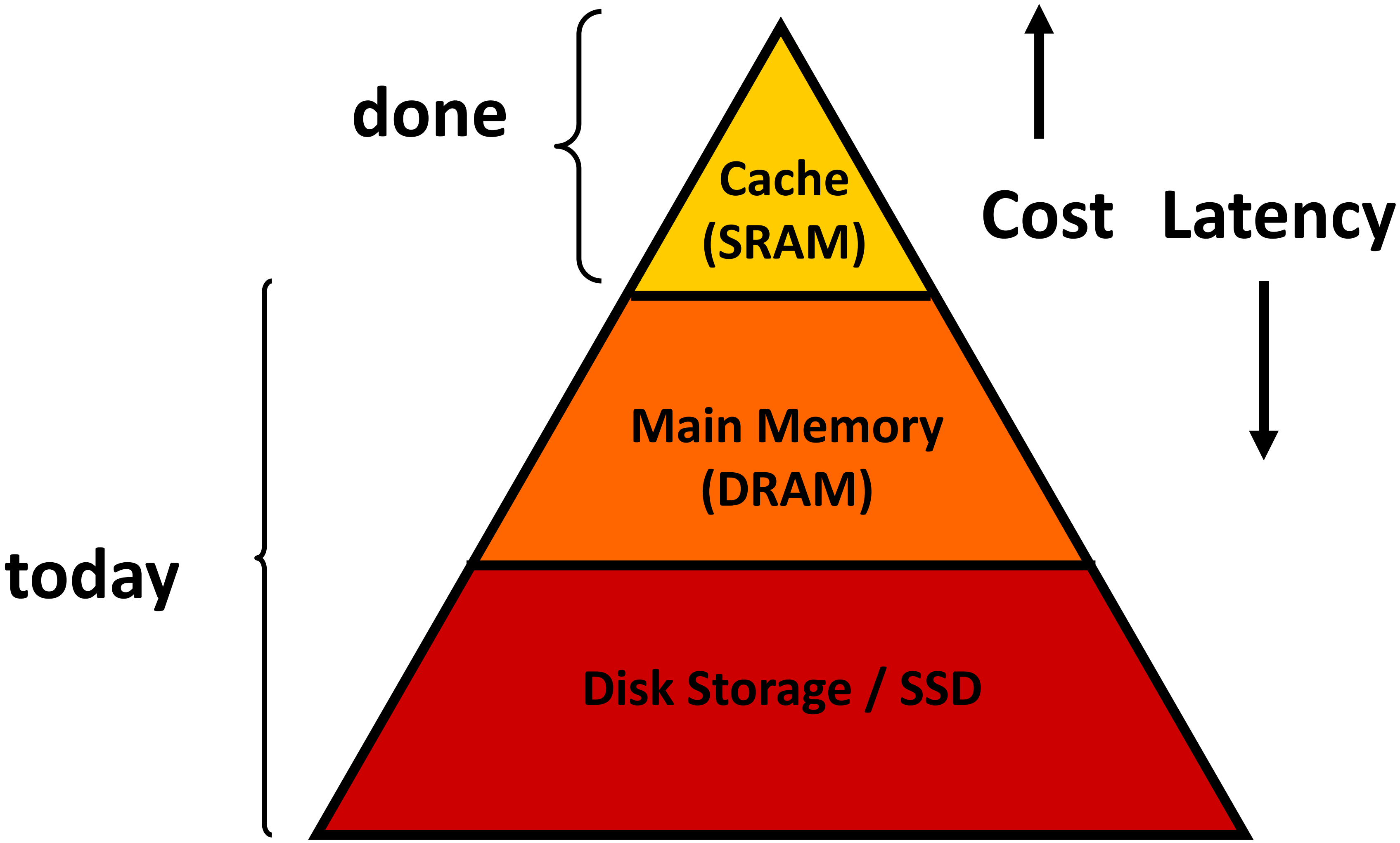
EECS 370 - Lecture 22

Virtual Memory Basics

Announcements

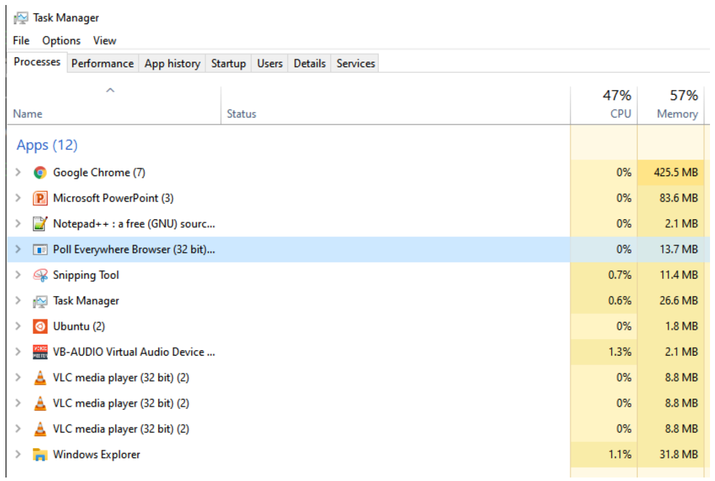
- P4
  - Last project!
  - Due Thu (4/13)
- HW 6
  - Last homework!
  - Due Monday (4/17)
- Final exam
  - …Last exam!
  - Thu (4/20) @ 10:30 am

