Building a Low-Fragmentation Memory System for 64-bit Games

Aaron MacDougall
Senior Systems Programmer – SCE London Studio

London Studio

UBM

Background

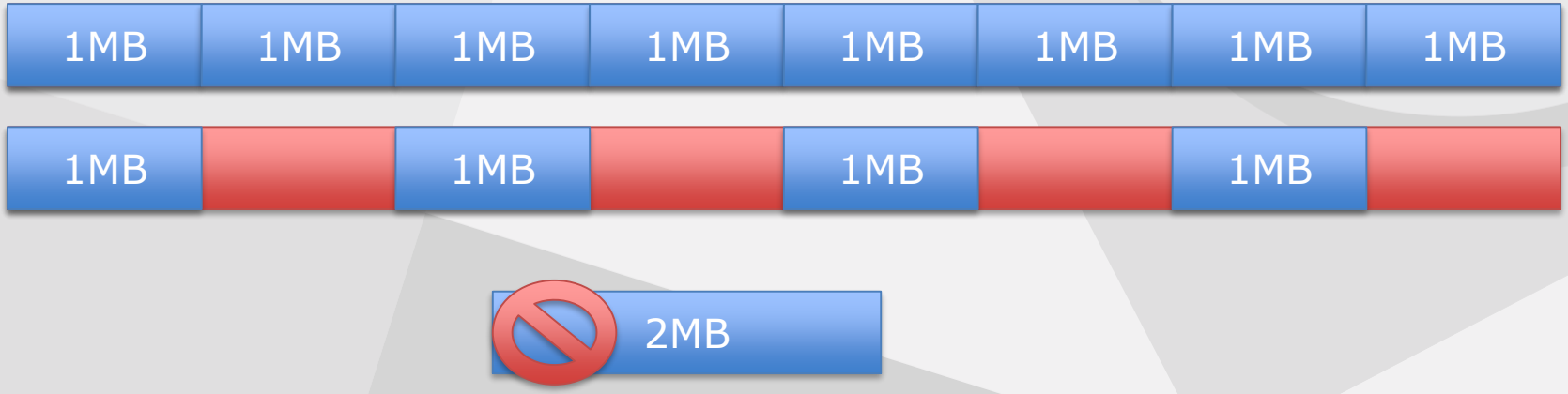
- Old memory system ported from PlayStation®3
- Fixed sized memory pools
- Emulated VRAM

Problems

- Wasted a lot of memory
  - Every pool sized for worst case
  - Overhead with small allocations
- Suffered from fragmentation
- Texture streaming impractical

Memory Fragmentation

- Heap fragmented in small non-contiguous blocks
- Allocations can fail despite enough memory
- Caused by mixed allocation lifetimes

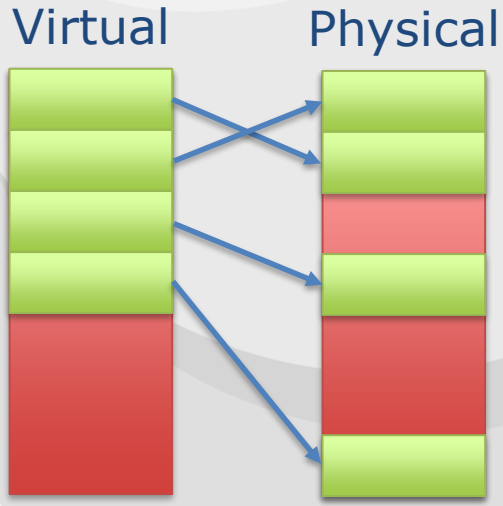


Design Goals

- Low fragmentation
- High utilisation
- Simple configuration
- Support PlayStation®4 OS and PC
- Support efficient texture streaming
- Comprehensive debugging support

Virtual Memory

- Process uses virtual addresses
- Virtual addresses mapped to physical addresses
- CPU looks up physical address
- Requires OS and hardware support



Benefits of Virtual Memory

- Reduced memory fragmentation
  - Fragmentation is ***address*** fragmentation
  - We use virtual addresses
  - Virtual address space is larger than physical
  - Contiguous virtual memory not contiguous in physical memory

