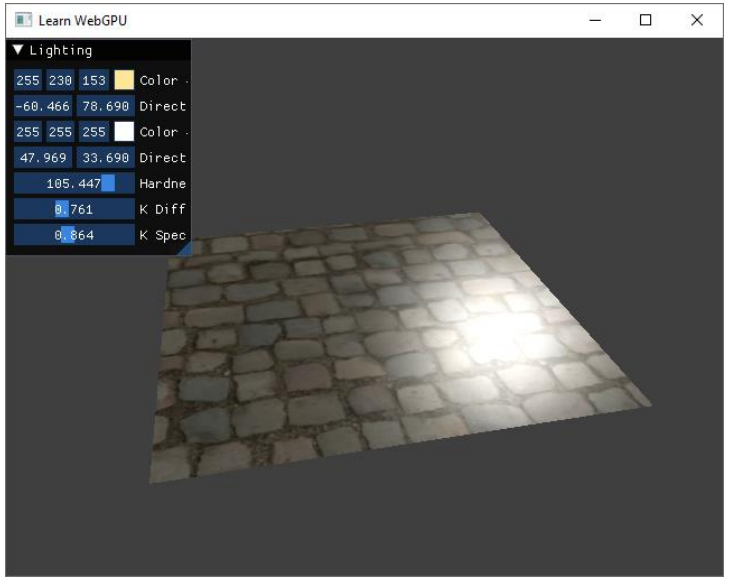


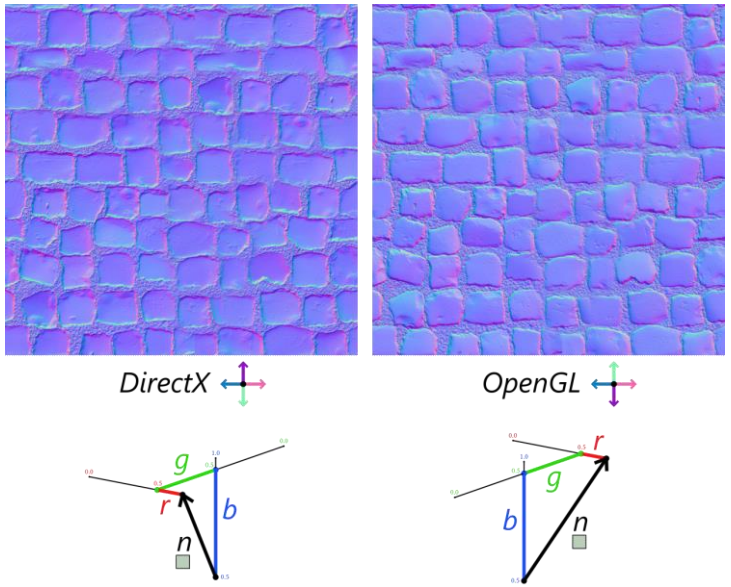


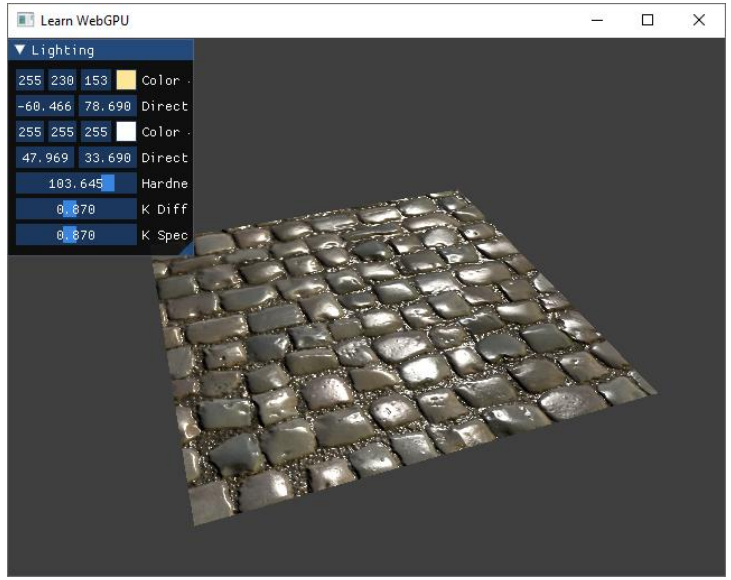


Normal Mapping

WHAT IS IT GOOD FOR?

