Configuring malloc for faster and smaller applications

Colin Honess

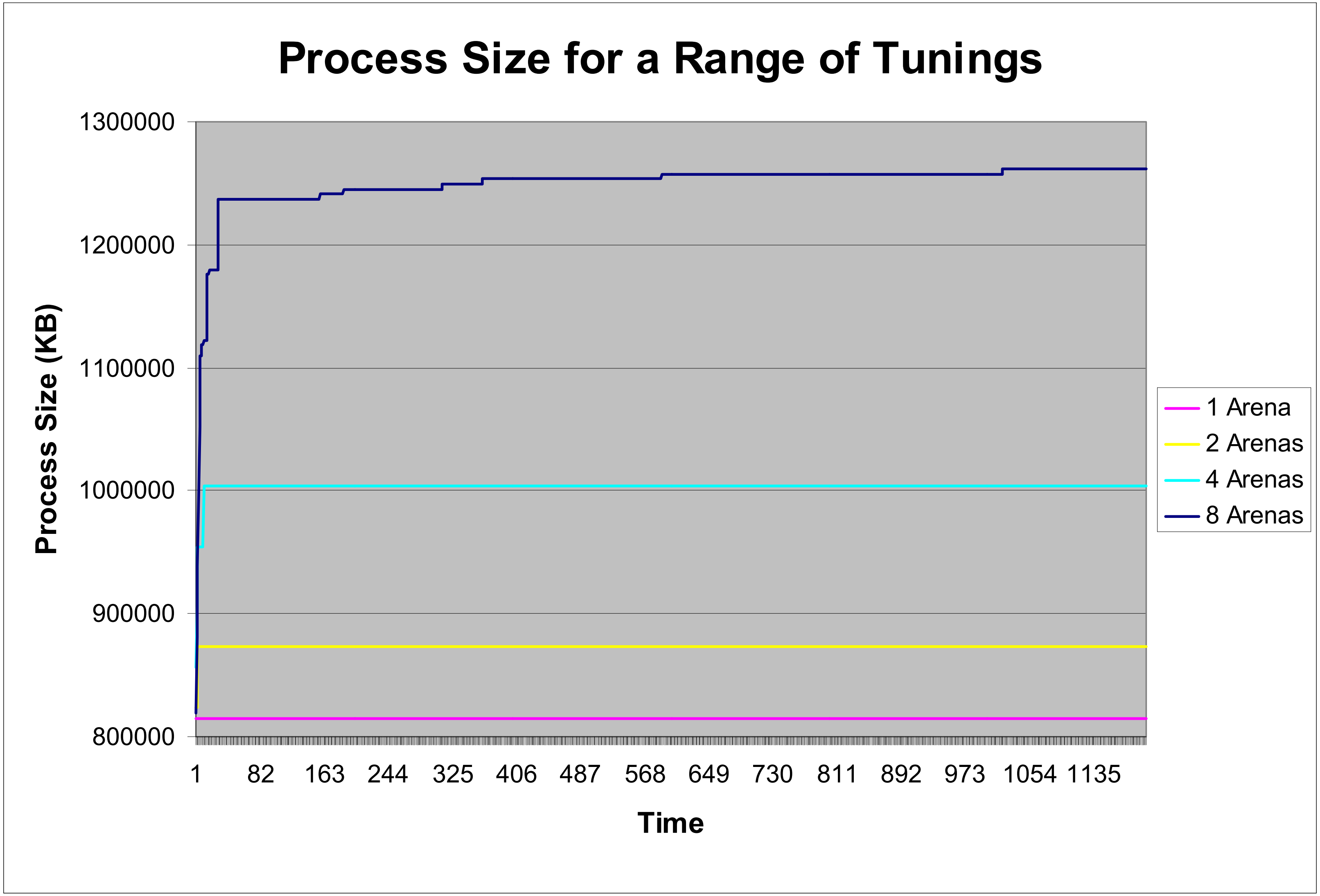
Strategic Deals Team, HP

HP WORLD 2003
Solutions and Technology Conference & Expo

What to expect

- Introduction to malloc
- How malloc is implemented on HP-UX
  - Basic malloc algorithms
  - Small block allocator
  - Multi-arena malloc
  - Thread-local cache
- How to monitor and tune malloc behavior.

Why should I care?



Why should I care?

Tuning	Result (operations/second)	% Improvement
1 Arena	165,652	-87%
8 Arenas (default)	1,282,808	0%
32 Arenas	1,469,479	14.5%
Thread-local cache	1,762,570	37%

When should I care?

When any of the following is important:

- Process size
- Stability in process size
- Performance

Particularly for multi-threaded applications

Using malloc – C examples

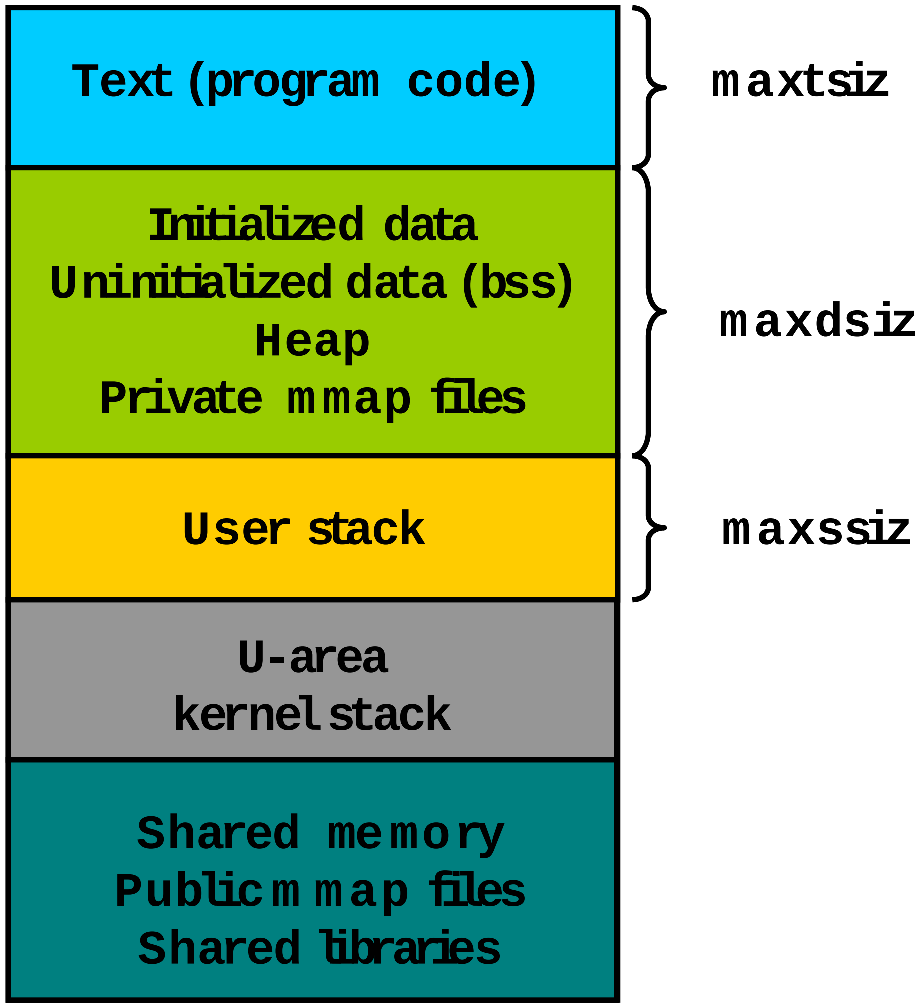
```
c = (char *)malloc(42);

f = (float *)malloc(sizeof(float));

fourints = (int *)calloc(sizeof(int),4);

pagealigned = valloc(8192);

f = (float *)realloc(f, 2*sizeof(float));

free(c);

newstr = strdup("hello world");
```

How much memory can be allocated through malloc?

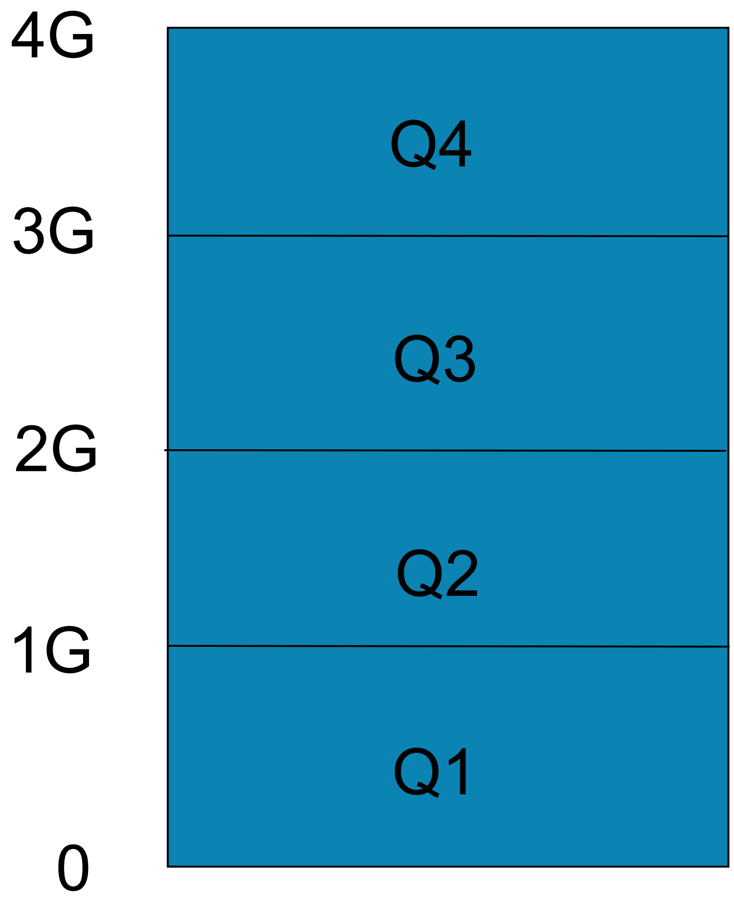
It depends…

- 32 bit or 64 bit
- 32 bit memory model
- Kernel parameters:
  - maxdsiz
  - maxdsiz_64bit
  - maxssiz
  - maxssiz_64bit
- Available swap space

