Scalloc

“Fast, Multicore-Scalable, Low-Fragmentation Memory Allocation through Large Virtual Memory and Global Data Structures”

(Presented) By Thomas

The three horsemen of allocator apocalypse

1

Fast Allocations

2

Fast Access

3

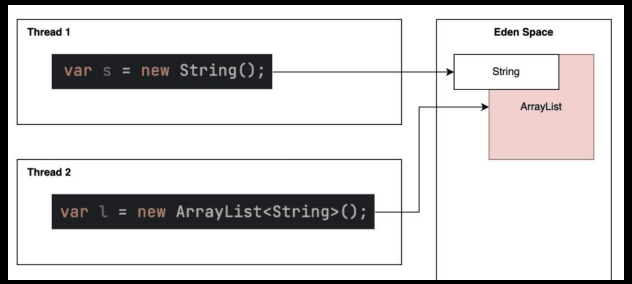
Low overhead and reuse

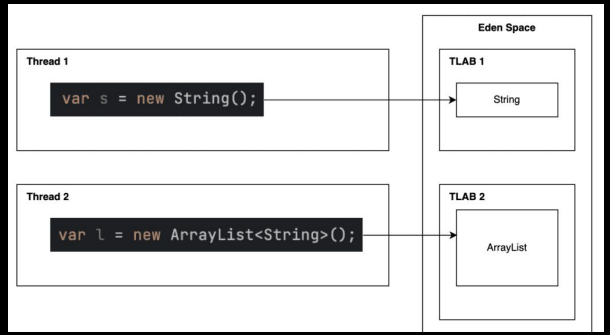
The three horsemen of allocator apocalypse

1

Fast Allocations

The common solution for fast allocations for multicore allocators is to support thread-local allocators, rather than relying on locks.



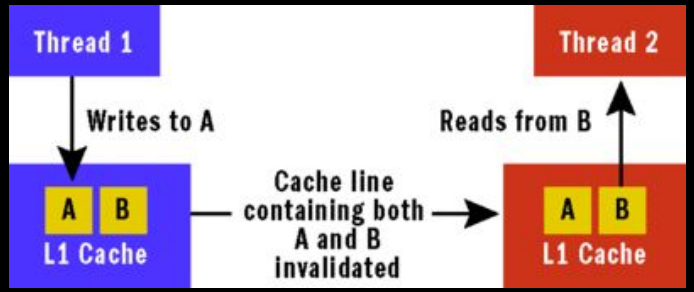


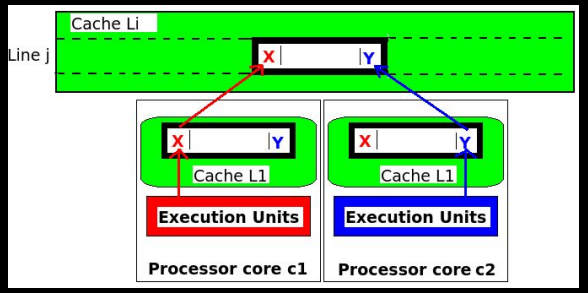
The three horsemen of allocator apocalypse

2

Fast Access

You need *spatial locality* (for optimal cache access), but you don't want false sharing (where modification of content in cache lines can cause unnecessary invalidations)



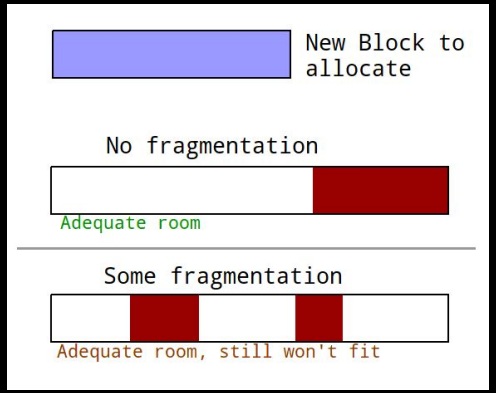


The three horsemen of allocator apocalypse

3

Low overhead and reuse

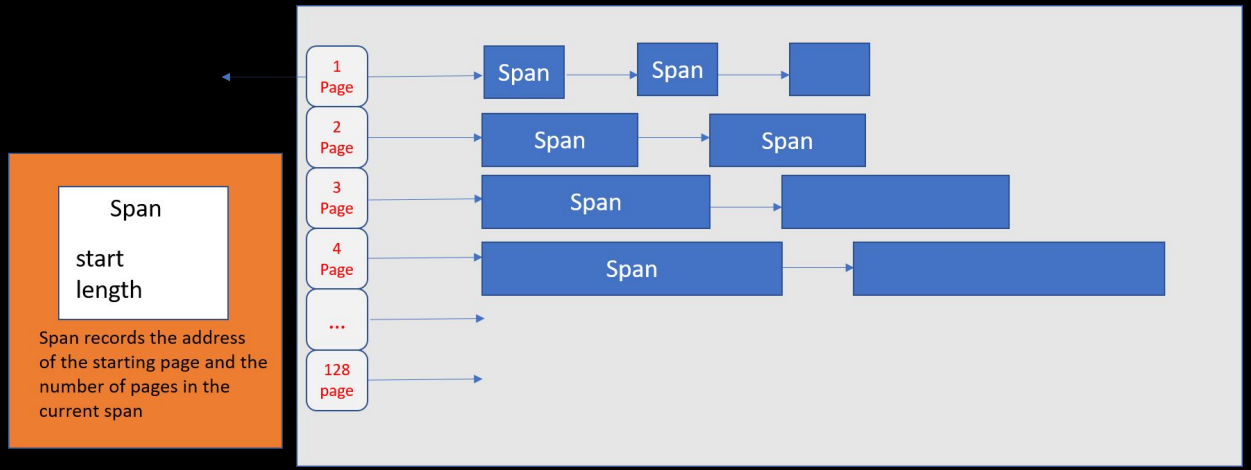
You need to handle defragmentation, quickly reallocating freed memory (even across thread-local allocators), while keeping global structures overhead to a minimum.



