Virtual Machines

David E. Culler

CS162 – Operating Systems and Systems Programming

http://cs162.eecs.berkeley.edu/

Lecture 15

October 22, 2019

Read 3Easy App B
A&D 10.2

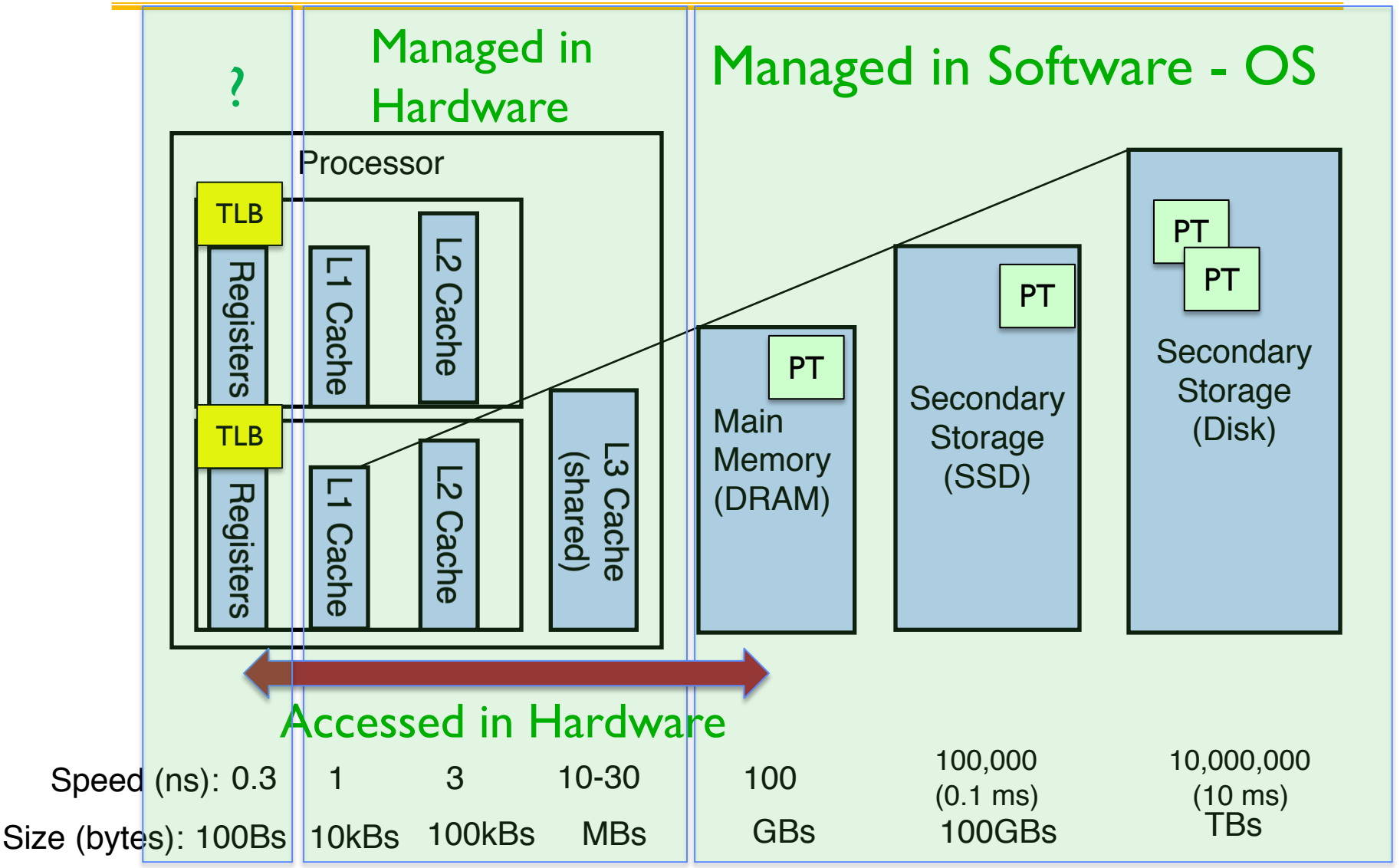
Mark your calendar

- **November 14, Guest Lecture by Dr. Eric Brewer, Google VP of Infrastructure**
  - **Former UCB Faculty**
  - **Founded Inktomi – first fast massive search engine**
    - » **inktomi.Berkeley.edu => inktomi.com**
  - **Responsible for re-architecting Google's internals**
  - **Including Kubernetes.**
- **Let's fill the room**
  - **Great chance to engage with one of the field's leading thinkers**

Paging Review

- Page Table provides a virtual address space for each process
  - Serves as the basis of protection
- Allows creation of the illusion of a very large memory dedicated to each
  - TLB translates every reference form Virtual to Physical
    - » Fast, High associativity, Small, OS manages consistency
- Also basis for sharing (MMAP), fast fork (copy—on-write), fast exec (load by swap)
- Key policy issue is page replacement policy
  - Approximation to LRU
- Clock algorithm (or N-th chance)
  - Scan through physical frames, replace first one that has not been accesses in last N passes (clock: N==1)
  - Reverse map: frame => PTE, accessed bit in PTE

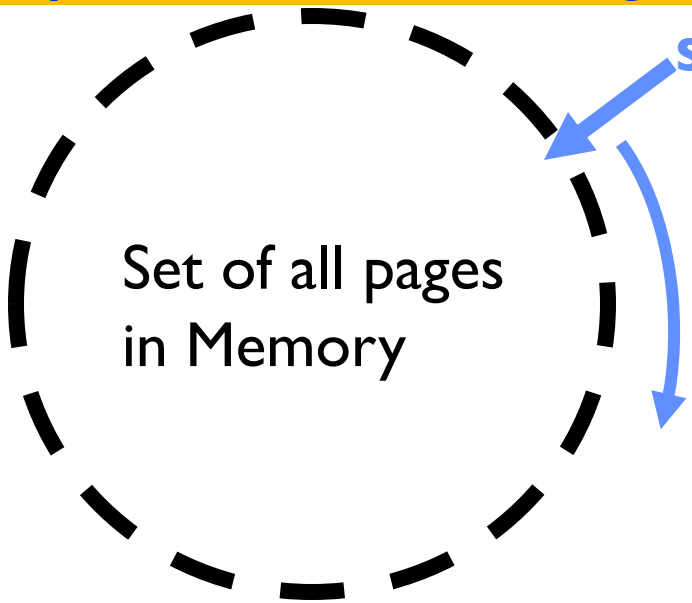
Management of the Memory Hierarchy



Demand Paging Mechanisms

- PTE makes demand paging implementable
  - Valid ⇒ Page in memory, PTE points at physical page
  - Not Valid ⇒ Page not in memory; use info in PTE to find it on disk when necessary
- Suppose user references page with invalid PTE?
  - Memory Management Unit (MMU) traps to OS
    - » Resulting trap is a "Page Fault"
  - What does OS do on a Page Fault?:
    - » Choose an old page to replace
    - » If old page modified ("D=1"), write contents back to disk
    - » Change its PTE and any cached TLB to be invalid
    - » Load new page into memory from disk
    - » Update page table entry, invalidate TLB for new entry
    - » Continue thread from original faulting location
  - TLB for new page will be loaded when thread continued!
  - While pulling pages off disk for one process, OS runs another process from ready queue
    - » Suspended process sits on wait queue

Recall: Clock Algorithm (aka Not Recently Used)



Single Clock Hand:

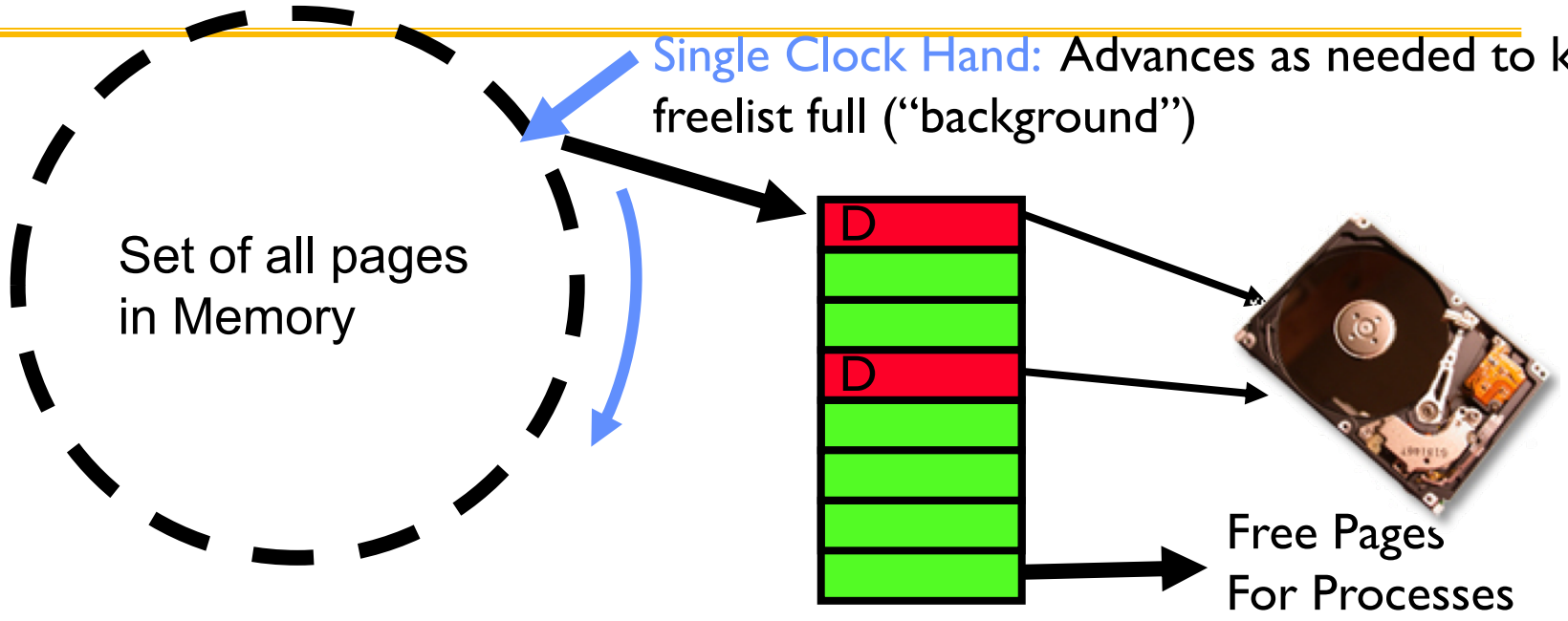
Advances through memory frames

Check for pages not used recently

Best candidates for eviction

- **Which bits of a PTE entry are useful to us?**
  - **Use:** Set when page is referenced; cleared by clock algorithm
  - **Modified:** set when page is modified, cleared when page written to disk
  - **Valid:** ok for program to reference this page
  - **Read-only:** ok for program to read page, but not modify
    - » For example for catching modifications to code pages!
- **Clock Algorithm: pages arranged in a ring**
  - On page fault (or background reaper):
    - » Advance clock hand (not real time)
    - » Check use bit: 1→used recently; clear and leave alone
0→selected candidate for replacement
  - Crude partitioning of pages into two groups: young and old

Recall: Clock, Free List, 2nd Chance



- **Keep set of free pages ready for use in demand paging**
  - **Freelist filled in background by Clock algorithm or other technique ("Pageout demon")**
  - **Dirty pages start copying back to disk when enter list**
- **Like VAX second-chance list**
  - **If page needed before reused, just return to active set**
- **Advantage: faster for page fault**
  - **Can always use page (or pages) immediately on fault**

Reverse Page Mapping (Sometimes called "Coremap")

- Physical page frames often shared by many different address spaces/page tables
  - All children forked from given process
  - Shared memory pages between processes
- Whatever reverse mapping mechanism that is in place must be fast
  - Must hunt down all page tables pointing at given page frame when freeing a page
  - Must hunt down all PTEs when seeing if pages "active"
- Implementation options:
  - For every page descriptor, keep linked list of page table entries that point to it
    - » Management nightmare – expensive
  - Linux: Object-based reverse mapping
    - » Link together memory region descriptors instead (much coarser granularity)

Allocation of Page Frames (Memory Pages)

- How do we allocate memory among different processes?
  - Does every process get the same fraction of memory? Different fractions?
  - Should we completely swap some processes out of memory?
- Each process needs *minimum* number of pages
  - Want to make sure that all processes that are loaded into memory can make forward progress
  - Example: IBM 370 – 6 pages to handle SS MOVE instruction:
    - » instruction is 6 bytes, might span 2 pages
    - » 2 pages to handle *from*
    - » 2 pages to handle *to*
- Possible Replacement Scopes:
  - Global replacement – process selects replacement frame from set of all frames; one process can take a frame from another
  - Local replacement – each process selects from only its own set of allocated frames

Fixed/Priority Allocation

- **Equal allocation (Fixed Scheme):**
  - Every process gets same amount of memory
  - Example: 100 frames, 5 processes → process gets 20 frames

- **Proportional allocation (Fixed Scheme)**
  - Allocate according to the size of process
  - Computation proceeds as follows:

    s_i = size of process p_i and $S = \sum s_i$

    m = total number of physical frames in the system

    a_i = (allocation for p_i) $= \frac{s_i}{S} \times m$

- **Priority Allocation:**
  - Proportional scheme using priorities rather than size
    - » Same type of computation as previous scheme
  - Possible behavior: If process p_i generates a page fault, select for replacement a frame from a process with lower priority number

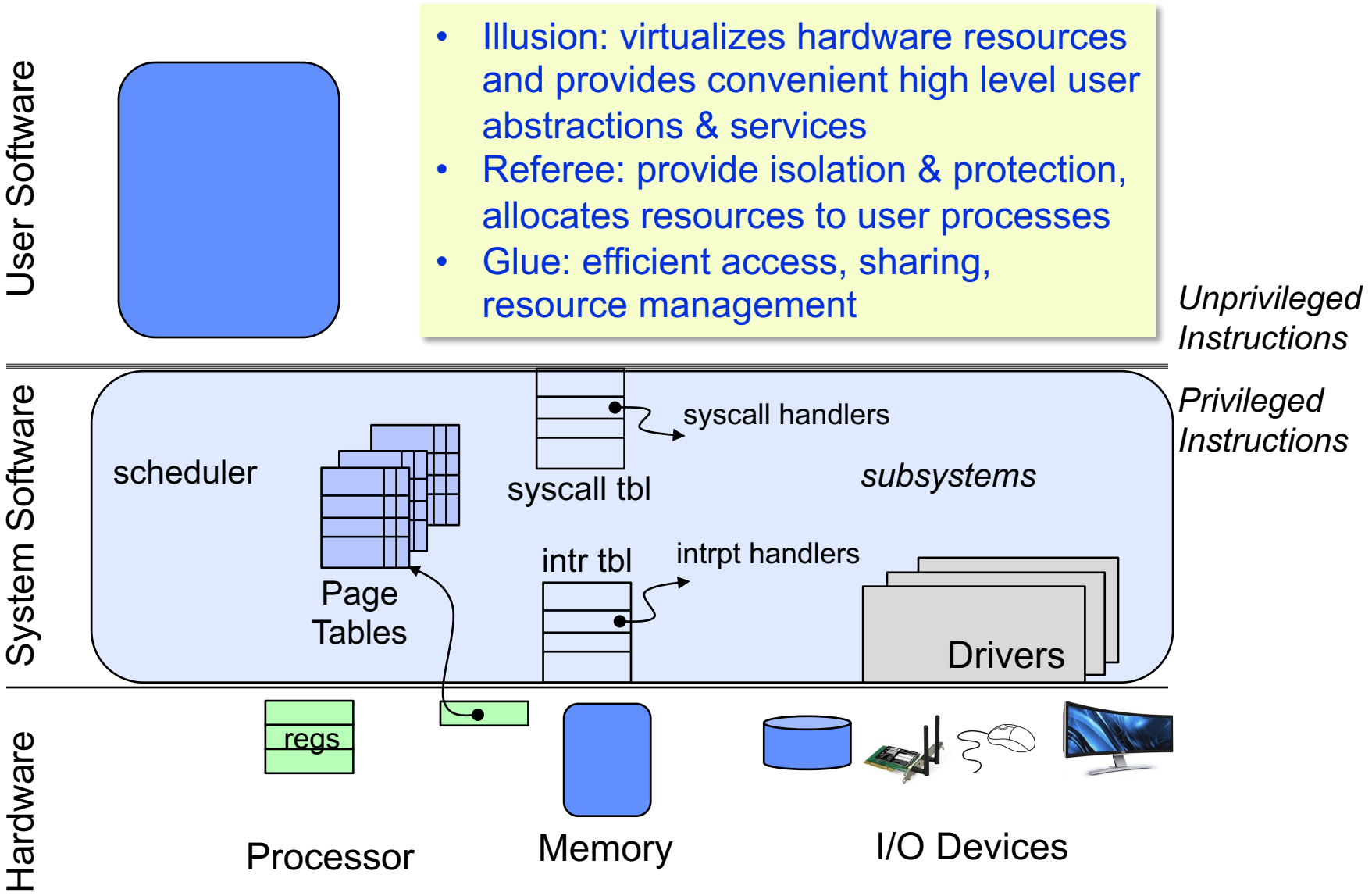
- **Perhaps we should use an adaptive scheme instead???**
  - What if some application just needs more memory?

What about Compulsory Misses?