Logical segments in a process

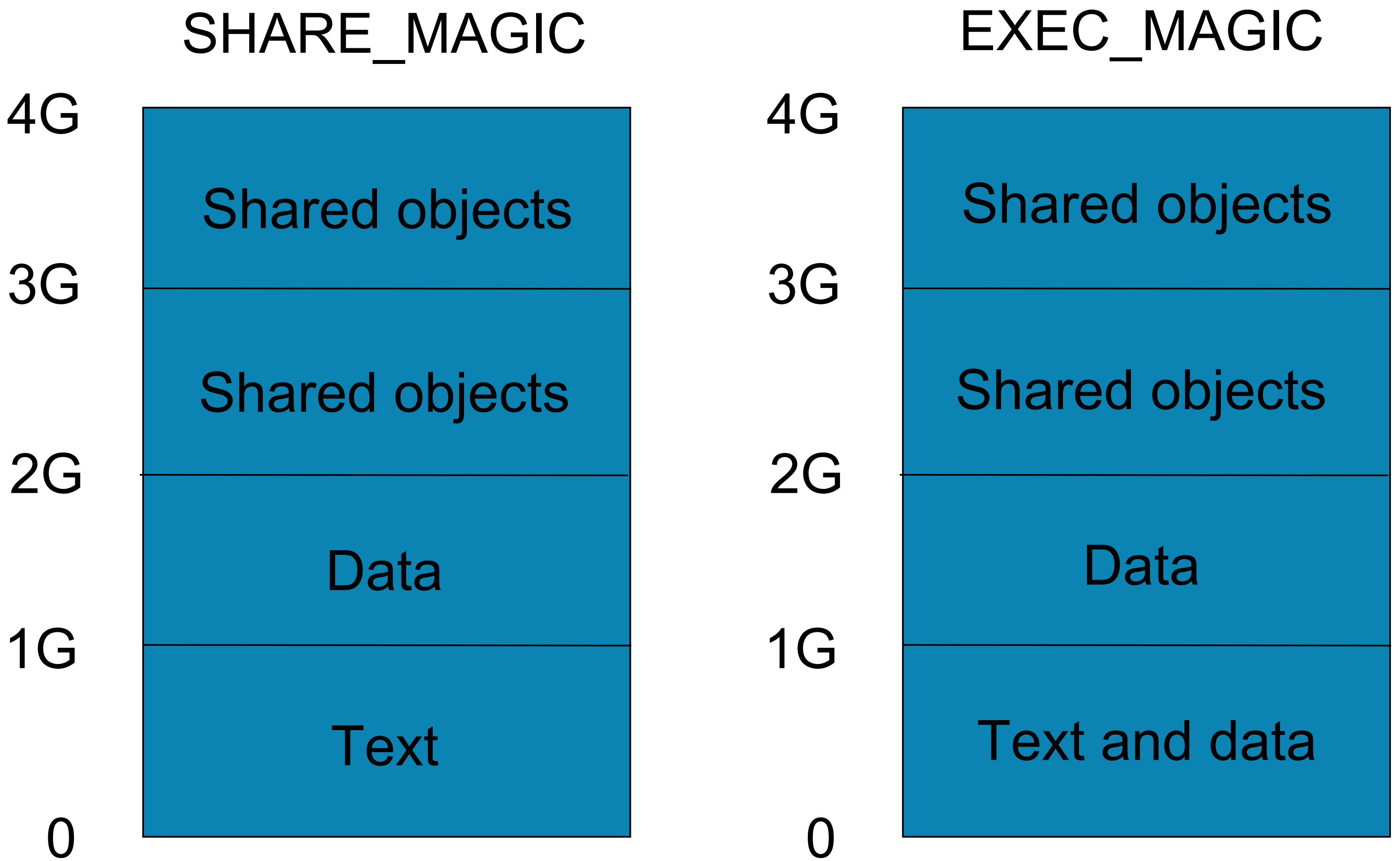


32 bit memory models

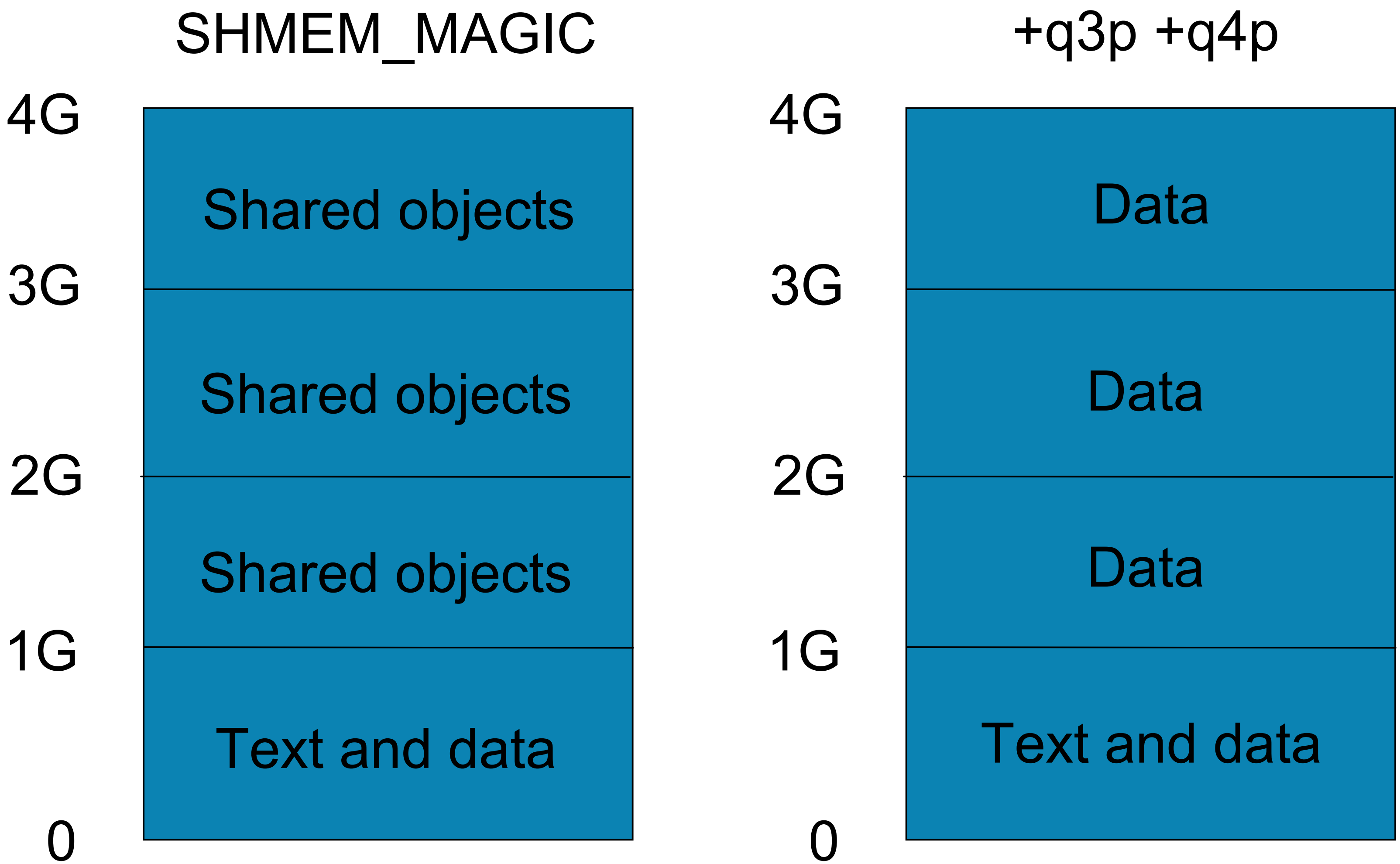


- Four 1GB quadrants into which we must place:
  - Text
  - Private data
  - Shared libraries
  - Shared memory
  - Memory mapped files
  - etc.
- Each quadrant is either shared or private
- Different memory models offer different mappings

