In Lieu of Swap: Analyzing Compressed RAM in Mac OS X and Linux

Golden G. Richard III

University of New Orleans and Arcane Alloy, LLC

golden@cs.uno.edu

Andrew Case

The Volatility Project

andrew@dfir.org

Who?

Professor of Computer Science and University Research Professor, Director, GNOCIA, University of New Orleans

http://www.cs.uno.edu/~golden

Digital forensics, OS internals, reverse engineering, offensive computing, pushing students to the brink of destruction, et al.

Founder, Arcane Alloy, LLC.

http://www.arcanealloy.com

Digital forensics, reverse engineering, malware analysis, security research, tool development, training.

Co-Founder, Partner / Photographer, High ISO Music, LLC.

http://www.highisomusic.com

Rock stars. Heavy Metal. Earplugs.

Forensics, Features, Privacy

- New OS features may negatively forensics
- e.g.:
  - Native whole disk encryption
  - Encrypted swap
  - Secure Recycle Bin / Trash facilities
  - Secure erasure during disk formatting
- Today's topic: **Compressed RAM**
- Claim: **positively impacts forensics**
- First some background

Where's the Evidence?

- Files and Deleted Files
- Filesystem metadata
- Application metadata
- Windows registry
- Print spool files
- Hibernation files
- Temp files
- Log files
- Slack space
- Swap files
- Browser caches
- Network traces

"Traditional" storage forensics

RAM: OS and app data structures

Memory analysis

Memory Analysis on RAM Dump

- Processes (dead/alive) ← e.g., discover unauthorized programs
- Open files ← e.g., detect keystroke loggers, hidden processes
- Network connections ← e.g., find backdoors, connections to contraband sites
- Volatile registry contents
- Volatile application data ← e.g., plaintext for encrypted material, chat messages, email fragments
- Other OS structures ← e.g., analysis of memory allocators to retrieve interesting structures, filesystem cache, etc.
- Encryption keys ← e.g., keys for whole disk encryption schemes

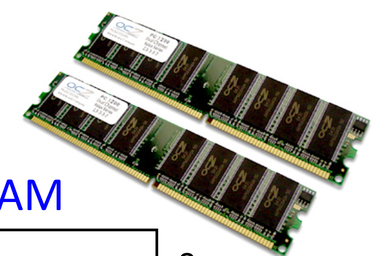
What's missing?

Memory Analysis: 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



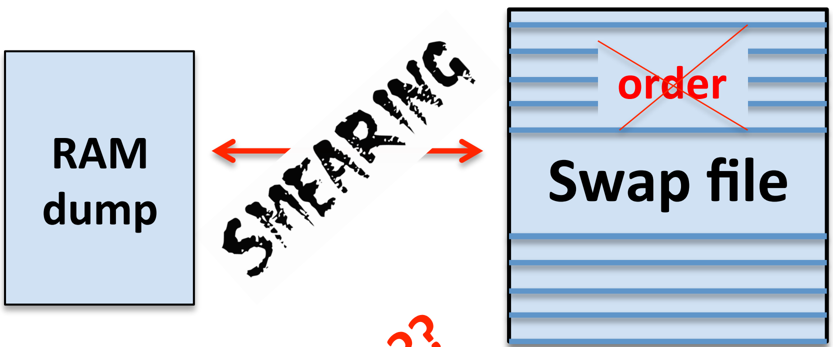
Live forensics / memory analysis: capture RAM contents

RAM

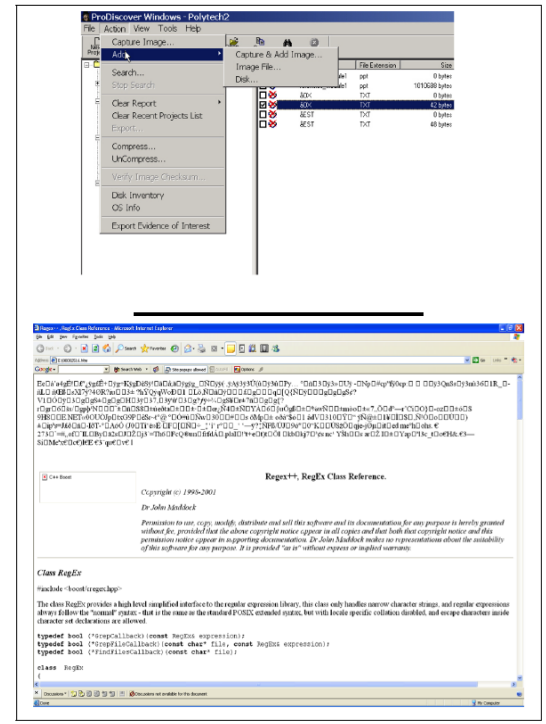
00000000000000000000000000000000	0
11110010010011010010010101010101	4096
01000001010010010010001000100010	…
111111111111111111111111111111111	…
. .	
0101010101010101010101010101010011	2GB