- **Recall that compulsory misses are misses that occur the first time that a page is seen**
  - **Pages that are touched for the first time**
  - **Pages that are touched after process is swapped out/swapped back in**
- **Clustering:**
  - **On a page-fault, bring in multiple pages "around" the faulting page**
  - **Since efficiency of disk reads increases with sequential reads, makes sense to read several sequential pages**
- **Working Set Tracking:**
  - **Use algorithm to try to track working set of application**
  - **When swapping process back in, swap in working set**

On to Virtual Machines …

OS software supports user level software



- Illusion: virtualizes hardware resources and provides convenient high level user abstractions & services
- Referee: provide isolation & protection, allocates resources to user processes
- Glue: efficient access, sharing, resource management

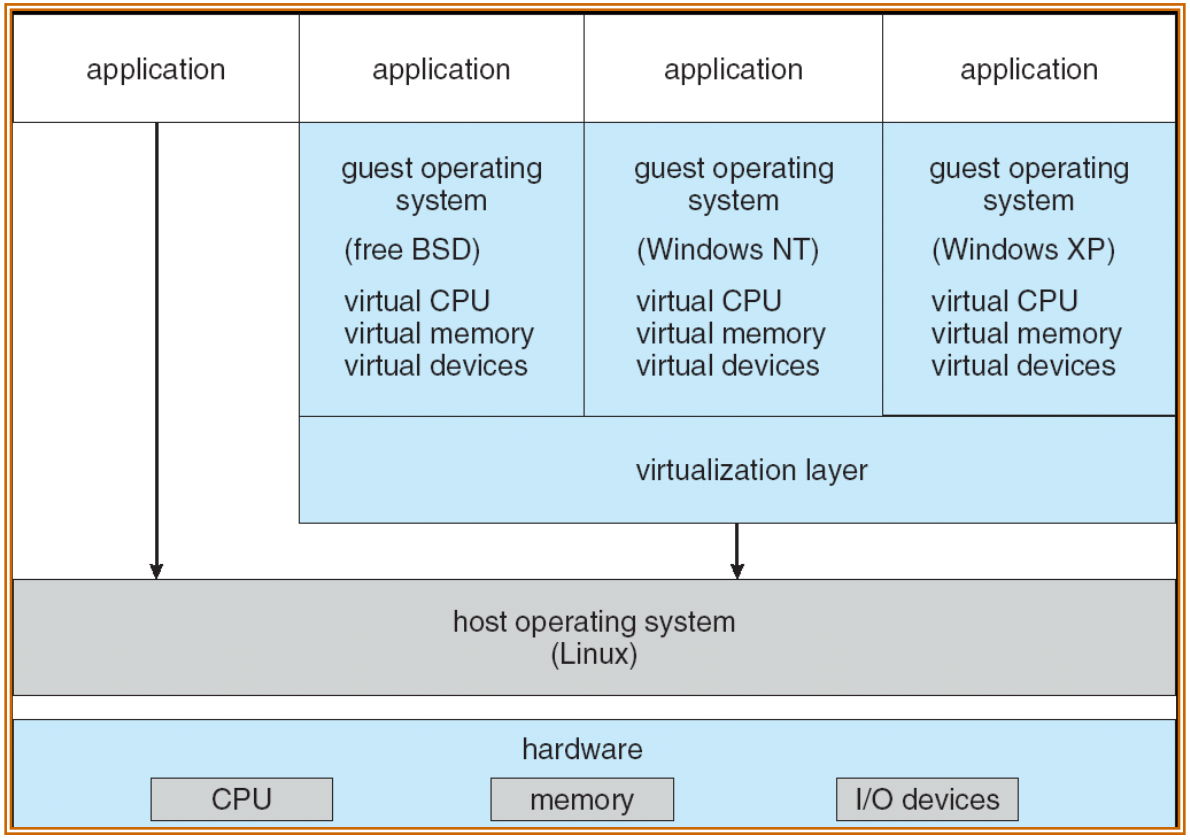
Today: turn system / user inside out

Recall (L1) Virtual Machines

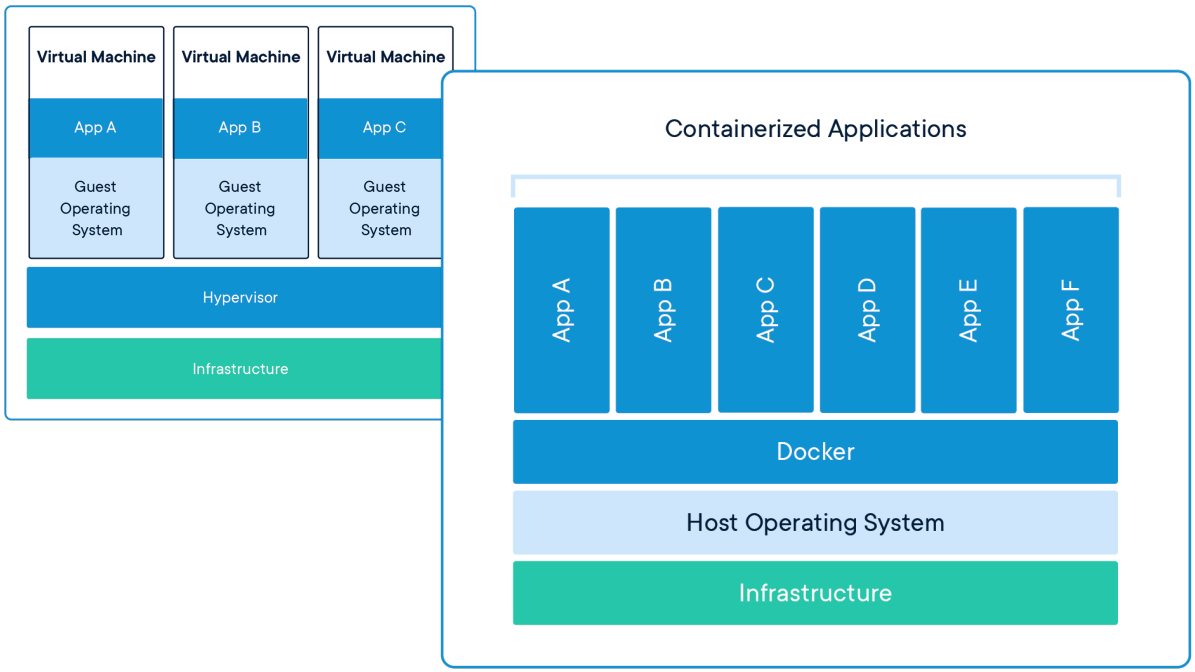
- Virtualize every detail of a hardware configuration so perfectly that you can run an operating system (and many applications) on top of it.
  - VMWare Fusion, Virtual box, Parallels Desktop, Xen, Vagrant
- Provides isolation
- Complete insulation from change
- The norm in the Cloud (server consolidation)
- Long history (60's in IBM OS development)
- All our work will take place INSIDE a VM
  - Vagrant (new image just for you)

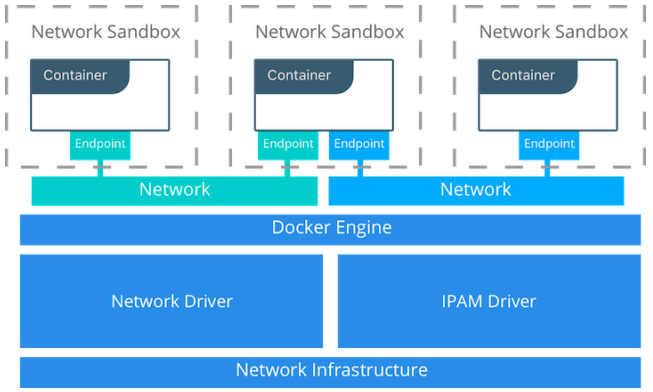
System Virtual Machines: Layers of OSs

- **Useful for OS development**
  - **When OS crashes, restricted to one VM**
  - **Can aid testing programs on other OSs**



Containers *virtualize the OS*



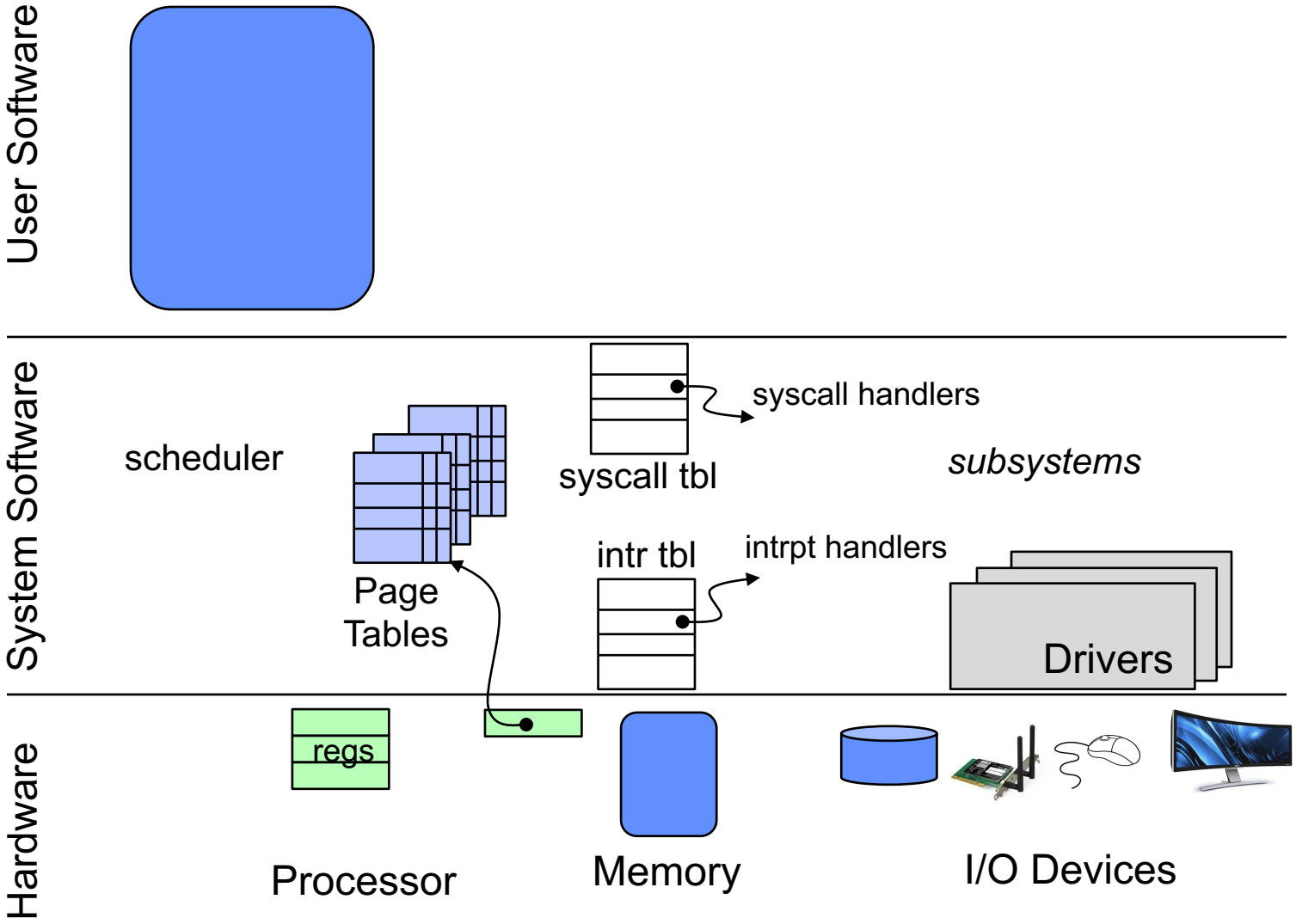


- **Roots in OS developments to provide protected systems abstraction, not just application abstraction**
  - **User-level file system (route syscalls to user process)**
  - **Cgroups – predictable, bounded resources (CPU, Mem, BW)**

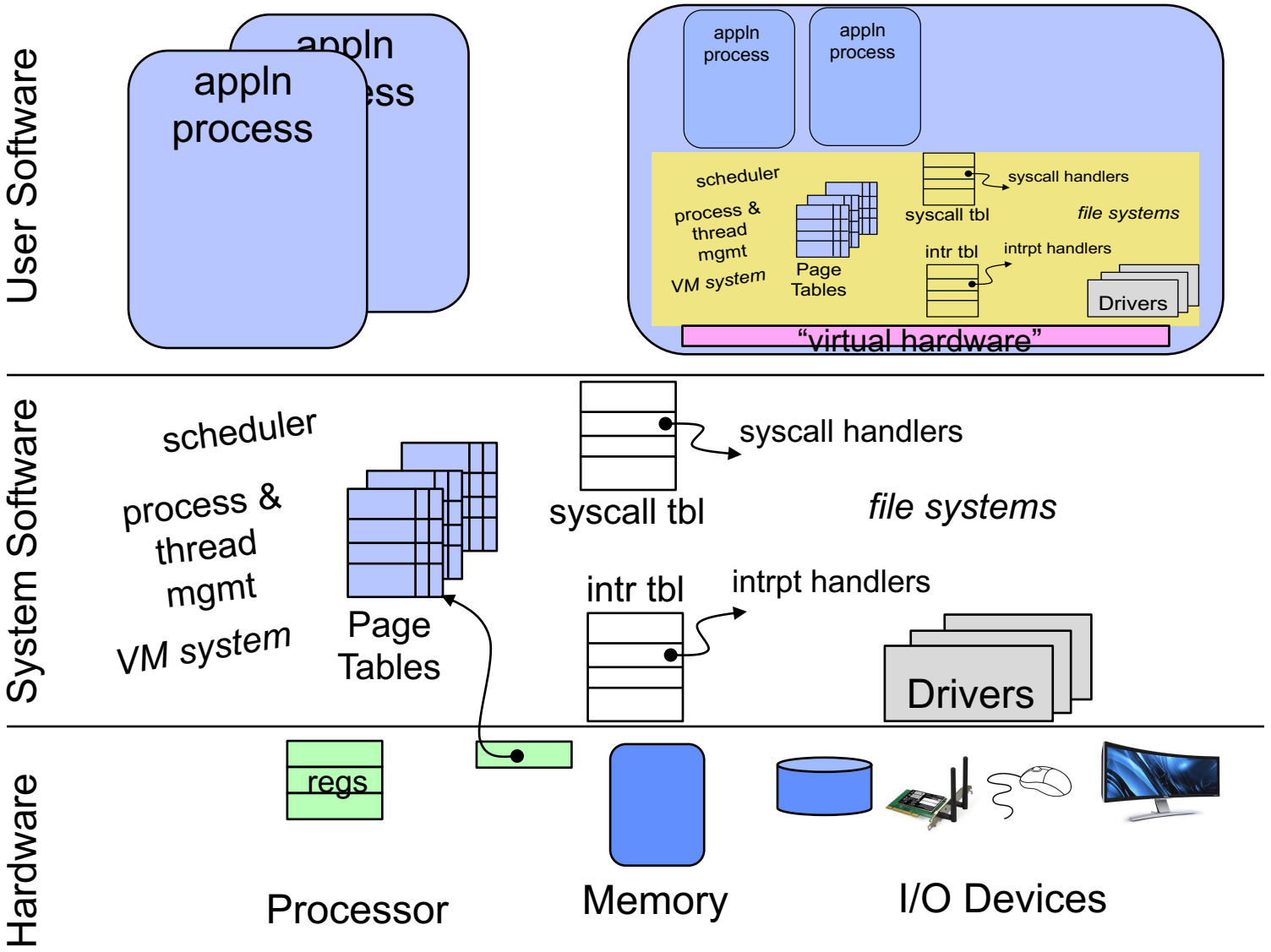
Motivation

- An "Application" is no longer a stand-alone executable or composition of vendor-supplied software, it is often a complex *platform* utilizing several deep software stacks, many processes, shared libraries and services, …
  - All of which evolve fairly rapidly
- To stand up a viable instance requires "all of it", so need to wrap up the "entire stack"
  - The OS and all its user-level daemons, the software installed in its file system
  - The hardware itself
- Want to isolate this entire ensemble from other ensembles, which may be on the same physical machine (server consolidation)
- Want to be able to allocate resources predictably

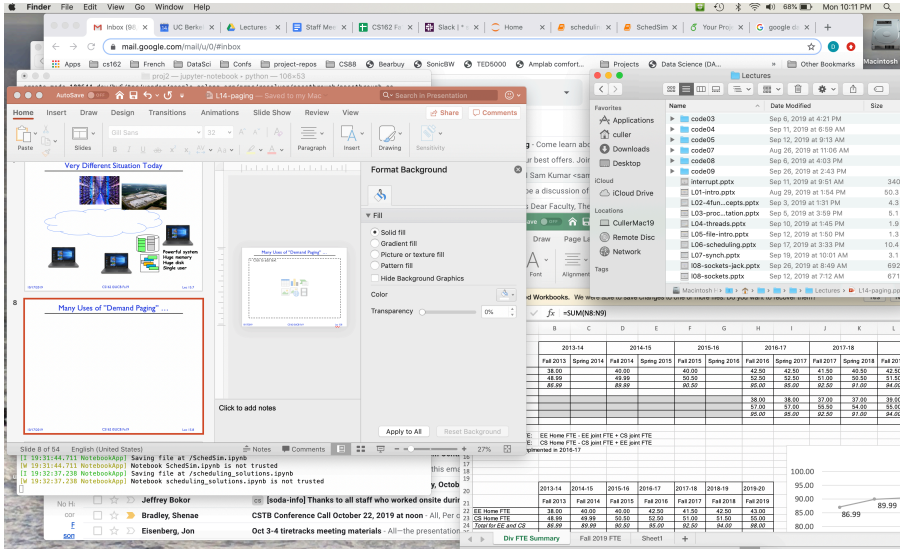
Our "Host Operating System"