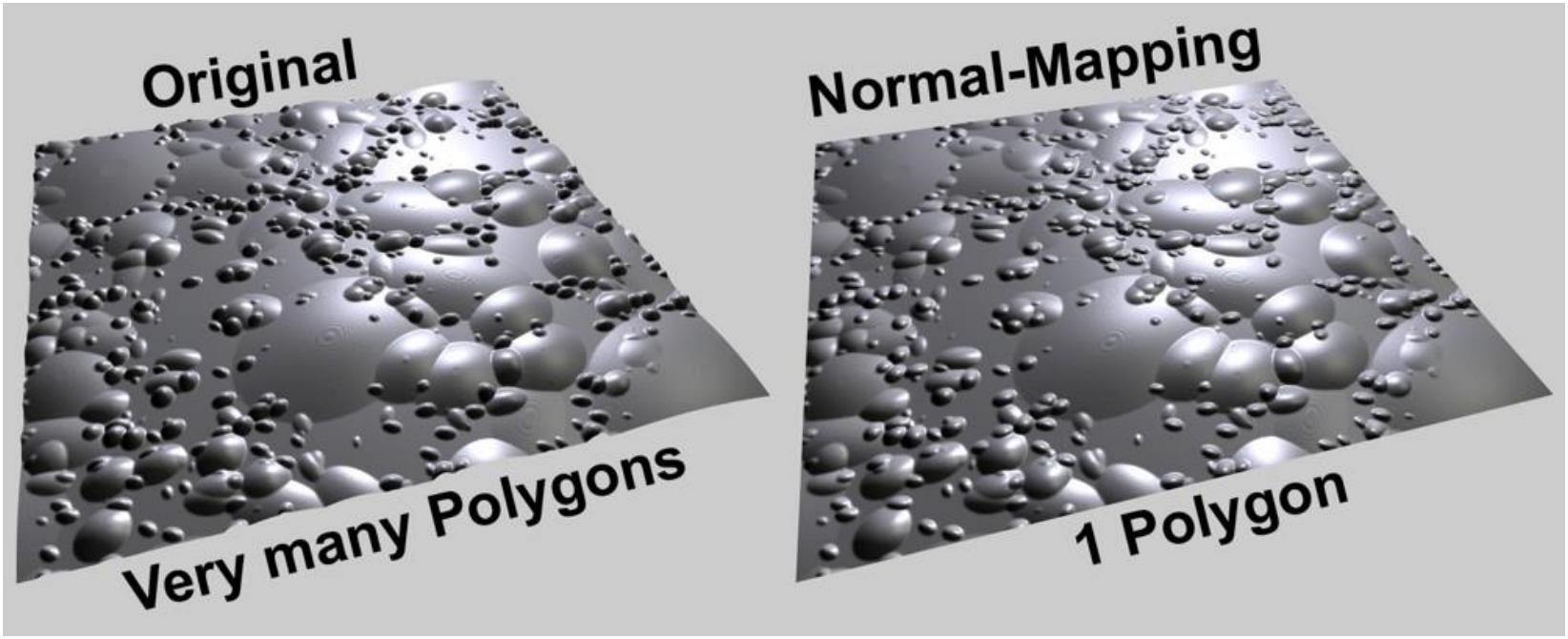






Normal Mapping

NORMAL MAPPING ADD SURFACE DETAILS
WITHOUT ADDING GEOMETRIC DETAILS



Normal Mapping

NORMAL MAPPING ADD SURFACE DETAILS
WITHOUT ADDING GEOMETRIC DETAILS

Normal Mapping

PERFECT FOR ADDING
DETAILS TO CHARACTERS

Parallax Mapping

- An extension to normal mapping
- Parallax mapping includes height
- Internal shadows can be calculated

Parallax Mapping

GREAT FOR GROUND AND WALL TEXTURES


Parallax Mapping
GREAT FOR ANY BONDED AREAS THAT NATURALLY BORDERING OTHER SURFACES

Render to texture

- Render to texture to imitate reflective surfaces
- **Make sure you do not render more than what is visible in mirror**

Render to texture

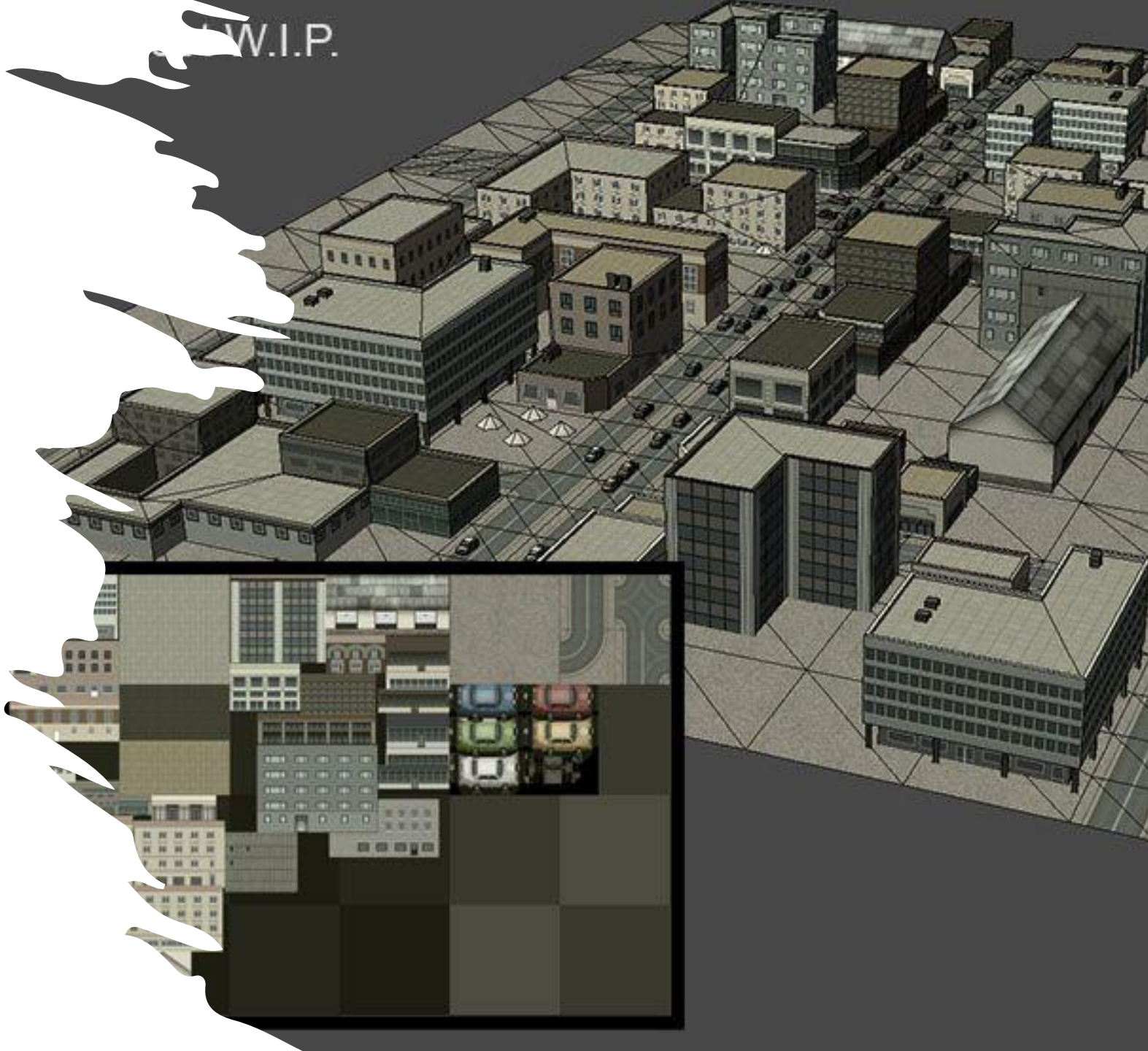
- Render to texture to imitate reflective surfaces
- **For unclear reflections like animated water, you can get away with only render very few things**

Render to 3D texture

- Rendering buildings in a city is very performance heavy
- Render rooms for buildings, that can be seen from outside, in 3D textures will reduce polygon count significantly

Atlas Textures

- A large texture that has multiple texures combined
- Why is this faster than having many small separate textures?
- What is considered a large texture?
- What does Unity consider to be a large texure?

