# VideoGames Profiling and Beyond

ЛЕСТА

# Что мы делаем?
# Разрабатываем игры!

**Вызовы:**

- Проекты развиваются, требования растут
- Внедрять улучшения необходимо «на живую»

**Особенности:**

- Огромная аудитория с разнообразным железом
- Наша задача – максимально оптимизировать производительность для всех устройств от МСТ до High-end ПК

# Игра – это система реального времени

## Что это значит:

- Все процессы (графика, физика, логика) должны выполняться в строго определённые временные рамки для создания плавного игрового процесса

## Показатели производительности:

- Минимальный FPS: 20 для базовой иллюзии движения
- Оптимальный FPS:  30, 60, а лучше – выше

## Зачем это нужно:

- Чтобы игроки не сталкивались с лагами, фризами и другими проблемами, которые могут испортить впечатление от игры

© «Мстители: Финал», реж. Энтони Руссо, Джо Руссо, правообладатель «Марвел Кэриктес, Инк., корпорация штата Делавэр»

# Особенности разработки

**На производительность влияют:**

- Изменения в коде
- Изменения в контенте

**Важно!**

- На каждом этапе изменений оценивать возможное влияние на производительность
- Как можно раньше детектировать проблему

# Пользователи профайлеров

## Художники

- FPS
- Объекты
- Draw Calls
- Размеры текстур
- Эффекты
- …

## Программисты

- Анализ вызовов
- Критические пути
- CPU-GPU синхронизации
- Спайки (фризы)
- …

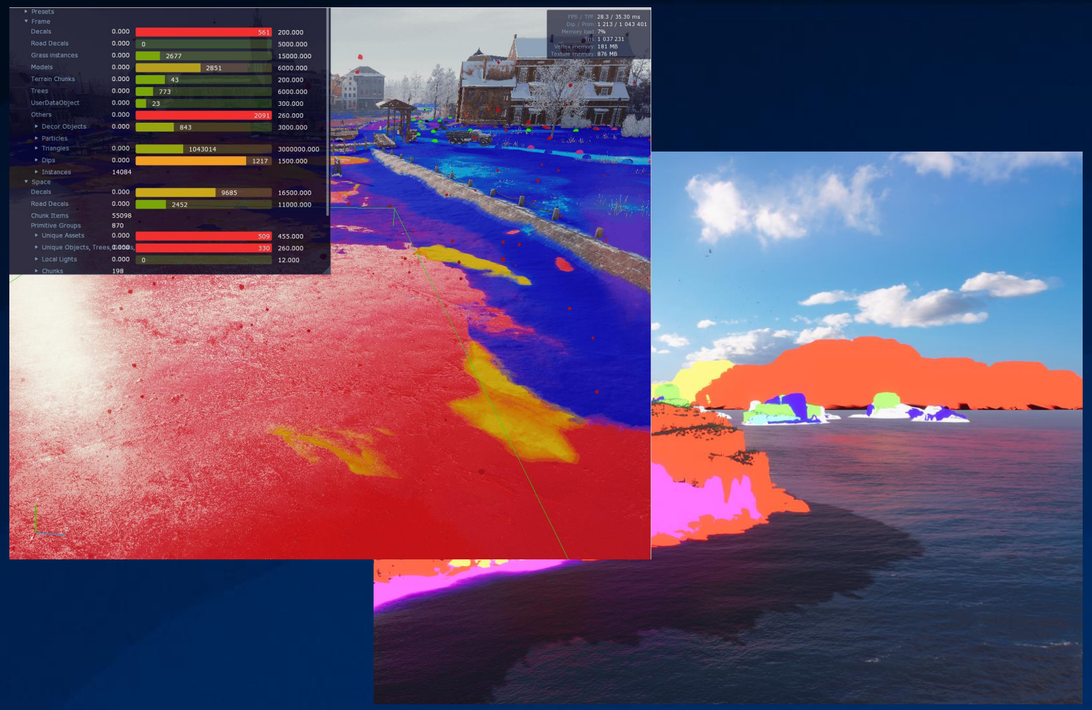
## Performance QA

- Агрегированные данные
- Сравнение прогонов
- Бюджетирование

# Виды профайлеров

## Встроенные в движок

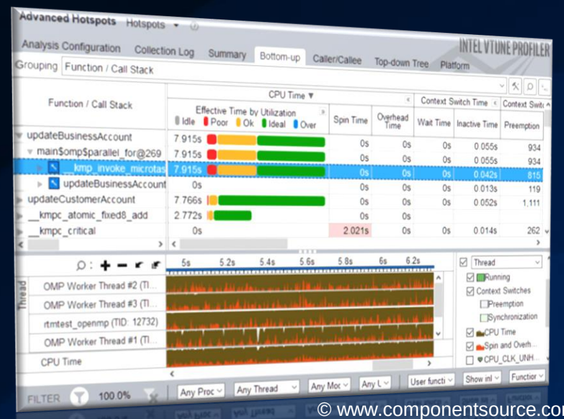
- Различные RealTime счётчики
  - Полигоны
  - FPS
  - Текстуры
  - Память
  - …
- Debug-режимы для разработчиков
- Агрегированные данные со счётчиков



# Виды профайлеров

## Внешние профайлеры

- CPU
  - Visual Studio Profiler
  - Intel Vtune Profiler
  - Perf
  - PySampler
- GPU
  - RenderDoc
  - NVIDIA Nsight Graphics
  - AMD Radeon GPU Profiler
  - Pix




© www.componentsource.com

© developer.nvidia.com

# Instrumented vs Sample-Based

## Sample Based Profiling

### Плюсы

- «Низкая» нагрузка на систему
- Простота использования
- Анализ в реальном времени
- Не требует изменений в коде

### Минусы

- Ограниченная точность
- Неполные данные
- Зависимость от частоты
- Риск неправильной интерпретации
- Проблемы в многопоточных системах

## Instrumented Profiling

### Плюсы

- Высокая точность
- Гибкость
- Поддержка многопоточности
- Возможность сбора контекста
- Идеально для сложных систем
- Интеграция со скриптами

### Минусы

- Более высокий оверхед
- Требует изменений кода

## Что показывает профайлер

- Время, затраченное на каждый аспект кадра: рендеринг, физика, логика, инпут …
- Проблемные участки: ботлнеки, «испорченные» кадры
- Фризы, просадки FPS
- Различные счётчики для верхнеуровневой оценки «на месте»

## Как использовать

- Обнаружение ботлнеков
- Оптимизация ассетов и кода
- Сравнение производительности до и после изменений (diff-based тестирование)

## Тестирование на разнообразном железе

- От «калькуляторов» до high-end машин
- Используем даже старые машины (не всегда легко найти)
- Активно тестируем на стендах, приближенных к реальным условиям

## Мотивация

- Выжимаем максимум из железа, чтобы радовать игроков красивой графикой
- Поддерживаем баланс между производительностью и визуальной составляющей

## Используемые инструменты

- Стресс-тестирование, тестирование контента, реплеи
- Разные профайлеры для разных юзеров/задач (художники, программисты, QA)

# Diff-based тестирование

## Основная идея

- Сравнение производительности между разными запусками (обычно 3 прогона)
- Анализ таймлайна, выделение проблемных участков
- Агрегация данных для получения средних значений

## Что анализируем?

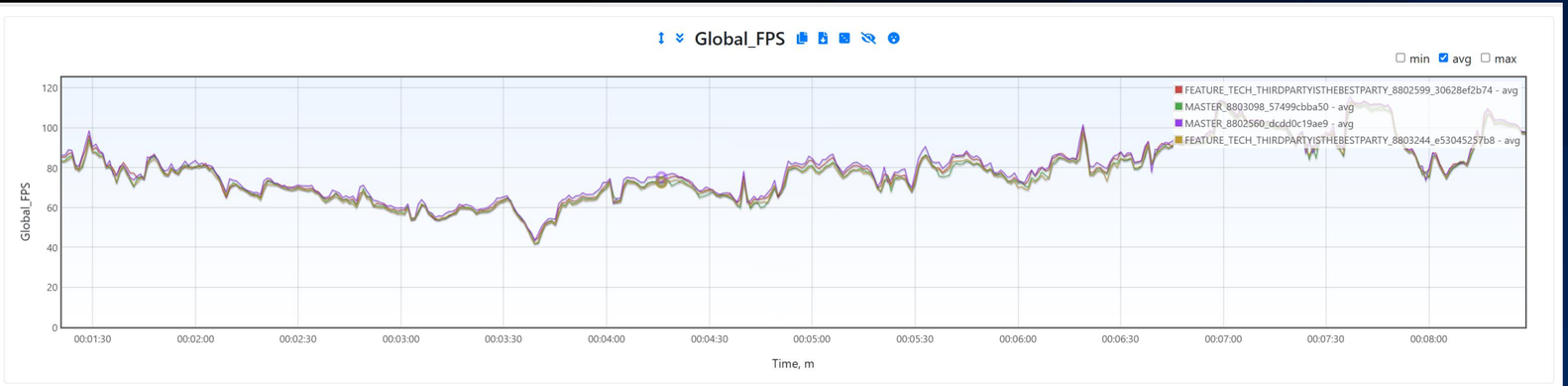
- «Испорченные» кадры (выделяются красным)
- Фризы, просадки FPS, ботлнеки
- Время, затраченное на рендеринг, физику, игровую логику, инпут и т.д.

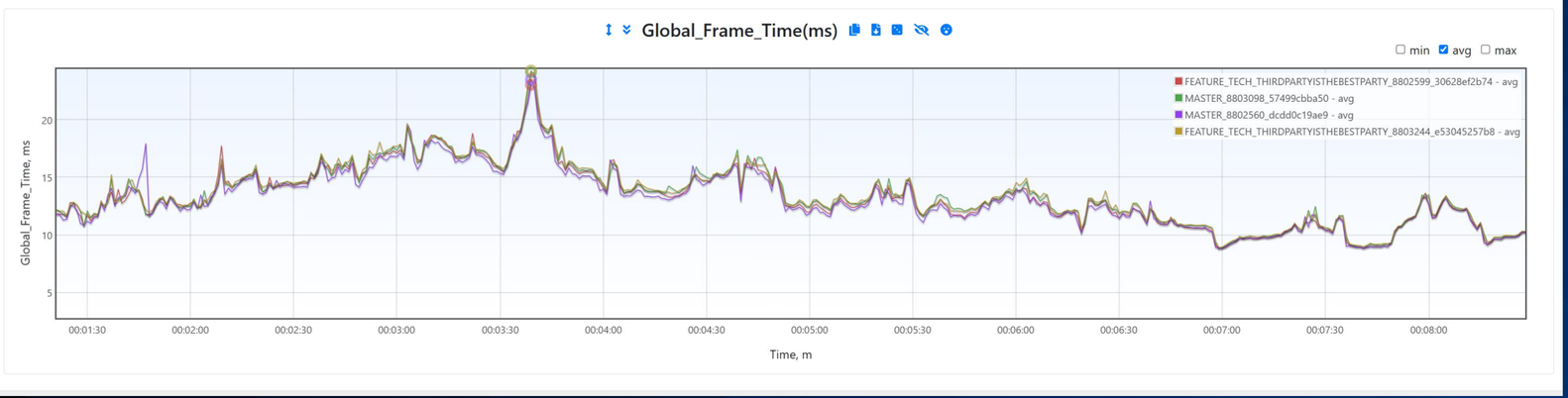
## Пример использования

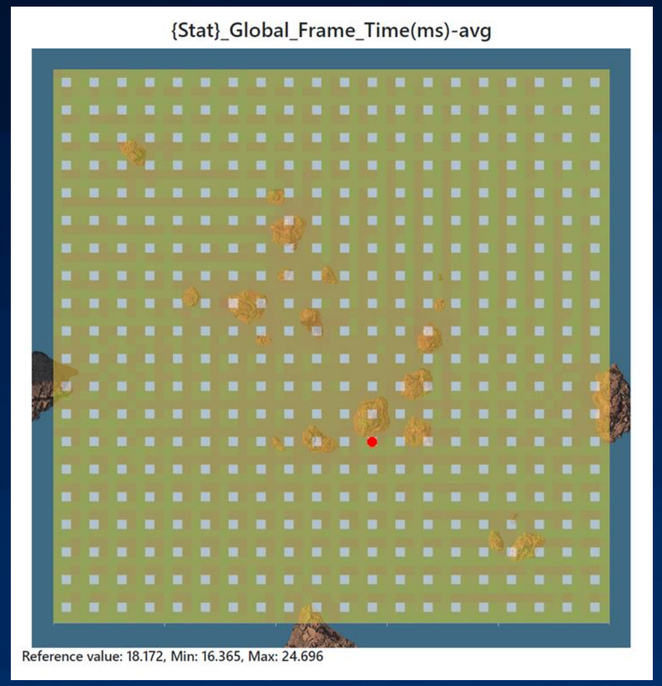
- Сравнение производительности до и после изменений
- Обнаружение проблем на ранних этапах разработки
- Регрессионное тестирование между версиями