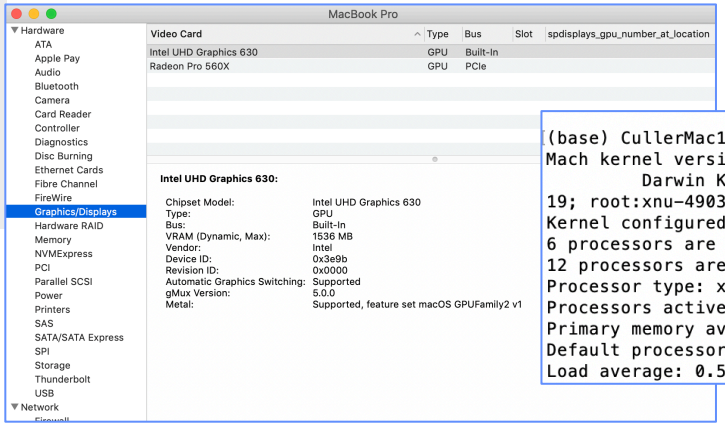
User-level "Guest" Operating System



Example: my Macbook Pro / OS-X

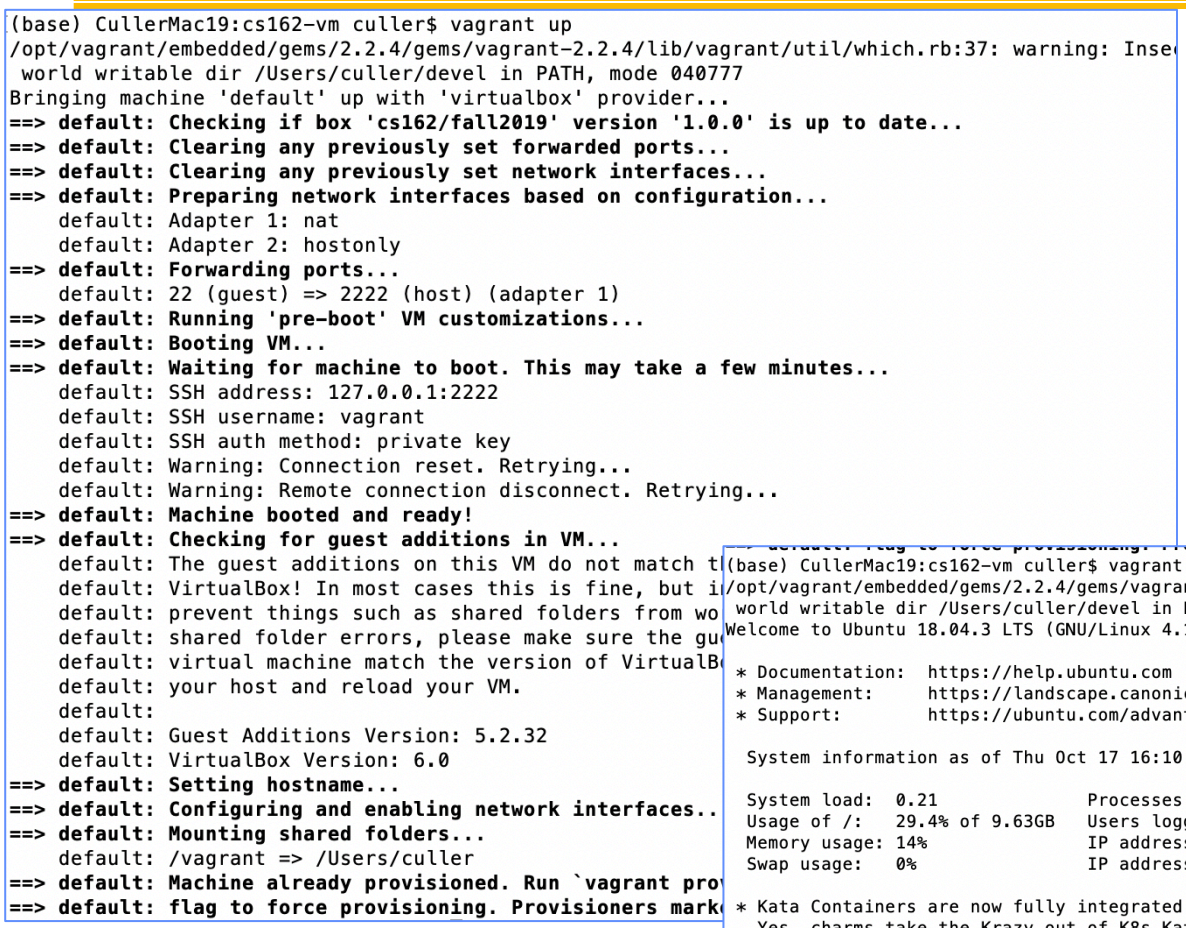


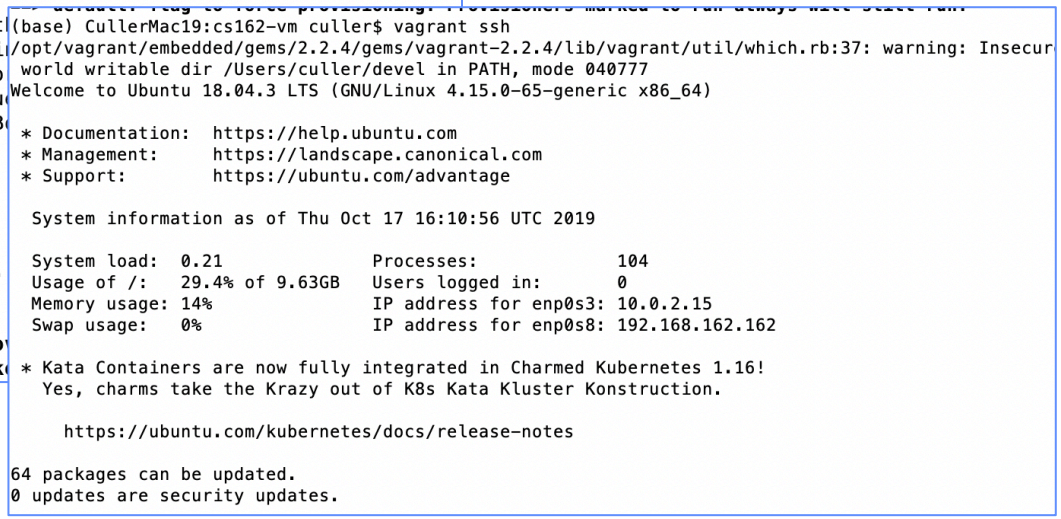
```
(base) CullerMac19:~ culler$ vm_stat
Mach Virtual Memory Statistics: (page size of 4096 bytes)
Pages free:                             1374276.
Pages active:                            887709.
Pages inactive:                          634340.
Pages speculative:                       254822.
Pages throttled:                              0.
Pages wired down:                        703546.
Pages purgeable:                          63546.
"Translation faults":                 2069239178.
Pages copy-on-write:                    54900953.
Pages zero filled:                    1485741210.
Pages reactivated:                       1061165.
Pages purged:                            2294989.
File-backed pages:                        573432.
Anonymous pages:                         1203439.
Pages stored in compressor:              1645906.
Pages occupied by compressor:             338775.
Decompressions:                         68562947.
Compressions:                           84641004.
Pageins:                                67082753.
Pageouts:                                  10069.
Swapins:                                71555118.
Swapouts:                               74956494.
```




```
(base) CullerMac19:~ culler$ hostinfo
Mach kernel version:
         Darwin Kernel Version 18.7.0: Thu Jun 20 18:42:21 PDT 20
19; root:xnu-4903.270.47~4/RELEASE_X86_64
Kernel configured for up to 12 processors.
6 processors are physically available.
12 processors are logically available.
Processor type: x86_64h (Intel x86-64h Haswell)
Processors active: 0 1 2 3 4 5 6 7 8 9 10 11
Primary memory available: 16.00 gigabytes
Default processor set: 408 tasks, 1875 threads, 12 processors
Load average: 0.53, Mach factor: 11.46
```

Up that Virtual Machine





In the Host OS

```
Processes: 409 total, 3 running, 406 sleeping, 1932 threads                                                 09:
Load Avg: 1.06, 1.16, 1.09  CPU usage: 0.83% user, 1.52% sys, 97.64% idle
SharedLibs: 316M resident, 53M data, 49M linkedit.
MemRegions: 118007 total, 3151M resident, 118M private, 1283M shared.
PhysMem: 10G used (3581M wired), 5770M unused.
VM: 2354G vsize, 1373M framework vsize, 71564960(0) swapins, 74956494(0) swapouts.
Networks: packets: 41765192/22G in, 22421128/7927M out. Disks: 18227275/582G read, 16666490/669G writte

PID    COMMAND      %CPU  TIME     #TH   #WQ  #PORTS MEM    PURG   CMPRS  PGRP  PPID  STATE
89197  siriknowledg 0.0   00:00.92 2     2    47     2668K  0B     1724K  89197 1     sleeping
89193  com.apple.DF 0.0   00:18.97 2     1    67     2684K  0B     1852K  89193 1     sleeping
82453  PAH_Extensio 0.0   00:28.51 3     1    235    16M    0B     7216K  82453 1     sleeping
75819  tzlinkd      0.0   00:00.01 2     2    14     452K   0B     448K   75819 1     sleeping
75787  MTLCompilerS 0.0   00:00.10 2     2    24     9032K  0B     9024K  75787 1     sleeping
75776  secd         0.0   00:00.84 2     2    36     3188K  0B     2552K  75776 1     sleeping
75098  DiskUnmountW 0.0   00:00.51 2     2    36     1412K  0B     1128K  75098 1     sleeping
75093  MTLCompilerS 0.0   00:00.06 2     2    21     5924K  0B     5916K  75093 1     sleeping
74938  ssh-agent    0.0   00:00.01 1     0    21     908K   0B     772K   74938 1     sleeping
56389  usbmuxd      0.0   00:02.39 3     1    38     1592K  0B     1008K  56389 1     sleeping
55576  SimulatorTra 0.0   00:27.65 3     1    137    9252K  0B     5212K  55576 1     sleeping
55575  com.apple.Co 0.0   00:28.03 3     1    143    10M    0B     6316K  55575 1     sleeping
54313  com.apple.Co 0.0   00:02.01 2     2    60     2608K  0B     1716K  54313 1     sleeping
53416  kcproxy      0.0   00:01.81 2     1    44     3488K  0B     2604K  53416 1     sleeping
16871  screencaptur 0.2   00:00.18 11    9    211    13M    84K    0B     16871 1     sleeping
16870  screencaptur 1.9   00:00.19 3     2    57     3048K+ 620K   0B     383   383   sleeping
16858  ssh          0.0   00:01.53 1     0    21     1448K  0B     0B     16857 16857 sleeping
16857  vagrant      0.0   00:00.00 5     0    17     612K   0B     0B     16857 16566 sleeping
16856  CoreServices 0.0   00:00.07 3     1    163    4544K  0B     0B     16856 1     sleeping
16690  mdworker_sha 0.0   00:00.02 3     1    50     3068K  0B     0B     16690 1     sleeping
16689  mdworker_sha 0.0   00:00.02 3     1    50     3192K  0B     0B     16689 1     sleeping
16688  mdworker_sha 0.0   00:00.02 3     1    50     3156K  0B     0B     16688 1     sleeping
16687  mdworker_sha 0.0   00:00.03 3     1    50     3188K  0B     0B     16687 1     sleeping
16678  mdworker_sha 0.0   00:00.04 4     1    58     5192K  0B     0B     16678 1     sleeping
16675  VBoxNetDHCP  0.0   00:00.02 2     0    15     4812K  0B     0B     16675 16627 sleeping
16674  VBoxHeadless 1.2   02:30.35 28    1    203    840M   0B     0B     16674 16627 sleeping
16645  writeconfig  0.0   00:00.01 2     2    18     980K   0B     964K   16645 1     sleeping
16627  VBoxSVC      0.0   00:06.52 15    2    100    7496K  0B     0B     16627 1     sleeping
16625  VBoxXPCOMIPC 0.0   00:02.84 1     0    21     2544K  0B     0B     16625 1     sleeping
16566  bash         0.0   00:00.03 1     0    21     924K   0B     0B     16566 16565 sleeping
```

In the vbox / vagrant

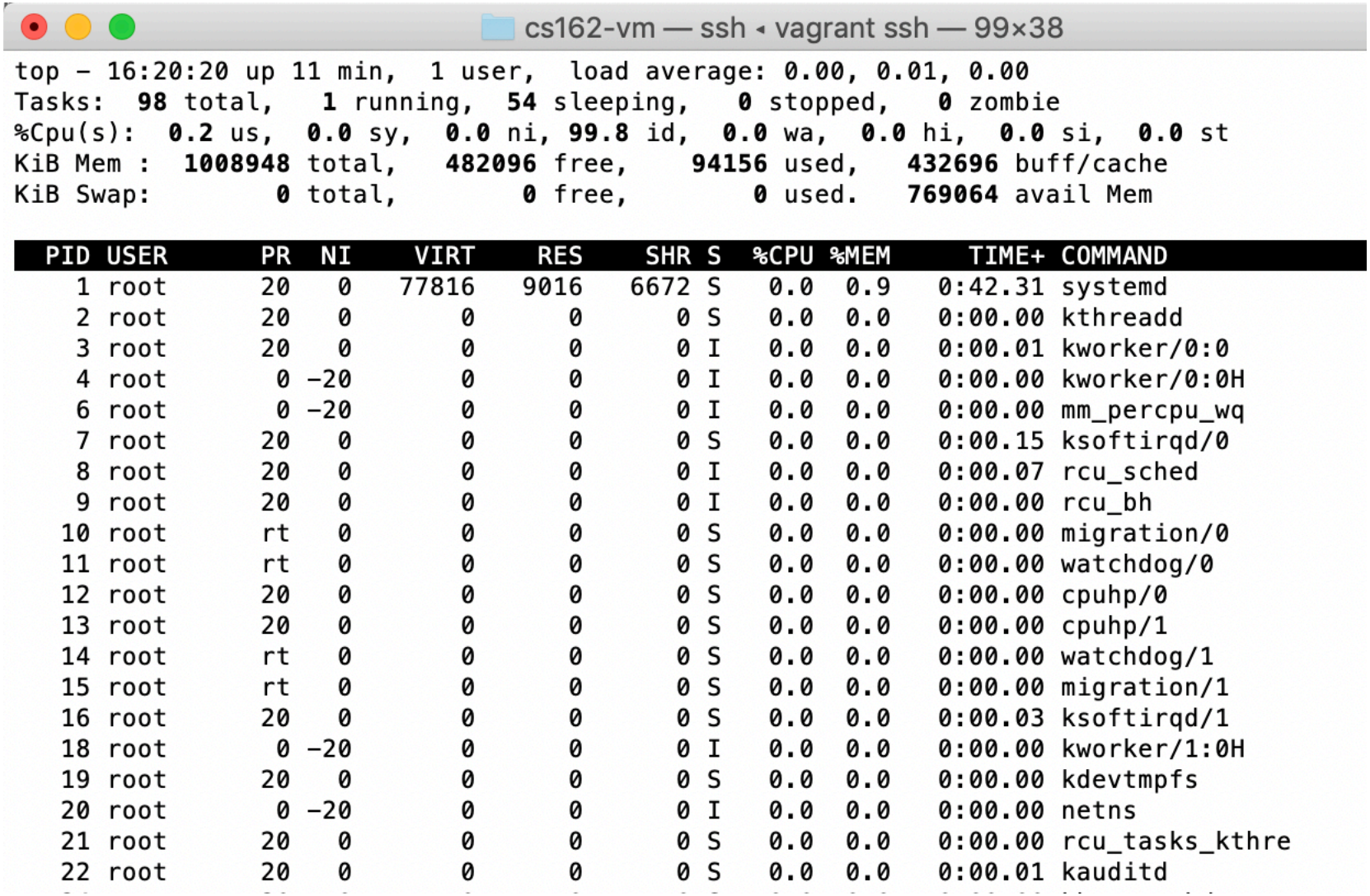
cs162-vm — ssh ◂ vagrant ssh — 99×38

```
[vagrant@development [16:17:00] ~ $ lshw -short
WARNING: you should run this program as super-user.
H/W path    Device  Class       Description
=========================================
                    system      Computer
/0                  bus         Motherboard
/0/0                memory      985MiB System memory
/0/1                processor   Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
/0/100              bridge      440FX - 82441FX PMC [Natoma]
/0/100/1            bridge      82371SB PIIX3 ISA [Natoma/Triton II]
/0/100/1.1          storage     82371AB/EB/MB PIIX4 IDE
/0/100/2            display     VirtualBox Graphics Adapter
/0/100/3    enp0s3  network     82540EM Gigabit Ethernet Controller
/0/100/4            generic     VirtualBox Guest Service
/0/100/5            multimedia  82801AA AC'97 Audio Controller
/0/100/7            bridge      82371AB/EB/MB PIIX4 ACPI
/0/100/8    enp0s8  network     82540EM Gigabit Ethernet Controller
/0/100/14   scsi2   storage     53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI
WARNING: output may be incomplete or inaccurate, you should run this program as super-user.
vagrant@development [16:17:42] ~ $
```

cs162-vm — ssh ◂

```
[vagrant@development [16:17:42] ~ $ sudo lshw -sho
H/W path           Device     Class       Description
=====================================================
                              system      VirtualBox
/0                            bus         VirtualBox
/0/0                          memory      128KiB BIOS
/0/1                          memory      985MiB System memory
/0/2                          processor   Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
/0/100                        bridge      440FX - 82441FX PMC [Natoma]
/0/100/1                      bridge      82371SB PIIX3 ISA [Natoma/Triton II]
/0/100/1.1                    storage     82371AB/EB/MB PIIX4 IDE
/0/100/2                      display     VirtualBox Graphics Adapter
/0/100/3           enp0s3     network     82540EM Gigabit Ethernet Controller
/0/100/4                      generic     VirtualBox Guest Service
/0/100/5                      multimedia  82801AA AC'97 Audio Controller
/0/100/7                      bridge      82371AB/EB/MB PIIX4 ACPI
/0/100/8           enp0s8     network     82540EM Gigabit Ethernet Controller
/0/100/14          scsi2      storage     53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI
/0/100/14/0.0.0    /dev/sda   disk        10GB HARDDISK
/0/100/14/0.0.0/1  /dev/sda1  volume      10238MiB EXT4 volume
/0/100/14/0.1.0    /dev/sdb   disk        10MB HARDDISK
vagrant@development [16:17:59] ~ $
```

Guest ubuntu Linux



And networking

```
(base) CullerMac19:~ culler$ ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
        options=1203<RXCSUM,TXCSUM,TXSTATUS,SW_TIMESTAMP>
        inet 127.0.0.1 netmask 0xff000000
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
        nd6 options=201<PERFORMNUD,DAD>
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
stf0: flags=0<> mtu 1280
XHC20: flags=0<> mtu 0
VHC128: flags=0<> mtu 0
XHC0: flags=0<> mtu 0
XHC1: flags=0<> mtu 0
en5: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ether ac:de:48:00:11:22
        inet6 fe80::aede:48ff:fe00:1122%en5 prefixlen 64 scopeid 0x8
        nd6 options=201<PERFORMNUD,DAD>
        media: autoselect (100baseTX <full-duplex>)
        status: active
ap1: flags=8802<BROADCAST,SIMPLEX,MULTICAST> mtu 1500
        ether f2:18:98:09:f1:9e
        media: autoselect
        status: inactive
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ether f0:18:98:09:f1:9e
        inet6 fe80::13:d73b:8f93:c347%en0 prefixlen 64 secured scopeid 0xa
        inet6 2607:f140:400:a009:c5f:cab5:e9c0:dd90 prefixlen 64 autoconf secu
        inet6 2607:f140:400:a009:9810:14f5:1815:8028 prefixlen 64 autoconf tem
        inet 10.142.39.7 netmask 0xfffffc00 broadcast 10.142.39.255
        nd6 options=201<PERFORMNUD,DAD>
        media: autoselect
```

```
        media: <unknown type>
        status: inactive
utun0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 2000
        inet6 fe80::c655:5dfe:1be1:cdb7%utun0 prefixlen 64 scopeid 0x12
        nd6 options=201<PERFORMNUD,DAD>
utun1: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1380
        inet6 fe80::13d6:f064:b6b5:5162%utun1 prefixlen 64 scopeid 0x17
        nd6 options=201<PERFORMNUD,DAD>
vboxnet0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        ether 0a:00:27:00:00:00
        inet 192.168.162.1 netmask 0xffffff00 broadcast 192.168.162.255
vboxnet1: flags=8842<BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ether 0a:00:27:00:00:01
```

