Stadia Adventures in Slow Server Code on Unity

with Scott Wardle

Google for Games

Scott Wardle

Senior Software Engineer, Google Stadia

scottwardle@google.com

Agenda

01.

Problems with Stadia Adventures Sample

Stadia Adventures

Unity game

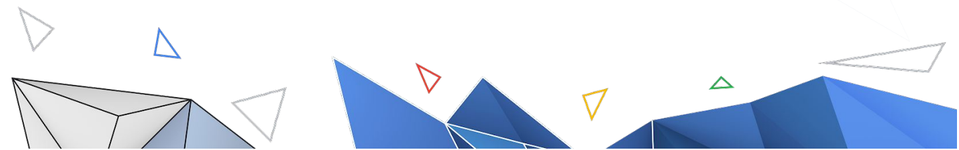
Unity game

- **Designed for Stadia** instances
- **Build** and run with **Unity Editor**

Dedicated server

C# Linux Process

- **Separate** sln for a **build system**
- Run on **Google Cloud** instances
- Deployed using **kubernetes**



A Good Idea!

Unity game

Unity game

- **Designed for Stadia** instances
- **Build** and run with **Unity Editor**

C# .DLL (Server thread)

- Build system still **separate sln**
- **Copy .DLL** to the Unity project
- **Magically re-compiled** to native
  .DLL → il2cpp → .cpp → .so file

But Some Trouble Happened

Unity game

Unity game

- **Designed for Stadia** instances
- **Build** and run with **Unity Editor**

C# .DLL (Server thread)

- Build system still **separate sln**
- **Copy .DLL** to the Unity project
- **Magically re-compiled** to native .DLL → il2cpp → .cpp → .so file

Issues

- C# server code has a separate sln to build, so **can't just build** in Unity Editor

But Some Trouble Happened

Unity game

Unity game

- **Designed for Stadia** instances
- **Build** and run with **Unity Editor**

C# .DLL (Server thread)

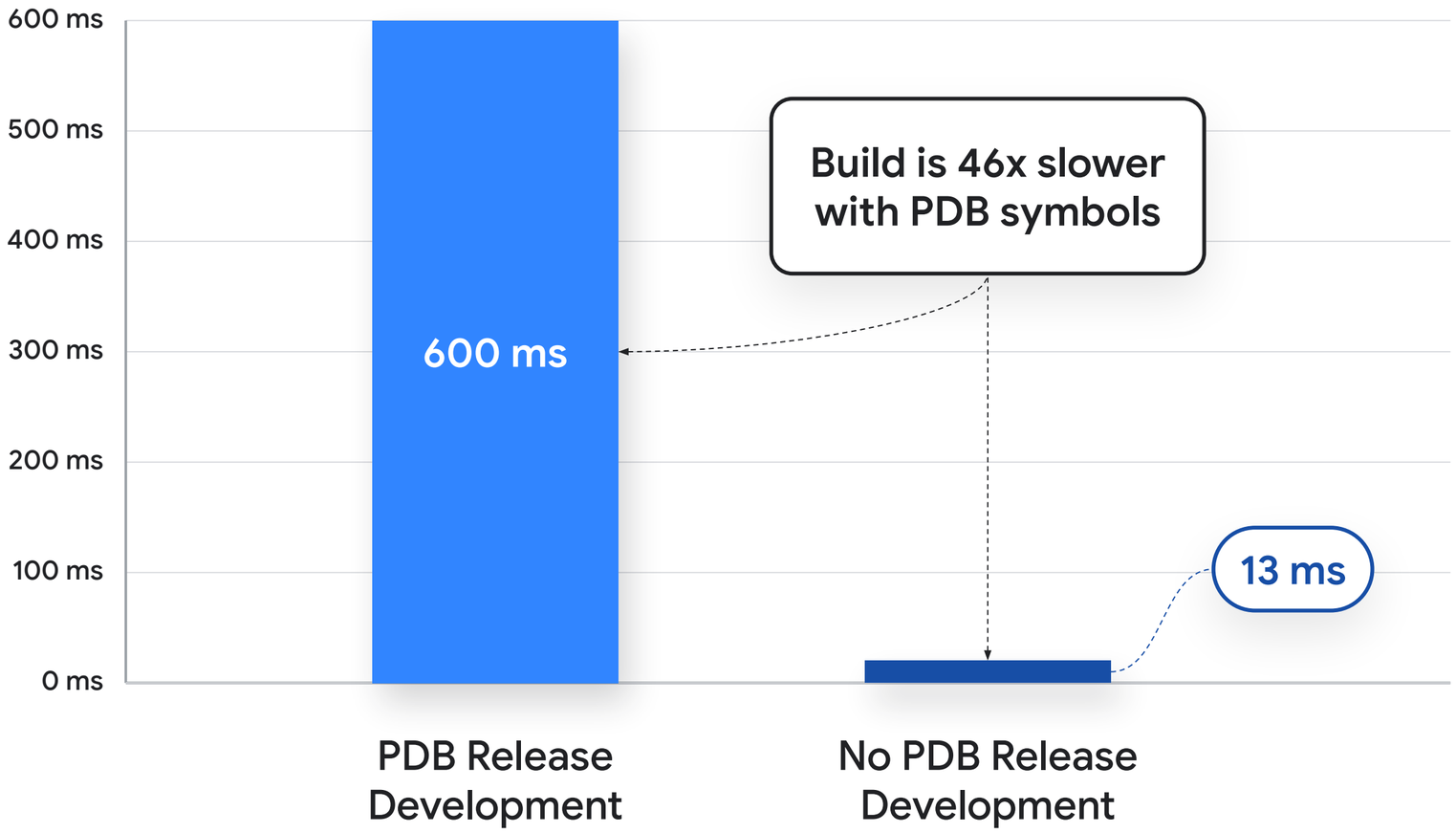
- Build system still **separate sln**
- **Copy .DLL** to the Unity project
- **Magically re-compiled** to native .DLL → il2cpp → .cpp → .so file

Issues

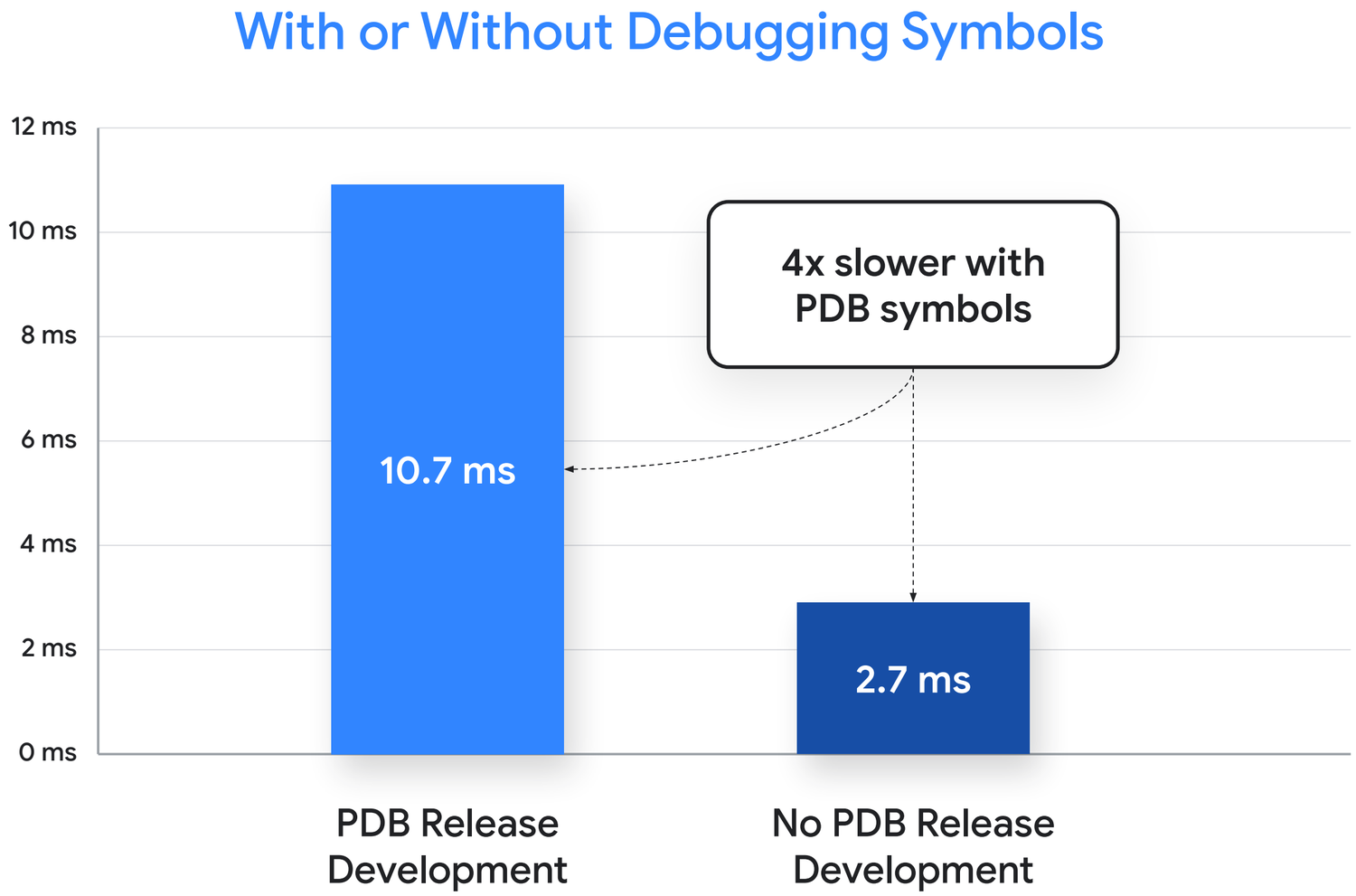
- C# server code has a separate sln to build, so **can't just build** in Unity Editor
- If we **copy the .PDB** symbols for debugger
- Frame rate **drops to 2fps**

Server Tick Performance Release Development Build

With or Without Debugging Symbols



Most of the Time in BroadPhaseCollisionCheck



Try to Optimize BroadPhaseCollisionCheck

- Code was **easy to multi thread**
- **8 cores** on Stadia
- Brute force didn’t work, **slower with more threads**

Using Orbit, Stadia's Native Profiler to Debug ASM

- **CHECK_SEQ_POINT** has a **cmpxchg** atomic op
- Unity debugger is a **cooperative debugger**
- One **sequence point** for **each step** in the debugger!
- So **atomic op** for **each step** in the debugger

```
CHECK_SEQ_POINT(methodExecutionContext, (g_sequencePointsAssemblyU2DCSharp + 31125));
fae1d93c5db: mov          ecx, dword ptr [r14 + 0x12ff68]          0     0.00 %
fae1d93c5e2: mov          eax, 0xffffffff                        667     2.09 %
fae1d93c5e7: lock cmpxchg dword ptr [r14 + 0x12ff68], ecx         13     0.04 %
fae1d93c5f0: test         eax, eax                              1428     4.48 %
fae1d93c5f2: jg           0x7fae1d93c5fb                           0     0.00 %
fae1d93c5f4: cmp          dword ptr [r12], 0                      35     0.11 %
fae1d93c5f9: je           0x7fae1d93c60d                          10     0.03 %
fae1d93c5fb: mov          rdi, qword ptr [rsp + 0x198]             0     0.00 %
fae1d93c603: mov          qword ptr [rsp + 0x40], rdi              0     0.00 %
fae1d93c608: call         0x7fae1d40dba0                           0     0.00 %
```

Using Orbit, Stadia's Native Profiler to Debug ASM

- **CHECK_SEQ_POINT** has a **cmpxchg** atomic op
- Unity debugger is a **cooperative debugger**
- One **sequence point** for **each step** in the debugger!
- So **atomic op** for **each step** in the debugger

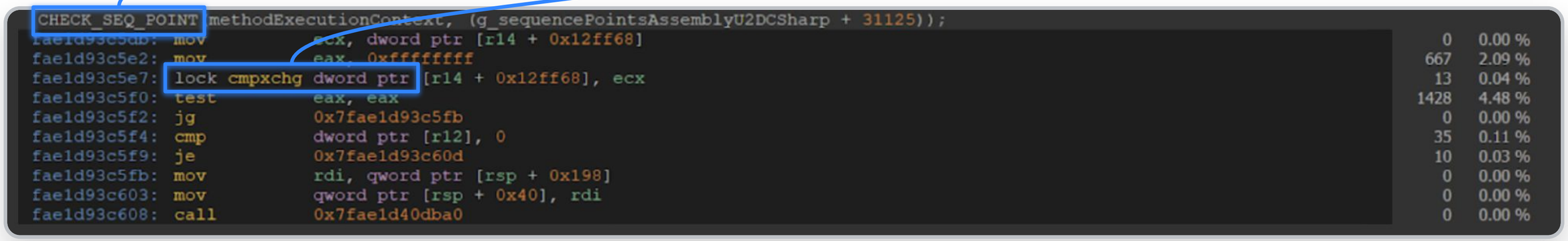
CHECK_SEQ_POINT function

Using Orbit, Stadia's Native Profiler to Debug ASM

- **CHECK_SEQ_POINT** has a **cmpxchg** atomic op
- Unity debugger is a **cooperative debugger**
- One **sequence point** for **each step** in the debugger!
- So **atomic op** for **each step** in the debugger

CHECK_SEQ_POINT function

Compare and Exchange



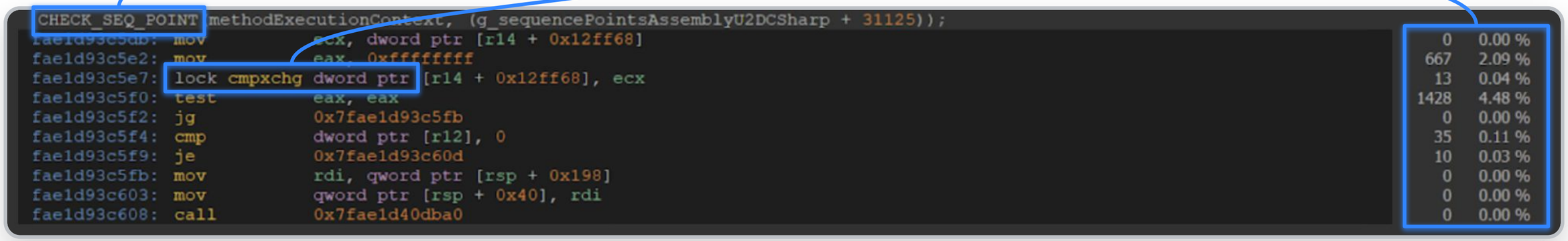
Using Orbit, Stadia's Native Profiler to Debug ASM

- **CHECK_SEQ_POINT** has a **cmpxchg** atomic op
- Unity debugger is a **cooperative debugger**
- One **sequence point** for **each step** in the debugger!
- So **atomic op** for **each step** in the debugger

CHECK_SEQ_POINT function

Compare and Exchange

Lots of profile samples



The Fix to Sequence Point's Memory Barrier

Got a fix from Unity

Version

- 2020.3.26
- 2021.2.8
- 2022.1.0b3
- 2022.2.0a1

Look in **<Unity version>\Editor\Data\il2cpp\libil2cpp\vm-utils\Debugger.h** for **AtomicReadIsActive** if you have this you need the fix.

```
static inline bool IsSequencePointActive(Il2CppSequencePoint *seqPoint)
{
    return il2cpp::os::Atomic::CompareExchange(&seqPoint->isActive, seqPoint->isActive, -1) > 0
    || g_unity_pause_point_active;
}


static inline bool IsSequencePointActive(Il2CppSequencePoint *seqPoint)
{
    return il2cpp::os::Atomic::LoadRelaxed(&seqPoint->isActive)
    || g_unity_pause_point_active;
}
```


The Fix to Sequence Point's Memory Barrier

Got a fix from Unity

Version

- 2020.3.26
- 2021.2.8
- 2022.1.0b3
- 2022.2.0a1

Look in **<Unity version>\Editor\Data\il2cpp\libil2cpp\vm-utils\Debugger.h** for **AtomicReadIsActive** if you have this you need the fix.