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



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 ;
  }
}
```

Background: Memory Management

2GB

- Need to allocate RAM to:
  - Operating system
  - Individual applications
  - All applications
    - Possibly fairly, or some other policy

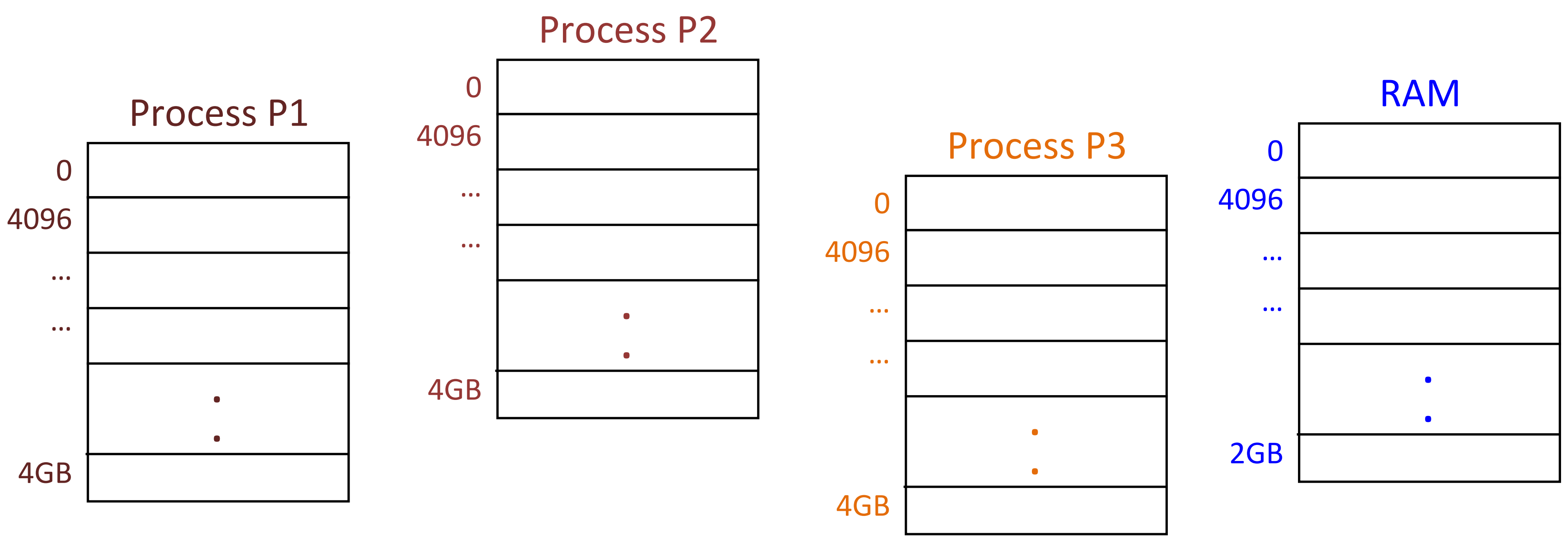
- OS's need to manage memory efficiently to feed RAM-hungry apps
- Ever-increasing "need" for more memory

Virtual Memory

- Provides contiguous, linear address space for processes
- Virtual address space is divided into fixed-size pages
  - Physical memory is divided into pages of same size
  - On x86 Intel, these are normally 4K
- OS keeps track of free and allocated pages
- Processes get only as many pages as they need

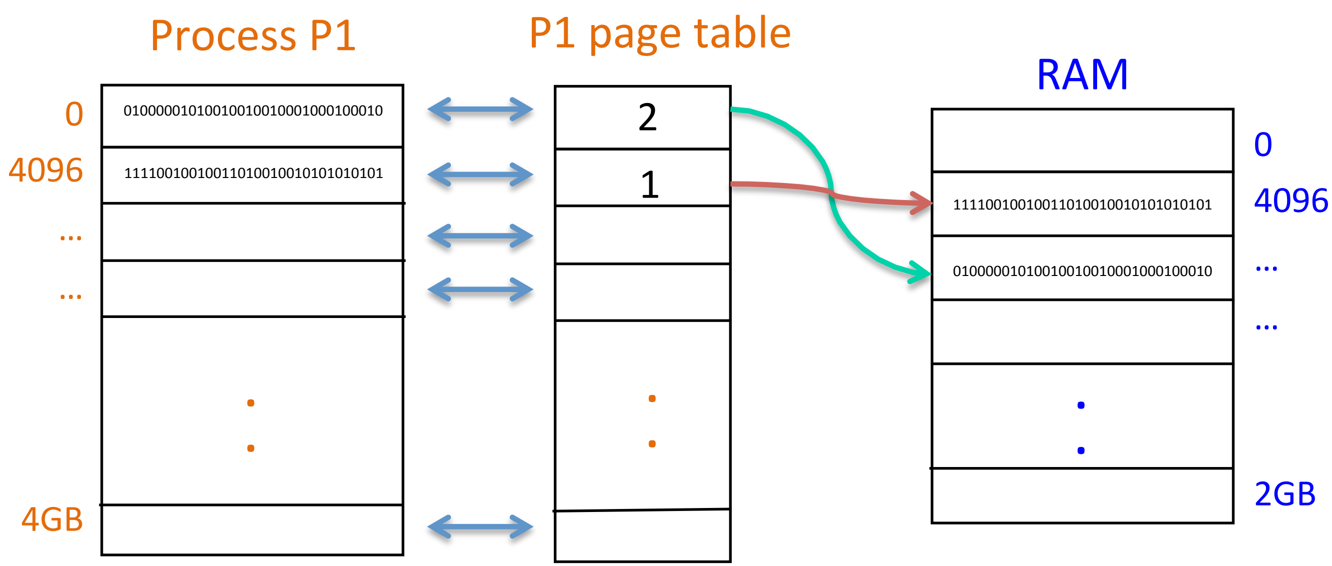
Logical vs. Physical Addresses

- Virtual address: Address whose scope is a particular process or OS kernel
- Physical address: Location in physical RAM



Paging: (Very) Simplified View

- Page tables (per process) are used to translate logical to physical addresses
- Pages for a particular process are generally not in contiguous order in RAM



64-bit Mode: 4-level Page Tables: 4K Pages

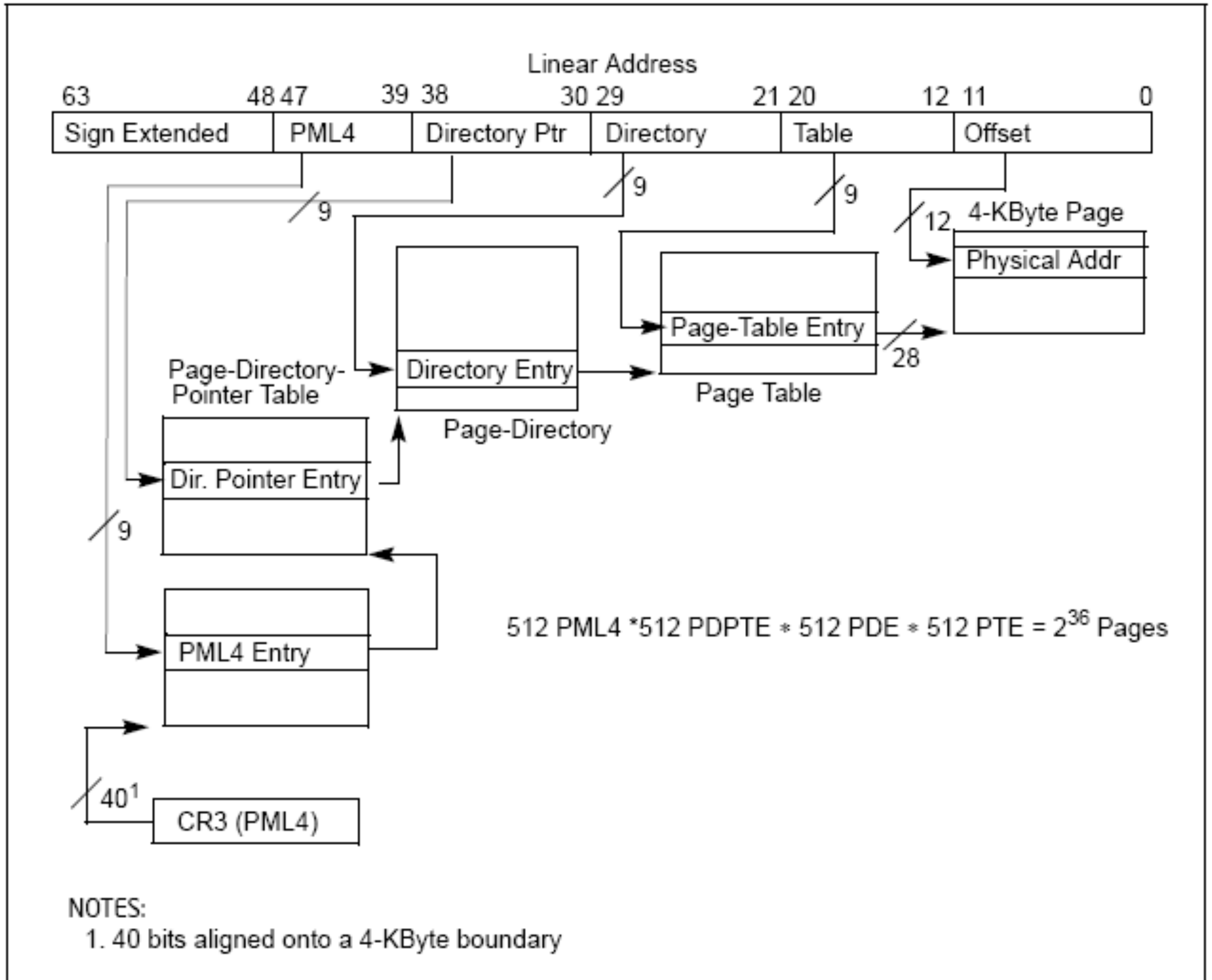


Figure 3-24. IA-32e Mode Paging Structures (4-KByte Pages)

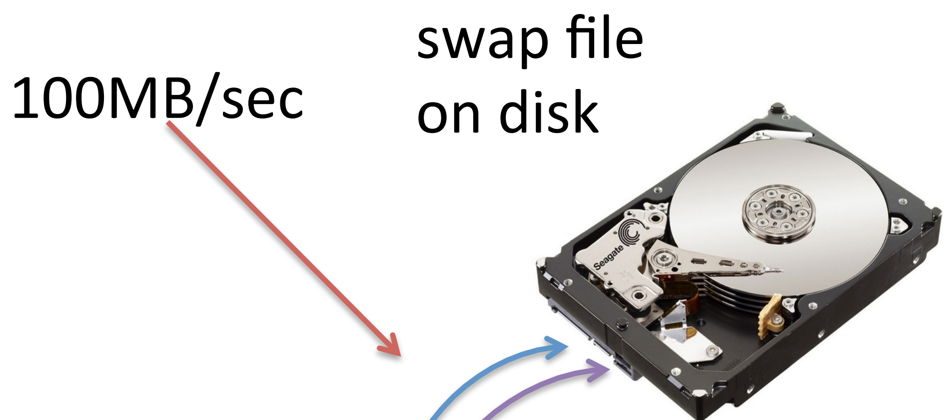
Stretching RAM

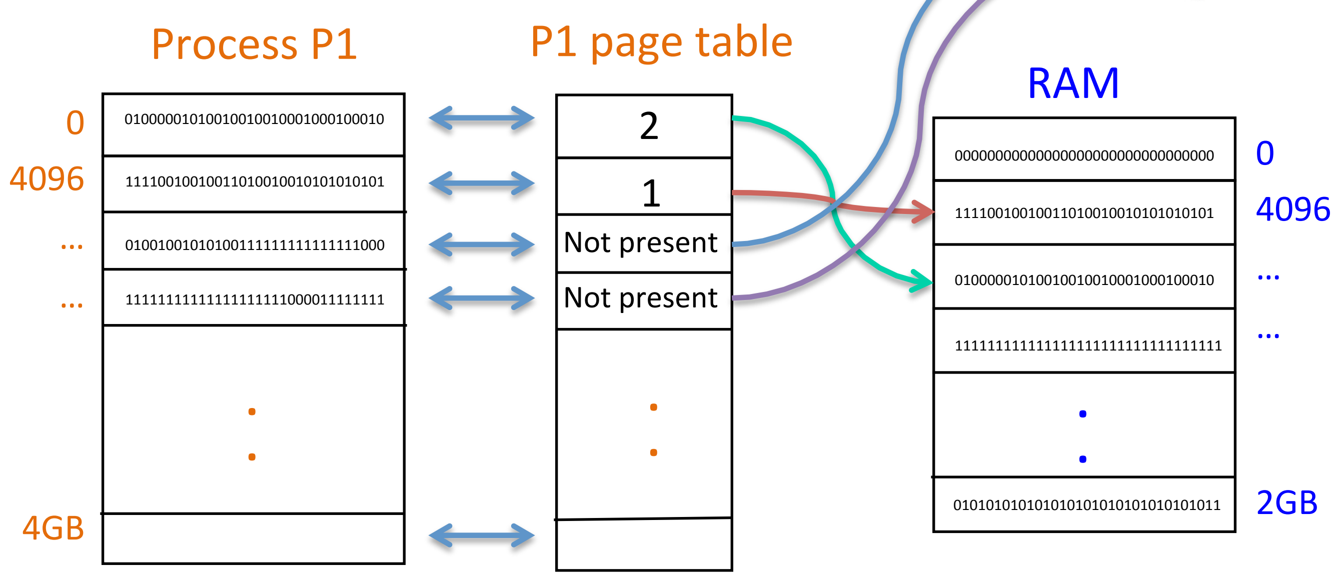
- Want to allocate more memory to processes than total amount of RAM
- Extend RAM by swapping pages to disk and retrieving as necessary
- Allows more (and bigger) processes to execute
- Workable, because:
  - Not all executing processes are active at once
  - Processes typically have “working sets” of pages, e.g., memory they’re actively using “now”

Virtual Memory with Swapping

For good performance, can't overdo it--the pipe between RAM and disk is small

Otherwise: THRASHING



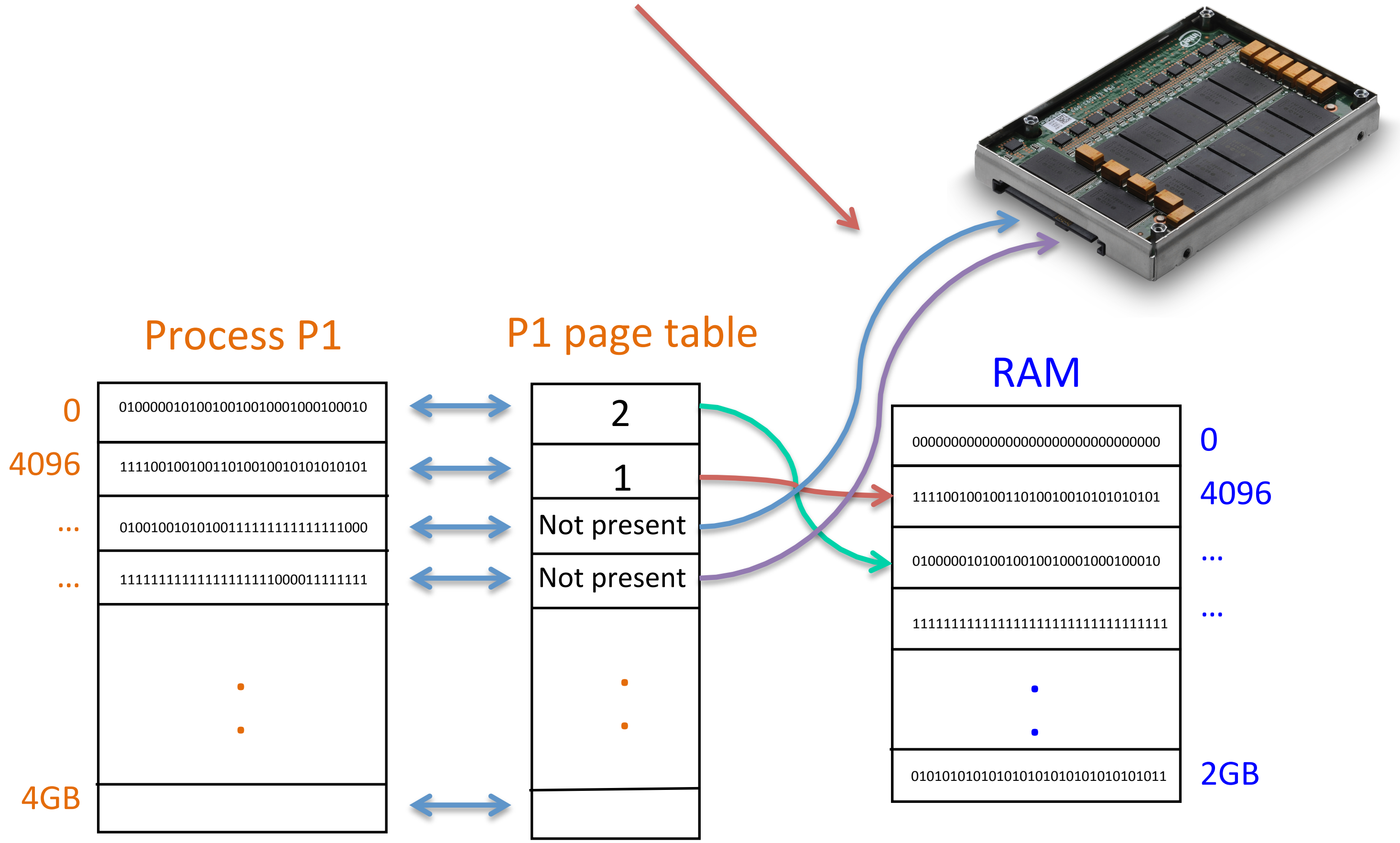


Virtual Memory with Swapping

SSDs substantially increase size of pipe

500GB/sec or so

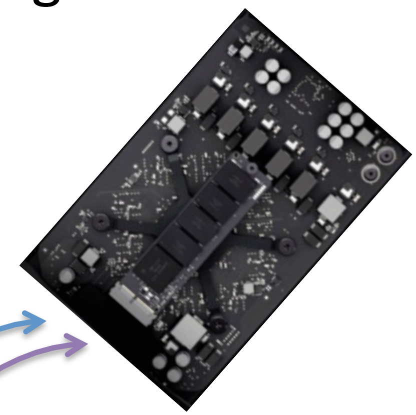
swap file on SSD

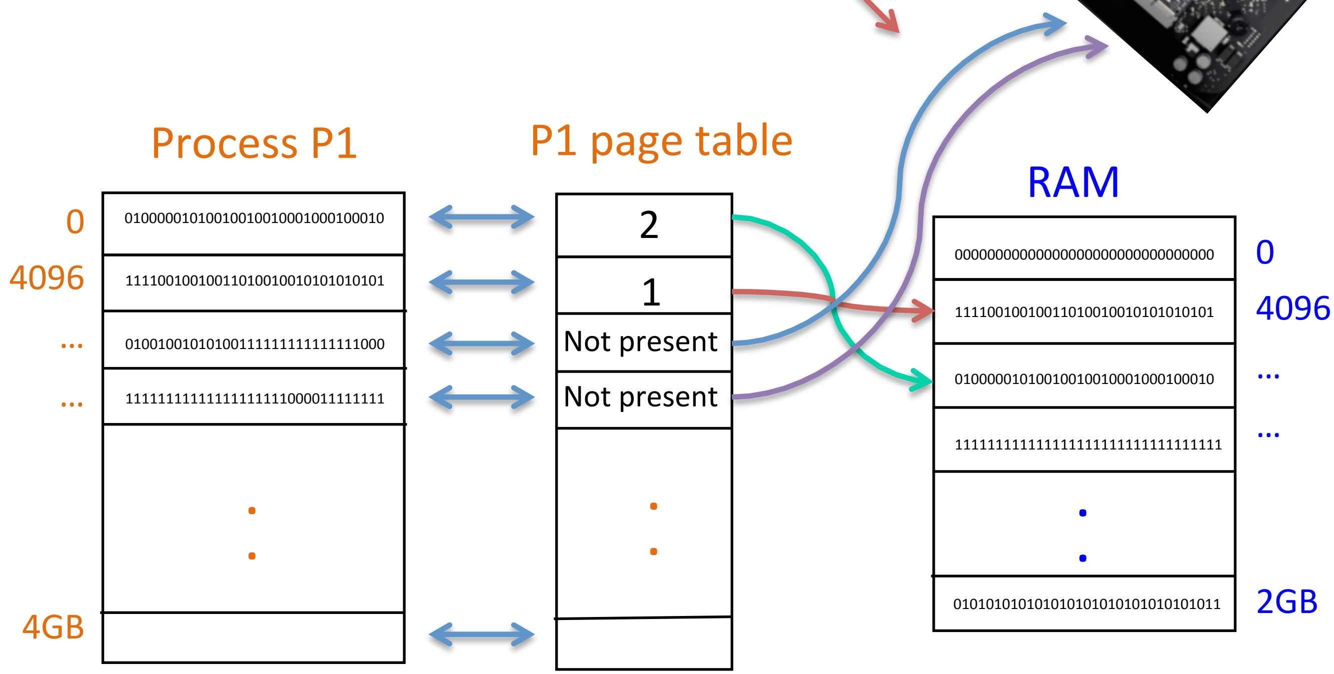


Virtual Memory with Swapping

up to 1.2GB/sec+ (e.g., new Mac Pro, newest iMacs)

swap file on PCIe flash storage





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?

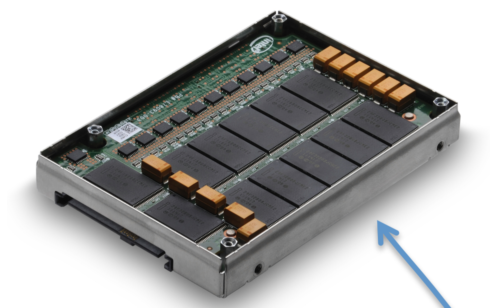
It's Worth It

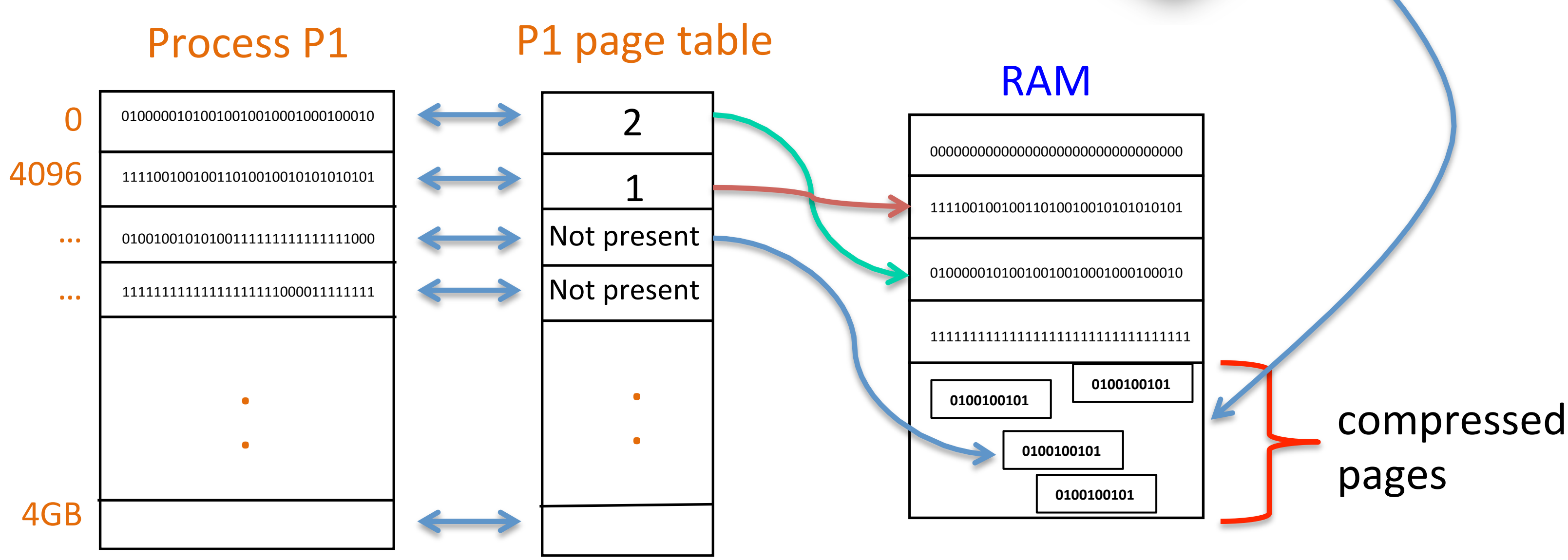
- Ultraportable laptops still RAM limited
  - e.g., current Macbook Air: Base 4GB, Max 8GB RAM
- Virtualization, even on laptops
- Malware analysts, VMWare Workstation, Fusion, etc.
- Swapping can cause serious performance issues
- New Mac Pro memory bandwidth: **60GB/sec**
- Max swap bandwidth: **1.2GB/sec**
- Modern CPUs compress and decompress very efficiently

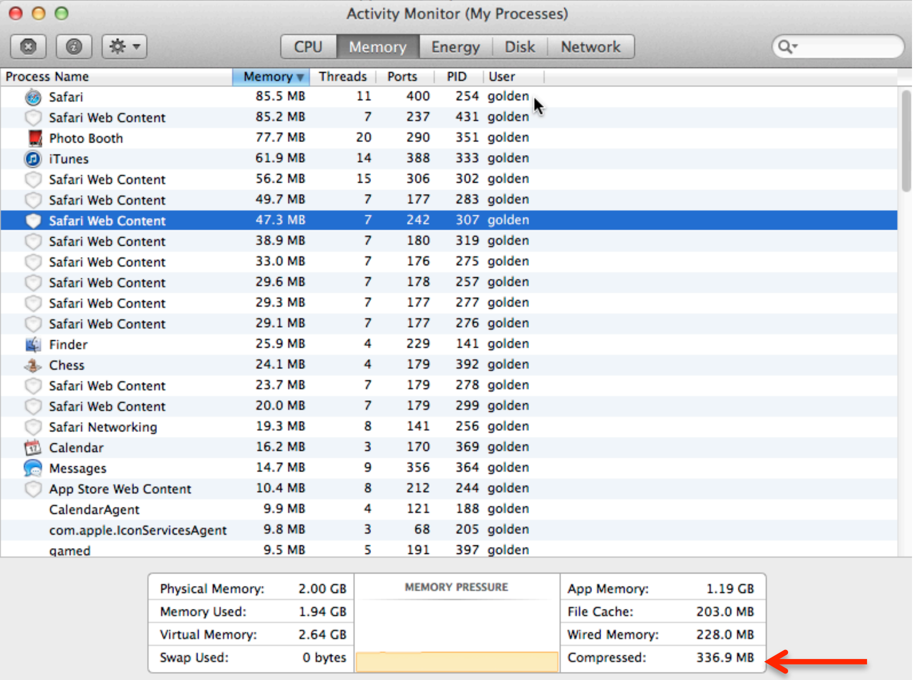
Virtual Memory with Compressed RAM

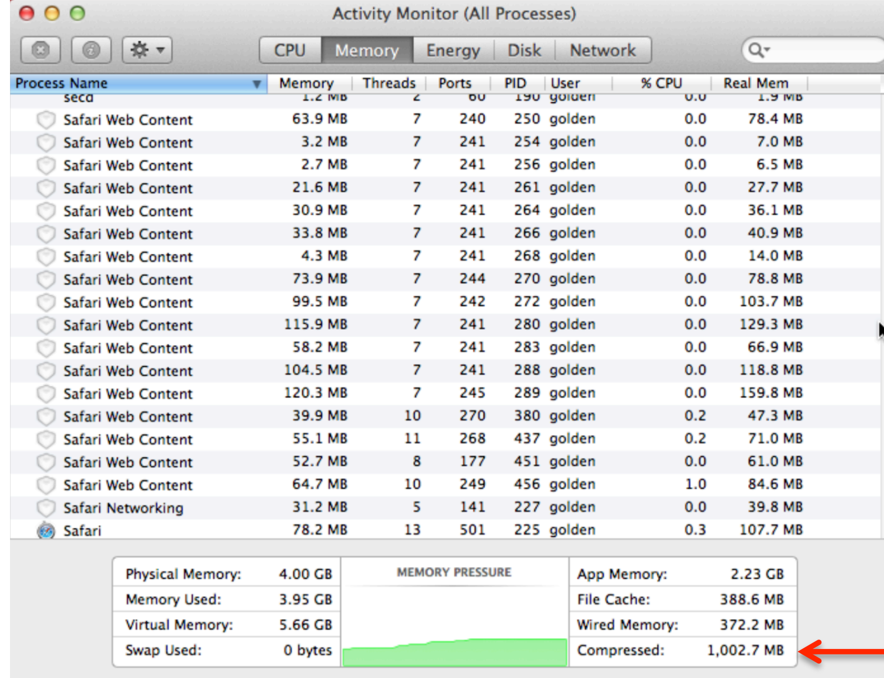
In Mac OS X Mavericks and Linux, set aside some RAM and **compress pages that would have been swapped due to memory pressure**

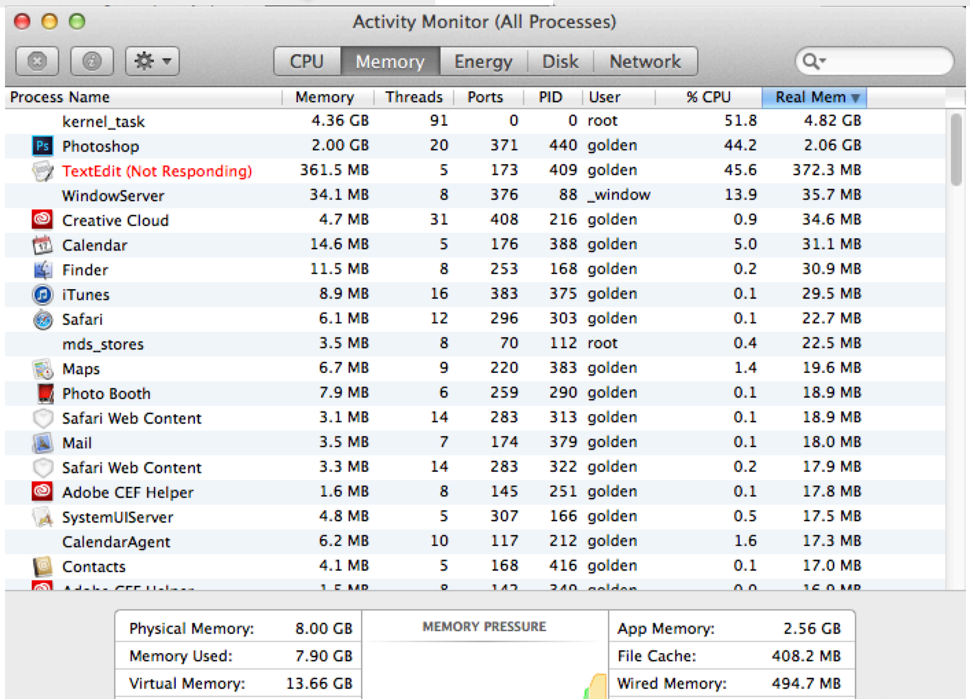
swap file on SSD






Activity Monitor (My Processes)
CPU Memory Energy Disk Network
Process Name Memory Threads Ports PID User
Safari 85.5 MB 11 400 254 golden
Safari Web Content 85.2 MB 7 237 431 golden