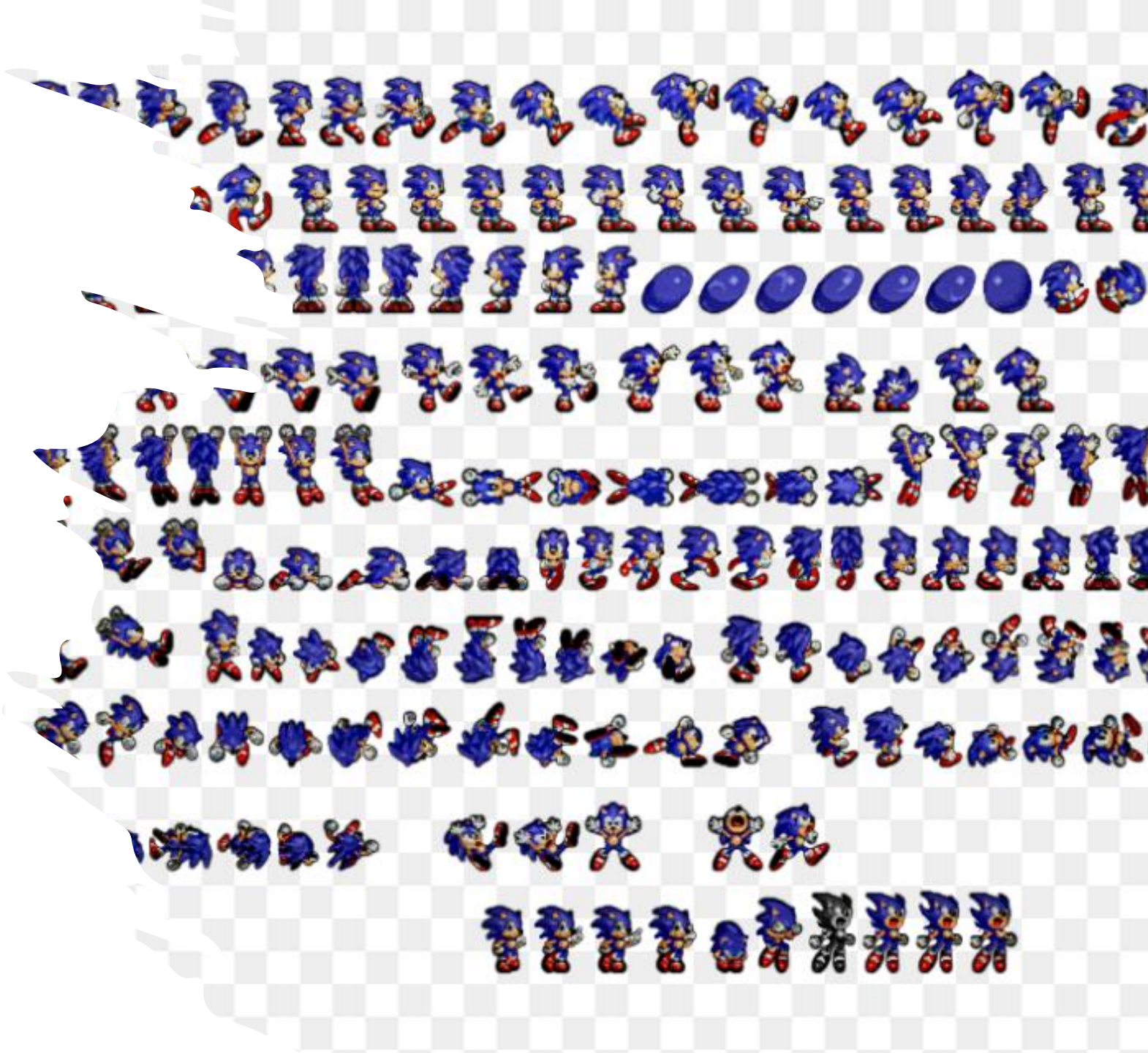


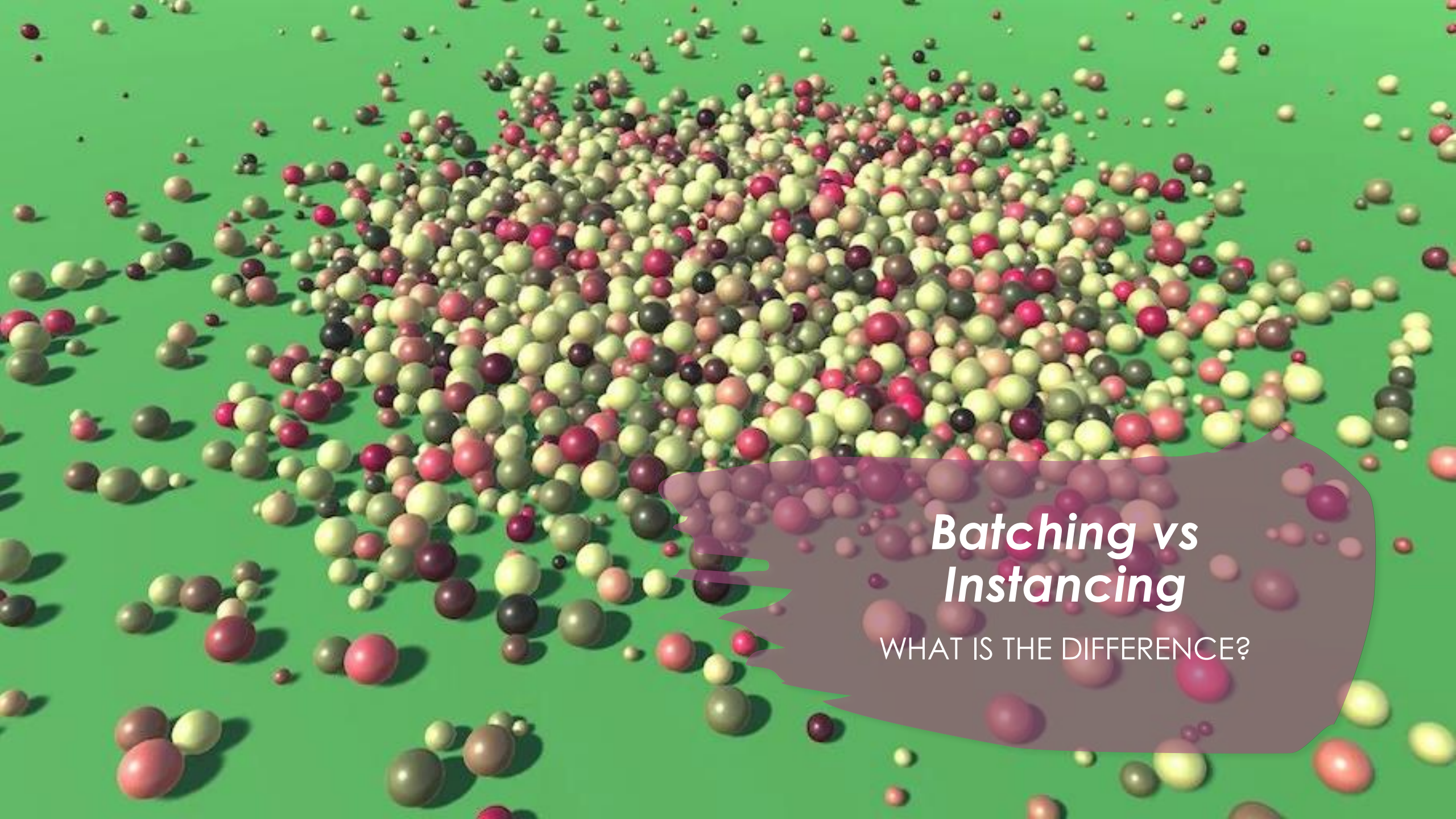
Atlas Texture
vs
Sprite Sheet

WHAT IS THE DIFFERENCE?