Older!! Slower!!

```
static inline bool IsSequencePointActive(Il2CppSequencePoint *seqPoint)
{
    return il2cpp::os::Atomic::CompareExchange(&seqPoint->isActive, seqPoint->isActive, -1) > 0
    || g_unity_pause_point_active;
}


static inline bool IsSequencePointActive(Il2CppSequencePoint *seqPoint)
{
    return il2cpp::os::Atomic::LoadRelaxed(&seqPoint->isActive)
    || g_unity_pause_point_active;
}
```


The Fix to Sequence Point's Memory Barrier

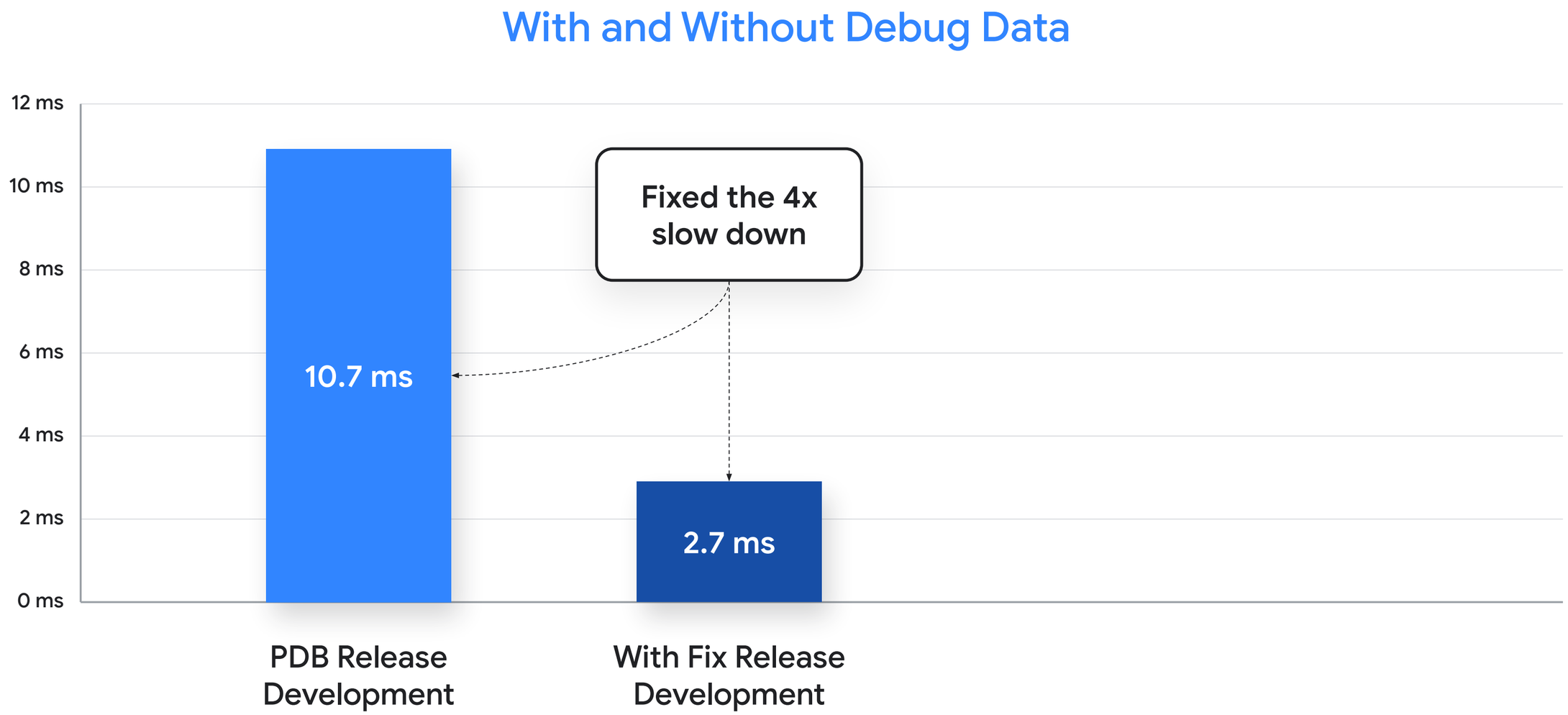
Got a fix from Unity

Version

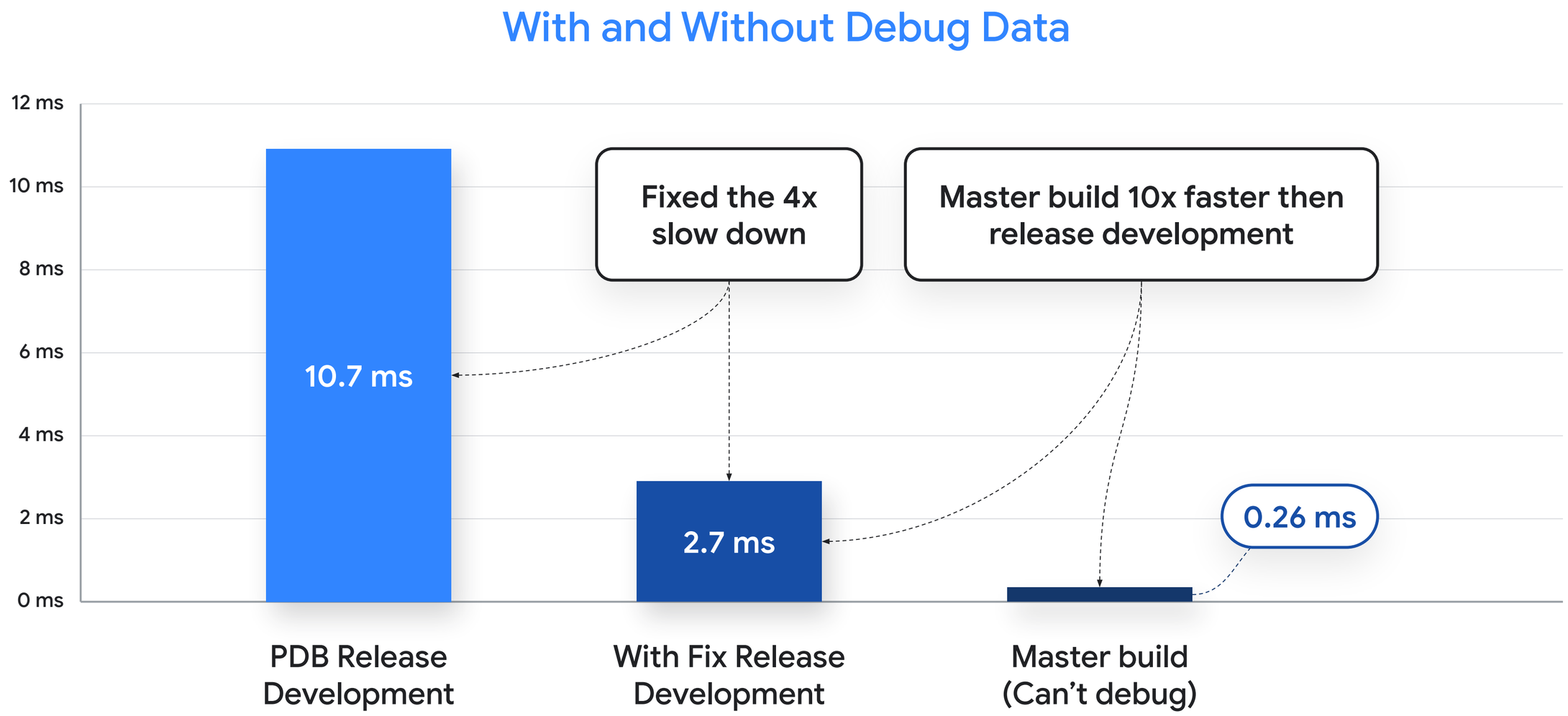
- 2020.3.26
- 2021.2.8
- 2022.1.0b3
- 2022.2.0a1

Look in **<Unity version>\Editor\Data\il2cpp\libil2cpp\vm-utils\Debugger.h** for **AtomicReadIsActive** if you have this you need the fix.

The Debugger Barrier Fix on BroadPhaseCollisionCheck



The Debugger Barrier Fix on BroadPhaseCollisionCheck



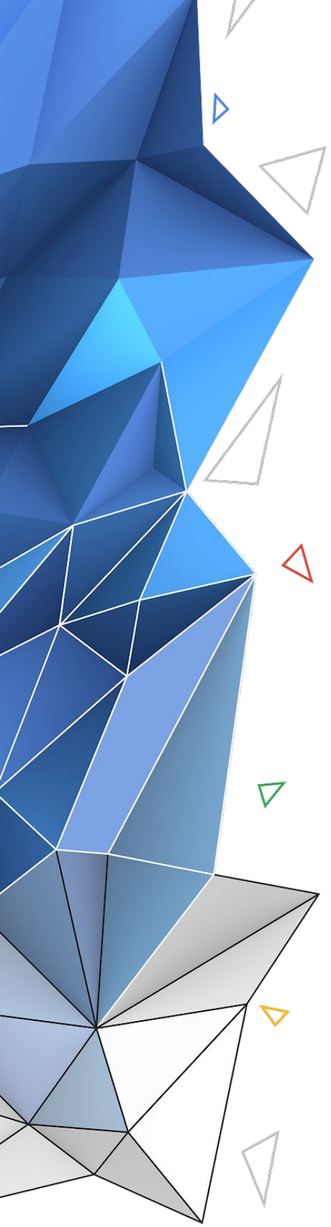
One bug down but...

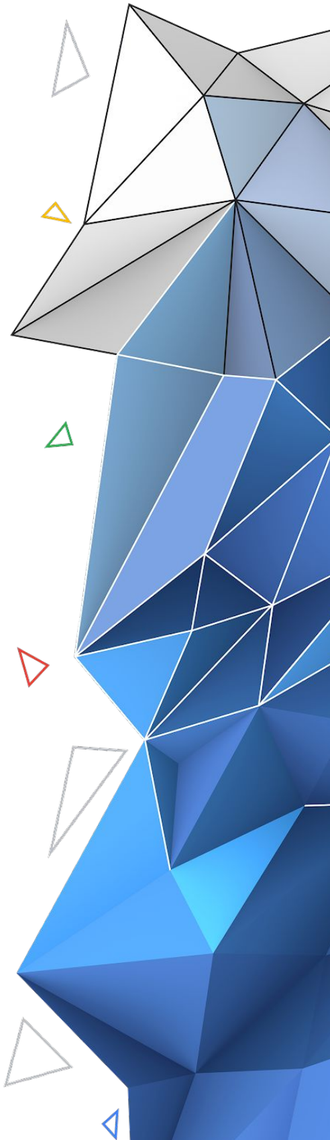
- More profiling reveals in the release development build we are **dropping frames**
- 13 ms on average but **failing to hit 60 hz** in some scenes
- **Can’t include .PDB** symbols yet
- Still need **2 build systems**

02.

What is Burst and Jobs

Overview





Unity's Jobs System

- **Multithreaded code** by creating jobs
- Uses a pool of worker threads **one thread per core**
- Jobs are **put onto a job queue** to execute
- Each worker thread **takes items from job queue**

Unity's Burst Compiler

- Burst is separate **C# compiler**
- Burst uses a limited C like **subset of C#**
- Uses LLVM to **build to native** code
- Uses the native debugger **not the managed debugger**
- No managed debugger so **no sequence points**
- The native debugger will **debug C# code** (not il2cpp output)

Burst limited subset of C#

- Use **Structs** and Primitive Types (**not Classes**)
- Use **NativeArray**<T> not List<T>
- NativeArray **memory is on the heap** and not Garbage collected
- **Call Dispose** on NativeArrays to free memory

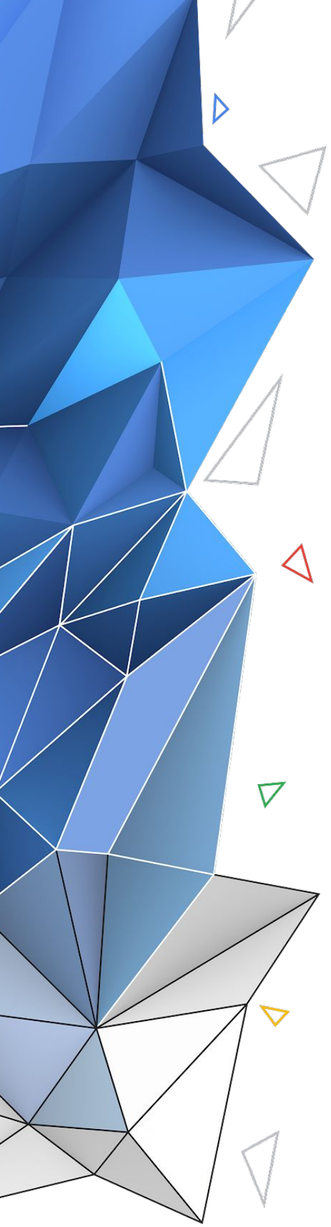
The relationship between Burst, jobs and C# runtime

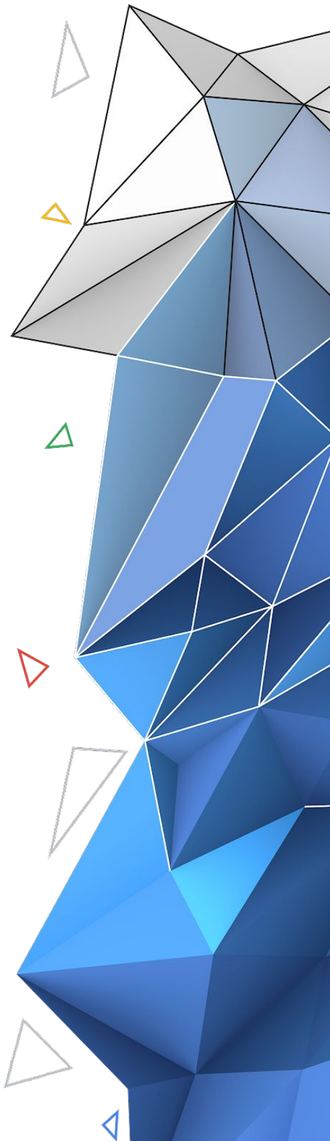
- Normally **Burst** code **only** can be run **as a job**
- So normally must redesign your code as **jobs to use Burst**
- The C# runtime does **not know Burst or jobs**, so you **must pin memory** before using
- Can run **managed C# as a job** if you **pin data** in the GC

02.

