SmartOS

Adding per-thread caching to libumem

Robert Mustacchi

What is libumem?

SmartOS

- A drop in replacement implementation of malloc(3C) and free (3C) Adding per-thread caching to libumem
- Which are the functions that are called to allocate memory in C programs
- The functions people call that causes VSZ and RSS go up in ps
- The tools to figure out where that’s coming from

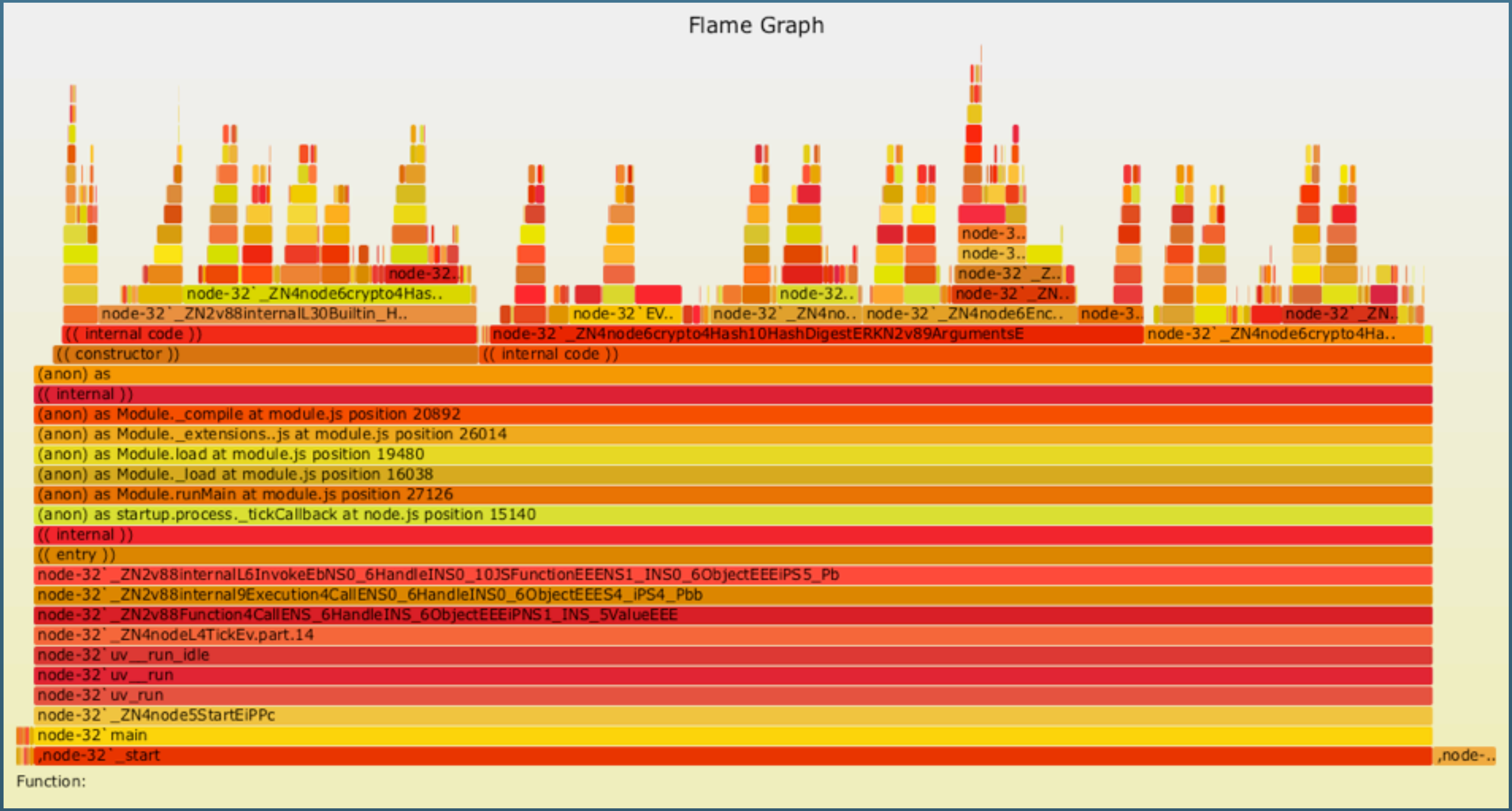
Why does it matter?

