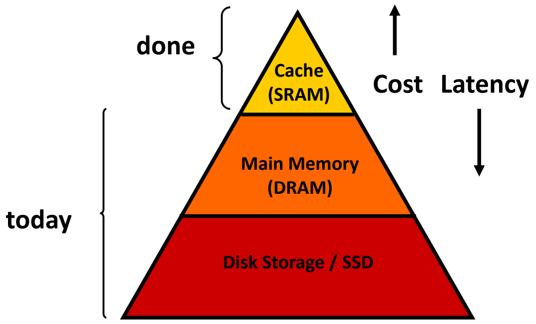
EECS 370

Virtual Memory Basics

Agenda

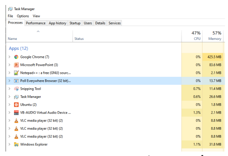
- **Motivation**
- Virtual Memory Principles
- Page Tables
- Class Problem

Storage Hierarchy



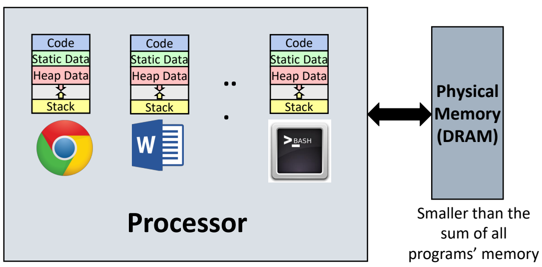
Memory: Two Issues

1. We've been working with the abstraction that all programs have full, private access to memory
   - But in practice, multiple programs run at the same time!



   - What happens if two programs try to write to the same memory address??

Revisit real system view—multitasking



Memory: Two Issues

2. Even if only one program is running, modern computers have 48-64 bit address spaces!
   - No computer actually has 18 exabytes (18 billion GBs)
   - What if a computer tries to write to address 0xFFFF…FFFF
   - Should it just crash??

Memory: Two Issues

- Modern systems use the same solution for both problems: **virtual memory**
  - In a nutshell, each program thinks it has full, private access to memory (it can safely index any address from 0x0-0xFFF…FFFF)
  - Hardware and software transparently maps these addresses to distinct addresses in DRAM and in hard disk / SSD
  - Focus for the next 3 lectures

Agenda

- Motivation
- **Virtual Memory Principles**
- Page Tables
- Class Problem

Basics of Virtual Memory

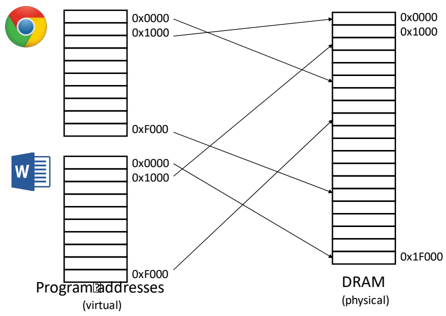
- Any time you see the word virtual in computer science and architecture it means "using a level of indirection".
  - Examples?

- Virtual memory hardware changes the virtual address the programmer sees into the physical one the memory chips see.

0x800 → **0x3C00**

Virtual address **Physical address**

Virtual Memory



How to Translate Addresses?

- Address Translation is not done entirely in hardware
- We'll get help in software via the **operating system**
- The operating system is a special set of programs
  - Always running (after the system boots)
  - Is in charge of **managing the hardware resources** for all other running programs
  - E.g. initializing memory for a starting program, managing the file system, choosing when a particular program gets to run…
  - … and translating virtual addresses into physical addresses!
- OS handles address translation by maintaining a data structure in main memory: **the page table**

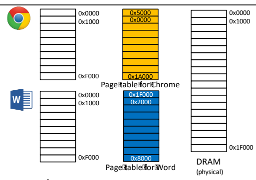
Virtual memory terminology

- Memory is divided into fixed-size chunks called **Pages** (e.g., 4KB for x86)
  - Size of physical page = size of virtual page
  - A virtual address consists of
    - A virtual page number
    - A page offset field (low order bits of the address)
  - Translating a virtual address into a physical address only requires translating the page numbers
    - The page offset will stay the same

	Virtual page number	Page offset
Virtual address	31 …	11 … 0

	Physical page number	Page offset
Physical address	?? …	11 … 0

Page Table

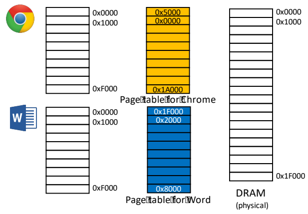


- Translate page numbers using **page tables**
- Contains address translation information, i.e. virtual page # ➔ physical page #
- Each process has its own page table
  - Maintained by operating system (OS)
- Page tables themselves are kept in memory by OS, and OS knows the physical address of the page tables
  - No address translation is required by the OS for accessing the page tables

Poll: What is the cost of this scheme? (select all that apply)
a) Uses up more memory
b) Takes longer to do loads and stores
c) Fewer addresses accessible by program
d) None of the above

Why Pages?

