C++ in Huge AAA Games

Nicolas Fleury, Technical Architect
Ubisoft Montreal

FARCRY3
PERSIA
OF TIME
GS

Outline

Situation

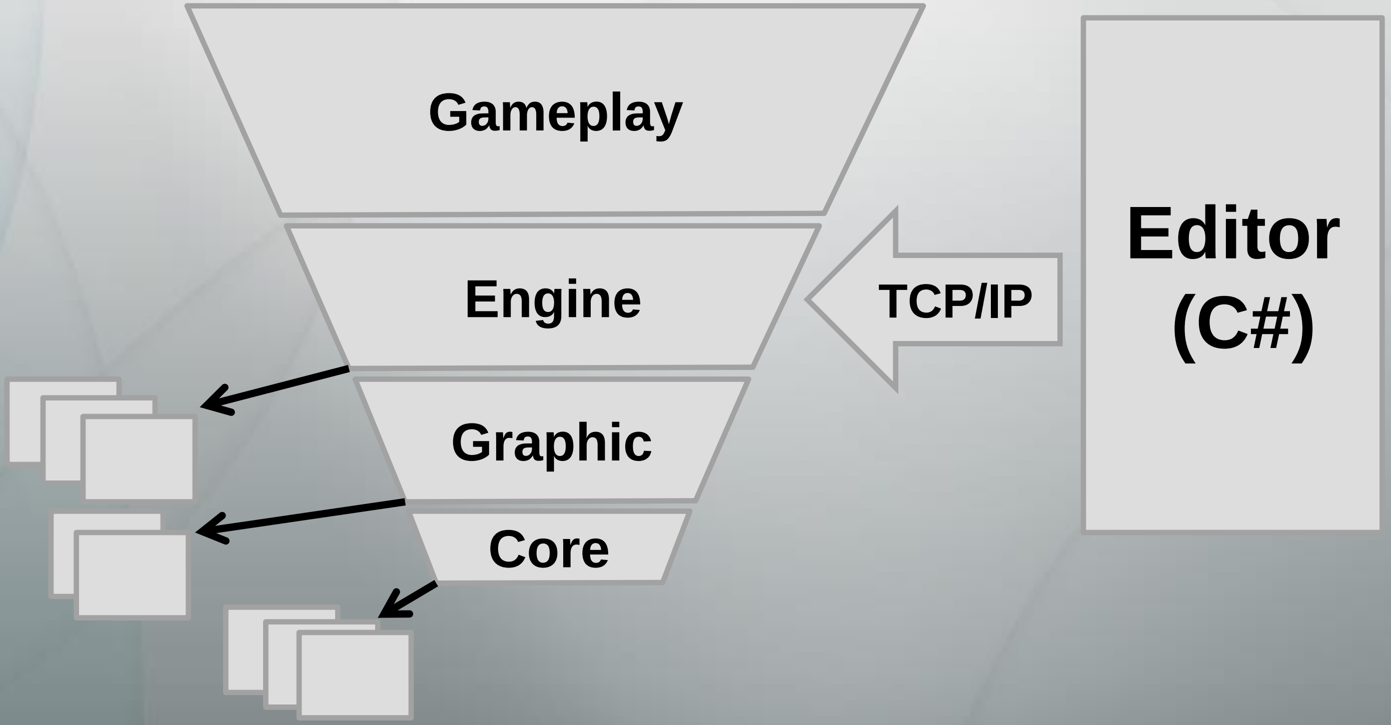
Ubisoft Montreal

- 2600+ employees. Biggest game studio in the world.
- Projects up to 1000 employees worldwide. Up to 400 in Mtl.
- Technology Group in Mtl of 300 developers.
- Windows-centric development environment.

Big Games

- Assassin's Creed Unity:
  - 6.5 M C++ LOC for entire team code.
  - 9 M more C++ LOC from outside project.
  - 5 M C# LOC.
- Rainbow Six: Siege:
  - 3.5 M C++ LOC for engine code from game team.
  - 4.5 M C++ LOC from Technology Group.
  - Rebuild All: 3 min to 5 min.

Code Structure



What we Don't Use

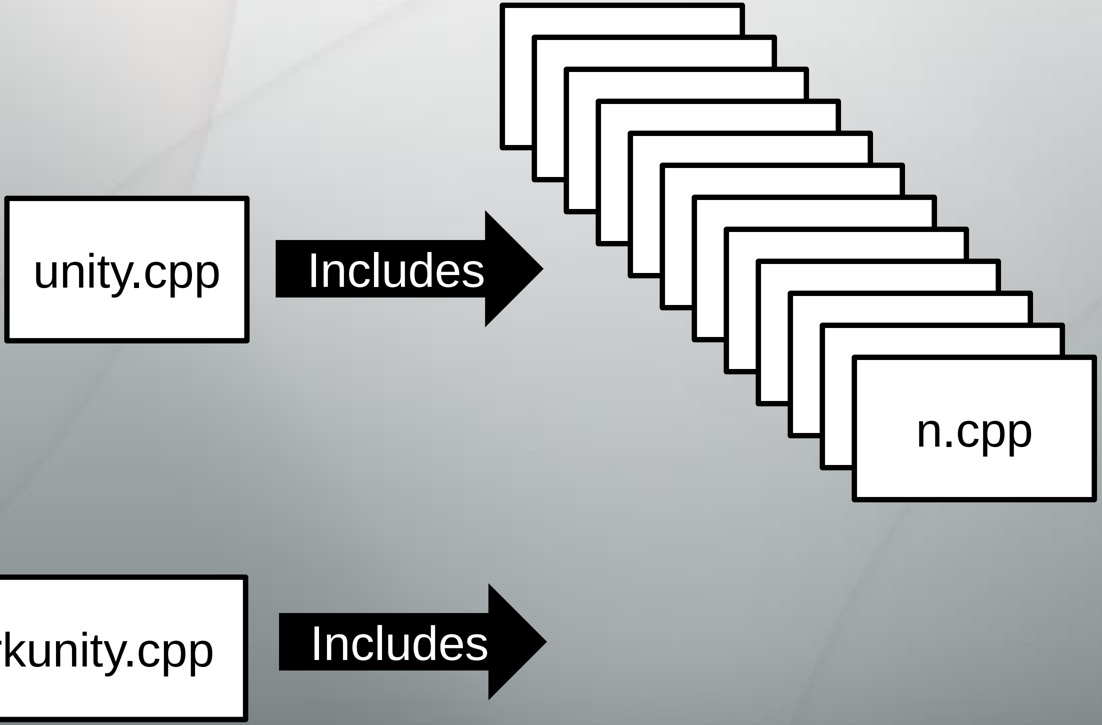
- No RTTI
- No Exception Handling
- No STL containers
- No Boost includes in Engine

