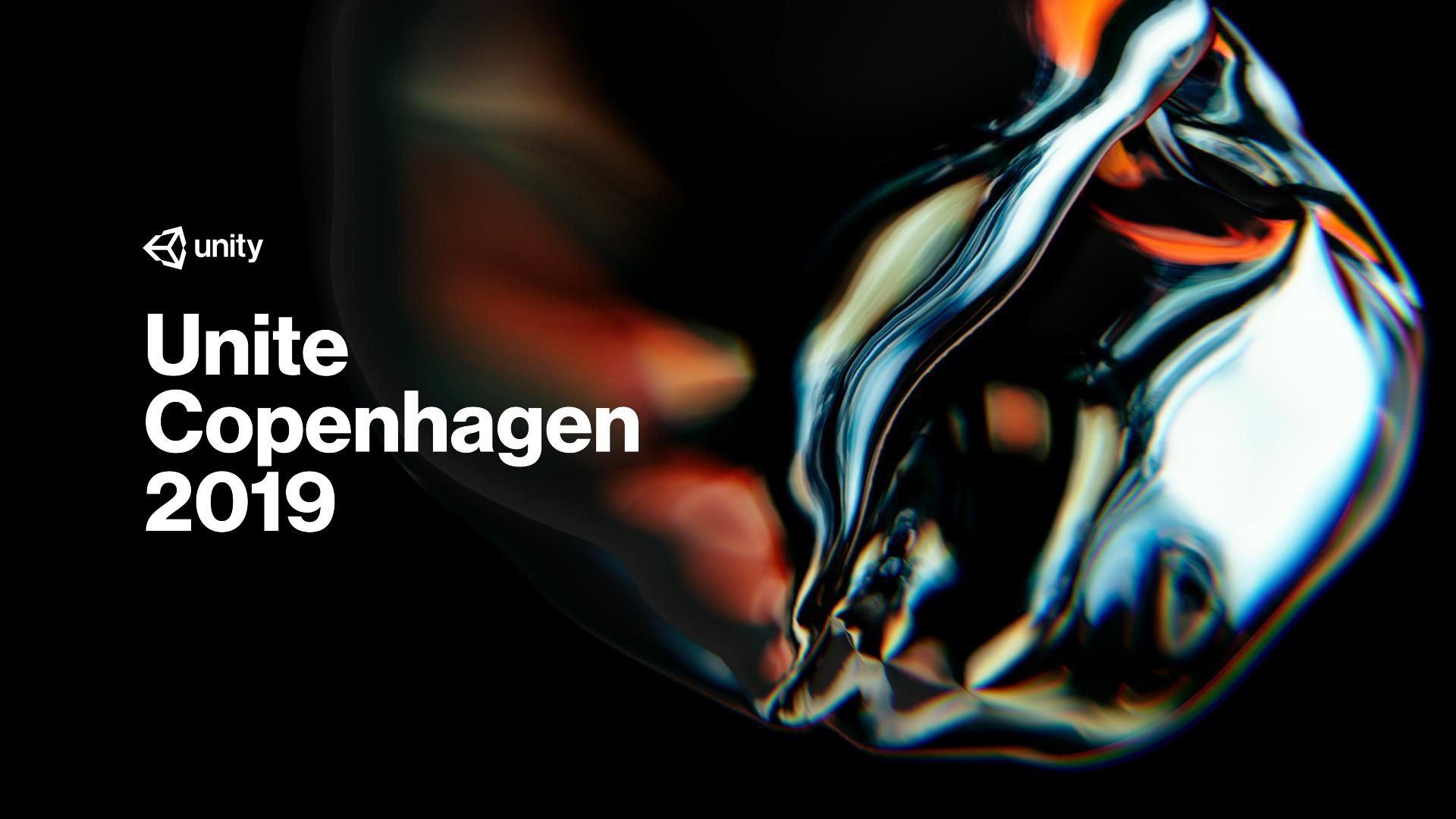