- Why have the idea of "pages"? Why not just have a mapping for each individual address?
  - Equivalent to asking: "why not have pages of size 1 byte?"
  - Otherwise - need a mapping entry for every single element of memory
  - The mapping data would take up as much space as the actual program data!
  - Also would screw up spatial locality of cache blocks (things contiguous in virtual memory wouldn't be contiguous in physical memory)

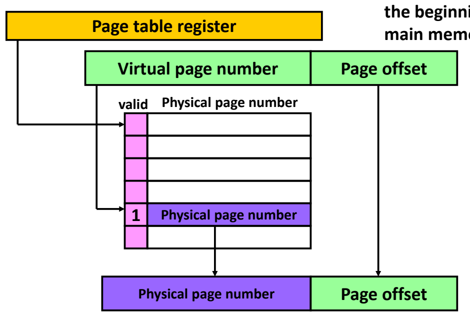


Agenda

- Motivation
- Virtual Memory Principles
- **Page Tables**
- Class Problem

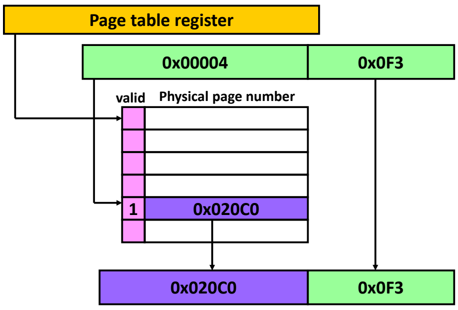
Page table components

The *Page table register* points to the beginning of the page table in main memory



Page table components - Example

Virtual address = 0x000040F3

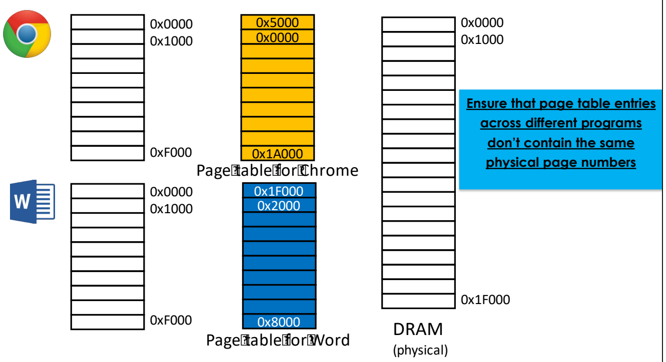


Physical address = 0x020C00F3

Virtual Memory Goals

- VM should provide the following 3 capabilities to the programs:
  1. **Transparency**
     - Don't need to know how other programs are using memory
  2. **Protection**
     - No program can access the data of any other program
  3. **Programs not limited by DRAM capacity**
     - Each program can have more data than DRAM size

1-2: How to achieve transparency & protection?



3. How to be not limited by DRAM capacity?

- Use disk as temporary space in case memory capacity is exhausted
  - This temporary space in disk is called swap partition in Linux-based systems
  - For fun check swap space in a linux system by:

  $: top

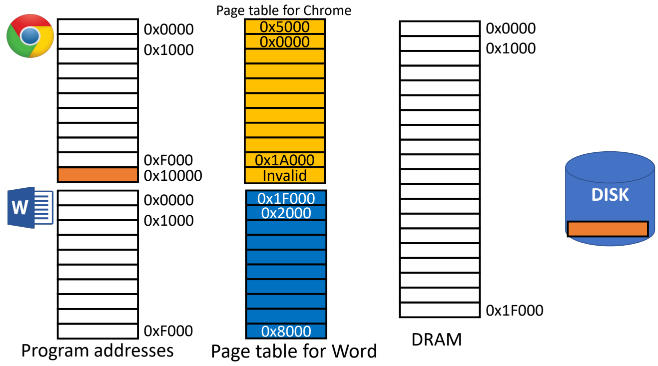
```
Tasks: 662 total,   1 running, 661 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.0 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  32704372 total, 10813444 used, 21890928 free,  1018840 buffers
KiB Swap: 35162108 total,    89248 used, 35072860 free.  7053764 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
60256 nehaag    20   0   25356   3356   2444 R   6.0  0.0   0:00.02 top
    1 root      20   0   38424   9040   2780 S   0.0  0.0   1:56.96 init
    2 root      20   0       0      0      0 S   0.0  0.0   0:02.21 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   6:20.75 ksoftirqd/0
```