Photo Booth 77.7 MB 20 290 351 golden
iTunes 61.9 MB 14 388 333 golden
Safari Web Content 56.2 MB 15 306 302 golden
Safari Web Content 49.7 MB 7 177 283 golden
Safari Web Content 47.3 MB 7 242 307 golden
Safari Web Content 38.9 MB 7 180 319 golden
Safari Web Content 33.0 MB 7 176 275 golden
Safari Web Content 29.6 MB 7 178 257 golden
Safari Web Content 29.3 MB 7 177 277 golden
Safari Web Content 29.1 MB 7 177 276 golden
Finder 25.9 MB 4 229 141 golden
Chess 24.1 MB 4 179 392 golden
Safari Web Content 23.7 MB 7 179 278 golden
Safari Web Content 20.0 MB 7 179 299 golden
Safari Networking 19.3 MB 8 141 256 golden
Calendar 16.2 MB 3 170 369 golden
Messages 14.7 MB 9 356 364 golden
App Store Web Content 10.4 MB 8 212 244 golden
CalendarAgent 9.9 MB 4 121 188 golden
com.apple.IconServicesAgent 9.8 MB 3 68 205 golden
gamed 9.5 MB 5 191 397 golden
Physical Memory: 2.00 GB
Memory Used: 1.94 GB
Virtual Memory: 2.64 GB
Swap Used: 0 bytes
MEMORY PRESSURE
App Memory: 1.19 GB
File Cache: 203.0 MB
Wired Memory: 228.0 MB
Compressed: 336.9 MB


Activity Monitor (All Processes)
CPU Memory Energy Disk Network
Process Name Memory Threads Ports PID User % CPU Real Mem
Safari Web Content 63.9 MB 7 240 250 golden 0.0 78.4 MB
Safari Web Content 3.2 MB 7 241 254 golden 0.0 7.0 MB
Safari Web Content 2.7 MB 7 241 256 golden 0.0 6.5 MB
Safari Web Content 21.6 MB 7 241 261 golden 0.0 27.7 MB
Safari Web Content 30.9 MB 7 241 264 golden 0.0 36.1 MB
Safari Web Content 33.8 MB 7 241 266 golden 0.0 40.9 MB
Safari Web Content 4.3 MB 7 241 268 golden 0.0 14.0 MB
Safari Web Content 73.9 MB 7 244 270 golden 0.0 78.8 MB