Memory as Spans

Thread-Local Allocation Buffers (TLABs) are generally designed through what is called a *span*.

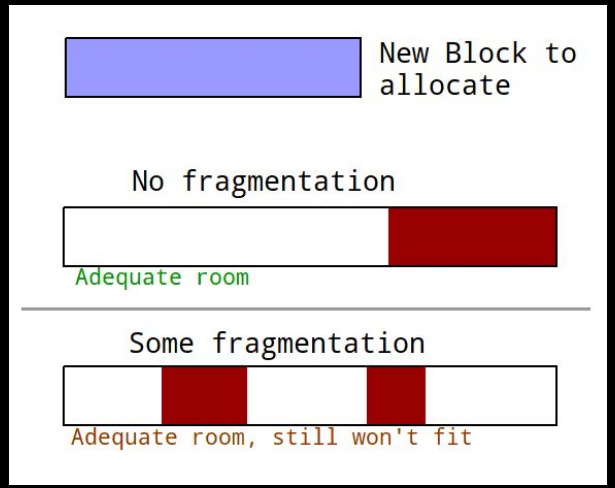
A span is a region of memory that is taken from the global allocator, reserved for the thread's use. A thread can have multiple spans.



Memory as Spans

A larger span is faster for allocations (as you have more pre-reserved space), but requires excess memory from the global allocator backend.

Same-sized spans allow for better reuse, but result in internal fragmentation.



The Scalloc method

1

Virtual spans, which are same-sized spans in virtual memory

2

A scalable global backend based on scalable concurrent data structures

3

Constant-time (modulo synchronization) frontend

Virtual Spans

1

Virtual Spans

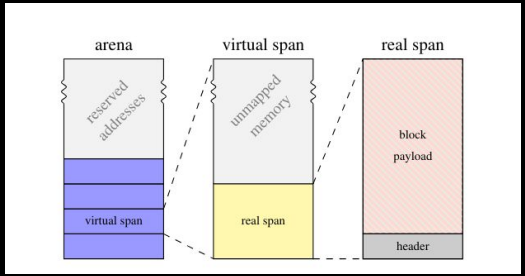
Modern hardware allows for *on-demand* paging, where the system only loads memory when it's needed.

Virtual Memory is memory that's not necessarily backed by physical memory (only backed on-demand), and could move onto disk.

Virtual Spans

1

Virtual Spans



Virtual spans are backed by real memory spans (fixed 2 MB).

The OS already handles physical memory fragmentation, and unused space in virtual spans only waste virtual address space.

Thanks to on-demand paging, it only has to manage fragmentation when resizing empty real spans.

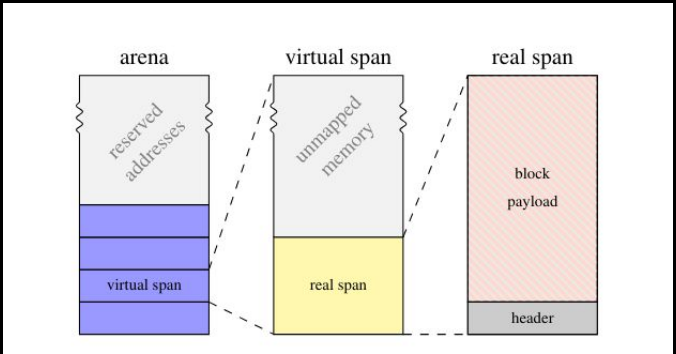
Virtual Spans

1

Virtual Spans

Virtual Spans exist within the *Arena*, which is a **32 TB virtual memory chunk** managed by the allocator.

As it's one chunk, you don't need lots of syscalls as it's just one big mmap.



Global Backends

2

Global Backend

The global backend is essentially a group of stacks with virtual spans.

Actual allocations are handled beforehand, with a dynamic virtual stack.

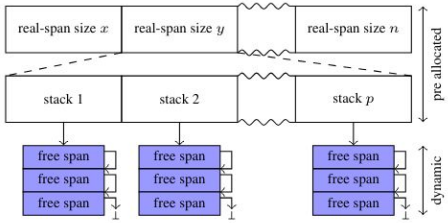


Figure 2: Span-pool layout

Fast Frontend

3

Fast Fontend

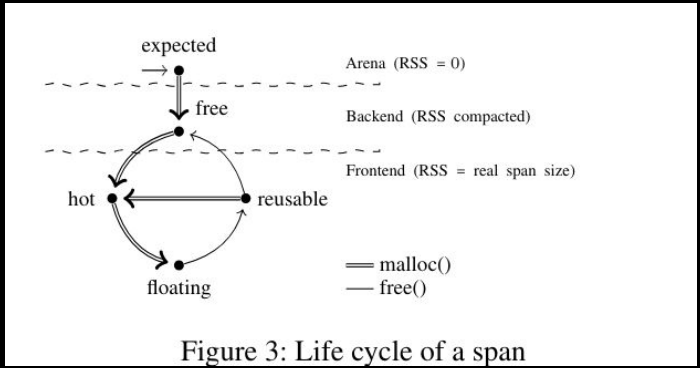


Figure 3: Life cycle of a span

Memory in the arena uses 0 physical memory, free is allocated in a real span backing a virtual span, hot for when new objects are allocated with free space, floating when there’s not much free space, and reusable when there’s new free space.

:-)