Storage Hierarchy



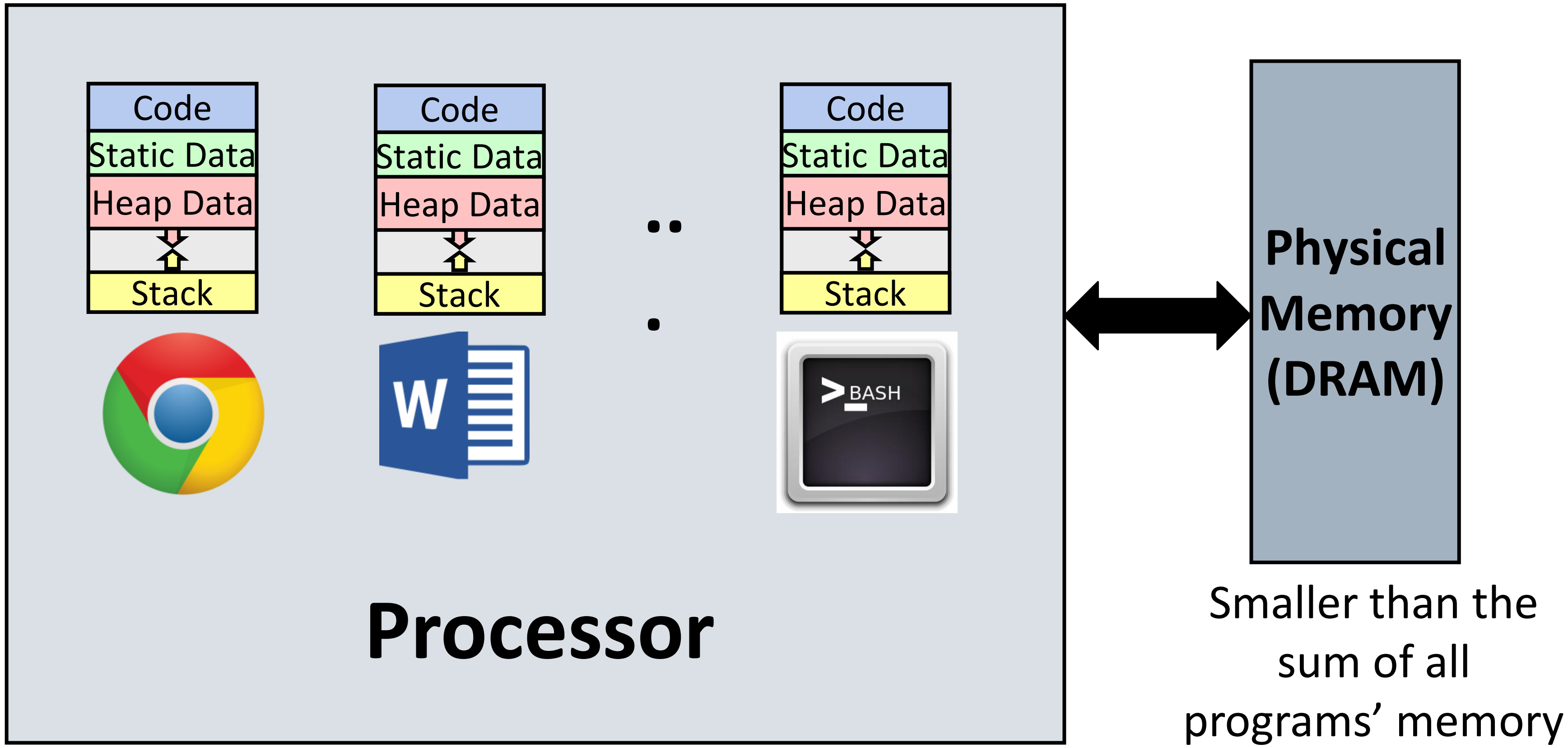
Memory: Two Issues

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

Basics of Virtual Memory

