

"Dude - Where's My RAM?" A deep dive into how Python uses memory.

David Malcolm, Red Hat dmalcolm@redhat.com

PyCon US 2011

top - 18:39:08 up 4 min, 4 users, load average: 0.57, 0.49, 0.22 Tasks: 226 total, 1 running, 225 sleeping, 0 stopped, 0 zombie Cpu(s): 0.3%us, 0.7%sy, 0.0%ni, 98.4%id, 0.6%wa, 0.0%hi, 0.0%si, 0.0%st Mem: 7988236k total, 4860028k used, 3128208k free, 77420k buffers <u>S</u>wap: 3833848k total, 0k used, 3833848k free, 382320k cached

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+ COMMAND
2654	david	20	0	3460m	3.3g	2036	S	0.0	42.9	0:02.37 python
2683	david	20	0	429m	314m	2032	S	0.0	4.0	0:00.22 python
2694	david	20	0	767m	97m	42m	S	0.0	1.2	0:02.03 chromium-browse
2805	david	20	0	574m	37m	27m	S	0.0	0.5	0:00.14 chromium-browse
2799	david	20	0	1128m	35m	19m	S	0.0	0.5	0:00.24 chromium-browse
2120	david	20	0	550m	34m	10m	S	0.0	0.4	0:01.01 gnome-settings-
2697	david	20	0	540m	34m	25m	S	0.0	0.4	0:00.14 chromium-browse
2767	david	20	0	1129m	32m	19m	S	0.0	0.4	0:00.21 chromium-browse
2764	david	20	0	1120m	30m	18m	S	0.0	0.4	0:00.12 chromium-browse
2796	david	20	0	548m	30m	16m	S	0.0	0.4	0:00.09 chromium-browse
2784	david	20	0	1120m	29m	17m	S	0.0	0.4	0:00.09 chromium-browse
2780	david	20	0	1120m	29m	17m	S	0.0	0.4	0:00.11 chromium-browse
2802	david	20	0	1120m	29m	17m	S	0.0	0.4	0:00.09 chromium-browse
2771	david	20	0	1120m	29m	17m	S	0.0	0.4	0:00.10 chromium-browse



top - 18:39:08 up 4 min, 4 users, load average: 0.57, 0.49, 0.22 Tasks: 226 total, 1 running, 225 sleeping, 0 stopped, 0 zombie Cpu(s): 0.3%us, 0.7%sy, 0.0%ni, 98.4%id, 0.6%wa, 0.0%hi, 0.0%si, 0.0%st Mem: 7988236k total, 4860028k used, 3128208k free, 77420k buffers <u>S</u>wap: 3833848k total, 0k used, 3833848k free, 382320k cached

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2654	david	20	0	3460m	3.3g	2036	S	0.0	42.9	0:02.37	python
2683	david	20	0	429m	314m	2032	S	0.0	4.0	0:00.22	python
2694	david	20	0	767m	97m	42m	S	0.0	1.2	0:02.03	chromium-browse
2805	david	20	0	574m	37m	27m	S	0.0	0.5	0:00.14	chromium-browse
2799	david	20	0	1128m	35m	19m	S	0.0	0.5	0:00.24	chromium-browse
2120	david	20	0	550m	34m	10m	S	0.0	0.4	0:01.01	gnome-settings-
2697	david	20	0	540m	34m	25m	S	0.0	0.4	0:00.14	chromium-browse
2767	david	20	0	1129m	32m	19m	S	0.0	0.4	0:00.21	chromium-browse
2764	david	20	0	1120m	30m	18m	S	0.0	0.4	0:00.12	chromium-browse
2796	david	20	0	548m	30m	16m	S	0.0	0.4	0:00.09	chromium-browse
2784	david	20	0	1120m	29m	17m	S	0.0	0.4	0:00.09	chromium-browse
2780	david	20	0	1120m	29m	17m	S	0.0	0.4	0:00.11	chromium-browse
2802	david	20	0	1120m	29m	17m	S	0.0	0.4	0:00.09	chromium-browse
2771	david	20	0	1120m	29m	17m	S	0.0	0.4	0:00.10	chromium-browse



Overview

An incomplete tour of:

- CPython's memory usage:
 - How it works
 - How it goes wrong
 - Some fixes
- Tools you can use to track memory usage
- Introduction to my new gdb-heap tool https://fedorahosted.org/gdb-heap/



Memory overuse: symptoms

- A given workload uses more memory than you'd like
- Memory leaks: gradual increase in the memory usage of a process
- High-water-mark memory usage
- Lack of sharing between different processes



CPython Memory overuse: causes

- Inefficient representations
- True leaks:
 - Bugs in refcounting
 - Bugs in tp_traverse
- Apparent leaks:
 - Too much caching (user code)
 - Too much caching (implementation code)
 - Heap fragmentation
- Lack of sharing: "read-only" access to objects generates writes to memory (ob_refcnt)

Memory overuse: fixes

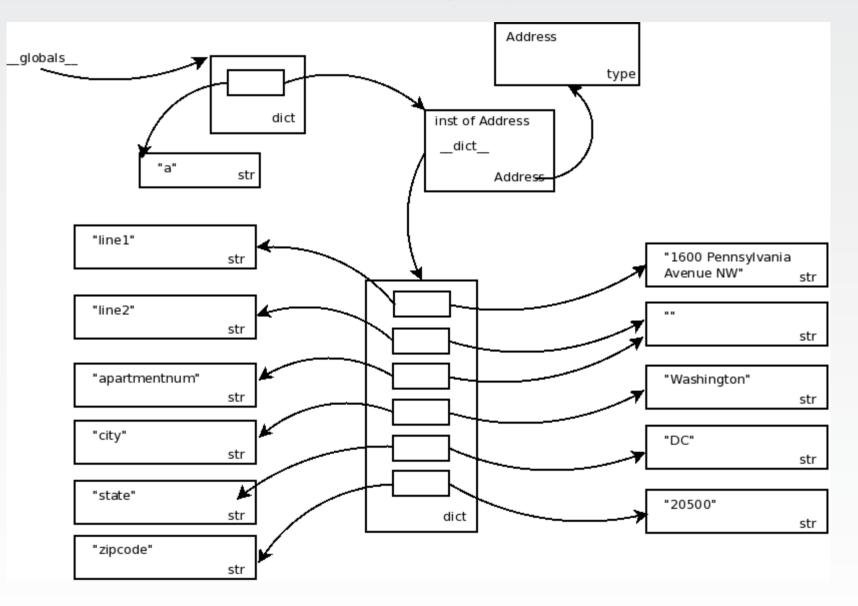
- Better data representations
- Fix the bugs
- Evict your caches
- Use PyPy (it's not just the JIT)



Python...