- Every program ends up calling malloc at some point in time
- In C++ calling new() ends up using malloc
- Garbage collected languages still end up calling malloc
- Examples:

  ```
  dtrace -n 'pid$target::malloc:entry{ @ = count(); }' -c kstat
  671598 (Perl)
  dtrace -n 'pid$target::malloc:entry{ @ = count(); }' -c hg
  3801 (Python)
  dtrace -n 'pid$target::malloc:entry{ @ = count(); }' -c 'vmadm list'
  2754 (Node.js)
  ```

Why does it matter?

SmartOS



How malloc works - circa 1988

SmartOS

- Only one thread can be in malloc at a time
- We keep track of free space in a binary tree
- Search through the binary tree to find something that fits your request
- If you can’t find something that fits, increase the size of the heap

Enter the slab allocator - circa 1994

- Created by Jeff Bonwick for Solaris 2.4
- Object caching
  - Lots of objects are commonly allocated and freed
  - Allocate a bunch and have them be primed
  - Rather than deallocate it every time, but it back on stand by
- Used all over the kernel
  - Inodes, ARC data, message blocks
- Lock is on the cache, only one thread can be in the cache at a time
- Adds support for basic debugging, use after free, etc.

Enter Magazines - Circa 1995

- Introduced by Jeff Bonwick in Solaris 2.5.
- Having to lock the entire object cache doesn’t scale with many CPUs
- Each CPU gets a magazine per cache (think automatic weapon)
- When the magazine runs out, grab the global cache lock and reload
- Magazine size is increased dynamically based on contention
- Continue to pad things onto hardware caches and use cache coloring

Enter libumem - 2001

- Introduced by Jonathan Adams and Jeff Bonwick in Solaris 9u3
- Take the kernel allocator and bring it to userland
- Brings all of the debugging to userland
  - Once you use ::findleaks, ::whatis, it’s hard to go back
- Use caches for malloc
- Use it in two ways:
  - Add -lumem to your Makefile
  - Use LD_PRELOAD=libumem.so

Where does libumem start to break down?

SmartOS

- Grabbing an uncontended lock in the kernel is cheap
  - It’s 4 instructions!
  - Optimized for non-contention
- Grabbing an unconteded lock in userland is more expensive
  - Userland locks need to deal with many more edge conditions
  - The kernel exploits unholy knowledge to accelerate in-kernel mutexes
- Every allocation requires you to grab a lock
- Some applications are bad actors, they do lots of mallocs and frees of the same sized data
  - We’ve seen on the order of 10k-100k per second
- Customers complain because other mallocs are faster

Focusing on the right problem

- Our problem is that we need to synchronize
  - Just rewriting everything to be lockfree using atomic operations doesn’t magically solve our problems
- Eliminate the synchronization
- Add a per-thread cache