What is Burst and Jobs

Example Burst job code





Example of job code

```
[BurstCompile]
struct DropPool : IJob
{
    public NativeArray<float> buffer1;
    public NativeArray<float> buffer2;
    public NativeArray<Color32> output;
    public NativeArray<int> frame;
    public NativeArray<Unity.Mathematics.Random> random;

    public void Init() { ... }
    public void Dispose() { ... }

    [BurstCompile]
    public void Execute() { ... }
}
```


Example of job code

Lots of NativeArrays

```
[BurstCompile]
struct DropPool : IJob
{
    public NativeArray<float> buffer1;
    public NativeArray<float> buffer2;
    public NativeArray<Color32> output;
    public NativeArray<int> frame;
    public NativeArray<Unity.Mathematics.Random> random;

    public void Init() { ... }
    public void Dispose() { ... }

    [BurstCompile]
    public void Execute() { ... }
}
```

Example of job code

Lots of NativeArrays

Create and Destroy NativeArrays at beginning or end of program

```
[BurstCompile]
struct DropPool : IJob
{
    public NativeArray<float> buffer1;
    public NativeArray<float> buffer2;
    public NativeArray<Color32> output;
    public NativeArray<int> frame;
    public NativeArray<Unity.Mathematics.Random> random;

    public void Init() { ... }
    public void Dispose() { ... }

    [BurstCompile]
    public void Execute() { ... }
}
```

Example of job code

Lots of NativeArrays

Create and Destroy NativeArrays at beginning or end of program

A Burst compiled member function to be run on a job

```
[BurstCompile]
struct DropPool : IJob
{
    public NativeArray<float> buffer1;
    public NativeArray<float> buffer2;
    public NativeArray<Color32> output;
    public NativeArray<int> frame;
    public NativeArray<Unity.Mathematics.Random> random;

    public void Init() { ... }
    public void Dispose() { ... }

    [BurstCompile]
    public void Execute() { ... }
}
```


```
public class UnityJobsBehaviour : MonoBehaviour
{
    DropPool m_dropPool;
    JobHandle jobHandle;

    public void Awake()
    {
        dropPool.Init();
    }
    public void OnDestroy()
    {
        dropPool.Dispose();
    }
    public void Update()
    {
        jobHandle = dropPool.Schedule();
        JobHandle.ScheduleBatchedJobs();
    }
    public void LateUpdate()
    {
        jobHandle.Complete();
        jobHandle.Dispose();
    }
}
```

Create NativeArrays at startup

```
public class UnityJobsBehaviour : MonoBehaviour
{
    DropPool m_dropPool;
    JobHandle jobHandle;

    public void Awake()
    {
        dropPool.Init();
    }
    public void OnDestroy()
    {
        dropPool.Dispose();
    }
    public void Update()
    {
        jobHandle = dropPool.Schedule();
        JobHandle.ScheduleBatchedJobs();
    }
    public void LateUpdate()
    {
        jobHandle.Complete();
        jobHandle.Dispose();
    }
}
```

Create NativeArrays at startup

Destroy NativeArrays on shutdown

```
public class UnityJobsBehaviour : MonoBehaviour
{
    DropPool m_dropPool;
    JobHandle jobHandle;

    public void Awake()
    {
        dropPool.Init();
    }
    public void OnDestroy()
    {
        dropPool.Dispose();
    }
    public void Update()
    {
        jobHandle = dropPool.Schedule();
        JobHandle.ScheduleBatchedJobs();
    }
    public void LateUpdate()
    {
        jobHandle.Complete();
        jobHandle.Dispose();
    }
}
```

Create NativeArrays at startup

Destroy NativeArrays on shutdown

Run jobs each frame

```
public class UnityJobsBehaviour : MonoBehaviour
{
    DropPool m_dropPool;
    JobHandle jobHandle;

    public void Awake()
    {
        dropPool.Init();
    }
    public void OnDestroy()
    {
        dropPool.Dispose();
    }
    public void Update()
    {
        jobHandle = dropPool.Schedule();
        JobHandle.ScheduleBatchedJobs();
    }
    public void LateUpdate()
    {
        jobHandle.Complete();
        jobHandle.Dispose();
    }
}
```

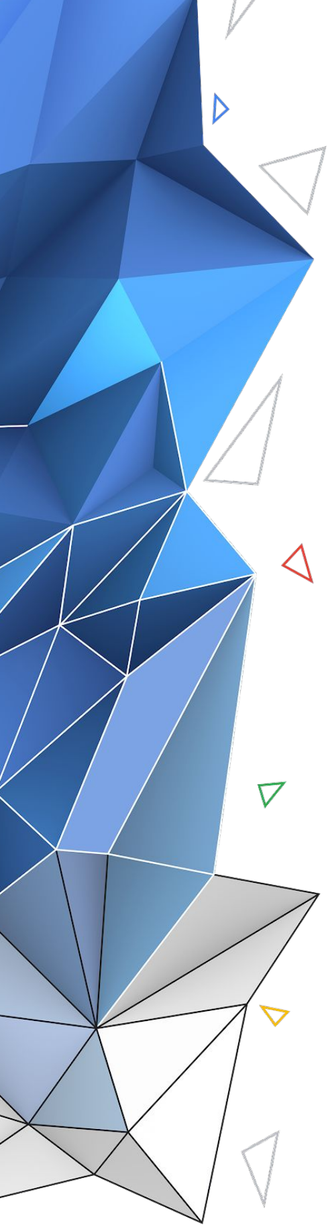
Create NativeArrays at startup

Destroy NativeArrays on shutdown

Run jobs each frame

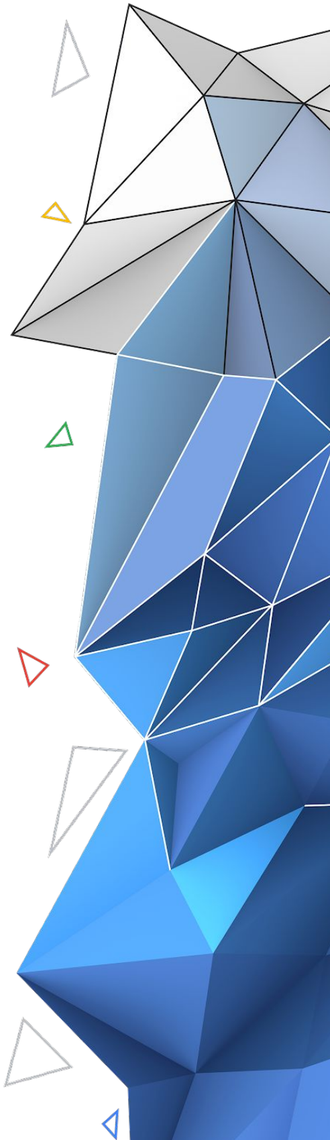
Later in the frame wait for jobs to be finished

```
public class UnityJobsBehaviour : MonoBehaviour
{
    DropPool m_dropPool;
    JobHandle jobHandle;

    public void Awake()
    {
        dropPool.Init();
    }
    public void OnDestroy()
    {
        dropPool.Dispose();
    }
    public void Update()
    {
        jobHandle = dropPool.Schedule();
        JobHandle.ScheduleBatchedJobs();
    }
    public void LateUpdate()
    {
        jobHandle.Complete();
        jobHandle.Dispose();
    }
}
```

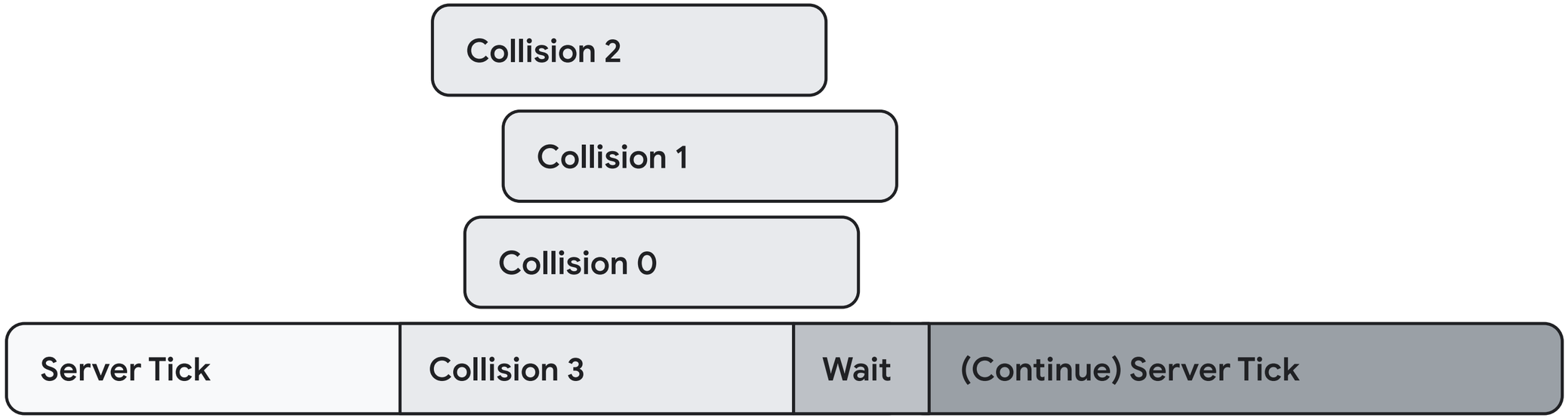


03.

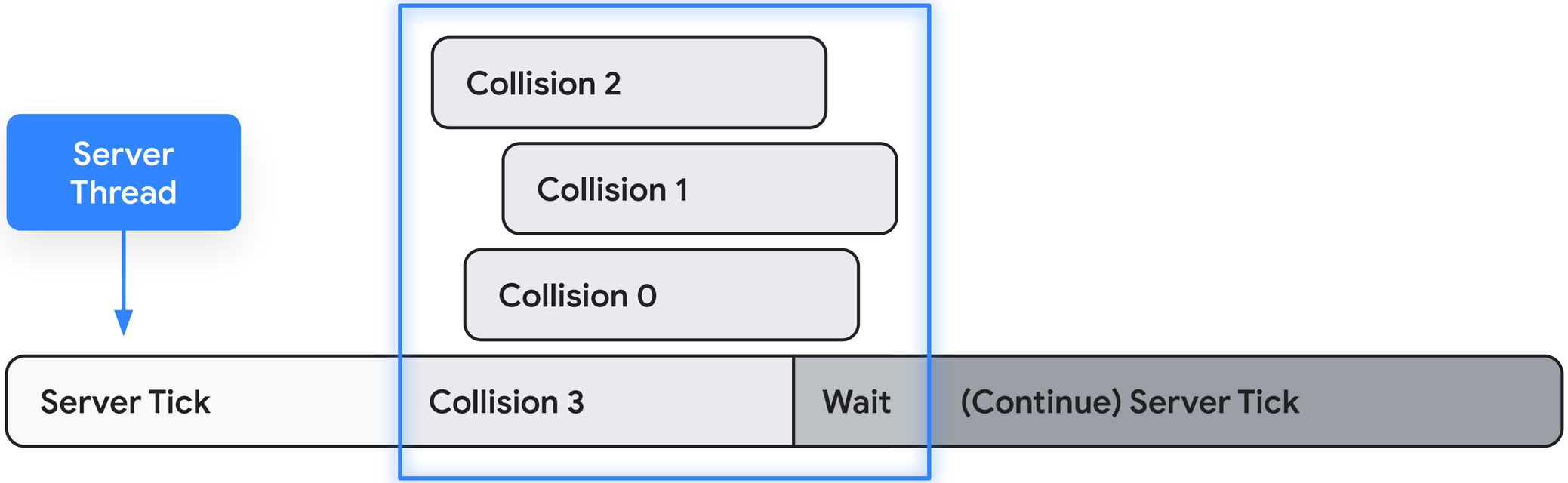
Try Burst and jobs



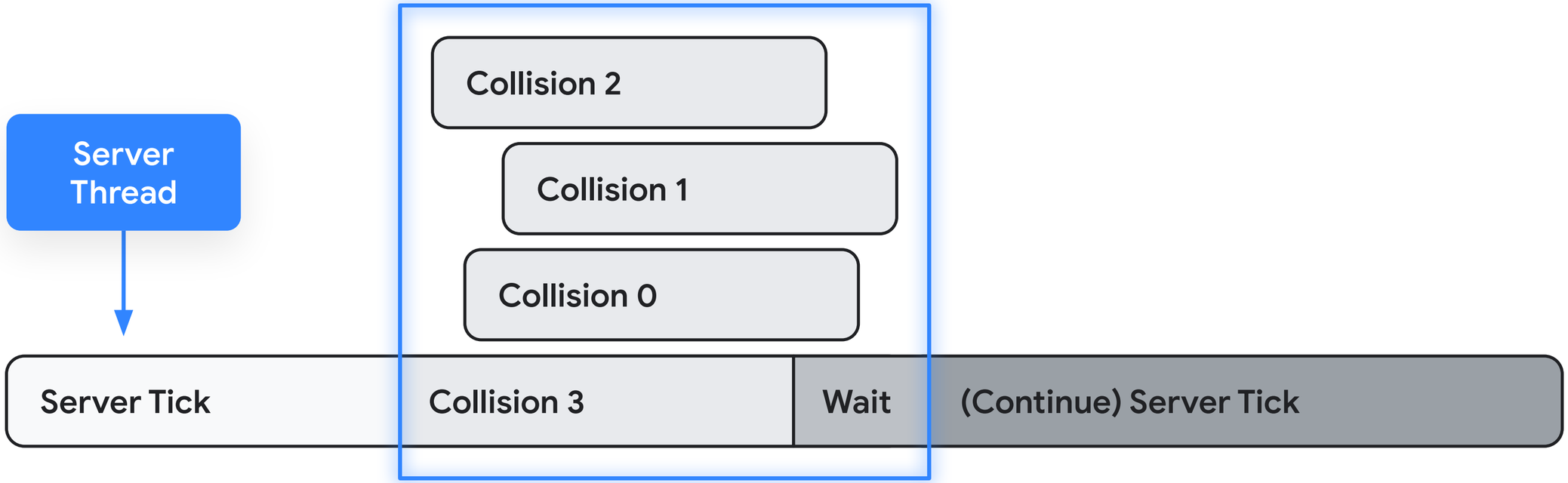
Fail #1 - Convert BroadPhaseCollisionCheck into a Burst Job