Iteration Time

FastBuild

- Replacing MSBuild for C++
- Open Source (permissive) made by Franta Fulin
- Smarter DLL dependencies
- Better CPU usage
- Distribution and caching
- Unity builds built-in

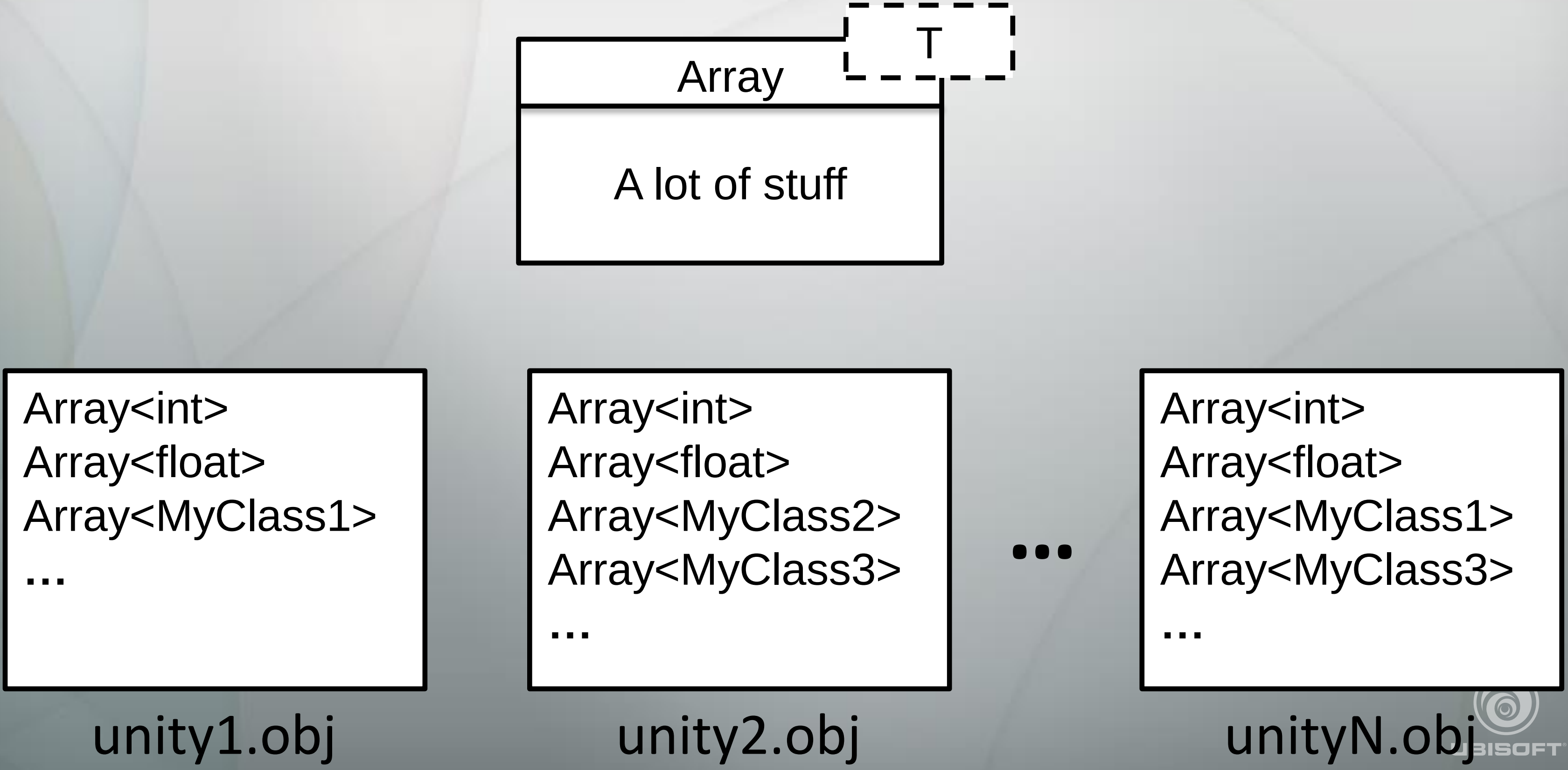
Unity Builds



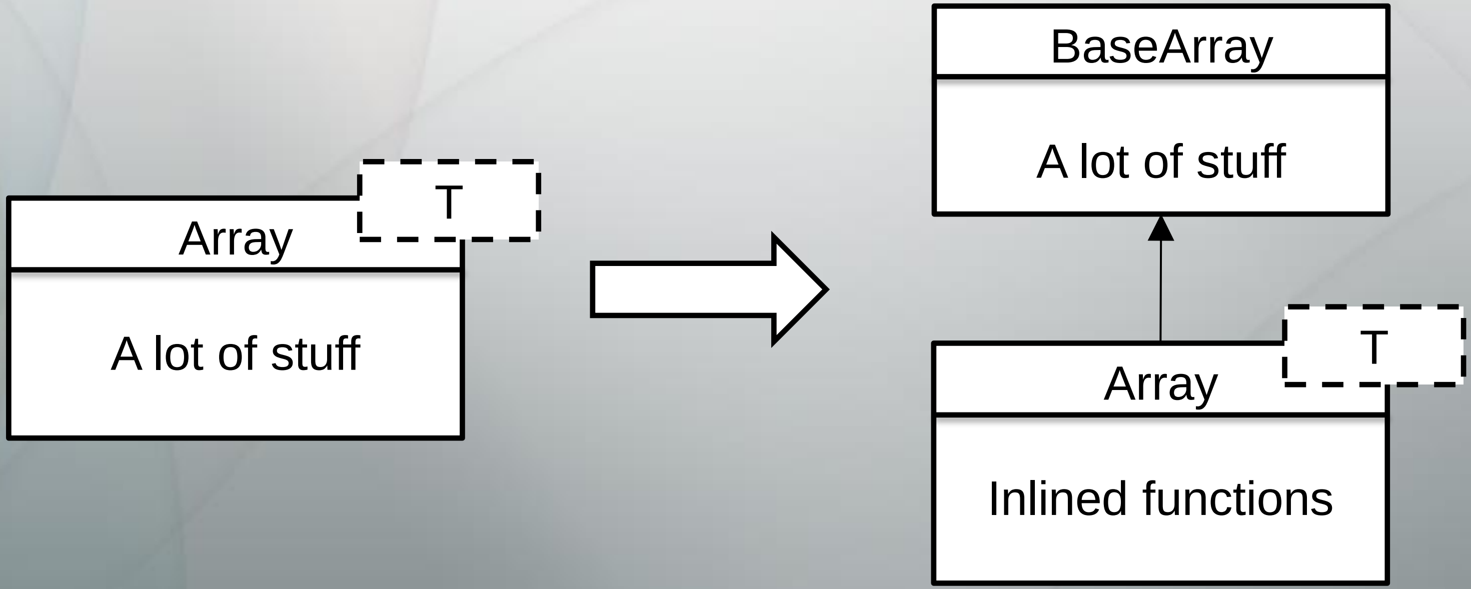
Other Points

- Precompiled headers
- /Ob1 in Debug targets
- Template classes with non-template base classes

Templates


T
Array
A lot of stuff
Array<int>
Array<float>
Array<MyClass1>
…
unity1.obj
Array<int>
Array<float>
Array<MyClass2>
Array<MyClass3>
…
unity2.obj
…
Array<int>
Array<float>
Array<MyClass1>
Array<MyClass3>
…
unityN.obj

Templates: Unify Code



Generated Code

- IDL for object model
- Generated code regions in corresponding .h and .cpp files
- Avoiding some meta-programming
- Custom Edit and Continue through our own programming language generating C++.