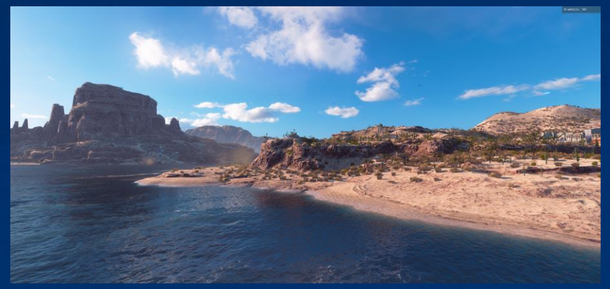
# Примеры отчётов профилирования

## На самом деле, типов намного больше



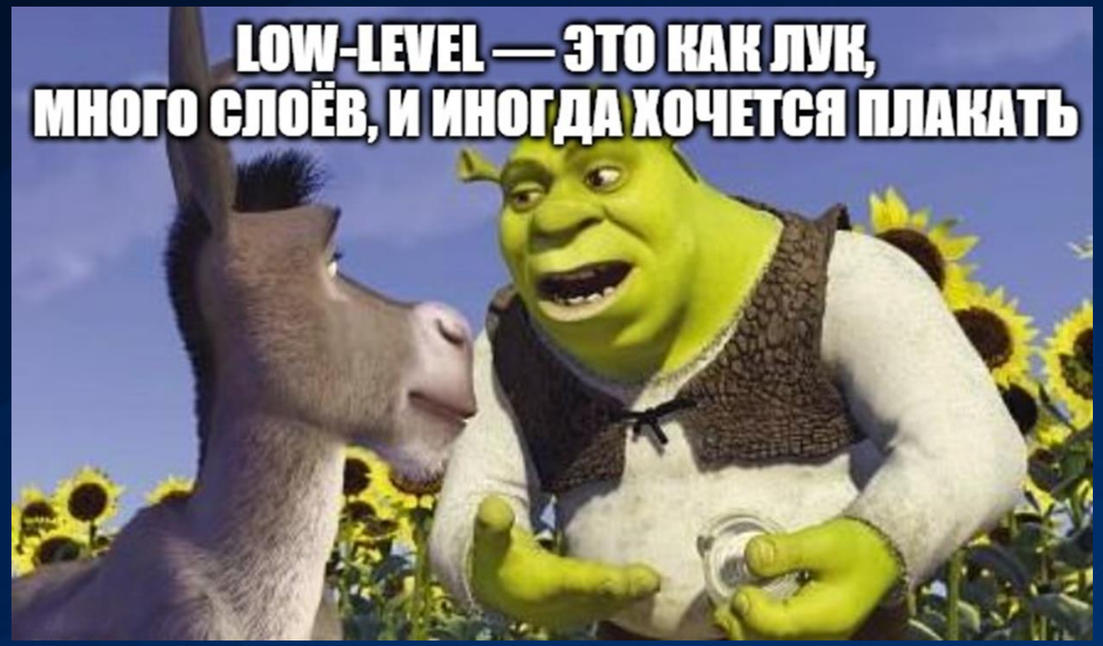



# ЧТО ОБСУДИМ

- Our Profiler Architecture
- CPU Timings
- GPU Profiling
- RAM & VRAM Profiling
- File IO Profiling
- Data Plots




©«Шрек», реж. Эндрю Адамсон, Вики Дженсон, правообладатель «ДримУоркс Анимейшн Л.Л.К.»

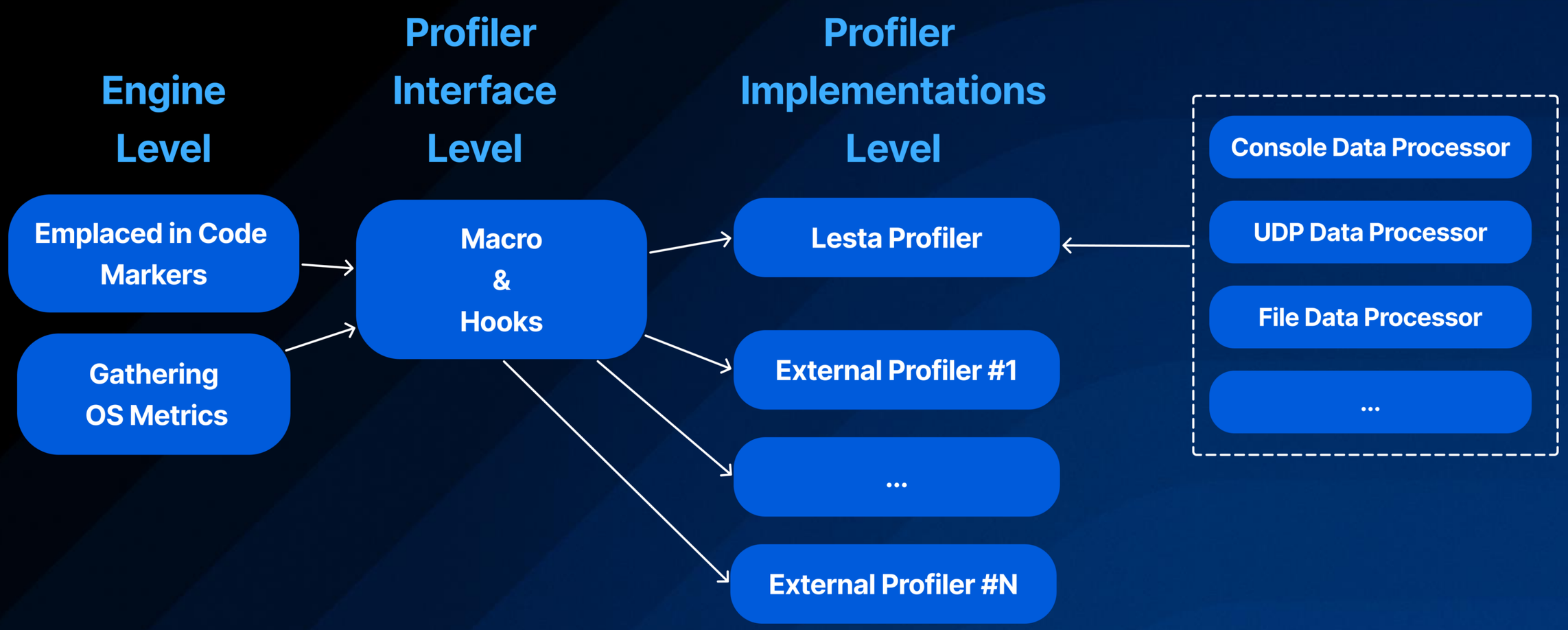
Будет немного Low-Level :)

# Our Profiler Architecture

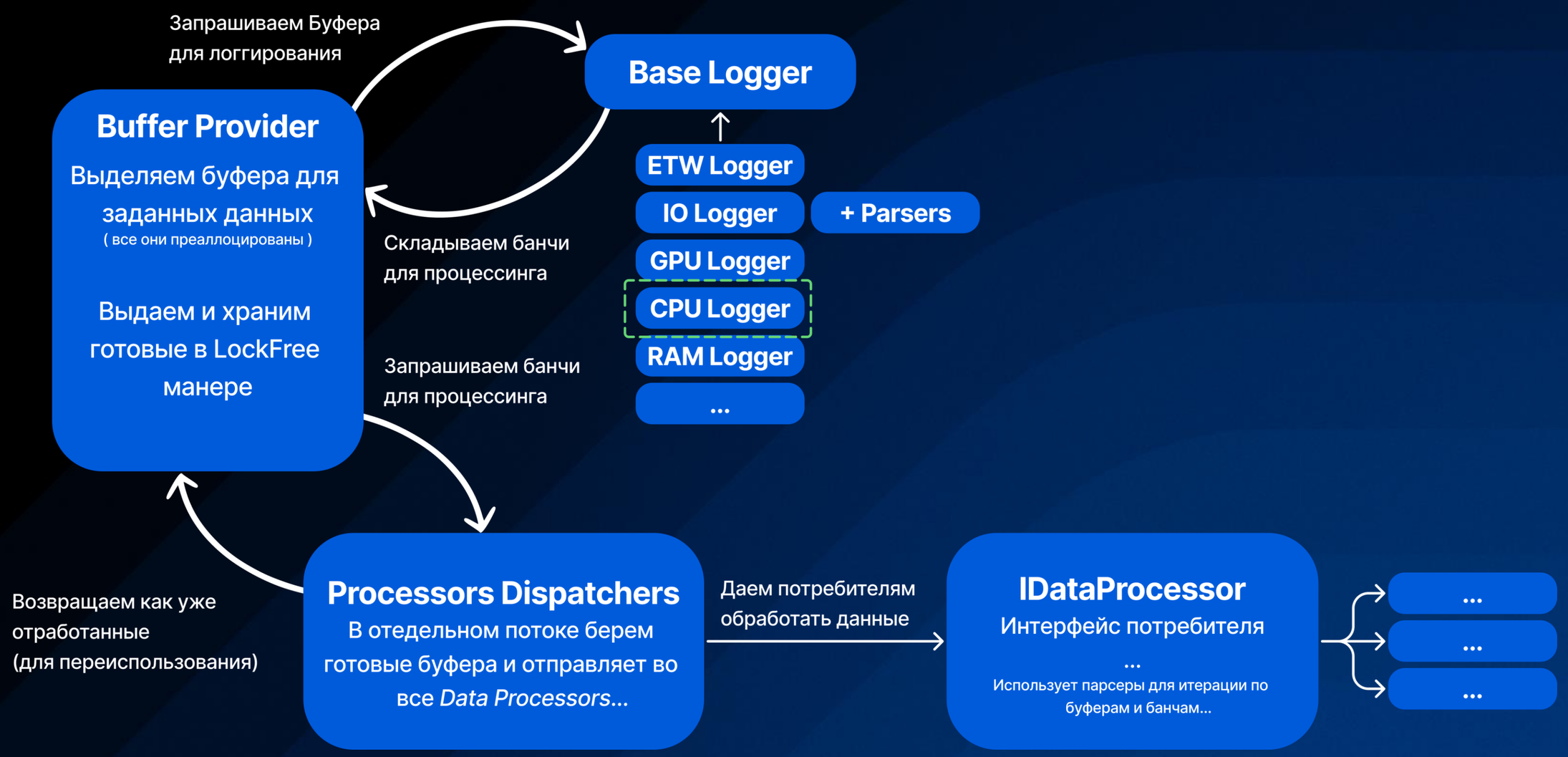
- Сбор данных CPU/GPU Timings, RAM, VRAM, IO и т.д. из различных источников
- Минимальный оверхэд, стараемся не влиять на измеряемые данные
- Разные сценарии использования для разных ситуаций
- Возможность легко подключать внешние профайлеры или запускаться без них

# Global Profiler Architecture

## Instrumented Profiling



# Our Profiler Implementation



# CPU Logger

## CPU Logger Data Structures

```
enum class EEventType : UInt32
{
    ZoneEnter = 0,
    ZoneLeave = 1,
    //...
};

struct EventHeader
{
    EEventType Type;
    TimeStamp TimeStamp;
};

template< EEventType Type >
struct EventPayload;

template<> struct EventPayload<EEventType::ZoneEnter>
{
    StringHandle zoneName;
};
template<> struct EventPayload<EEventType::ZoneLeave>
{
};

// ...
```

## CPU Logger

```
class CPULogger: public ILogger< CPULogger >
{
public:
    template< EEventType Type, typename... PayloadArgs >
    static void WriteEvent( PayloadArgs&&... args )
    {
        void* Ptr = ThreadContext::Get().GetBuffer( sizeof(EventHeader) + sizeof(EventPayload<Type>) );
        WriteStruct<EventHeader>( Ptr, Type, Profiling::GetTickCount() );
        WriteStruct<EventPayload<Type>>( Ptr, Forward<PayloadArgs>(args)... );
    }
    // ...
private:
    struct ThreadContext
    {
        ThreadContext();
        ~ThreadContext();
        // ...

        static ThreadContext& Get()
        {
            static thread_local ThreadContext SThreadContext;
            return SThreadContext;
        }
    };
    static Array< ThreadContext > m_threadContexts;
    static Array< Buffer > m_deadThreadContextData;
};

namespace Profiling
{
    void EnterCPUZone( const char* LiteralName )
    {
        CPULogger::WriteEvent< EEventType::ZoneEnter >( MakeStaticStringHandle( LiteralName ) );
    }
    //...
}
```