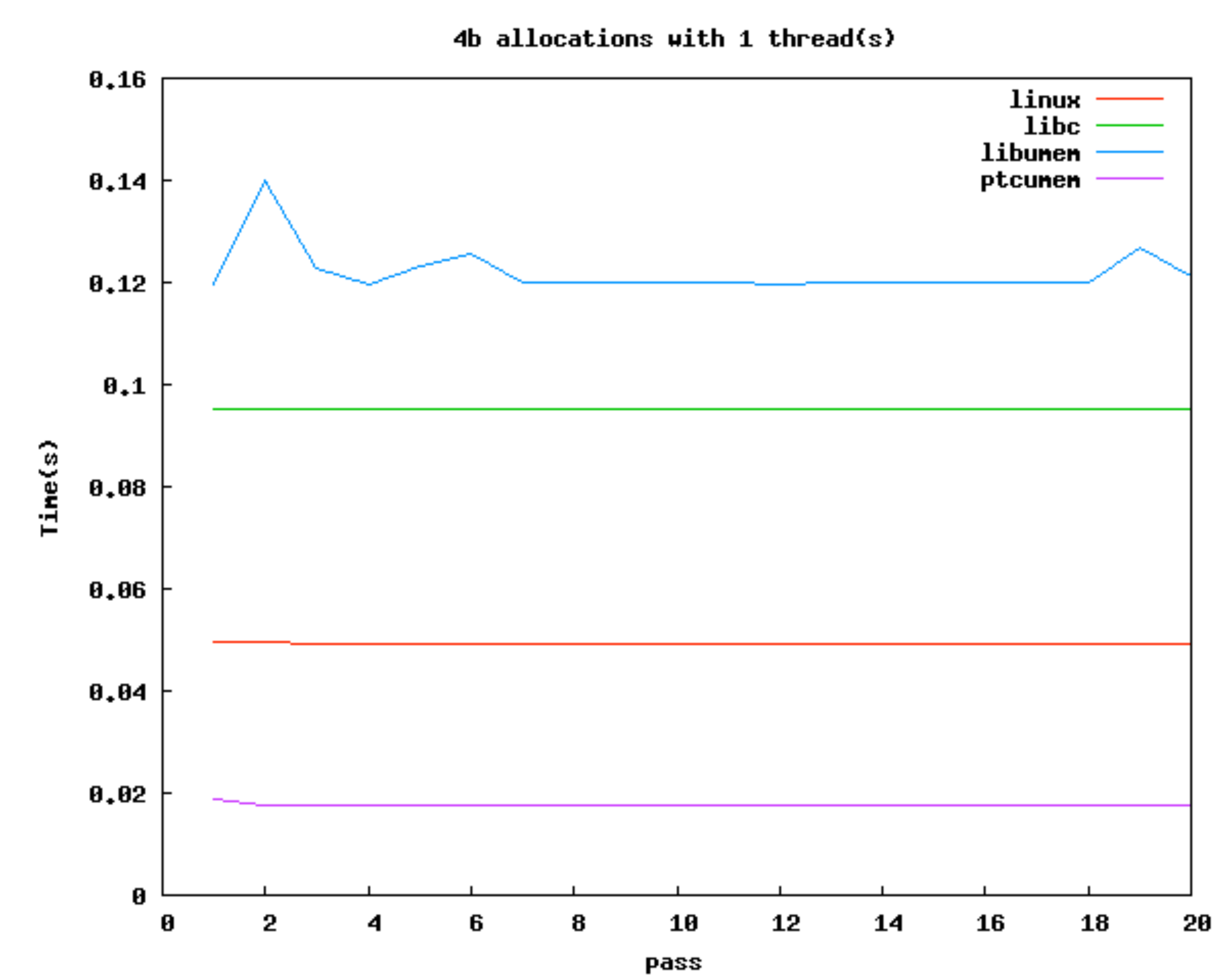
Careful planning

- When you allocate memory a tag is prepended
  - It is either 8 or 16 bytes (depending on size and architecture)
  - We don’t want to increase that
- We don’t want to increase fragmentation
- We want to build our cache based on the umem cache size
- The size of the caches aren’t known at compile time

