



# Lecture 15: Virtual Memory

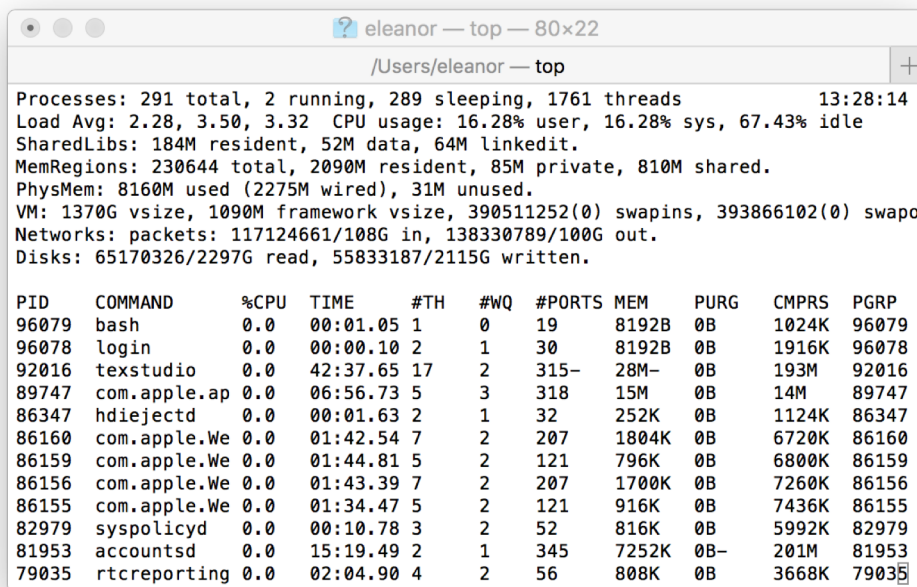
---

CS 105

October 29, 2019

# Multiprocessing

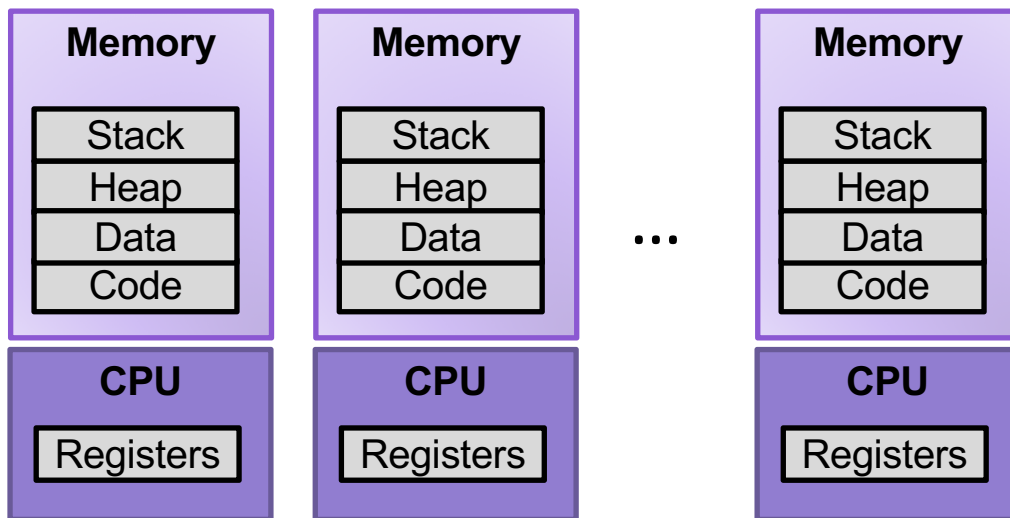
- Computer runs many processes simultaneously
- Running program “top” on Mac
  - System has 123 processes, 5 of which are active
  - Identified by Process ID (PID)



```
eleanor — top — 80x22
/Users/eleanor — top
Processes: 291 total, 2 running, 289 sleeping, 1761 threads      13:28:14
Load Avg: 2.28, 3.50, 3.32  CPU usage: 16.28% user, 16.28% sys, 67.43% idle
SharedLibs: 184M resident, 52M data, 64M linkedit.
MemRegions: 230644 total, 2090M resident, 85M private, 810M shared.
PhysMem: 8160M used (2275M wired), 31M unused.
VM: 1370G vsize, 1090M framework vsize, 390511252(0) swapins, 393866102(0) swapo
Networks: packets: 117124661/108G in, 138330789/100G out.
Disks: 65170326/2297G read, 55833187/2115G written.

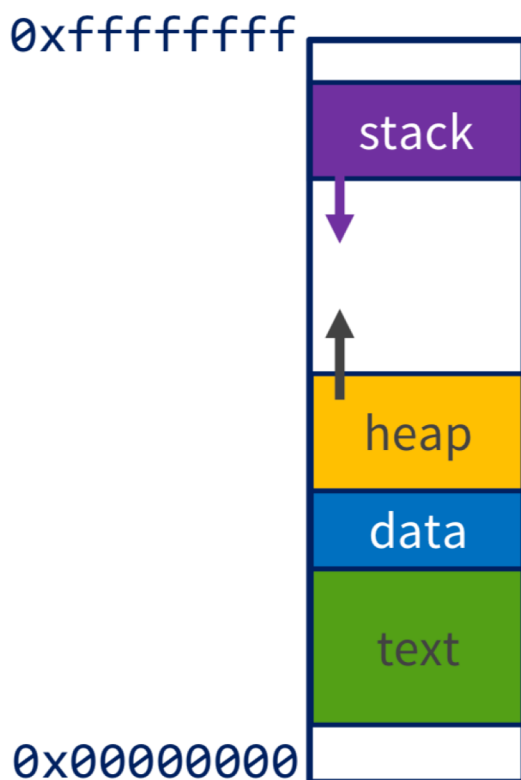
PID   COMMAND      %CPU  TIME    #TH   #WQ   #PORTS MEM    PURG   CMPRS  PGRP
96079  bash         0.0   00:01.05 1     0     19     8192B  0B     1024K 96079
96078  login        0.0   00:00.10 2     1     30     8192B  0B     1916K 96078
92016  texstudio    0.0   42:37.65 17    2     315-   28M-   0B     193M  92016
89747  com.apple.ap 0.0   06:56.73 5     3     318    15M    0B     14M   89747
86347  hdiejectd   0.0   00:01.63 2     1     32     252K   0B     1124K 86347
86160  com.apple.We 0.0   01:42.54 7     2     207    1804K  0B     6720K 86160
86159  com.apple.We 0.0   01:44.81 5     2     121    796K   0B     6800K 86159
86156  com.apple.We 0.0   01:43.39 7     2     207    1700K  0B     7260K 86156
86155  com.apple.We 0.0   01:34.47 5     2     121    916K   0B     7436K 86155
82979  syspolicyd  0.0   00:10.78 3     2     52     816K   0B     5992K 82979
81953  accountsd   0.0   15:19.49 2     1     345    7252K  0B-    201M  81953
79035  rtcreporting 0.0   02:04.90 4     2     56     808K   0B     3668K 79035
```