Fail #1 - Convert BroadPhaseCollisionCheck into a Burst Job

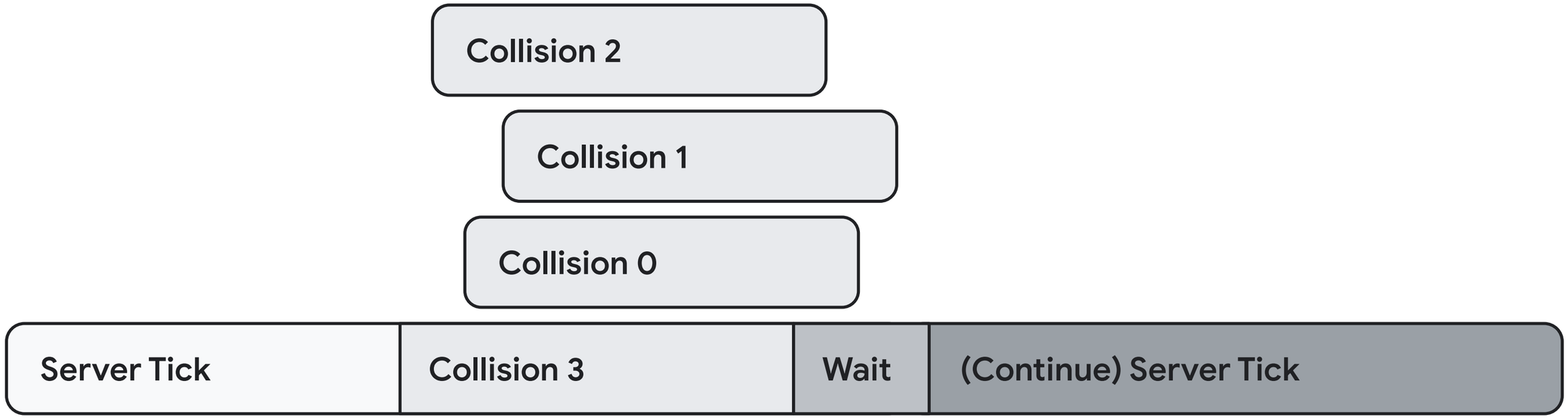


Fail #1 - Convert BroadPhaseCollisionCheck into a Burst Job

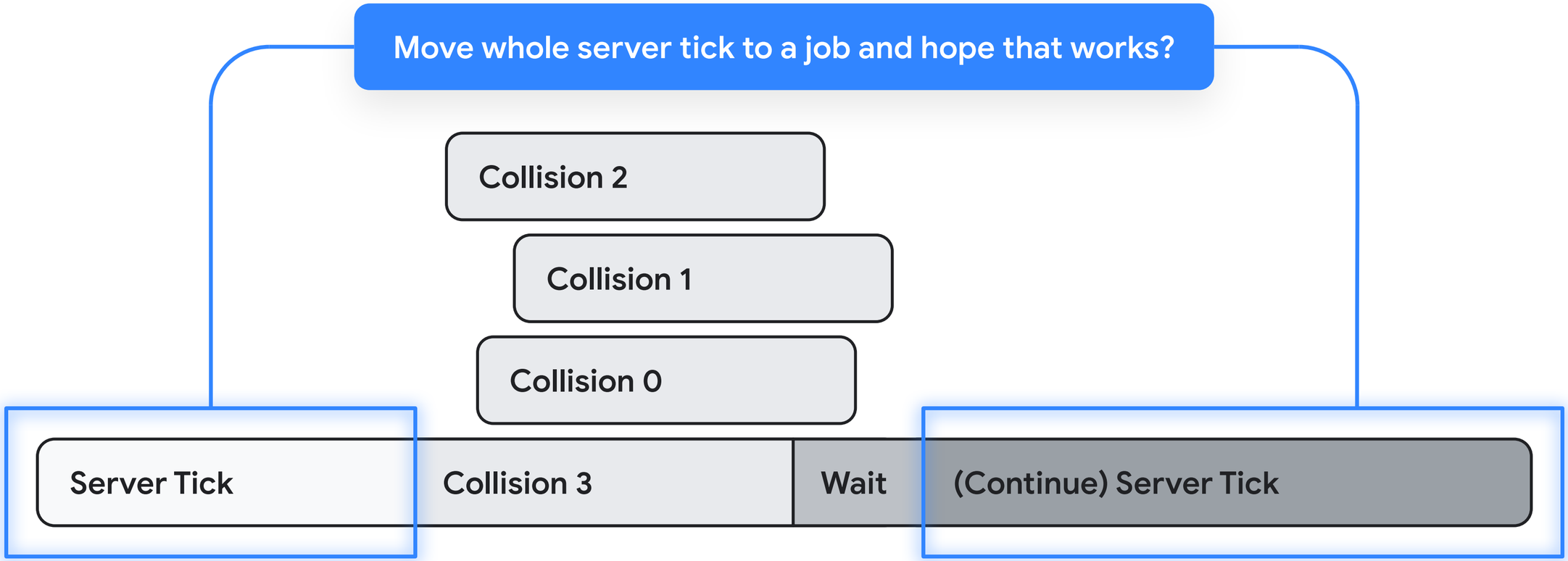


Error! Job have to be run from main thread.

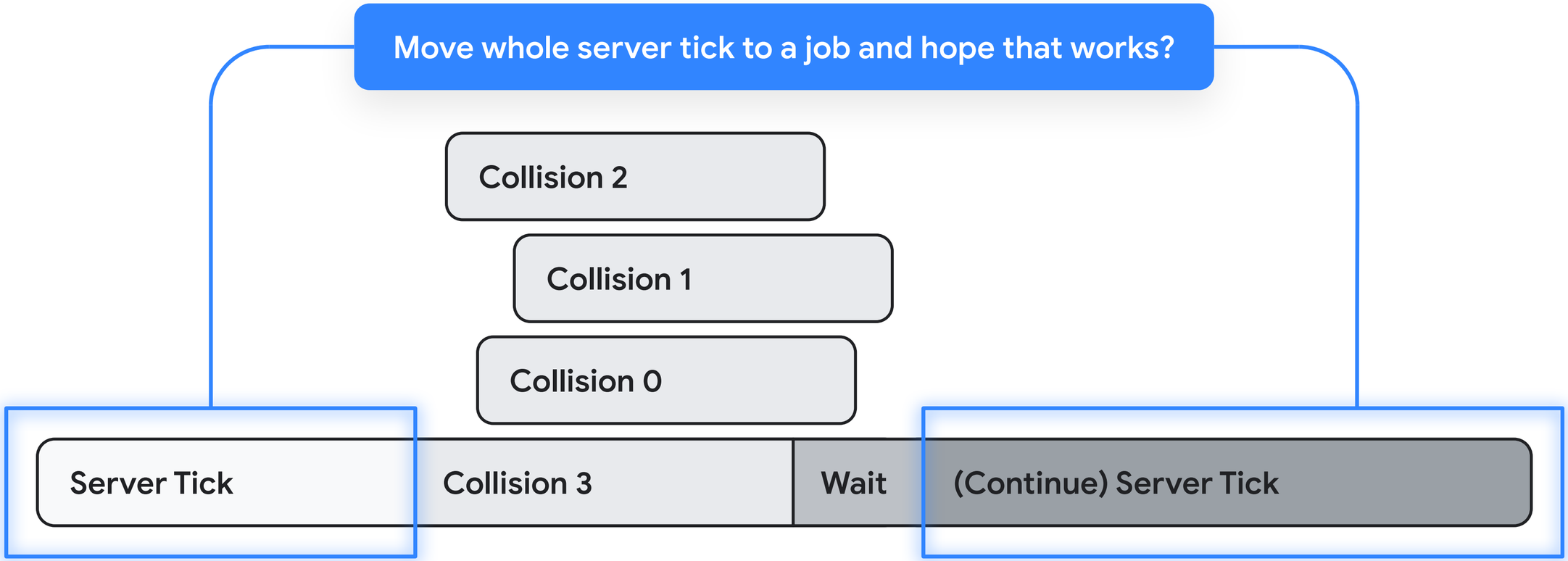
Fail #2 - Move Whole Server Tick to a Job



Fail #2 - Move Whole Server Tick to a Job



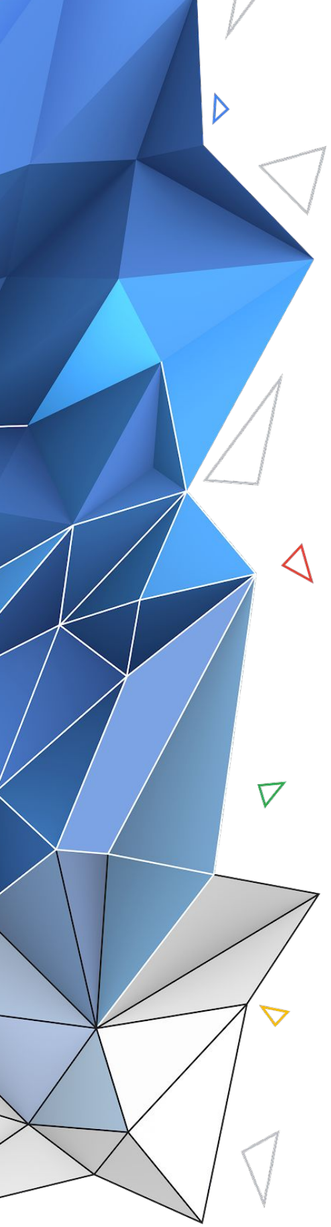
Fail #2 - Move Whole Server Tick to a Job



Error! Can't run a job from another job.

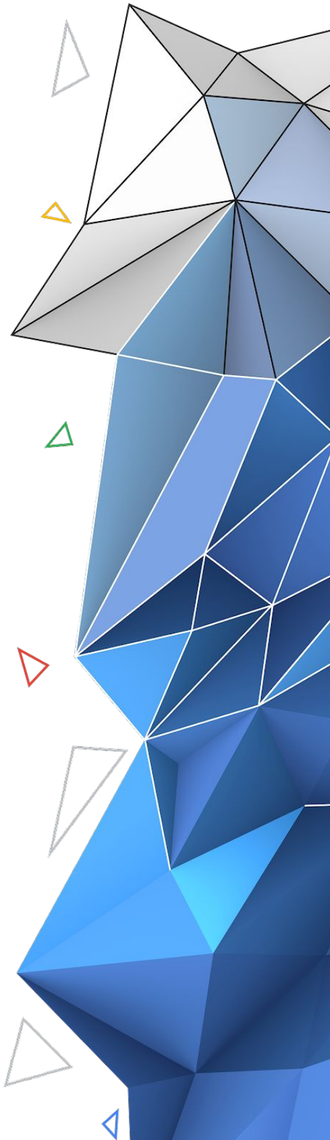
What to Do When You Get Stuck

- **Explain problem** to leadership
- Explain the **value** and the **cost**
  - I'll **learn Burst** and jobs
  - and **make** this **presentation**
- Make **simple test app**, but not too simple



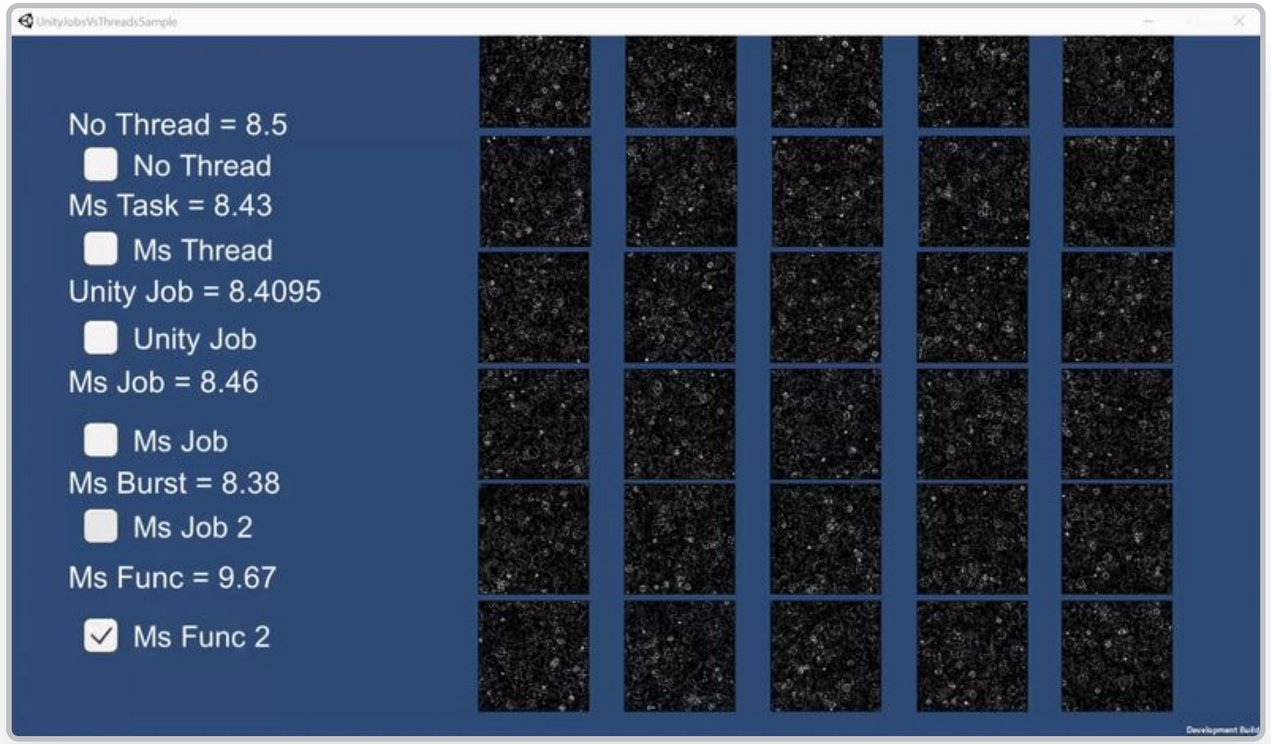
04.

Create Test App



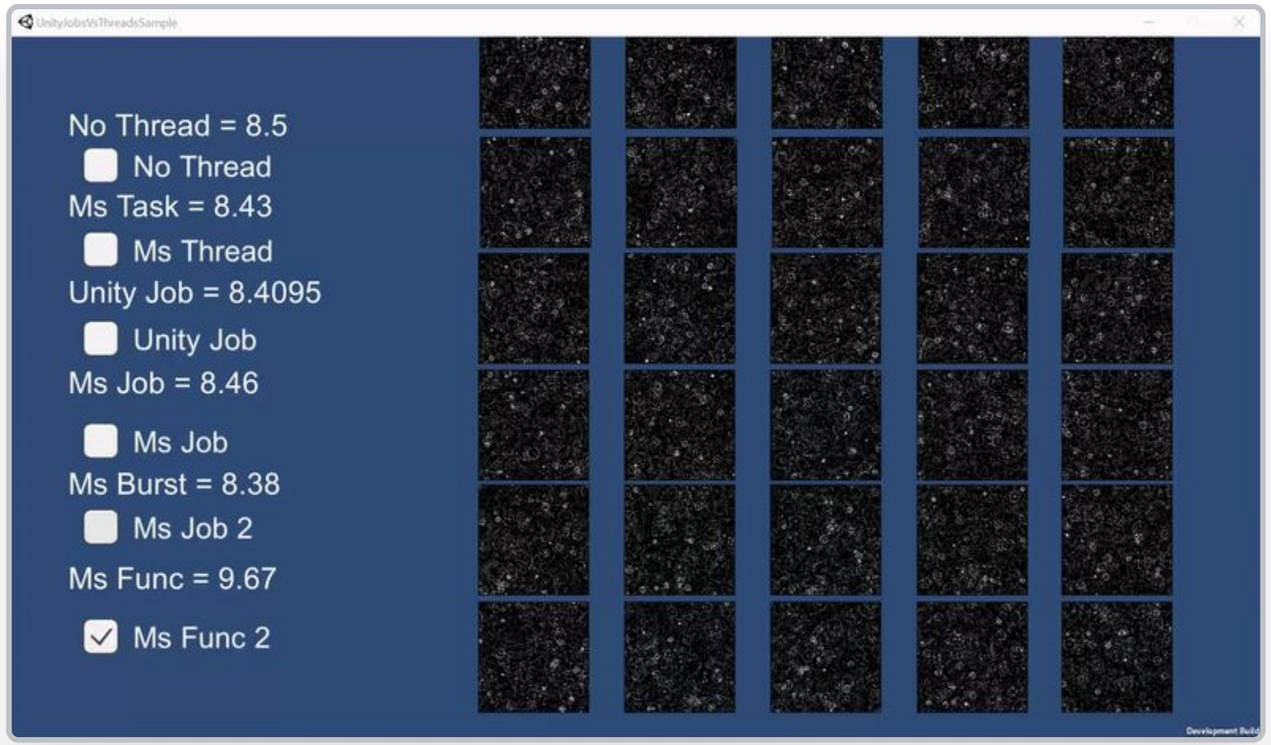
Rain Drops

- Sample that **generated textures** on the CPU
- Started with **one job** or thread per texture
- Then tried **multiple jobs** and dependency



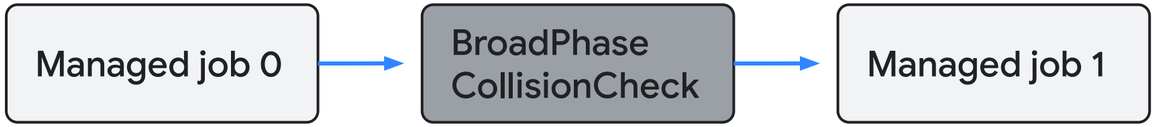
Rain Drops

- Sample that **generated textures** on the CPU
- Started with **one job** or thread per texture
- Then tried **multiple jobs** and dependency



Could I do this?