Dynamic Generation

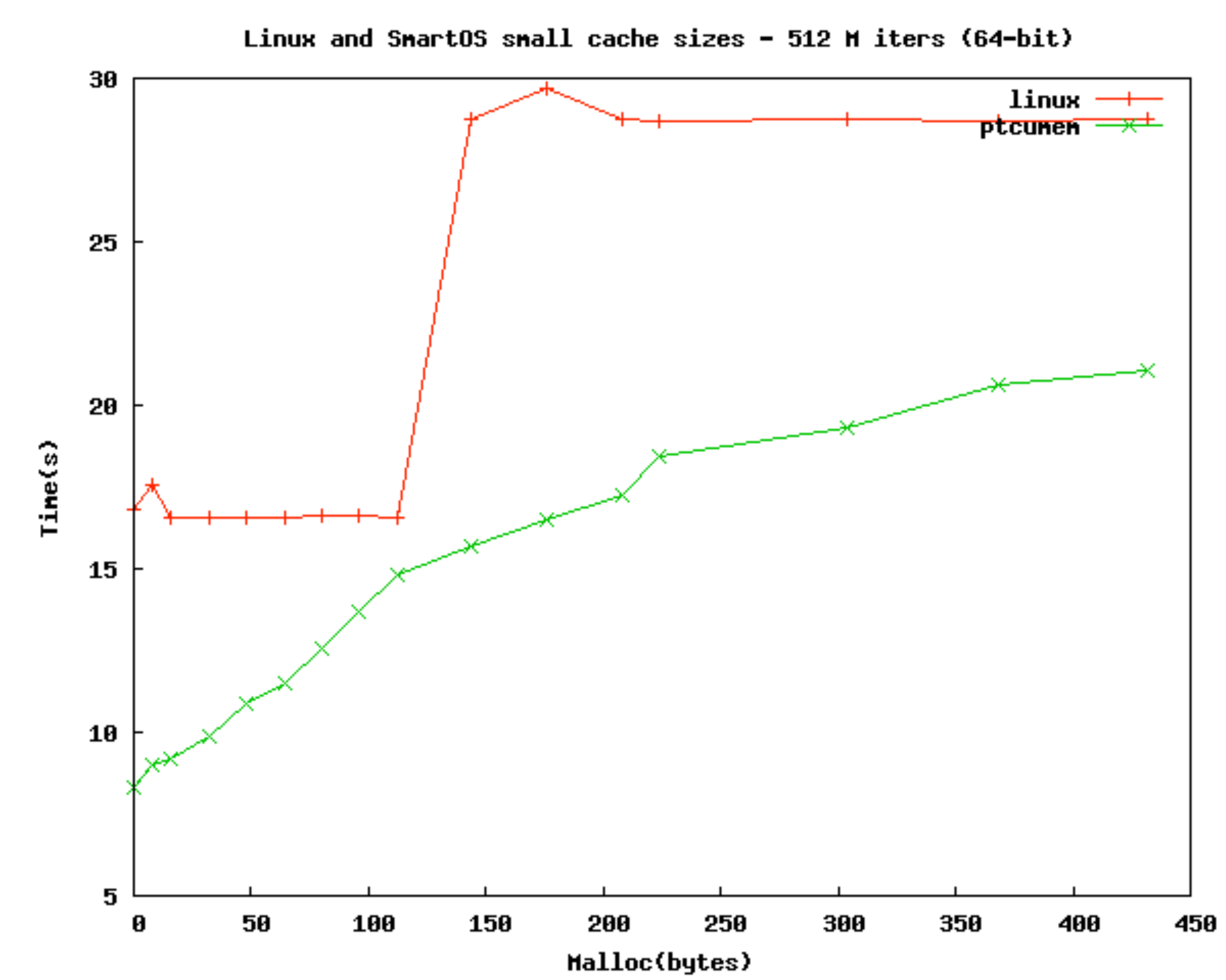
- libc gives each thread 16 slots in the thread's uberdata
- During umem_init we determine the final set of cache sizes
- Each uberdata slot is the head of a linked list of recently freed buffers
- We use that information and dynamically generate the machine code for malloc and free
  - Need to avoid loads for performance
- Each slot maps directly to one of the first sixteen caches
- Threads free their cache when they exit