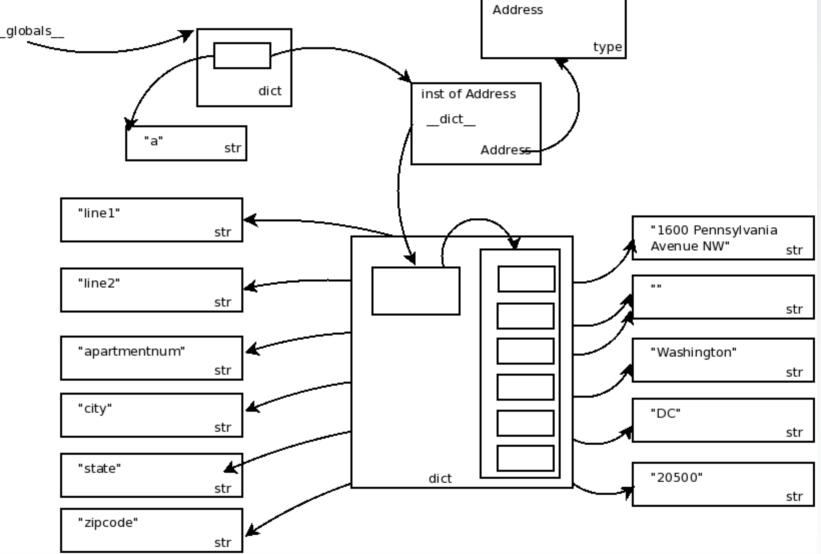


...as an object reference graph





...as allocated blocks of memory (CPython)





sys.getsizeof(obj)

- "How big is this object, in bytes?"
 - Includes buffers allocated for obj
 - Doesn't include other objects referenced by obj
 - CPython 2.6 onwards
 - Implemented for all of the builtin types
 - 3rd party extension modules might get this wrong



Sizes of fundamental types, in bytes (optimized CPython)

Туре	32-bit	64-bit
int (python2) (Single instances for range [-5, 257); custom allocator)	12	24
long (python2) / int (python3) (Single instances for range [-5, 257) on py3k)	14 + (2 * num "digits")	30 + (2 * num "digits")
str (python2) / bytes (python3) (Single instances for empty string and all length-1 values)	24 + length	40 + length
unicode (python2) / str (python3) (Single instances for empty string and all length-1 values for Latin-1)	28 + (2 or 4 * length)	52 + (2 or 4 * length)
list	32 + (4 * length)	72 + (8 * length)
tuple (Shared instance of empty tuple; custom allocator)	24 + (4 * length)	64 + (8 * length)
float (Custom allocator)	16	24



Sizes of small dictionaries

(optimized CPython)

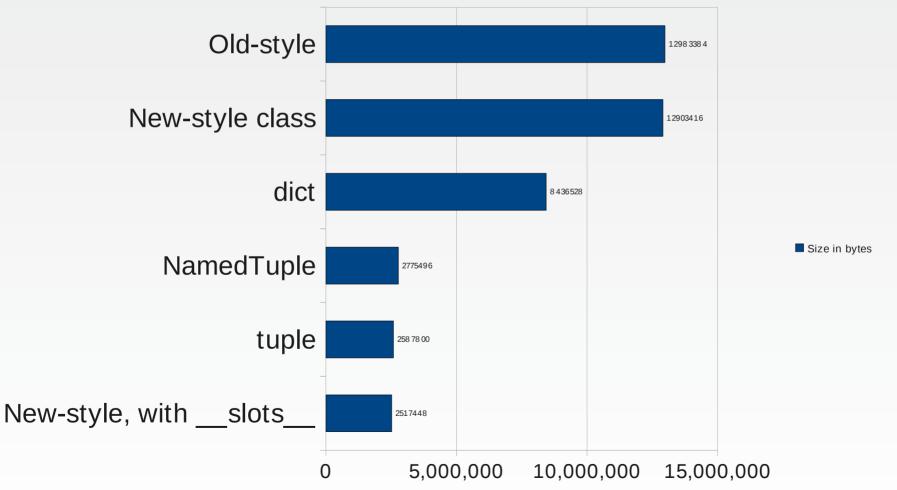
This directly affects the sizes of most objects in memory: __dict___

	32-bit	64-bit
Base size	136 + (12 * per external PyDictEntry tables)	280 + (24 per external PyDictEntry table)
0-5 entries	136	280
6-21 entries	520	1048
22-85 entries	1672	3352



Different object representations can have very different sizes

10,000 instances of a 6-field type (CPython)





The __slots__ optimization in CPython

```
class NewStyle(object):
      slots = ('line1', 'line2', 'apartmentnum',
          'city', 'state', 'zipcode')
    def init (self,
                 <u>line1, line2, apartmentnum,</u>
                 city, state, zipcode):
        self.line1 = line1
        self.line2 = line2
        self.apartmentnum = apartmentnum
        self.city = city
        self.state = state
        self.zipcode = zipcode
```