Sprite Atlas

- A Sprite Atlas is an Asset that consolidates several Textures into a single combined Texture
- Unity can call this single Texture to issue a single draw call instead of multiple draw calls to access the packed Textures all at once at a smaller performance overhead
- In addition, the Sprite Atlas API provides you with control over how to load the Sprite Atlases at your Project's run time





Batching vs Instancing

WHAT IS THE DIFFERENCE?

What do they have in common?

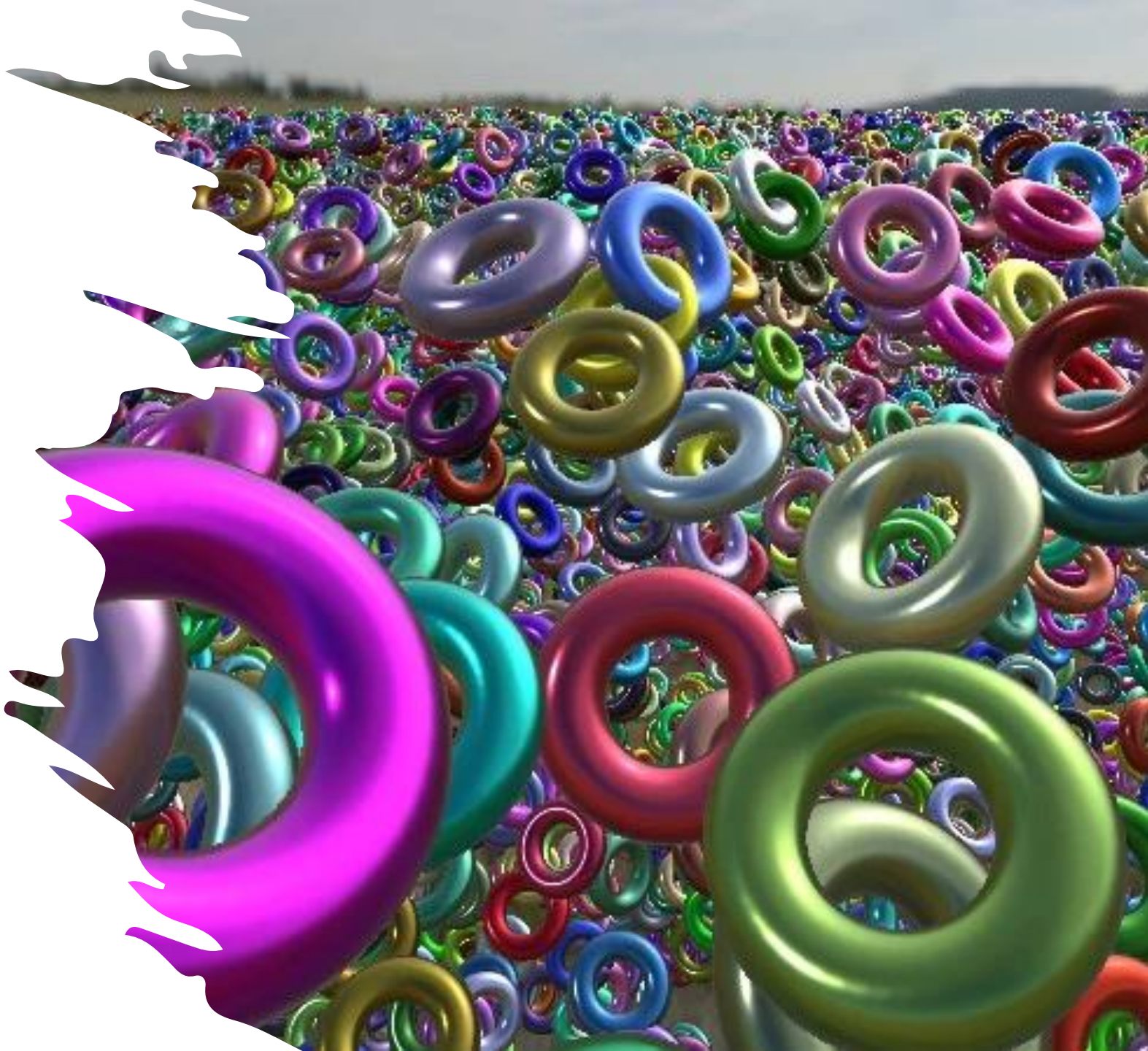
- Allow more objects to be drawn on screen
- Reducing the rendering load for the graphics card
- Less draw calls!