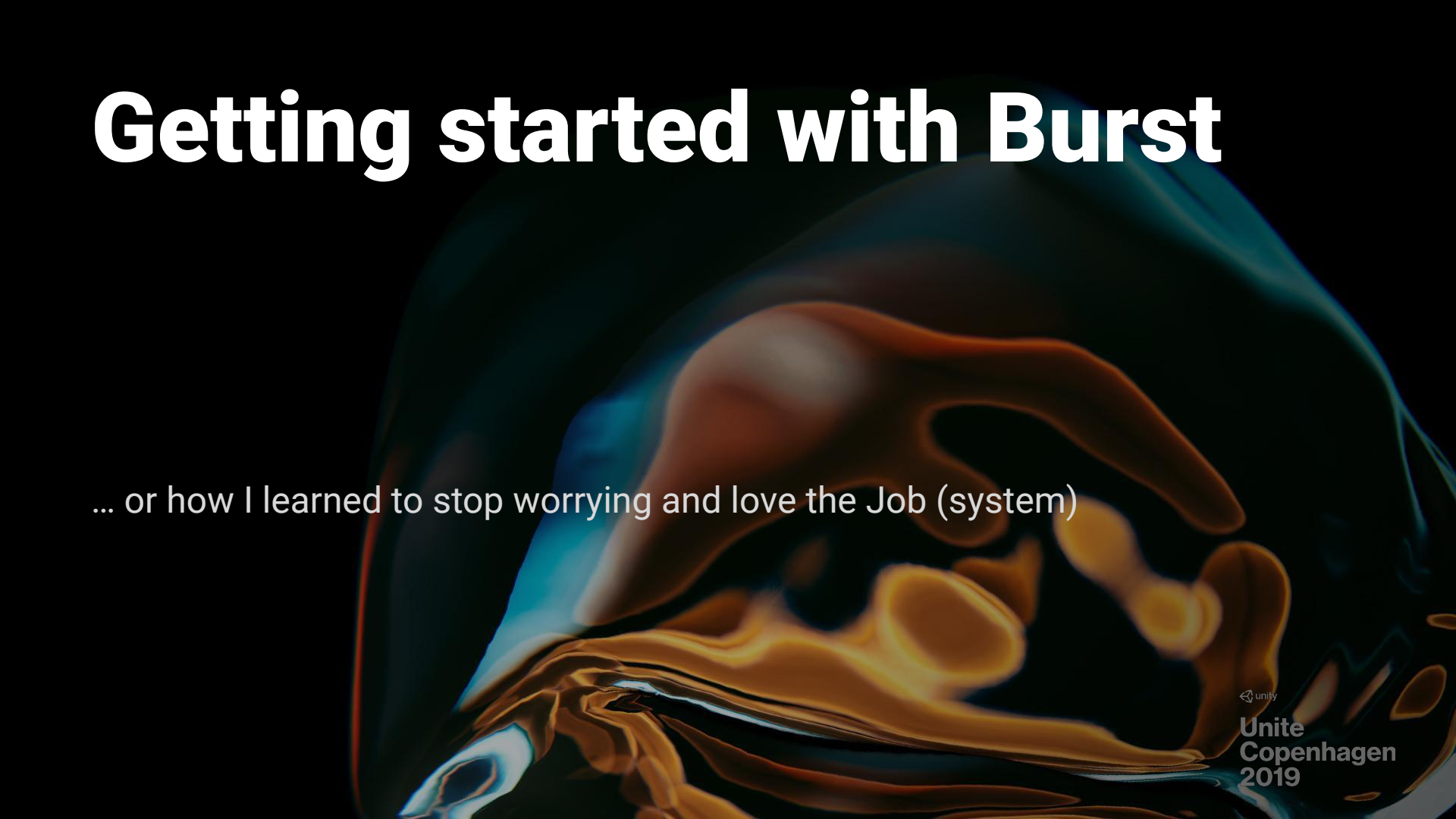