Tools

- .obj Analyzer
  - Total symbol sizes for all translation units together
- Useless #include Remover
  - We have our own tool
  - Google’s include-what-you-use looks better

Performance

Performance Importance

- Last console generation was 8 years
- 90/10 principle: 10% of code running 90% of time.
- Frame rate reality pushing us.

Example

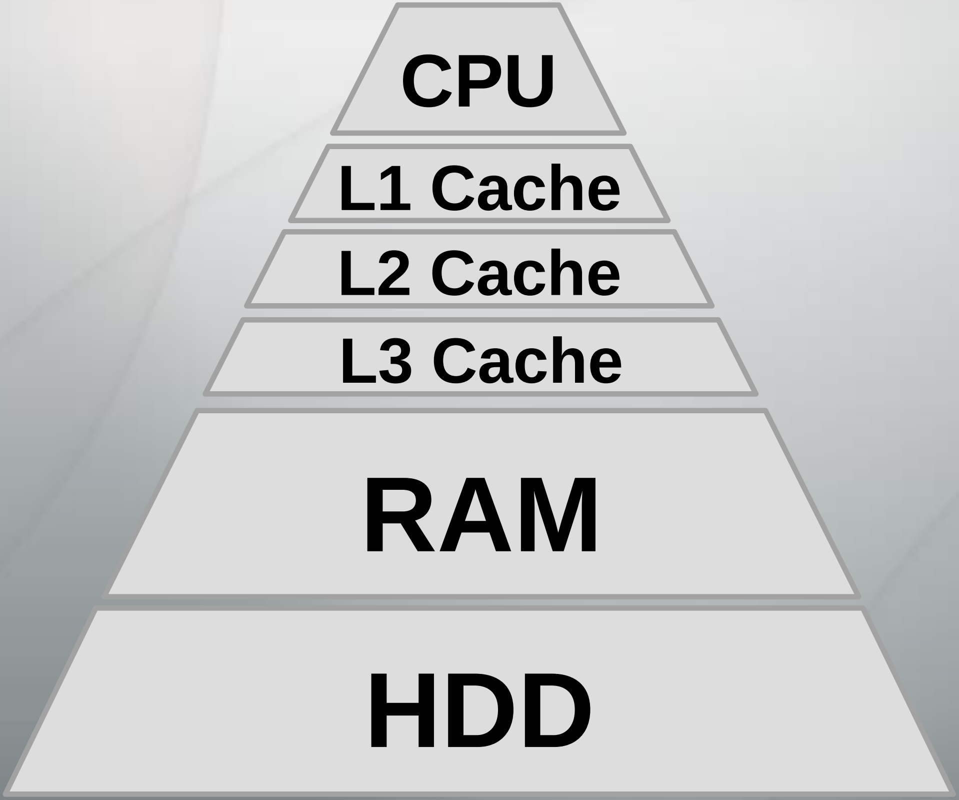
```
struct Data
{
    Data() { for (int i = 0; i < 64; ++i)
        values[i] = i; }
    int values[64];
};


Data* data = new Data[1 << 20];  // huge size
```

Example

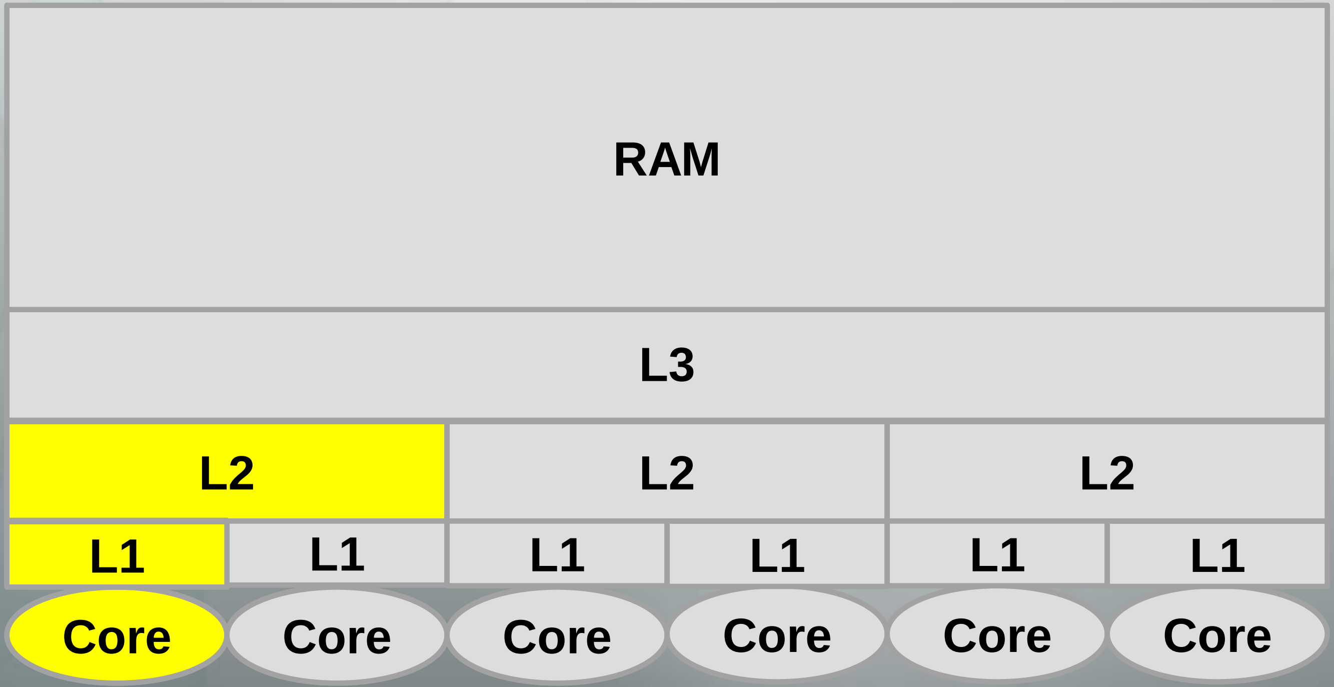
```
int total = 0;

for (int i = 0; i < size1; ++i)
    for (int j = 0; j < size2; ++j)
        total += data[j].values[i];

for (int j = 0; j < size2; ++j)
    for (int i = 0; i < size1; ++i)
        total += data[j].values[i];
```