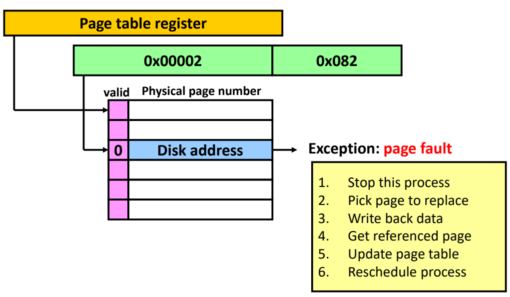

2. How to be not limited by DRAM capacity?

- We can mark a page table entry as "Invalid", indicating that the data for that page doesn't exist in main memory, but instead is located on the disk
- Looking up a page table entry that corresponds to disk is called a **page fault**

2. How to be not limited by DRAM capacity?



Page faults



How do we find it on disk?

- That is not a hardware problem! Go take EECS 482! ☺
- This is the operating system's job. Most operating systems partition the disk into logical devices (C: , D: , /home, etc.)
- They also have a hidden partition to support the disk portion of virtual memory
  - Swap partition on UNIX machines
  - You then index into the correct page in the swap partition.

Agenda

- Motivation
- Virtual Memory Principles
- Page Tables
- **Class Problem**

Class Problem

- Given the following:
  - 4KB page size, physical memory of 16KB, page table stored in physical page 0 and can never be evicted, 20 bit, byte-addressable virtual address space.
  - The page table initially has virtual page 0 in physical page 1, virtual page 1 in physical page 2 and no valid data in other physical pages.
- Fill in the table on the next slide for each reference
  - Note: like caches we'll use LRU when we need to replace a page.

Class Problem (continued)

4KB page size, physical memory of 16KB, page table stored in physical page 0 and can never be evicted, 20 bit, byte-addressable virtual address space.

The page table initially has virtual page 0 in physical page 1, virtual page 1 in physical page 2 and no valid data in other physical pages.

4KB → 12 bit → 3 hex offset

4 pages

(virtual page table size: $2^{20-12} = 256$ (entries))

Virt addr	Virt page	Page fault?	Phys addr
0x00F0C	0	N	1~ (0 occupied)
0x01F0C	1	N	2~
0x20F0C	20 → 3	Y	3~
0x00100	0	N	1~
0x00200	0	N	1~
0x30000	30 → 3	Y	2~ (替换, LRU)
0x01FFF	1	Y	3~ (替换, LRU)
0x00200	0	N	1~

Poll: How many hex digits should the page number be?

Class Problem (continued)

4KB page size, physical memory of 16KB, page table stored in physical page 0 and can never be evicted, 20 bit, byte-addressable virtual address space.

The page table initially has virtual page 1 in physical page 1, virtual page 1 in physical page 2 and no valid data in other physical pages.

Virt addr	Virt page	Page fault?	Phys addr
0x00F0C	0x0	N	0x1F0C
0x01F0C	0x1	N	0x2F0C
0x20F0C	0x20	Y (into 3)	0x3F0C
0x00100	0x0	N	0x1100
0x00200	0x0	N	0x1200
0x30000	0x30	Y (into 2)	0x2000
0x01FFF	0x1	Y (into 3)	0x3FFF
0x00200	0x0	N	0x1200

Size of the page table

- How big is a page table entry?
  - For 32-bit virtual address:
    - If the machine can support 1GB = 2^{30} bytes of physical memory and we use pages of size 4KB = 2^{12},
    - then the physical page number is 30-12 = 18 bits.
      Plus another valid bit + other useful stuff (read only, dirty, etc.)
    - Let say about 3 bytes.
- How many entries in the page table?
  - 1 entry per virtual page
  - ARM virtual address is 32 bits – 12 bit page offset = 20
  - Total number of virtual pages = 2^{20}
- Total size of page table = Number of virtual pages
  * Size of each page table entry
  = $2^{20} \times 3$ bytes ~ 3 MB