Meliae

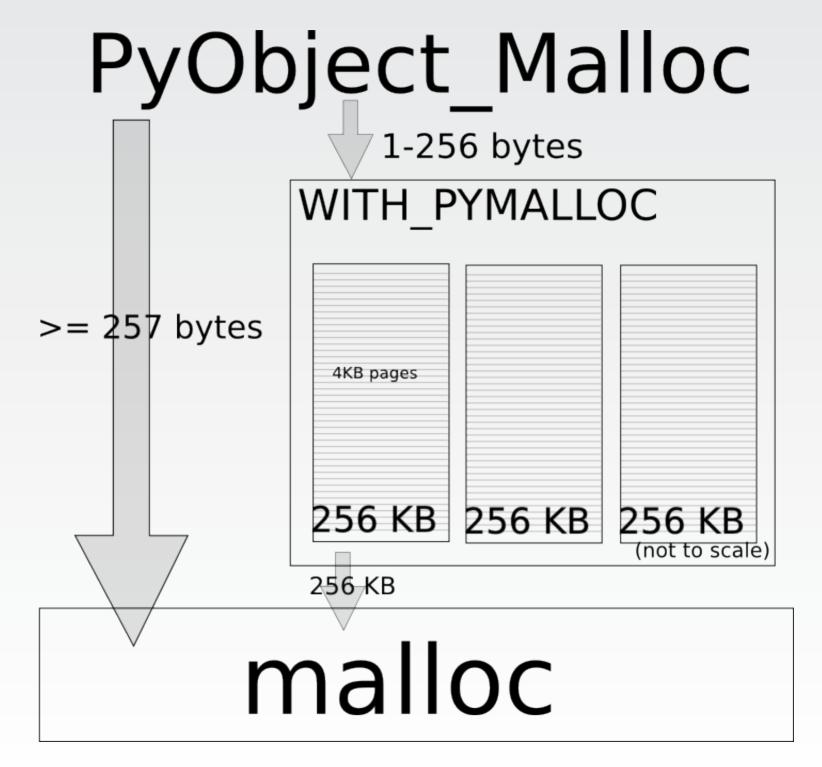
https://launchpad.net/meliae

- Python extension module for taking a snapshot of the object reference graph, as seen by the GC.
 - Everything, via gc.get_objects()
 - Everything "downstream" of a given object
- Dumps object reference graph in JSON form
 - address, type, ____name___, len, value, refs
- Tools for analyzing results

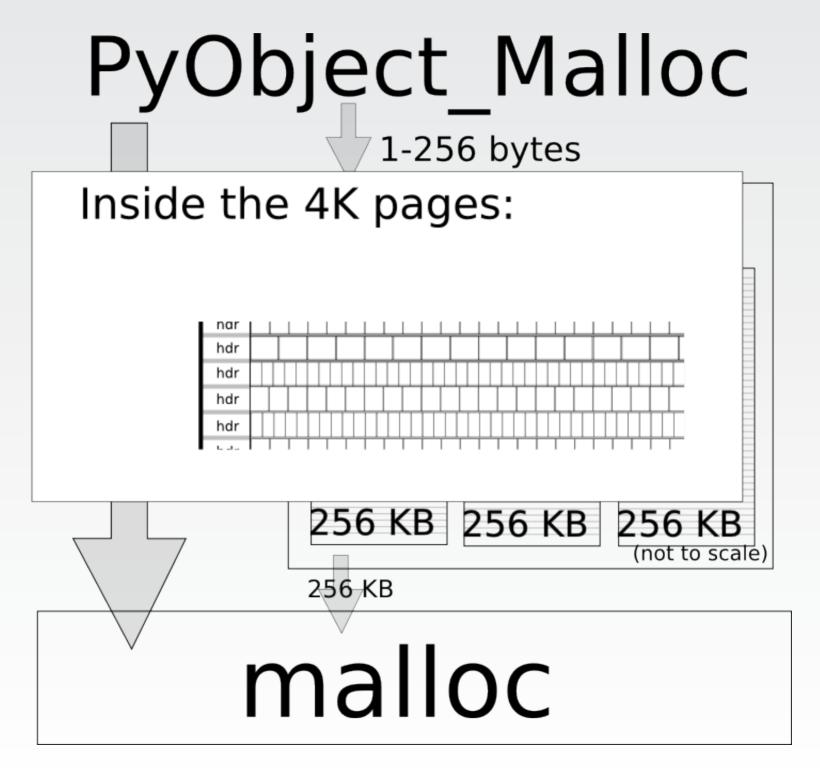


Below the object reference graph...





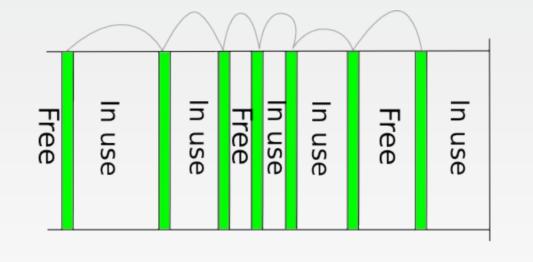


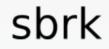




Malloc in GNU libc: boundary tags

• sbrk and anonymous mmap regions:







00400000-00401000 r-x	p 0000	00000 fd:	91 2635	565
00600000-00602000 rw-		00000 fd:		
00602000-006c7000 rw-		00000 00:0		
3ce2000000-3ce201e000	r-xp	000000000	fd:01	6116
3ce221d000-3ce221e000	rp	0001d000	fd:01	6116
3ce221e000-3ce221f000	rw-p	0001e000	fd:01	6116
3ce221f000-3ce2220000	rw-p	000000000	00:00	0
3ce2400000-3ce2574000	r-xp	000000000	fd:01	6166
3ce2574000-3ce2774000	p	00174000	fd:01	6166
3ce2774000-3ce2778000	rp	00174000	fd:01	6166
3ce2778000-3ce2779000	rw-p	00178000	fd:01	6166
3ce2779000-3ce277e000	rw-p	000000000	00:00	0
3ce2800000-3ce2817000	r-xp	000000000	fd:01	6169
3ce2817000-3ce2a16000	p	00017000	fd:01	6169
3ce2a16000-3ce2a17000	rp	00016000	fd:01	6169
3ce2a17000-3ce2a18000	rw-p	00017000	fd:01	6169
3ce2a18000-3ce2a1c000	rw-p	000000000	00:00	0
3ce2c00000-3ce2c02000	r-xp	000000000	fd:01	6172
3ce2c02000-3ce2e02000	p	00002000	fd:01	6172
3ce2e02000-3ce2e03000	rp	00002000	fd:01	6172
3ce2e03000-3ce2e04000	rw-p	00003000	fd:01	6172
3ce3000000-3ce3083000	r-xp	000000000	fd:01	6174
3ce3083000-3ce3282000	p	00083000	fd:01	6174
3ce3282000-3ce3283000	rp	00082000	fd:01	6174
3ce3283000-3ce3284000	rw-p	00083000	fd:01	6174
3ce3c00000-3ce3c3b000	r-xp	000000000	fd:01	4570
3ce3c3b000-3ce3e3b000	p	0003b000	fd:01	4570
3ce3e3b000-3ce3e43000	rw-p	0003b000	fd:01	4570
3ce3e43000-3ce3e44000	rw-p	000000000	00:00	Θ
3cf1400000-3cf1402000	r-xp	000000000	fd:01	6200
3cf1402000-3cf1601000	p	00002000	fd:01	6200
:				

/usr/bin/python
/usr/bin/python
[heap]
/lib64/ld-2.12.2.so
/lib64/ld-2.12.2.so
/lib64/ld-2.12.2.so
/11004/10 2:12:2:30
/lib64/libc-2.12.2.so
/lib64/libc-2.12.2.so
/lib64/libc-2.12.2.so
/lib64/libc-2.12.2.so
/lib64/libpthread-2.12.2.so
/lib64/libpthread-2.12.2.so
/lib64/libpthread-2.12.2.so
/lib64/libpthread-2.12.2.so
/lib64/libdl-2.12.2.so
/lib64/libdl-2.12.2.so
/lib64/libdl-2.12.2.so
/lib64/libdl-2.12.2.so
/lib64/libm-2.12.2.so
/lib64/libm-2.12.2.so
/lib64/libm-2.12.2.so
/lib64/libm-2.12.2.so
/lib64/libreadline.so.6.1
/lib64/libreadline.so.6.1
/lib64/libreadline.so.6.1
/libe//libutil 2 12 2
/lib64/libutil-2.12.2.so
/lib64/libutil-2.12.2.so



00400000-00401000 r-x	0000	00000 fd:0	91 2635	565
00600000-00602000 rw-	0000	00000 fd:0	91 2635	565
00602000-006c7000 rw-	0000	00000 00:0	0 0	
3ce2000000-3ce201e000	r-xp	000000000	fd:01	6116
3ce221d000-3ce221e000	rp	0001d000	fd:01	6116
3ce221e000-3ce221f000	rw-p	0001e000	fd:01	6116
3ce221f000-3ce2220000	rw-p	000000000	00:00	Θ
3ce2400000-3ce2574000	r-xp	000000000	fd:01	6166
3ce2574000-3ce2774000	p	00174000	fd:01	6166
3ce2774000-3ce2778000	rp	00174000	fd:01	6166
3ce2778000-3ce2779000	rw-p	00178000	fd:01	6166
3ce2779000-3ce277e000	rw-p	000000000	00:00	Θ
3ce2800000-3ce2817000	r-xp	000000000	fd:01	6169
3ce2817000-3ce2a16000	p	00017000	fd:01	6169
3ce2a16000-3ce2a17000	rp	00016000	fd:01	6169
3ce2a17000-3ce2a18000	rw-p	00017000	fd:01	6169
3ce2a18000-3ce2a1c000	rw-p	000000000	00:00	Θ
3ce2c00000-3ce2c02000	r-xp	000000000	fd:01	6172
3ce2c02000-3ce2e02000	p	00002000	fd:01	6172
3ce2e02000-3ce2e03000	rp	00002000	fd:01	6172
3ce2e03000-3ce2e04000	rw-p	00003000	fd:01	6172
3ce3000000-3ce3083000	r-xp	000000000	fd:01	6174
3ce3083000-3ce3282000	p	00083000	fd:01	6174
3ce3282000-3ce3283000	rp	00082000	fd:01	6174
3ce3283000-3ce3284000	rw-p	00083000	fd:01	6174
3ce3c00000-3ce3c3b000	r-xp	000000000	fd:01	4570
3ce3c3b000-3ce3e3b000	p	0003b000	fd:01	4570
3ce3e3b000-3ce3e43000	rw-p	0003b000	fd:01	4570
3ce3e43000-3ce3e44000	rw-p	000000000	00:00	0
3cf1400000-3cf1402000	r-xp	000000000	fd:01	6200
3cf1402000-3cf1601000	p	00002000	fd:01	6200
:				

/usr/bin/python /usr/bin/python
[heap]
/lib64/ld-2.12.2.so
/lib64/ld-2.12.2.so
/lib64/ld-2.12.2.so
(1)
/lib64/libc-2.12.2.so /lib64/libc-2.12.2.so
/lib64/libc-2.12.2.so
/lib64/libc-2.12.2.so
/ (1004/ (100 2.12.2.30
/lib64/libpthread-2.12.2.so
/lib64/libpthread-2.12.2.so
/lib64/libpthread-2.12.2.so
/lib64/libpthread-2.12.2.so
/lib64/libdl-2.12.2.so
/lib64/libdl-2.12.2.so
/lib64/libdl-2.12.2.so
/lib64/libdl-2.12.2.so
/lib64/libm-2.12.2.so
/lib64/libm-2.12.2.so
/lib64/libm-2.12.2.so
/lib64/libm-2.12.2.so
/lib64/libreadline.so.6.1 /lib64/libreadline.so.6.1
/lib64/libreadline.so.6.1
/lib64/libutil-2.12.2.so
/lib64/libutil-2.12.2.so