# Multiprocessing: The Illusion



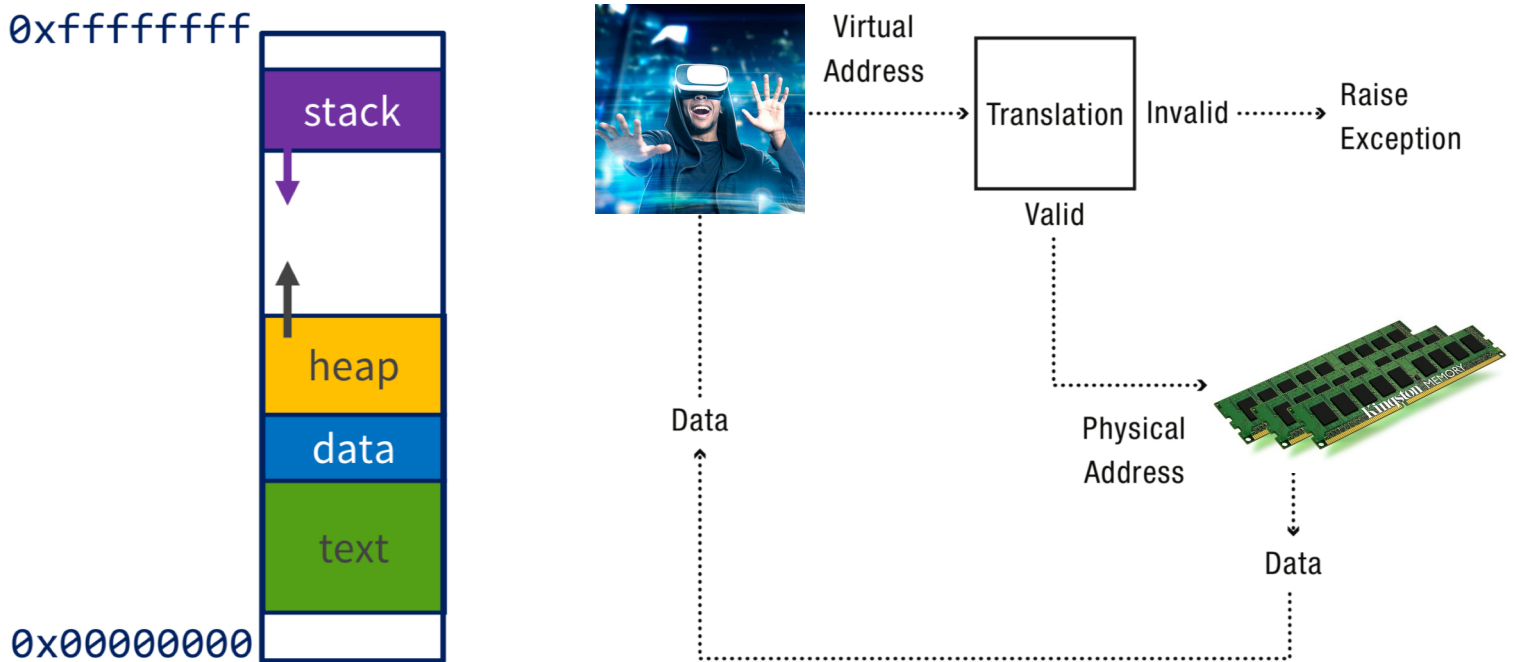
- Process provides each program with two key abstractions:
  - **Logical control flow**
    - Each program seems to have exclusive use of the CPU
    - Provided by kernel mechanism called **context switching**
  - **Private address space**
    - Each program seems to have exclusive use of main memory.
    - Provided by kernel mechanism called **virtual memory**

# Virtual Memory Goals



- **Isolation:** don't want different process states collided in physical memory
- **Utilization:** want best use of limited resource
- **Virtualization:** want to create illusion of more resources
- **Efficiency:** want fast reads/writes to memory
- **Sharing:** want option to overlap for communication