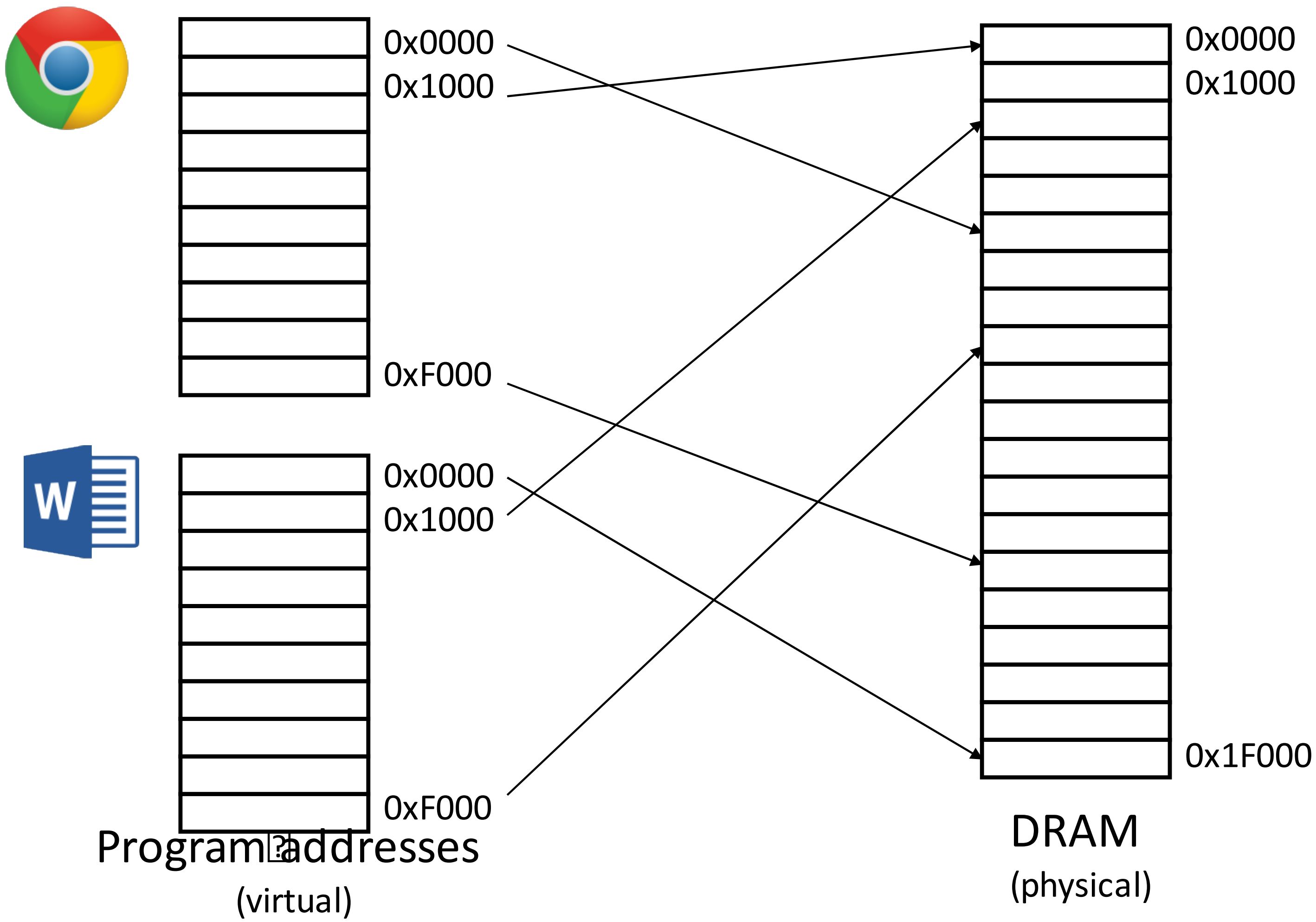
- Any time you see the word virtual in computer science and architecture it means "using a level of indirection".
  - Examples?