3ce2e03000-3ce2e04000 rw-p 00003000 fd:01 6172 3ce3000000-3ce3083000 r-xp 00000000 fd:01 6174 3ce3083000-3ce3282000 ---p 00083000 fd:01 6174 3ce3282000-3ce3283000 r--p 00082000 fd:01 6174 3ce3283000-3ce3284000 rw-p 00083000 fd:01 6174 3ce3c00000-3ce3c3b000 r-xp 00000000 fd:01 4570 3ce3c3b000-3ce3e3b000 ---p 0003b000 fd:01 4570 3ce3e3b000-3ce3e43000 rw-p 0003b000 fd:01 4570 3ce3e43000-3ce3e44000 rw-p 00000000 00:00 0 3cf1400000-3cf1402000 r-xp 00000000 fd:01 6200 3cf1402000-3cf1601000 ---p 00002000 fd:01 6200 3cf1601000-3cf1602000 r--p 00001000 fd:01 6200 3cf1602000-3cf1603000 rw-p 00002000 fd:01 6200 3cf3c00000-3cf3d71000 r-xp 00000000 fd:01 268718 3cf3d71000-3cf3f70000 ---p 00171000 fd:01 268718 3cf3f70000-3cf3fac000 rw-p 00170000 fd:01 268718 3cf3fac000-3cf3fba000 rw-p 00000000 00:00 0 3cf4800000-3cf4824000 r-xp 00000000 fd:01 6211 3cf4824000-3cf4a23000 ---p 00024000 fd:01 6211 3cf4a23000-3cf4a28000 rw-p 00023000 fd:01 6211 7ffffle50000-7ffffle55000 r-xp 00000000 fd:01 394363 7ffffle55000-7ffff2054000 ---p 00005000 fd:01 394363 7ffff2054000-7ffff2056000 rw-p 00004000 fd:01 394363 7ffff2056000-7ffff7ee7000 r--p 00000000 fd:01 274057 7ffff7ee7000-7ffff7f59000 rw-p 00000000 00:00 0 7ffff7f5a000-7ffff7fe1000 rw-p 00000000 00:00 0 7ffff7ff5000-7ffff7ffc000 r--s 00000000 fd:01 267489 7ffff7ffc000-7ffff7ffe000 rw-p 00000000 00:00 0 7ffff7ffe000-7ffff7fff000 r-xp 00000000 00:00 0 7ffffffe7000-7fffffff000 rw-p 00000000 00:00 0 ffffffffff600000-fffffffff601000 r-xp 00000000 00:00 0 (END)

/lib64/libdl-2.12.2.so /lib64/libm-2.12.2.so /lib64/libm-2.12.2.so /lib64/libm-2.12.2.so /lib64/libm-2.12.2.so /lib64/libreadline.so.6.1 /lib64/libreadline.so.6.1 /lib64/libreadline.so.6.1 /lib64/libutil-2.12.2.so /lib64/libutil-2.12.2.so /lib64/libutil-2.12.2.so /lib64/libutil-2.12.2.so /usr/lib64/libpython2.6.so.1.0 /usr/lib64/libpython2.6.so.1.0 /usr/lib64/libpython2.6.so.1.0 /lib64/libtinfo.so.5.7 /lib64/libtinfo.so.5.7 /lib64/libtinfo.so.5.7 /usr/lib64/python2.6/lib-dynload/readline.so /usr/lib64/python2.6/lib-dynload/readline.so /usr/lib64/python2.6/lib-dynload/readline.so

/usr/lib64/gconv/gconv-modules.cache

/usr/lib/locale/locale-archive

[vdso] [stack] [vsyscall]



3ce2e03000-3ce2e04000	rw-p	00003000	fd:01	6172	
3ce3000000-3ce3083000	r-xp	000000000	fd:01	6174	
3ce3083000-3ce3282000	p	00083000	fd:01	6174	
3ce3282000-3ce3283000	rp	00082000	fd:01	6174	
3ce3283000-3ce3284000	rw-p	00083000	fd:01	6174	
3ce3c00000-3ce3c3b000	r-xp	000000000	fd:01	4570	
3ce3c3b000-3ce3e3b000	p	0003b000	fd:01	4570	
3ce3e3b000-3ce3e43000	rw-p	0003b000	fd:01	4570	
3ce3e43000-3ce3e44000	rw-p	000000000	00:00	Θ	
3cf1400000-3cf1402000	r-xp	000000000	fd:01	6200	
3cf1402000-3cf1601000	p	00002000	fd:01	6200	
3cf1601000-3cf1602000	rp	00001000	fd:01	6200	
3cf1602000-3cf1603000	rw-p	00002000	fd:01	6200	
3cf3c00000-3cf3d71000	r-xp	000000000	fd:01	268718	
3cf3d71000-3cf3f70000	p	00171000	fd:01	268718	
3cf3f70000-3cf3fac000	rw-p	00170000	fd:01	268718	
3cf3fac000-3cf3fba000	rw-p	000000000	00:00	Θ	
3cf4800000-3cf4824000	r-xp	000000000	fd:01	6211	
3cf4824000-3cf4a23000	p	00024000	fd:01	6211	
3cf4a23000-3cf4a28000	rw-p	00023000	fd:01	6211	
7ffff1e50000-7ffff1e55		-xp 00000		:01 394363	
7ffff1e55000-7ffff2054		p 00005		:01 394363	
7ffff2054000-7ffff2056		~w-p 0000∠		:01 394363	
7ffff2056000-7ffff7ee7		<u>p</u> 00000		1:01 274057	
7ffff7ee7000-7ffff7f59		~w-p 00000		0:00 0	
7ffff7f5a000-7ffff7fe1		-w-p 00000		0:00 0	
7ffff7ff5000-7ffff7ffc		<u>s 00000</u>		1:01 267489	
7ffff7ffc000-7ffff7ffe		~w-p 00000		0:00 0	
7ffff7ffe000-7ffff7ff		-xp 00000		0:00 0	
7ffffffe7000-7ffffffff		_w-p 00000		0:00 0	
<u> </u>	fffff	f601000 r-	-xp 000	000000 00:00	0
(END)					