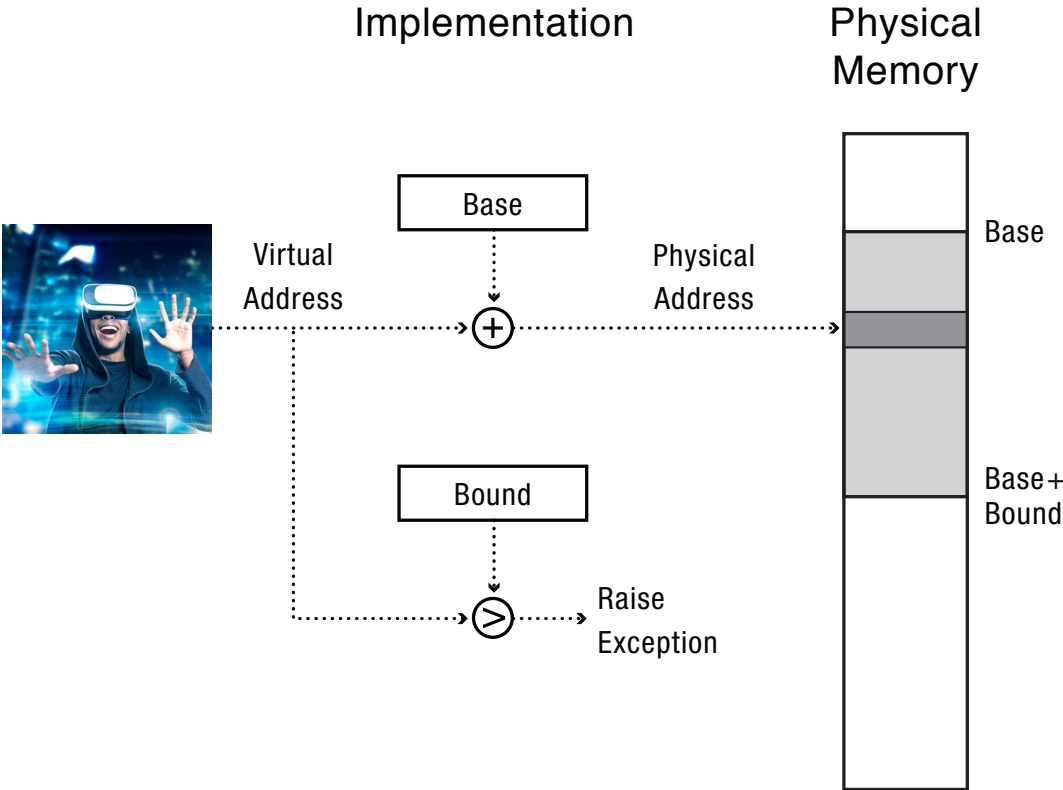
# Address Translation



# Base-and-Bound



# Base-and-Bound



# Segmentation





# Segmentation

## Implementation



Virtual Address

Segment	Offset
---------	--------

Segment Table

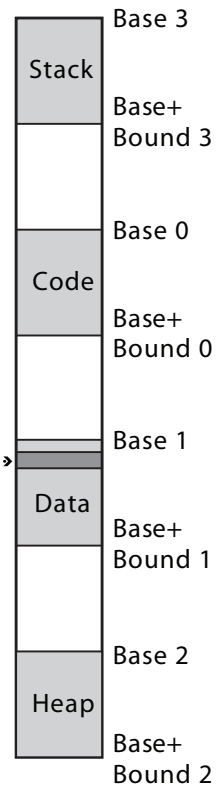
Base	Bound	Access
		Read
		R/W
		R/W
		R/W