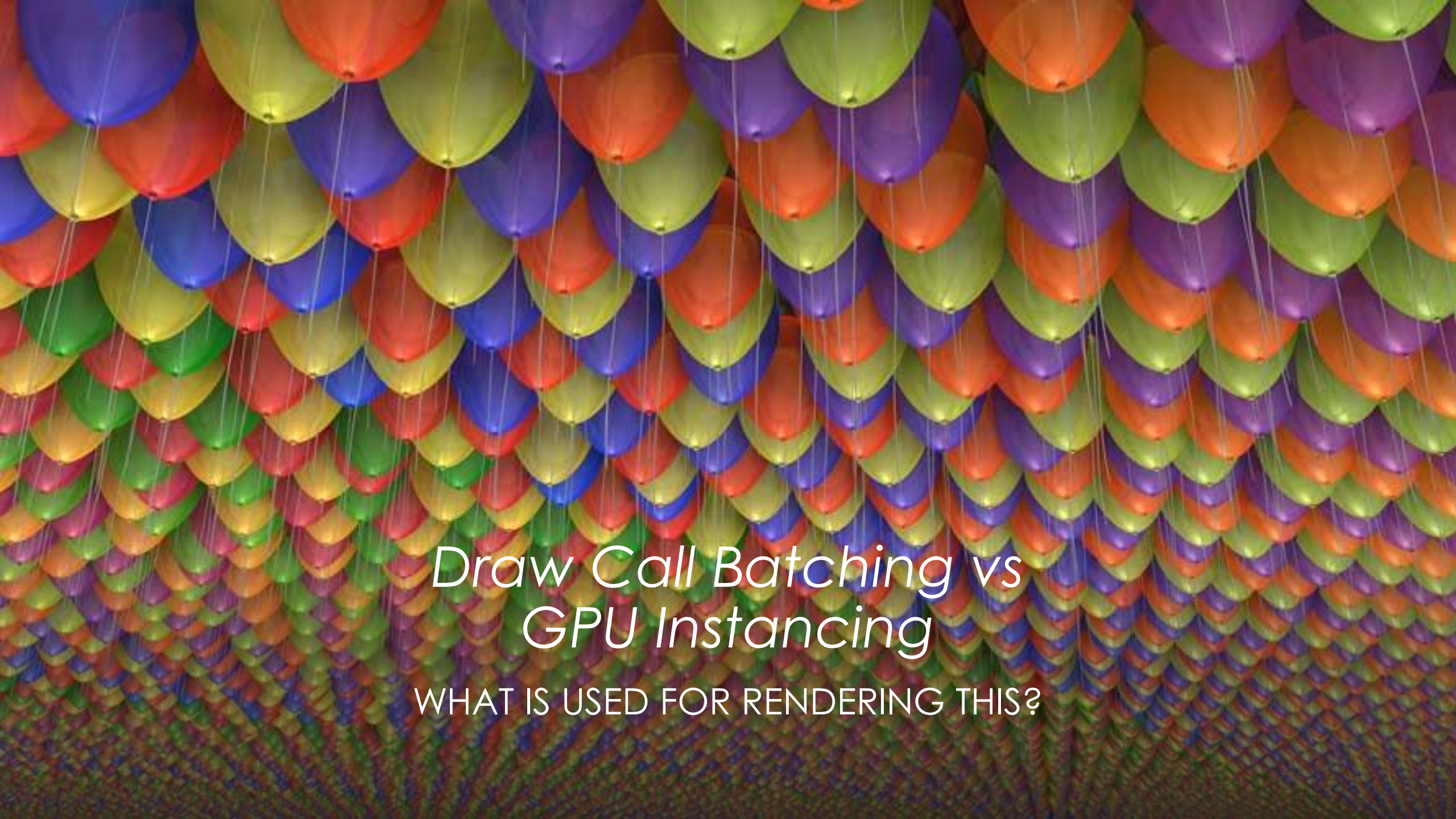
Draw Call Batching in Unity

- Batching means to store multiple assets into one GPU buffer
- All geometry properties are merged into buffers and sent to the graphics card
- A buffer can max contain 65.536 vertices
- All assets in a batch must share same material, but using atlas texture can add variety
- Skinned mesh can't be batched, due to CPU transformation processing
- Unity only batches objects of the same type

Draw Call GPU Instancing

- Instancing means to instance a GPU buffer multiple times in one draw call
- An instance can be transformed by offsets to variety to location, rotation and scale
- An instance can max contain 65.536 vertices
- All assets must share same materials but using atlas texture can add variety
- Skinned mesh is not a problem



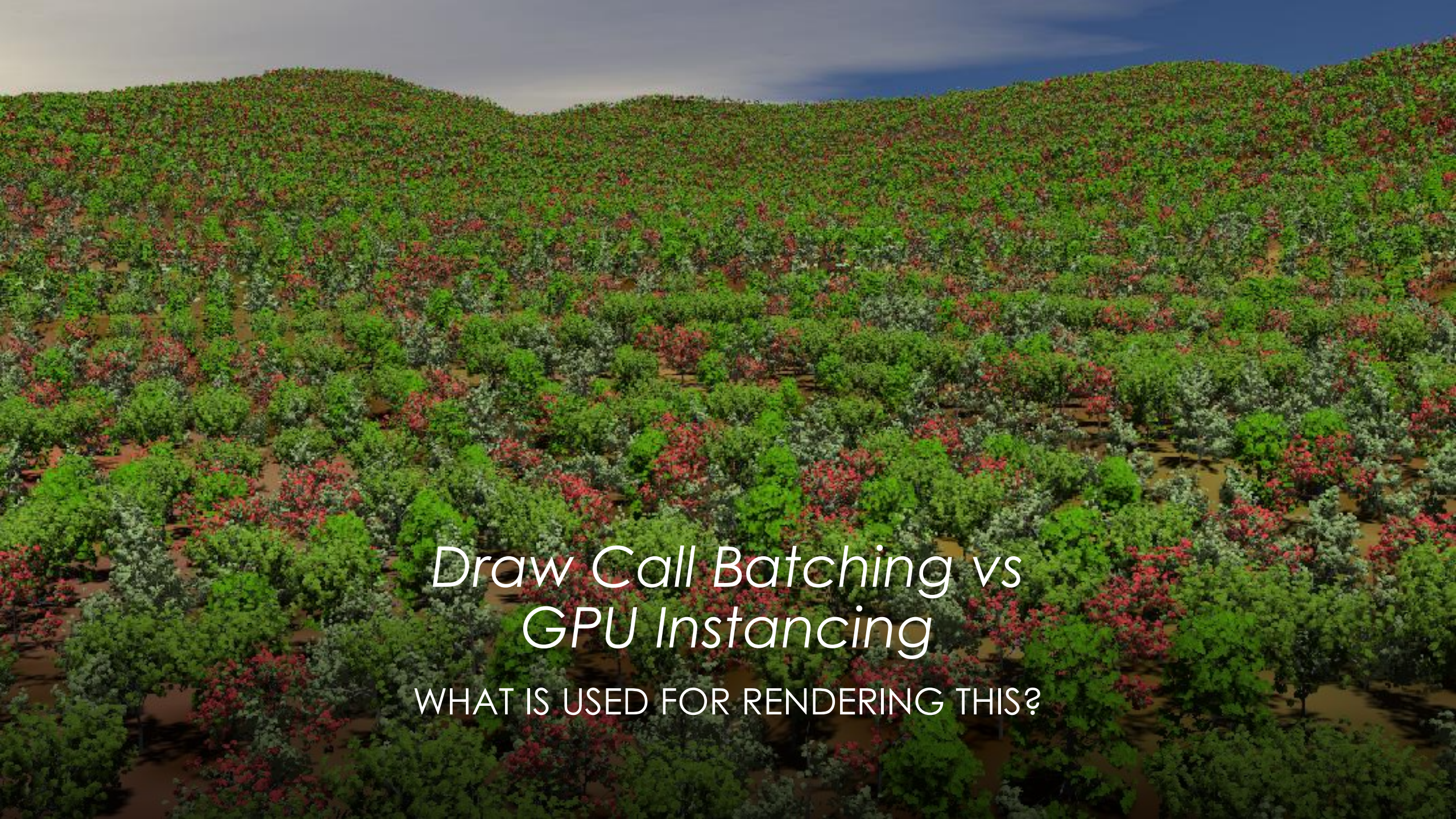


Draw Call Batching vs GPU Instancing

WHAT IS USED FOR RENDERING THIS?