/lib64/libdl-2.12.2.so /lib64/libm-2.12.2.so /lib64/libm-2.12.2.so /lib64/libm-2.12.2.so /lib64/libm-2.12.2.so /lib64/libreadline.so.6.1 /lib64/libreadline.so.6.1 /lib64/libreadline.so.6.1 /lib64/libutil-2.12.2.so /lib64/libutil-2.12.2.so /lib64/libutil-2.12.2.so /lib64/libutil-2.12.2.so /usr/lib64/libpython2.6.so.1.0 /usr/lib64/libpython2.6.so.1.0 /usr/lib64/libpython2.6.so.1.0 /lib64/libtinfo.so.5.7 /lib64/libtinfo.so.5.7 /lib64/libtinfo.so.5.7 /usr/lib64/python2.6/lib-dynload/readline.so /usr/lib64/python2.6/lib-dynload/readline.so /usr/lib64/python2.6/lib-dynload/readline.so /usr/lib/locale/locale-archive /usr/lib64/gconv/gconv-modules.cache

[vdso] [stack] [vsyscall]



How can we track down problems at the lower levels?

- Valgrind is one way...
 - But it introduces behavior changes
 - Slows things down
 - Can't attach to a running program
 - ... or deal with a core dump



Introducing gdb-heap

https://fedorahosted.org/gdb-heap/

- gdb 7 onwards can be scripted via in Python
- I've written Python code to analyze how the process being debugged is using its heap
- Doesn't require co-operation of the process under investigation
 - Usable on coredumps



gdb-heap (GNU)

- ~2600 lines of Python code, analyzing:
 - GNU libc's malloc/free implementation
 - CPython's allocators and objects
 - Other stuff: C++, GTK's GType
- Very low level
 - Great at dealing with problems deeper down in the stack (e.g. within a C library)
 - Analysis of processes built from mixed technologies e.g. C, C++ and Python



What gdb-heap does

- Locates the allocator implementations:
 - GNU libc's malloc/free implementation
 - CPython's PYMALLOC implementation
 - Other allocators?
- Walks the blocks of memory in use:
 - by malloc/realloc (sbrk and mmap)
 - Detects and subdivides the 256KB blocks used by PYMALLOC
- Tries to categorize every allocated block



Categorizing allocations "in situ"

- Casting to (PyObject*)
 - Do fields look valid? e.g. ((PyObject*)ptr)->ob_type
 - Also, it tries the (PyGC_Head) offset
 - If so: categorize using the ob_type
- Example:
 - a block of memory (void*)ptr that has ((PyObject*)(ptr+sizeof(PyGC_Head))->ob_type == &PyDict_Type
- Looking for a C++ v-table at the top of the block
- Do the bytes look like a C-style string?

...and other heuristics



Categorizing allocations via cross-references

- Follow statically-allocated pointers: do they point at this block?
 - Example: static PyObject *interned;
- Look at all the already-allocated blocks
 - Do any of them contain pointers?
 - Use this to categorize the blocks they point to
 - Rinse and repeat, with a scoring system
 - Example:
 - given an block known to be a python instance, categorize the <u>dict</u>, and the ma_table (if not ma_smalltable)



(gdb) heap Blocks retrieve	ed 10000				
Blocks analyzed					
Domain	Kind		Detail	Count	Allocated size
python	str			6,711	479,288
cpython	PyDictEntry table			167	456,944
cpython	PyDictEntry table	int	terned	1	200,704
python	str	by	tecode	647	91,960
uncategorized		32	bytes	2,862	91,584
python	code			647	82,816
uncategorized		4128	bytes	19	78,432
python	dict			256	73,776
python	function			610	73,200
python	wrapper_descriptor			905	72,400
uncategorized		72	bytes	844	60,768
uncategorized		592	bytes	99	58,608
uncategorized		64	bytes	801	51,264
uncategorized		49168	bytes	1	49,168
uncategorized		12304	bytes	3	36,912
python	type			37	36,192
uncategorized		36048	bytes	1	36,048
C	string data			102	35,000
python	builtin_function_or_method			461	33,192
uncategorized		288	bytes	90	25,920
python	method_descriptor			349	25,128



uncategorized		80 bytes	290	23,200
uncategorized		3088 bytes	6	18,528
python	ABCMeta		16	15,360
python	weakref		167	14,696
uncategorized		96 bytes	153	14,688
pyarena	freed pool chunk		227	14,632
uncategorized		88 bytes	162	14,256
python	set		60	13,920
python	member descriptor		191	13,752
cpython	PyDictEntry table	ABCMeta. dict	16	13,312
python	tuple		142	13,304
pyarena	pool header overhead		271	13,008
uncategorized		688 bytes	17	11,696
uncategorized		1008 bytes	11	11,088
cpython	PyListObject ob item table		86	10,640
uncategorized		480 bytes	22	10,560
uncategorized		464 bytes	21	9,744
uncategorized		1216 bytes	8	9,728
uncategorized		1184 bytes	8	9,472
uncategorized		448 bytes	21	9,408
uncategorized		496 bytes	18	8,928
uncategorized		8240 bytes	1	8,240
uncategorized		8208 bytes	1	8,208
uncategorized		7968 bytes	1	7,968
uncategorized		784 bytes	10	7,840
python	getset_descriptor		103	7,416



uncategorized		104 byt		7,384
uncategorized		1040 byt	es 7	7,280
python	list		92	6,624
uncategorized		112 byt	es 59	6,608
uncategorized		128 byt	es 50	6,400
uncategorized		1200 byt	es 5	6,000
uncategorized		120 byt	es 43	5,160
uncategorized		48 byt	es 101	4,848
python	dict	ABCMeta. dict	16	4,608
uncategorized		544 byt	es 8	4,352
uncategorized		4096 byt	es 1	4,096
Type <retur< td=""><td><pre>m> to continue, or q <return></return></pre></td><td>to quit</td><td></td><td></td></retur<>	<pre>m> to continue, or q <return></return></pre>	to quit		
uncategorized		136 byt	es 29	3,944
uncategorized		656 byt	es 6	3,936
uncategorized		3776 byt	es 1	3,776
uncategorized		752 byt	es 5	3,760
uncategorized		624 byt	es 6	3,744
uncategorized		1232 byt	es 3	3,696
cpython	PySetObject setentry table	-	4	3,648
python	module		64	3,584
uncategorized		160 byt	es 21	3,360
uncategorized		144 byt	es 23	3,312
uncategorized		640 byt	es 5	3,200
uncategorized		512 byt	es 6	3,072
python	frozenset		13	3,016
python	unknown		25	2,888