+

Physical Address

Raise Exception

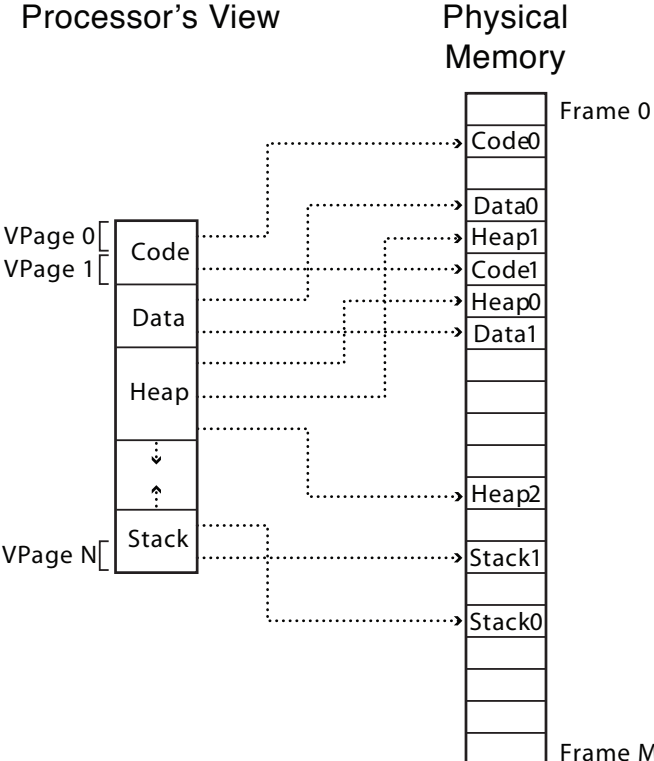
## Physical Memory



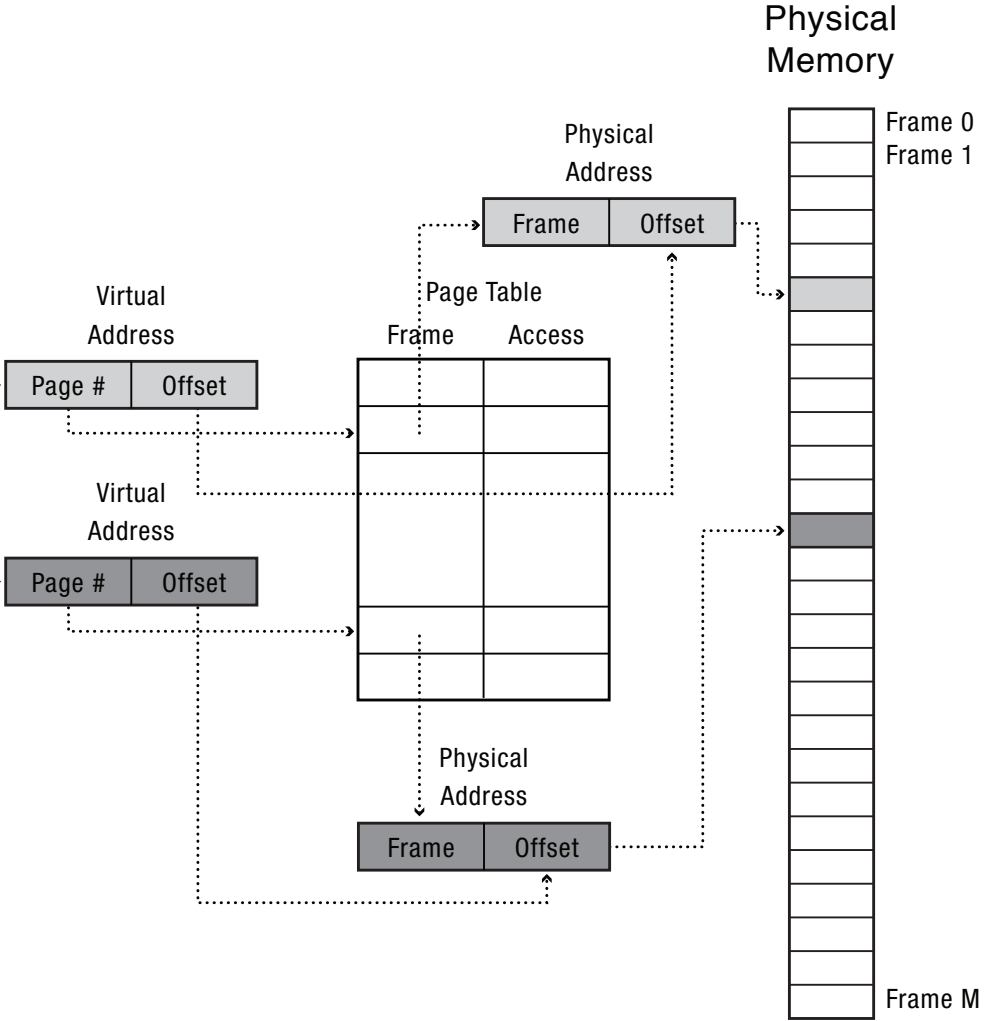
# Paging



# Paging



# Paging



# Memory as a Cache

- each page table entry has a valid bit
- for valid entries, frame indicates physical address of page in memory
- a **page fault** occurs when a program requests a page that is not currently in memory
  - takes time to handle, so context switch
  - evict another page in memory to make space (which one?)

# Page Replacement Algorithms

- **Random:** Pick any page to eject at random
  - Used mainly for comparison
- **FIFO:** The page brought in earliest is evicted
  - Ignores usage
- **OPT:** Belady's algorithm
  - Select page not used for longest time
- **LRU:** Evict page that hasn't been used for the longest
  - Past could be a good predictor of the future
- **MRU:** Evict the most recently used page
- **LFU:** Evict least frequently used page