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

0x800 → **Disk ID 803C4**

Virtual address **Physical address**

Virtual Memory



Virtual memory terminology

- Divide memory in chunks of **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

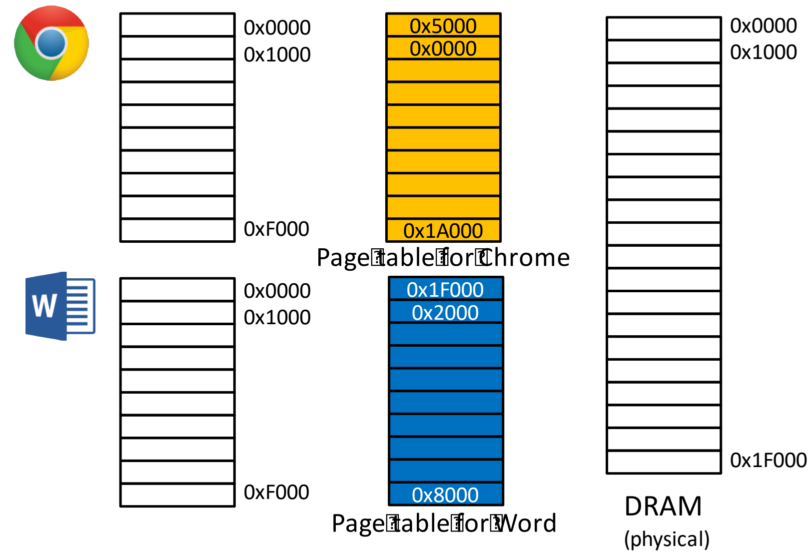
Virtual address

Virtual page number	Page offset
31	11 … 0

Physical address

Physical page number	Page offset
31	11 … 0

Page Table



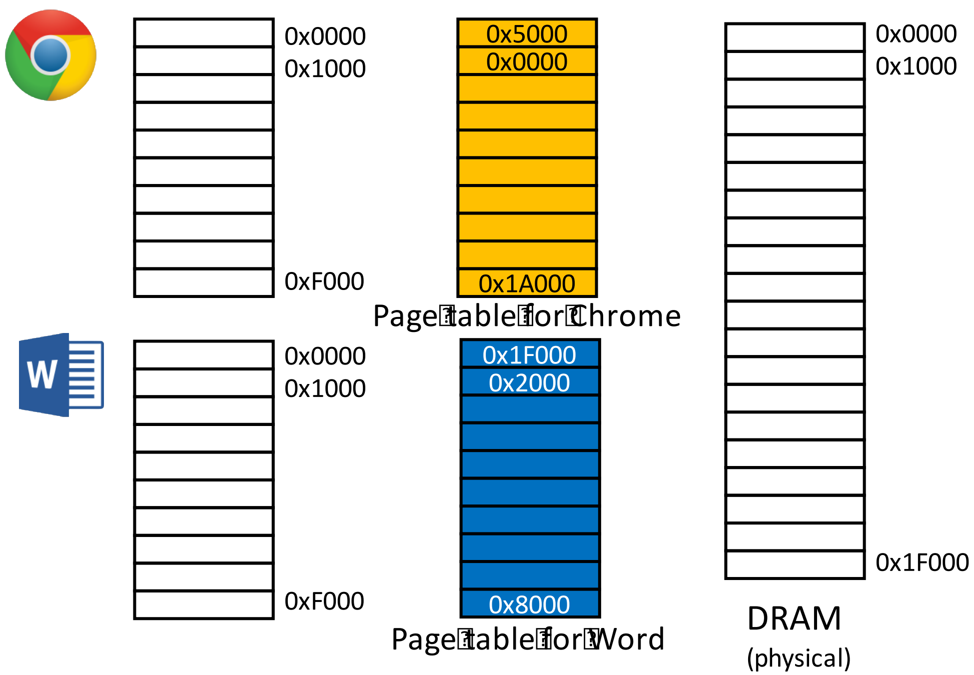
- Translate page numbers using **page tables**
- Contains address translation information, i.e. virtual page # ➔ physical page #
- Each process has its own page table
  - Maintained by operating system (OS)
- Page tables themselves are kept in memory by OS, and OS knows the physical address of the page 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