Draw Call Batching vs GPU Instancing

WHAT IS USED FOR RENDERING THIS?


Draw Call Batching vs
GPU Instancing
WHAT IS USED FOR RENDERING THIS?

Draw Call Batching vs GPU Instancing

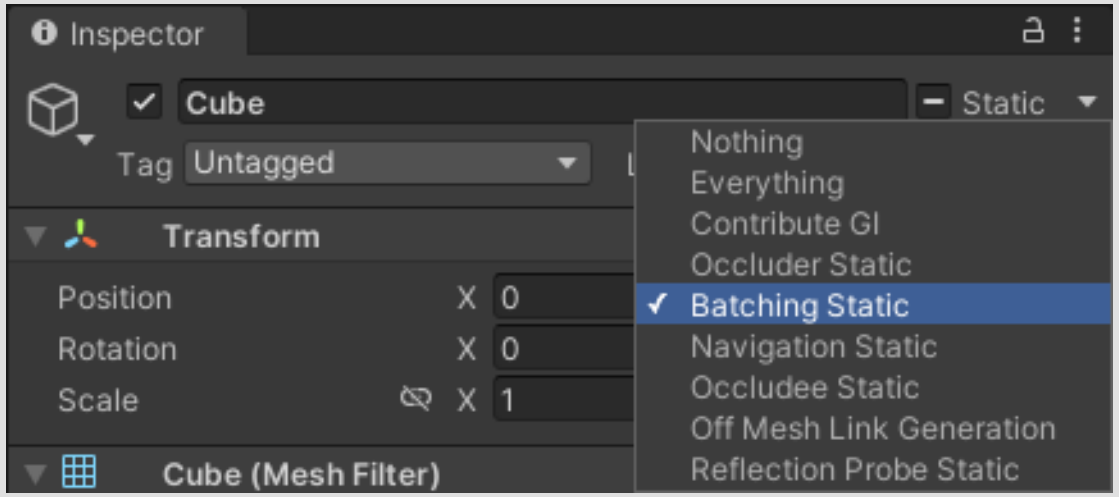
WHAT IS USED FOR RENDERING THIS?

Draw Call Batching vs GPU Instancing

WHAT IS USED FOR RENDERING THIS?

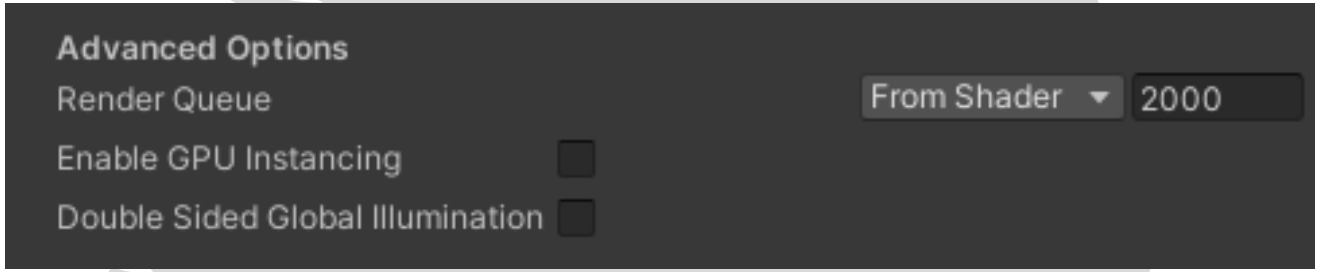
Batching Static Meshes in Unity

- Unity automatically batches the specified static meshes into the same draw call if they fulfill the criteria described in the common usage information



