Multiplayer Applications and Games Project on Unity DOTS Architecture

CANDIDATE:

MICHELE RIGHI

SUPERVISOR:

DOTT. PAOLO BELLAVISTA

CO-SUPERVISOR:

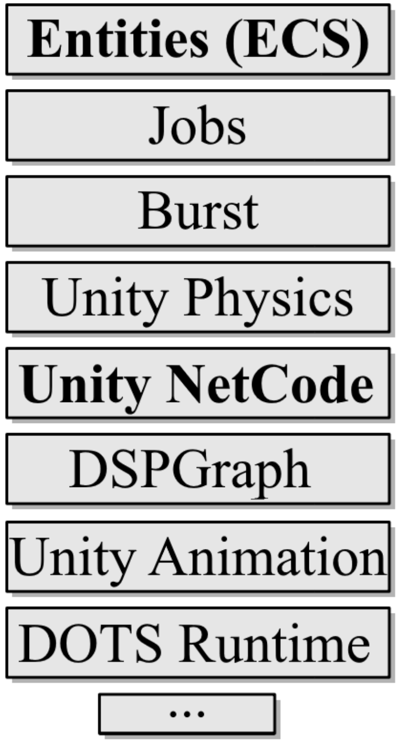
DOTT. ANDREA GARBUGLI

Introduction

- In 2018 Unity started **rebuilding** the core of its game engine with the Data-Oriented Technology Stack.

- **Traditional architecture** limited by the **component** model (GameObject and MonoBehaviour).
  - Class **overhead** (GameObjects and MonoBehaviours are classes).
  - **Data scattered** in memory, due to references.
  - CPU **multiple cores not used**.

- Solution: Unity **Data-Oriented Technology Stack** (DOTS).



Entities (ECS)

Jobs

Burst

Unity Physics

Unity NetCode

DSPGraph

Unity Animation

DOTS Runtime

...

Unity: main DOTS packages.

Main Topics

Entity Component System (ECS)

NetCode

Prototypes

Experimental Results

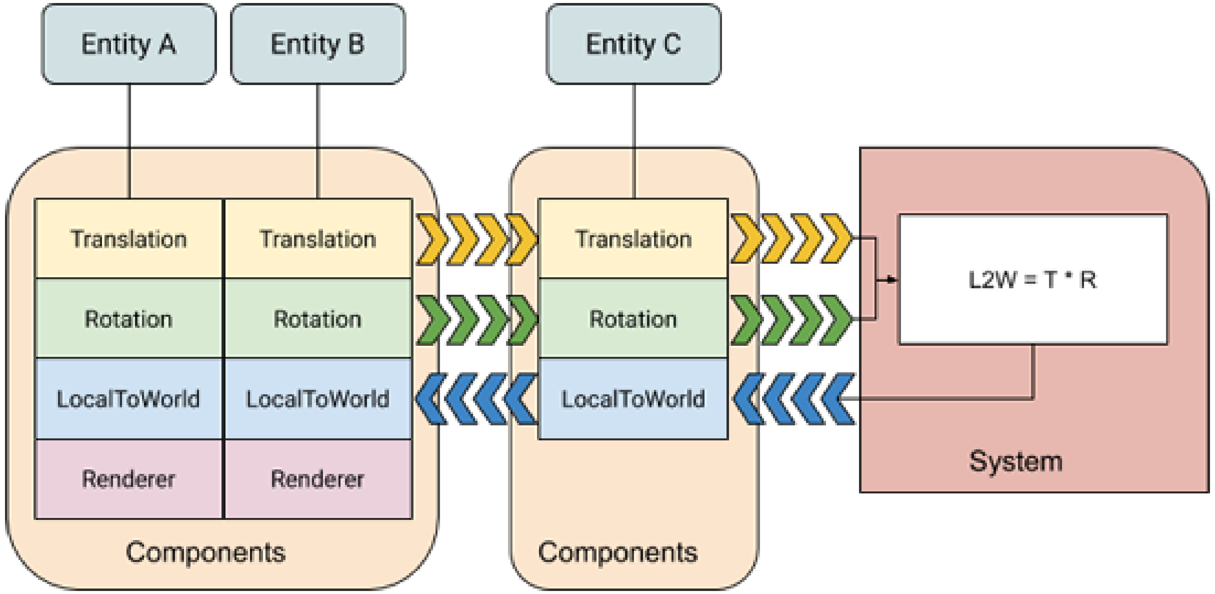
Conclusions

Entity Component System (ECS)