Fail #3, Model of Stadia Adventures Was Too Simple

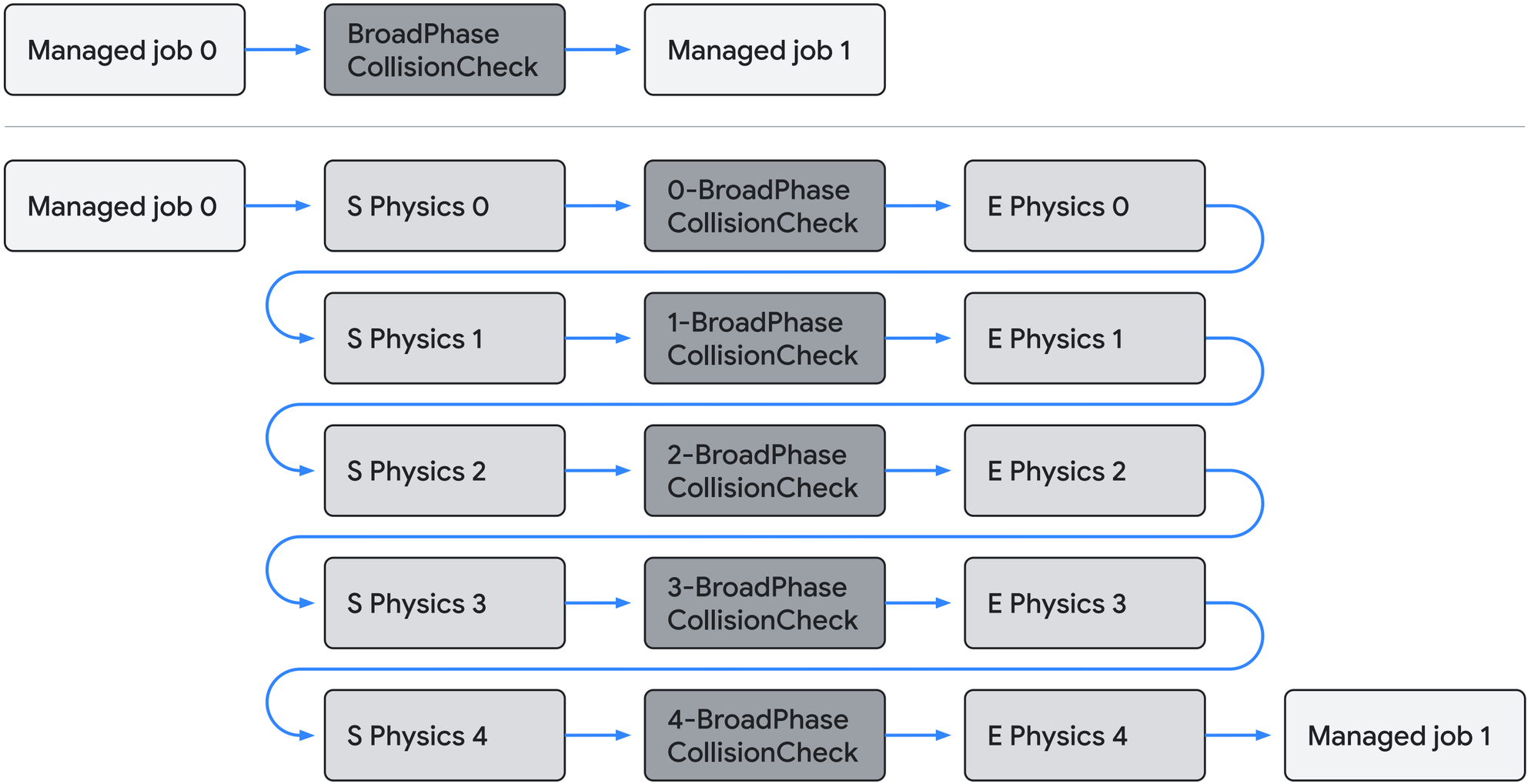


Fail #3, Model of Stadia Adventures Was Too Simple

Not this!

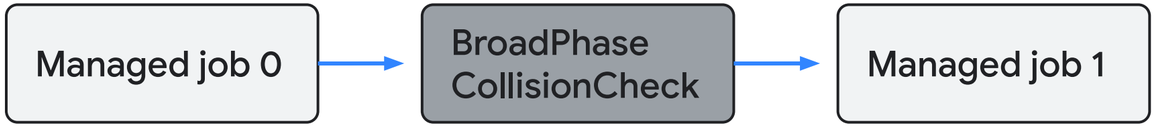
Fail #3, Model of Stadia Adventures Was Too Simple

Not this!

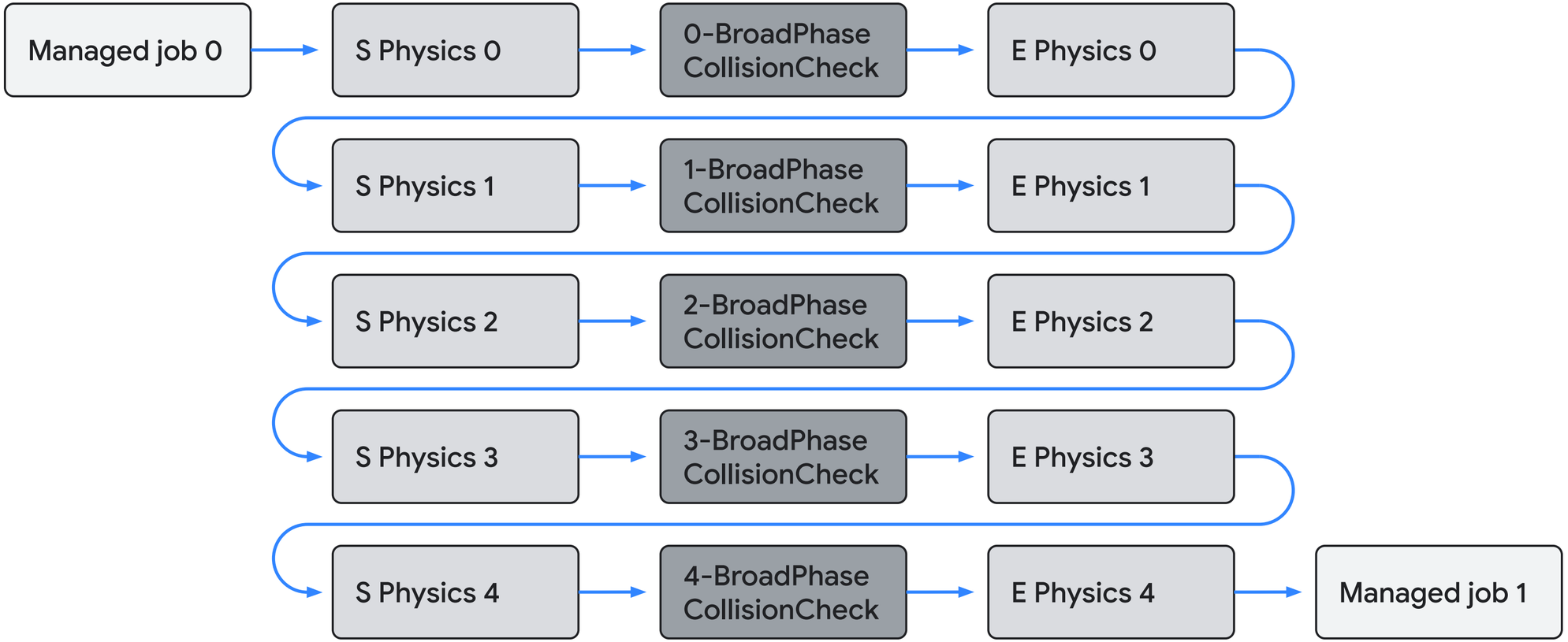


Fail #3, Model of Stadia Adventures Was Too Simple

Not this!



But this!

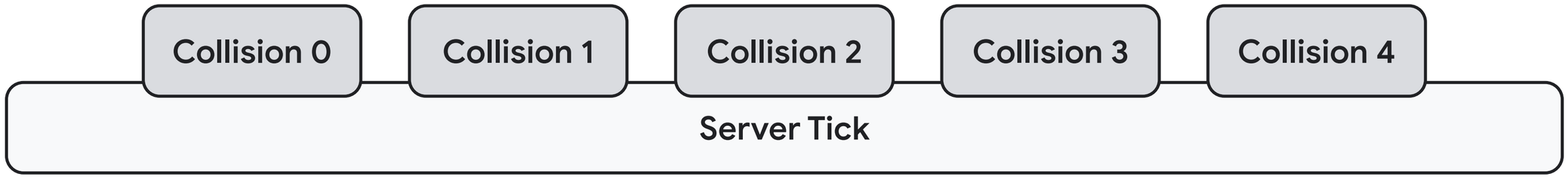


05.

Separating Burst from Jobs

Found Out About Burst.CompileFunctionPointer

- I have a **friend** who works on **Job System team** (Hi Kevin)
- He said look at **Burst.CompileFunctionPointer**
- Could use **Burst without jobs!**
- The Server Tick could just **call a Burst function**



Burst.CompileFunctionPointer

- Need to use a **static function** pass in all **context as parameter**
- Require **“unsafe”** code
  - Very worried!
  - Very scary!
  - Turned out to be not a big deal
- Not async, so it is **harder to use extra threads**
  - But **not needed** for this problem
- **Not much is written** about Burst.CompileFunctionPointer so a lot of **trial and error** was used

06.

Results

Resulting code

Structure Managed Job Calling a Burst Function

```
DropPoolManaged
GCHandle m_dropPoolBurstCallHandle;
DropPoolBurstCall m_dropPoolBurstCall;
ExecuteStaticDelegate m_FunctionPtr;
```


```
DropPoolBurstCall
NativeArray<float> buffer1;
NativeArray<float> buffer2;
NativeArray<Color32> output;
NativeArray<int> frame;
NativeArray<Unity.Mathematics.Random> random;
```

Structure Managed Job Calling a Burst Function

```
DropPoolManaged
GCHandle m_dropPoolBurstCallHandle;
DropPoolBurstCall m_dropPoolBurstCall;
ExecuteStaticDelegate m_FunctionPtr;
```

Has A

```
DropPoolBurstCall
NativeArray<float> buffer1;
NativeArray<float> buffer2;
NativeArray<Color32> output;
NativeArray<int> frame;
NativeArray<Unity.Mathematics.Random> random;
```

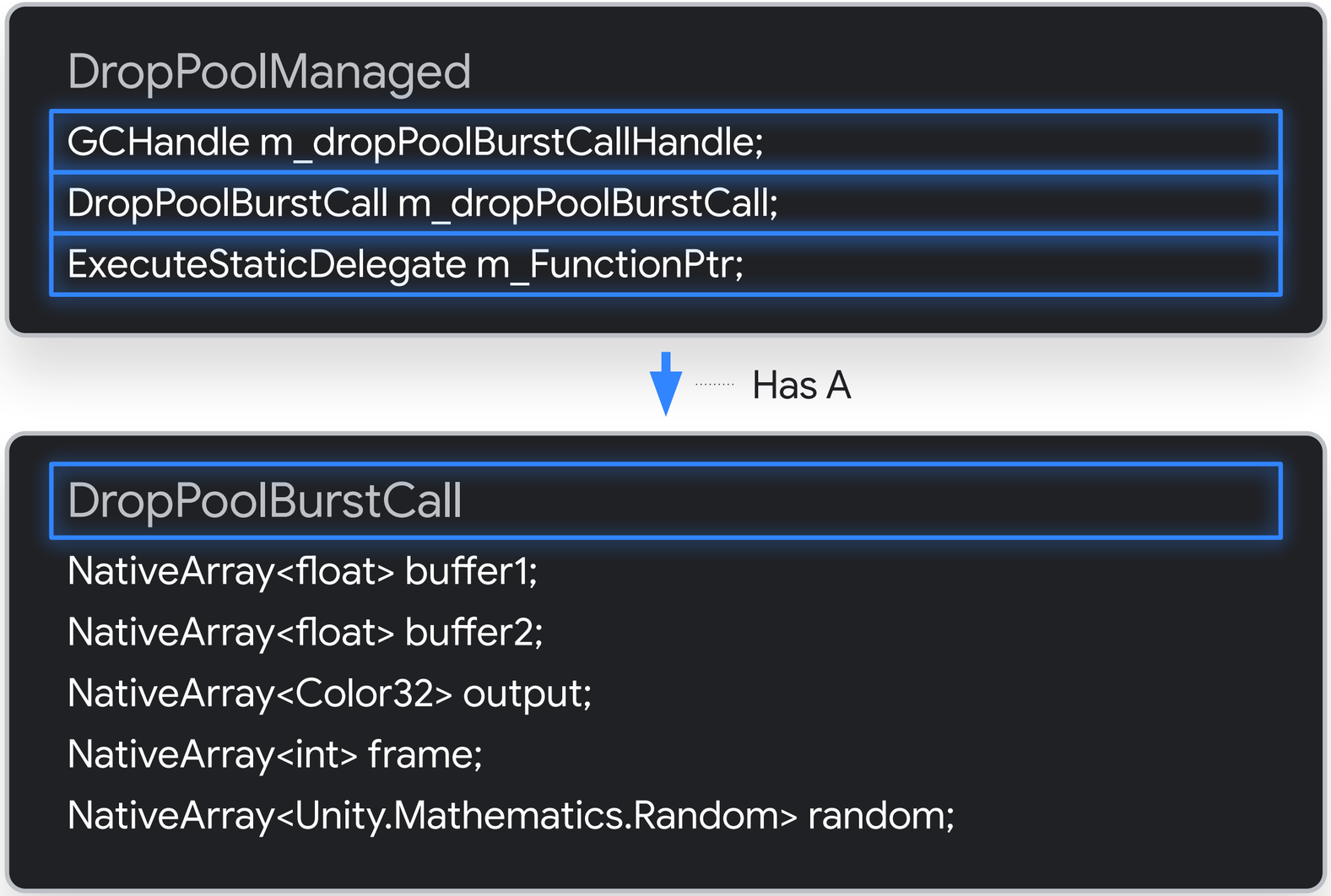
Structure Managed Job Calling a Burst Function

```
DropPoolManaged
GCHandle m_dropPoolBurstCallHandle;
DropPoolBurstCall m_dropPoolBurstCall;
ExecuteStaticDelegate m_FunctionPtr;
```