# CPU Timings

- Приложение реального времени, требуется выдавать 60 FPS, а лучше – больше
- 60 fps ≈ 16 ms на полную симуляцию игрового мира
- Огромное кол-во вычислений
- Многопоточная среда

# CPU Timings

- Как точно измерять время
- Немного RAII для удобства
- Многопоточность
- Примитивы синхронизации
- ETW
- Переключения контекстов

# Способы измерить точно время

- C++ std::chrono::high_resolution_clock
- WinApi QueryPerformanceCounter + QueryPerformanceFrequency
- Intrinsics __rdtsc

# Как точно измерять время

## RDTSC Frequency Calculation

```
LARGE_INTEGER freq;
QueryPerformanceFrequency( &freq );

LARGE_INTEGER startQpc;
LARGE_INTEGER endQpc;

// Note: RDTSC instruction is not serializing so we should use memory barriers.
MemoryBarrier();
QueryPerformanceCounter( &startQpc );
const UInt64 startTsc = __rdtsc();
MemoryBarrier();

// Spin 1/32 sec.
do
{
    QueryPerformanceCounter( &endQpc );
}
while ( ( endQpc.QuadPart - startQpc.QuadPart ) < freq.QuadPart / 32 );

MemoryBarrier();
const UInt64 endTsc = __rdtsc();

// Calculate CPU time stamp frequency (g_cpuTscFrequency) here because we can't
// use QPC frequency. They may differ on different platforms.
g_cpuTscFrequency = (( endTsc - startTsc) * freq.QuadPart) / (endQpc.QuadPart - startQpc.QuadPart);
```

## Timing Functions API

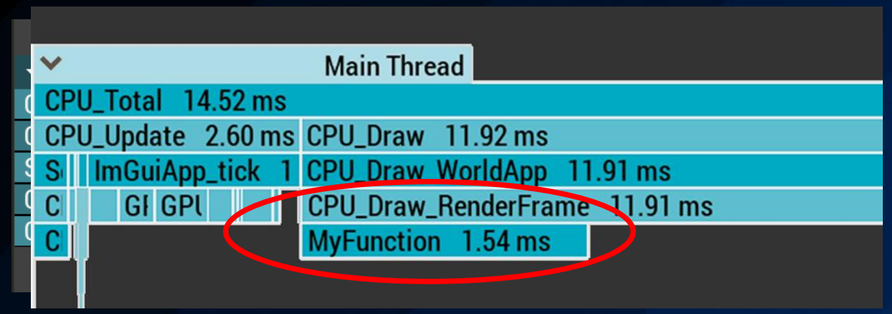
```
namespace Profiling
{
    inline UInt64 GetTickCount()
    {
        return __rdtsc();
    }

    inline UInt64 GetTicksPerSecond()
    {
        return g_cpuTscFrequency;
    }
}
```

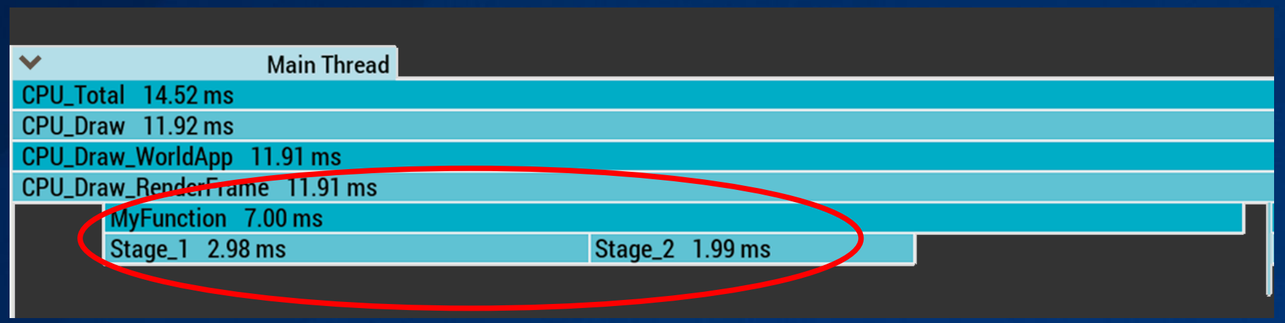
# Немного RAII для удобства

```
void MyFunction()
{
    Profiling::EnterCPUZone( "MyFunction" );

    // Your code is here.

    Profiling::LeaveCPUZone();
}
```

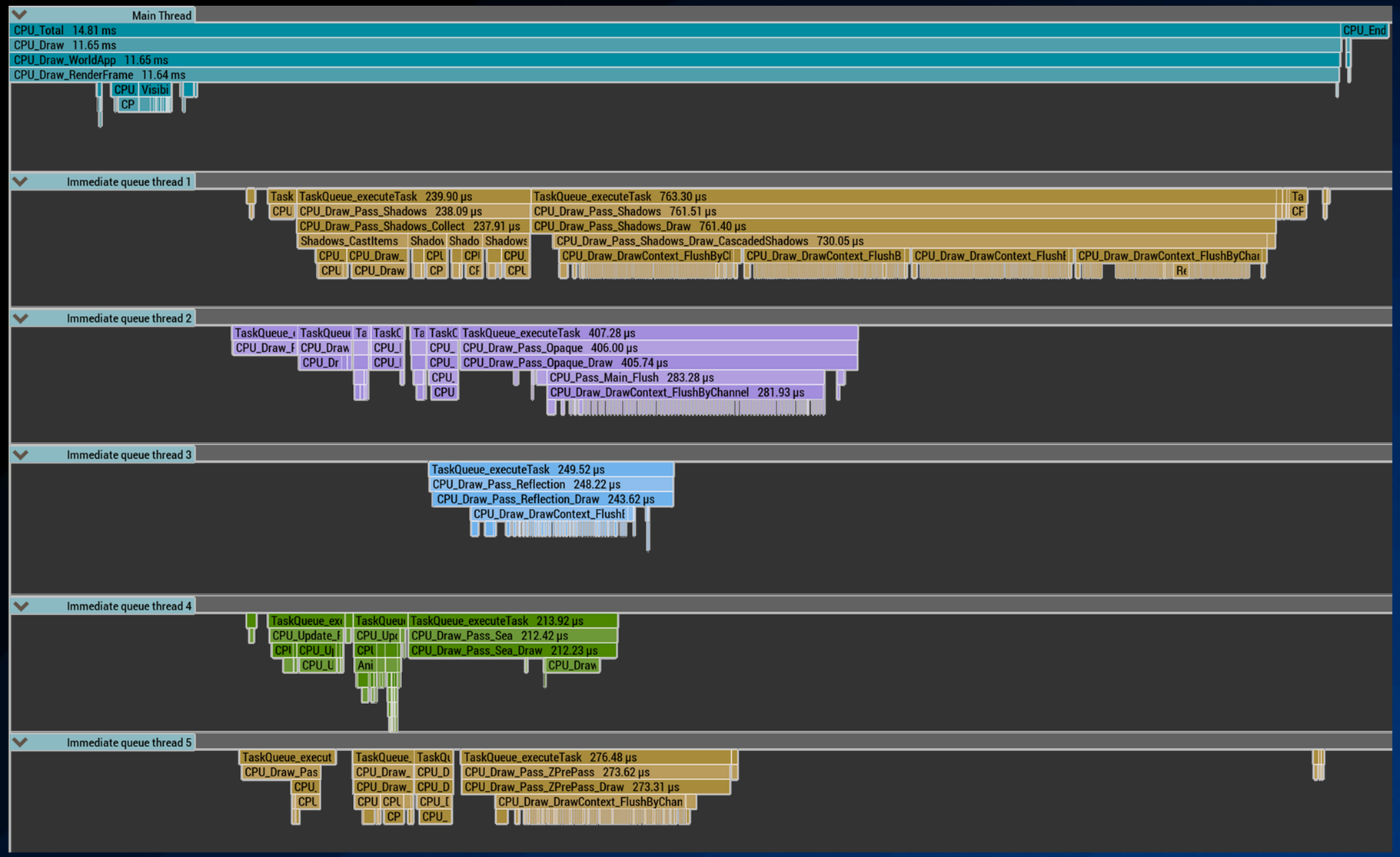
```
struct CPUZoneScope
{
public:
    explicit CPUZoneScope( StringView literalName )
    {
        Profiling::EnterCPUZone( literalName );
    }
    ~CPUZoneScope()
    {
        Profiling::LeaveCPUZone();
    }
};

#define PROFILE_CPU_ZONE CPUZoneScope profiledZone( __FUNCTION__ );
#define PROFILE_NAMED_CPU_ZONE( name ) CPUZoneScope profiledZone( name );
```

```
void MyFunction()
{
    PROFILE_CPU_ZONE;

    // Stage 1.
    {
        PROFILE_NAMED_CPU_ZONE( "Stage_1" );
        // ...
    }

    // Stage 2.
    {
        PROFILE_NAMED_CPU_ZONE( "Stage_2" );
        // ...
    }

    // ...
}
```



# Многопоточность



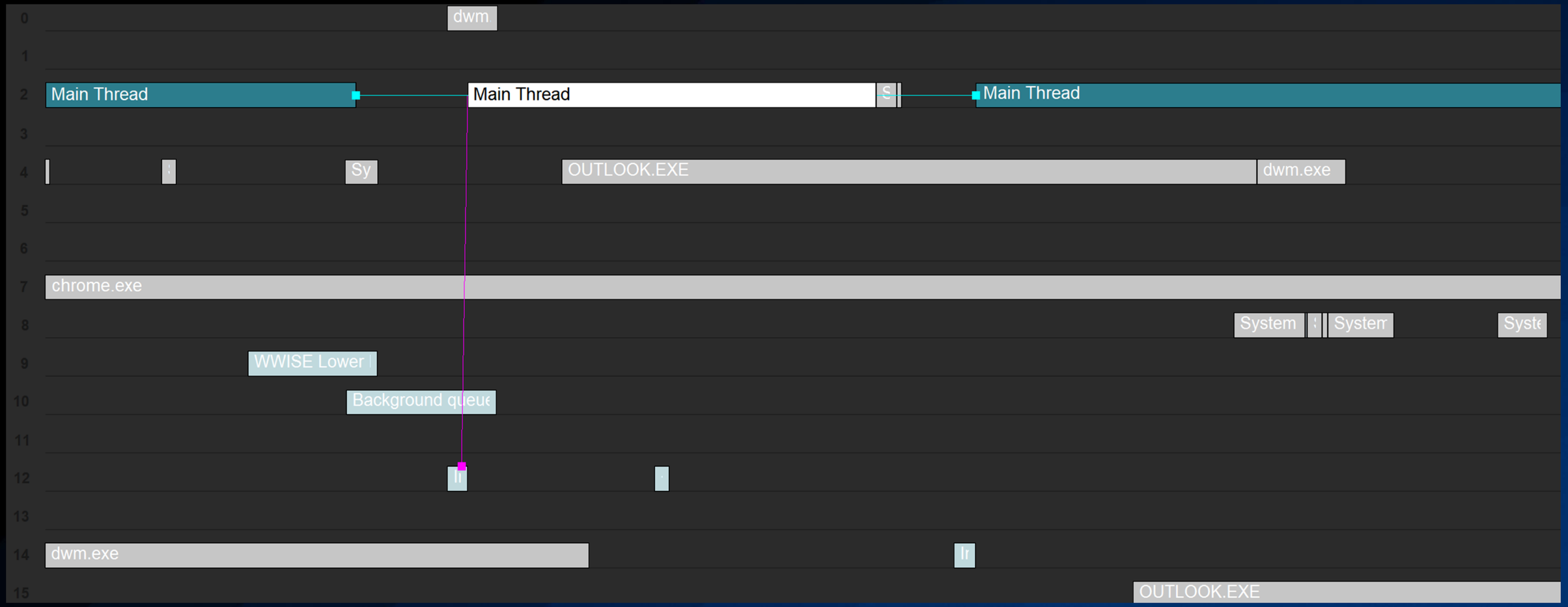
Thread Naming Trick

```
// WinApi hack for thread naming
// see: https://docs.microsoft.com/ru-ru/visualstudio/debugger/how-to-set-a-thread-name-in-native-code
void SetThreadName( DWORD dwThreadID, const char* threadName )
{
    const DWORD MS_VC_EXCEPTION = 0x406D1388;

#pragma pack( push, 8 )
    typedef struct tagTHREADNAME_INFO
    {
        DWORD dwType;       // Must be 0x1000.
        LPCSTR szName;      // Pointer to name (in user addr space).
        DWORD dwThreadID;   // Thread ID (-1=caller thread).
        DWORD dwFlags;      // Reserved for future use, must be zero.
    } THREADNAME_INFO;
#pragma pack( pop )

    THREADNAME_INFO info;
    info.dwType = 0x1000;
    info.szName = threadName;
    info.dwThreadID = dwThreadID;
    info.dwFlags = 0;

#pragma warning( push )
#pragma warning( disable: 6320 6322 )
    __try
    {
        RaiseException( MS_VC_EXCEPTION, 0, sizeof( info ) / sizeof( ULONG_PTR ), (ULONG_PTR*)&info );
    }
    __except ( EXCEPTION_EXECUTE_HANDLER )
    {
    }
#pragma warning( pop )
}
```