unity
Unite
Copenhagen
2019

Getting started with Burst



… or how I learned to stop worrying and love the Job (system)

unity
Unite
Copenhagen
2019

Burst Team


the ultimate compilation
BASIC
FAIRLY OLD KIDS ON THE BLOCK
(except Marco)
PARENTAL
ADVISORY
EXPLICIT LAYOUT

Agenda

- What is Burst
- Adding Burst to an existing project
- Common pitfalls
  - What you can’t do with Burst
  - What you can do with Burst
- Where to get help
- Future

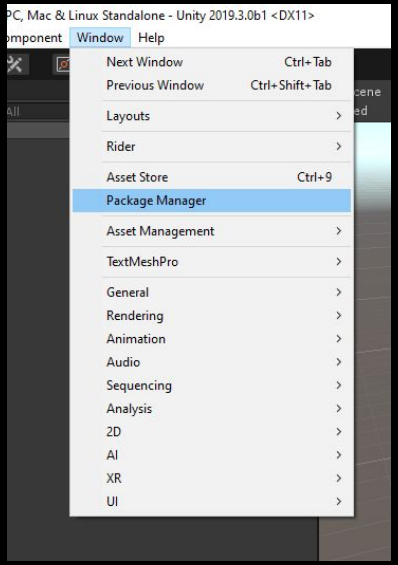
What is Burst

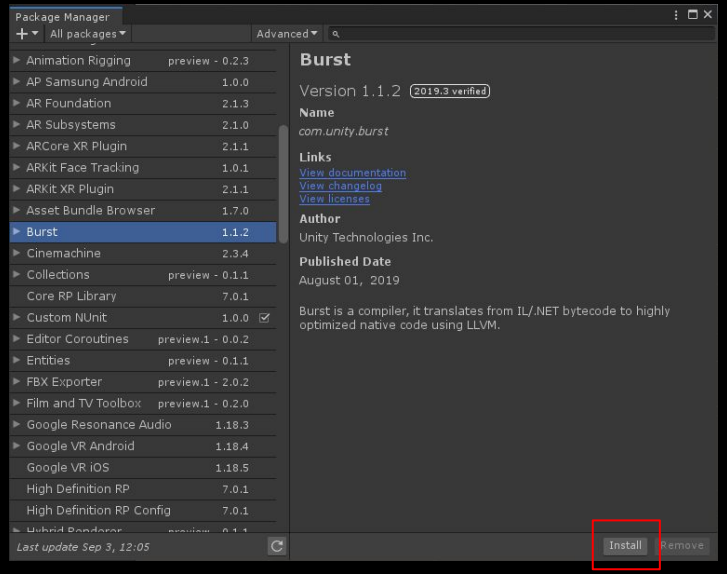
- Burst is a compiler for (some of) your code
- … that makes it run really fast
- It translates .NET bytecode into highly optimised native code
- It works with Unity’s job system
- It is part of the Data Oriented Technology Stack (DOTS)

Demo - Adding Burst to an existing project

Burst package

- Open Package Manager
- Install Burst package





Move code to the job system

- No Job System - No Burst
- We are running the Burst compiled code on the main thread
- The job system allows you to do more
  - Run across multiple threads
  - Schedule work and await completion
  - Setup dependencies between jobs

Move code to the job system

- Create a Job

```
struct MyJob : IJob
{
    // ... Copy data required here
    public void Execute()
    {
        // ... Copy code here
    }
}
```

- Initialise and run the job (inline on the main thread)

```
var t = new MyJob { /* initialise data references here */ };
t.Run();
```

Using Burst to compile code

- Add [BurstCompile] attribute to the job struct

```
[BurstCompile]
struct MyJob : IJob
{
    public Map.MapDataStore map;
    public NativeArray<float> stepField;
    public NativeArray<Vector2> flowField;
    public NativeList<Vector2Int> openSet;
    public NativeList<Vector2Int> nextSet;
    public PathType pathType;

    static readonly Vector2Int[] moveDirs = new Vector2Int[] {new Vector2Int(0,1),
                                        new Vector2Int(1,1),
                                        new Vector2Int(1,0),
                                        new Vector2Int(1,-1),
                                        new Vector2Int(0,-1),
                                        new Vector2Int(-1,-1),
                                        new Vector2Int(-1,0),
                                       new Vector2Int(-1,1)};
    public void Execute()
    {
```

How it works (from 10,000 feet)

When compilation happens

Editor (Just In Time - JIT)

- When a job is scheduled, Burst schedules background compilation for that job
- Original Mono version is used in Play mode until Burst-compiled version is ready
- Use `[BurstCompile(Synchronous=true)]` to force synchronous compilation (if you want to ensure Mono version is never used, such as for profiling)

Player (Ahead Of Time - AOT)

- When building player, Burst compiles your code immediately after normal C# compilation
- Your game ships with precompiled code
- This works the same whether using a Mono or an IL2CPP based player.