Virtual Address Space

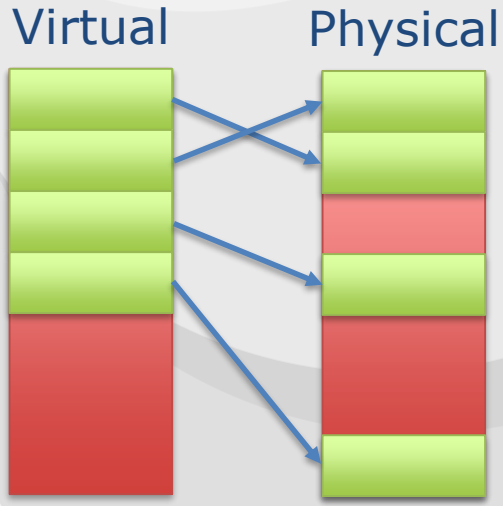
944GB

Physical Memory

8GB

Memory Pages

- Mapped in pages
- x64 supports:
  - 4kB and 2MB pages
- PlayStation®4 OS uses:
  - 16kB (4x4kB) and 2MB
- GPU has more sizes



Page Sizes

- 2MB pages fastest
- 16kB pages wastes less memory
- We use 64kB (4x16kB pages)
  - Smallest optimal size for PlayStation®4 GPU
- Also use 16kB for special cases

Onion Bus & Garlic Bus

- CPU & GPU can access both
  - But at different bandwidths

- Onion = fast CPU access
- Garlic = fast GPU access

Flexible & Direct Memory on PlayStation®4

- Same virtual address space
- Flexible
  - 512MB pre-allocated by OS
  - 16kB pages mapped to Onion (CPU bus)
- Direct
  - 16kB or 2MB pages
  - Must be allocated and mapped to Onion or Garlic (GPU bus)
- Both emulated on PC using 64kB pages

Our Memory System

- Splits up the entire virtual address space
- Physical memory mapped on demand
- Allocator modules manage their own space
- Each module specialised
- Allocator objects are the interface to the system

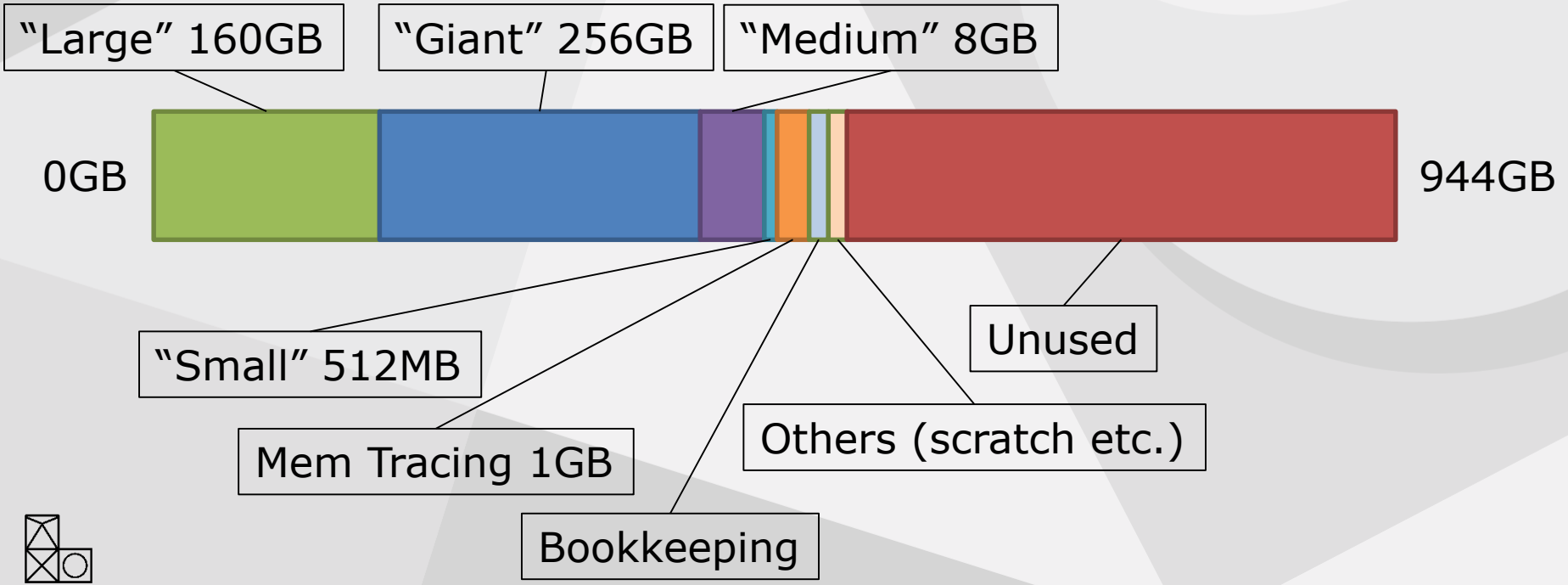
Allocator

```
class Allocator
{
public:
     virtual void* Allocate(size_t size, size_t align) = 0;
     virtual void Deallocate(void* pMemory) = 0;
     virtual size_t GetSize(void* pMemory) { return 0; }

     const char* GetName(void) const;
};
```