## CriticalSection Hooks

```
class CriticalSection: public ILockObject< CriticalSection >
{
public:
    CriticalSection()
        : m_acquired( false )
    {
        InitializeCriticalSection( &m_section );
    }

    ~CriticalSection()
    {
        DeleteCriticalSection( &m_section );
    }

    void Lock()
    {
        PROFILE_LOCK_TRY( &m_section );
        EnterCriticalSection( &m_section );
        assert( m_acquired == false );
        PROFILE_LOCK_ACQUIRED( &m_section );

        m_acquired = true;
    }

    Bool TryLock()
    {
        PROFILE_LOCK_TRY( &m_section );
        if ( TryEnterCriticalSection( &m_section ) )
        {
            assert( m_acquired == false );
            m_acquired = true;
            PROFILE_LOCK_ACQUIRED( &m_section );

            return true;
        }

        PROFILE_LOCK_FAILED( &m_section );
        return false;
    }

    void Unlock()
    {
        m_acquired = false;
        LeaveCriticalSection( &m_section );
        PROFILE_LOCK_RELEASED( &m_section );
    }

private:
    CRITICAL_SECTION m_section;
    Bool m_acquired;
};
```

# Примитивы синхронизации



***Event Tracing for Windows (ETW)*** – механизм для трассирования и логирования событий произошедших в user-mode приложениях и kernel-mode драйверах.

## События, собираемые через ETW:

- Process Start/End
- Thread Start/End
- CSwitch / ReadyThread
- Image Load/Unload
- File Create/Destroy/Rename/Map/Unmap/Read/Write
- Virtual Alloc/Free + Page Faults

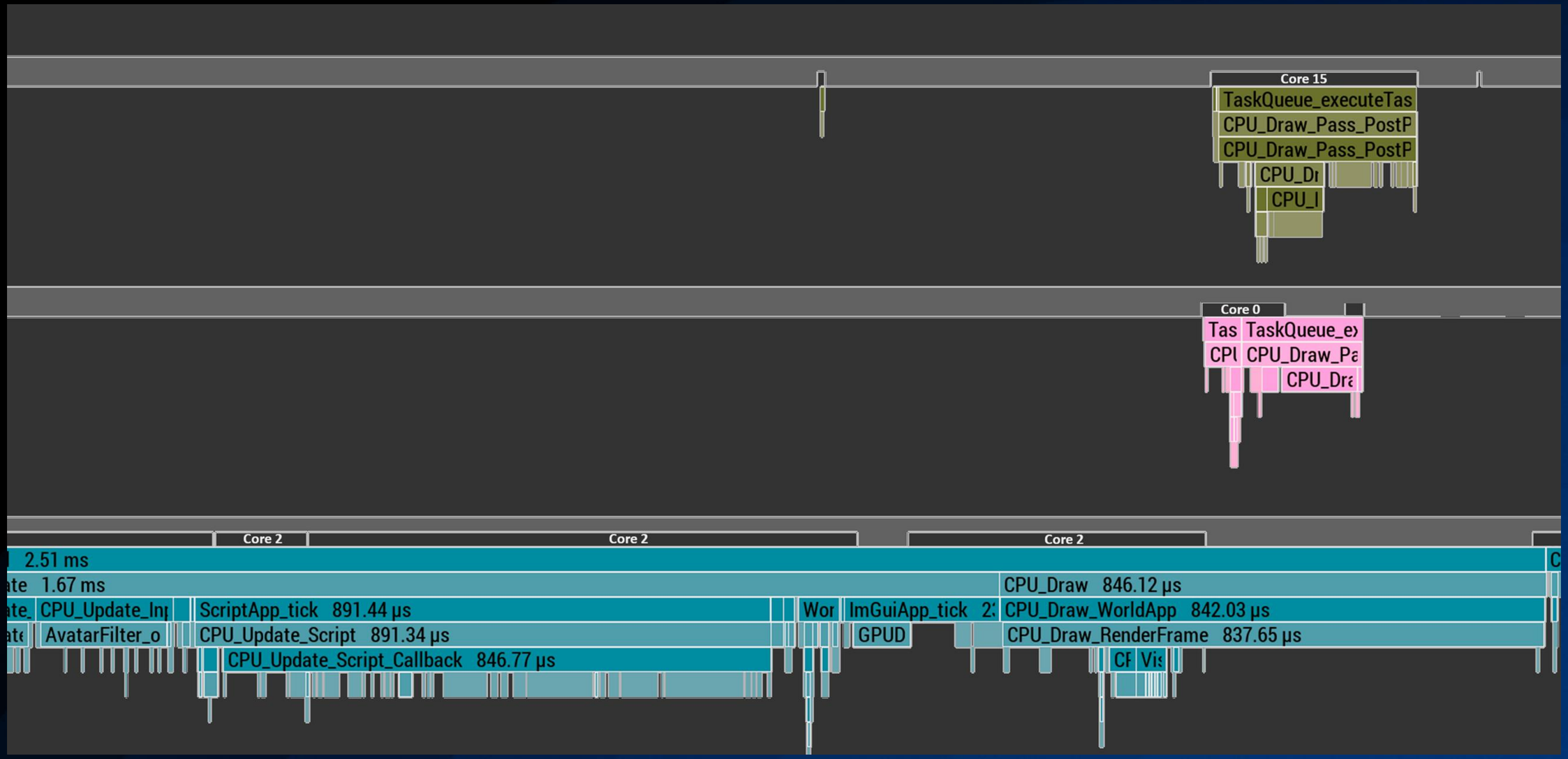
# Переключения контекстов

ETW Logger Sample

```
void ETWTraceCallback( PEVENT_RECORD record )
{
    // https://learn.microsoft.com/en-us/windows/win32/etw/cswitch
    struct CSwitch
    {
        // ...
    };

    static constexpr UCHAR CONTEXT_SWITCH_OPCODE = 36;

    if ( record->EventHeader.EventDescriptor.Opcode !=
CONTEXT_SWITCH_OPCODE )
        return;

    if ( sizeof( CSwitch ) != record->UserDataLength )
        return;

    const CSwitch* cs = reinterpret_cast< const CSwitch* >( record-
>UserData );

    ContextSwitchRecord res;
    res.timeStamp       = record->EventHeader.TimeStamp.QuadPart;
    res.newThreadId     = cs->NewThreadId;
    res.oldThreadId     = cs->OldThreadId;
    res.reason          = cs->OldThreadWaitReason;
    res.processor       = static_cast< UInt8 >( record-
>BufferContext.ProcessorIndex );

    reinterpret_cast< ETWTracer* >( record->UserContext )-
>m_callback( res );
}
```

```
ETWTracer* CreateETWTracer( ETWEventCallback&& callback )
{
    TraceInfo info = {};
    // ...

    // Stop active trace if any.
    TraceInfo tempInfo = info;
    ControlTrace(NULL, KERNEL_LOGGER_NAME, &tempInfo.base, EVENT_TRACE_CONTROL_STOP);

    // Start trace!
    TRACEHANDLE traceHandle;
    ULONG res = StartTrace( &traceHandle, KERNEL_LOGGER_NAME, &info.base );

    // Create tracer.
    ETWTracer* tracer = new ETWTracer();

    // Open log.
    EVENT_TRACE_LOGFILE traceLog = {};
    // ...
    traceLog.EventRecordCallback    = ETWTraceCallback;
    traceLog.Context                = tracer;
    tracer->m_handle = OpenTrace( &traceLog );

    struct Internal
    {
        static void TraceThreadEntryPoint( void* userData )
        {
            ETWTracer* tracer = reinterpret_cast< ETWTracer* >( userData );
            ProcessTrace( &tracer->m_handle, 1, NULL, NULL );
        }
    };

    tracer->m_thread.Start(&Internal::TraceThreadEntryPoint, tracer, "ETW Trace Thread");

    return tracer;
}
```

# Переключения контекстов



# GPU Profiling

- Время кадра на GPU так же критично, как и на CPU
- Некоторые конфигурации GPU Bound
- Работает асинхронно CPU
- Синхронизация CPU и GPU

# GPU Profiling

- Как измерять GPU время
- Немного RAII для удобства
- Сбор информации об DrawCall и не только
- Практика – как подвязаться к Vtbl экземпляра класса
- Практика – как подвязаться к Vtbl Com-объекта

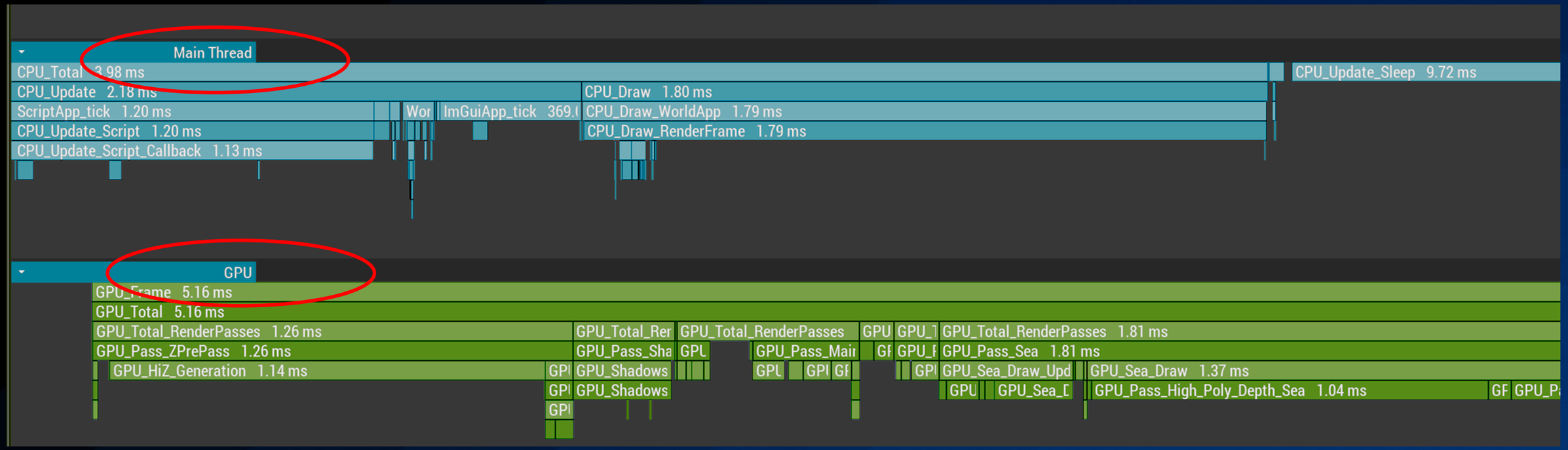
# Как измерять GPU время

## GPU Timing Profiler

```
void BeginFrame( ID3D12CommandQueue* commandQueue, UInt64 frameCounter )
{
    m_frameCounter = frameCounter;

    // Prepare per-frame data.
    FrameData& frame = m_frames[ m_frameCounter % NUM_FRAMES ];
    commandQueue->GetClockCalibration( (UINT64*)&frame.gpuSyncStamp,
(UINT64*)&frame.cpuSyncStamp );
}

void EndFrame( ID3D12CommandQueue* commandQueue )
{
    commandQueue->GetTimestampFrequency( &gpuFrequency );

    // Process the oldest frame.
    FrameData& oldFrame = m_frames[ ( m_frameCounter + 1 ) % NUM_FRAMES ];

    UInt64* resolvedStamps = nullptr;
    oldFrame.readbackBuffer->Map( 0, nullptr, (void**)&resolvedStamps );
    {
        //...
        const Double gpuDeltaSecs = Double( resolvedStamps[ queryIndex ] -
oldFrame.gpuSyncStamp ) / Double( gpuFrequency );
        TimeStamp cpuStamp = oldFrame.cpuSyncStamp + SecToTimeStamps( gpuDeltaSecs );

        switch ( e.type )
        {
            case Event::EType::Enter:   Profiling::EnterGPUZone( EGPUQueue::Graphics,
e.literalName, cpuStamp ); break;
            case Event::EType::Leave:   Profiling::LeaveGPUZone( EGPUQueue::Graphics,
cpuStamp ); break;
        }
    }
    oldFrame.readbackBuffer->Unmap( 0, nullptr );
}
```

```
void EnterScope( const Char* literalName, UInt32 commandListId,
ID3D12GraphicsCommandList* apiCommandList )
{
    FrameData& frame = m_frames[ m_frameCounter % NUM_FRAMES ];
    // ...
    apiCommandList->EndQuery( frame.queryHeap.Get(),
D3D12_QUERY_TYPE_TIMESTAMP, UINT( queryIndex ) );
    //...
}

void ResolveEvents( UInt32 commandListId, ID3D12GraphicsCommandList*
apiCommandList )
{
    FrameData& frame = m_frames[ m_frameCounter % NUM_FRAMES ];

    apiCommandList->ResolveQueryData( frame.queryHeap.Get(),
D3D12_QUERY_TYPE_TIMESTAMP,
            NUM_COMMAND_LIST_EVENTS * commandListId, UINT( cmdListData.count ),
            frame.readbackBuffer.Get(), sizeof( UInt64 ) *
NUM_COMMAND_LIST_EVENTS * commandListId );
}
```