AOT additional notes

- Burst currently requires native toolchains (even on desktops)
  - https://docs.unity3d.com/Packages/com.unity.burst@1.1/manual/index.html#burst-aot-requirements
- Most platforms build a shared library
  - Windows, Mac, Linux, Android for example
- Some platforms are statically linked
  - iOS for example

[BurstCompile] attribute

- [BurstCompile]
  - FloatMode (Default, Strict, Deterministic, Fast)
    - Defaults to Strict
  - FloatPrecision (Standard, High, Medium, Low)
    - Defaults to Medium
    - Use unity.mathematics
  - CompileSynchronously
    - Only affects Editor, not standalone Player builds
    - Blocks game until compilation has finished

Common pitfalls

… or confession time!

What you can’t do with Burst

- **Can’t** use reference types (`class`, `string`)
- **Can’t** access classic GameObject/Component code
- **Can’t** write to static variables
- **Can’t** try/catch

What you can’t do with Burst

- https://github.com/Unity-Technologies/GettingStartedWithBurst-Unite2019
- Extra commits
  - Profile Markers
  - C# Arrays -> NativeArrays
  - Map data class -> struct
  - Lists -> NativeLists
  - Lookup table (moveDirs) tweaked

Dealing with arrays

- 2 Dimensional Arrays are converted to 1 Dimensional

```
int[,] occupants;

occupants = new int[width, height];

var value = occupants[x, y];
```

Becomes e.g.

```
int[] occupants;

occupants = new int [width * height];

var value = occupants[x + y * width];
```

Dealing with arrays

- Arrays are converted to NativeArrays

```
int[] occupants;

occupants = new int [width * height];
```

Becomes e.g.

```
NativeArray<int> occupants;

occupants = new NativeArray<int>(width * height, Allocator.Persistent);

// and a destroy method was added to properly dispose of the array

occupants.Dispose();
```

Dealing with Map class

- Map was used as a reference to avoid duplicating data

```
class Map
{
    public NativeArray<MapTile> tiles;
    public NativeArray<int> occupants;
    public int width; public int height;
}
```

Moved the data into a local struct

```
class Map
{
    struct MapDataStore
    {
        public NativeArray<MapTile> tiles;
        public NativeArray<int> occupants;
        public int width; public int height;
    }
    public MapDataStore mapDataStore;
}
```

Dealing with List

- Lists are converted to NativeLists

```
List<Vector2Int> openSet = new List<Vector2Int>();

while (openSet.Count > 0)
```

Becomes e.g.

```
NativeList<Vector2Int> openSet = new NativeList<Vector2Int>(Allocator.Persistent);

while (openSet.Length > 0)

// and because we are using a native container, disposes were added

openSet.Dispose();
```

Dealing with Static Data

- MoveDirs was declared as

```
Vector2Int[] moveDirs = new Vector2Int[] {new Vector2Int(0,1),
                                          new Vector2Int(1,1),
                                          //....
```

Became :

```
static readonly Vector2Int[] moveDirs = new Vector2Int[] {new Vector2Int(0,1),
                                        new Vector2Int(1,1),
                                        ///....
```

- Note this is the only supported use of C# arrays!
- readonly applies to moveDirs, not its contents.
- Possible to modify outside of burst!

Dealing with containers

- If you want a temporary Native container

```
occupants = new NativeArray<int>(width * height, Allocator.TempJob);

// do some work

occupants.Dispose();
```

- Note, Native containers do not have ref indexers

```
NativeArray<Vector2Int> openSet = new NativeArray<Vector2Int>(Allocator.Persistent);

openSet[0] = new Vector2Int(0,0);
```

❌

```
openSet[0].y = 4;           // compilation error, cannot modify struct member access
```

✔️

```
var t = openSet[0];
t.y = 4;
openSet[0] = t;             // fine, if a little verbose
```

What you can do with Burst

- **Can** use `struct` and other value types
- **Can** be used with ECS
- **Can** use generic types
- **Can** improve performance

Floating-point performance

- Use `FloatMode.Fast` to speed up floating-point calculations

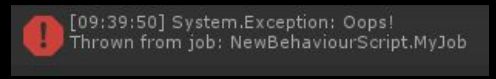
```
[BurstCompile(FloatMode = FloatMode.Fast)]
struct MyJob : IJob
{
```

- Sacrifices accuracy
- Burst Inspector defaults to showing the results as if FloatMode.Fast was used!