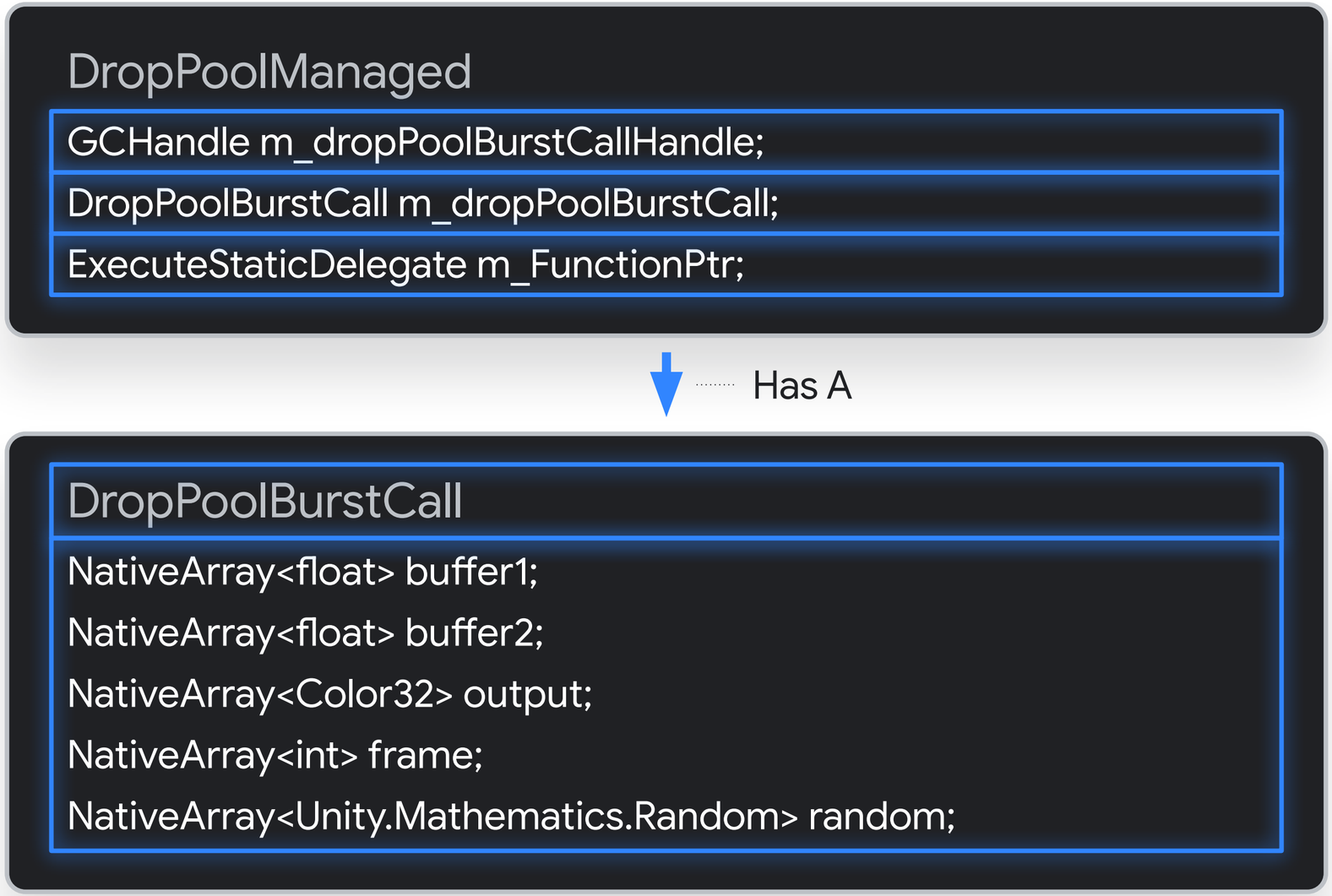
Has A

```
DropPoolBurstCall
NativeArray<float> buffer1;
NativeArray<float> buffer2;
NativeArray<Color32> output;
NativeArray<int> frame;
NativeArray<Unity.Mathematics.Random> random;
```

Structure Managed Job Calling a Burst Function



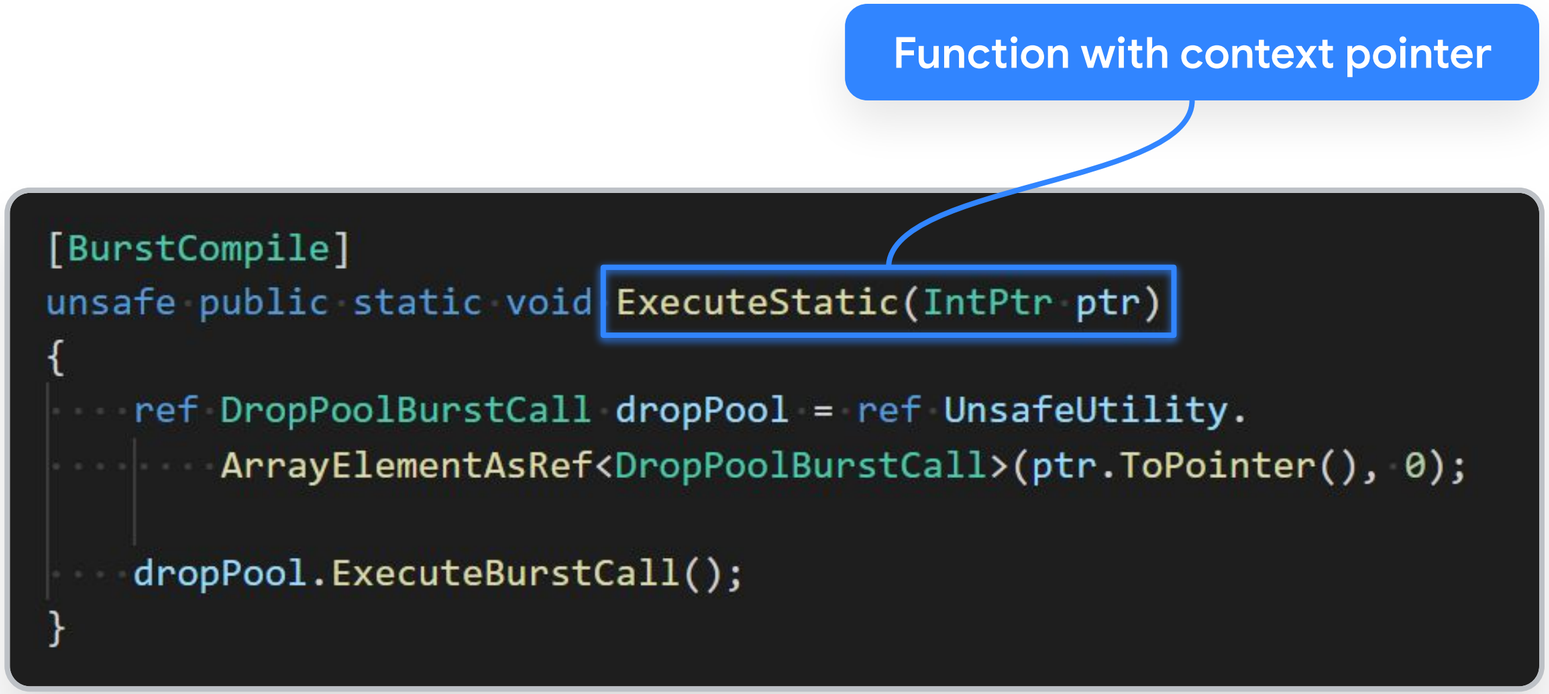
Structure Managed Job Calling a Burst Function



The Burst Function

```
[BurstCompile]
unsafe public static void ExecuteStatic(IntPtr ptr)
{
    ref DropPoolBurstCall dropPool = ref UnsafeUtility.
        ArrayElementAsRef<DropPoolBurstCall>(ptr.ToPointer(), 0);

    dropPool.ExecuteBurstCall();
}
```


The Burst Function



The Burst Function

Function with context pointer

Get the 0th element from the pointer as a ref

```
[BurstCompile]
unsafe public static void ExecuteStatic(IntPtr ptr)
{
    ref DropPoolBurstCall dropPool = ref UnsafeUtility.
        ArrayElementAsRef<DropPoolBurstCall>(ptr.ToPointer(), 0);

    dropPool.ExecuteBurstCall();
}
```


The Burst Function

Function with context pointer

Get the 0th element from the pointer as a ref

```
[BurstCompile]
unsafe public static void ExecuteStatic(IntPtr ptr)
{
    ref DropPoolBurstCall dropPool = ref UnsafeUtility.
        ArrayElementAsRef<DropPoolBurstCall>(ptr.ToPointer(), 0);

    dropPool.ExecuteBurstCall();
}
```

Update all of the NativeArrays

Managed Job Calling a Burst Function

```
public DropPoolManaged()
{
    m_dropPoolBurstCall.Init();

    m_dropPoolBurstCallHandle = GCHandle.Alloc(m_dropPoolBurstCall, GCHandleType.Pinned);

    // Create a Invoke able function pointer
    m_FunctionPtr = BurstCompiler.CompileFunctionPointer<ExecuteStaticDelegate>(DropPoolBurstCall.ExecuteStatic).Invoke;
}

public void Execute()
{
    m_timer.Restart();

    m_FunctionPtr(m_dropPoolBurstCallHandle.AddrOfPinnedObject());

    m_timer.Stop();
}
```


Managed Job Calling a Burst Function

Init all of the NativeArrays for my rain pools

```
public DropPoolManaged()
{
    m_dropPoolBurstCall.Init();

    m_dropPoolBurstCallHandle = GCHandle.Alloc(m_dropPoolBurstCall, GCHandleType.Pinned);

    // Create a Invoke able function pointer
    m_FunctionPtr = BurstCompiler.CompileFunctionPointer<ExecuteStaticDelegate>(DropPoolBurstCall.ExecuteStatic).Invoke;
}

public void Execute()
{
    m_timer.Restart();

    m_FunctionPtr(m_dropPoolBurstCallHandle.AddrOfPinnedObject());

    m_timer.Stop();
}
```


Managed Job Calling a Burst Function

Init all of the NativeArrays for my rain pools

Pin the GC handle so GC does not move the data while burst is running. Copy operation! But only the NativeArray ptr size!

```
public DropPoolManaged()
{
    m_dropPoolBurstCall.Init();

    m_dropPoolBurstCallHandle = GCHandle.Alloc(m_dropPoolBurstCall, GCHandleType.Pinned);

    // Create a Invoke able function pointer
    m_FunctionPtr = BurstCompiler.CompileFunctionPointer<ExecuteStaticDelegate>(DropPoolBurstCall.ExecuteStatic).Invoke;
}

public void Execute()
{
    m_timer.Restart();

    m_FunctionPtr(m_dropPoolBurstCallHandle.AddrOfPinnedObject());

    m_timer.Stop();
}
```


Managed Job Calling a Burst Function

Init all of the NativeArrays for my rain pools

Pin the GC handle so GC does not move the data while burst is running. Copy operation! But only the NativeArray ptr size!

Create a function pointer to my burst static function

```
public DropPoolManaged()
{
    m_dropPoolBurstCall.Init();

    m_dropPoolBurstCallHandle = GCHandle.Alloc(m_dropPoolBurstCall, GCHandleType.Pinned);

    // Create a Invoke able function pointer
    m_FunctionPtr = BurstCompiler.CompileFunctionPointer<ExecuteStaticDelegate>(DropPoolBurstCall.ExecuteStatic).Invoke;
}

public void Execute()
{
    m_timer.Restart();

    m_FunctionPtr(m_dropPoolBurstCallHandle.AddrOfPinnedObject());

    m_timer.Stop();
}
```


Managed Job Calling a Burst Function

Init all of the NativeArrays for my rain pools

Pin the GC handle so GC does not move the data while burst is running. Copy operation! But only the NativeArray ptr size!

Create a function pointer to my burst static function

Managed job code before calling burst function pointer

```
public DropPoolManaged()
{
    m_dropPoolBurstCall.Init();

    m_dropPoolBurstCallHandle = GCHandle.Alloc(m_dropPoolBurstCall, GCHandleType.Pinned);

    // Create a Invoke able function pointer
    m_FunctionPtr = BurstCompiler.CompileFunctionPointer<ExecuteStaticDelegate>(DropPoolBurstCall.ExecuteStatic).Invoke;
}

public void Execute()
{
    m_timer.Restart();

    m_FunctionPtr(m_dropPoolBurstCallHandle.AddrOfPinnedObject());

    m_timer.Stop();
}
```


Managed Job Calling a Burst Function

Init all of the NativeArrays for my rain pools

Pin the GC handle so GC does not move the data while burst is running. Copy operation! But only the NativeArray ptr size!

Create a function pointer to my burst static function

Managed job code before calling burst function pointer

Call burst function and pass in the pinned global context

```
public DropPoolManaged()
{
    m_dropPoolBurstCall.Init();
    m_dropPoolBurstCallHandle = GCHandle.Alloc(m_dropPoolBurstCall, GCHandleType.Pinned);

    // Create a Invoke able function pointer
    m_FunctionPtr = BurstCompiler.CompileFunctionPointer<ExecuteStaticDelegate>(DropPoolBurstCall.ExecuteStatic).Invoke;
}

public void Execute()
{
    m_timer.Restart();
    m_FunctionPtr(m_dropPoolBurstCallHandle.AddrOfPinnedObject());

    m_timer.Stop();
}
```

Managed Job Calling a Burst Function

Init all of the NativeArrays for my rain pools

Pin the GC handle so GC does not move the data while burst is running. Copy operation! But only the NativeArray ptr size!

Create a function pointer to my burst static function

Managed job code before calling burst function pointer

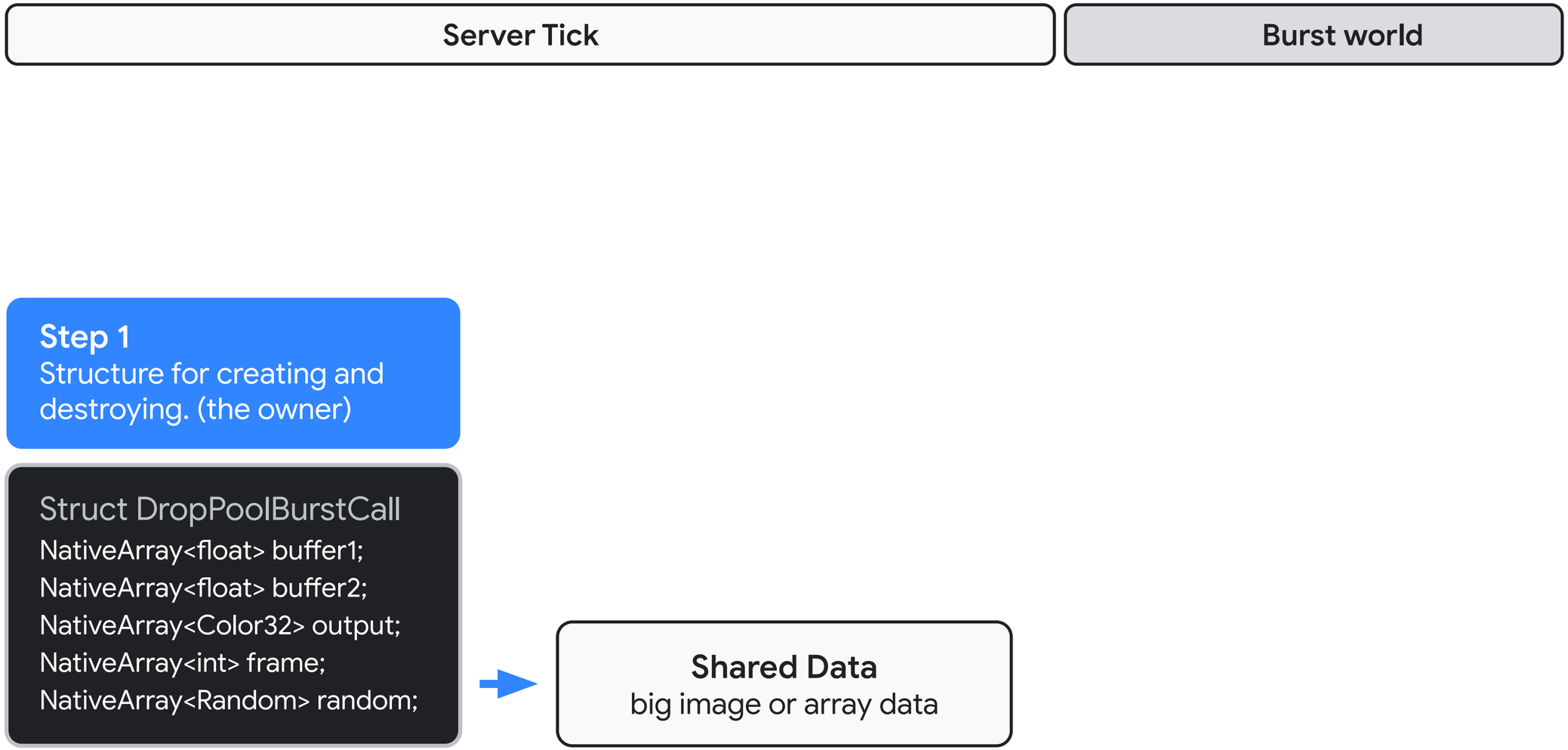
Call burst function and pass in the pinned global context