# CPU & GPU Timelines



# Немного RAII для удобства

```cpp
void MyFunction( CommandList& commandList )
{
    commandList.EnterGPUZone( "MyFunction" );

    // Your code is here.

    commandList.LeaveGPUZone();
}
```

```cpp
struct GPUZoneScope
{
public:
    explicit GPUZoneScope( CommandList& commandList,
         StringView literalName )
        :   m_commandList( commandList )
    {
        m_commandList.EnterGPUZone( literalName );
    }
    ~GPUZoneScope()
    {
        m_commandList.LeaveGPUZone();
    }

private:
    CommandList& m_commandList;
};

#define PROFILE_GPU_ZONE( commandList )
GPUZoneScope profiledZone( commandList, __FUNCTION__ );

#define PROFILE_NAMED_GPU_ZONE( commandList, name )
GPUZoneScope profiledZone( commandList, name );
```

```cpp
void MyFunction( CommandList& commandList )
{
    PROFILE_GPU_ZONE( commandList );

    // Stage 1.
    {
        PROFILE_NAMED_GPU_ZONE( commandList, "Stage 1" );
        // ...
    }

    // Stage 2.
    {
        PROFILE_NAMED_GPU_ZONE( commandList, "Stage 2" );
        // ...
    }

    // ...
}
```

# Какие данные еще интересно собирать с GPU

- Draw Call Pipeline Statistics (см. изображение)
- Dispatch Call
- Clear
- Copy
- Present

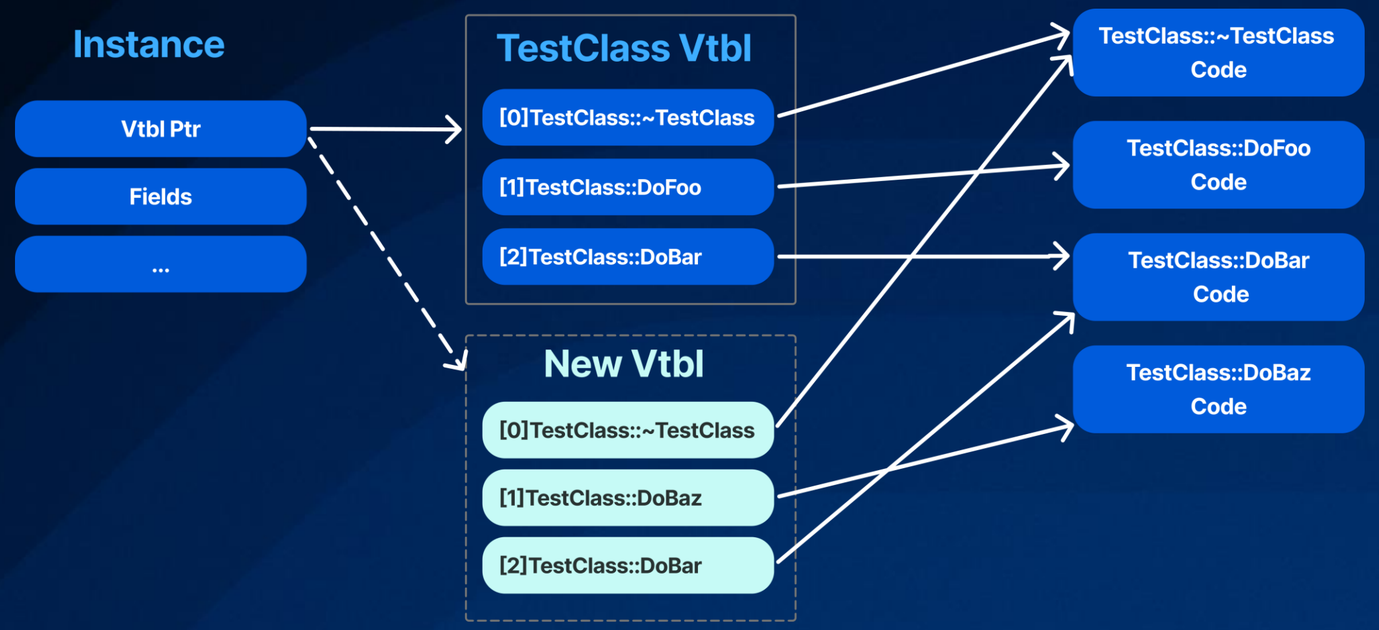
C++

```
typedef struct D3D12_QUERY_DATA_PIPELINE_STATISTICS {
  UINT64 IAVertices;
  UINT64 IAPrimitives;
  UINT64 VSInvocations;
  UINT64 GSInvocations;
  UINT64 GSPrimitives;
  UINT64 CInvocations;
  UINT64 CPrimitives;
  UINT64 PSInvocations;
  UINT64 HSInvocations;
  UINT64 DSInvocations;
  UINT64 CSInvocations;
} D3D12_QUERY_DATA_PIPELINE_STATISTICS;
```

https://learn.microsoft.com/en-us/windows/win32/api/d3d12/ns-d3d12-d3d12_query_data_pipeline_statistics

# Практика – как подвязаться к Vtbl экземпляра класса

```
class BaseClass
{
public:
    virtual ~BaseClass() = default;

    virtual void DoFoo( Int32 i )
    {
        debug( "BaseClass::DoFoo( %d )", i );
    }
    virtual void DoBar()
    {
        debug( "BaseClass::DoBar()" );
    }
};

class TestClass: public BaseClass
{
public:
    ~TestClass() override = default;

    void DoFoo( Int32 i ) override
    {
        debug( "TestClass::DoFoo( %d )", i );
    }

    void DoBar() override
    {
        debug( "TestClass::DoBar()" );
    }

    void DoBaz( Int32 i )
    {
        debug( "TestClass::DoBaz( %d )", i );
    }
};
```



# Практика – как подвязаться к Vtbl экземпляра класса

## Utility Functions

```
void** GetVtbl( void* instancePtr )
{
    return *(void***)instancePtr;
}

void ChangeVtbl( void* instancePtr, void** newVtbl )
{
    *(void**)(instancePtr) = newVtbl;
}

template< typename F >
void ReplaceVtblEntry( void** vtbl, SizeT index, F f )
{
    mem::Copy( &vtbl[ index ], &f, sizeof( void* ) );
}

SizeT GuessVtblSize( void** vtbl )
{
    for ( SizeT i = 0; ; ++i )
    {
        MEMORY_BASIC_INFORMATION memInfo = {};
        VirtualQuery( vtbl[ i ], &memInfo,
            sizeof( MEMORY_BASIC_INFORMATION ) );

        DWORD TestFlags = PAGE_EXECUTE | PAGE_EXECUTE_READ |
PAGE_EXECUTE_READWRITE;

        if ( !( memInfo.Protect & TestFlags ) )
        {
            return i;
        }
    }
}
```

## Vtbl Patching

```
    BaseClass* Instance = new TestClass();

    void** OrigVtbl = GetVtbl( Instance );
    SizeT VtblSize = GuessVtblSize( OrigVtbl );

    // Create a new vtbl.
    void** NewVtbl = reinterpret_cast< void** >( mem::Malloc( VtblSize * sizeof( void* ) ) );
    mem::Copy( NewVtbl, OrigVtbl, VtblSize * sizeof( void* ) );

    // Replace Foo with Baz
    ReplaceVtblEntry( NewVtbl, 1, &TestClass::DoBaz );

    // Change instance vtbl.
    ChangeVtbl( Instance, NewVtbl );

    Instance->DoFoo( 123 );
    Instance->DoBar();

// Prints:
// ---
// TestClass::DoBaz( 123 )
// TestClass::DoBar()


    // Restore original state.
    ChangeVtbl( Instance, OrigVtbl );

    mem::Free( NewVtbl );
    delete Instance;
```

# Практика – как подвязаться к Vtbl экземпляра класса

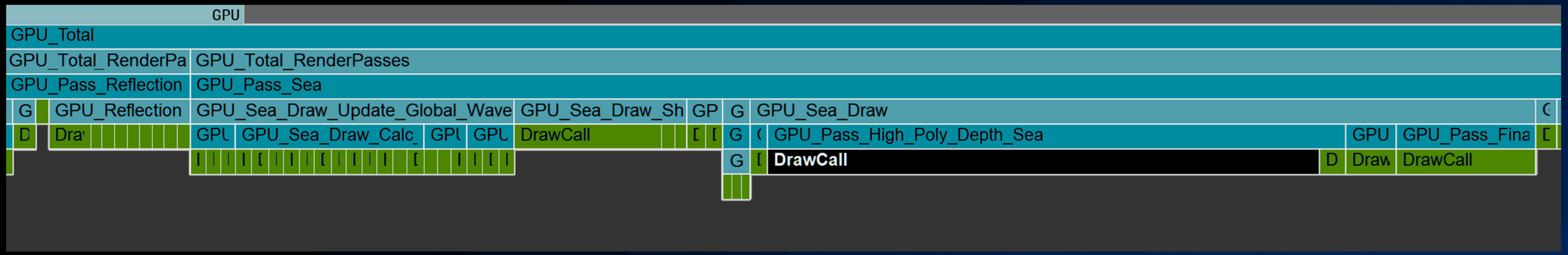
```
class TestClass: public BaseClass
{
public:
    ~TestClass() override = default;

    void _stdcall DoFoo( Int32 i ) override
    {
        debug( "TestClass::DoFoo( %d )", i );
    }

    void _stdcall DoBar() override
    {
        debug( "TestClass::DoBar()" );
    }

    void _stdcall DoBaz( Int32 i )
    {
        debug( "TestClass::DoBaz( %d )", i );
    }
};

void _stdcall DoXyz( BaseClass* pThis, Int32 i )
{
    debug( "DoXyz( %d )", i );
};
```

```
    // ...

    // Replace Foo with Baz
    ReplaceVtblEntry( NewVtbl, 1, &DoXyz );

    Instance->DoFoo( 123 );
    Instance->DoBar();

// Prints:
// ---
// DoXyz( 123 )
// TestClass::DoBar()

 // ...
```

# Практика – как подвязаться к Vtbl Com-объекта

```
#define CINTERFACE
#include <d3d12.h>

void __stdcall OurCreateShaderResourceView( ID3D12Device* This, ID3D12Resource* pResource, const D3D12_SHADER_RESOURCE_VIEW_DESC* pDesc,
D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor )
{
    // You can do whatever you want here.

    // Call original method.
    m_OldVtbl->CreateShaderResourceView( This, pResource, pDesc, DestDescriptor );
}



// Guess D3DDevice vtbl size (driver specific).
SizeT VtblSize = GuessVtblSize( GetVtbl( D3DDevice ) );

// Allocate a new Vtbl.
void** NewVtbl = reinterpret_cast< void** >( mem::Malloc( VtblSize * sizeof( void* ) ) );
mem::Copy( NewVtbl, D3DDevice->lpVtbl, sizeof(void*) * VtblSize );

// Remember original Vtbl.
m_OldVtbl = D3DDevice->lpVtbl;

// Replace Vtbl.
*(void**)&D3DDevice->lpVtbl = NewVtbl;

// Replace methods.
D3DDevice->lpVtbl->CreateShaderResourceView = OurCreateShaderResourceView;
// ...
```