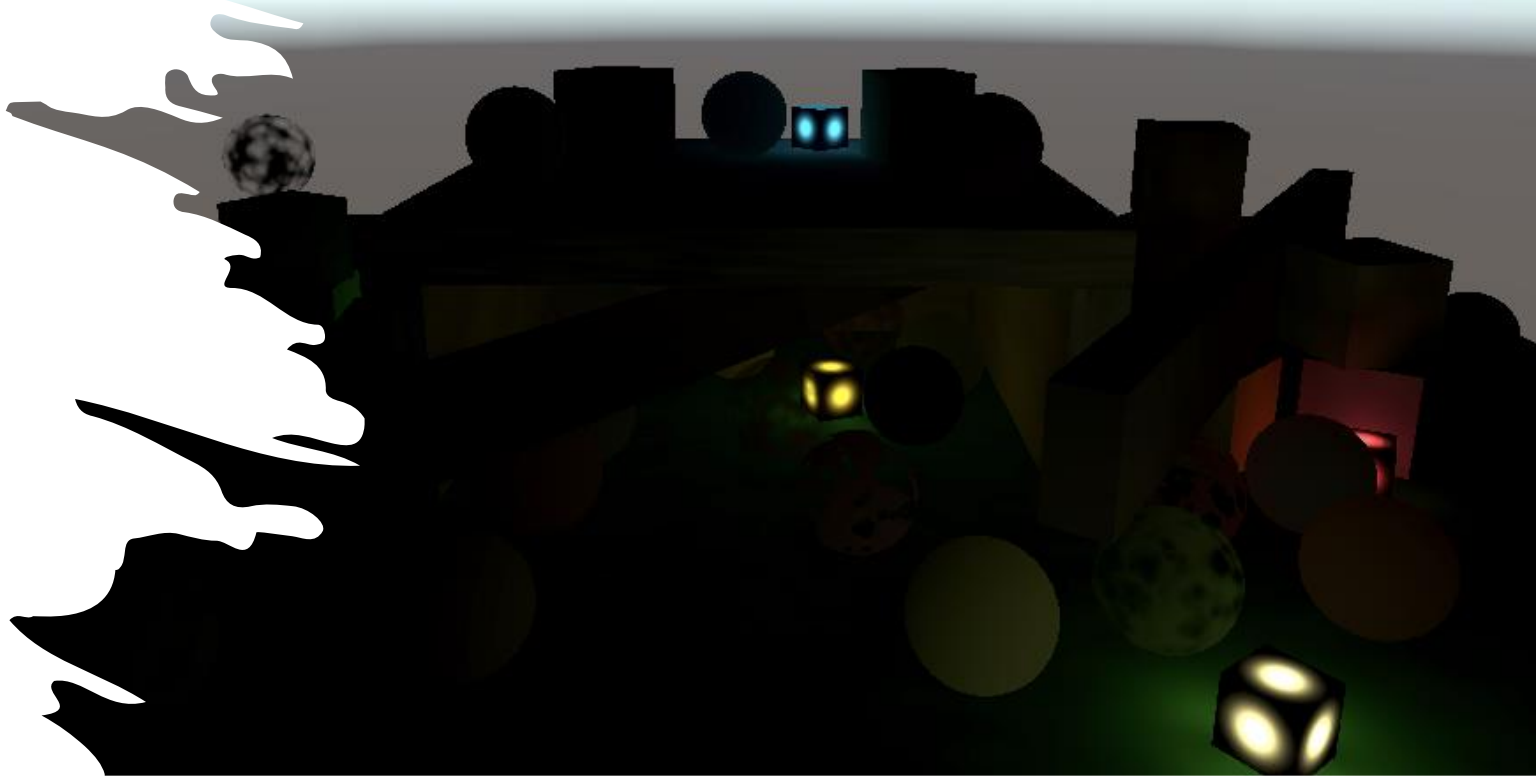
GPU Instancing in Unity

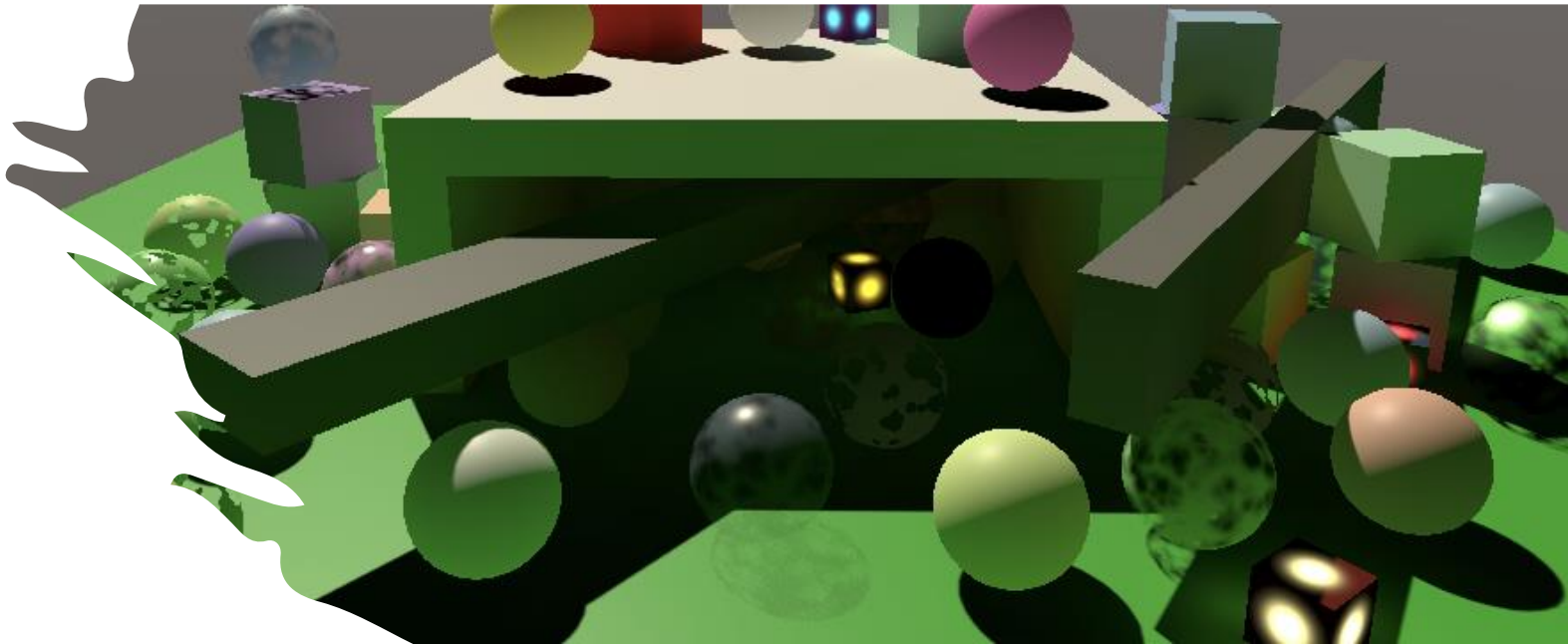
- To use GPU instancing for a material, select the Enable GPU Instancing option in the Inspector
- GPU instancing supports Unity's Baked Global Illumination system
- Unity Standard Shaders and surface shaders support GPU instancing and Unity's Baked Global Illumination system by default



Light Maps & Probes

- Bake lighting when possible to boost performance
- Baked lighting can be mixed with real-time lighting