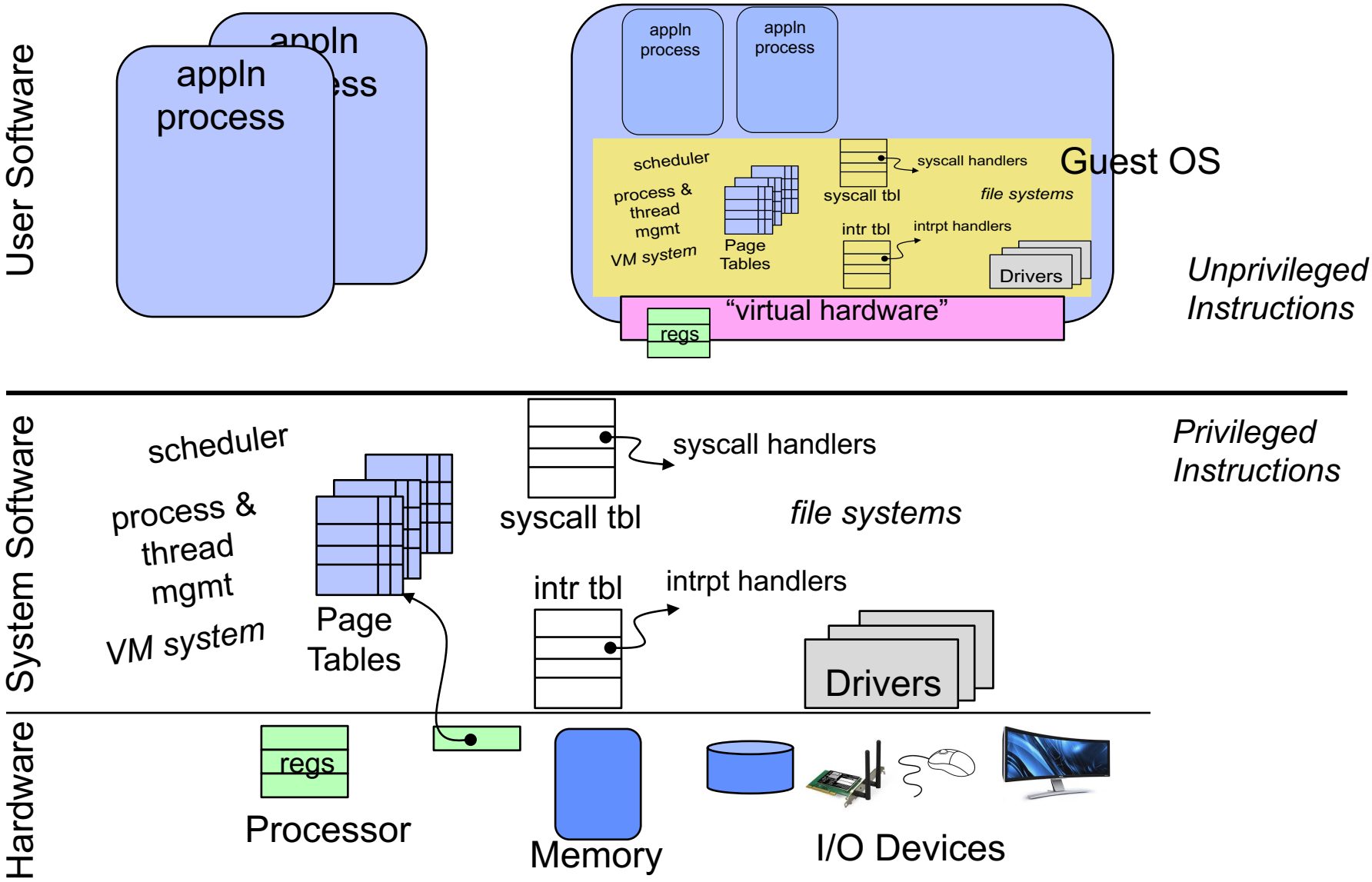
```
vagrant@development [16:23:20] ~ $ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::19:d6ff:fe3c:94c8  prefixlen 64  scopeid 0x20<link>
        ether 02:19:d6:3c:94:c8  txqueuelen 1000  (Ethernet)
        RX packets 6486  bytes 4872047 (4.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3038  bytes 285494 (285.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.162.162  netmask 255.255.255.0  broadcast 192.168.1
        inet6 fe80::a00:27ff:fed8:b072  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:d8:b0:72  txqueuelen 1000  (Ethernet)
        RX packets 312  bytes 28704 (28.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 292  bytes 41012 (41.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 52  bytes 4447 (4.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 52  bytes 4447 (4.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ɔment [16:23:27] ~ $
```

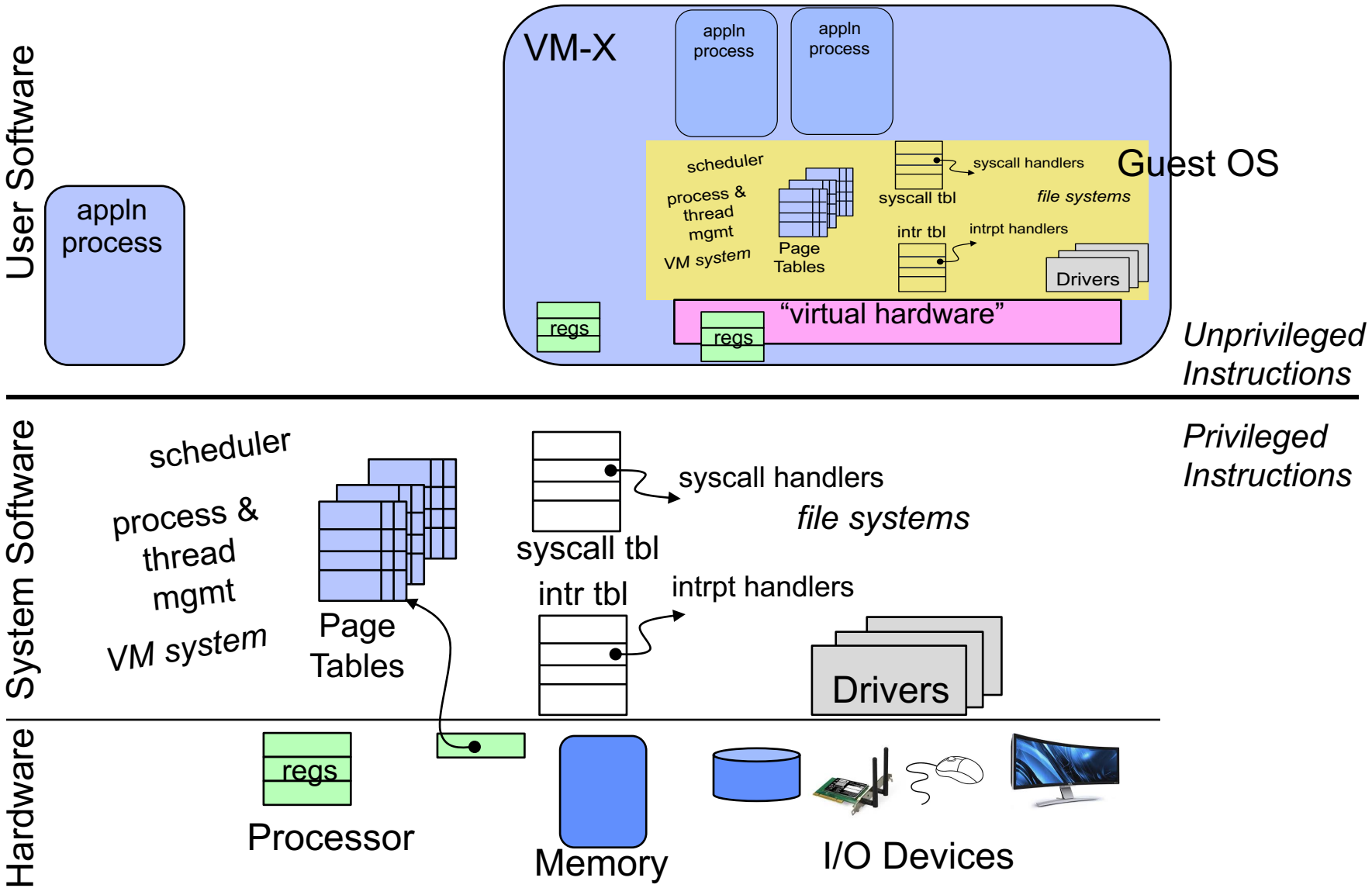
User-level "Guest" Operating System



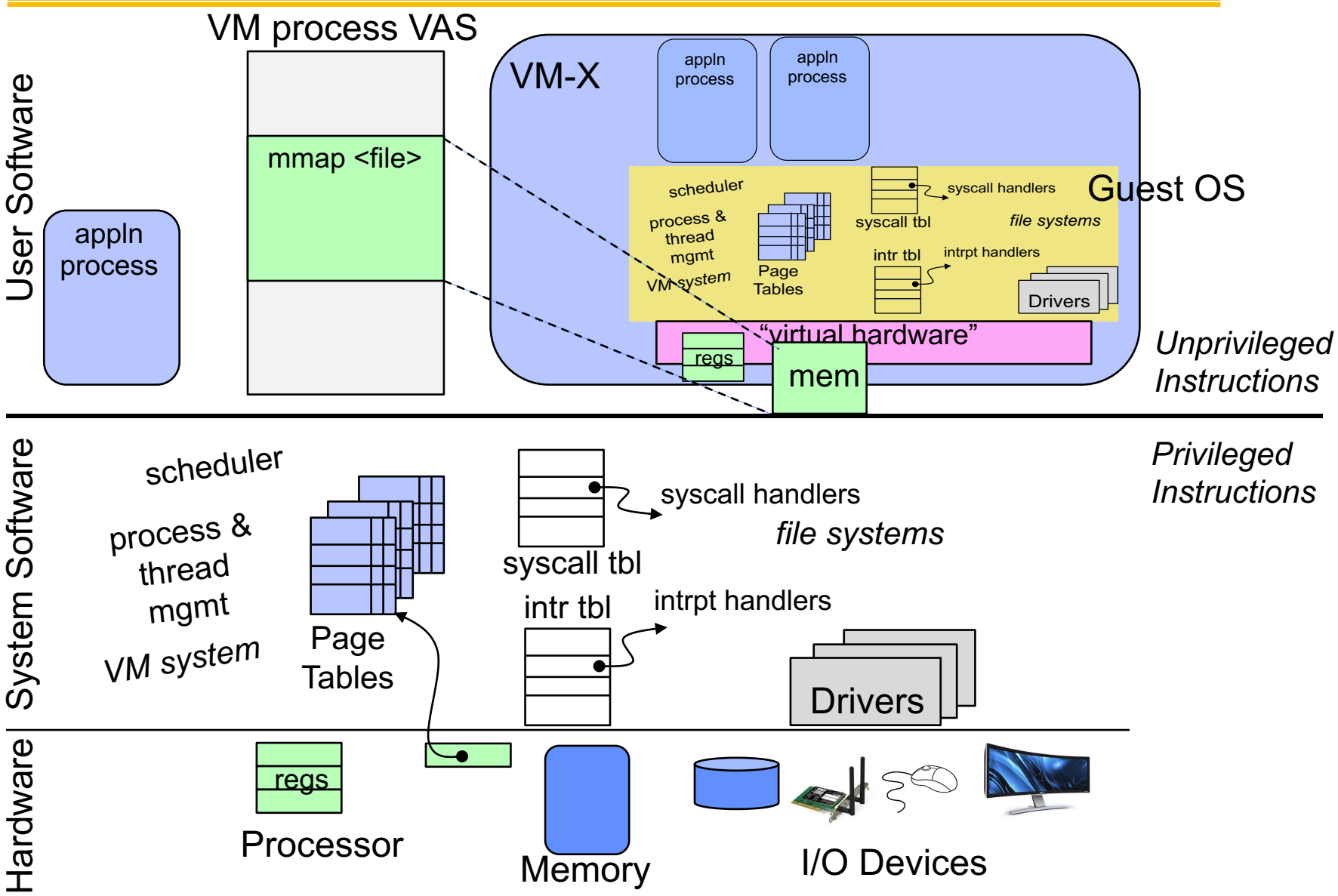
Software Emulation of Hardware

- **Example: QEMU for x86**
  - Used to do MIPS R3000 (subset) emulation as 61C project
- **User software emulates the behavior of every single instruction**
  - Data structure for Processor, Memory, I/O, etc
  - Code for Instruction Cycle: Fetch Instruction, Decode, Fetch Operands, Perform Operation, Store Results, Update PC
  - Emulate privilege levels, interrupts, MMU, . . . too
  - Load (i.e., boot) image of the operating system
  - Initializes (virtual) HW, loads pgms, schedules threads, etc.
- **Popular in the 90's**
  - run Windows (x86) on your MAC (M68000), …
  - Software Fault Isolation
- **Lots of dynamic translation optimizations to reduce overhead from 1000-fold to 2-10 fold**
  - Want more like 10-20% overhead

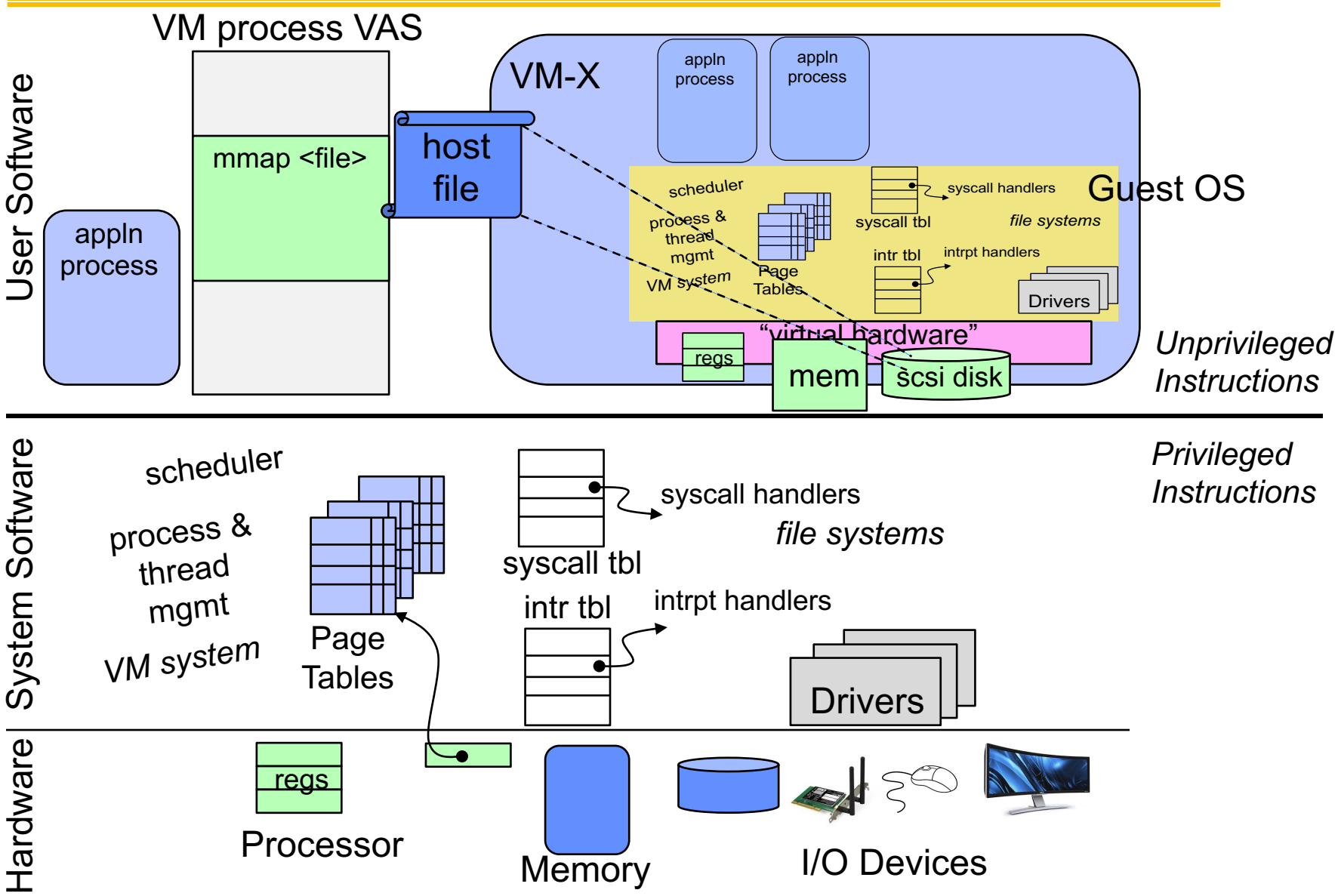
Guest Virtual Hardware: proc regs



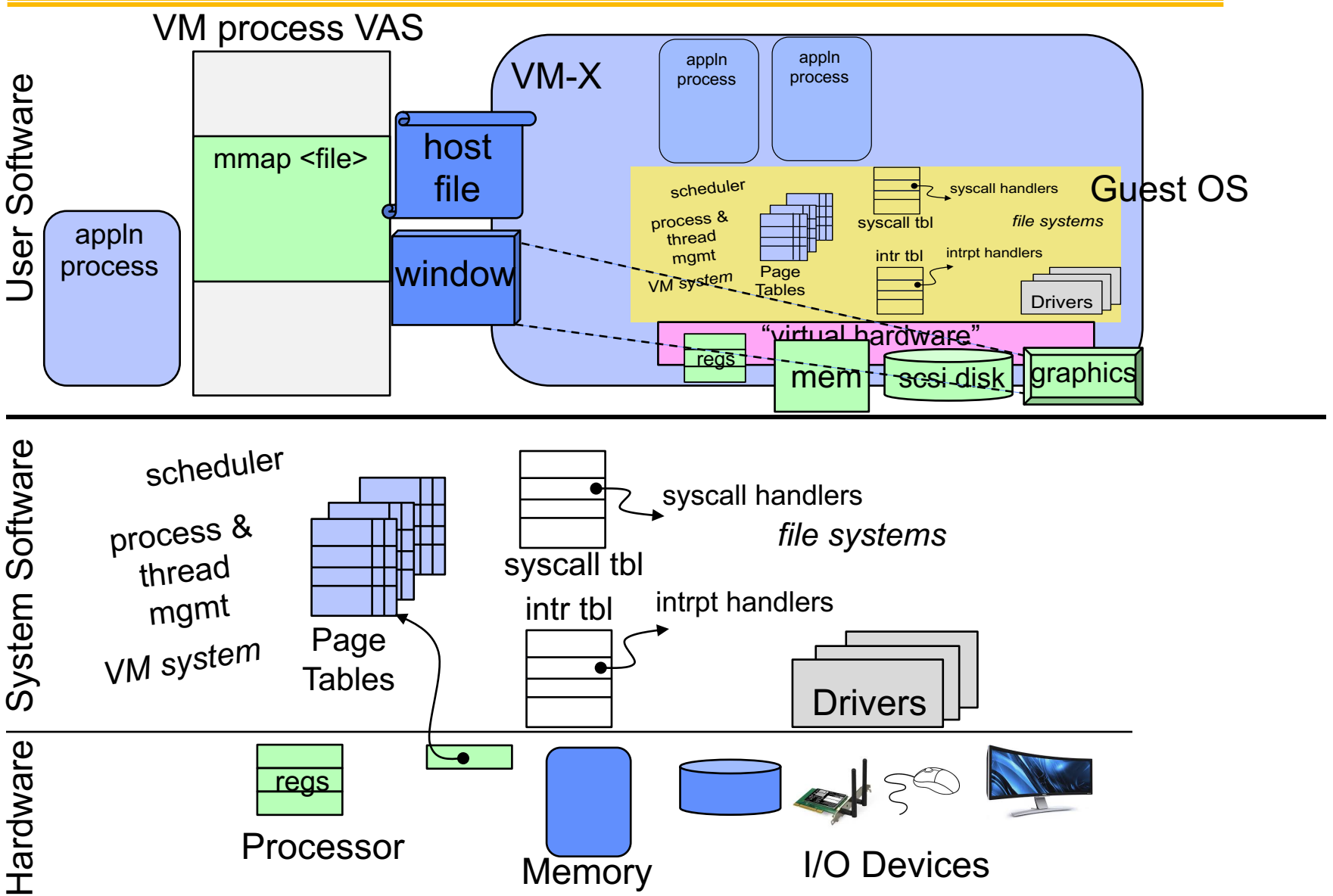
Virtual Hardware: "physical mem"