- **Entities**. The concrete «**things**» that populate the game at runtime. They are comparable to the **keys** (numeric IDs) of a database.

- **Components**. The **data** associated with entities. They store the state but don't contain any kind of logic. They are comparable to the **tuples** of a database.

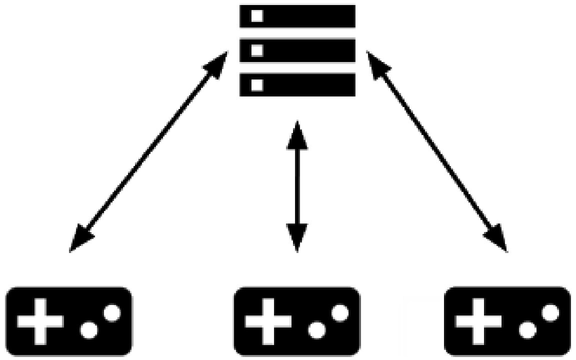
- **Systems**. Allow to implement the **logic** that transform the compontent data from its current state to its next one. They are comparable to the **queries** of a database.



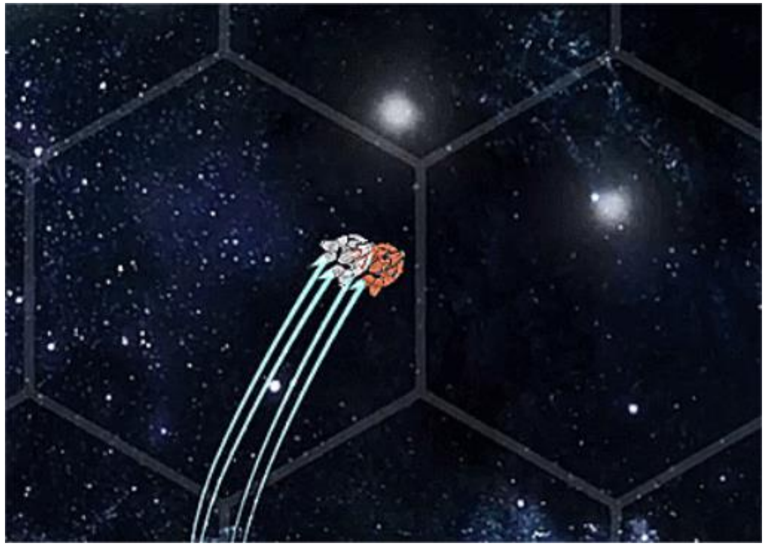
ECS example.

NetCode

- Network topology based on a client / server model with **authoritative server**.
- Reduction of latency through the use of **client-side prediction**.



Authoritative server model.