# Практика – как подвязаться к Vtbl Com-объекта




| | |
|---|---|
| Num Primitives at IA | 300672 |
| VS Invocations | 357676 |
| HS Invocations | 300672 |
| DS Invocations | 6477483 |
| GS Invocations | 0 |
| Num Primitives Pre-Clipping | 6847440 |
| Num Primitives Post-Clipping | 5018282 |
| PS Invocations | 475945 |

# RAM & VRAM Profiling

- Так же важно отслеживать, как и время выполнения
- Может бить по времени выполнения
- Один из основных источников «спайков»
- Может происходить в самых неожиданных местах

# RAM & VRAM Profiling

- VRAM Profiling
- RAM Profiling
- Allocation Gathering
- Allocation Hooking

# VRAM Profiling

```
// Combined name (space, camera, name).
m_instanceBuffer = device.CreateBuffer( EBufferType::GenericBuffer, m_instances.Data(), m_instances.Capacity(), sizeof( GPUIcon ), EUsage::Dynamic,
    MakeResourceName( "Space_Meadow", "OpaqueCamera", "IconInstancingBuffer" ) );

// Name based on file name.
outTexture = device.CreateTexture2D( EFormat::BC1, ETextureFlags::None, 1024, 1024, 1, EUsage::Immutable, *Data, MakeResourceName( FileName ) );


struct GPUMemoryStats
{
    struct Stats
    {
        SizeT       Usage;      // Amount of allocated memory.
        SizeT       Budget;     // Budget provided by graphics API.
        Atomic64    Resource;   // Amount of calculated memory.
    };

    Stats   CpuMemory;  // Shared RAM
    Stats   GpuMemory;  // VRAM
};

// Local memory.
DXGI_QUERY_VIDEO_MEMORY_INFO MemoryInfo = {};
Adapter->QueryVideoMemoryInfo( 0, DXGI_MEMORY_SEGMENT_GROUP_LOCAL, &MemoryInfo );

m_memoryStats.GpuMemory.Usage = MemoryInfo.CurrentUsage;
m_memoryStats.GpuMemory.Budget = MemoryInfo.Budget;

// Non-local memory.
DXGI_QUERY_VIDEO_MEMORY_INFO MemoryInfo = {};
Adapter->QueryVideoMemoryInfo( 0, DXGI_MEMORY_SEGMENT_GROUP_NON_LOCAL, &MemoryInfo );

m_memoryStats.CpuMemory.Usage = MemoryInfo.CurrentUsage;
m_memoryStats.CpuMemory.Budget = MemoryInfo.Budget;
```

# GPU Resources Tree

| Size (KB) | Width | Height | Texture | Group | Used in |
|---|---|---|---|---|---|
| 2256 | 760 | 760 | Minimap target: result | RT | |
| 0 | 4 | 4 | texture/GPUTextureCompressor result map: "ImposterBaker" 8x4 | RT | |
| 2730 | 2048 | 1024 | Algae_03_ImposterAtlas_aux | Trees | |
| 21 | 128 | 128 | content/location/textures/LMG001_alpha_a | Objects | content/location/misc/gardens/LGHRT44/lods/LGHRT44_lod1.visual |
| 21 | 128 | 128 | Bees_to_Honey_misc_asf_01_alpha_a | Decals | |
| 10 | 128 | 128 | content/location/textures/DEA34_tent_a | Objects | content/location/DEA34/DEA34.visual |
| 5 | 64 | 64 | content/location/vechicle/textures/LVA063_n | Objects | content/vechicle/automobile/LLDDA/lods/lod1.visual |
| 341 | 512 | 512 | dyndecals/damage_dec_6_d | Decals | |
| 1365 | 512 | 512 | resources/maps/gizmo/square | Other | |
| 2168 | 593 | 468 | PostFX_lensBlur/PostFX_PostFX/D2F8 | RT | |
| 4096 | 1024 | 512 | texture/HDR bloom map #0_2 | RT | |
| 85 | 256 | 256 | 16_Bees_to_Honey_misc_road_18_alpha_a | Decals | |
| 8192 | 2048 | 1024 | texture/ImposterBaker G-Buffer aux map #1 | RT | |
| 682 | 1024 | 1024 | content/nature/tile/textures/LNT016_BTH_09_od | Landscape | content/nature/tile/LNT016_BTH_09/lods/LNT016_BTH_09_lod1.visual |
| 682 | 1024 | 1024 | maps/16_Bees_to_Honey/16_BTH_HD_LA_08_mg | Decals | |
| 682 | 1024 | 1024 | content/underwater/textures/TILED/LNU007_od | Landscape | content/underwater/LNU007/LNU007 |
| 42 | 256 | 128 | content/location/building/city/textures/LBC056_n | Objects | content/location/building/city/LBC067/lods/LBC067_lod1.visua |
| 42 | 128 | 256 | content/location/misc/fence/textures/LMF050_n | Objects | content/location/misc/fence/LMF060/LMF060.visual |
| 682 | 1024 | 1024 | content/tile/textures/LNT016_BTH_04_lava_dry_od | Landscape | content/tile/LNT016_BTH_04_lava_dry/lods/lod2.visual |
| 1365 | 1024 | 1024 | content/location/tile/textures/LNT016_BTH_02_blend | Landscape | content/location/tile/LNT016_BTH_02/lods/LNT016_BTH_02_lod1. |
| 10 | 128 | 128 | content/location/sailboat/textures/LSS032_Boat_mg | Objects | content/location/sailboat/LSS034/lods/LSS034_lod1.visual<br>content/location/sailboat/LSS034/lods/LSS034_lod2.visual |
| 0 | 16 | 4 | resources/textures/yellow_ruler | Other | |
| ... | ... | | ... | | ... |

```
GPU Resources
├── Textures
│   ├── Environment
│   │   ├── Skybox
│   │   │   ├── Day
│   │   │   └── Night
│   │   ├── Terrain
│   │   │   ├── texture
│   │   │   └── …
│   ├── Characters
│   │   ├── Hero
│   │   │   ├── texture
│   │   │   └── …
│   │   └── Enemy
│   │       ├── texture
│   │       └── …
│   └── UI
│       ├── texture
│       └── …
├── Models
│   ├── Characters
│   │   ├── hero.fbx
│   │   └── enemy.fbx
│   ├── Environment
│   │   ├── tree.fbx
│   │   └── rock.fbx
│   └── Props
│       ├── barrel.fbx
│       └── crate.fbx
├── Buffers
│   ├── Vertex Buffers
│   │   ├── hero.vbo
│   │   └── enemy.vbo
│   ├── Index Buffers
│   │   ├── hero.ibo
│   │   └── enemy.ibo
│   └── Uniform Buffers
│       ├── camera_ubo
│       └── light_ubo
└── Render Targets
    ├── Main Frame Buffer
    ├── Shadow Map
    └── Post-Processing Buffer
```

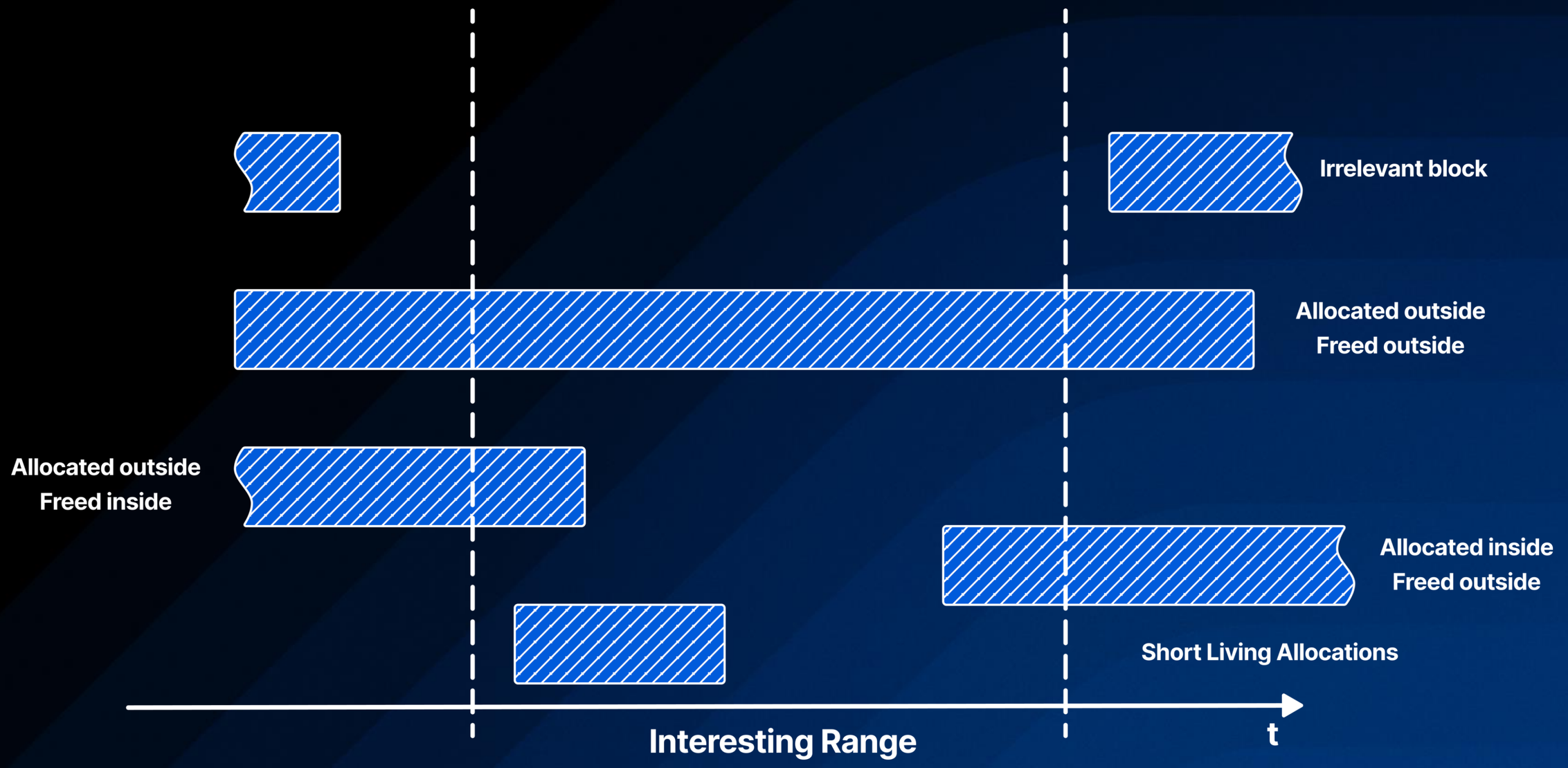
# Что собираем

- Базово подвязываемся на malloc/realloc/free
- Подвязываемся на требуемые аллокаторы (например из third-party)
- Статистику из OS: Virtual Memory, Commit Size, Working Set, Page File и т.д.
- Отдельно собираем низкоуровневые аллокации HeapAlloc/VirtualAlloc

## Трекинг общих аллокаций

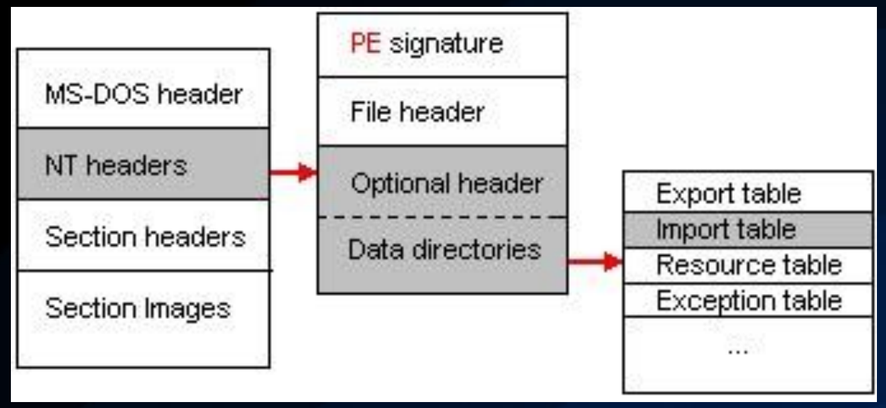
```
// We have our own allocation functions:
void* Allocate( SizeT Size );
void* Reallocate( void* OldPtr, SizeT Size );
void Free( void* Ptr );

// We've overrided new/delete operators

UInt32 GGenericHeap = Profiling::RegisterHeap( "Generic" );

void* Allocate( SizeT Size )
{
    // Do actual allocation.
    // ...

    Profiling::TrackAllocation( GGenericHeap, Ptr, Size );
    return Ptr;
}

// The same way we handle all allocations. From specific allocators.
// The same way we gather GPU allocations.
// We collect stacktrace as well, so we can detect memory leaks, or check live-allocations.
```