Example – GeneralAllocator

```
void* GeneralAllocator::Allocate(size_t size, size_t align)
{
    if (SmallAllocator::Belongs(size, align))
        return SmallAllocator::Allocate(size, align);
    else if (m_mediumAllocator.Belongs(size, align))
        return m_mediumAllocator.Allocate(size, align);
    else if (LargeAllocator::Belongs(size, align))
        return LargeAllocator::Allocate(size, m_mappingFlags);
    else if (GiantAllocator::Belongs(size, align))
        return GiantAllocator::Allocate(size, m_mappingFlags);

    return nullptr;
}
```

Our Virtual Address Space

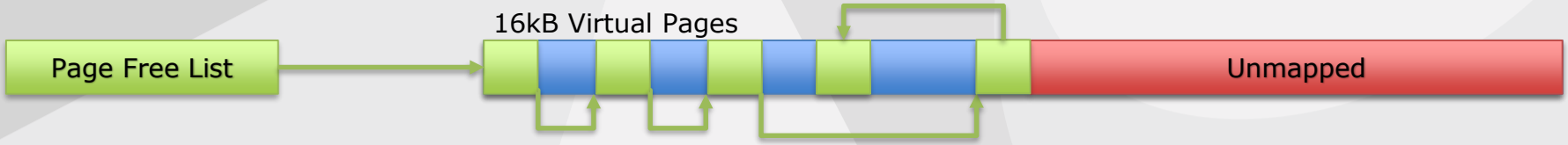


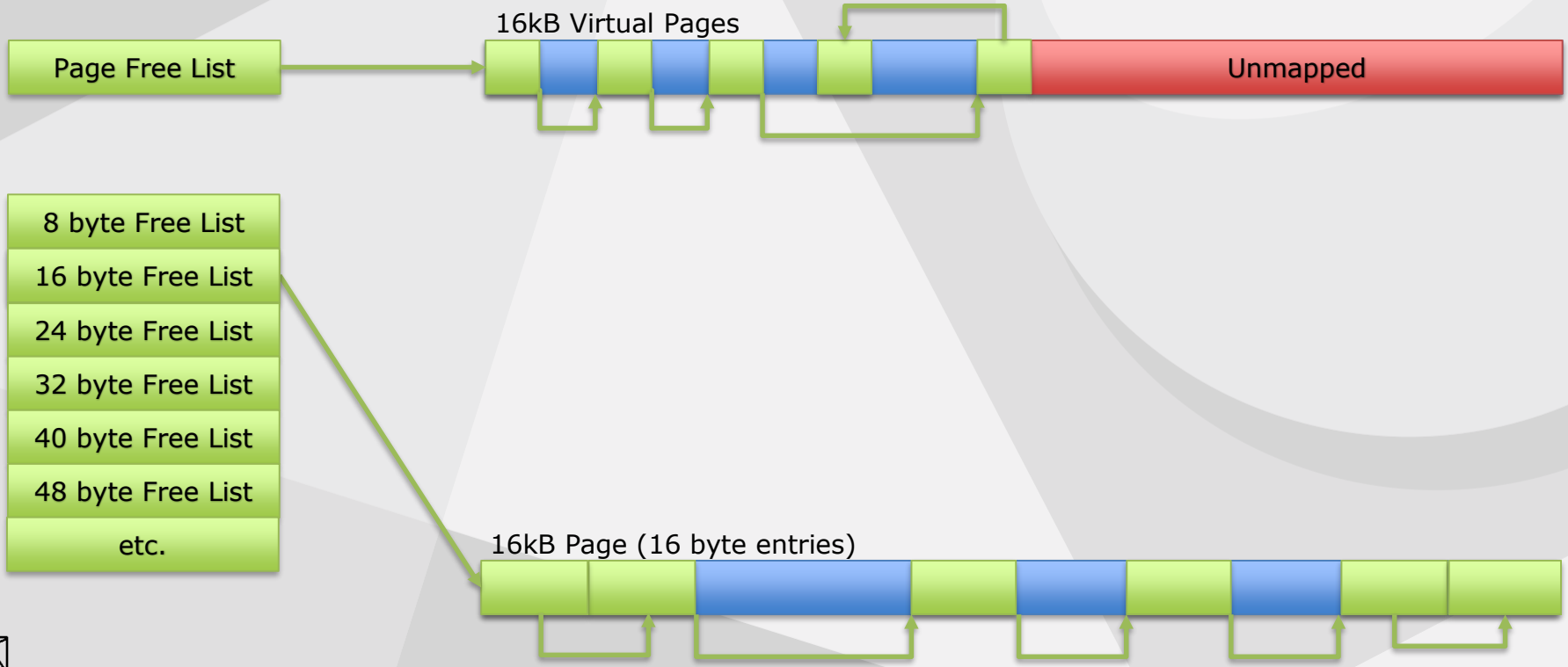
Physical Memory on PlayStation®4

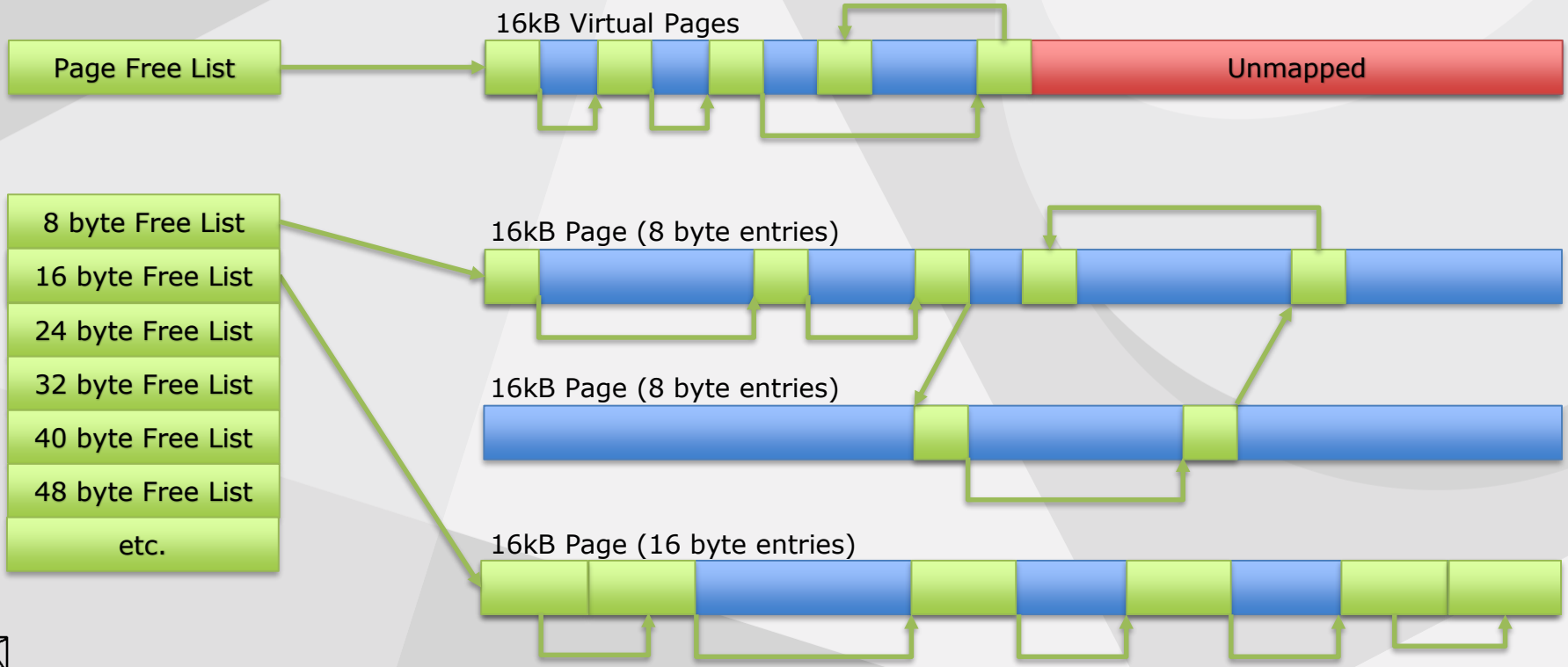
- Flexible memory already allocated
- Direct memory split into 64kB pages
- Allocated and deallocated on demand
  - Memory bus set when allocated
- Two free lists containing unused pages
  - Onion
  - Garlic