Throwing exceptions

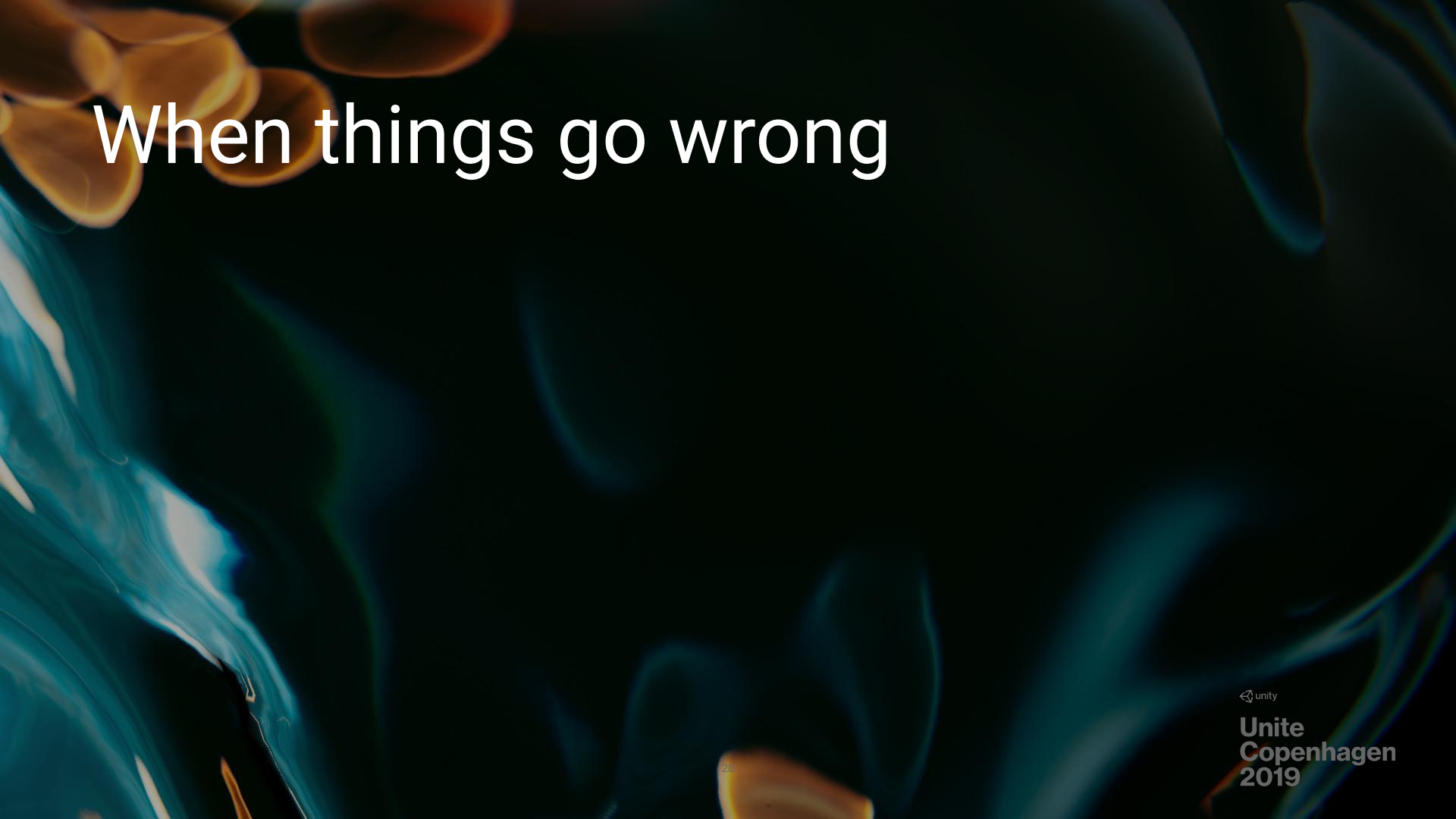
- You can throw exceptions
- … **but** only do this in the Editor… because throwing exceptions crashes standalone players
- Exception messages are output as errors to console
- … and then your code will continue running
- Can't use `try/catch`

```
throw new Exception("Oops!");
```