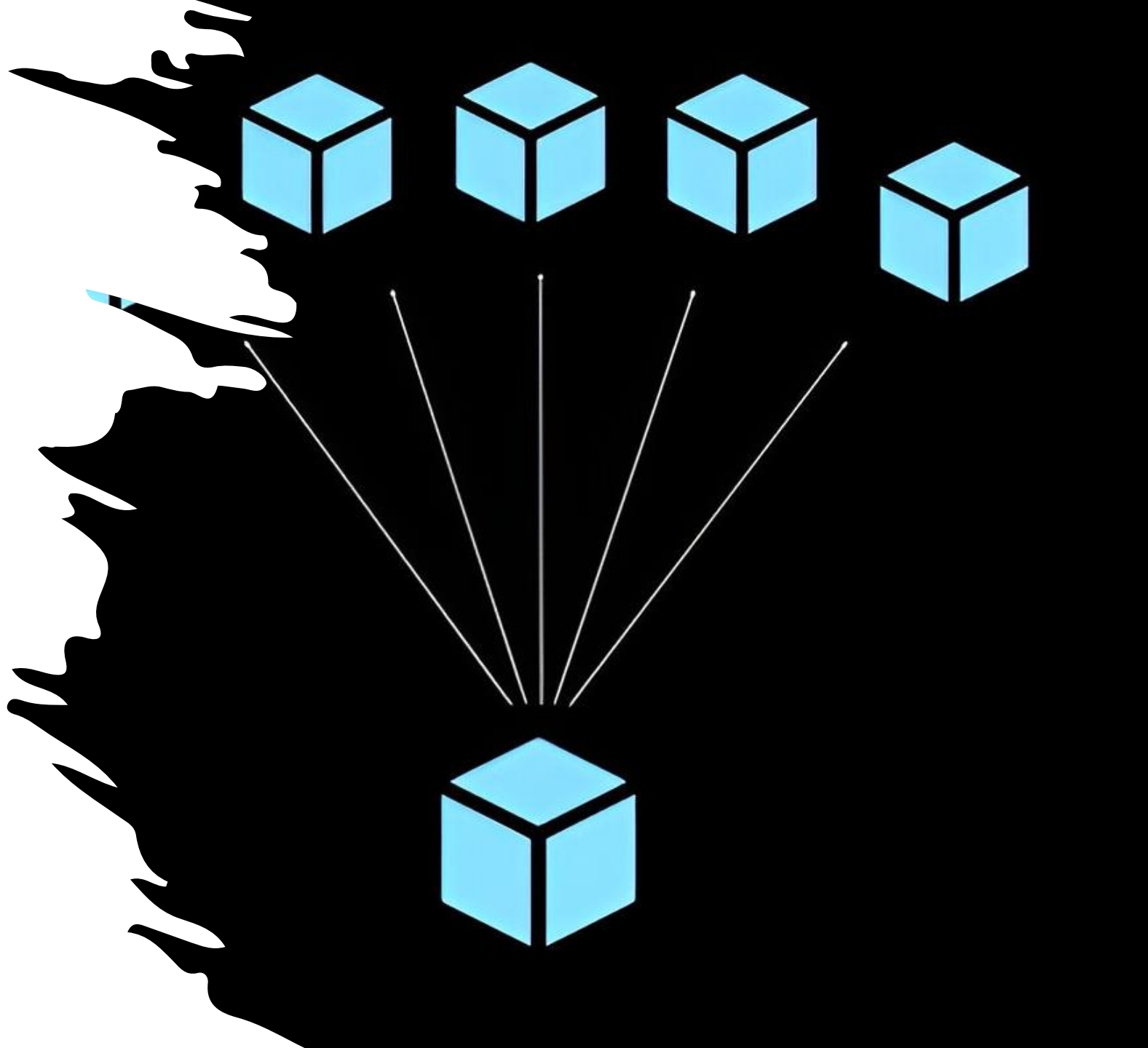


Occlusion Culling

- Only render objects within field of view by splitting up your game world into sections
- Rendering objects that the player cannot see would be a waste of resources

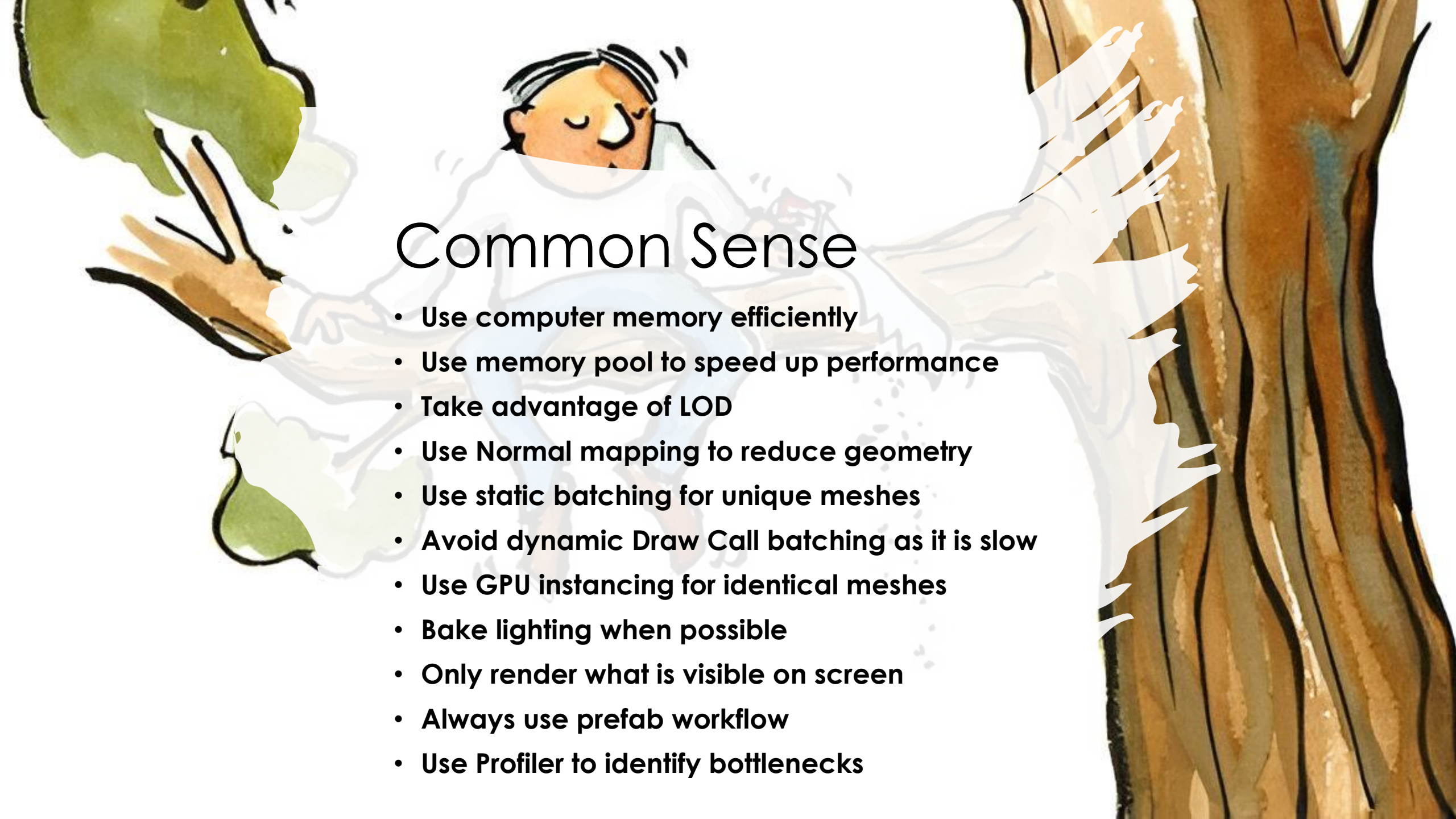
Prefab Workflow

- How many are using prefabs?
- Generally speaking, always use prefab workflow!
- Does Prefab Workflow improve performance?
- Loading time?



Unity Profiler

USE PROFILER TO IDENTIFY PERFORMANCE BOTTLENECKS



Common Sense

- Use computer memory efficiently
- Use memory pool to speed up performance
- Take advantage of LOD
- Use Normal mapping to reduce geometry
- Use static batching for unique meshes
- Avoid dynamic Draw Call batching as it is slow
- Use GPU instancing for identical meshes
- Bake lighting when possible
- Only render what is visible on screen
- Always use prefab workflow
- Use Profiler to identify bottlenecks