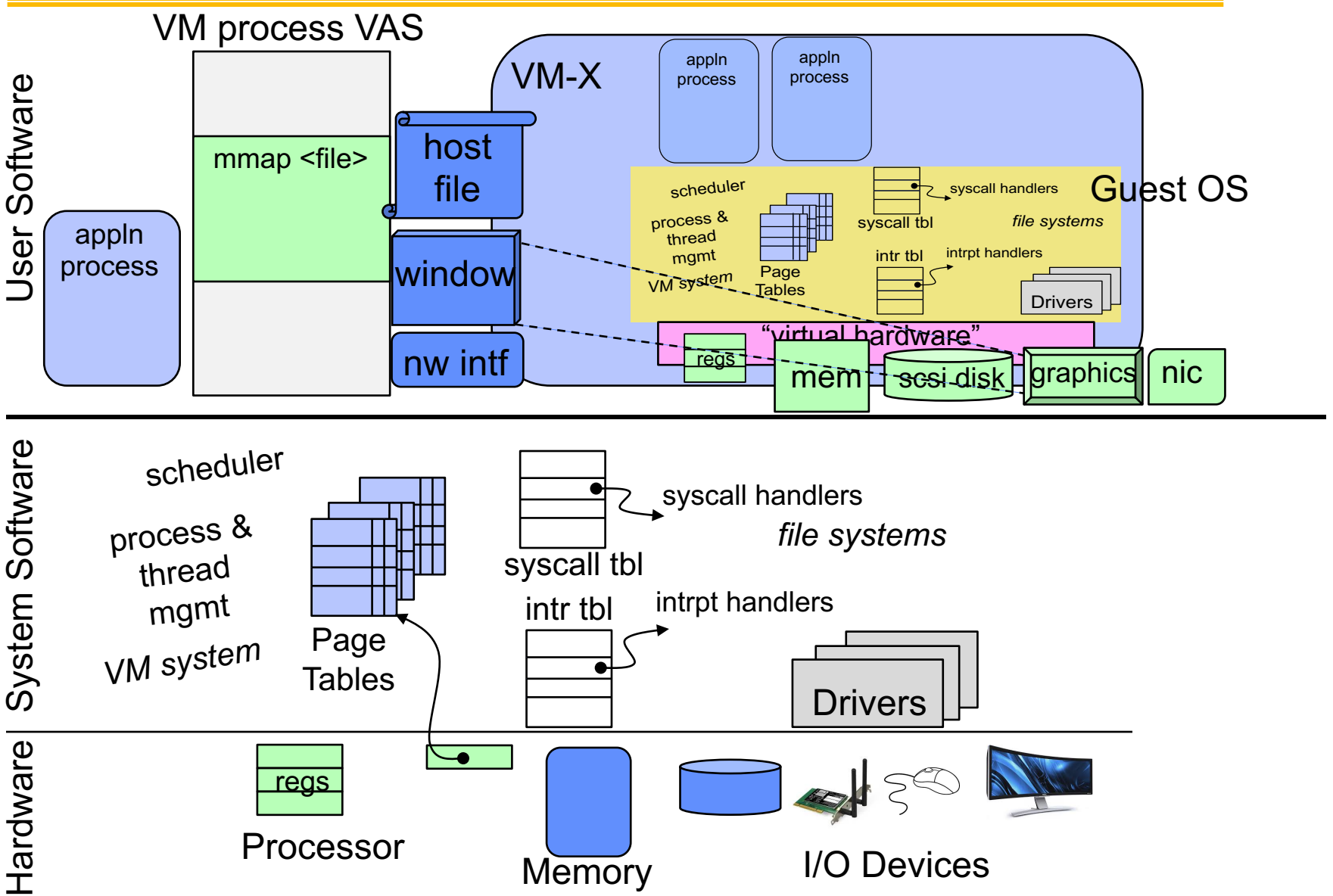
Virtual Hardware: "I/O Device"



Virtual Hardware: "I/O Devices"



Virtual Hardware: "I/O Devices"



Challenges

- How do we provide virtual address spaces for each of the guest processes?
  - Host OS has the REAL page tables
- When Guest process does syscall (trap), how does it vector through the Guest OS Syscall Table?
  - Interrupts go through the physical interrupt vector
- How does the guest OS protect itself from its guest application processes & each other?
- Guest OS executes as Unprivileged
  - How does it set up its page table? Disable interrupts? …
  - Driver Read/Write to I/O devices? Handle interrupts?
- How do interrupts get delivered to Guest OS?
- Deal with the huge diversity of I/O devices that might be on the host machine?

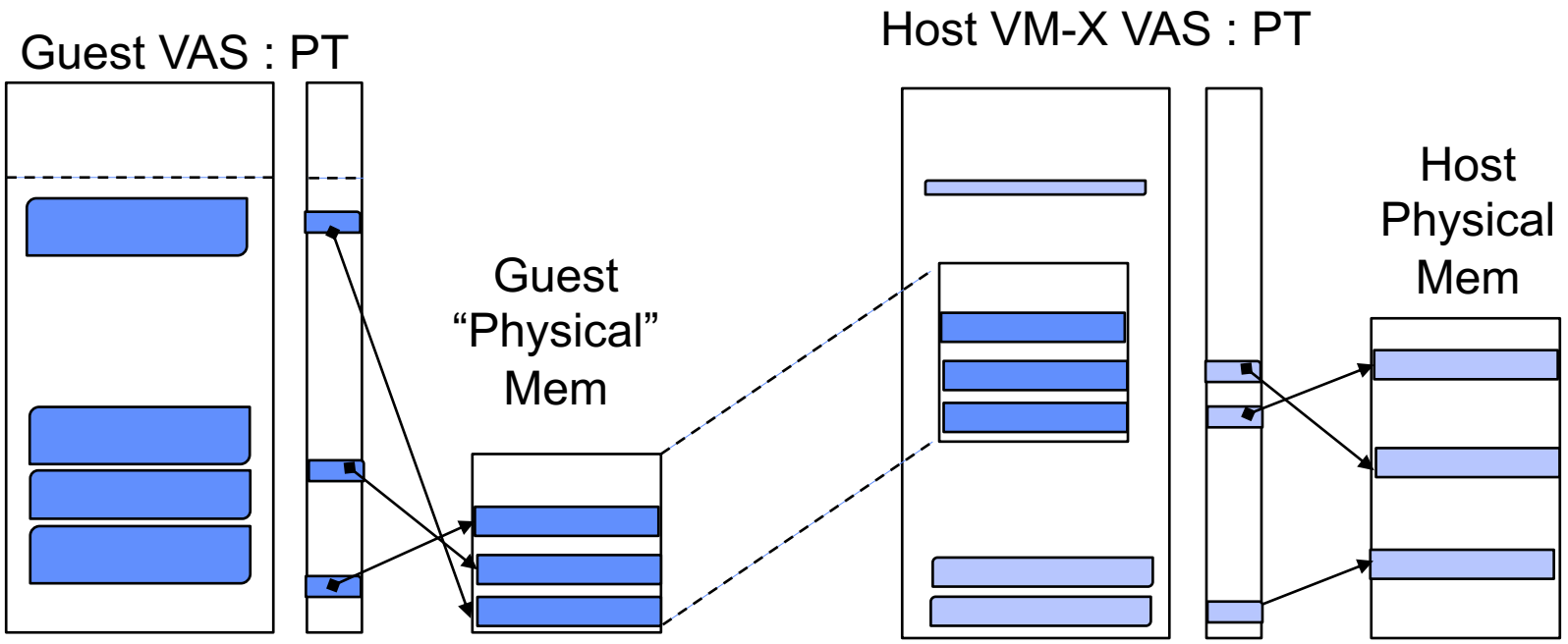
I/O diversity (1st Try)

- Guest OS is configured with drivers for a few particular (virtual) I/O devices
  - They claim to be these (often old) devices (SCSI disk, …)
- Driver accesses its device through:
- Programmed I/O to memory-mapped registers
  - Read status registers, Write Control Registers
  - Page Table translates to specific physical addresses of I/O device registers
- Direct Memory Access (DMA)
  - Bulk transfer to/from a memory buffer & device
- Device raises interrupt line when operations complete
- => Catch these operations and emulate the device on the host object (file, NW, window)

Guest User Level Memory Access

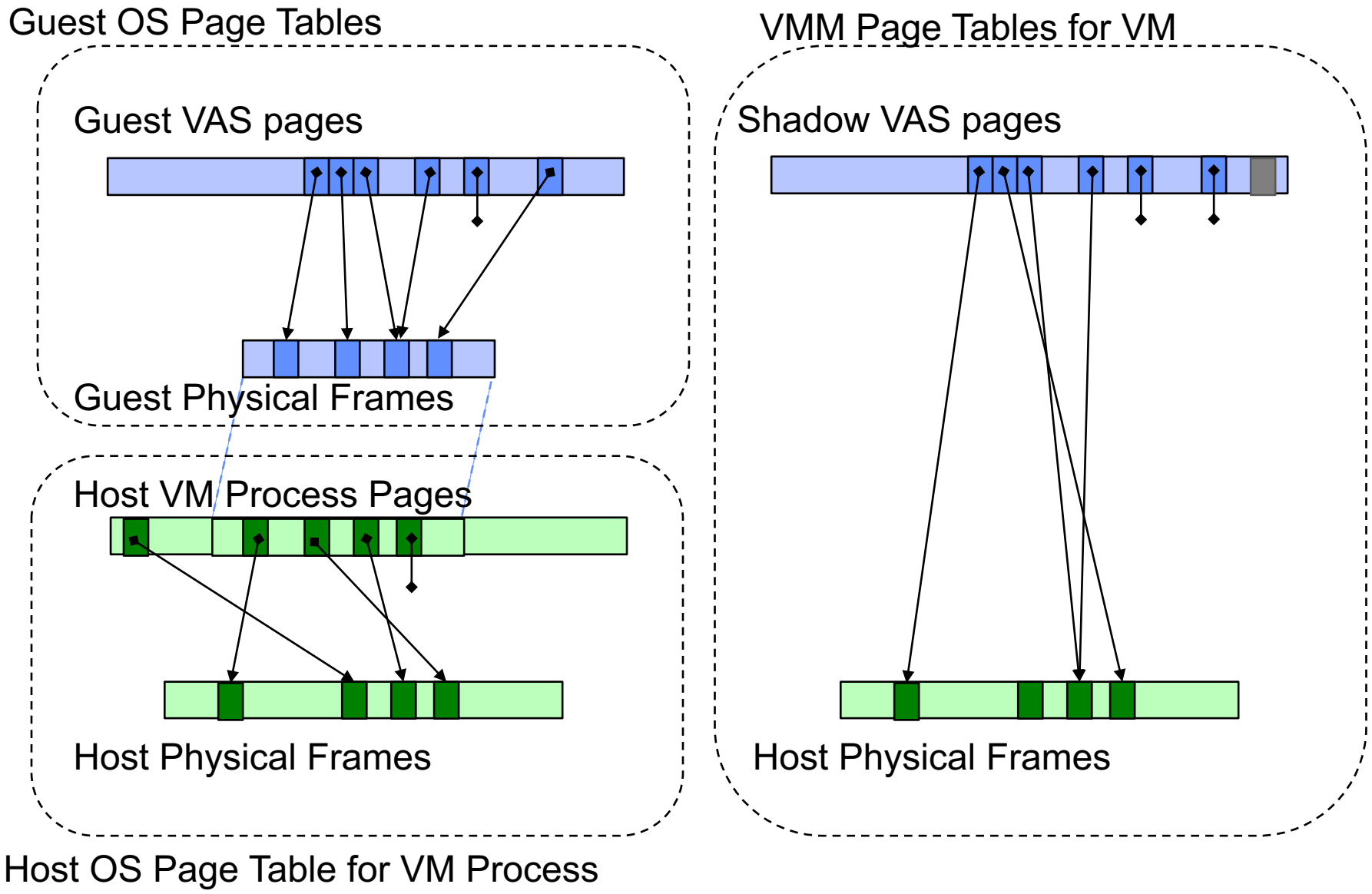
- Guest User Process accesses a virtual address space set up by the Guest OS
- But hardware translates the Virtual to Physical Address through the REAL Page Table
  - Not the Guest OS Page Table
- ???

The Two Translations

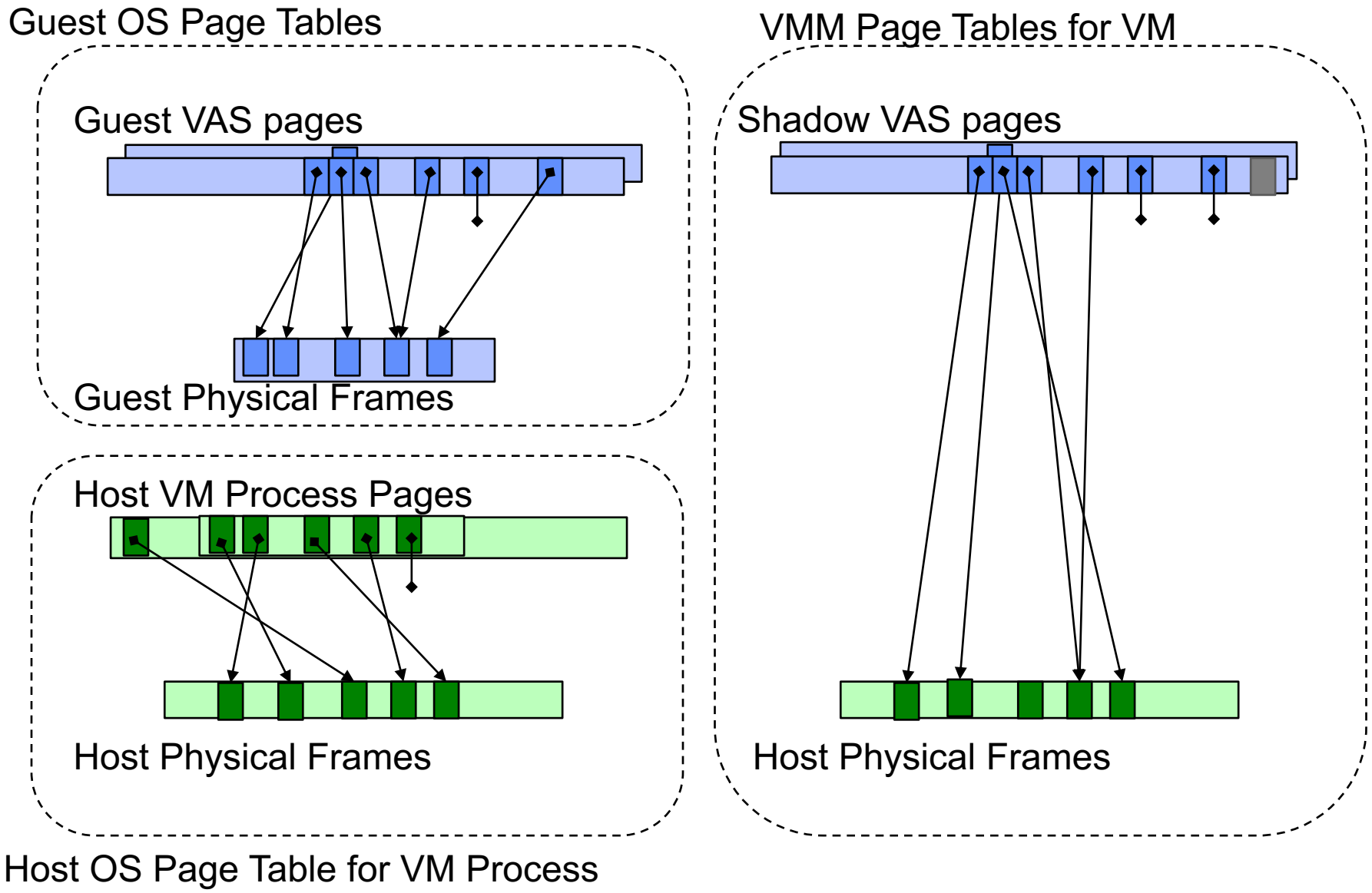


- **Guest process operates on virtual addresses**
  - **It can't see what physical address these translate into**
- **Guest OS set up mapping to "guest mem"**
  - **It thinks these are physical addresses**
  - **They are actually within a mapped region of the VM process**
- **Host OS maps VM process VAS to physical mem**

