# Fachprojekt DET WS 2019/2020
# - Unity DOTS -









Marco Pleines | Dortmund, 08.10.2019

# Data Oriented Technology Stack

Marco Pleines | Dortmund, 08.10.2019

# Data Oriented Technology Stack

~~objektorientiert~~ -> datenorientiert

- Entity Component System
  - Entitäten verfügen über Komponenten mit Daten, System verarbeitet Daten
- **Job System**
  - Multithreaded Code
- Burst Compiler
  - Code wird stark optimiert kompiliert (in Bezug auf Jobs und ECS)

# Struct vs Class

- Structs (value type) arbeiten mit Werten und nicht mit Referenzen (Klasse = reference Type)
  - Daten werden bei Structs stets kopiert!
- Reference types können null sein, denn diese zeigen im wesentlichen nur auf ein Objekt
- Value types können nicht null sein, da in dem Adressraum keine Referenzen sondern Werte gefunden werde

# Ein Beispiel Job

```
public struct ToughJob: IJob
{
    public void Execute()
    {
        float value = 0;
        for (int i  = 0; i < 50000; i++)
        {
            value = math.exp10(math.sqrt(value));
        }
    }
}
```

[1]

# Beispiele die nicht gehen

```
public struct SomeJob : IJob
{
    public float pos;
    public void Execute()
    {
        GameObject newGo = new GameObject("Job GO");
        newGo.transform.position = new Vector3(pos, pos, pos);
    }
}
```

UnityException: Internal_CreateGameObject can only be called from the main thread.

# Beispiele die nicht gehen

```
public struct SomeJob : IJob
{
    public GameObject go;
    public float pos;
    public void Execute()
    {
        go.transform.position = new Vector3(pos, pos, pos);
    }
}
```

InvalidOperationException: SomeJob.go is not a value type. Job structs may not contain any reference types.

# Was darf also ein Job machen?

- Explizit Daten (value types) verarbeiten

- z.B. Mesh berechnen
- Das Mesh wird aber erst am Ende des Jobs im Main Thread wieder verwertet

```
public struct SomeJob : IJob
{
    public float x;
    public float y;
    public float result;

    public void Execute()
    {
        result = Mathf.PerlinNoise(x, y);
    }
}
```

# NativeContainer

- NativeArray
- NativeList
- NativeHashMap
- NativeQueue

Vorsicht: Nur eindimensionale Arrays!

```
NativeArray<int> array = new NativeArray<int>(20, Allocator.Temp);
```

# NativeContainer

- Workaround: Jagged Arrays

```
float[][][] jaggedArray = new float[3][][];

float[,,] multiDimArray = new float[3, 3, 3];
```

- Workaround: Flattened Array
  - z.B. mittels LINQ: SelectMany()

# NativeContainer

- Implementiert das Safety System, was im wesentlichen Race Conditions verhindern soll [0]
- Verwendung nur von “blittable types”

```
NativeArray<float3> result = new NativeArray<float3>(9, Allocator.Persistent);
SomeJob job = new SomeJob();
job.result = result;
JobHandle handle = job.Schedule();
handle.Complete();
result.Dispose();
```

# Job Types

IJob

IJobParallelFor

IJobParallelForTransform

# Using Statements

- using Unity.Jobs;
- using Unity.Mathematics;

  (Muss via PackageManager installiert werden)
- using Unity.Collections;
- using UnityEngine.Jobs;

# References

[0] Unity Manual

[1] Getting Started with the Job System in Unity 2019

[2] Various Noise Functions