On my PC:
8 times faster

Memory Hierarchy

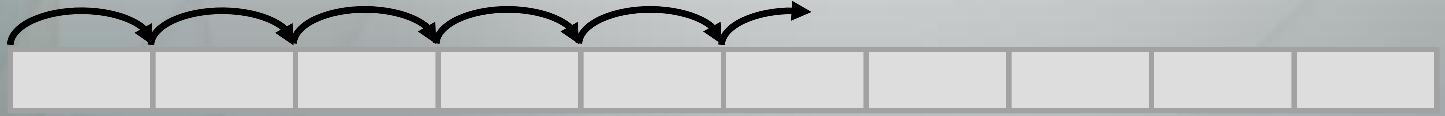


Cache



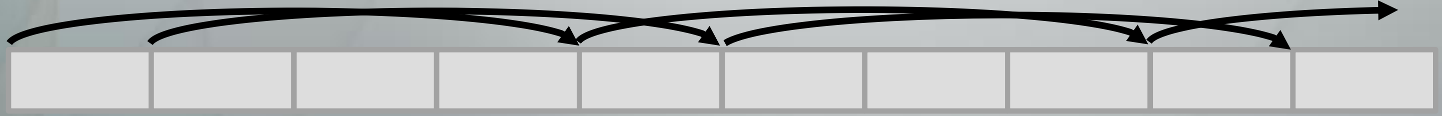
Data Cache Miss

```
for (int j = 0; j < size2; ++j)
    for (int i = 0; i < size1; ++i)
        total += data[j].values[i];
```



Data Cache Miss

```
for (int j = 0; j < size2; ++j)
    for (int i = 0; i < size1; ++i)
        total += data[j].values[i];
```



Another Example

```
struct MyClass {
    int64_t m_Total = 0;
    void UpdateTotal(int* values, int count);
};
```

Another Example

```
for (int i = 0; i < count; ++i)
{
    m_Total += values[i];
}

int64_t total = 0;
for (int i = 0; i < count; ++i)
{
    total += values[i];
}
m_Total = total;
```

On my PC: 12 times faster

SingletonStorers

```
struct MyLibSingletonStorer
{
    MyManager m_MyManager;
    MyOtherManager m_MyOtherManager;
    ...
};
```

Singletons

```
template <typename T>
class Singleton {
  protected: Singleton() { ms_Inst = this; }
  private: static T* ms_Inst = nullptr;
  public: static T* GetInst() { return m_Inst;}
};

class MyManager : public Singleton<MyManager> …
```

GlobalSingleton

```
struct MyLibSingletonStorer
{
  GlobalSingleton<MyManager>::Scope m_MyManager;
  MyOtherManager m_MyOtherManager;
  ...
};
```

GlobalSingleton

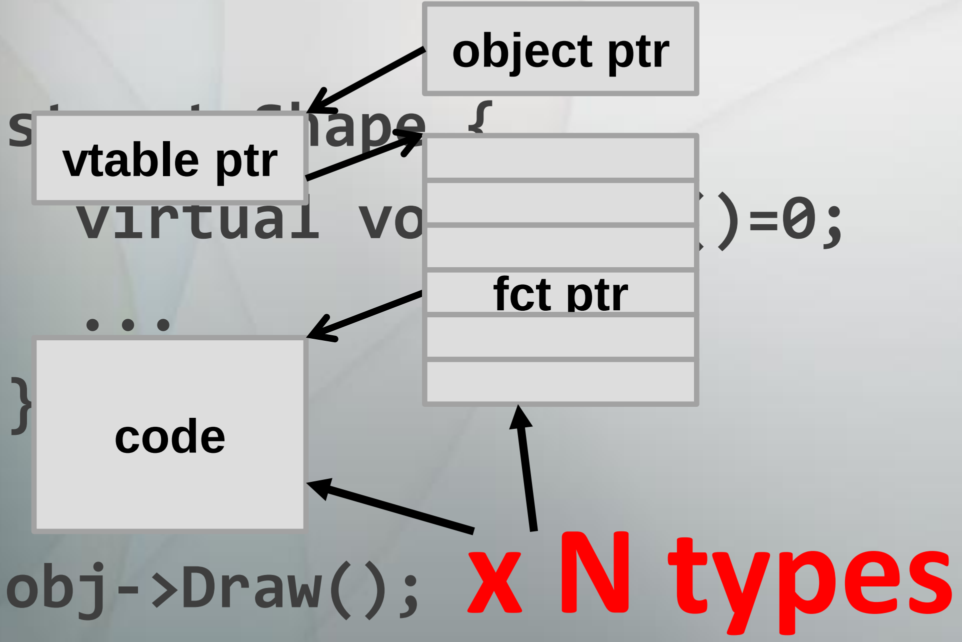
```
void Construct() { new (&m_Data.m_Buffer)T(); }
void Destroy() { GetInst().~T(); }
T& GetInst() { return *(T*)&m_Data.m_Buffer; }
```

Code Cache Miss

```
struct Shape {
  virtual void Draw()=0;
  ...
};

obj->Draw();
```

```
switch (shapeType) {
  case CIRCLE_SHAPE:
    ...
```