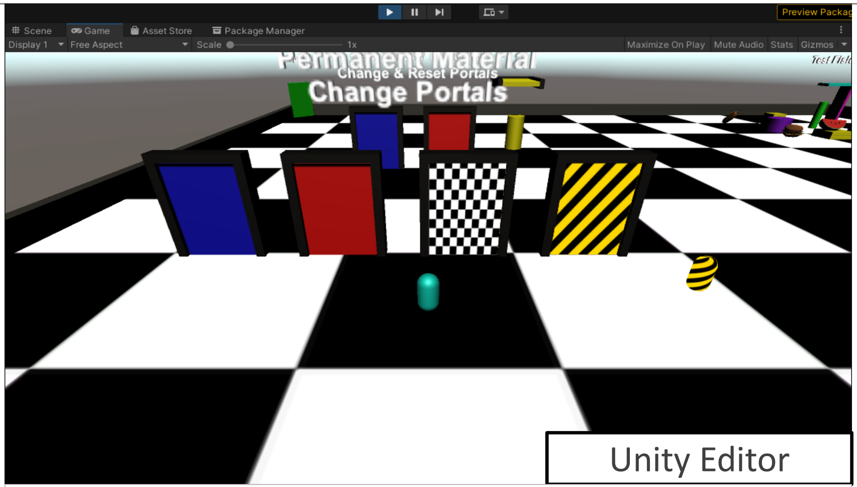


Client-side prediction example (white ship).

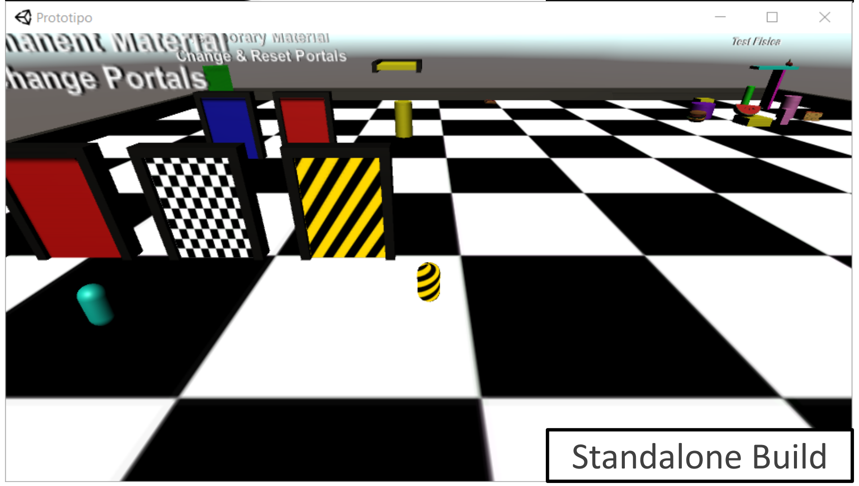
Prototypes

PROTOTYPE 1: COMPLETE VIDEOGAME

- Application based on **ECS** architecture, provided by the Entities package.
- Networking implemented through the use of the **NetCode** package.
- Physical simulation realized using the **Physics** package.
- Gameplay features: **change-color portals**, **teleports**, **collectibles pick up**.