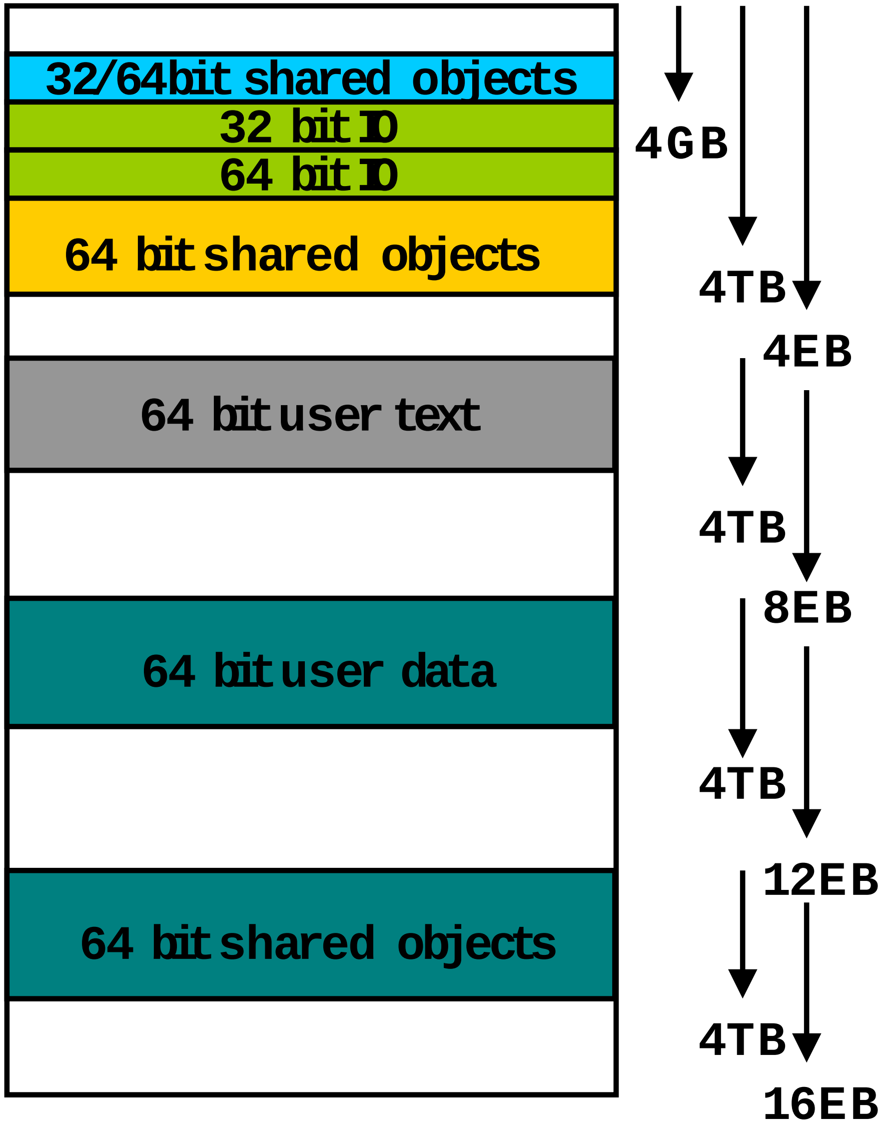
32 bit memory models



32 bit memory models



64 bit process layout



How much memory can be allocated through malloc?

32 bit process:

– SHARE_MAGIC	<1GB
– EXEC_MAGIC	2GB for text and data
– SHMEM_MAGIC	1GB for text and data
– Private q3 & q4	4GB for text and data

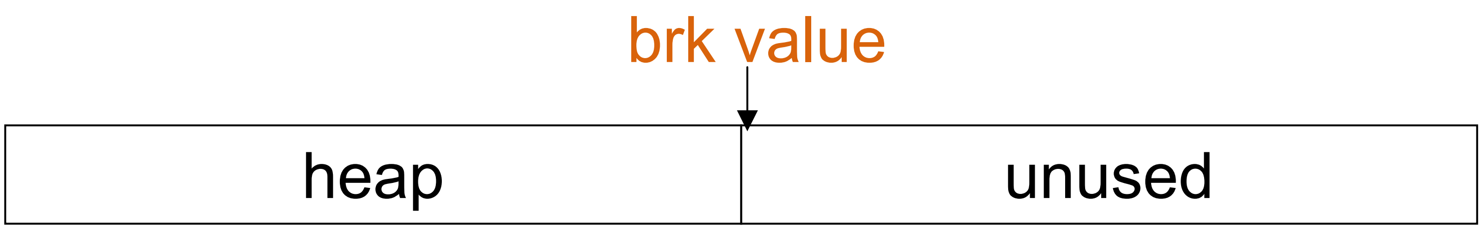
64 bit process: <4TB

Tune:

- maxdiz, maxdsiz_64bit
- maxssiz, maxssiz_64bit
- Available swap space

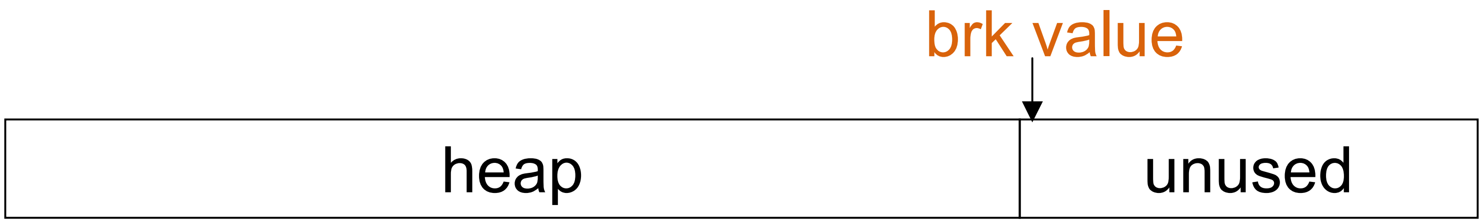
Controlling the size of the heap: brk()

brk value is the address of the first location beyond the end of the data segment:

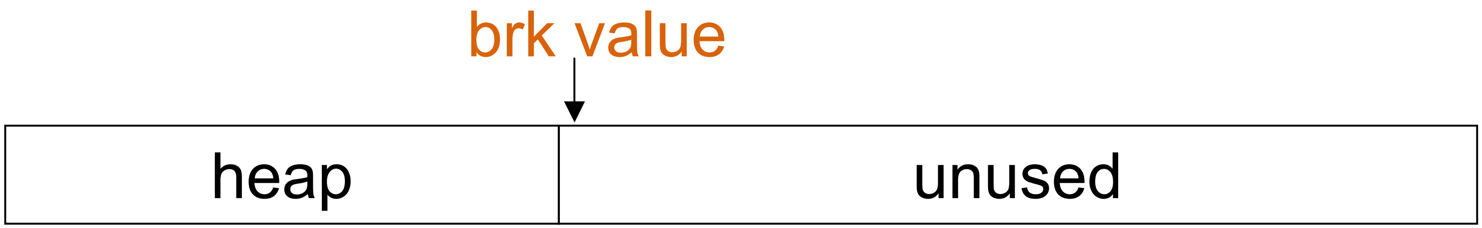


Controlling the size of the heap: brk()

brk value can be increased through brk() or sbrk() to allocate more space to the heap:

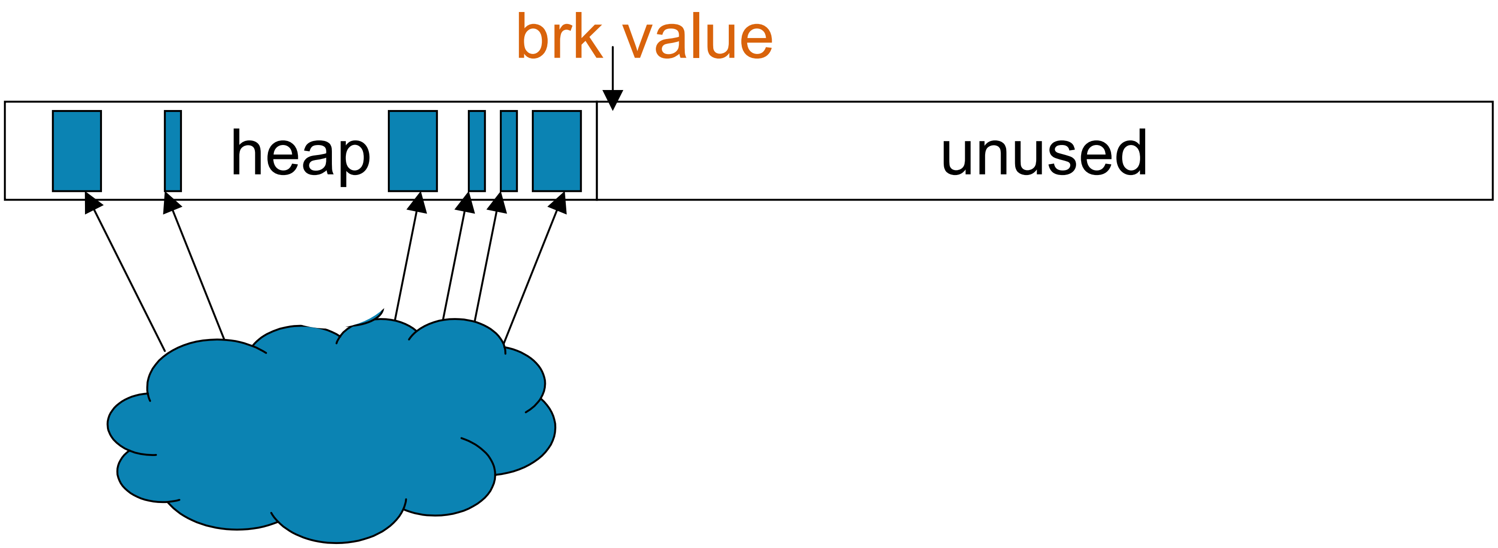


brk value can be decreased to truncate the heap:



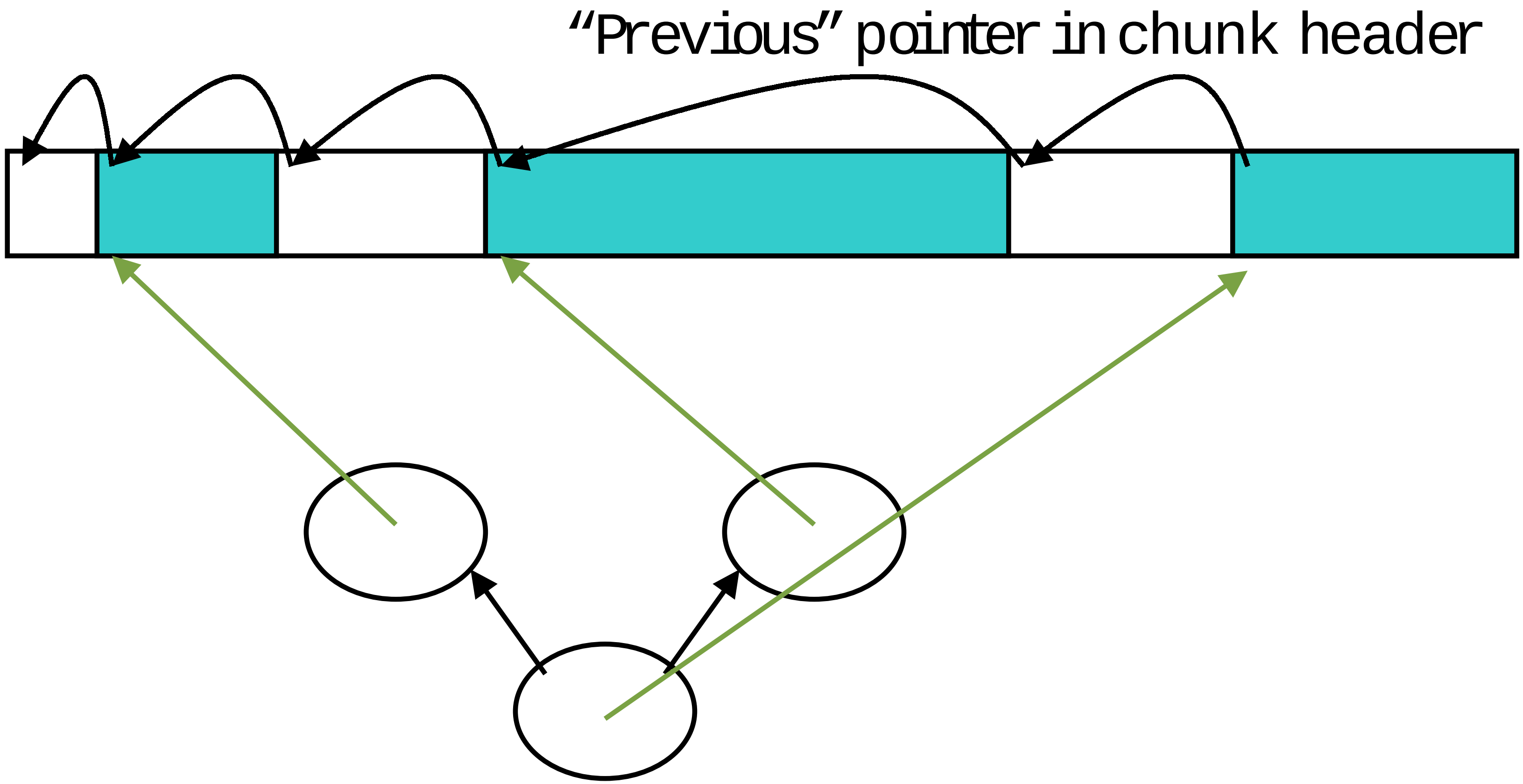
Malloc keeps track of free space within the heap

…but we cannot release arbitrary memory within the heap



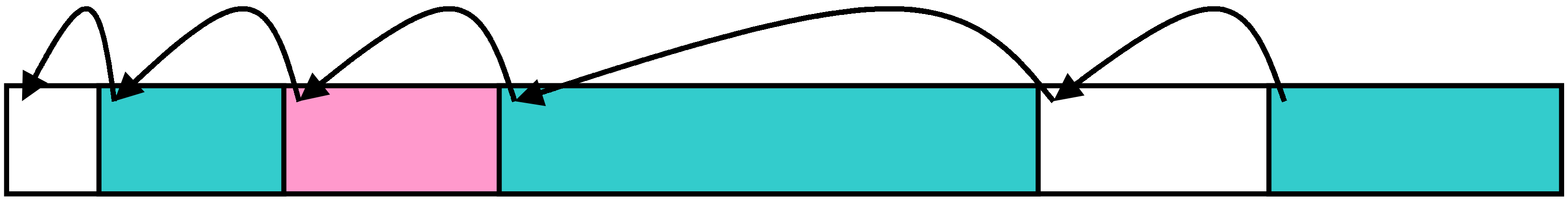
So malloc keeps track of memory the application has freed, and will reuse it to satisfy future requests

The free tree



Free tree ordered by size of free chunk

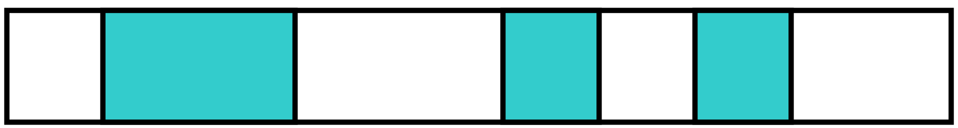
Free space is coalesced during free



1. Search for adjacent free memory
2. Remove adjacent free memory from free tree
3. Adjust previous pointer in following chunk
4. Insert coalesced chunk into free tree

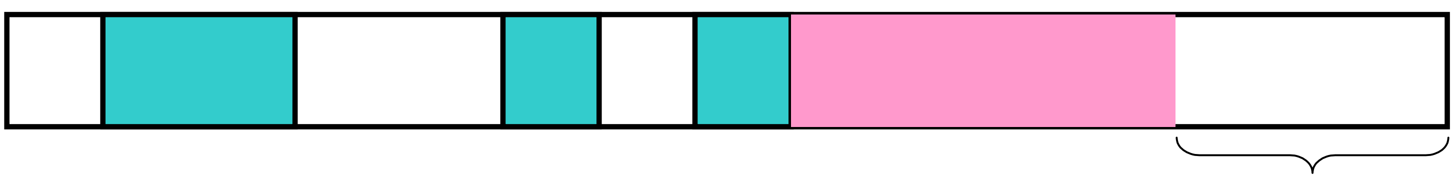
Arena is expanded in large chunks

Want to allocate a chunk:



No suitable free space, so expand arena …

Might expand by more than is needed to reduce frequency of expansion



Surplus space is placed into the free tree

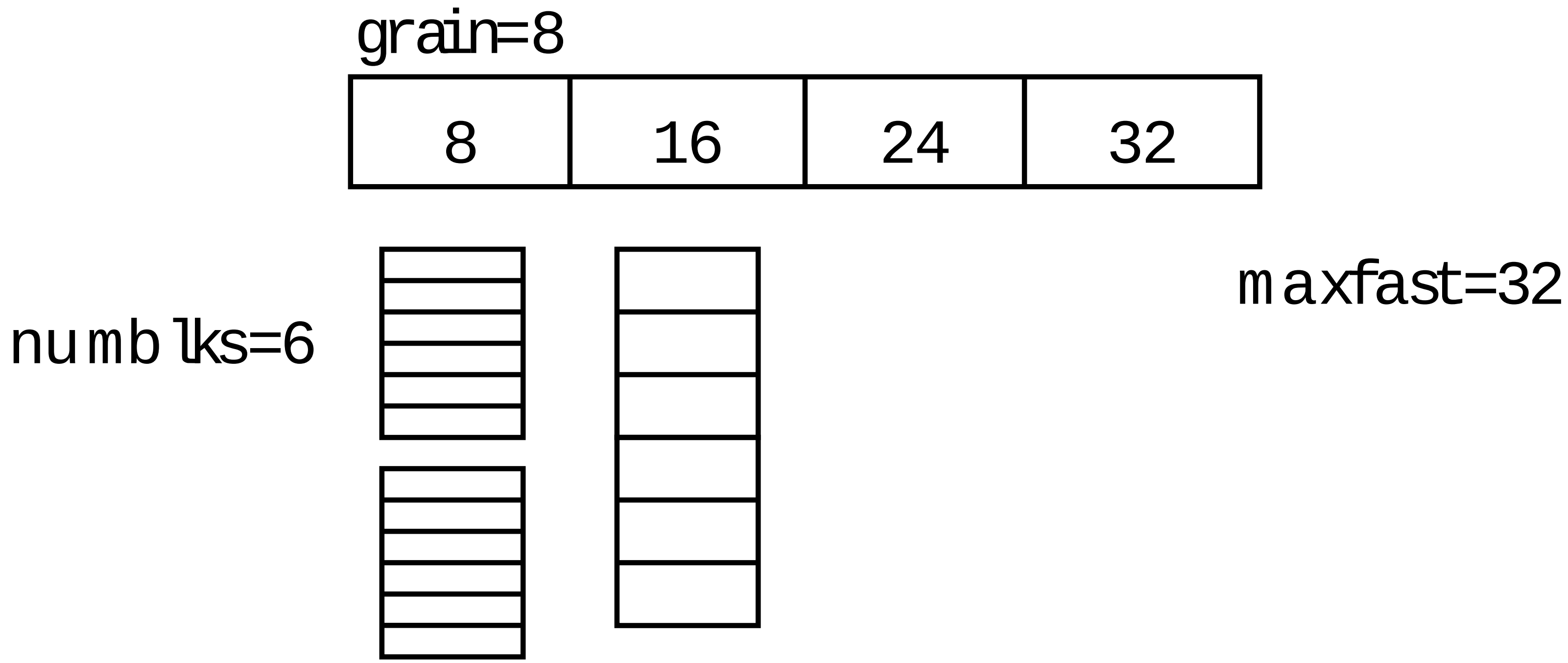
Stop and reflect

- The heap typically only ever expands
- Implement a best-fit approach to allocation
- Pre-allocate for performance reasons
- Coalesce on free
- Free tree exists outside of managed memory – good for vhand, cache and TLB
- Cost of managing free tree can be considerable

Small block allocator (SBA)

- Pools of fixed size blocks
- Small allocations are rounded up to one of the fixed sizes
- Allocations and frees are very quick – simple linked list operations
- Significant overhead:
  - Allocate more space than asked for
  - Pre-allocate to populate the pool when first used
- Pools are memory taken from the regular allocator
  - Never returned to the regular allocator, even when free
  - Cannot coalesce SBA and regular free space

Small block allocator (SBA)



SBA pros and cons

- Reduced allocation/free costs:
  - Hash, then linked list operation
  - No tree manipulation
  - No coalescing
- Increased memory overhead:
  - Pre-allocation based on NUMBLKS
  - Inflexible fixed-sized blocks
- Can help fragmentation
- Can cause fragmentation

Tuning the HP-UX SBA

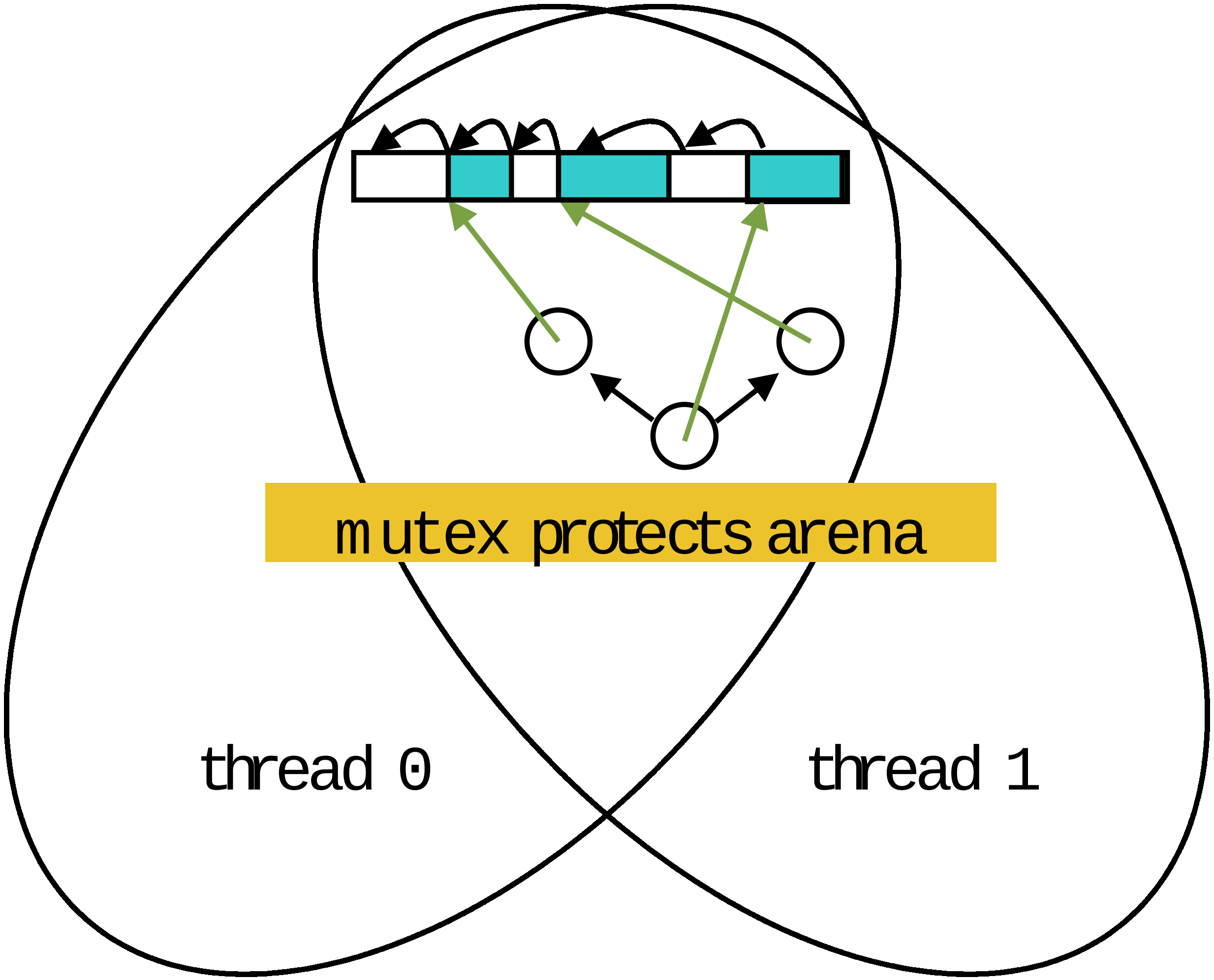
- Most platforms have SBA, but traditionally it must be enabled through application calls to mallopt()
  - Optimal tuning may vary by use
  - Most application vendors don't bother
- HP-UX allows SBA to be enabled through an environment variable:

```
_M_SBA_OPTS=maxfast:numblks:grain
```

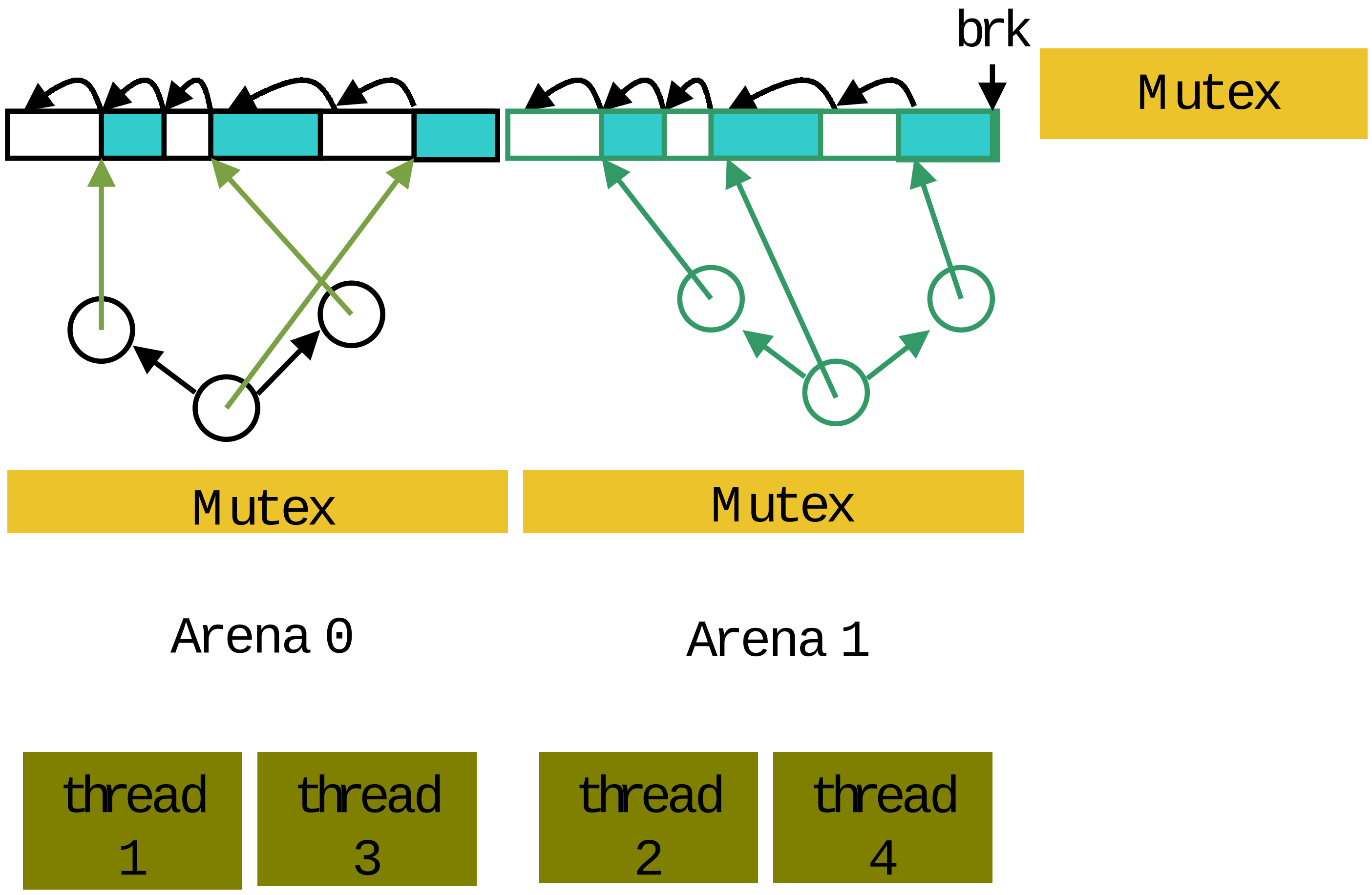
e.g.

```
# export _M_SBA_OPTS=512:100:64
# ./myapp
```