Per-thread caching in action



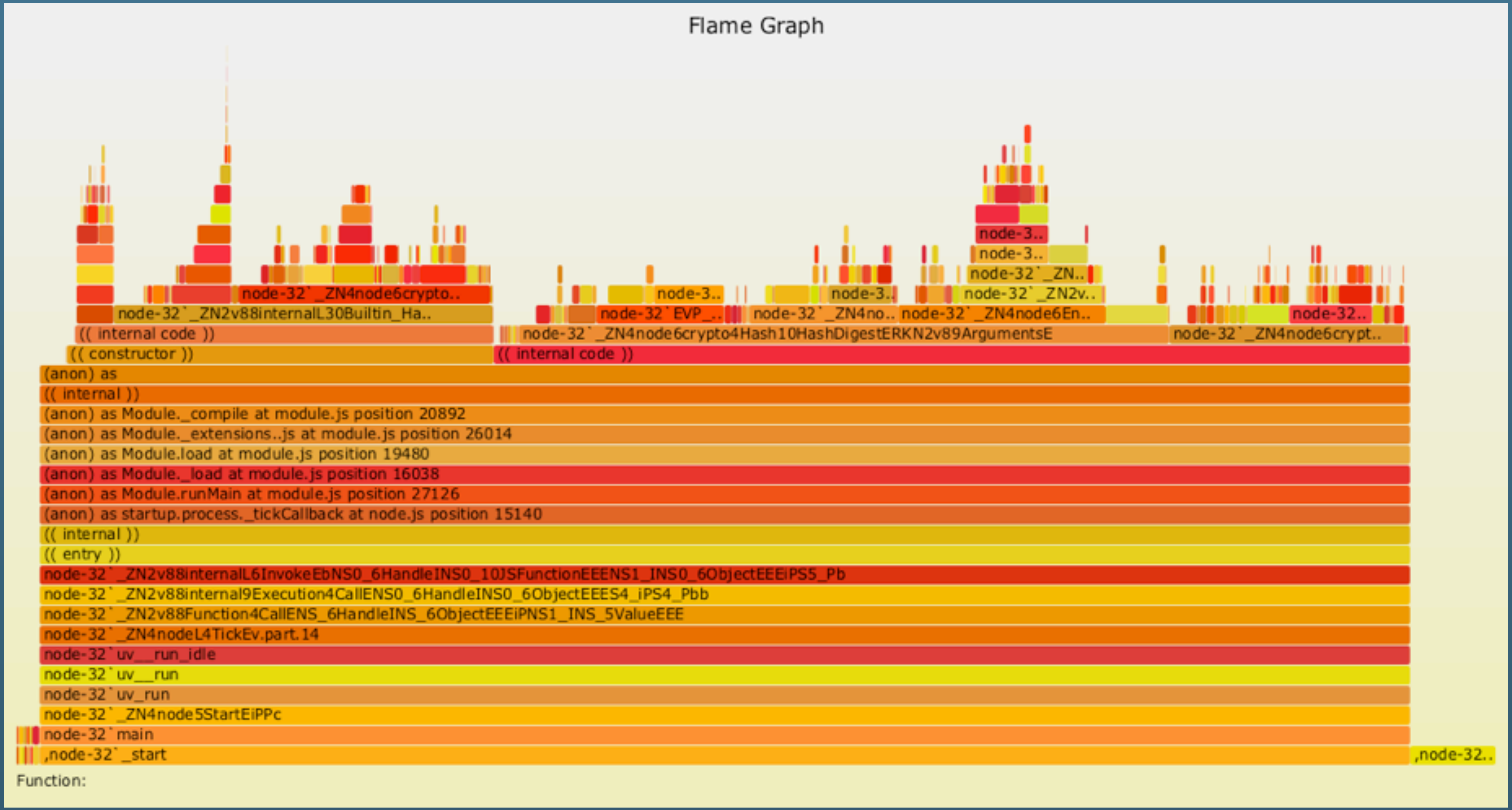
Per-thread caching in action

SmartOS

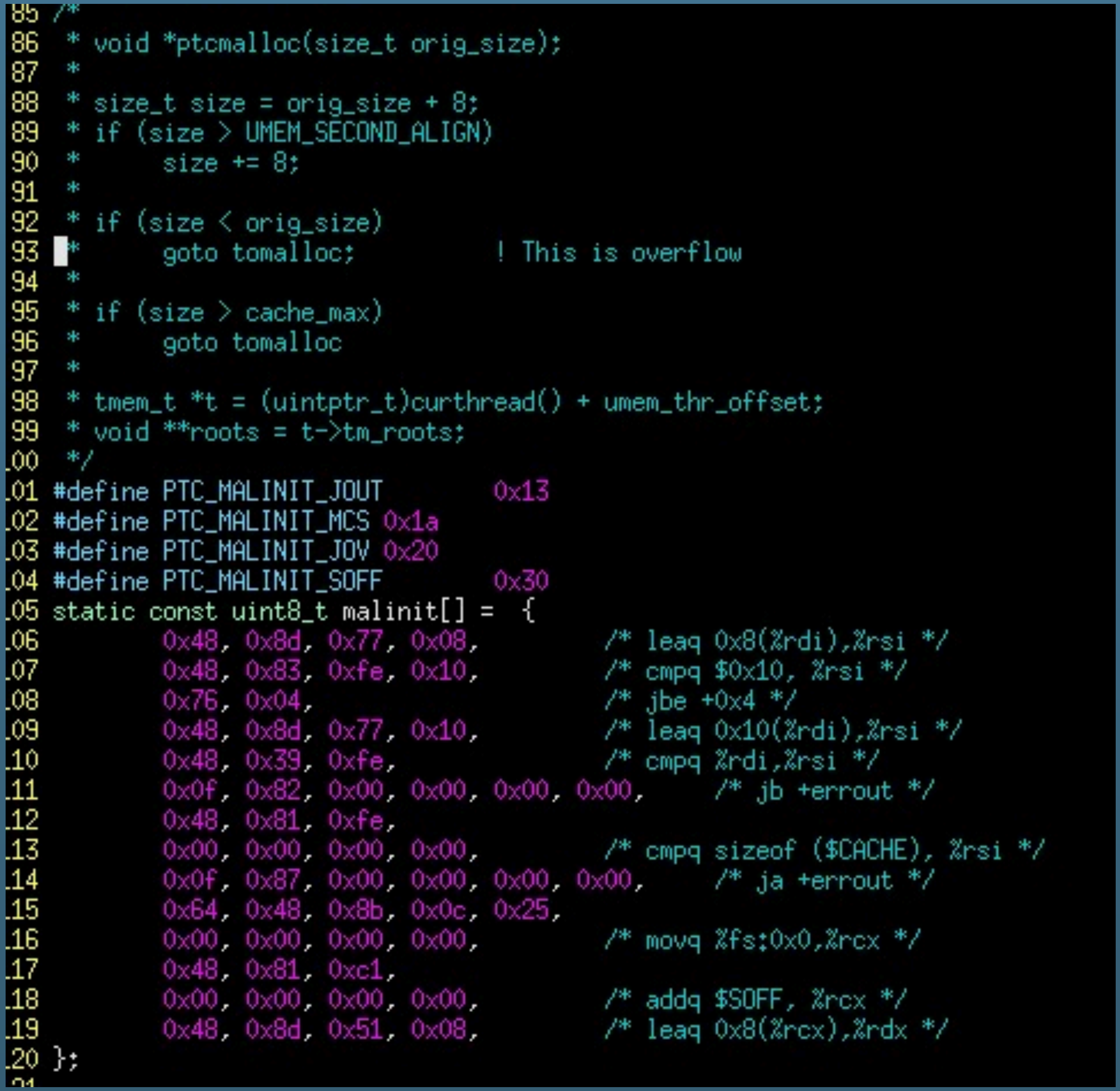


Our original flamegraph

SmartOS



Peaking under the hood



```
/*
 * void *ptcmalloc(size_t orig_size);
 *
 * size_t size = orig_size + 8;
 * if (size > UMEM_SECOND_ALIGN)
 *      size += 8;
 *
 * if (size < orig_size)
 *      goto tomalloc;          ! This is overflow
 *
 * if (size > cache_max)
 *      goto tomalloc
 *
 * tmem_t *t = (uintptr_t)curthread() + umem_thr_offset;
 * void **roots = t->tm_roots;
 */
#define PTC_MALINIT_JOUT        0x13
#define PTC_MALINIT_MCS 0x1a
#define PTC_MALINIT_JOV 0x20
#define PTC_MALINIT_SOFF        0x30
static const uint8_t malinit[] =  {
        0x48, 0x8d, 0x77, 0x08,         /* leaq 0x8(%rdi),%rsi */
        0x48, 0x83, 0xfe, 0x10,         /* cmpq $0x10, %rsi */
        0x76, 0x04,                     /* jbe +0x4 */
        0x48, 0x8d, 0x77, 0x10,         /* leaq 0x10(%rdi),%rsi */
        0x48, 0x39, 0xfe,               /* cmpq %rdi,%rsi */
        0x0f, 0x82, 0x00, 0x00, 0x00, 0x00,     /* jb +errout */
        0x48, 0x81, 0xfe,
        0x00, 0x00, 0x00, 0x00,         /* cmpq sizeof ($CACHE), %rsi */
        0x0f, 0x87, 0x00, 0x00, 0x00, 0x00,     /* ja +errout */
        0x64, 0x48, 0x8b, 0x0c, 0x25,