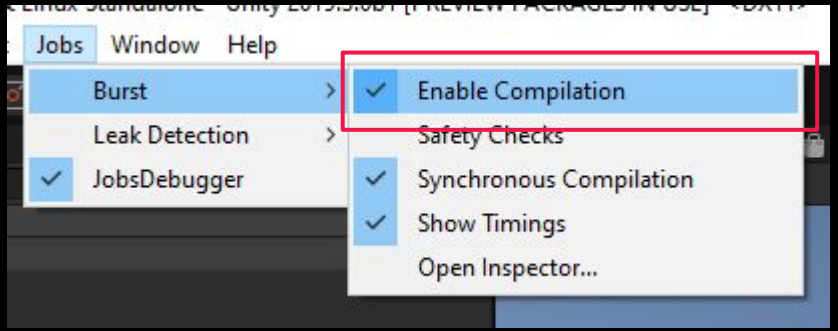
Demo - Copying To Avoid Converting Types

When things go wrong



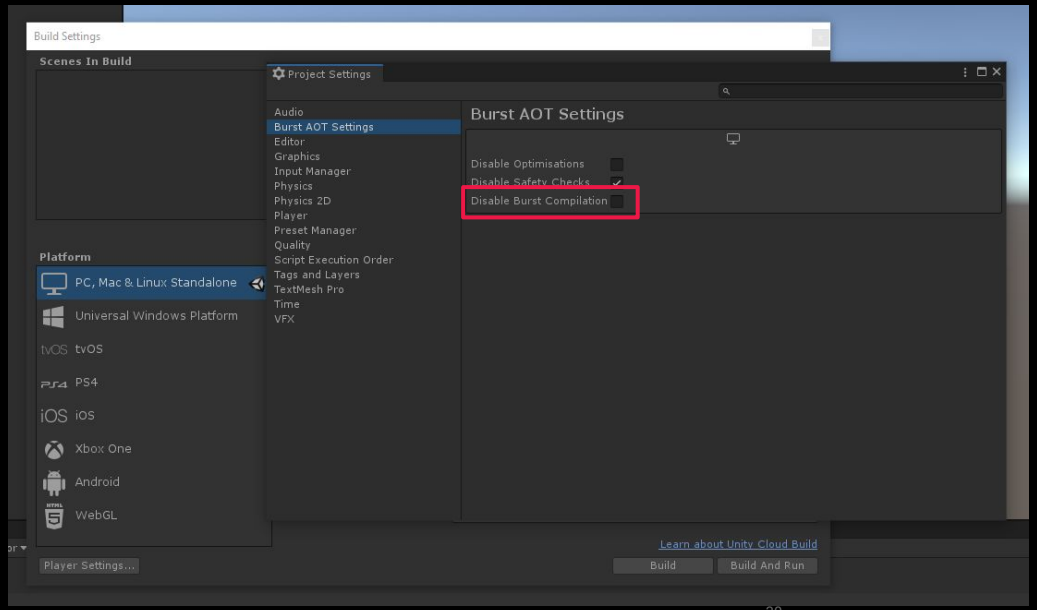
Enable Burst compilation in Editor

- Ensure you have ticked Enable Compilation in the Burst menu



Enable Burst compilation in Player

- Ensure you have **not** ticked Disable Burst Compilation in player build settings



Burst Inspector

- Lists all the jobs in your project
- Jobs that are not burst-compiled are greyed-out
- You can view the resulting assembler etc
- [Enhanced Disassembly] = intermixed asm & c#

Dealing with runtime crashes

- Look in the player.log and locate the stack trace :

```
========== OUTPUTTING STACK TRACE ==================

0x00007FFB930A131E (lib_burst_generated) fb9e181e1474d8fe95fc697646aad17d
0x00007FFB05F03DDB (UnityPlayer) UnityMain
0x00007FFB05F0445C (UnityPlayer) UnityMain
.....
```