uncategorized			bytes	16	2,816
uncategorized		208	bytes	13	2,704
uncategorized		168	bytes	15	2,520
uncategorized		152	bytes	16	2,432
uncategorized		200	bytes	12	2,400
uncategorized		400	bytes	6	2,400
uncategorized		2272	bytes	1	2,272
uncategorized		528	bytes	4	2,112
python	sre.SRE Pattern			6	2,056
python				19	1,976
uncategorized		184	bytes	9	1,656
uncategorized		232	bytes	7	1,624
uncategorized		320	bytes	5	1,600
uncategorized		224	bytes	7	1,568
uncategorized		192	bytes	8	1,536
python	functools.partial			14	1,232
uncategorized		304	bytes	4	1,216
uncategorized		240	bytes	5	1,200
uncategorized		576	bytes	2	1,152
uncategorized		560	bytes	2	1,120
python	instancemethod			14	1,120
uncategorized		8	bytes	139	1,112
pyarena	alignment wastage			5	1,040
uncategorized		960	bytes	1	960
python	dict	_Printerc	lict	3	864
uncategorized		216	bytes	4	864



uncategorized		416 bytes	2	832
cpython	PyDictEntry table	CodecInfodict	1	784
uncategorized		352 bytes	2	704
uncategorized		336 bytes	2	672
uncategorized		672 bytes	1	672
python	dict	Quitter. dict	2	576
python	classmethod descriptor		7	504
uncategorized		248 bytes	2	496
python	classmethod		8	448
python	file		3	432
python	long		13	376
python	property		4	352
cpython	PyDictObject	Environ. dict	1	288
python	dict	CodecInfo. dict	1	288
uncategorized		272 bytes	1	272
uncategorized		256 bytes	1	256
python	unicode		5	240
uncategorized		40 bytes	6	240
python	Printer		3	192
uncategorized	—	24 bytes	7	168
python	sys.flags		1	144
python	Quitter		2	128
Type <return></return>	to continue, or q <return< td=""><td>> to quit</td><td></td><td></td></return<>	> to quit		
python	sys.float_info		1	112
python	staticmethod		2	112
_ python	CodecInfo		1	104



uncategorized		256	bytes	1	256
python	unicode			5	240
uncategorized		40	bytes	6	240
python	_Printer			3	192
uncategorized		24	bytes	7	168
python	sys.flags			1 2	144
python	Quitter			2	128
Type <return></return>	to continue, or q <return> [.]</return>	to quit			
python	sys.float_info			1	112
python	staticmethod			2	112
python	CodecInfo			1	104
python	PyCapsule			2 1 2 1	96
python	_Environ	old-	style	1	72
python	exceptions.MemoryError			1	72
python	exceptions.RuntimeError			1	72
python	_Helper			1	64
python	sys.version_info			1	64
python	cell			1	56
uncategorized		56	bytes	1	56
uncategorized		16	bytes	3	48
python	imp.NullImporter			3	48
cpython	PyUnicodeObject buffer			5	40
python	sys.long info			1	40
python	object			2	32
			TOTAL	18,745	2,628,896



Query language

- (gdb) heap select criteria
- Currently implemented criteria:
 - and, or, ==
 - "domain", "kind", "detail", "size"



(gdb) heap select kind="_intblock"									
Start	End	Domain	Kind	Detail					
0x000000000061bf60	0x000000000061c34f	cpython	intblock		00	00	00	00	00
0x000000000061c350	0x000000000061c73f	cpython	_intblock		60	bf	61	00	00
0x000000000061c740	0x000000000061cb2f	cpython	intblock		50	c3	61	00	00
0x000000000061cb30	0x000000000061cf1f	cpython	_intblock		40	c7	61	00	00
0x000000000061cf20	0x000000000061d30f	cpython	_intblock		30	cb	61	00	00
0x000000000061d310	0x000000000061d6ff	cpython	_intblock		20	cf	61	00	00
0x000000000061d700	0x000000000061daef	cpython	_intblock		10	d3	61	00	00
0x0000000000650f00	0x00000000006512ef	cpython	_intblock		00	d7	61	00	00
0×0000000000661200	0x00000000006615ef	cpython	_intblock		00	0f	65	00	00