Small Allocation Module

- Majority of allocations are <= 64 bytes
- ~250,000 allocations - ~25MB
- Pack together to prevent fragmentation
  - 16kB pages of same-sized allocations
  - No headers


16kB Virtual Pages
Page Free List
Unmapped

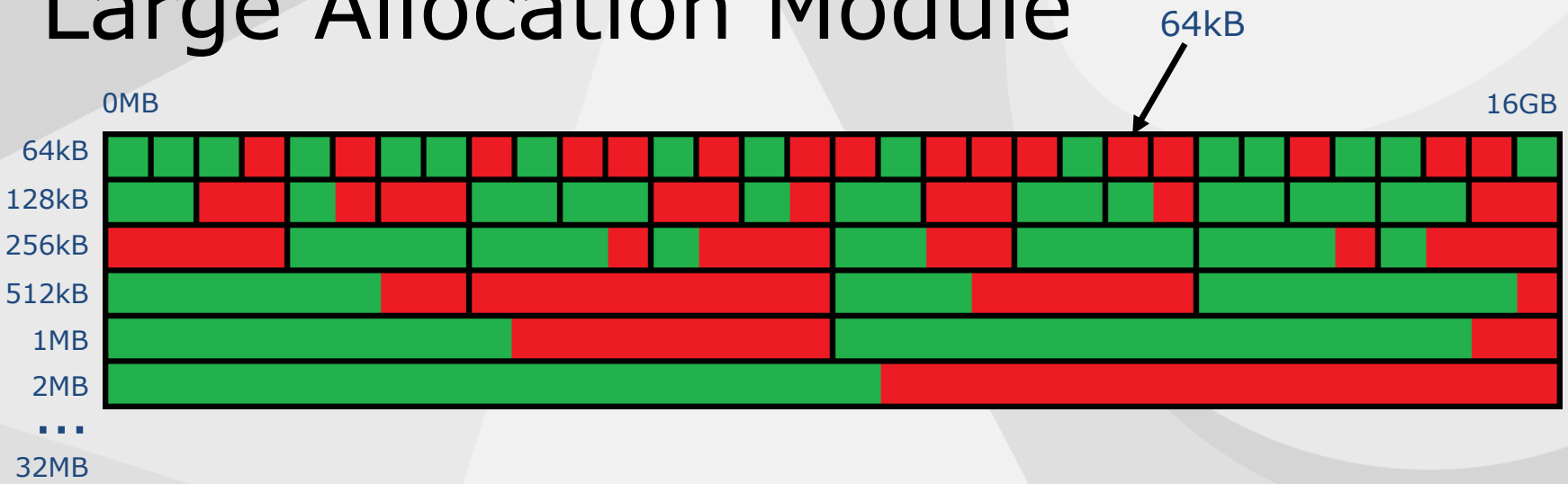

16kB Virtual Pages
Page Free List
Unmapped
8 byte Free List
16 byte Free List
24 byte Free List
32 byte Free List
40 byte Free List
48 byte Free List
etc.
16kB Page (16 byte entries)


16kB Virtual Pages
Page Free List
Unmapped
8 byte Free List
16 byte Free List
24 byte Free List
32 byte Free List
40 byte Free List
48 byte Free List
etc.
16kB Page (8 byte entries)
16kB Page (8 byte entries)
16kB Page (16 byte entries)

Small Allocation Module Pros & Cons

+ Tiny implementation
+ Very low wastage
+ Makes use of flexible memory
+ Fast
- Difficult to detect memory stomps

Large Allocation Module



- Reserves huge virtual address space (160GB)
- Each table divided into equal sized slots
- Maps and unmaps 64kB pages on demand
- Guarantees contiguous memory