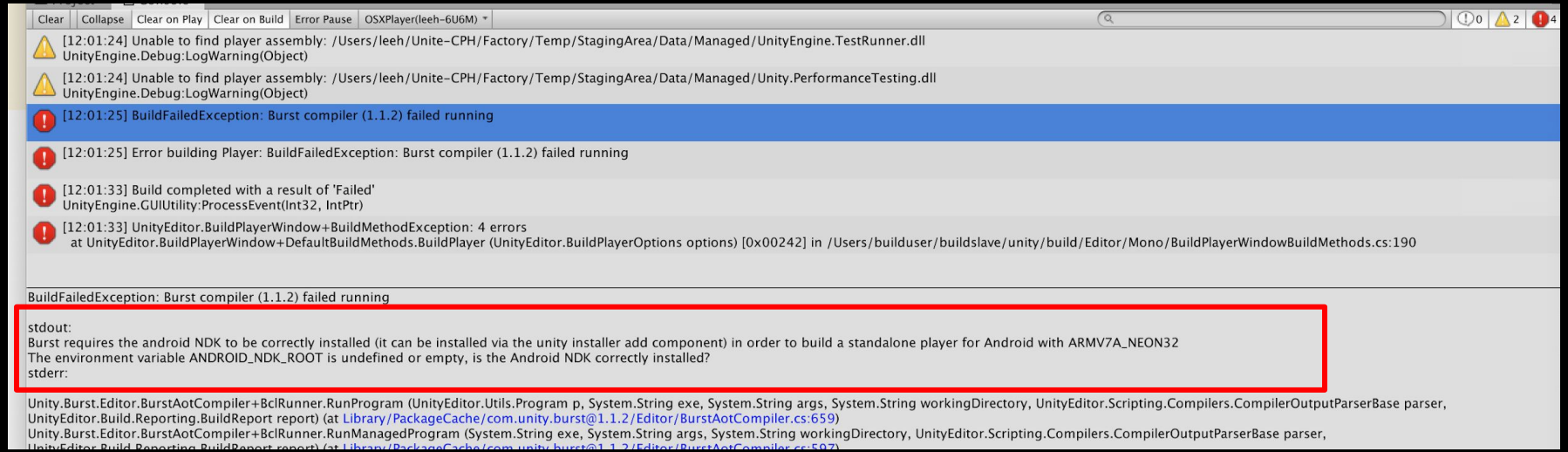
Dealing with runtime crashes

- Look in the lib_burst_generated.txt for the previously highlighted hash
- Scan along the line (backwards from the hash), looking for ::Execute
- The name of the job struct and the parent class can be seen

PublicKeyToken=**null::Execute(FlowField+MyJob**&, Assembly-CSharp, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null|System.IntPtr, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089|System.IntPtr, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089|Unity.Jobs.LowLevel.Unsafe.JobRanges&, UnityEngine.CoreModule, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null|System.Int32, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089)--**fb9e181e1474d8fe95fc697646aad17d**

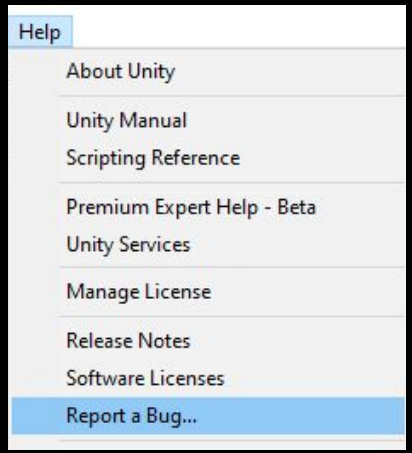
Finding more detail on errors

- Burst Failures when building standalone player
- Sometimes you have to click for more information



Where to get help

- Read the user guide:
  https://docs.unity3d.com/Packages/com.unity.burst@1.1/manual/index.html
- Ask a question in the DOTS forum:
  https://forum.unity.com/forums/data-oriented-technology-stack.147/
- Please report bugs! [via menu in Unity]