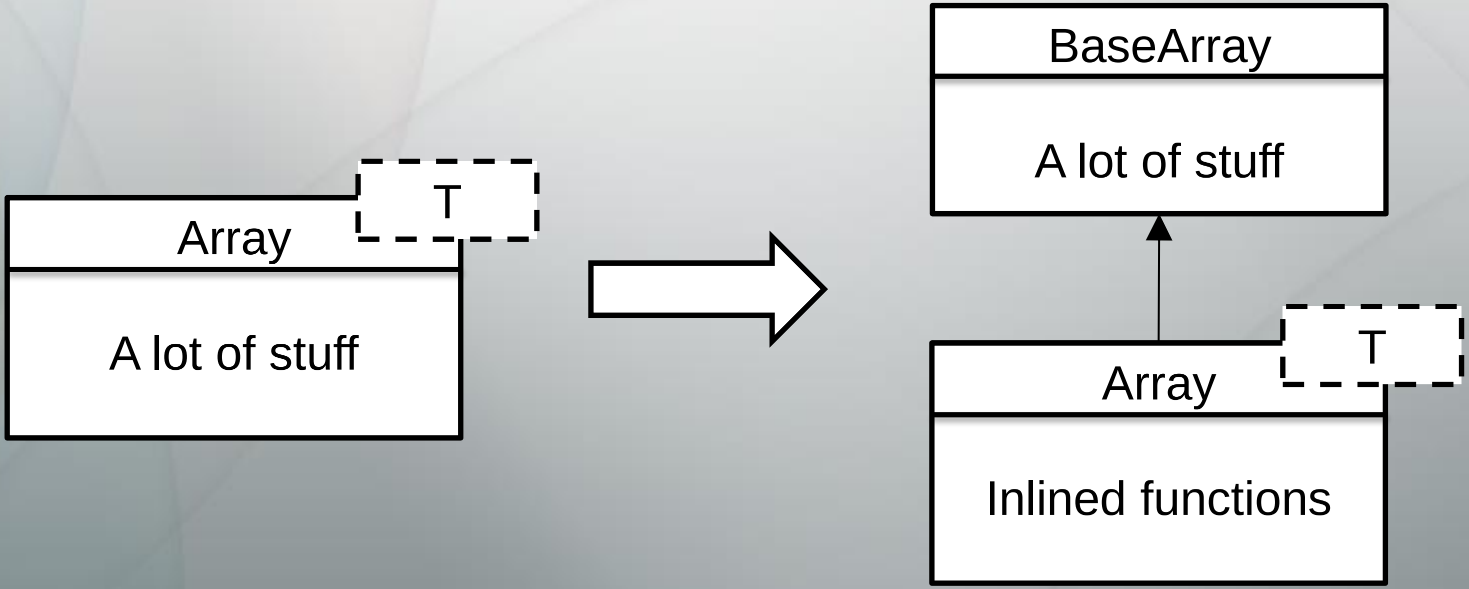
Code Cache Miss



```
s     Shape {
  virtual vo    ()=0;
  ...
}

obj->Draw();
```

```
switch (shapeType) {
  case CIRCLE_SHAPE:
    ...
```

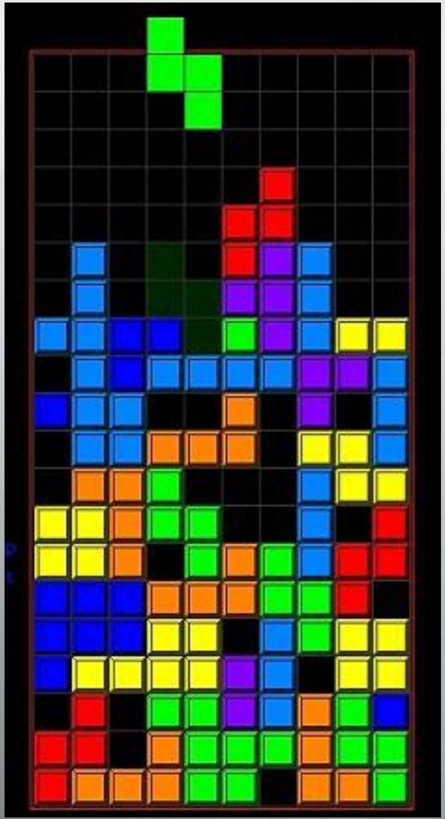
Code Cache Miss



Avoiding Heap

- Heavy
- Global
- Fragmentation

Avoiding Heap



Avoiding Heap

```
void Foo()
{
    Array<ubiU32> values;
    ...
```

Avoiding Heap

```
void Foo()
{
    InplaceArray<ubiU32, 8> values;
    ...
```

Avoiding Heap

```
if (IsPtrOnStack(this))
    FrameAllocator::Allocate(...);
else
    ...
```

Debugging

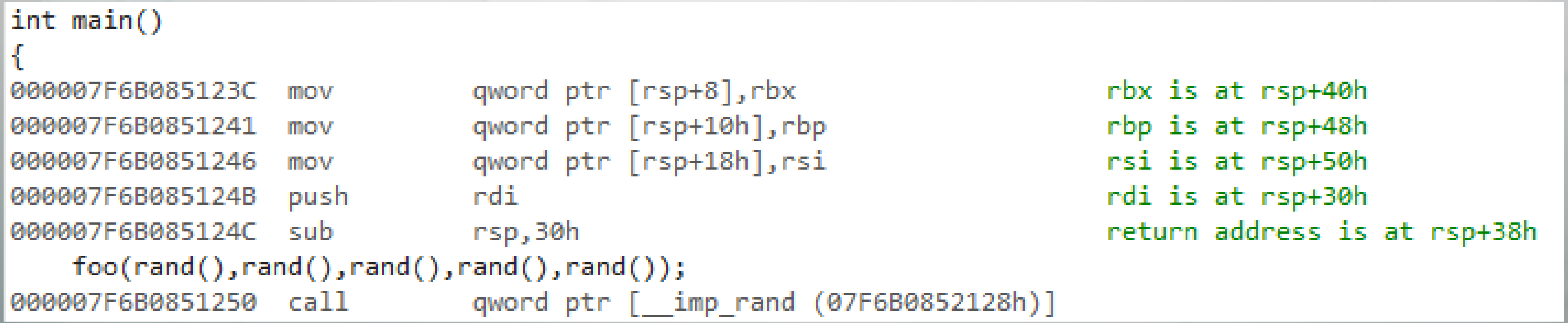
Challenges

- Huge multithreaded codebase
- Some bugs only reproducible in optimized targets
- Avoid recompiling for debug options
- Debug targets must be fast to be usable

Some Disabled Stuff

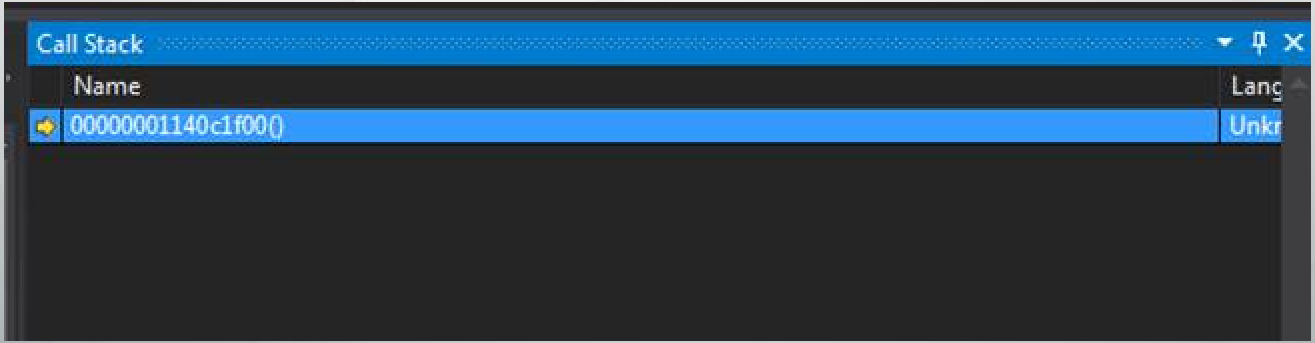
- Debug iterators
- Visual Studio Debugger Heap (_NO_DEBUG_HEAP=1)
- Windows Fault Tolerant Heap