Texture Streaming

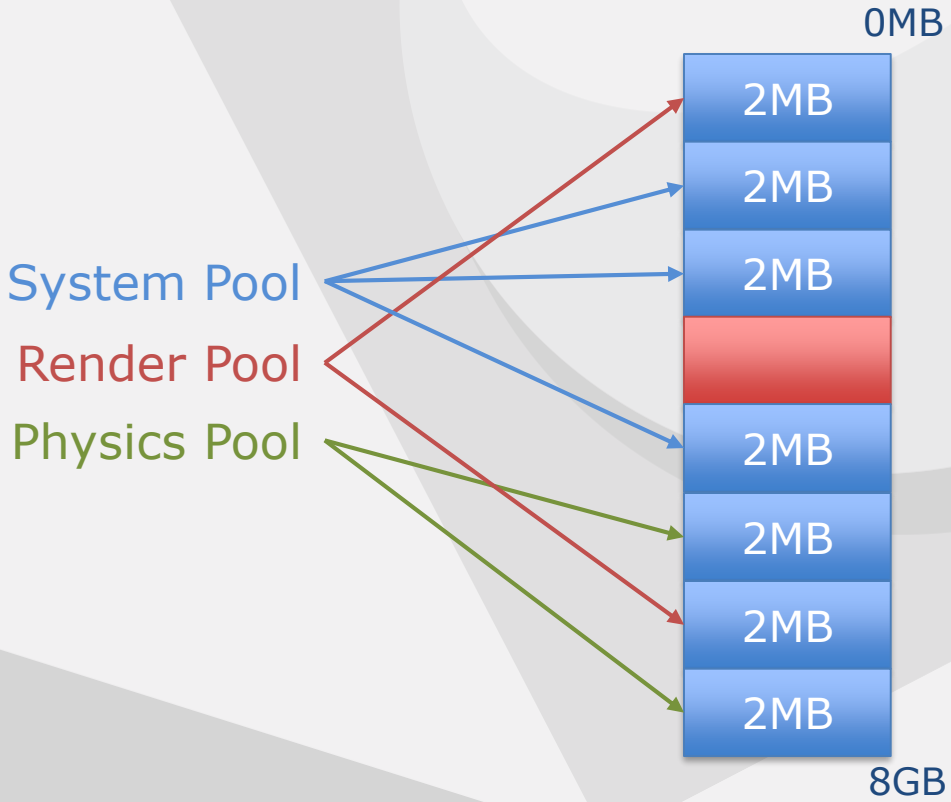
- Reserve large allocation slot
  - Rounded up to nearest pow 2
- Load max of smallest mip and 64kB
- Map and unmap pages on demand
- No need to copy or defrag

Large Allocation Module Pros & Cons

+ No headers
+ Simple implementation (~200 lines of code)
+ No fragmentation
- Size rounded up to page size
- Mapping and unmapping kernel calls relatively slow

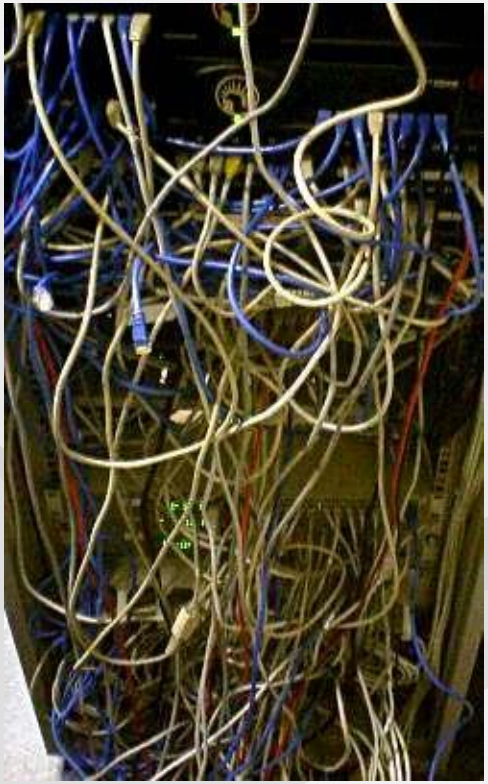
Medium Allocation Modules

- Medium
- Headerless



Medium Allocation Module

- All other sizes go here
- Non-contiguous virtual pages
- Grows and shrinks
- Traditional doubly linked list with headers
- Unsuitable for Garlic memory
  - Headers stored with data
- Pow2 free lists



Headerless Allocation Module

- Used for GPU allocations
  - Small to medium allocations
- Hash table lookup

Allocator Types

- GeneralAllocator
- VramAllocator
- MappedAllocator
- GpuScratchAllocator
- FrameAllocator
- …

```
class Allocator
{
public:
        virtual void* Allocate(
                size_t size,
                size_t align) = 0;
        virtual void Deallocate(
                void* pMemory) = 0;
        virtual size_t GetSize(void* pMem);
        const char* GetName(void) const;
};
…
MM_NEW(pAllocator) MyType();
```