Future plans

Coming in 2019.3 / Burst 1.2.0

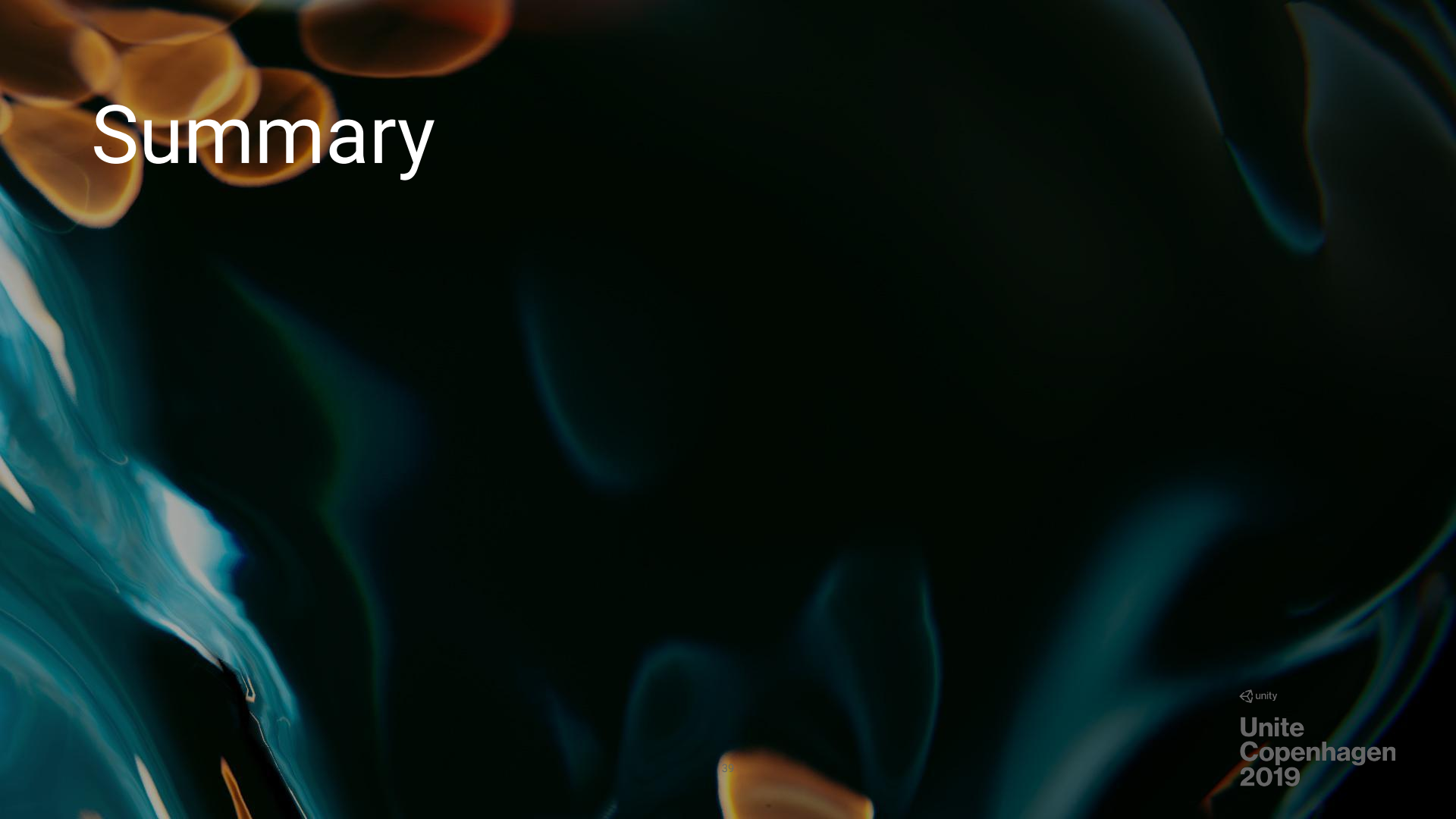
- LLVM version 8
- Various bugs squashed
- Entity Command Buffer (ECB)
- Improved compilation times

You can test out these features in 1.2.0-preview.5 Today

Roadmap

- Determinism
- Debugging
  - E.g. Logging messages from a burst compiled job
- Improved performance of generated code
- Compilation time
- Cross-compilation
- Additional Platforms

Summary



Summary

- Burst is Almost Free Performance
- Job System gets you 99% of the way
- Copy/Burst/Copy can be enough

Thankyou!

https://github.com/Unity-Technologies/GettingStartedWithBurst-Unite2019