Debugging Release Code



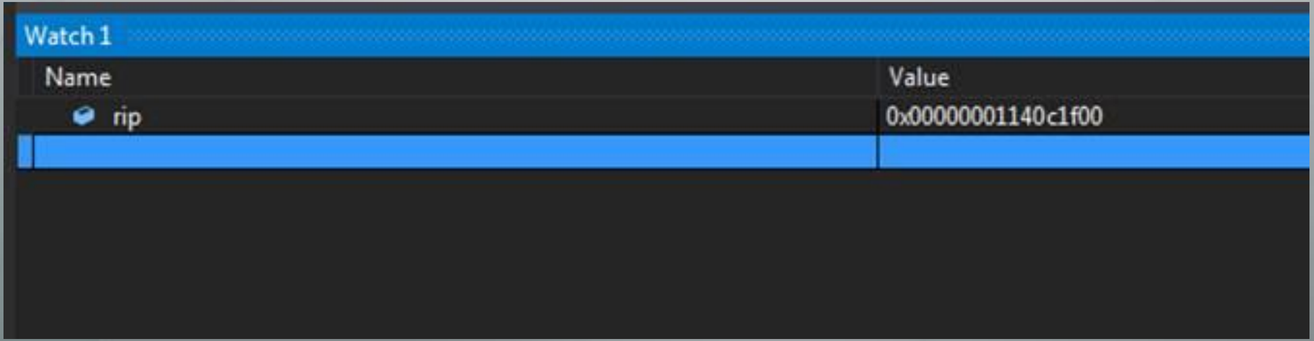
```
int main()
{
000007F6B085123C  mov          qword ptr [rsp+8],rbx                      rbx is at rsp+40h
000007F6B0851241  mov          qword ptr [rsp+10h],rbp                    rbp is at rsp+48h
000007F6B0851246  mov          qword ptr [rsp+18h],rsi                    rsi is at rsp+50h
000007F6B085124B  push         rdi                                        rdi is at rsp+30h
000007F6B085124C  sub          rsp,30h                                    return address is at rsp+38h
    foo(rand(),rand(),rand(),rand(),rand());
000007F6B0851250  call         qword ptr [__imp_rand (07F6B0852128h)]
```