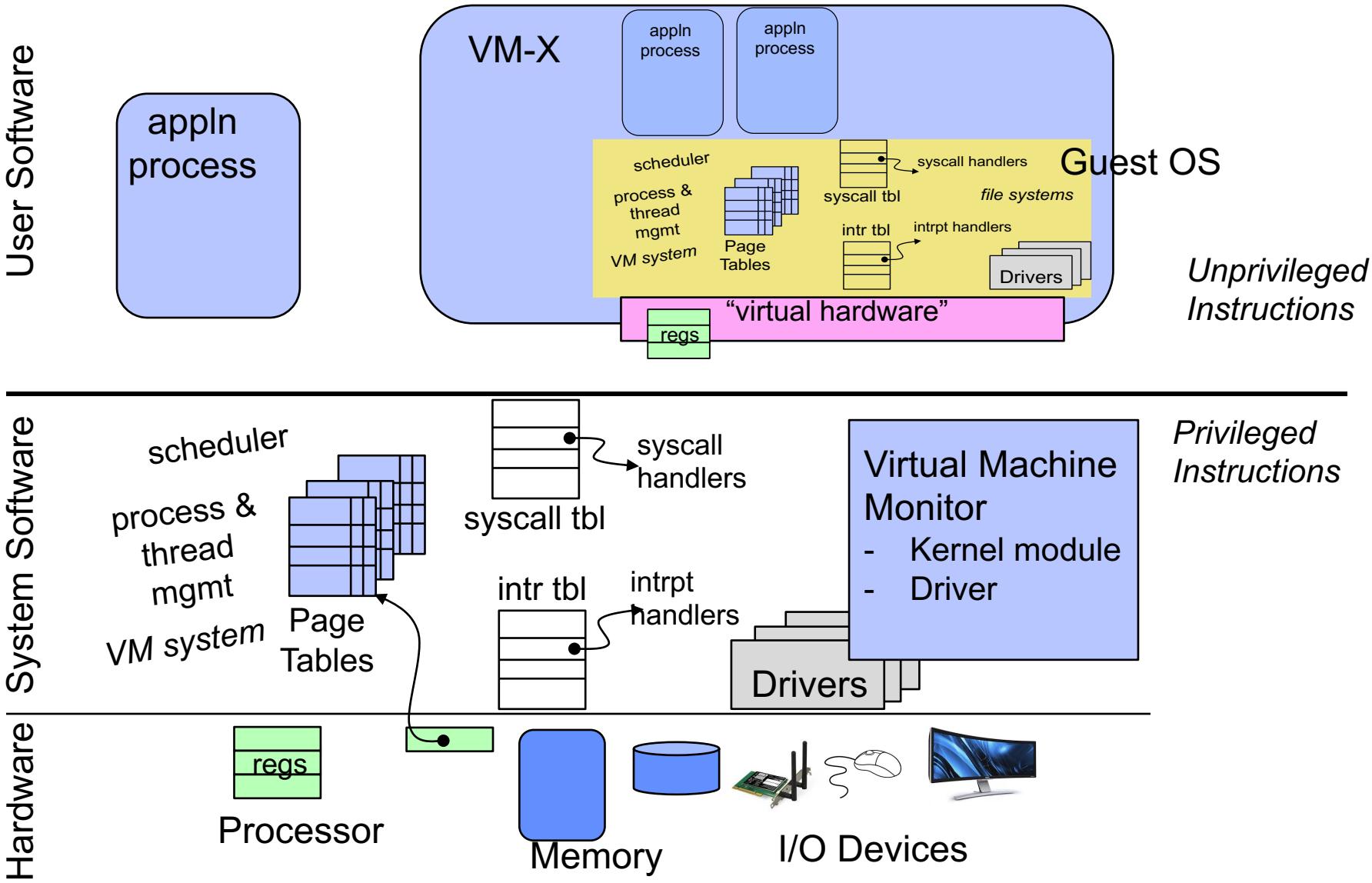
VMM Shadow Page Tables



VMM Shadow Page Tables (cont)



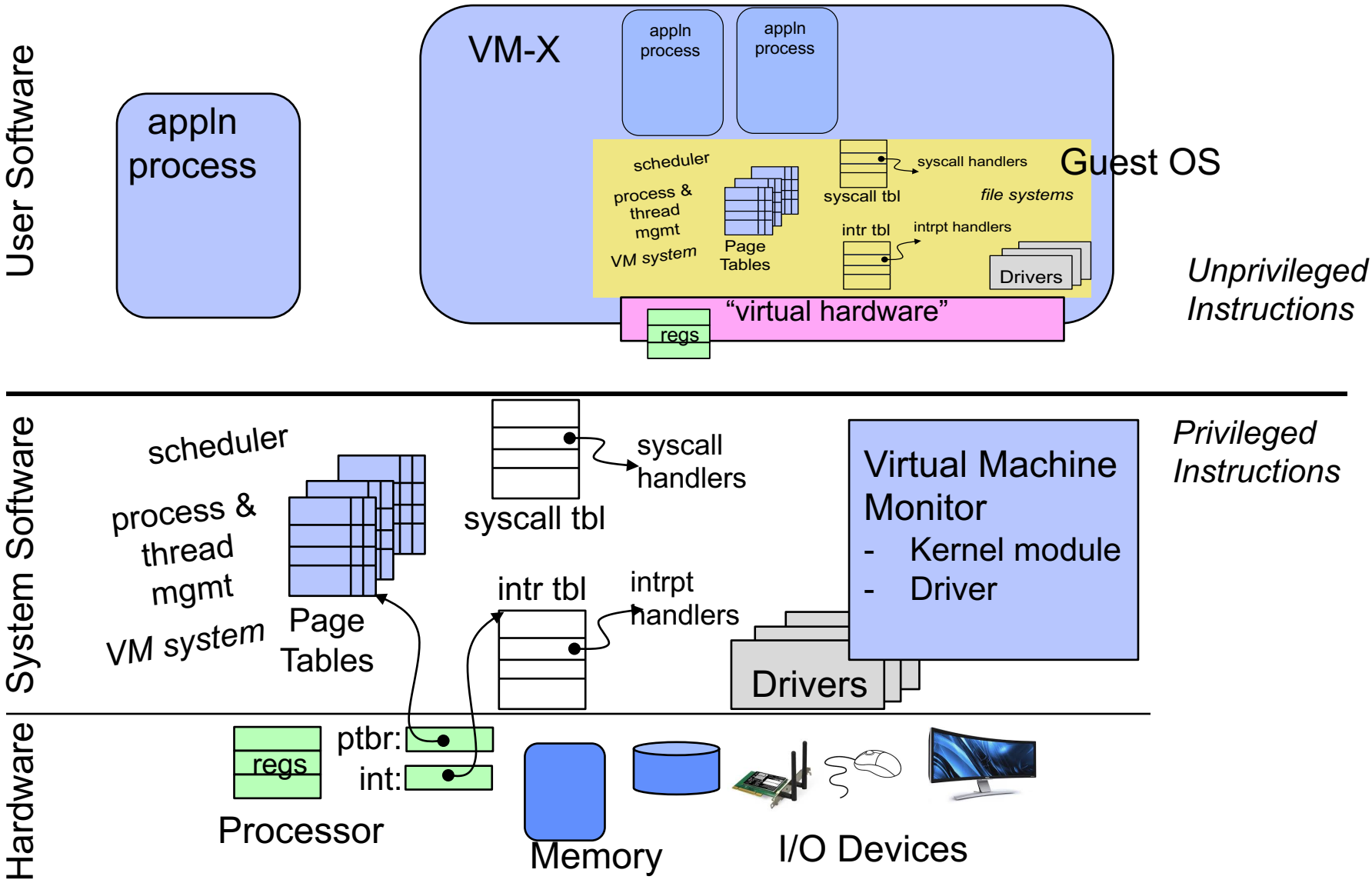
Virtual Machine Monitor



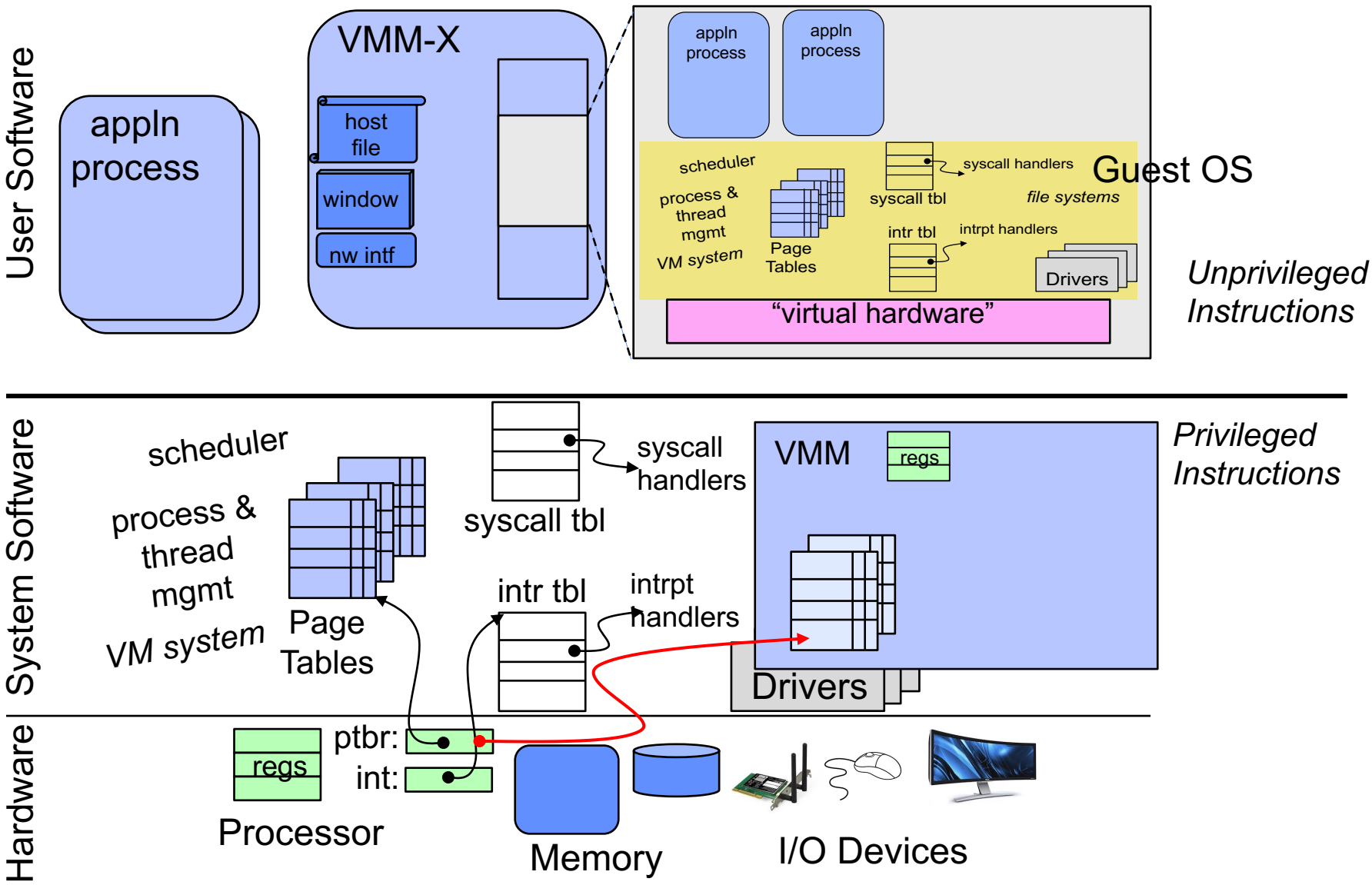
Virtual Machine Monitor

- Extends the host operating system using established dynamic mechanisms
  - Kernel module / Driver
  - Requires administrative privileges – but not reboot
  - Has visibility into page table management, interrupt control, hardware configuration
- Maintains shadow page tables for every page table in the guest OS
  - Mapping from Guest OS Process VAS to Host Physical Mem
- How do these shadow page tables get used?

Virtual Machine Monitor



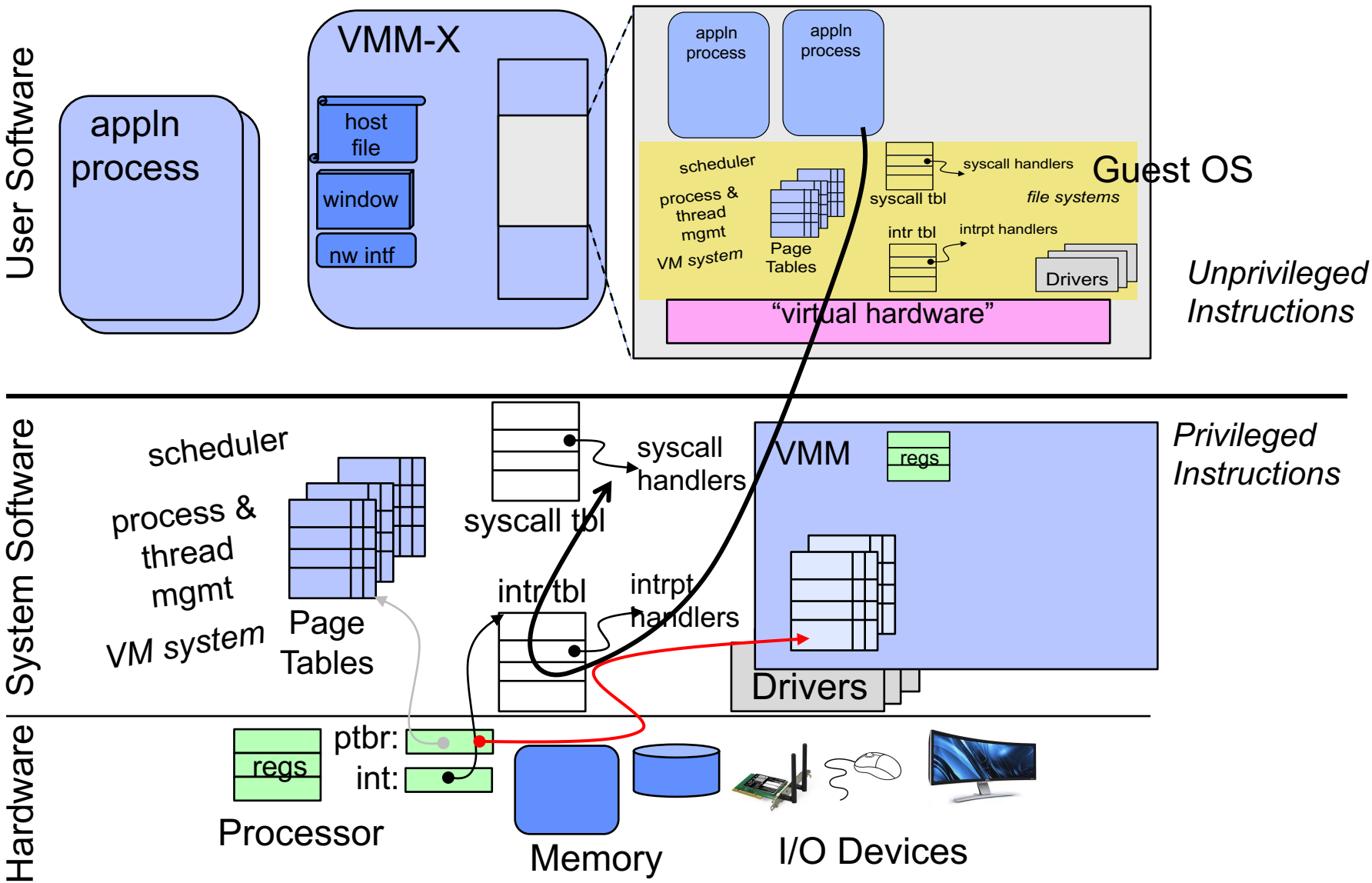
VMM – VM – VMM extension process



VM Guest OS Processes

- Host OS schedules VMM threads
- VMM switches to shadow page table for guest OS's current process
- Resumes Guest OS Thread & Process through return-from-interrupt
- Guest OS process runs with composite translation: Guest VA => VMMX VA => PA
  - Behaves as it Guest OS page table was in use
  - Guest OS region inaccessible to its user processes (protected)
- What happens is there is an interrupt?
- How does it make a SYSCALL
- What about the Guest OS?
  - It is not trusted by Host OS, Unpriviledged !