After burst function call copy data to the managed world

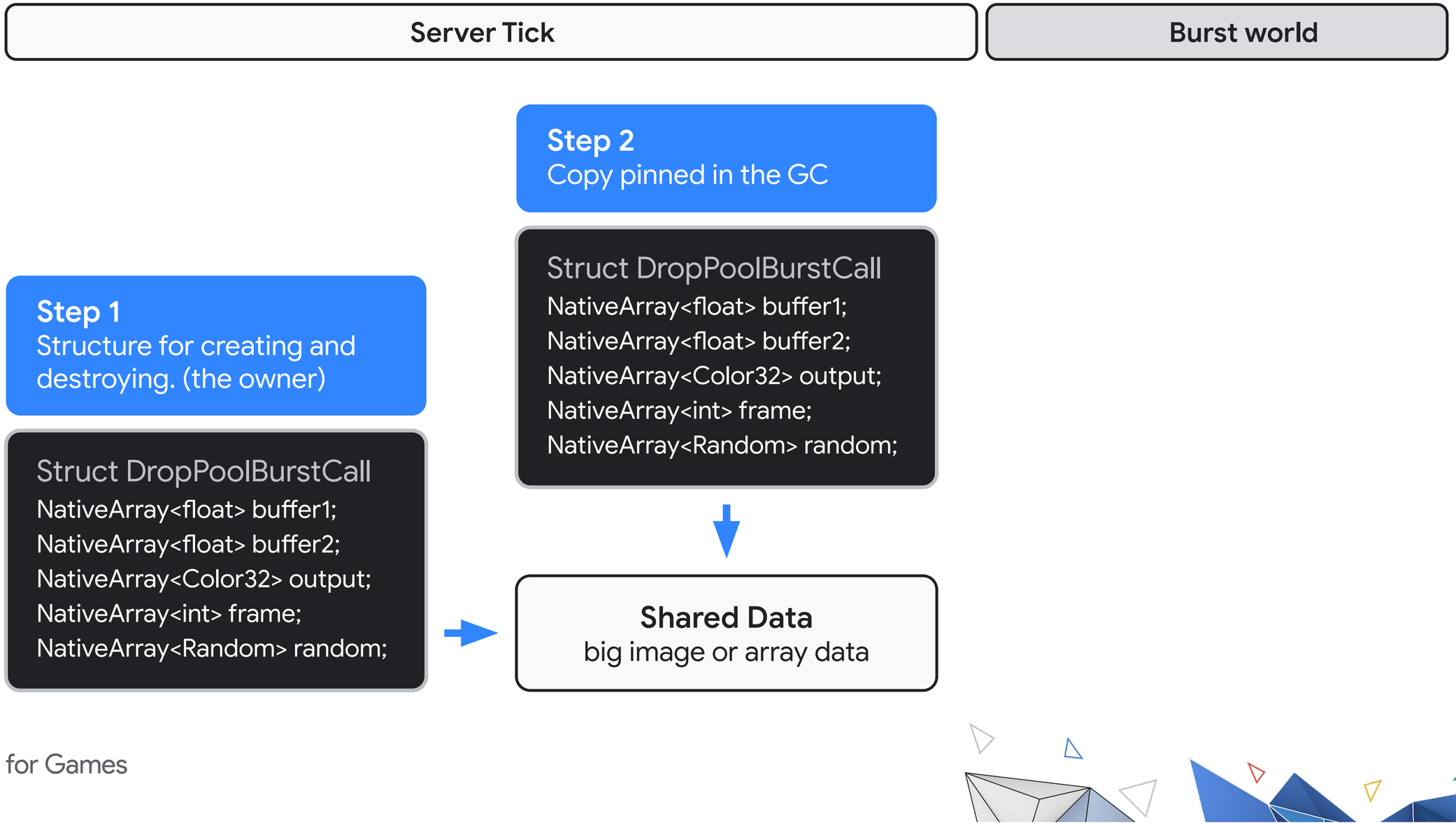
```
public DropPoolManaged()
{
    m_dropPoolBurstCall.Init();
    m_dropPoolBurstCallHandle = GCHandle.Alloc(m_dropPoolBurstCall, GCHandleType.Pinned);

    // Create a Invoke able function pointer
    m_FunctionPtr = BurstCompiler.CompileFunctionPointer<ExecuteStaticDelegate>(DropPoolBurstCall.ExecuteStatic).Invoke;
}

public void Execute()
{
    m_timer.Restart();
    m_FunctionPtr(m_dropPoolBurstCallHandle.AddrOfPinnedObject());
    m_timer.Stop();
}
```

Is There a Better Way Alloc My Struct? Probably?

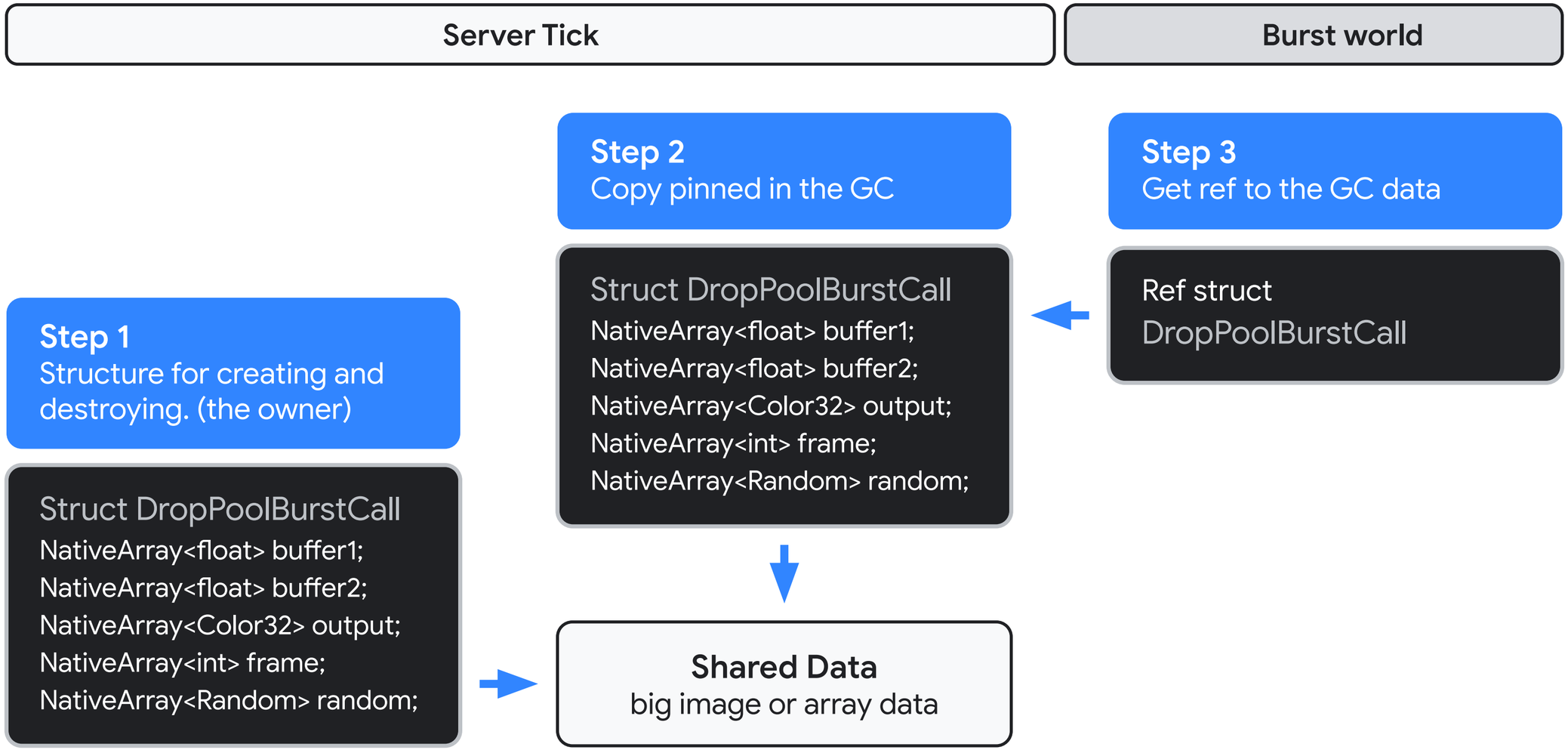
Is There a Better Way Alloc My Struct? Probably?



Is There a Better Way Alloc My Struct? Probably?



Is There a Better Way Alloc My Struct? Probably?



Burst Function Dispose

```
public void Dispose()
{
    m_dropPoolBurstCall.Dispose();

    m_dropPoolBurstCallHandle.Free();
}
```

```
public void Dispose()
{
    IntPtr ptr = m_dropPoolBurstCallHandle.AddrOfPinnedObject();
    unsafe {
        ref DropPoolBurstCall dropPool = ref UnsafeUtility.
                ArrayElementAsRef<DropPoolBurstCall>(ptr.ToPointer(), 0);
        dropPool.Dispose();
    }
    m_dropPoolBurstCallHandle.Free();
}
```


Burst Function Dispose

free NativeArray owned data (big images)

```
public void Dispose()
{
    m_dropPoolBurstCall.Dispose();

    m_dropPoolBurstCallHandle.Free();
}
```

```
public void Dispose()
{
    IntPtr ptr = m_dropPoolBurstCallHandle.AddrOfPinnedObject();
    unsafe {
        ref DropPoolBurstCall dropPool = ref UnsafeUtility.
            ArrayElementAsRef<DropPoolBurstCall>(ptr.ToPointer(), 0);
        dropPool.Dispose();
    }
    m_dropPoolBurstCallHandle.Free();
}
```


Burst Function Dispose

free NativeArray owned data (big images)

Unpin the GC data (ptr and sizes)

```
public void Dispose()
{
    m_dropPoolBurstCall.Dispose();

    m_dropPoolBurstCallHandle.Free();
}
```

```
public void Dispose()
{
    IntPtr ptr = m_dropPoolBurstCallHandle.AddrOfPinnedObject();
    unsafe {
        ref DropPoolBurstCall dropPool = ref UnsafeUtility.
            ArrayElementAsRef<DropPoolBurstCall>(ptr.ToPointer(), 0);
        dropPool.Dispose();
    }
    m_dropPoolBurstCallHandle.Free();
}
```


Burst Function Dispose

free NativeArray owned data (big images)

Unpin the GC data (ptr and sizes)

```
public void Dispose()
{
    m_dropPoolBurstCall.Dispose();

    m_dropPoolBurstCallHandle.Free();
}
```

VS

```
public void Dispose()
{
    IntPtr ptr = m_dropPoolBurstCallHandle.AddrOfPinnedObject();
    unsafe {
        ref DropPoolBurstCall dropPool = ref UnsafeUtility.
            ArrayElementAsRef<DropPoolBurstCall>(ptr.ToPointer(), 0);
        dropPool.Dispose();
    }
    m_dropPoolBurstCallHandle.Free();
}
```


Burst Function Dispose

free NativeArray owned data (big images)

Unpin the GC data (ptr and sizes)

free NativeArray owned data (big images) using ref GC data

```
public void Dispose()
{
    m_dropPoolBurstCall.Dispose();

    m_dropPoolBurstCallHandle.Free();
}
```

VS

```
public void Dispose()
{
    IntPtr ptr = m_dropPoolBurstCallHandle.AddrOfPinnedObject();
    unsafe {
        ref DropPoolBurstCall dropPool = ref UnsafeUtility.
            ArrayElementAsRef<DropPoolBurstCall>(ptr.ToPointer(), 0);
        dropPool.Dispose();
    }
    m_dropPoolBurstCallHandle.Free();
}
```


Burst Function Dispose

free NativeArray owned data (big images)

Unpin the GC data (ptr and sizes)

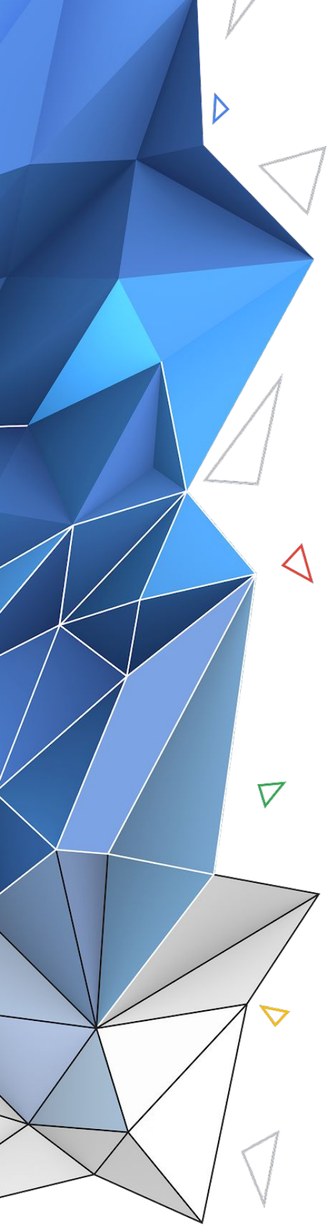
```
public void Dispose()
{
    m_dropPoolBurstCall.Dispose();

    m_dropPoolBurstCallHandle.Free();
}
```

VS

free NativeArray owned data (big images) using ref GC data

UNKNOWN_OBJECT_TYPE has been deallocated, it is not allowed to access it

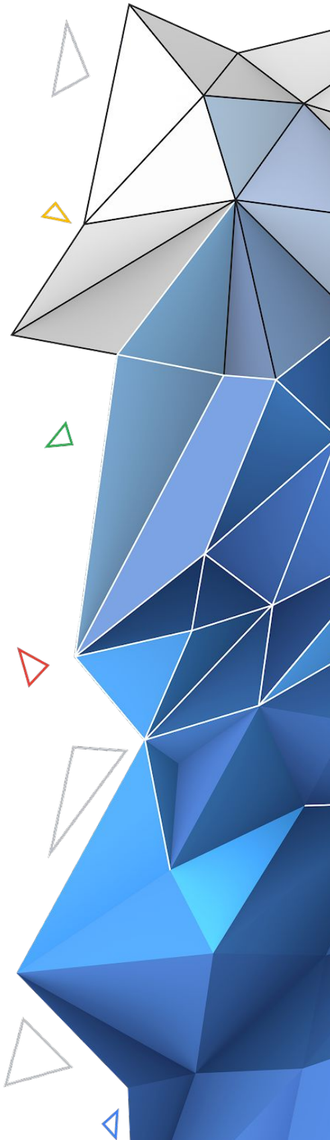
```
public void Dispose()
{
    IntPtr ptr = m_dropPoolBurstCallHandle.AddrOfPinnedObject();
    unsafe {
        ref DropPoolBurstCall dropPool = ref UnsafeUtility.
            ArrayElementAsRef<DropPoolBurstCall>(ptr.ToPointer(), 0);
        dropPool.Dispose();
    }
    m_dropPoolBurstCallHandle.Free();
}
```