# Allocation Gathering

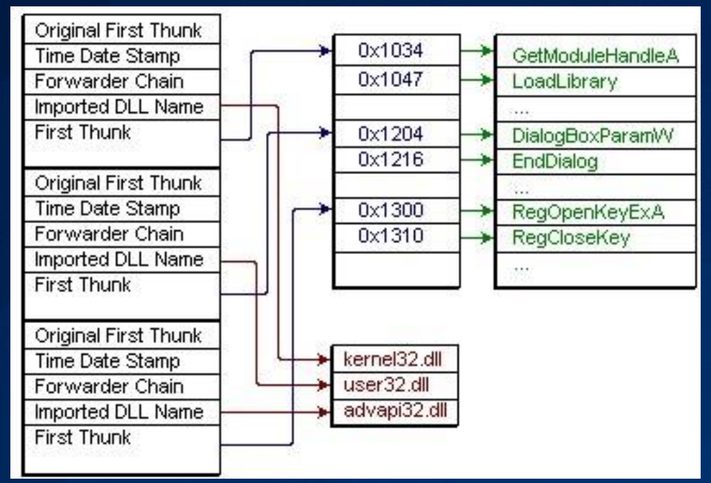


## Call Disassembling

```
// From User32.dll Module.
MessageBeep( MB_ICONSTOP );
00007FF76CA70D28 B9 10 00 00 00      mov        ecx,10h
00007FF76CA70D2D FF 15 25 0A 74 00   call       qword ptr [__imp_MessageBeep (07FF76D1B1758h)]

// From Our Module.
MyFunction( 123 );
00007FF76CA70D33 B9 7B 00 00 00      mov        ecx,7Bh
00007FF76CA70D38 E8 C3 22 08 00      call       MyFunction (07FF76CAF3000h)
```




| | |
|---|---|
| Data Directories | 0 Export Table |
| | 1 Import Table |
| | 2 Resource Table |
| | 3 Exception Table |
| | 4 Certificate File |
| | 5 Relocation Table |
| | 6 Debug Data |
| | 7 Architecture Data |
| | 8 Global Ptr |
| | 9 Thread Local Storage Table |
| | 10 Load Config Table |
| | 11 Bound Import Table |
| | 12 Import Address Table |
| | 13 Delay Import Descriptor |
| | 14 COM+ Runtime Header |
| | 15 Reserved |




# GetModuleHandle

# Allocation Hooking

```
void SetHook( void** FunctionPtr )
{
    for ( const auto& Hook : GHooks )
    {
        if ( *FunctionPtr == *Hook.OriginalFunction )
        {
            DWORD OldProtect;
            VirtualProtect( FunctionPtr, sizeof(PVOID), PAGE_READWRITE, &OldProtect );
            WriteProcessMemory( GetCurrentProcess(), FunctionPtr, &Hook.HookFunction, sizeof(Hook.HookFunction), nullptr );
            VirtualProtect( FunctionPtr, sizeof(PVOID), OldProtect, &OldProtect );
        }
    }
}

PIMAGE_IMPORT_DESCRIPTOR SetHooks( HMODULE ModuleHandle )
{
    ULONG UnusedSize;
    PIMAGE_IMPORT_DESCRIPTOR Import = (PIMAGE_IMPORT_DESCRIPTOR)ImageDirectoryEntryToData( ModuleHandle, TRUE, IMAGE_DIRECTORY_ENTRY_IMPORT, &UnusedSize );
    if ( !Import )
        return nullptr;

    for ( PIMAGE_IMPORT_DESCRIPTOR ImportDesc = Import; ImportDesc->Name; ImportDesc++ )
    {
        PIMAGE_THUNK_DATA pThunk = (PIMAGE_THUNK_DATA)( (PBYTE)ModuleHandle + ImportDesc->FirstThunk );
        if ( !pThunk )
            continue;

        for ( ; pThunk->u1.Function; pThunk++ )
        {
            void** FunctionPtr = (void**)&pThunk->u1.Function;
            SetHook( FunctionPtr );
        }
    }
    return Import;
}
```

# Allocation Hooking

```
struct VirtualMemoryOriginalFunctions
{
    VirtualAllocFunc    VirtualAlloc;
    VirtualFreeFunc     VirtualFree;
    // ...
};

static VirtualMemoryOriginalFunctions GVirtualMemoryOriginalFunctions;

HookInfo GHooks[] =
{
//      Module  Name,   Function Name,  Original Function Ptr,                                New Function Ptr
    {   "Kernel32.dll", "VirtualAlloc", (void**)&GVirtualMemoryOriginalFunctions.VirtualAlloc,  OurVirtualAlloc },
    {   "Kernel32.dll", "VirtualFree",  (void**)&GVirtualMemoryOriginalFunctions.VirtualFree,   OurVirtualFree },
};

UInt32 GVirtualMemoryHeap = Profiling::RegisterHeap( "VirtualMemory" );

extern "C"
{
    LPVOID WINAPI OurVirtualAlloc( LPVOID lpAddress, SIZE_T dwSize, DWORD flAllocationType, DWORD flProtect )
    {
        // Register this call in our profiler.
        Profiling::TrackAllocation( GVirtualMemoryHeap, lpAddress, dwSize );

        // Call original.
        return GVirtualMemoryOriginalFunctions.VirtualAlloc( lpAddress, dwSize, flAllocationType, flProtect );
    }

    //...
}

#pragma init_seg(lib)
static HookInitializer GHookInitializer( GHooks );
```

## DirectX 12 в фоне аллоцирует данные

Call Stack

View all Threads | Show External Code

| Name | Language |
|---|---|
| OurHeapAlloc(void * hHeap, unsigned long dwFlags, unsigned __int64 dwBytes) Line 196 | C++ |
| D3D12.dll!wil::details_abi::heap_buffer::reserve(unsigned __int64) | Unknown |
| D3D12.dll!wil::details_abi::UsageIndex<enum wil_details_ServiceReportingKind,unsigned int,0>::RecordUsage() | Unknown |
| D3D12.dll!wil::details::WilApiImpl_RecordFeatureUsage(unsigned int,unsigned int,unsigned int,char const *) | Unknown |
| D3D12.dll!wil::details::WilApi_RecordFeatureUsage() | Unknown |
| D3D12.dll!wil_details_RecordCachedUsage() | Unknown |
| D3D12.dll!wil::details::EnabledStateManager::RecordCachedUsageUnderLock() | Unknown |
| D3D12.dll!wil::details::EnabledStateManager::OnTimer(void) | Unknown |
| D3D12.dll!wil::details_abi::SubscriptionList::OnSignaled(class wil::srwlock &) | Unknown |
| D3D12.dll!<lambda_152aa9d2a3a0648fa2fa7fcef514b376>::<lambda_invoker_cdecl>(void *) | Unknown |
| KernelBase.dll!wil::details_abi::SubscriptionList::OnSignaled(class wil::rtlsrwlock &) | Unknown |
| KernelBase.dll!wil::details::FeatureStateManager::FlushUsage(void) | Unknown |
| KernelBase.dll!wil::details::FeatureStateManager::OnTimer(void) | Unknown |
| ntdll.dll!TppTimerpExecuteCallback() | Unknown |
| ntdll.dll!TppWorkerThread() | Unknown |
| kernel32.dll!BaseThreadInitThunk() | Unknown |
| ntdll.dll!RtlUserThreadStart() | Unknown |

## Одна из аллокациий при создании окна

Call Stack

Search (Ctrl+E) | View all Threads | Show External Code

| Name | Language |
|---|---|
| e!OurHeapAlloc(void * hHeap, unsigned long dwFlags, unsigned __int64 dwBytes) Line 196 | C++ |
| KernelBase.dll!LocalAlloc() | Unknown |
| msctf.dll!CThreadInputMgr::CreateInstance() | Unknown |
| msctf.dll!CicBridge::InitIMMX() | Unknown |
| msctf.dll!_CtfImeCreateThreadMgr() | Unknown |
| msctf.dll!CtfImeCreateThreadMgr() | Unknown |
| imm32.dll!CtfImmTIMActivate() | Unknown |
| imm32.dll!ImmSetActiveContext() | Unknown |
| user32.dll!FocusSetIMCContext() | Unknown |
| user32.dll!ImeSystemHandler() | Unknown |
| user32.dll!ImeWndProcWorker() | Unknown |
| user32.dll!ImeWndProcW(struct HWND__ *,unsigned int,unsigned __int64,__int64) | Unknown |
| user32.dll!UserCallWinProcCheckWow(struct _ACTIVATION_CONTEXT *,__int64 (*)(struct tagWND *,unsigned int,unsigned __int64,__int6... | Unknown |
| user32.dll!DispatchClientMessage() | Unknown |
| user32.dll!__fnDWORD() | Unknown |
| ntdll.dll!KiUserCallbackDispatcherContinue() | Unknown |
| win32u.dll!NtUserShowWindow() | Unknown |
| [redacted]::CreateWindow([redacted] | C++ |
| [redacted]::RegisterForm([redacted]) Line 815 | C++ |
| [redacted] | C++ |
| [redacted] | C++ |
| [redacted] | C++ |
| [redacted]WinMain(HINSTANCE__ * hInstance, HINSTANCE__ * hPrevInstance, char * lpCmdLine, int nCmdShow) Line 546 | C++ |

## DirectX 12 делает аллокации при выставлении барьеров

Call Stack

Search (Ctrl+E) | View all Threads | Show External Code

| Name | Language |
|---|---|
| [redacted]e!OurHeapAlloc(void * hHeap, unsigned long dwFlags, unsigned __int64 dwBytes) Line 196 | C++ |
| ucrtbase.dll!_malloc_base() | Unknown |
| ucrtbase.dll!__crt_state_management::wrapped_invoke<void * (*)(unsigned __int64),unsigned __int64,void *>(void * (*)(unsigned __int6... | Unknown |
| d3d12SDKLayers.dll!operator new(unsigned __int64) | Unknown |
| d3d12SDKLayers.dll!std::vector<struct D3D12_VIDEO_EXTENSION_COMMAND_PARAMETER_INFO,class std::allocator<struct D3D12_VI... | Unknown |
| d3d12SDKLayers.dll!CResourceState::Initialize(unsigned int) | Unknown |
| d3d12SDKLayers.dll!NDebug::CCastableCommandList<struct ID3D12GraphicsCommandListNext>::UpdateResourceBarrierImpl(struct D... | Unknown |
| d3d12SDKLayers.dll!NDebug::CGraphicsCommandList::UpdateResourceBarrierImpl(struct D3D12_RESOURCE_BARRIER const *,class ND... | Unknown |
| d3d12SDKLayers.dll!NDebug::CCommandList<struct ID3D12GraphicsCommandListNext>::UpdateResourceBarrier(struct D3D12_RESOU... | Unknown |
| d3d12SDKLayers.dll!NDebug::CCommandList<struct ID3D12GraphicsCommandListNext>::PerResourceBarrierProcessing(struct D3D12_... | Unknown |
| d3d12SDKLayers.dll!NDebug::CGraphicsCommandList::PerResourceBarrierProcessing(struct D3D12_RESOURCE_BARRIER const *,unsign... | Unknown |
| d3d12SDKLayers.dll!NDebug::CCommandList<struct ID3D12GraphicsCommandListNext>::ResourceBarrier(unsigned int,struct D3D12_R... | Unknown |
| [redacted]::CommandList::FlushTransitionQueueImpl([redacted] | C++ |
| [redacted]CommandList::FlushTransitionQueue() Line 476 | C++ |
| [redacted]CommandList::UpdateBufferAsync([redacted]ne 288 | C++ |
| [redacted]CommandList::UpdateBuffer([redacted] | C++ |