Unity Editor



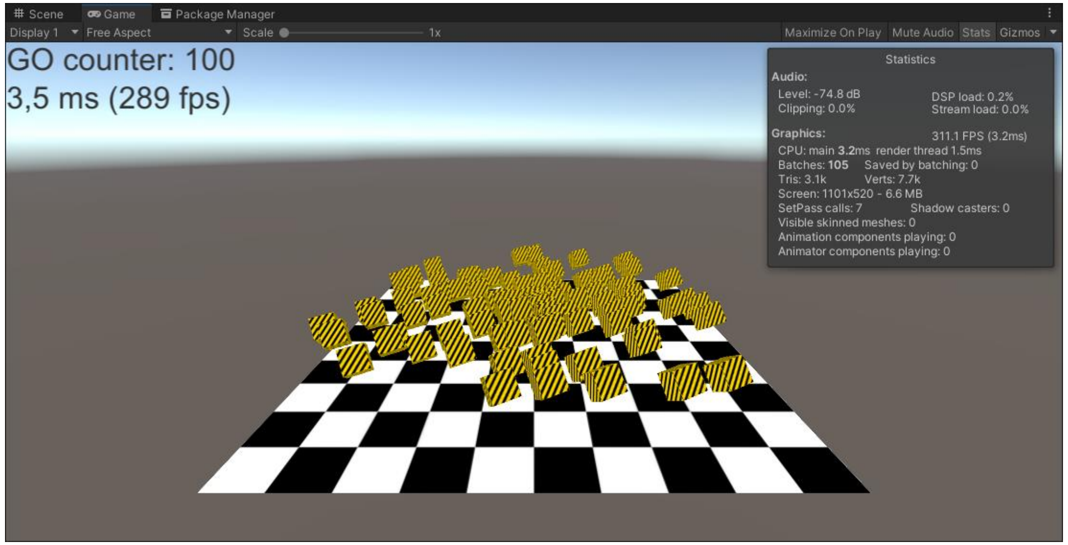
Standalone Build

Prototypes

PROTOTYPE 2: STRESS TEST

- Small application that generates an arbitrary number of cubes which are then rotated.
- Rotation of the cubes implemented in different ways:

1. GameObject + MonoBehaviour (traditional architecture).
2. ECS «vanilla».
3. ECS + Jobs.
4. ECS + Parallel Jobs.
5. ECS + Parallel Jobs + Burst.



```
GO counter: 100
3,5 ms (289 fps)
```

```
Statistics
Audio:
  Level: -74.8 dB                DSP load: 0.2%
  Clipping: 0.0%                 Stream load: 0.0%
Graphics:                        311.1 FPS (3.2ms)
  CPU: main 3.2ms  render thread 1.5ms
  Batches: 105    Saved by batching: 0
  Tris: 3.1k      Verts: 7.7k
  Screen: 1101x520 - 6.6 MB
  SetPass calls: 7            Shadow casters: 0
  Visible skinned meshes: 0
  Animation components playing: 0
  Animator components playing: 0
```

Stress Test Prototype (100 cubes).

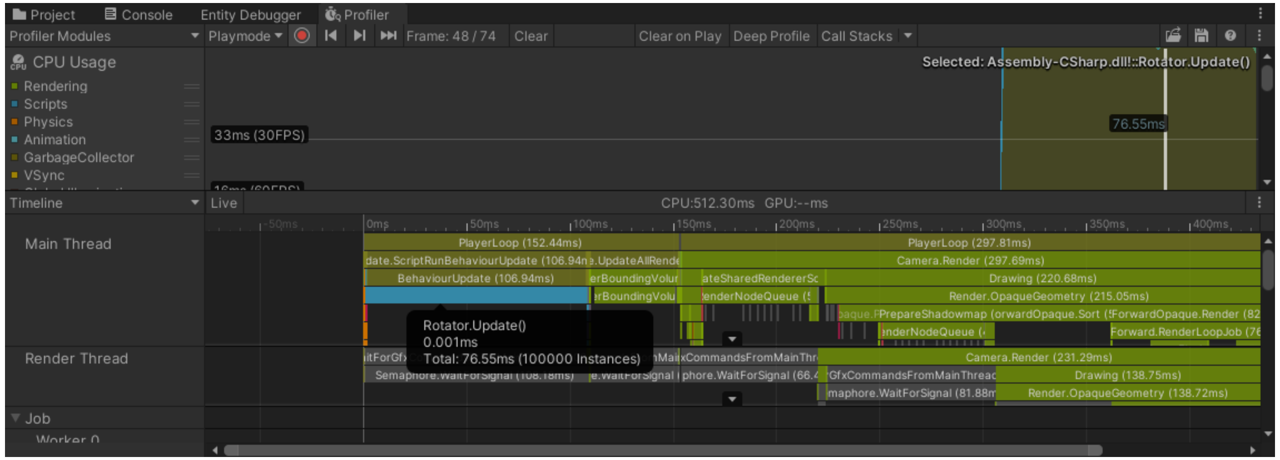
Experimental Results

- The development of the prototypes and the related tests were performed on the following **architecture**:
  - Intel® Core^{TM} i7-7700HQ processor with 4 cores (8 logic processors) and 2.80GHz clock frequency.
  - 16GB RAM.
  - NVIDIA GeForce GTX 1060 video card.
  - Microsoft Windows 10 Home (x64) Operating System.
- The tests were performed, for each solution, on 10, 100, 1.000, 10.000, 100.000 and 1.000.000 cubes.