Safari Web Content 99.5 MB 7 242 272 golden 0.0 103.7 MB
Safari Web Content 115.9 MB 7 241 280 golden 0.0 129.3 MB
Safari Web Content 58.2 MB 7 241 283 golden 0.0 66.9 MB
Safari Web Content 104.5 MB 7 241 288 golden 0.0 118.8 MB
Safari Web Content 120.3 MB 7 245 289 golden 0.0 159.8 MB
Safari Web Content 39.9 MB 10 270 380 golden 0.2 47.3 MB
Safari Web Content 55.1 MB 11 268 437 golden 0.2 71.0 MB
Safari Web Content 52.7 MB 8 177 451 golden 0.0 61.0 MB
Safari Web Content 64.7 MB 10 249 456 golden 1.0 84.6 MB
Safari Networking 31.2 MB 5 141 227 golden 0.0 39.8 MB
Safari 78.2 MB 13 501 225 golden 0.3 107.7 MB
Physical Memory: 4.00 GB
Memory Used: 3.95 GB
Virtual Memory: 5.66 GB
Swap Used: 0 bytes
MEMORY PRESSURE
App Memory: 2.23 GB
File Cache: 388.6 MB
Wired Memory: 372.2 MB
Compressed: 1,002.7 MB


Activity Monitor (All Processes)
CPU Memory Energy Disk Network
Process Name Memory Threads Ports PID User % CPU Real Mem
kernel_task 4.36 GB 91 0 0 root 51.8 4.82 GB
Photoshop 2.00 GB 20 371 440 golden 44.2 2.06 GB
TextEdit (Not Responding) 361.5 MB 5 173 409 golden 45.6 372.3 MB
WindowServer 34.1 MB 8 376 88 _window 13.9 35.7 MB
Creative Cloud 4.7 MB 31 408 216 golden 0.9 34.6 MB
Calendar 14.6 MB 5 176 388 golden 5.0 31.1 MB
Finder 11.5 MB 8 253 168 golden 0.2 30.9 MB
iTunes 8.9 MB 16 383 375 golden 0.1 29.5 MB
Safari 6.1 MB 12 296 303 golden 0.1 22.7 MB
mds_stores 3.5 MB 8 70 112 root 0.4 22.5 MB
Maps 6.7 MB 9 220 383 golden 1.4 19.6 MB
Photo Booth 7.9 MB 6 259 290 golden 0.1 18.9 MB
Safari Web Content 3.1 MB 14 283 313 golden 0.1 18.9 MB
Mail 3.5 MB 7 174 379 golden 0.1 18.0 MB