## Наша аллокация через aligned_malloc

Call Stack

Search (Ctrl+E) | View all Threads | Show External Code

| Name | Language |
|---|---|
| [redacted]!OurHeapAlloc(void * hHeap, unsigned long dwFlags, unsigned __int64 dwBytes) Line 196 | C++ |
| ucrtbased.dll!heap_alloc_dbg_internal(const unsigned __int64 size, const int block_use, const char * const file_name, const int line_nu... | C++ |
| ucrtbased.dll!heap_alloc_dbg(const unsigned __int64 size, const int block_use, const char * const file_name, const int line_number) Lin... | C++ |
| ucrtbased.dll!_malloc_dbg(unsigned __int64 size, int block_use, const char * file_name, int line_number) Line 496 | C++ |
| ucrtbased.dll!_aligned_offset_malloc_dbg(unsigned __int64 size, unsigned __int64 alignment, unsigned __int64 offset, const char * file_... | C++ |
| ucrtbased.dll!_aligned_malloc(unsigned __int64 size, unsigned __int64 alignment) Line 557 | C++ |
| [redacted]RootAllocator::AlignedMalloc(unsigned __int64 numBytes, unsigned __int64 align) Line 42 | C++ |
| [redacted]::MemoryManager::AlignedMalloc(unsigned __int64 numBytes, unsigned __int64 align) Line 209 | C++ |
| [redacted]::AlignedMalloc(unsigned __int64 numBytes, unsigned __int64 align) Line 514 | C++ |
| [redacted]operator new(unsigned __int64 numBytes) Line 641 | C++ |
| [redacted] | C++ |
| [redacted] | C++ |

# File IO Profiling

- Время загрузки карт и персонажей
- Стриминг данных во время игры
- Сотни гигабайт уже пожатого контента

# Особенности

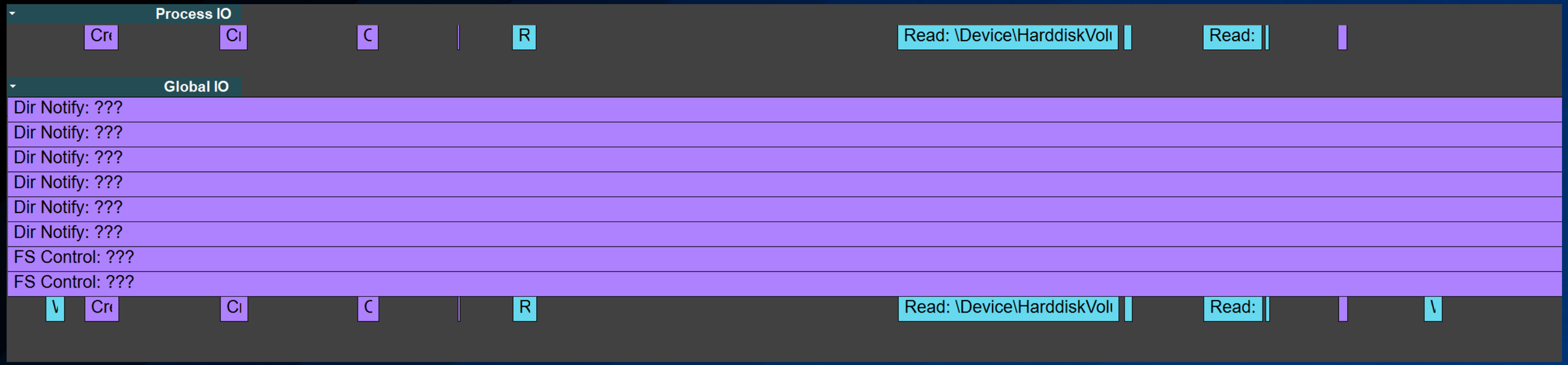
- В основном хватает CPU зон
- Можем упереться в диск или CPU (декомпрессия данных)
- Не всегда тривиально, например, File Mapping и Overlapped IO
- Эффективное представление данных
- Требуется обнаруживать неэффективные паттерны работы

# Детали реализации

- Покрыты отдельными CPU зонами
- Санитарные метрики
- ETW для отслеживания событий как внутри процесса, так и системы
- Хуки на CreateFile и так далее

# Информация, которую собираем

- Тип операции Create, Close, Flush, Read, Write, Query Info, Set Info, Delete, Rename, FSCTL Command, Dir Enum, Dir Notify
- Время начала и продолжительность
- Имя файла
- Процесс
- Поток



# Data Plots

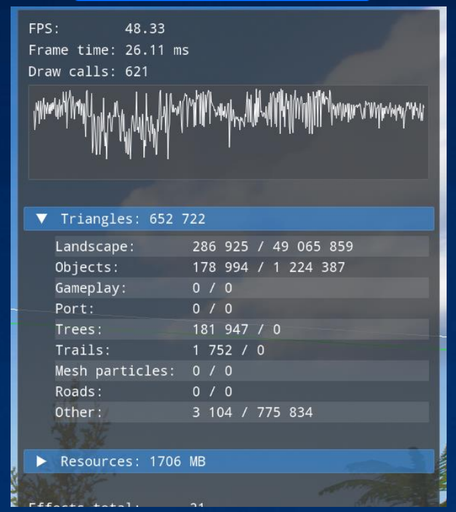
- Числовые/процентные значения по времени
- Используется для «маленького» профайлера
- Отображаем зоны
- Данные из операционной системы

ЛЕСТА

# Data Plots

## Performance QA Analysis Tool

## Art Analysis Tool



# Data Plots

```
// Call it once a frame or more.
PROFILE_COUNTER("Lighting.LocalLights.Visible", GetVisibleLightsCount());
PROFILE_COUNTER("Lighting.LocalLights.Count", m_localLights.Size());


PROFILE_COUNTER("Triangles.MapModels", GStatsTriByType[EStats::MAP_MODELS]);
PROFILE_COUNTER("Triangles.ShipModels", GStatsTriByType[EStats::SHIP_MODELS]);
PROFILE_COUNTER("Triangles.Forest", GStatsTriByType[EStats::FOREST]);
PROFILE_COUNTER("Triangles.Terrain", GStatsTriByType[EStats::TERRAIN]);
PROFILE_COUNTER("Triangles.Water", GStatsTriByType[EStats::WATER]);
PROFILE_COUNTER("Triangles.FX", GStatsTriByType[EStats::PARTICLES]);

PROFILE_COUNTER("Particles.ScreenCoverage", ScreenCoverage);
PROFILE_COUNTER("Particles.PixelsDrawn", PixelsDrawn);

PROFILE_COUNTER("Global.MemoryProcessSizeWin", m_versionInfo->WorkingSet() >> 10 );
PROFILE_COUNTER("Global.VirtualMemoryWin", (m_versionInfo->TotalVirtualMemory() - m_versionInfo->AvailableVirtualMemory()) >> 10 );
```

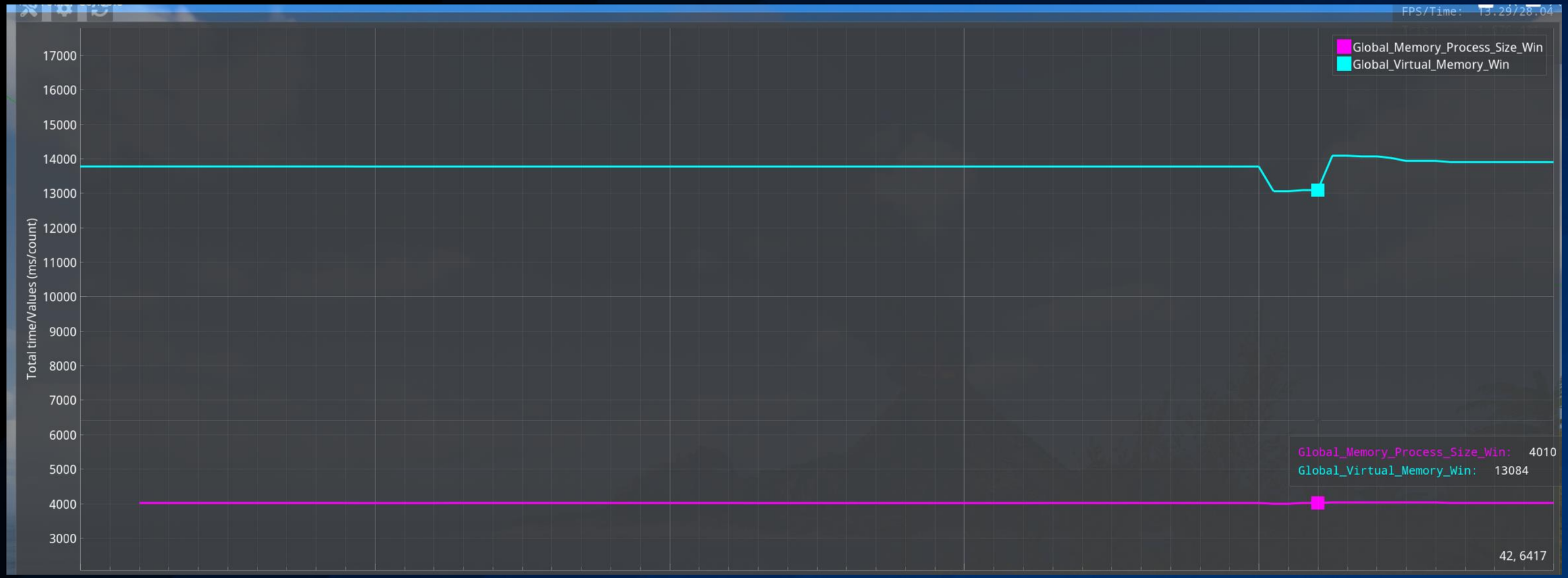
# Data sampling and Aggregation

- Average
- Min
- Max
- Median
- Sum
- Count

# Что собираем из ОС

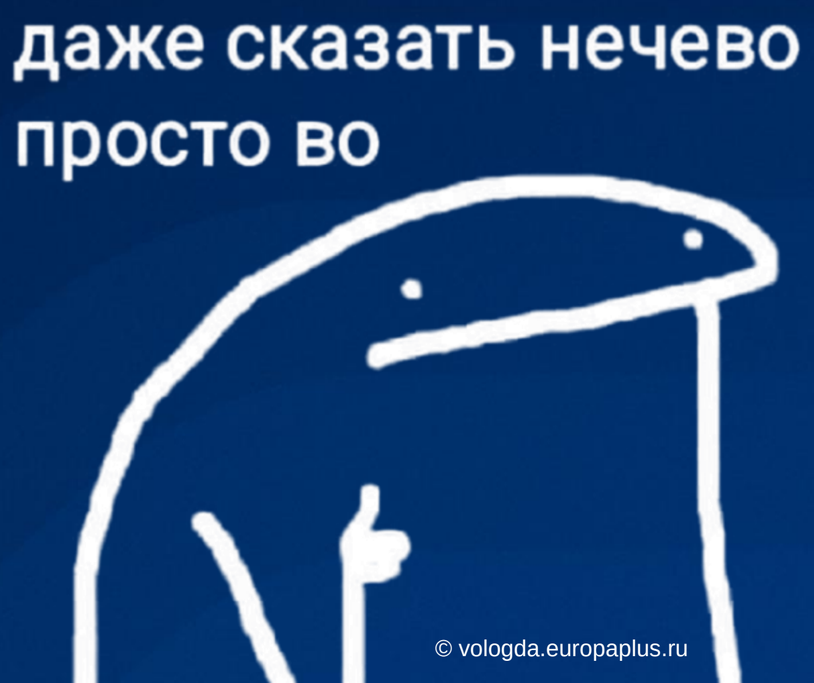
- Virtual Memory: Reserved, Committed, Private, Shared (Peak + Current)
- Physical Memory: Working Set (Peak + Current) + Page Faults/sec.
- Page File: Reads, Writes
- IO: Reads, Writes, Others (Count, Total, Bytes/sec.)
- И некоторые еще :)

# Примеры метрик собираемых из ОС



ЛЕСТА

# ИДЕАЛЬНЫЙ ПРОФАЙЛЕР – ТОЧЕН, БЫСТР, «ПРОСТ», УДОБЕН




© vologda.europaplus.ru

# Вопросы

ЛЕСТА

# ЛИТЕРАТУРА


- https://en.wikipedia.org/wiki/Time_Stamp_Counter
- https://learn.microsoft.com/en-us/windows/win32/etw/about-event-tracing
- https://learn.microsoft.com/en-us/windows/win32/direct3d12/timing
- https://www.ntcore.com/files/inject2it.htm
- https://www.youtube.com/watch?v=vsvhlKlwu88
- https://www.youtube.com/watch?v=AjTl53I_qzY

# Спасибо за внимание!

ЛЕСТА