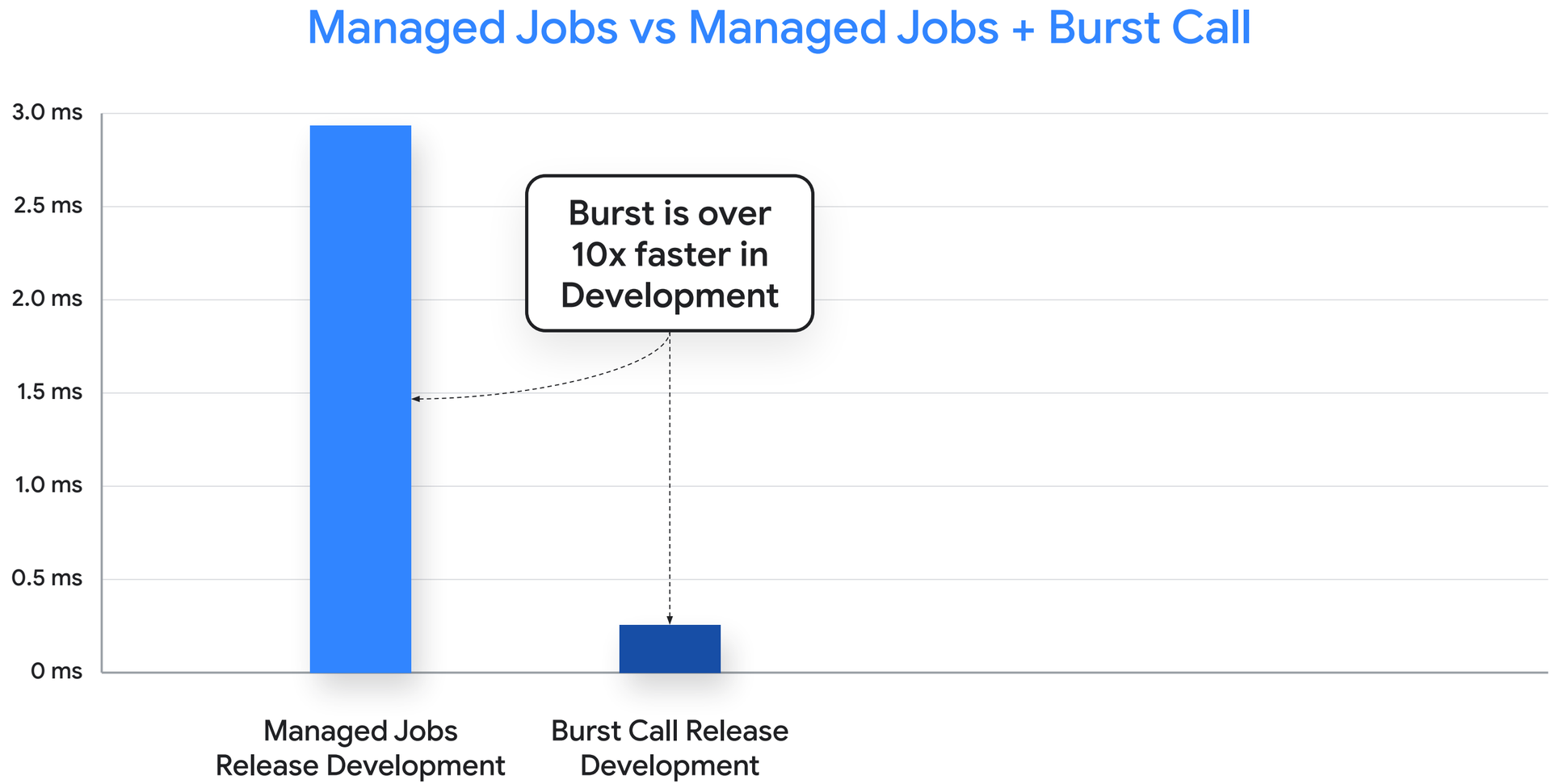
06.

Results

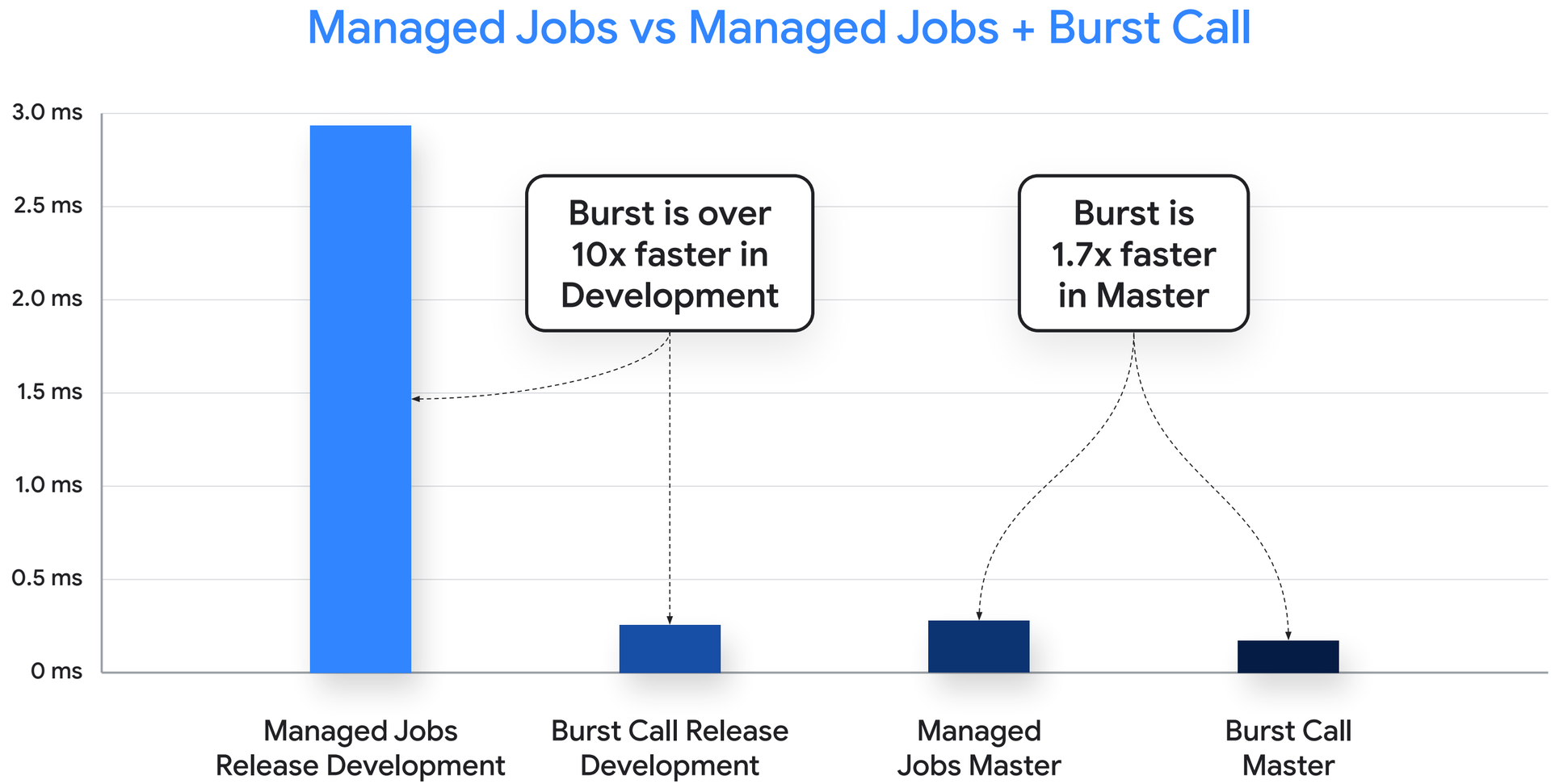
Profiling results



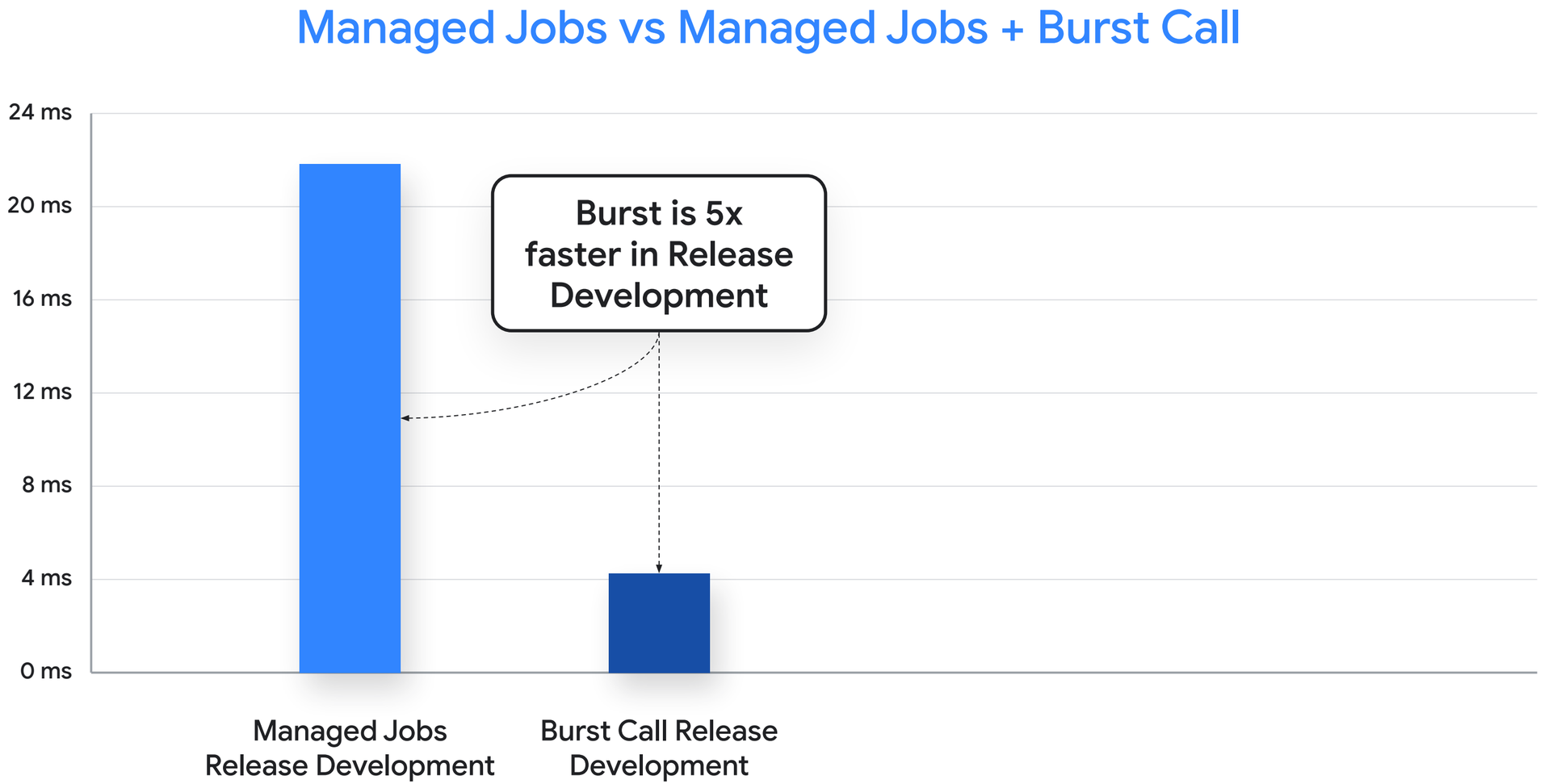
Stadia Adventure BroadPhaseCollisionCheck



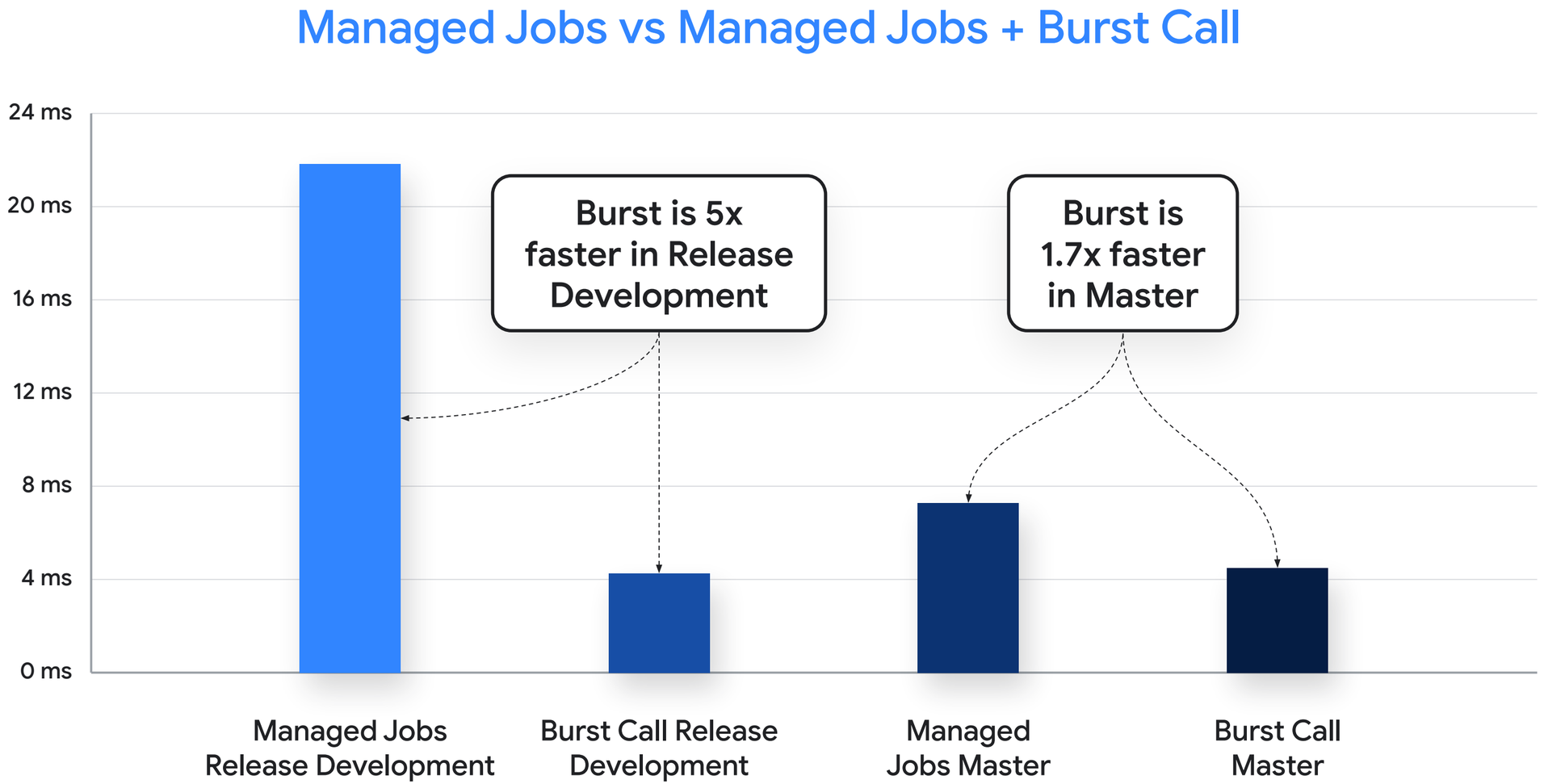
Stadia Adventure BroadPhaseCollisionCheck



RainDrop Updating Pool



RainDrop Updating Pool



07.

Summary

Summary

- Managed code in Development builds **can be faster 25-50x**
  - **Unity** has a **5x** faster fix for Release Development build
    - Get the barrier fix! (2020.3.26, 2021.2.8, 2022.1.0b3, 2022.2.0a1)
  - **Burst** gives another **5x to 10x**
- Create test versions of tricky problems:
  - Make a test app, but **don’t simplify** your model **too much**

Summary

Decoupling Burst and jobs

- Use **Burst.CompileFunctionPointer**
- Used **UnsafeUtility.ReadArrayElement<T>** to reference context data
- Used **unsafe code** and **pinning** memory in the GC
- GC data **pinning makes a copy** but only your NativeArray's pointers and sizes
- Can run **any C# managed code in jobs** with some effort and pinning GC

Thank you!

Scott Wardle

Senior Software Engineer, Google Stadia

scottwardle@google.com

Google for Games

Rain Drop sample code

GitHub link

https://github.com/swardle/unity-jobs-vs-ms-threads-sample