GPU Scratch Allocator

- Used by renderer for per frame allocations
- Double buffered
- No need to deallocate
- Protected with atomics

GPU Scratch Allocator – Pros & Cons

+ No headers or bookkeeping
+ No fragmentation
+ Fast!
- Fixed size
- Worst-case alignment wastes space

Frame Allocator

- Frames pushed and popped
- No need to free memory
- Unique to each thread
- Useful for temp work buffers

```
#include <ls_common/memory/ScratchMem.h>

struct Elem
{
    …
};

void ProcessElements(size_t numElements)
{
    ls::ScratchMem frame;

    Elem* pElements =
        (Elem*)MM_ALLOC_P(
            &frame,
            sizeof(Elem) * numElements
        );
}
```

Frame Allocator Pros & Cons

+ No headers or bookkeeping
+ No fragmentation
+ No synchronisation
+ Fast!
- Careful passing pointers around!

Thread Safety

- Mutexes at lowest level
- Allocator instances not protected
- Frame allocator has no locks
- Nice and simple ☺

Performance

- Performance not the focus
  - Still important
- Mapping/unmapping slow
- No noticeable difference
  - Don’t allocate much during game
  - File loading is bottleneck

Clear Values

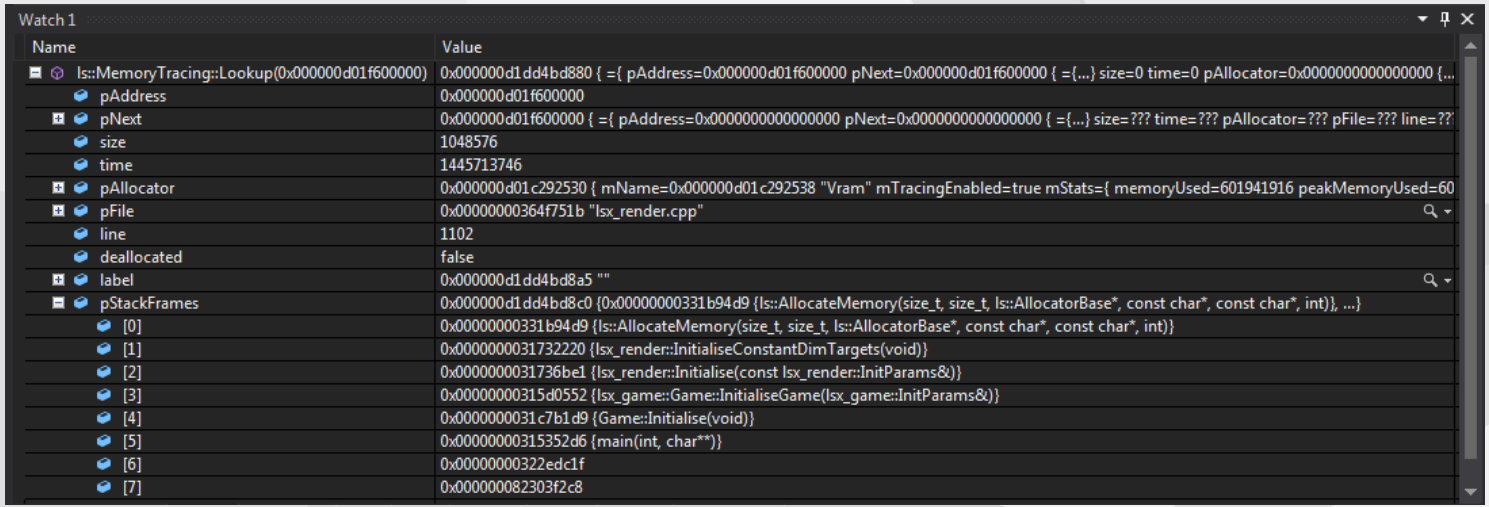
- memset to byte value
  - Keep it memorable
- `0xFA` – **F**lexible memory **a**llocated
- `0xFF` – **F**lexible memory **f**ree
- `0xDA` – **D**irect memory **a**llocated
- `0xDF` – **D**irection memory **f**ree
- `0xA1` – Memory **al**located
- `0xDE` – Memory **de**allocated

Statistics

- Track everything possible
- Live graphs available
- Recorded by automated tests