Multi-threaded processes and malloc



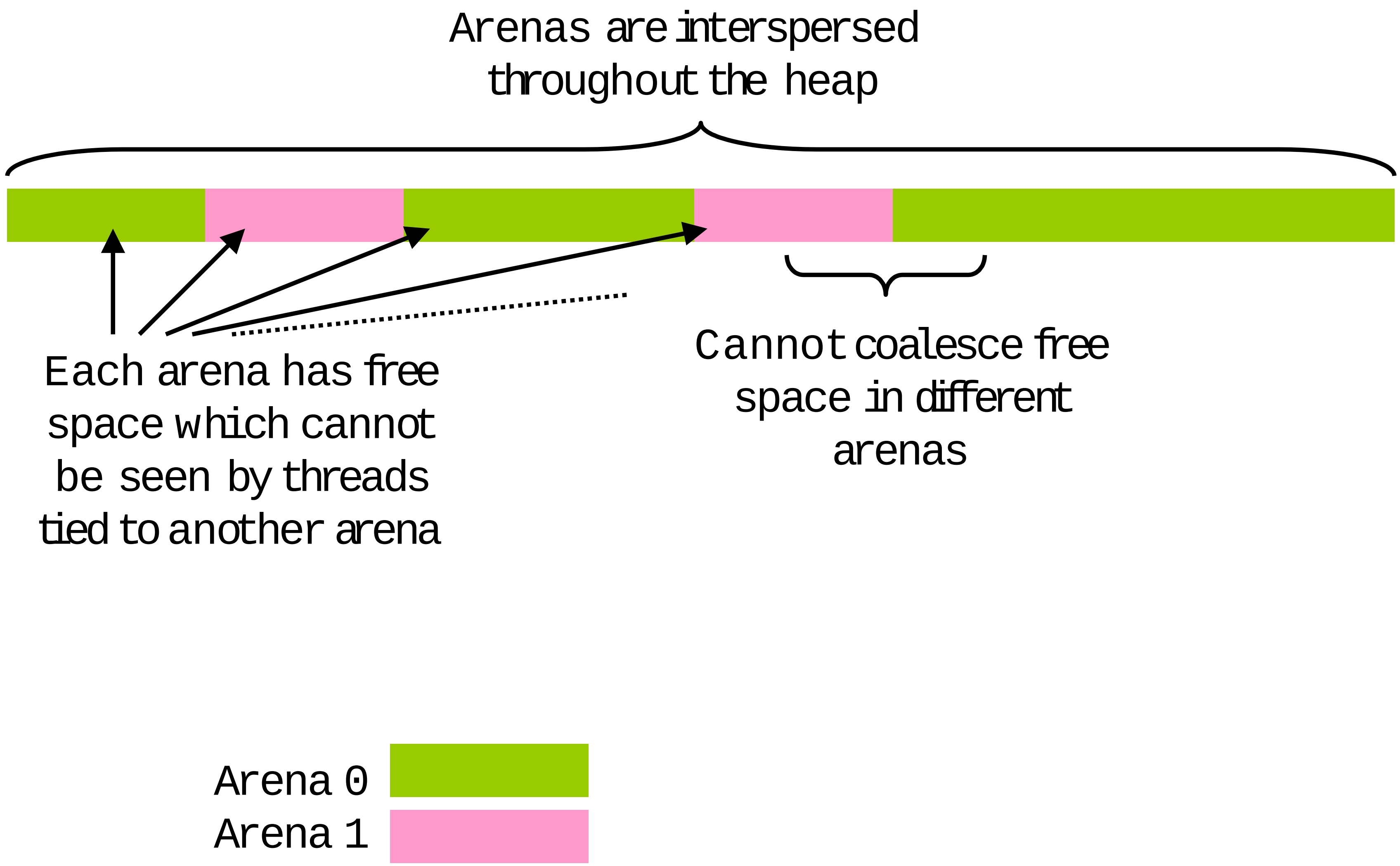
Multiple arenas in multi-threaded processes



Multi-arena malloc

- By default, multi-threaded processes have eight arenas (single-threaded have one)
- Each thread is assigned to an arena for all its allocations, for its lifetime
- Assignment is based on simple load balancing (earlier versions round-robin)
- Memory is always released into the arena from which it was allocated
  - Prevents leaks arising from repeated allocations by one thread being freed by another into a different arena
  - Still some potential for mutex contention, but greatly reduced.

Multi-arena malloc has high overhead



Pros and cons of multi-arena malloc

- Potentially enormous performance improvement for malloc-intensive multi-threaded processes
- Increased memory use:
  - Free space in one arena cannot be allocated by a thread tied to another
  - Free space from different arenas cannot be coalesced
  - Each arena has its own SBA if enabled
  - Applications can take a lot longer to settle to stable memory use

Tuning the HP-UX multi-arena malloc

Multi-arena malloc is controlled through an environment variable:

```
_M_ARENA_OPTS=num arenas:expansion pages
```

where expansion pages is 1 – 4096 4K pages (default 32)

Example:

```
# export _M_ARENA_OPTS=4:128
# ./myapp
```

Thread local cache

- Private cache that requires no locking
- Caches a definable number of previously used blocks
- Provides deferred coalescing
- Organized into buckets by size
  - buckets cover a power-of-2, e.g. 2^8 – $(2^{(9)}-1)$
  - blocks ordered by size within bucket
  - replacement policies exist for each bucket and the entire cache
- Caches of exiting threads are stored for reuse
- A retirement age defines how long we store a cache awaiting reuse before discarding

Tuning the thread local cache

- Three parameters:
  - bucket_size

    The average number of blocks per bucket (max is 4xbucket_size).

    Legal values: 0 (disable) through (8x4096)
  - number_of_buckets

    Largest block cached will be $2^{(number\_of\_buckets)}$

    Legal values: 8 through 32
  - retirement_age

    Number of minutes for which we'll retain unused caches for possible reuse

    Legal values: 0 through (24 x 60)

Tuning the thread local cache

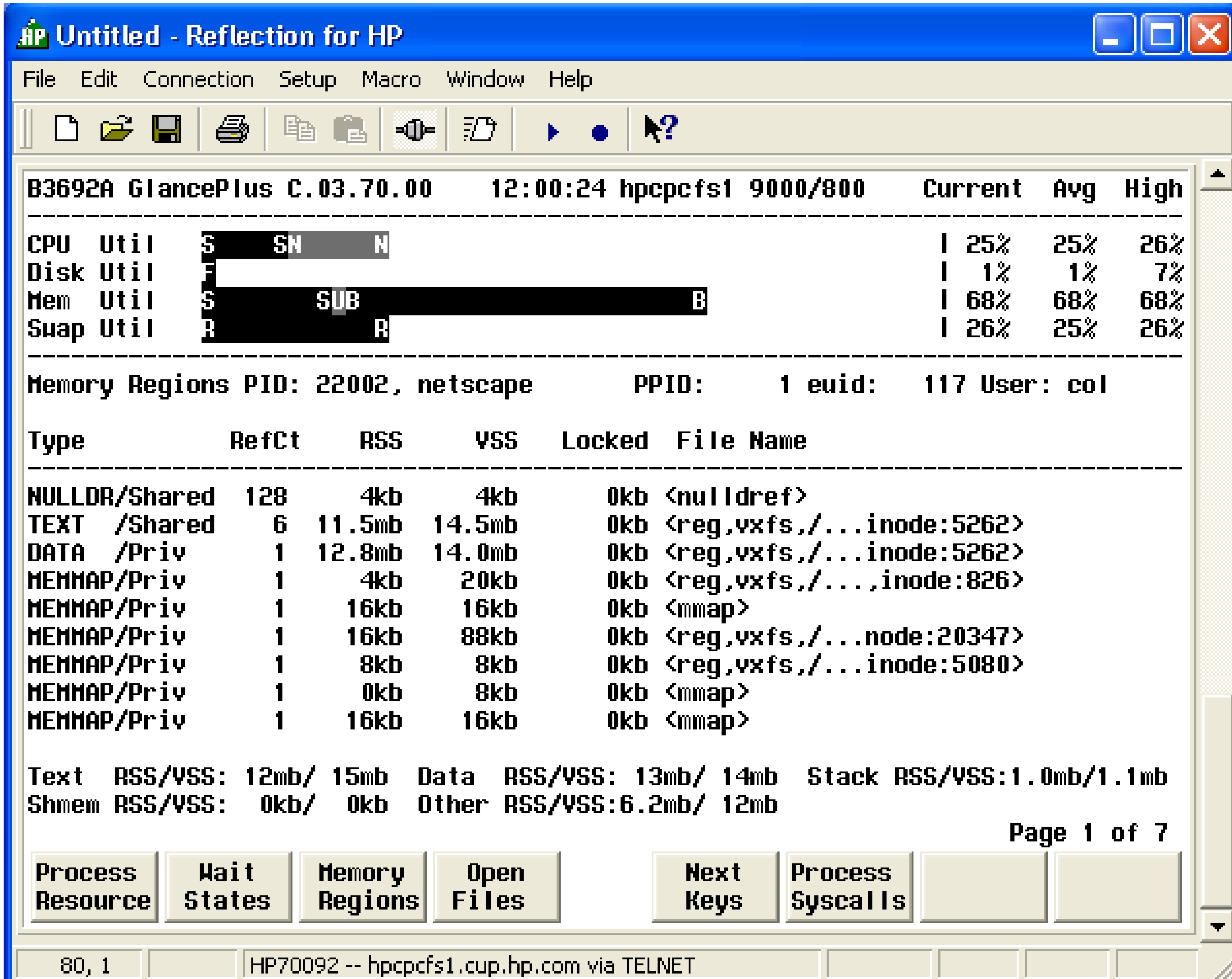
- Tune through an environment variable:

```
_M_CACHE_OPTS=bucket_size:buckets:retirement_age
```