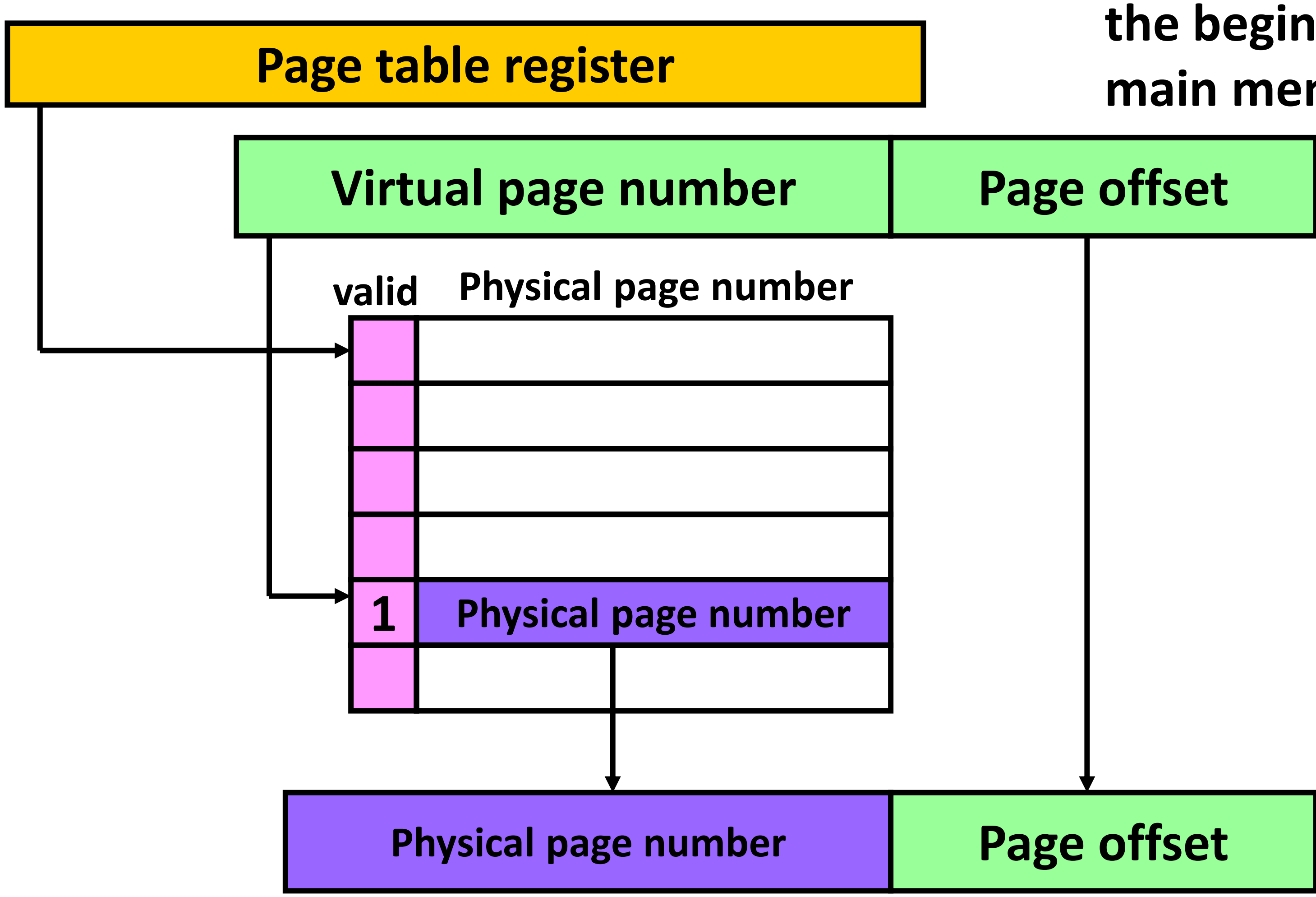
Why Pages?