# Thrashing

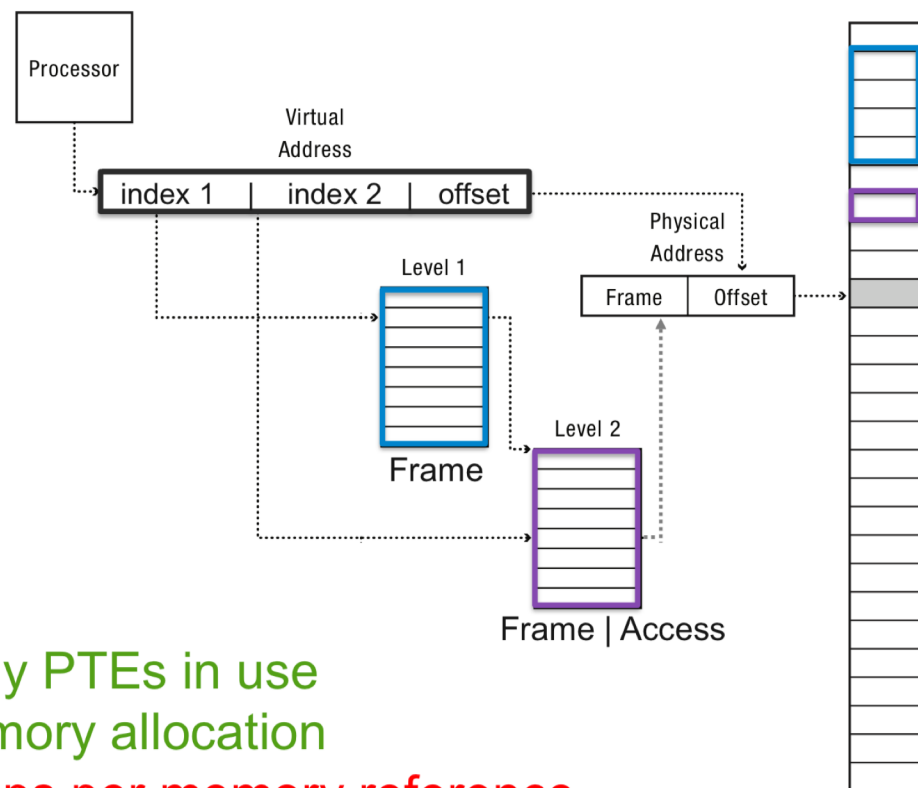
- working set is the collection of a pages a process requires in a given time interval
- if it doesn't fit in memory, program with thrash

# Efficient Paging

- How big should our pages be?
  - How much internal fragmentation will there be?
  - How big is the page table?
    - Example: consider 64-bit address space, 4KB ( $2^{12}$ ) page size, assume each page table entry is 8 bytes.
- **Performance:** every data/instruction access requires *two* memory accesses:
  - One for the page table
  - One for the data/instruction



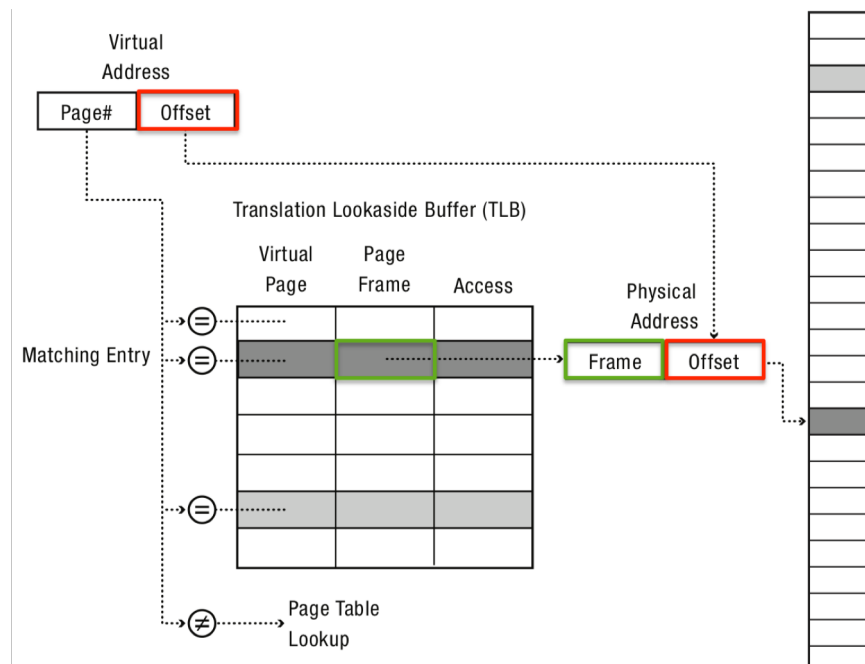
# Multi-level page tables



- + Allocate only PTEs in use
- + Simple memory allocation
- **more** lookups per memory reference

# Translation Look-aside Buffer (TLB)

- **Translation lookaside buffer (TLB):** special cache for page table entries



# Example: Core i7 Address Translation