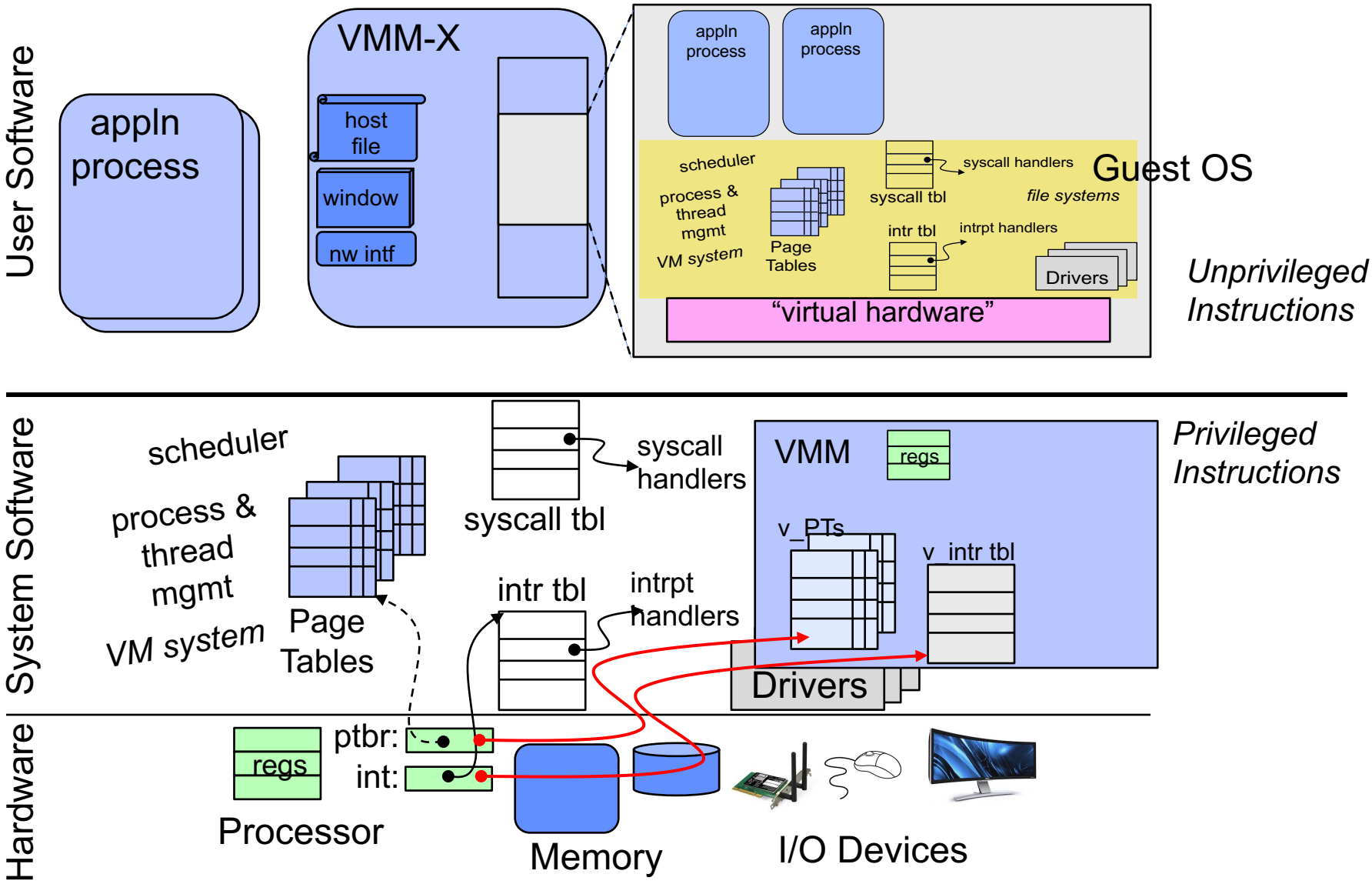
Interrupts while in VM ???



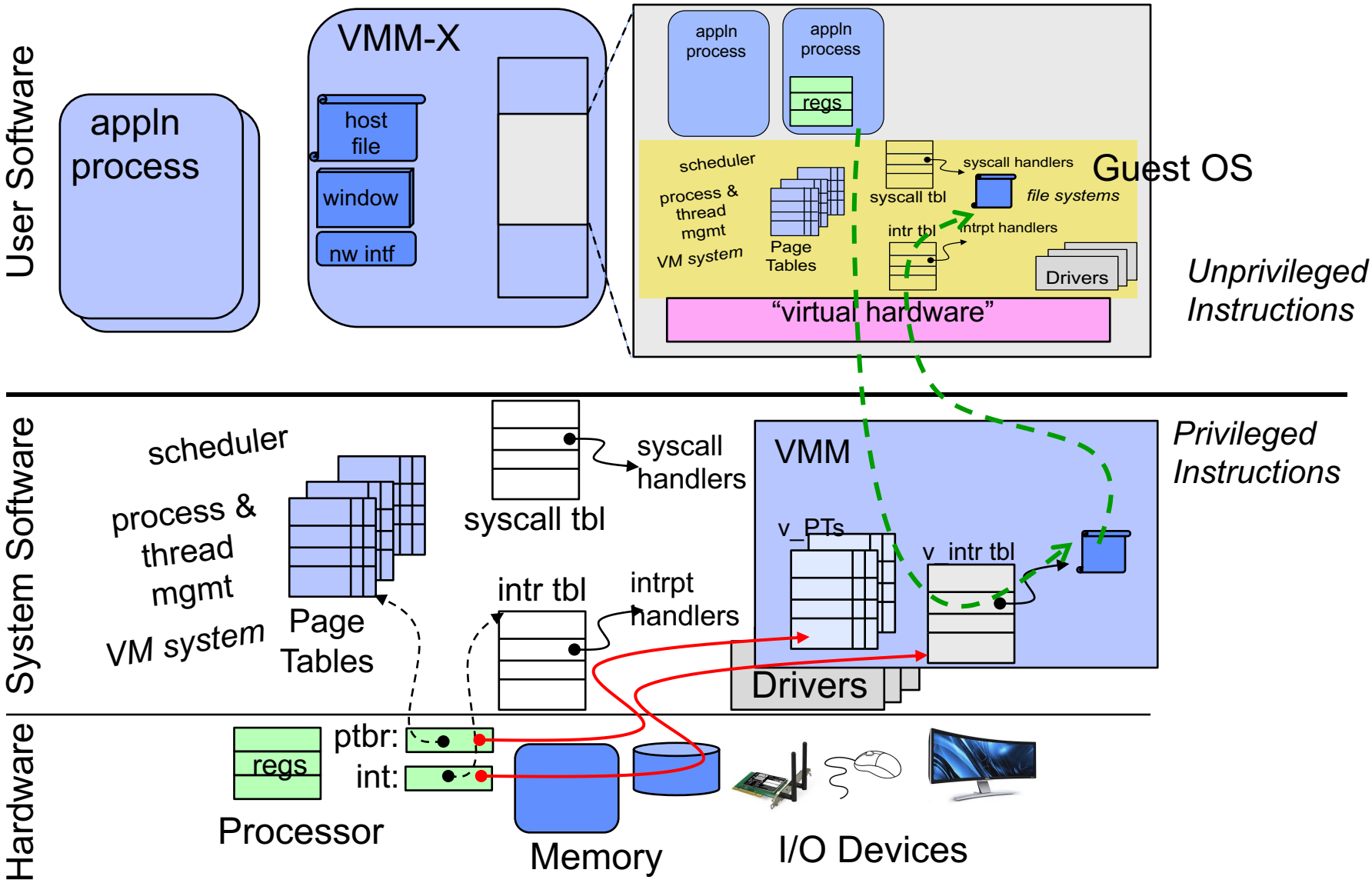
Interrupts

- Most interrupts have to do with hardware devices and need to be handled by Host OS
  - Regardless of whether Host process, Guest OS process or Guest OS is running
- But when Guest processes fault/trap it needs to be handled by the Guest OS
  - SYSCALL, Divide by zero, …
  - Page Fault ???
- Virtual Machine Monitor needs to be involved in Interrupt handling when VM is “running”

VMM: Interrupt Handling



Guest Process SYSCALL int

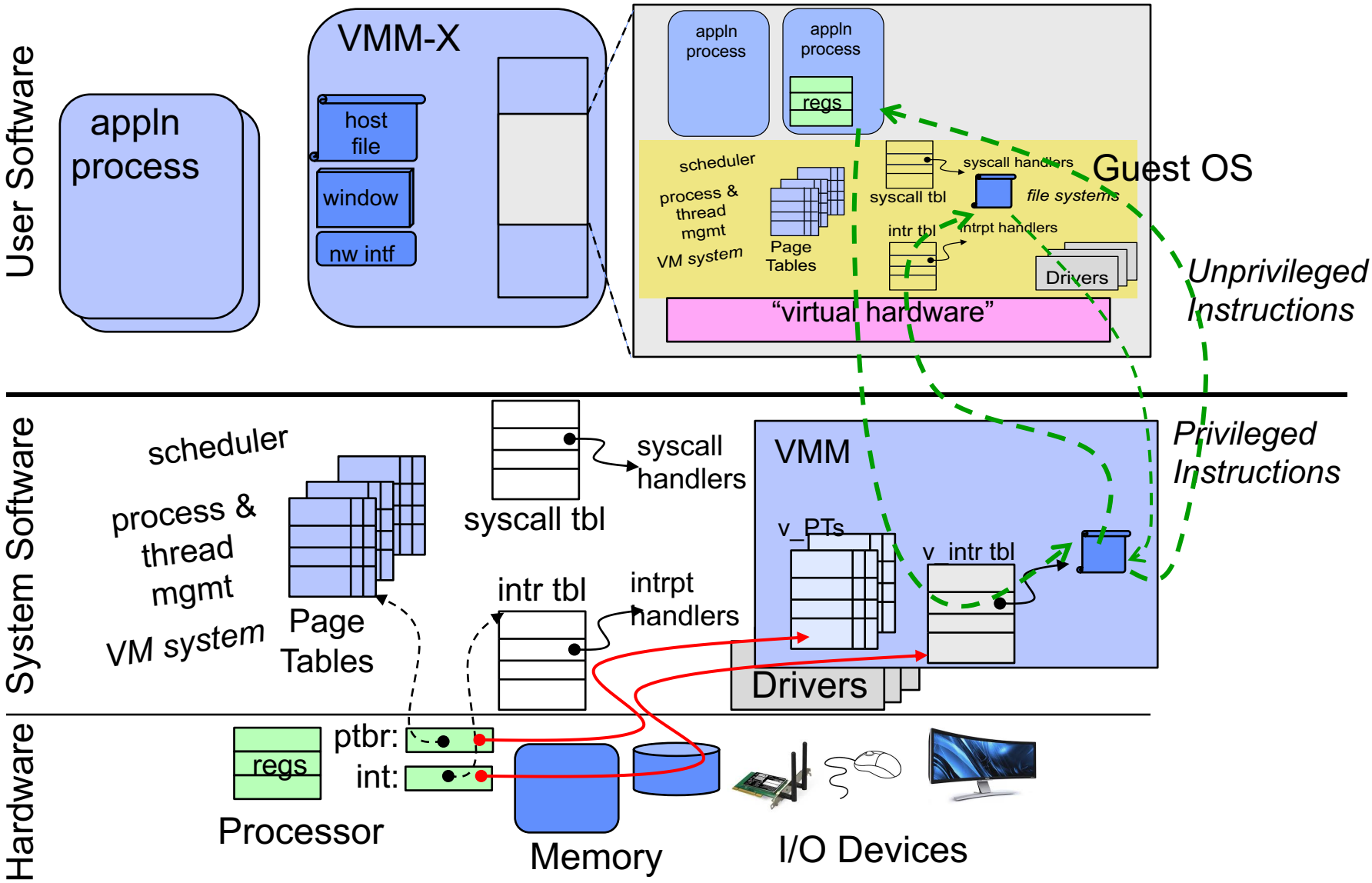


Guest Process Syscall

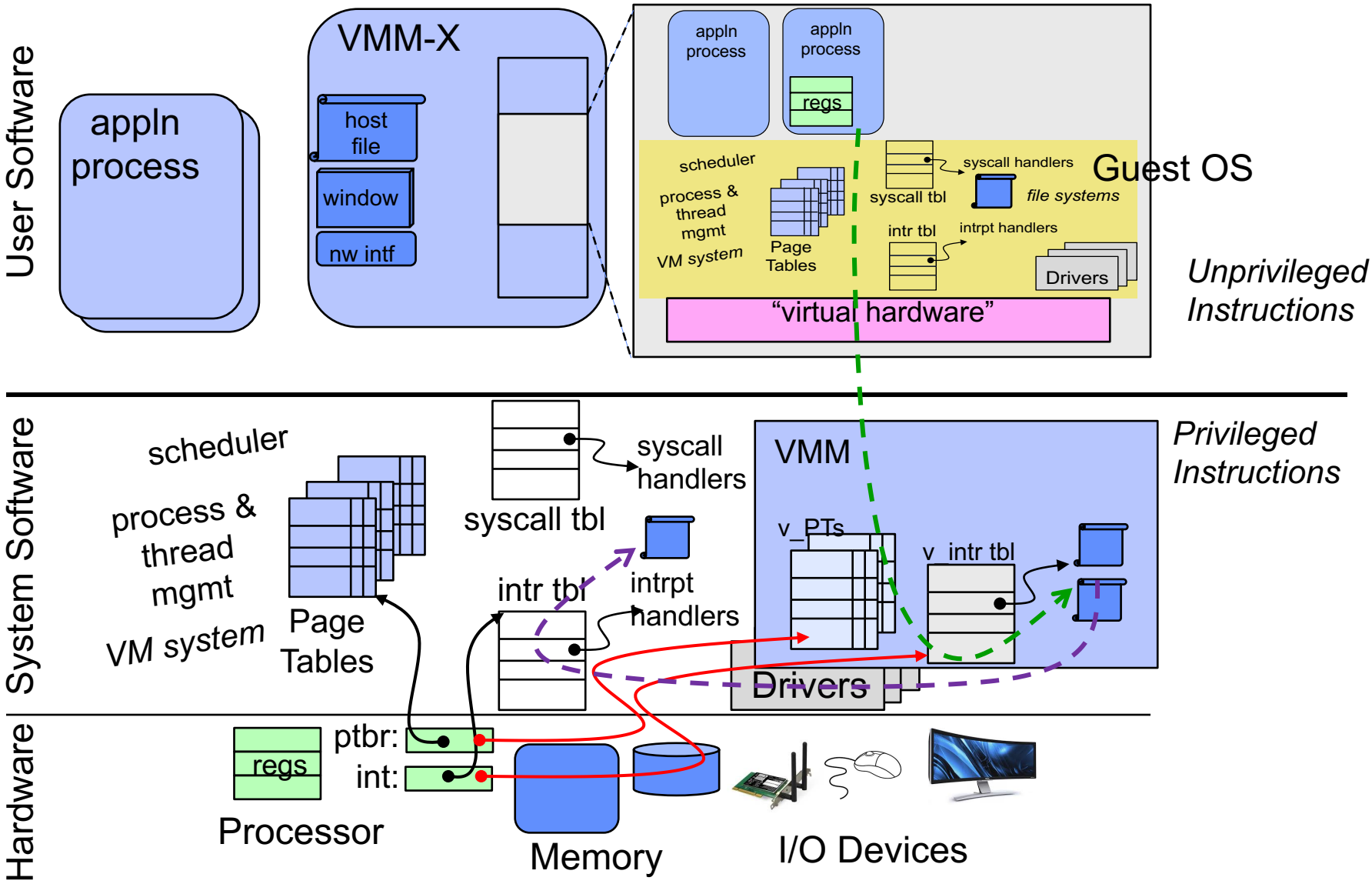
- Traps to VMM
- VMM can dispatch the exception to the Guest OS syscall hander (through its interrupt vector), which will process args, dispatch to particular system call, etc.
- The Guest OS return-from-interrupt resumes the VMM
- The VMM can then resume the Guest OS process with the syscall result

- VMM serves as intermediary between the Guest Processes and the Guest OS

Guest Process SYSCALL int return



Host Interrupt during VM



VMM Interrupt Processing

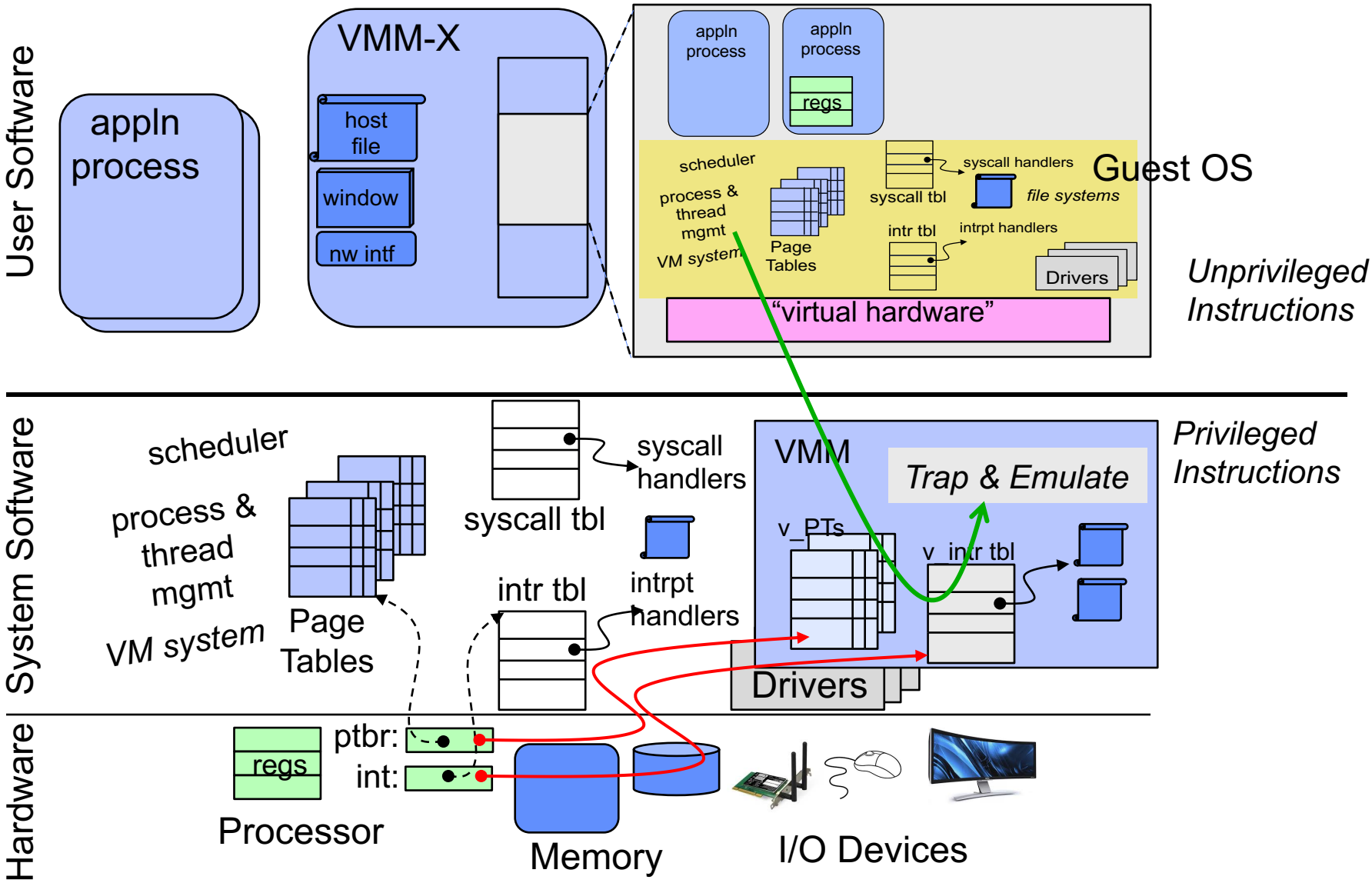
- During VM
- VMM catches interrupts and either redirects them back to the Host OS or up to the Guest OS
- Otherwise, the Host OS receives interrupts
- Those that involve the VMM will be delivered to it like other drivers

- But, Guest OS is not privileged
  - It is running at user level (some VMMs run it at PL=1)
- What happens when it executes privileged instructions?
  - Access to kernel region? Disable/Enable Interrupts? …

Virtualizable Instruction Set Arch.