- 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)

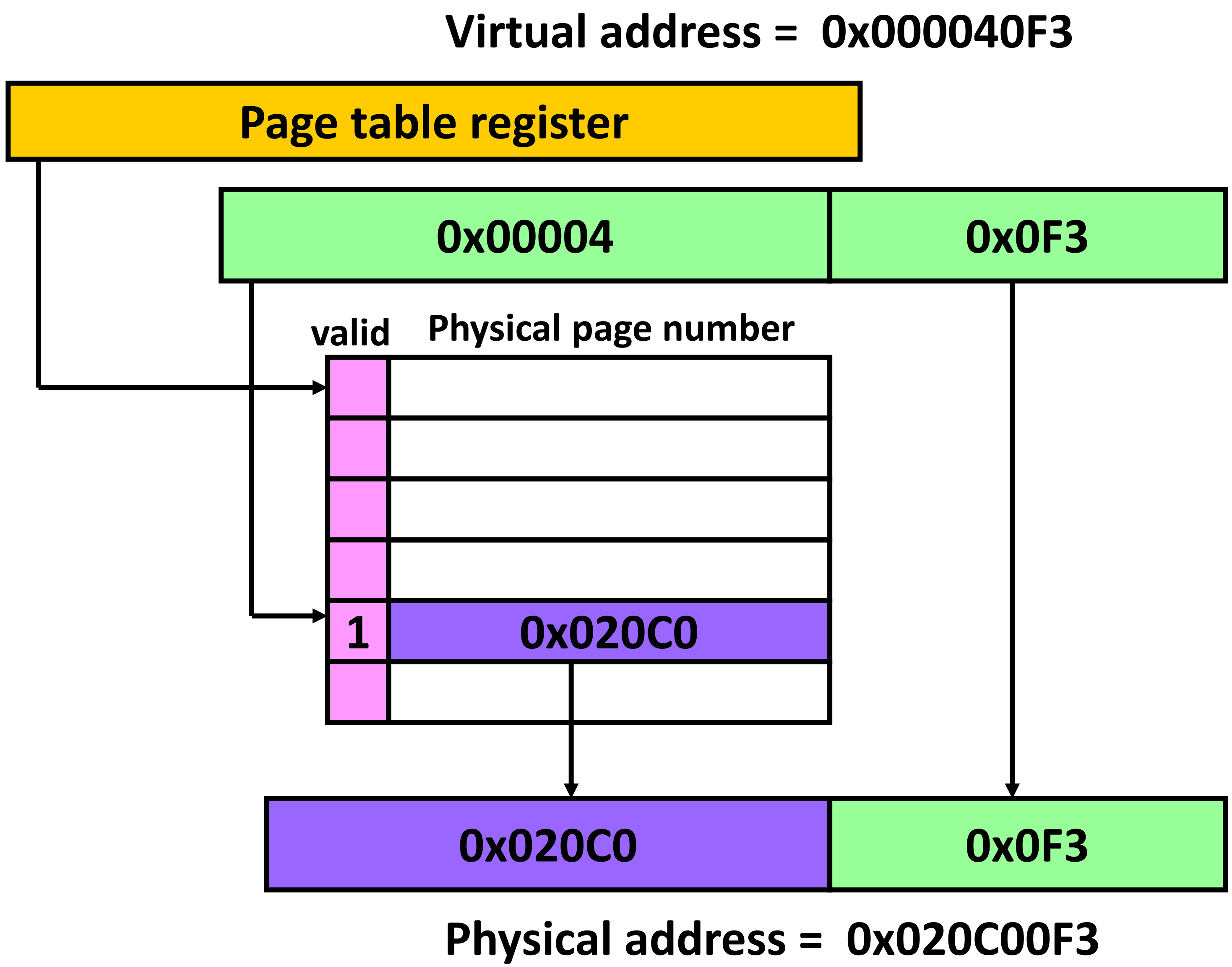


Page table components

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



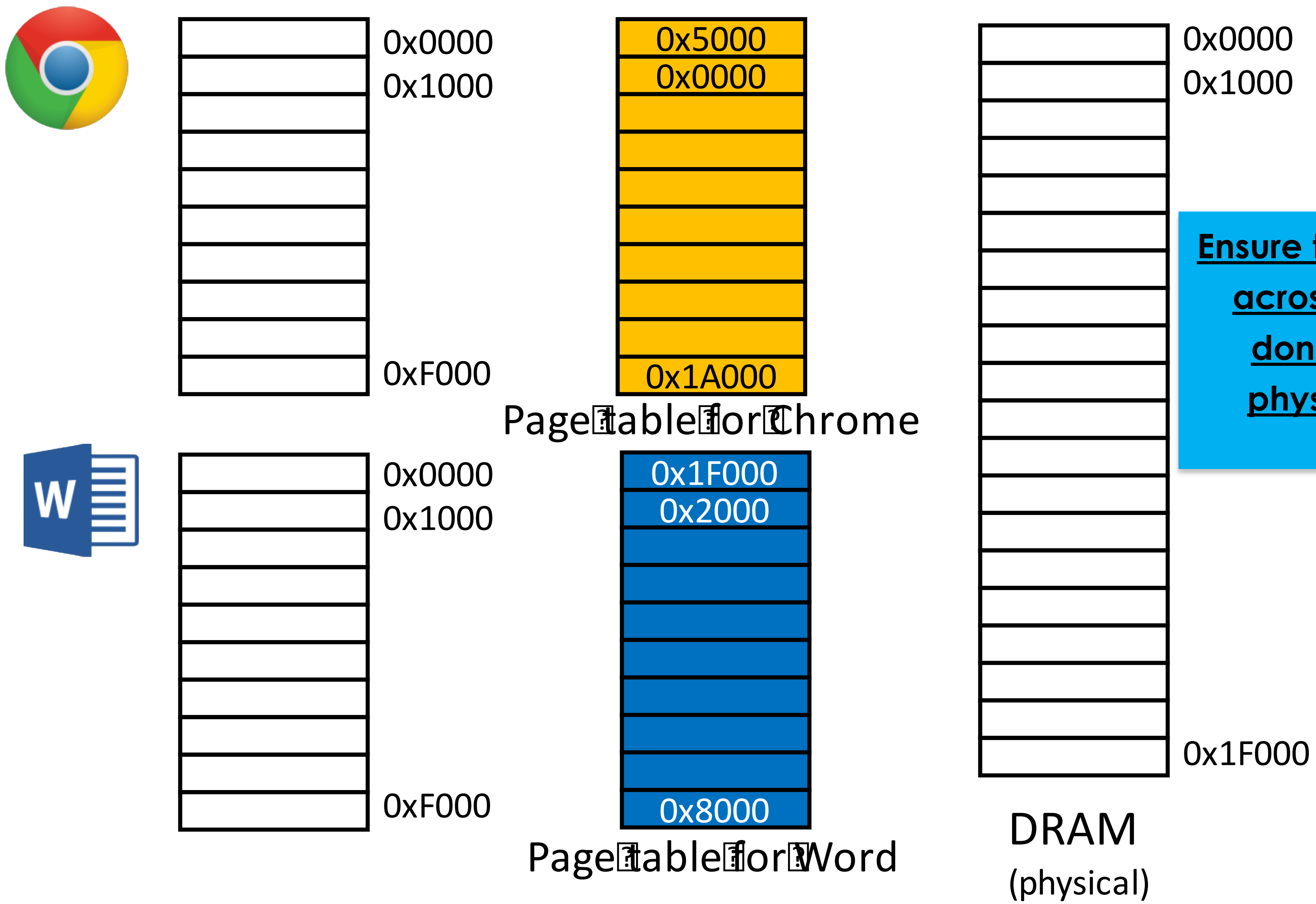
Page table components - Example



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 modify the data of any other program
  3. **Programs not limited by DRAM capacity**
     - Each program can have more data than DRAM size