Fragmentation of the heap

- Lots of free memory but in chunks that are too small
- Steps we take to avoid fragmentation
  - Best-fit allocation
  - Coalescing of free space
  - Separation of malloc metadata
- Things that increase fragmentation:
  - Dividing heap into many separate management domains through SBA, multi-arena and thread local cache
  - Small allocation units:
    - numblks for SBA
    - Arena expansion unit in _M_ARENA_OPTS
- Multiple arenas take longer to reach steady state

How much memory is in the heap? GlancePlus



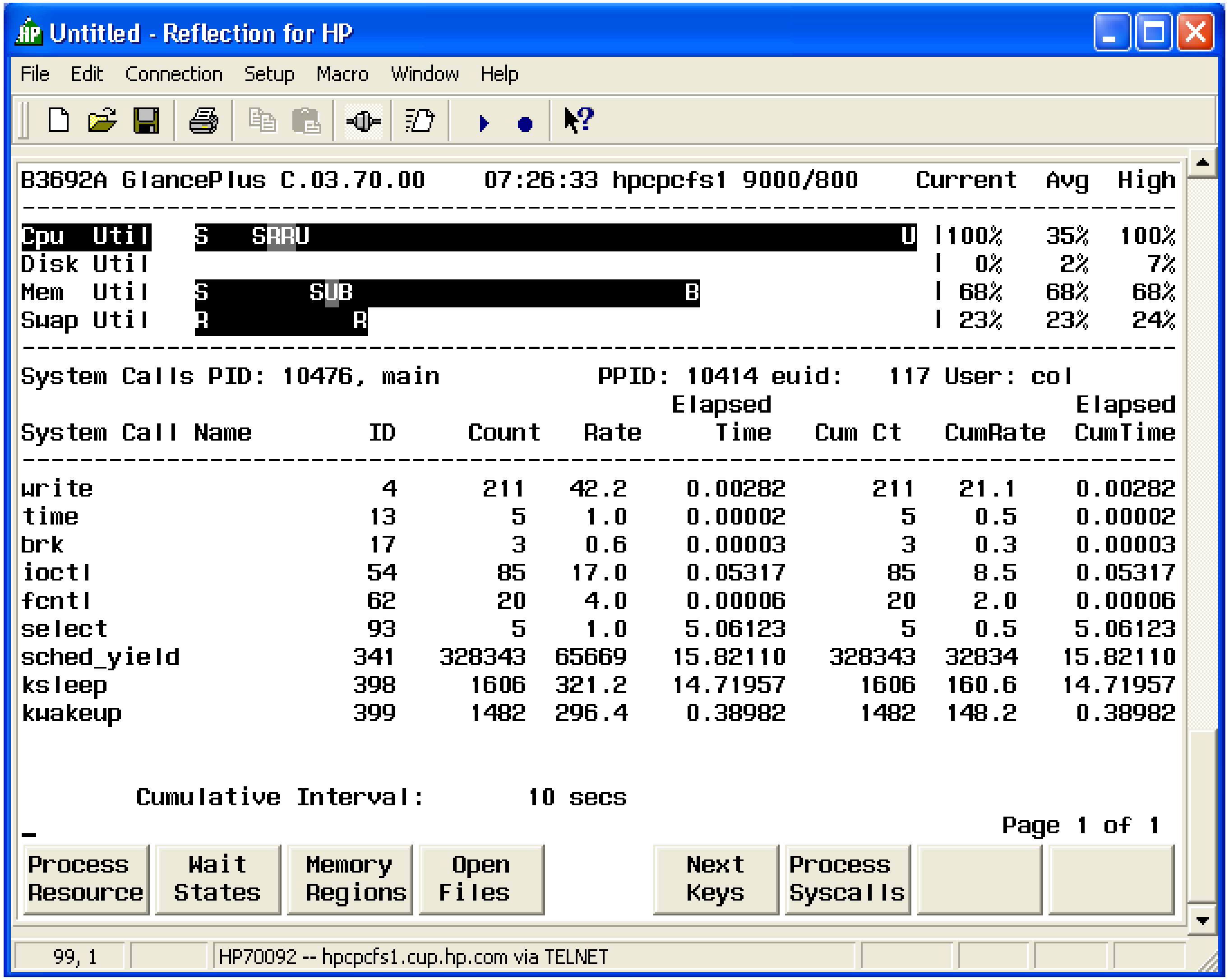
How is the memory used? mallinfo() and memorymap()

- mallinfo(3C) and memorymap(3C) can be called from your application
- mallinfo() returns structure, memorymap() prints to stdout
- mallinfo() sums data for all arenas, memorymap() shows each arena separately

mallinfo()

`arena`	total space in arena
`fsmblks`	number of bytes in free small blocks
`fordblks`	number of bytes in free ordinary blocks
`ordblks`	number of ordinary blocks
`smblks`	number of small blocks
`uordblks`	number of bytes in ordinary blocks in use
`usmblks`	number of bytes in small blocks in use

Would I benefit from more arenas?



Untitled - Reflection for HP

File Edit Connection Setup Macro Window Help

B3692A GlancePlus C.03.70.00 07:26:33 hpcpcfs1 9000/800 Current Avg High

		Current	Avg	High
Cpu Util	S SRRU U	100%	35%	100%
Disk Util		0%	2%	7%
Mem Util	S SUB B	68%	68%	68%
Swap Util	R R	23%	23%	24%

System Calls PID: 10476, main PPID: 10414 euid: 117 User: col

System Call Name	ID	Count	Rate	Elapsed Time	Cum Ct	CumRate	Elapsed CumTime
write	4	211	42.2	0.00282	211	21.1	0.00282
time	13	5	1.0	0.00002	5	0.5	0.00002
brk	17	3	0.6	0.00003	3	0.3	0.00003
ioctl	54	85	17.0	0.05317	85	8.5	0.05317
fcntl	62	20	4.0	0.00006	20	2.0	0.00006
select	93	5	1.0	5.06123	5	0.5	5.06123
sched_yield	341	328343	65669	15.82110	328343	32834	15.82110
ksleep	398	1606	321.2	14.71957	1606	160.6	14.71957
kwakeup	399	1482	296.4	0.38982	1482	148.2	0.38982

Cumulative Interval: 10 secs

Page 1 of 1

Process Resource | Wait States | Memory Regions | Open Files | Next Keys | Process Syscalls

99, 1 | HP70092 -- hpcpcfs1.cup.hp.com via TELNET

Which mutex?
Random samples with gdb

```
# gdb main
(gdb) run
<ctrl c>
Program received signal SIGINT, Interrupt.
(gdb) info threads
  6 system thread 707215  0x800003ffff638f74 in __ksleep ()
   from /usr/lib/pa20_64/libc.2
  5 system thread 707214  0x4000000000005ae8 in busy_loop ()
  4 system thread 707213  0x800003ffff7b936c in pthread_mutex_unlock ()
   from /usr/lib/pa20_64/libpthread.1
  3 system thread 707212  0x4000000000005ae8 in busy_loop ()
* 2 system thread 707220  0x800003ffff5d9c2c in .stub ()
   from /usr/lib/pa20_64/libc.2
  1 system thread 707208  0x800003ffff63bc54 in _select_sys ()
   from /usr/lib/pa20_64/libc.2
(gdb) t 6
(gdb) bt
#0  0x800003ffff638f74 in __ksleep () from /usr/lib/pa20_64/libc.2
#1  0x800003ffff7b8dc0 in pthread_mutex_lock ()
   from /usr/lib/pa20_64/libpthread.1
#2  0x800003ffff64b780 in __thread_mutex_lock () from /usr/lib/pa20_64/libc.2
#3  0x800003ffff5dbc70 in .stub () from /usr/lib/pa20_64/libc.2
#4  0x800003ffff5d9c2c in .stub () from /usr/lib/pa20_64/libc.2
#5  0x800003ffff5debdc in malloc () from /usr/lib/pa20_64/libc.2
#6  0x800003ffff74fdf8 in malloc (size=728) at lmt_libc.c:139
#7  0x4000000000003b10 in routine ()
#8  0x800003ffff7b5da0 in __pthread_body () from /usr/lib/pa20_64/libpthread.1
#9  0x800003ffff7bf874 in __pthread_start () from /usr/lib/pa20_64/libpthread.1
(gdb)
```

Which mutex? Breakpoint at __ksleep()

- __ksleep() called when sleeping for mutex
- When setting breakpoint in shared library must make private:

```
# pxdb -s enable testprog
```

- Set a breakpoint at __ksleep():

```
(gdb) break __ksleep
Breakpoint 1 at 0xc000000001f5f48
```

- Script some commands for when breakpoint 1 is hit:

```
(gdb) commands 1
>backtrace
>continue
>end
```

- Continue execution of the program:

```
(gdb) continue
```

Would I benefit from enabling the SBA?