My Callstack is RIP



Call Stack

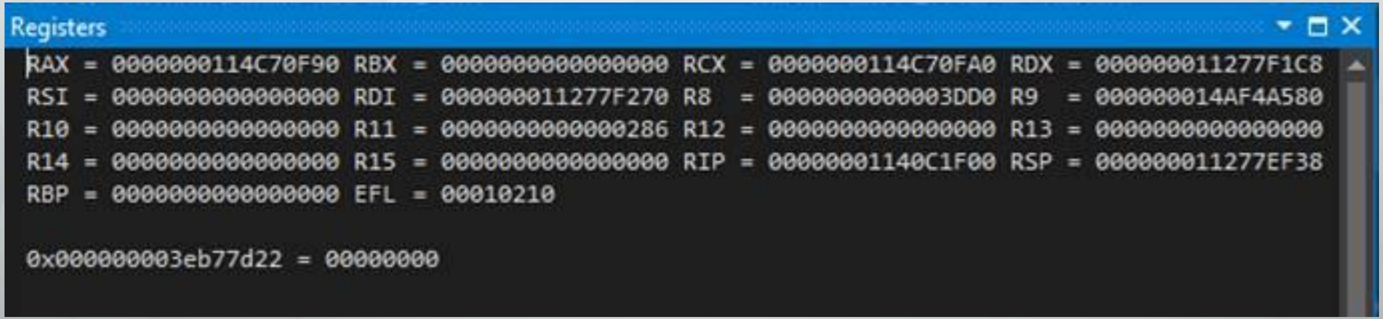
Name	Lang
00000001140c1f00()	Unk



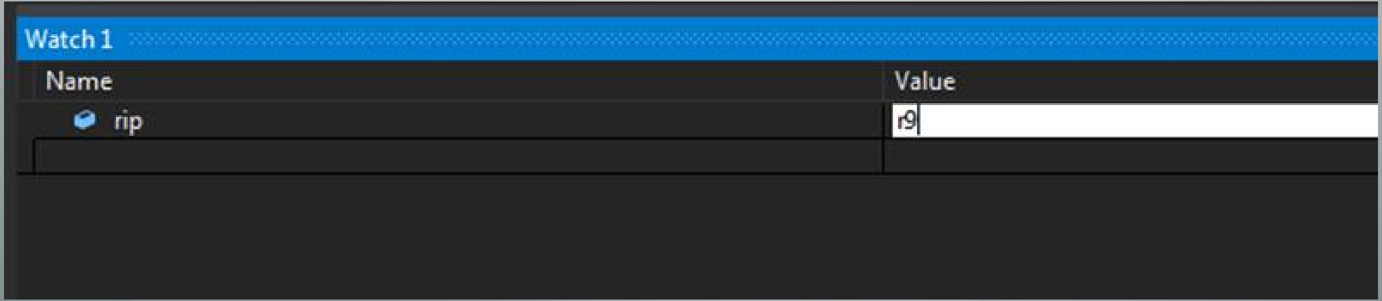
Watch 1

Name	Value
rip	0x00000001140c1f00

My Callstack is RIP

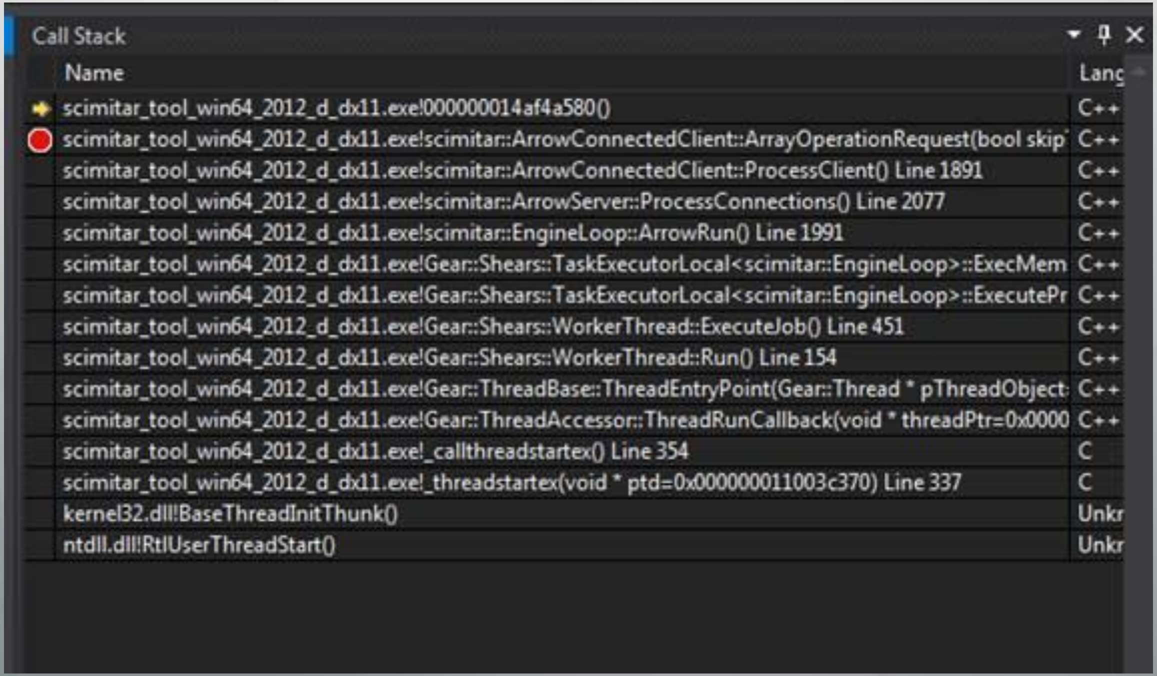

Registers

```
RAX = 0000000114C70F90 RBX = 0000000000000000 RCX = 0000000114C70FA0 RDX = 000000011277F1C8
RSI = 0000000000000000 RDI = 000000011277F270 R8  = 0000000000003DD0 R9  = 000000014AF4A580
R10 = 0000000000000000 R11 = 0000000000000286 R12 = 0000000000000000 R13 = 0000000000000000
R14 = 0000000000000000 R15 = 0000000000000000 RIP = 00000001140C1F00 RSP = 000000011277EF38
RBP = 0000000000000000 EFL = 00010210

0x000000003eb77d22 = 00000000
```


Watch 1

Name	Value
rip	r9

My Callstack is RIP

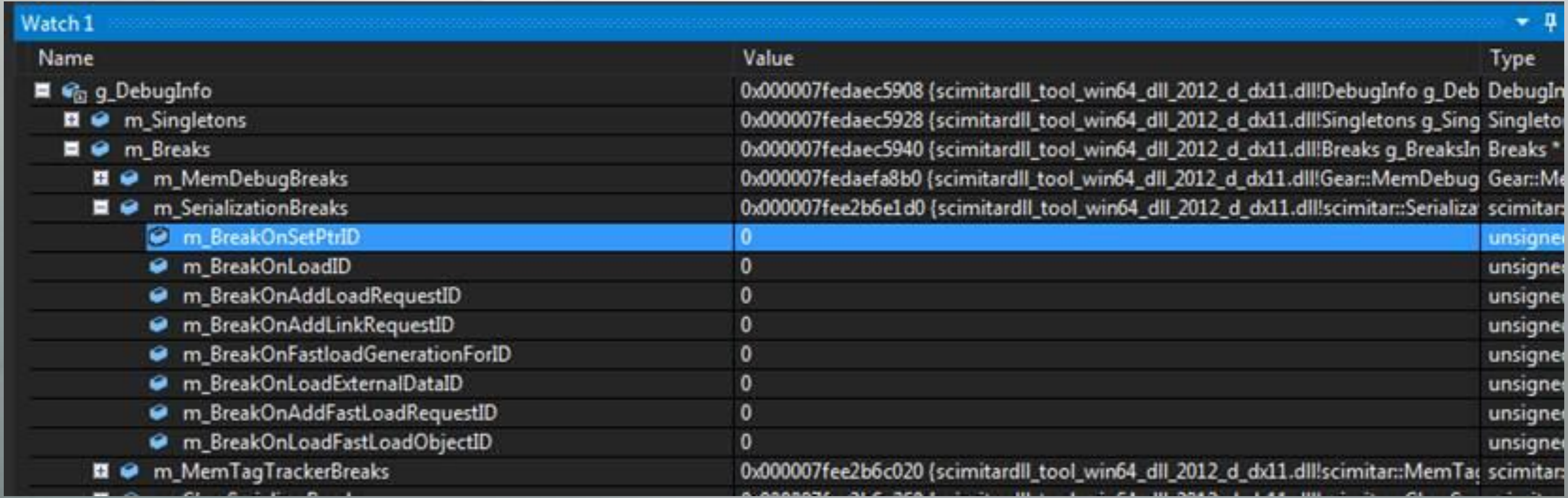


```
Name                                                                                              Lang
scimitar_tool_win64_2012_d_dx11.exe!000000014af4a580()                                            C++
scimitar_tool_win64_2012_d_dx11.exe!scimitar::ArrowConnectedClient::ArrayOperationRequest(bool skip   C++
scimitar_tool_win64_2012_d_dx11.exe!scimitar::ArrowConnectedClient::ProcessClient() Line 1891     C++
scimitar_tool_win64_2012_d_dx11.exe!scimitar::ArrowServer::ProcessConnections() Line 2077         C++
scimitar_tool_win64_2012_d_dx11.exe!scimitar::EngineLoop::ArrowRun() Line 1991                    C++
scimitar_tool_win64_2012_d_dx11.exe!Gear::Shears::TaskExecutorLocal<scimitar::EngineLoop>::ExecMem C++
scimitar_tool_win64_2012_d_dx11.exe!Gear::Shears::TaskExecutorLocal<scimitar::EngineLoop>::ExecutePr C++
scimitar_tool_win64_2012_d_dx11.exe!Gear::Shears::WorkerThread::ExecuteJob() Line 451             C++
scimitar_tool_win64_2012_d_dx11.exe!Gear::Shears::WorkerThread::Run() Line 154                    C++
scimitar_tool_win64_2012_d_dx11.exe!Gear::ThreadBase::ThreadEntryPoint(Gear::Thread * pThreadObject: C++
scimitar_tool_win64_2012_d_dx11.exe!Gear::ThreadAccessor::ThreadRunCallback(void * threadPtr=0x0000 C++
scimitar_tool_win64_2012_d_dx11.exe!_callthreadstartex() Line 354                                 C
scimitar_tool_win64_2012_d_dx11.exe!_threadstartex(void * ptd=0x000000011003c370) Line 337        C
kernel32.dll!BaseThreadInitThunk()                                                                Unkr
ntdll.dll!RtlUserThreadStart()                                                                    Unkr
```