Safari Web Content 3.3 MB 14 283 322 golden 0.2 17.9 MB
Adobe CEF Helper 1.6 MB 8 145 251 golden 0.1 17.8 MB
SystemUIServer 4.8 MB 5 307 166 golden 0.5 17.5 MB
CalendarAgent 6.2 MB 10 117 212 golden 1.6 17.3 MB
Contacts 4.1 MB 5 168 416 golden 0.1 17.0 MB
Physical Memory: 8.00 GB
Memory Used: 7.90 GB
Virtual Memory: 13.66 GB
Swap Used: 674.0 MB
MEMORY PRESSURE
App Memory: 2.56 GB
File Cache: 408.2 MB
Wired Memory: 494.7 MB
Compressed: 4.46 GB

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

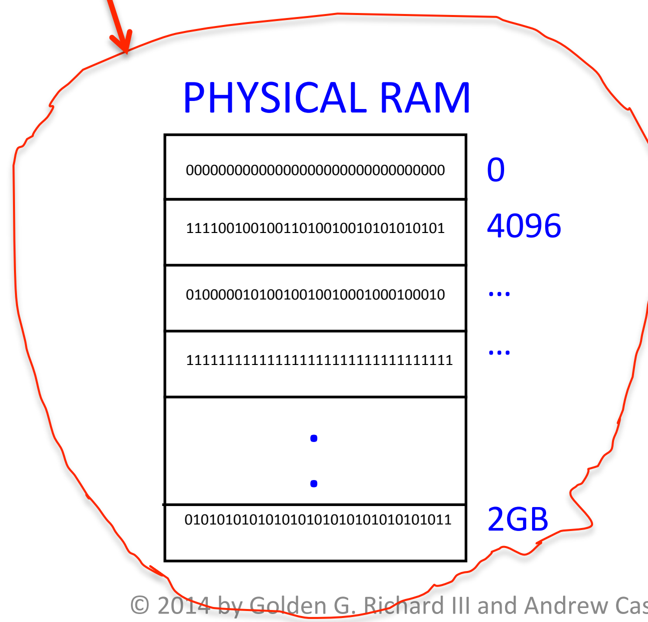
Compressed RAM: Forensic Impact

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—compressed data opaque or ignored

Our tools enable analysis of compressed RAM



Mavericks VM: Too Much Detail

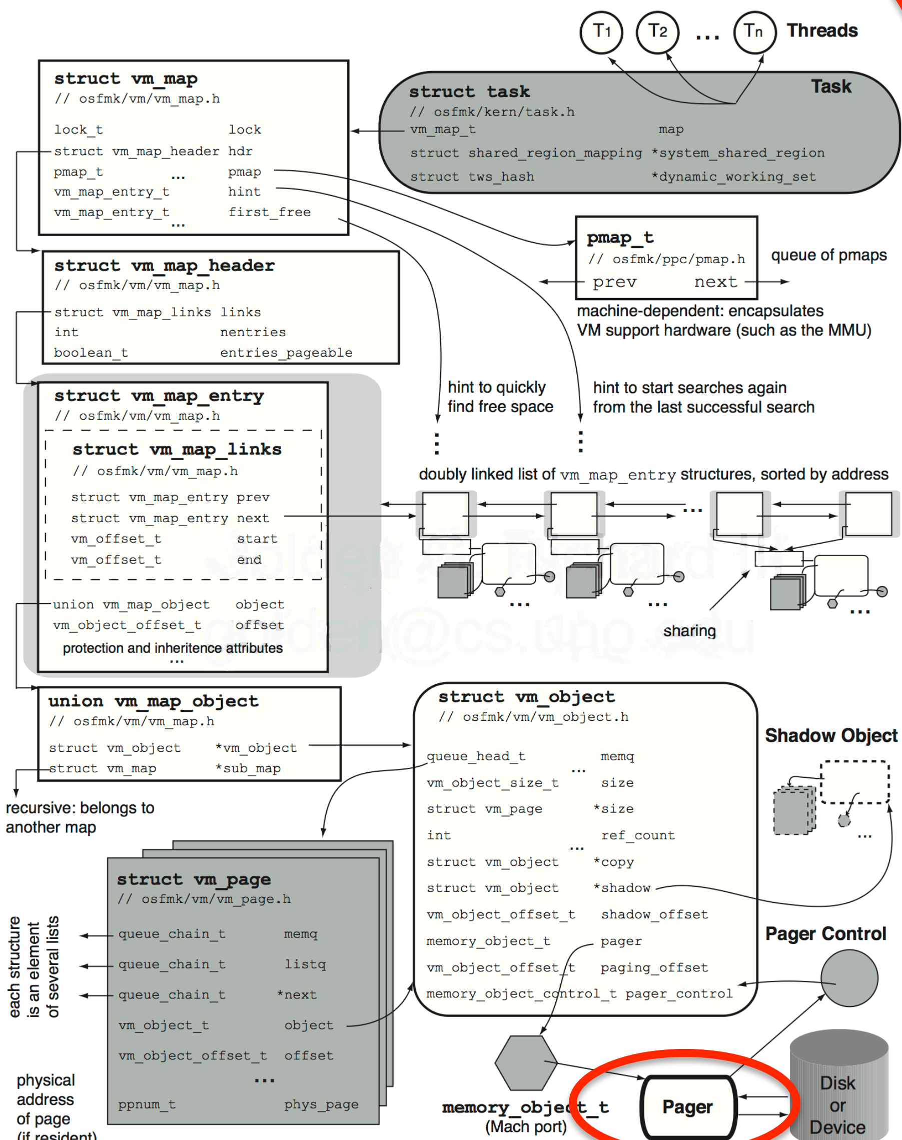
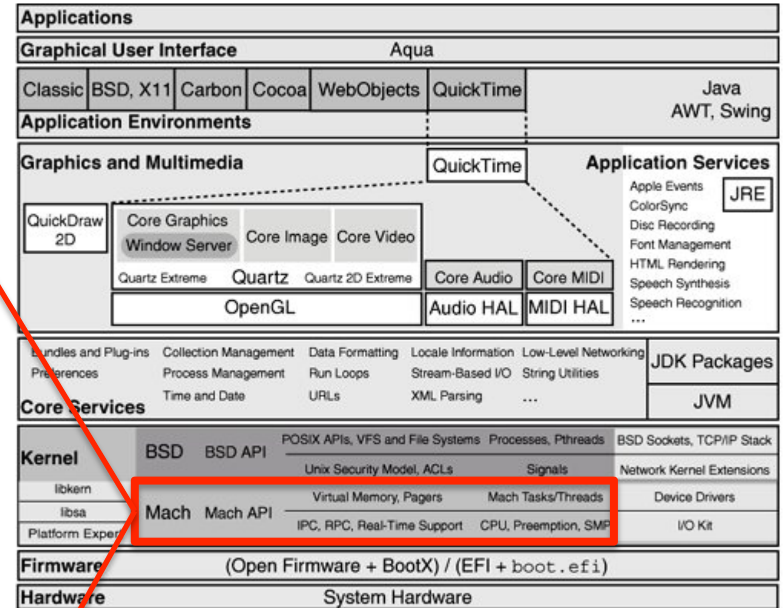


FIGURE 8–6 Details of the Mac OS X Mach VM architecture



Mac OS X Mavericks Implementation

- Implemented 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

Mavericks 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

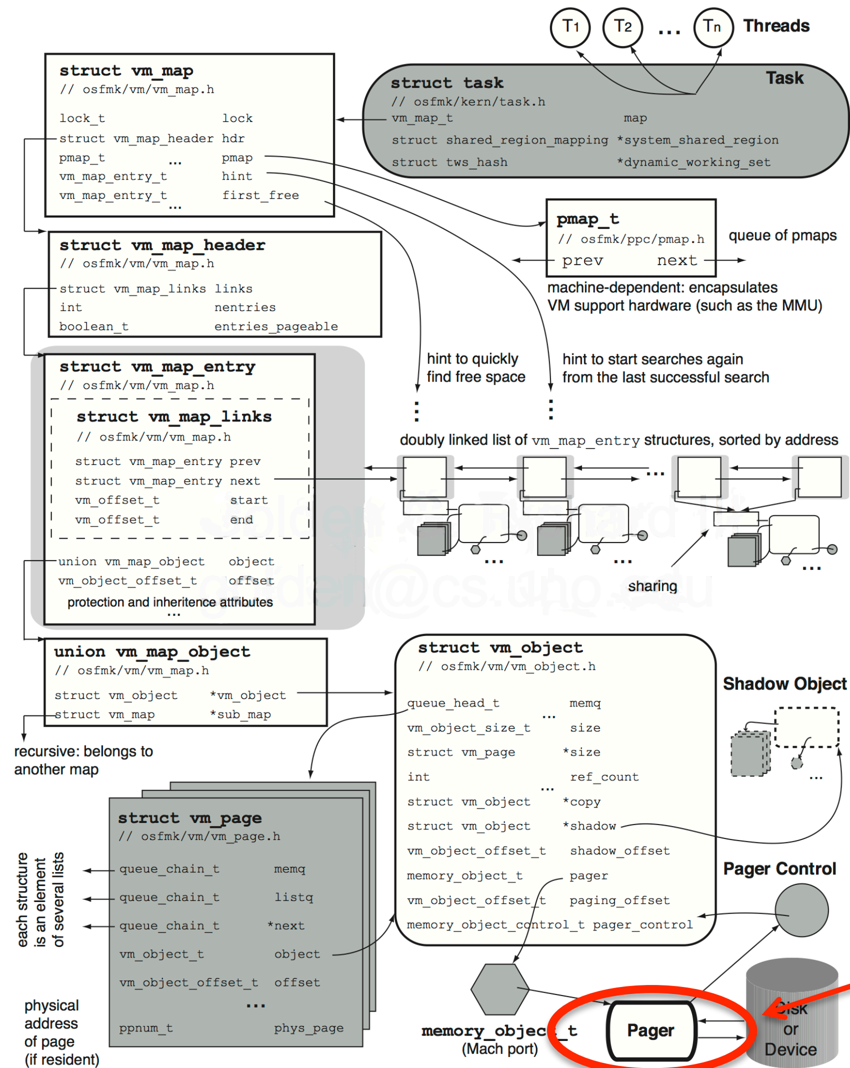


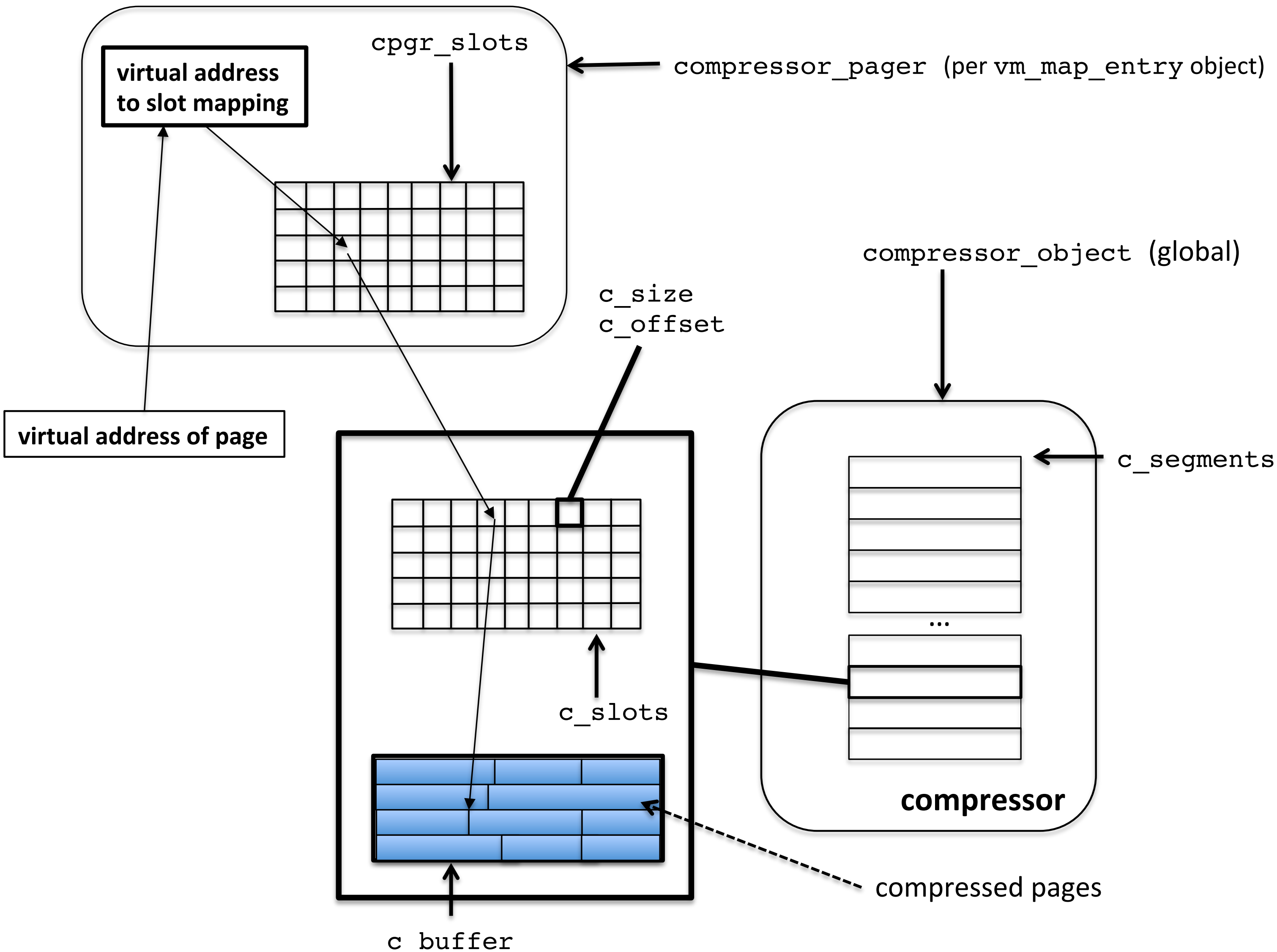
FIGURE 8–6 Details of the Mac OS X Mach VM architecture

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

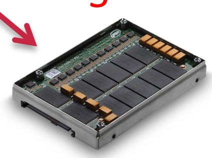
```
int swap_writepage(struct page *page, ... )
{
    int ret = 0;
    if (try_to_free_swap(page)) {
        unlock_page(page);
        goto out;
    }
    if (frontswap_store(page) == 0) {
        set_page_writeback(page);
        unlock_page(page);
        end_page_writeback(page);
        goto out;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