Experimental Results

- Rotation of 100,000 GameObjects, via MonoBehavior:

FPS	~2,5
Rotator ms	~77
Total CPU ms	~430

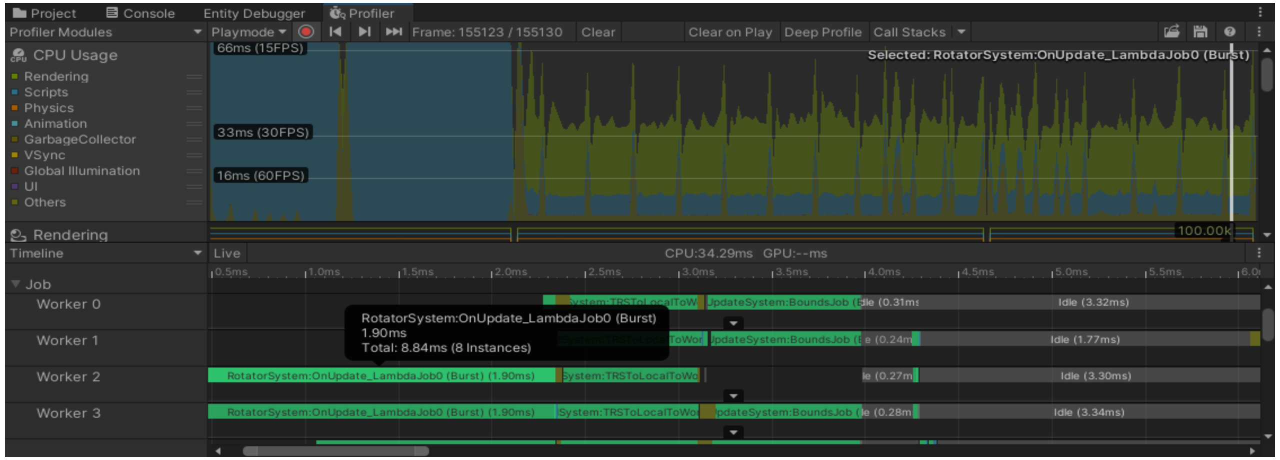


Selected: Assembly-CSharp.dll!::Rotator.Update()

Rotator.Update()
0.001ms
Total: 76.55ms (100000 Instances)

- Rotation of 100,000 Entities Using ECS + Jobs + Burst:

FPS	~25,2
RotatorSys ms	~8,84
Total CPU ms	~39,8

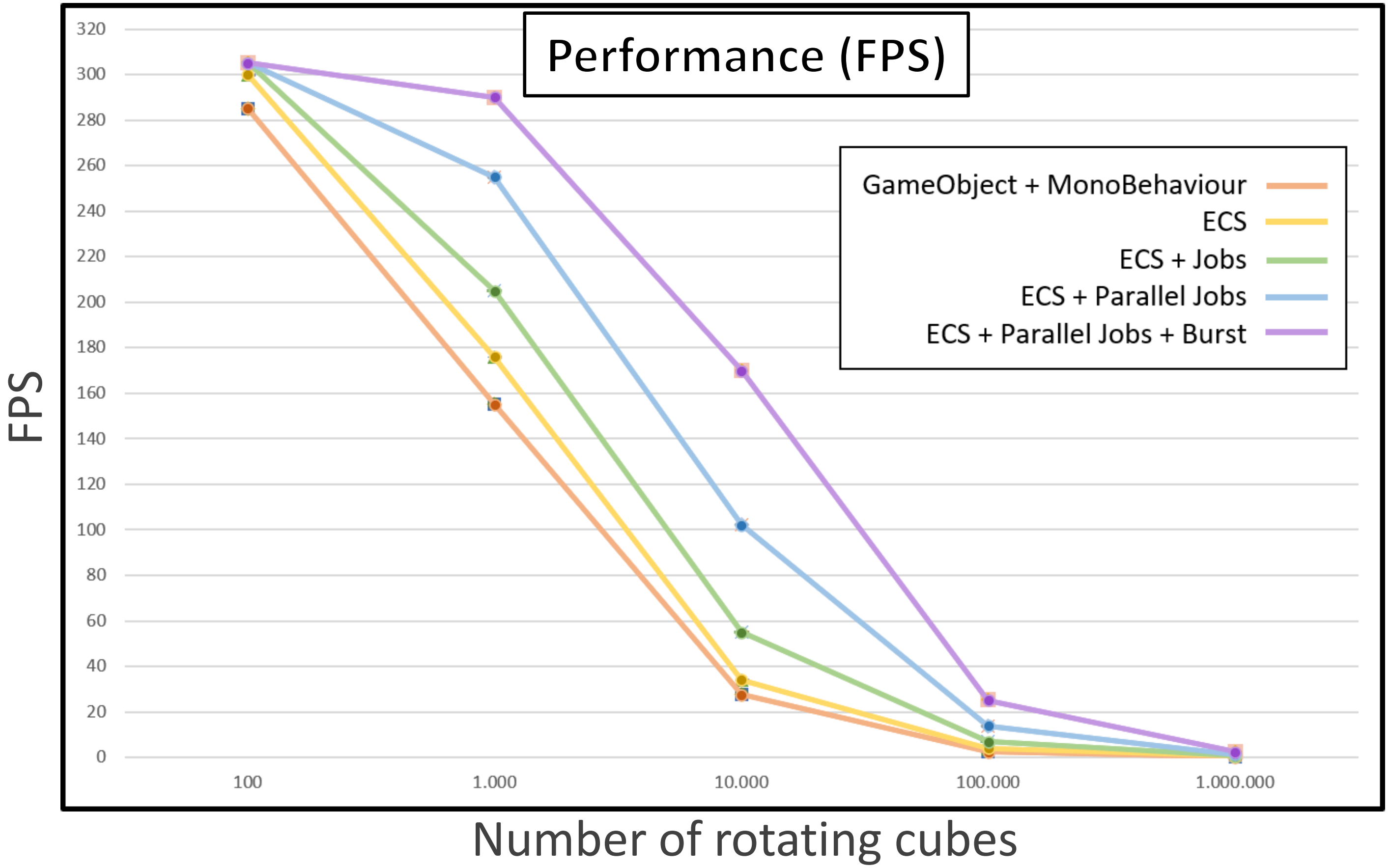


Selected: RotatorSystem:OnUpdate_LambdaJob0 (Burst)

RotatorSystem:OnUpdate_LambdaJob0 (Burst)
1.90ms
Total: 8.84ms (8 Instances)

Experimental Results

- The graph shows the **difference in FPS** between the various implementations.
- ECS and the use of DOTS packages lead to an **improvement in performance**, compared to the traditional architecture based on GameObject.



Conclusions

PROS

- **Separation** of the data and behaviour **logic**.
- Highly **readable** and **reusable code**.
- **Maximized** use of resources, especially **CPU** and **cache**, thanks to data layout.
- Reduction of consumption (longer battery life).
- Network model with **minimal latency**.

CONS

- Still under development.
- Most packages are still in preview, therefore they will be subject to possible changes.
- Some or parts of the features present in the traditional architecture not yet supported.

Conclusions

DOTS FUTURE DEVELOPMENTS

- Official DOTS release.
- Extension of conversion support.
- Reduction of the code needed to implement the NetCode execution flow.
- Addition of further useful interfaces for analysis.

PROTOTYPE FUTURE DEVELOPMENTS

- Pre-match lobby.
- Scoreboard.
- Inventory system.
- In-depth re-assessment of network latency.

Unity DOTS

PERFORMANCE BY DEFAULT