- An instruction set is *virtualizable* if all "sensitive" instructions cause a trap if executed in unprivileged mode…
- x86 ISAs (till last several years) were not virtualizable in a strict sense and hugely complex
  - See VMware paper for incredible work arounds to get effective virtualizations
  - Recent generations of x86 have improved support for virtualization
- We'll assume all virtualization sensitive actions by the Guest OS cause a trap to the VMM
  - Access to kernel region, Update PTBR, CLI/STI, …
  - Change its page tables, …

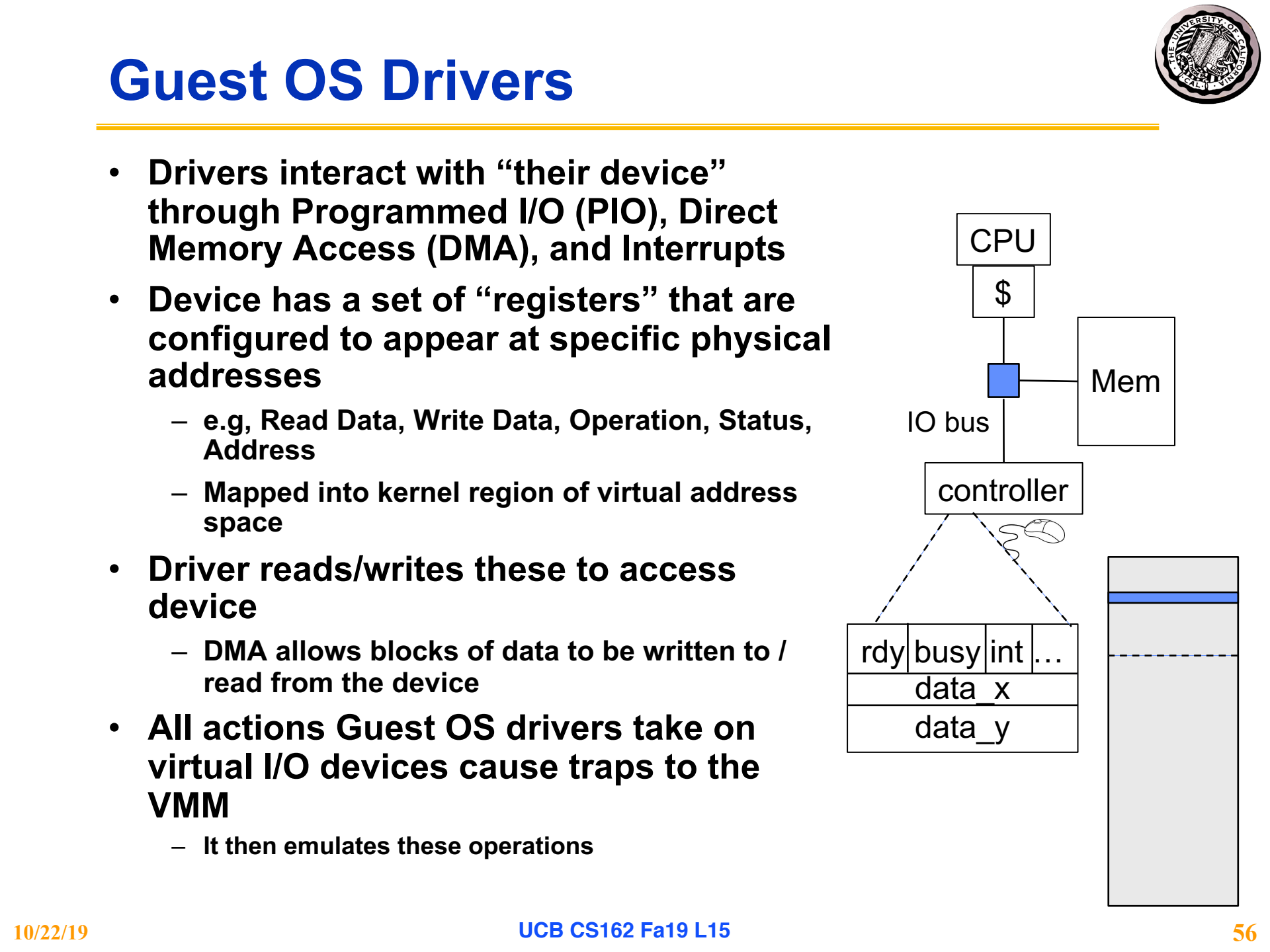
Host Interrupt during VM



Trap and Emulate

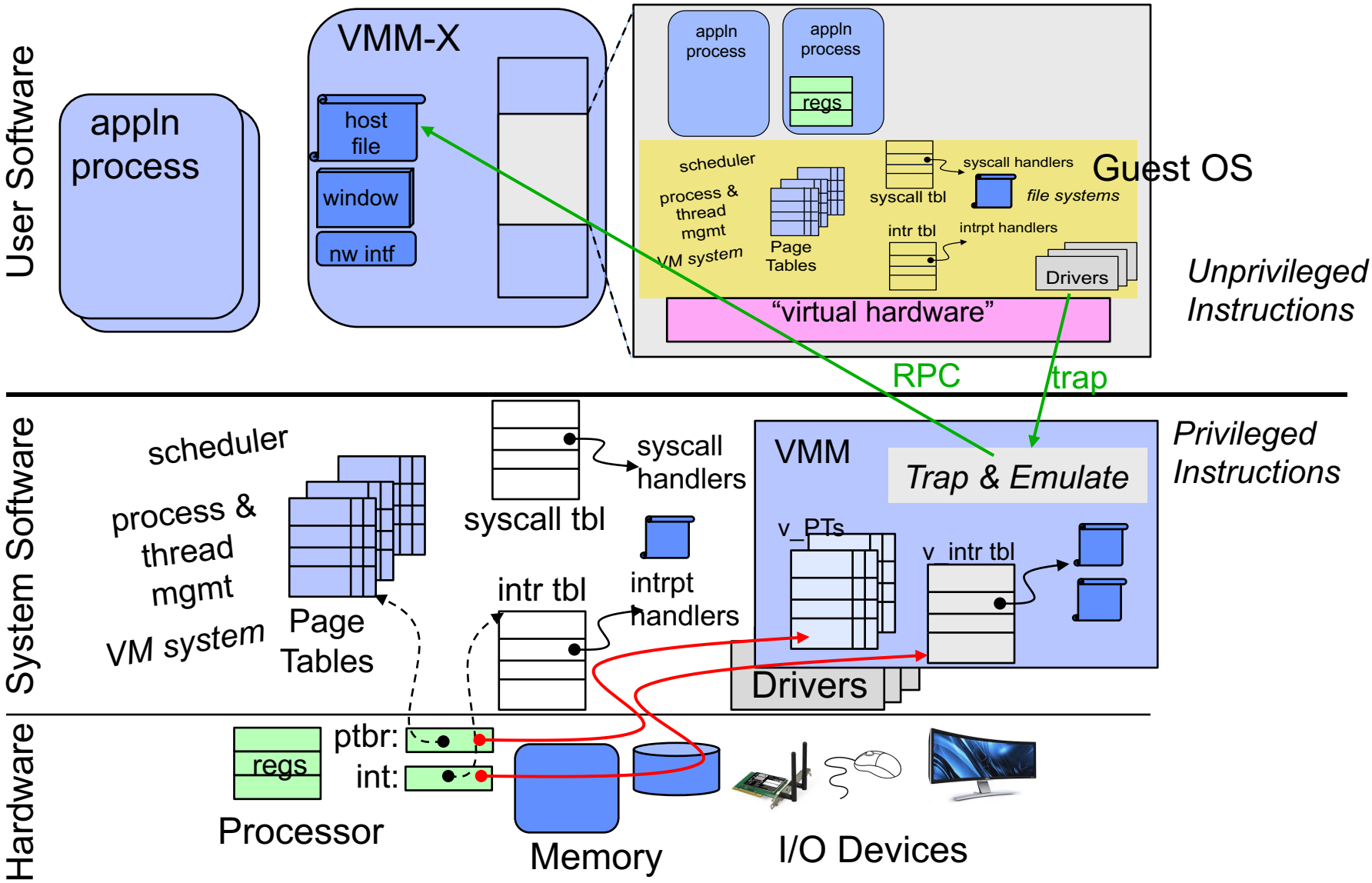
- When Guest OS tries to access it kernel region it traps to VMM
  - Updates to page tables, scheduling threads, switching processes, interrupts,
- The VMM decodes what Guest OS is doing (basic blocks of multiple instructions) and emulates them, updating the Guest OS data structures as if it had done it itself.
- The VMM "sees" everything the Guest OS tries to do and can take action
  - If Guest OS updates a PTE, the VMM updates the shadow PTE
  - If it switches PTs, the VMM switches shadows
  - When it irets to process thread, VMM gives that process the right shadow

Guest OS Drivers



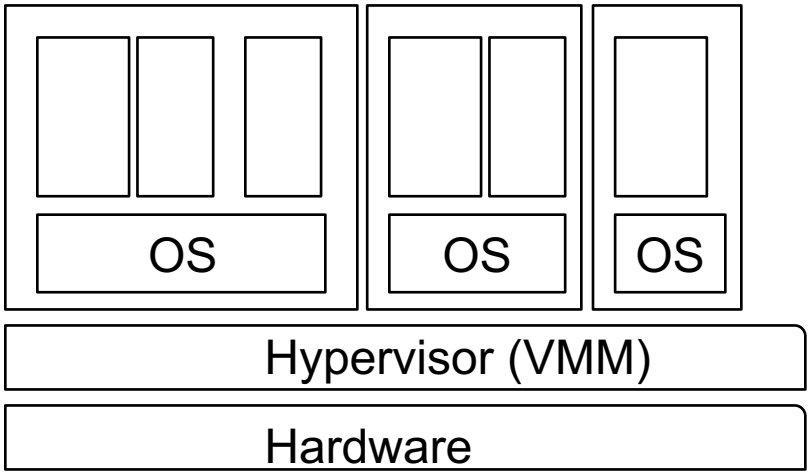
- Drivers interact with "their device" through Programmed I/O (PIO), Direct Memory Access (DMA), and Interrupts
- Device has a set of "registers" that are configured to appear at specific physical addresses
  - e.g, Read Data, Write Data, Operation, Status, Address
  - Mapped into kernel region of virtual address space
- Driver reads/writes these to access device
  - DMA allows blocks of data to be written to / read from the device
- All actions Guest OS drivers take on virtual I/O devices cause traps to the VMM
  - It then emulates these operations

Host Interrupt during VM



Other forms of Virtual Machines

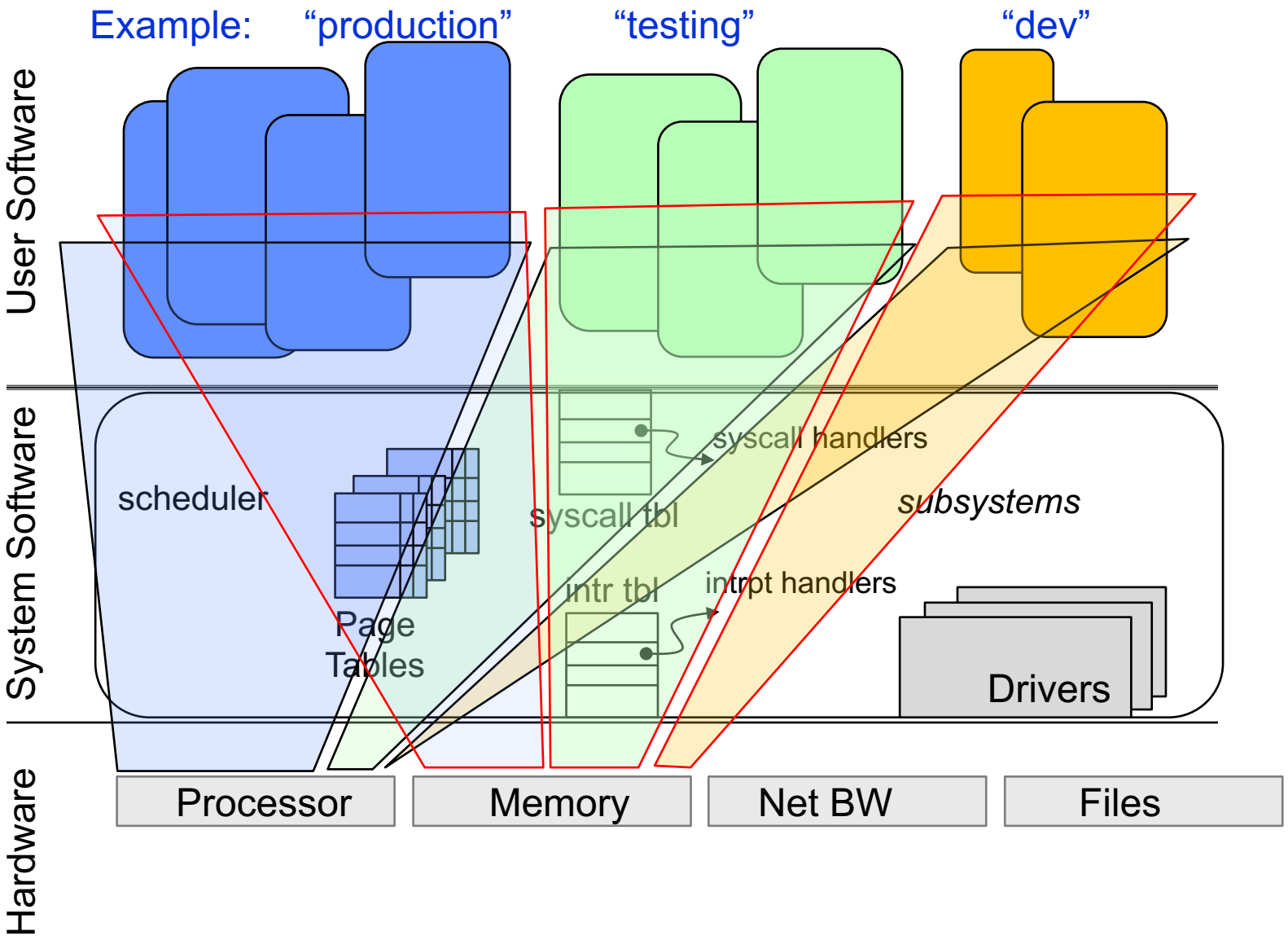
- We have described Hosted Virtual Machines
  - VMware fusion, Virtual Box, KVM, …
- Para-virtualization: Guest OS is modified to work collaboratively with the Host VMM
  - VMM presents simplified machine to Guest OS
- Hypervisor: VMM resides under all the "Guest" OS – peers, none is the special host
  - Much cleaner relationship between VMM and OS's



Stepping Back

- In creating a virtual machine we configured a set of resources within which all of its activities – its OS and all its processes – would operate
  - Total amount of physical memory (the MMAP)
  - Total size of all its disk storage (the file backing its disk)
  - Total network bandwidth
- These constraints are valuable even without a distinct (possibly heterogenous) Guest OS
- Modern operating systems provide *performance isolation*, in addition to traditional protection isolation
  - Without the additional machinery and capability of VMs

Performance Isolation: CGroups



CGroups

- Identify collections of processes that will be treated as a *group* for resource allocation
  - Groups can have hierarchical structure
  - i.e., a Group can be comprised of subgroups
  - Process parent-child relationship defines a hierarchy
    - » Generalize this beyond `fork( )`
- Set of key resource dimensions
  - Processor share (CPU), CPU set (bound to particular cores)
  - Physical memory share,
  - block IO, net priority, net class
  - Namespace (ns), i.e., containers
- Resource limiting, prioritization, accounting and control
- Containers define a collection of libraries and executables that should be a group

How is all the cgroups info recorded?

- Unix-based systems use `/proc` to represent information about processes
  - `/proc/<pid>` describes process <pid>
- `/proc/cgroups/*`
  - `Describes what controllers are implemented`
- Each cgroup controller has a directory under /sys/fs/cgroup/<controller>
  - `/sys/fs/cgroup/cpu/production`
  - `/sys/fs/cgroup/memory/foo/memory.limit_in_bytes`
- Kernel monitors and controls processes/threads in accordance with cgroup controllers

Summary

- For 50+ years operating systems focused on virtualizing resources to provide clean, protected services to user processes
  - Illusion of near infinite resources, despite constraints
  - Protection and isolation, Glue (accounting was there too)
- Past 15+ years, with ever more resources, focus on *predictable* share (proportions) of resources (as part of virtualization)
  - Predictability: Service level agreements focus on 95th or 99th percentile
  - Metering and monitoring to "groups" of processes, as defined by system administrative goals (also CFS, …)
- Virtual machines bring the illusion down to the lowest level and provide an extreme form of resource partitioning
  - Re-purpose the mechanisms used for protection, isolation and system extension (drivers) to permit near-perfect emulation
  - Intricate interplay of (in kernel) Virtual Machine Monitor and (user level) VMM-X process to allow Guest OS as if on Physical Machine and Guest OS User Processes as if on Guest OS
    - » VMM interposes between hardware/Host OS and Guest OS (Page Table & Interrupt Tbl)
- Shadow Page Table (composition of two PTs) to translate Guest OS process VAS => MMAP region => Physical
- Trap and Emulate
- Control Groups & Containers provide resource limiting w/o VM

If you want to dig further

- Bringing Virtualization to the x86 Architecture with the Original VMware Workstation. E. Bugnion, S. Devine, M. Rosenblum, J. Sugerman, E. Y. Wang, ACM Trans. Comput. Syst. 2012.
  - Extremely detailed look at what it takes to pull off VM in older x86
  - Good set of references
  - Newer systems have better hardware support for virtualization (e.g., KVM)

- Xen and the Art of Virtualization, 2003 (paravirtualization)
- Disco: Running Commodity Operating Systems on Scalable Multiprocessors (1997) – revived concept of VMMs largely lost since the 70s