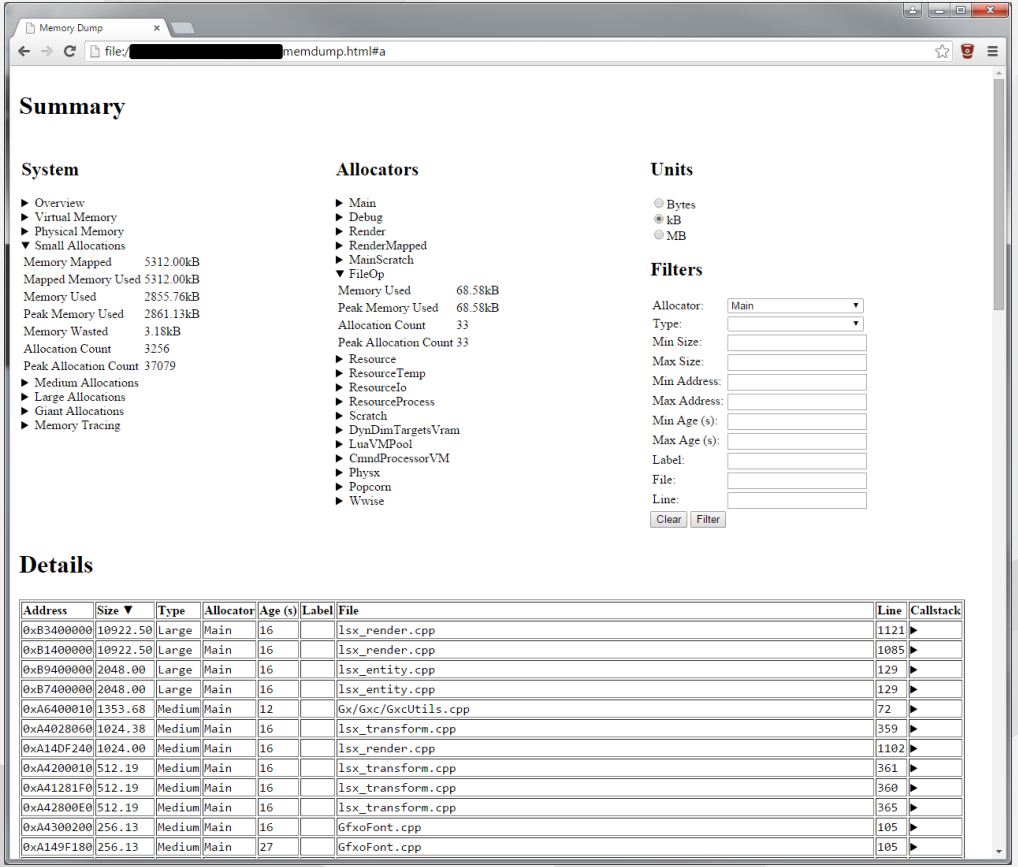
Tracing

- `ls::MemoryTracing::Lookup(0x000000D01F600000)`
  - Watch window function call
  - Works for addresses in the middle of a block



Tracing

- Accessed using iterators
  - Write to TTY
  - Dump to HTML file
- Dump on:
  - Demand
  - Out of memory
  - Leak detection



London Studio

Memory Header Guards

- Free bytes in medium allocation headers
- Detect memory stomps
  - Often too late
- Easy to spot 1ee7 speak in memory view ☺
  - 0xA110C8
  - 0xDE1E7E

Memory Block Sentinels

- Bypass normal allocators
- Each allocation in own page
- Unmapped pages before and after
- Crash on over/under write

Memory Protection Flags on PlayStation®4

- Allocs protected using memory protection flags
  - Specified by each allocator instance
- Crash when CPU or GPU accesses wrong memory
- Prevents
  - Stomps from CPU/GPU
  - Unintentional read/write using slow bus
  - Wasting page tables

PlayStation®4 GPU Debugging

- Keep mapping table at fixed address
  - Stores bus and protection flags
  - Two-stage lookup table to save space
- Renderer validates addresses before submit
- Modify shaders on load
  - Check address before read/write

Summary

- Modern consoles have rich virtual memory support
- Virtual memory provides many options
- Design your memory system around your allocation patterns
  - Analysis is important
  - Small allocations are a good place to start
- Modularised allocators make customisation easy!
- Debug features are vital!

Thanks

- Mark Cerny
- Mike Schaadt
- Joe Milner-Moore
- Simon Hall
- Mark Lintott

Questions?

aaron_macdougall@scee.net