        0x00, 0x00, 0x00, 0x00,         /* movq %fs:0x0,%rcx */
        0x48, 0x81, 0xc1,
        0x00, 0x00, 0x00, 0x00,         /* addq $SOFF, %rcx */
        0x48, 0x8d, 0x51, 0x08,         /* leaq 0x8(%rcx),%rdx */
};
```

Tuning and introspection

- Default cache size is 1 MB
- Tuned via UMEM_OPTIONS=perthread_cache=size
- Bryan Cantrill added support to ::umastat to see what’s being used

```
memory    %   %   %   %   %   %   %   %   %   %   %   %   %   %   %   %   %   %
tid  cached cap   8  16  24  32  40  48  56  64  80  96 112 128 160 192 224 256
--- ------- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
  1    210K  20   0   0   2   1   0   0   0   0   0   0   0   0  93   0   1   0
  2       0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  3       0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

References

- Original slab allocator paper: http://static.usenix.org/publications/library/proceedings/bos94/full_papers/bonwick.a
- Magazines and vmem paper: http://static.usenix.org/publications/library/proceedings/usenix01/full_papers/bonwick/bonwick_html/
- Per-thread caching details: http://dtrace.org/blogs/rm/2012/07/16/per-thread-caching-in-libumem/
- libumem dcmds overview: https://blogs.oracle.com/jwadams/entry/debugging_with_libumem_and_mdb
- libumem debugging: http://developers.sun.com/solaris/articles/libumem_library.html