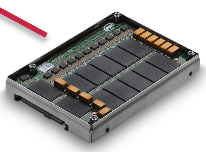
/mm/page_io.c

```
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;
}
```

e.g., zswap

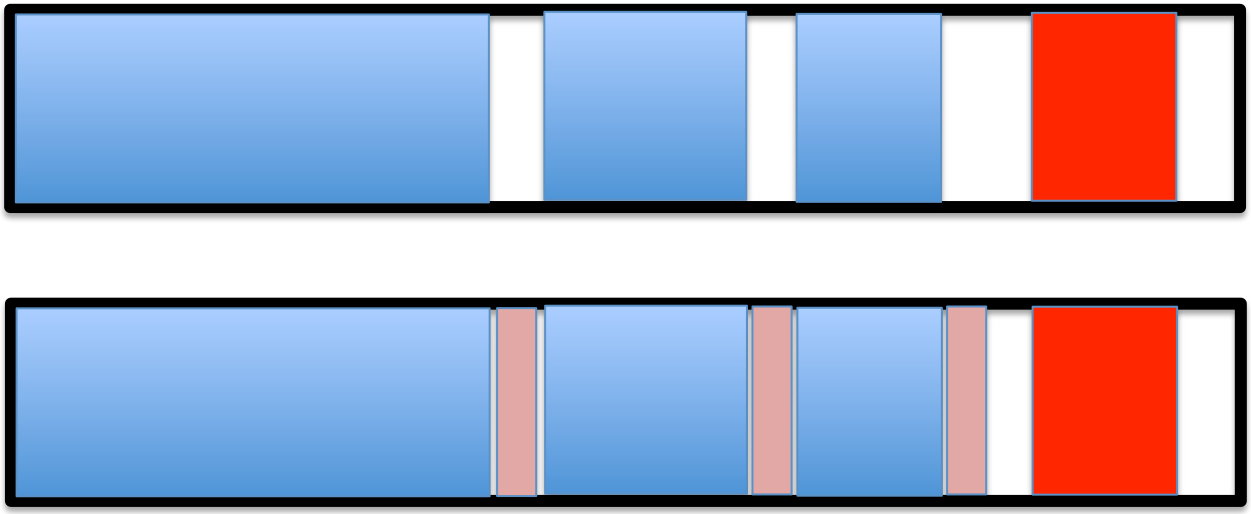
e.g., zram,
it's just a stranger block device

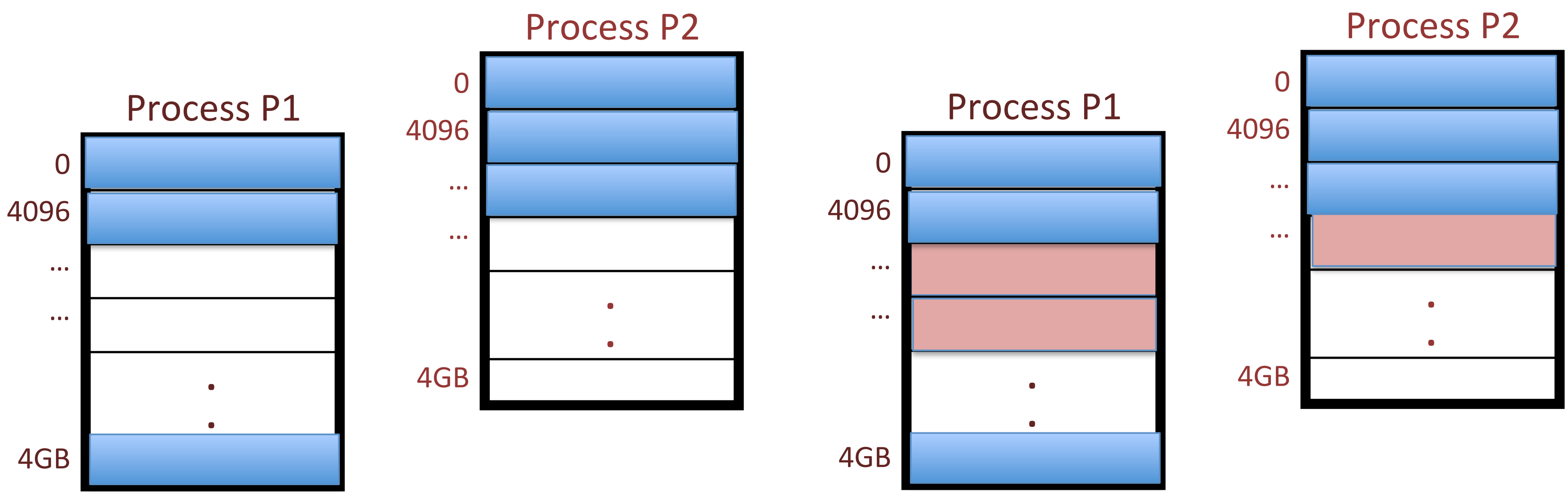
Page eventually read from disk



Current Reality + Goal

Physical memory dump:





Volatility

- Most popular memory analysis framework
- Open source
- Portable, written in Python 😐
- Supports analysis of Windows, Linux, Mac
- Plugins add new functionality
- Our contribution: New plugins to support compressed RAM analysis