- Look for CPU spent in malloc-related functions, e.g.:
  - malloc, free
  - real_malloc, real_free
  - tree-insert, tree_cut, tree_concatenate…
- Best tool is prospect

  www.hp.com/go/prospect

Would I benefit from enabling the SBA?

pcnt	Hits	Secs	Routine name	Filename
21%	95	0.95	pthread_mutex_unlock	/usr/lib/hpux32/libpthread.so.1
17%	76	0.76	routine	/home/col/malloc/main
11%	51	0.51	real_malloc	/usr/lib/hpux32/libc.so.1
10%	43	0.43	busy_loop	/home/col/malloc/main
9%	41	0.41	real_free	/usr/lib/hpux32/libc.so.1
8%	36	0.36	pthread_mutex_lock	/usr/lib/hpux32/libpthread.so.1
6%	25	0.25	rand_r	/usr/lib/hpux32/libc.so.1
4%	16	0.16	free	/usr/lib/hpux32/libc.so.1
4%	16	0.16	__thread_mutex_lock	/usr/lib/hpux32/libc.so.1
3%	15	0.15	__thread_mutex_unlock	/usr/lib/hpux32/libc.so.1
3%	12	0.12	malloc	/usr/lib/hpux32/libc.so.1
2%	9	0.09	_malloc	/usr/lib/hpux32/libc.so.1

Would I benefit from enabling the thread-local cache?

- Look for CPU spent in mutex- and malloc-related functions, e.g.:
  - pthread_mutex_lock/unlock
  - malloc, free
  - real_malloc, real_free
  - tree-insert, tree_cut, tree_concatenate…
- Best tool is prospect

  www.hp.com/go/prospect

Would I benefit from enabling the thread-local cache?

pcnt	Hits	Secs	Routine name	Filename
21%	95	0.95	pthread_mutex_unlock	/usr/lib/hpux32/libpthread.so.1
17%	76	0.76	routine	/home/col/malloc/main
11%	51	0.51	real_malloc	/usr/lib/hpux32/libc.so.1
10%	43	0.43	busy_loop	/home/col/malloc/main
9%	41	0.41	real_free	/usr/lib/hpux32/libc.so.1
8%	36	0.36	pthread_mutex_lock	/usr/lib/hpux32/libpthread.so.1
6%	25	0.25	rand_r	/usr/lib/hpux32/libc.so.1
4%	16	0.16	free	/usr/lib/hpux32/libc.so.1
4%	16	0.16	__thread_mutex_lock	/usr/lib/hpux32/libc.so.1
3%	15	0.15	__thread_mutex_unlock	/usr/lib/hpux32/libc.so.1
3%	12	0.12	malloc	/usr/lib/hpux32/libc.so.1
2%	9	0.09	_malloc	/usr/lib/hpux32/libc.so.1

A Case Study: mallbench

- 10 concurrent threads
- Repeatedly:
  - 50% chance of calling malloc
  - Do some work
  - 50% chance of calling free
- Try to keep memory use within desired bounds
- Each allocation selects a size at random from 17 predefined sizes
- Results in malloc operations per second

Default Configuration

- HP-UX 11.23 on Itanium:
  - 8 arenas
  - Expansion unit 32 pages
  - SBA enabled:
    - maxfast=512, numblks=100, grain=16
  - Thread-local cache disabled
- 1x1 thread model
- 2,580,792 ops/sec
- Data segment:
  - RSS 4.0MB
  - VSS 5.0MB

Default configuration – Prospect profile

```
pcnt Routine name                Filename

 21% routine                     /home/col/malloc/main
 19% pthread_mutex_unlock        /usr/lib/hpux32/libpthread.so.1
 12% busy_loop                   /home/col/malloc/main
 11% real_malloc                 /usr/lib/hpux32/libc.so.1
 10% pthread_mutex_lock          /usr/lib/hpux32/libpthread.so.1
  7% real_free                   /usr/lib/hpux32/libc.so.1
  7% rand_r                      /usr/lib/hpux32/libc.so.1
  3% __thread_mutex_lock         /usr/lib/hpux32/libc.so.1
  3% free                        /usr/lib/hpux32/libc.so.1
  2%.stub                        /home/col/malloc/main
  2% __thread_mutex_unlock       /usr/lib/hpux32/libc.so.1
  1% malloc                      /usr/lib/hpux32/libc.so.1
  0% _malloc                     /usr/lib/hpux32/libc.so.1
  0% tree_insert                 /usr/lib/hpux32/libc.so.1
  0% tree_concatenate            /usr/lib/hpux32/libc.so.1
  0% T_82_27f5_cl_tree_delete /usr/lib/hpux32/libc.so.1
```

Remove mutex contention

- Engage thread-local cache
- _M_CACHE_OPTS=1024:16:20
- 5,074,398 ops/sec (+97%)
- Data segment:
  - RSS 5.5MB (+37%)
  - VSS 6.6MB (+32%)

Remove mutex contention

```
pcnt Routine name            Filename

 36% routine                 /home/col/malloc/main
 17% busy_loop               /home/col/malloc/main
 15% get_cached_block        /usr/lib/hpux32/libc.so.1
  7% rand_r                  /usr/lib/hpux32/libc.so.1
  6% cache_ordinary_block    /usr/lib/hpux32/libc.so.1
  6% add_to_cache            /usr/lib/hpux32/libc.so.1
  4% free                    /usr/lib/hpux32/libc.so.1
  3% cache_small_block       /usr/lib/hpux32/libc.so.1
  3% malloc                  /usr/lib/hpux32/libc.so.1
  1% arena_id                /usr/lib/hpux32/libc.so.1
  1% div32U                  /usr/lib/hpux32/libc.so.1
```

Expand the SBA

- _M_CACHE_OPTS=1024:16:20
- _M_SBA_OPTS=16348:32:256
- 6,070,517 ops/sec (+135%)
- Data segment:
  - RSS 5.4MB (+35%)
  - VSS 7.5MB (+50%)

Expand the SBA

```
pcnt Routine name                Filename

 44% routine                     /home/col/malloc/main
 13% busy_loop                   /home/col/malloc/main
 11% rand_r                      /usr/lib/hpux32/libc.so.1
  9% get_cached_block            /usr/lib/hpux32/libc.so.1
  6% add_to_cache                /usr/lib/hpux32/libc.so.1
  6% cache_small_block           /usr/lib/hpux32/libc.so.1
  5% free                        /usr/lib/hpux32/libc.so.1
  2% malloc                      /usr/lib/hpux32/libc.so.1
  2% div32U                      /usr/lib/hpux32/libc.so.1
  1% arena_id                    /usr/lib/hpux32/libc.so.1
  0% cache_ordinary_block        /usr/lib/hpux32/libc.so.1
```

Reduce the number of arenas

- _M_CACHE_OPTS=1024:16:20
- _M_SBA_OPTS=16348:32:256
- _M_ARENA_OPTS=1:32
- 6,114,896 ops/sec (+138%)
- Data segment:
  - RSS 4.6MB (+15%)
  - VSS 5.7MB (+14%)

Reduce the number of arenas

```
pcnt Routine name                Filename

 43% routine                     /home/col/malloc/main
 15% busy_loop                   /home/col/malloc/main
  9% rand_r                      /usr/lib/hpux32/libc.so.1
  8% get_cached_block            /usr/lib/hpux32/libc.so.1
  7% add_to_cache                /usr/lib/hpux32/libc.so.1
  4% free                        /usr/lib/hpux32/libc.so.1
  4% malloc                      /usr/lib/hpux32/libc.so.1
  4% cache_small_block           /usr/lib/hpux32/libc.so.1
  3% div32U                      /usr/lib/hpux32/libc.so.1
  2% arena_id                    /usr/lib/hpux32/libc.so.1
  1% cache_ordinary_block        /usr/lib/hpux32/libc.so.1
```

HP WORLD 2003

Solutions and Technology Conference & Expo

Interex, Encompass and HP bring you a powerful new HP World.

interex
shared knowledge • shared power

encompass
AN HP USER GROUP

hp
invent