Memory Tagging

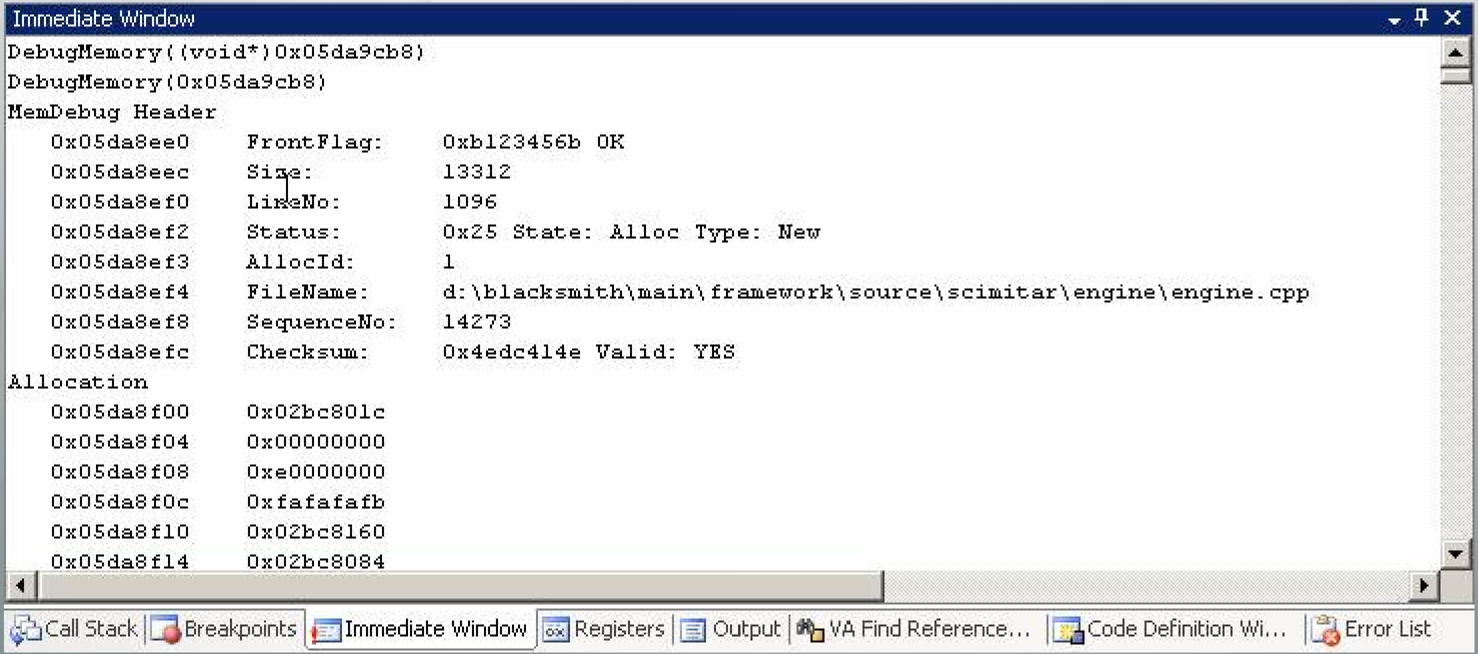
```
Particle* particle =
  ubiNew(Particle, "FX Particle", fxManager);
```

Breaks



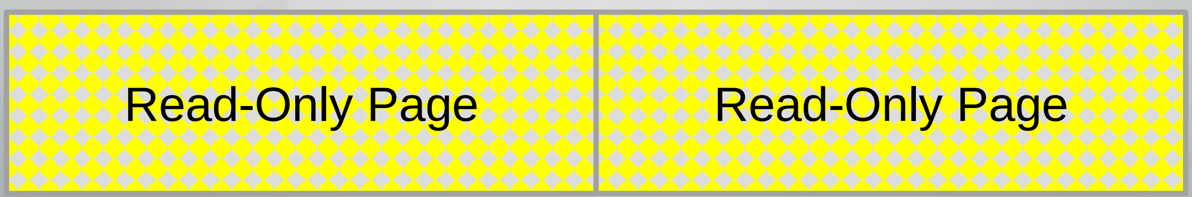
```
Watch 1
Name                                        Value                                                                                  Type
⊟ g_DebugInfo                               0x000007fedaec5908 {scimitardll_tool_win64_dll_2012_d_dx11.dll!DebugInfo g_Deb        DebugIn
  ⊞ m_Singletons                            0x000007fedaec5928 {scimitardll_tool_win64_dll_2012_d_dx11.dll!Singletons g_Sing      Singleto
  ⊟ m_Breaks                                0x000007fedaec5940 {scimitardll_tool_win64_dll_2012_d_dx11.dll!Breaks g_BreaksIn      Breaks *
    ⊞ m_MemDebugBreaks                      0x000007fedaefa8b0 {scimitardll_tool_win64_dll_2012_d_dx11.dll!Gear::MemDebug         Gear::Me
    ⊟ m_SerializationBreaks                 0x000007fee2b6e1d0 {scimitardll_tool_win64_dll_2012_d_dx11.dll!scimitar::Serializa    scimitar
        m_BreakOnSetPtrID                   0                                                                                      unsigne
        m_BreakOnLoadID                     0                                                                                      unsigne
        m_BreakOnAddLoadRequestID           0                                                                                      unsigne
        m_BreakOnAddLinkRequestID           0                                                                                      unsigne
        m_BreakOnFastloadGenerationForID    0                                                                                      unsigne
        m_BreakOnLoadExternalDataID         0                                                                                      unsigne
        m_BreakOnAddFastLoadRequestID       0                                                                                      unsigne
        m_BreakOnLoadFastLoadObjectID       0                                                                                      unsigne
    ⊞ m_MemTagTrackerBreaks                 0x000007fee2b6c020 {scimitardll_tool_win64_dll_2012_d_dx11.dll!scimitar::MemTag       scimitar
```

Memory Corruption



```
DebugMemory((void*)0x05da9cb8)
DebugMemory(0x05da9cb8)
MemDebug Header
   0x05da8ee0    FrontFlag:    0xb123456b OK
   0x05da8eec    Size:         13312
   0x05da8ef0    LineNo:       1096
   0x05da8ef2    Status:       0x25 State: Alloc Type: New
   0x05da8ef3    AllocId:      1
   0x05da8ef4    FileName:     d:\blacksmith\main\framework\source\scimitar\engine\engine.cpp
   0x05da8ef8    SequenceNo:   14273
   0x05da8efc    Checksum:     0x4edc414e Valid: YES
Allocation
   0x05da8f00    0x02bc801c
   0x05da8f04    0x00000000
   0x05da8f08    0xe0000000
   0x05da8f0c    0xfafafafb
   0x05da8f10    0x02bc8160
   0x05da8f14    0x02bc8084
```

Memory Corruption



References

http://realtimecollisiondetection.net/pubs/GDC03_Ericson_Memory_Optimization.ppt

http://fastbuild.org/docs/home.html

http://blog.teachbook.com.au/index.php/2012/02/memory-hierarchy/

http://tfpsly.free.fr/english/optimization.html

http://www.gamasutra.com/view/feature/132084/sponsored_feature_common_.php

http://fgiesen.wordpress.com/2014/07/07/cache-coherency/

Questions?