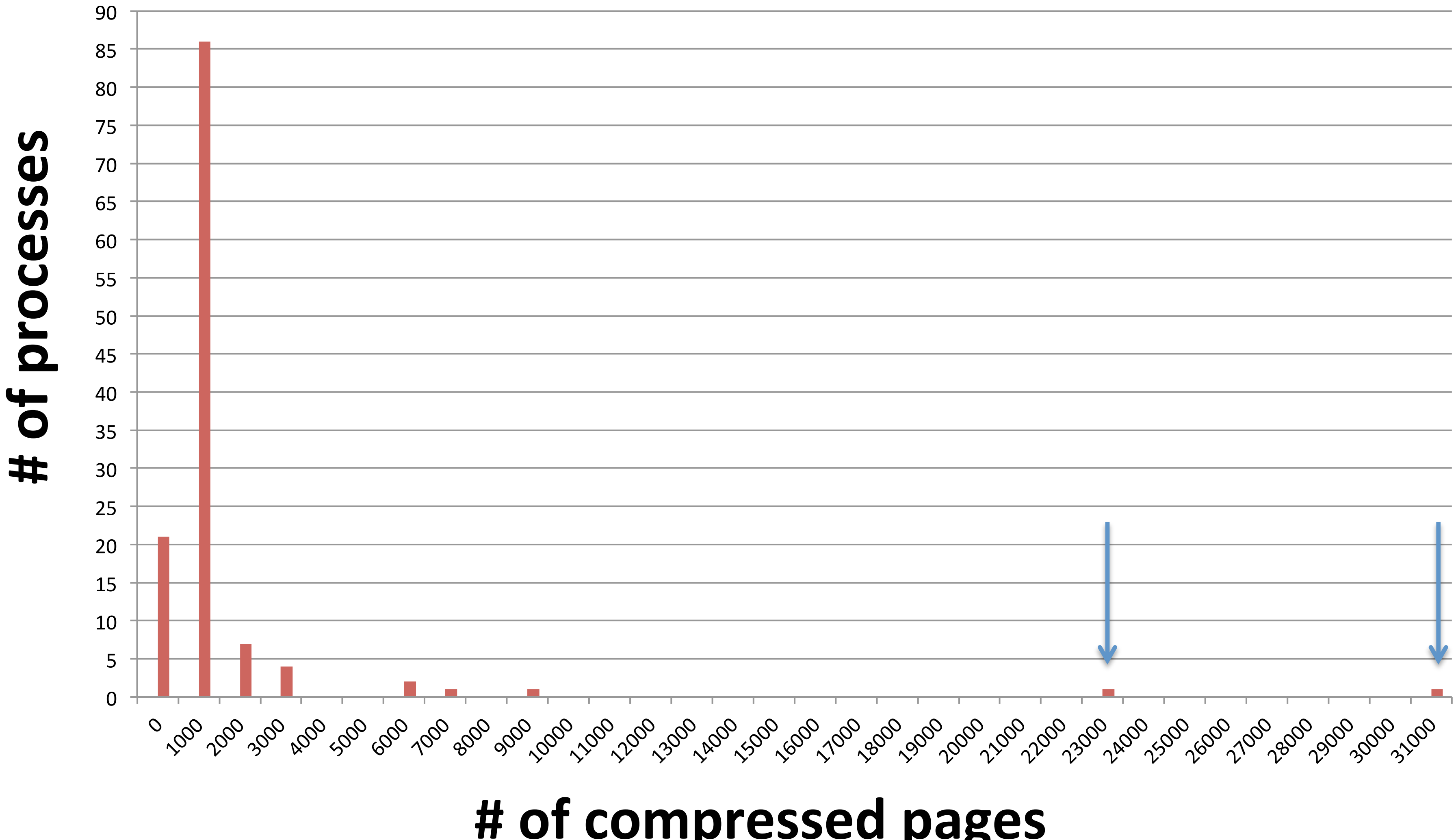
New Plugins for Volatility

- **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 decompression algorithms
- Required detailed analysis of new compressor pager
- But not entire page fault → decompression path

New / 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
- Involves deeper analysis of OS VM systems
- Much more complex (not your problem, we're not complaining)
- We ♥ OS internals

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  @@@@  .6@@@P'
    @@@@@@@@@@@  @@@@   @@@@ ` j@@@P*"*+Y7  @@@@ .6@
   `_  @@@@     @@@@   @@@@  .@@@7 .   ` @@@@.6@@@P  ` !!!!; .
    . @@@@  ' @@@@   @@@@  :@@@! !:   @@@@7@@@K  `; !!!! ' `'
      @@@@  . @@@@   @@@@  `%@@@.    . @@@@`7@@@b . !!!! :
    ! @@@@    @@@@   @@@@  \@@@$+,,+4b  @@@@ `7@@@b  !!!!
      @@@@  : @@@@   @@@@   `7%S@@hX!P' @@@@  `7@@@b !!!! .
    : """""    """""   """""  :.  `^"^`   """""   `""""""" """"
     `- .  .     _._     `            _._     _  . -
         , ` ,glllllllllg,    `-: '   .~ . . . ~.  `
           ,jllllllllllllllp,  .!'  .+. . . . . . .+. `.
         ` jllllllllllllllllllll  `  +. . . . . . . . .+  .
       .  jlllllllllllllllllllllll   . . . . . . . . . . .
          .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: 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

WKdm Benchmarks

- Kaplan's original C version
- Apple's assembler version
- My Python implementation (needed for integrating compressed RAM analysis in Volatility)
- A version (for fun, and a reality check for Python) in go, that I wrote in a few hours

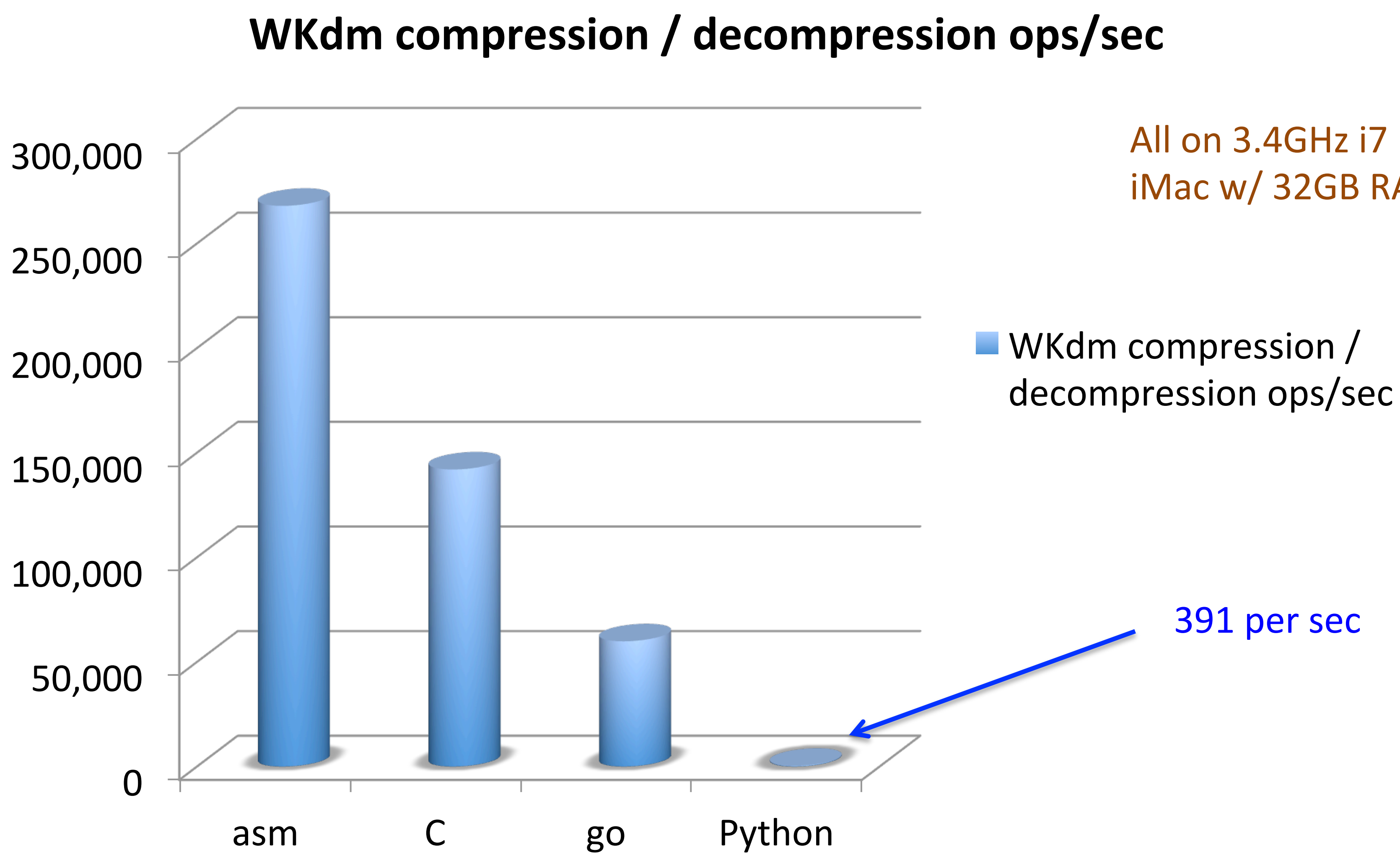
```
L_nonpartial:
      jl          L_ZERO_TAG
      cmpb    $2, -1(%rsi)
      je          L_MISS_TAG

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++

L_

L_
```

```
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
```

```
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++;
}
```

```
for next_tag < tags_area_end {
     tag = (tempTagsArray[next_tag / 4] & SINGLE_BYTE_MASKS[next_tag % 4]) >> uint32((((next_tag) % 4) * 8))
nt32((((next_qp) % 4) * 8))
nt32((((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 (Sorry, Python)



All on 3.4GHz i7
iMac w/ 32GB RAM

Compressed RAM: Conclusions

- Significantly reduces swapping to disk
- Used very aggressively in Mac OS X Mavericks
- Soothes RAM ←→ swap file consistency problems for memory analysis
- New plugins make more evidence available
- Essential for "complete picture" in memory analysis
- Our plugins integrated into Volatility
  - Check github (released in early September)
  - https://github.com/volatilityfoundation/volatility
  - Transparent decompression support in Volatility 3.0

Questions/Comments?

- Golden G. Richard III
  - http://www.cs.uno.edu/~golden/
  - golden@cs.uno.edu
  - @nolaforensix
- Andrew Case
  - http://www.dfir.org
  - andrew@dfir.org
  - @attrc