(gdb)



```
(qdb) heap select kind="str" and detail="bytecode" and size<100
            Start
                                   End
                                        Domain
                                                Kind
                                                        Detail
                   0x00007ffff7ee87c7
0x00007ffff7ee8780
                                        python
                                                 str
                                                               01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 1a 00
                                                      bytecode
0x00007ffff7ee8ae0
                    0x00007ffff7ee8b27
                                        python
                                                                01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 1a 00
                                                 str
                                                      bytecode
                   0x00007ffff7eebadf
0x00007ffff7eebab0
                                        pvthon
                                                                01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 04 00
                                                 str
                                                      bytecode
0x00007ffff7eebae0
                    0x00007ffff7eebb0f
                                                                01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 04 00
                                        python
                                                 str
                                                      bvtecode
                    0x00007ffff7eebb6f
                                                      bytecode
                                                                01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 04 00
                                        python
0x00007ffff7eebb40
                                                 str
0x00007ffff7eebde0
                   0x00007ffff7eebe0f
                                        python
                                                      bytecode
                                                               01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 04 00
                                                 str
                   0x00007ffff7eebe3f
0x00007ffff7eebe10
                                        python
                                                                01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 04 00
                                                      bytecode
                                                 str
0x00007ffff7eebe70
                   0x00007ffff7eebe9f
                                        python
                                                      bytecode
                                                                01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 04 00
                                                 str
                                        python
0x00007ffff7eed620
                   0x00007ffff7eed66f
                                                      bytecode 01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 24 00
                                                 str
0x00007ffff7eee430
                   0x00007ffff7eee46f
                                        python
                                                      bytecode 01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 16 00
                                                 str
0x00007ffff7eeebb0
                   0x00007ffff7eeebef
                                        python
                                                 str
                                                      bytecode 01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 16 00
                   0x00007ffff7eeec6f
                                                      bytecode 01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 14 00
0x00007ffff7eeec30
                                        python
                                                 str
0x00007ffff7eeef70
                   0x00007ffff7eeefaf
                                                      bytecode 01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 11 00
                                        python
                                                 str
0x00007ffff7ef1978
                  0x00007ffff7ef19cf
                                        python
                                                 str
                                                      bytecode 01 00 00 00 00 00 00 00 c0 ae f8 f3 3c 00 00 00 29 00
---Type <return> to continue, or q <return> to quit---q
Ouit
(gdb) print (PyObject*)0x00007ffff7ee8780
$2 = 't\x00\x00\x87\x00\x00f\x01\x00d\x01\x00\x86\x00\x00|\x00\x00D\x83\x01\x00\x83\x01\x00S'
(gdb) python from dis import dis
(gdb) python dis('t\x00\x00\x87\x00\x00f\x01\x00d\x01\x00\x86\x00\x00|\x00\x00D\x83\x01\x00\x83\x01\x00S')
                                0 (0)
         0 LOAD GLOBAL
         3 LOAD CLOSURE
                                0
         6 BUILD TUPLE
                                1
         9 LOAD CONST
                                1(1)
         12 MAKE CLOSURE
                                0
        15 LOAD FAST
                                0 (0)
        18 GET ITER
         19 CALL FUNCTION
                                1
```



(gdb) heap cpython-allocators								
Objects/obmalloc.c: 3 arenas								
struct arena_object*	256KB buffer location Free pools							
0x000000000006021a0 0x00007ffff7ee7010	0 - 0x00007ffff7f2700f 17 / 63							
0x00000000000602170 0x00007ffff7f5a010	0 - 0x00007ffff7f9a00f							
0x00000000000602140 0x00007ffff7f9b010	0 - 0x00007ffff7fdb00f 0 / 63							



Caveats

- Too low-level?
 - doesn't know about garbage collector generations (yet)
 - doesn't understand the object reference graph (yet)
- Categorization can be hit-or-miss
- Could use more speed optimizations
- Ironically, uses too much RAM



Success story

- Analysing RAM usage of Fedora's software updater
 - identified large blocks of C library data
 - being kept alive via a cache at the Python level
 - being fixed for next release



Future plans

- Support for apr (httpd, mod_wsgi, etc)
- Anything else?



Summary

- A brief tour of some aspects of CPython's memory usage
- Introduction to a new tool for tracking problems down

http://fedorahosted.org/gdb-heap/



Q & A



Additional material



...via per-type allocators (tp_new/tp_alloc) (CPython)

- Specialized allocators for int, float, etc
 - High-water-mark behavior (until 2.6/3.0)
- PyDict_New



...via _PyObject_GC_Malloc (CPython)

- Reference-owning objects get a hidden PyGC_Head at the front:
 - 12 bytes on a 32-bit build of python
 - 32 bytes on a 64-bit build
- Allocation can trigger a garbage-collection
 - ...every 700 of such allocations, by default



Missing slide: reference counting (CPython)



Missing slide: the garbage collector (CPython)



Virtual memory

- All of the above are divided into 4K pages
- Not all of them are necessarily used by physical memory
- Some of them may be shared
 - Not as many as you'd like (ob_refcnt)
- KVM, KSM, and ob_refcnt



Investigative tools

- sys.getsizeof()
- Debug build
- Meliae
- Heapy
- valgrind
- gdb-heap



Debug build of Python

- Adds: _ob_next, _ob_prev pointers to the top of every object
- Doubly-linked list of all objects that think they're alive
- Adds sys.gettotalrefcount()
 - If this continually increases you may have a leak
 - Or a cache...
- Set "PYTHONDUMPREFS=1" in the environment to see all objects still alive on exit



Неару

http://guppy-pe.sourceforge.net/#Heapy



Valgrind (Linux)

- Low-level memory-leak finder
- Instrumented malloc/free
- As of CPython 2.7, 3.2, CPython can detect if it's being run under valgrind and automatically turn off the arena allocator



Fixing memory leaks: the crude and nasty way

- Kill the process at regular intervals
- Let it die, e.g. after 100 requests

For when you need to keep a system running, and haven't diagnosed the root cause of the leak yet

Obviously not ideal, but may save you in an emergency



Fixing memory leaks: the better way

Figure out the root cause Fix that



How do memory leaks happen?

- High-tide marks: caches
- ob_refcnt errors
- tp_traverse errors
- C and C++ bugs
- SWIG errors



High-tide marks: caches

- in user code: caches that are never purged
- allocator caches that aren't purged (older pythons)
 - the fixed arena high-watermark issue (fixed in 2.5a1; also RHEL 5.6)
 - the per-type allocators (int/float/method/frame/CFunction/tuple/unicode) (fixed in 2.6alpha1 and 3.0a3)

 See gcmodule.c:clear_freelists: http://svn.python.org/view?view=revision&revision=60797

• One-time initialization (e.g. in wrapped libraries)



Bugs: ob_refcnt

Objects with too high an ob_refcnt:

• think they're alive, but nothing references them.

Detection:

- Debug build of python
- Valgrind
- gdb-heap?
- meliae?



Bugs: tp_traverse

- Classes that don't call Py_VISIT on every held reference in their tp_traverse
- Classes that don't have a tp_traverse callback



Differences in a debug build of CPython

- Obmalloc arenas are disabled
- Extra padding around every allocation