1. How to achieve transparency & protection?



Ensure that page table entries across different programs don't contain the same physical page numbers

2. 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`

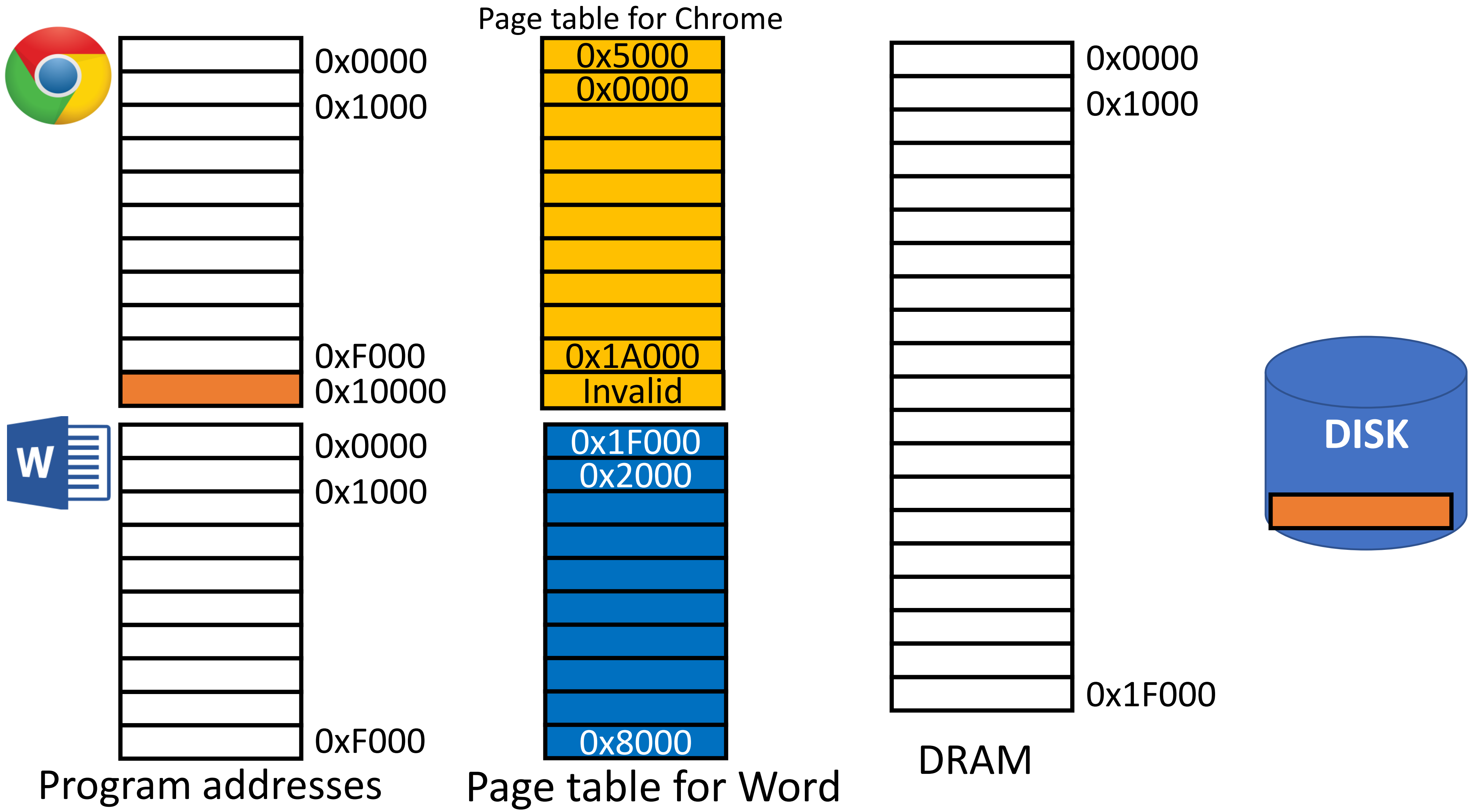
```
1. ssh
Tasks: 662 total,   1 running, 661 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.0 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  32704372 total, 10813444 used, 21890928 free,  1018840 buffers
KiB Swap: 35162108 total,    89248 used, 35072860 free.  7053764 cached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

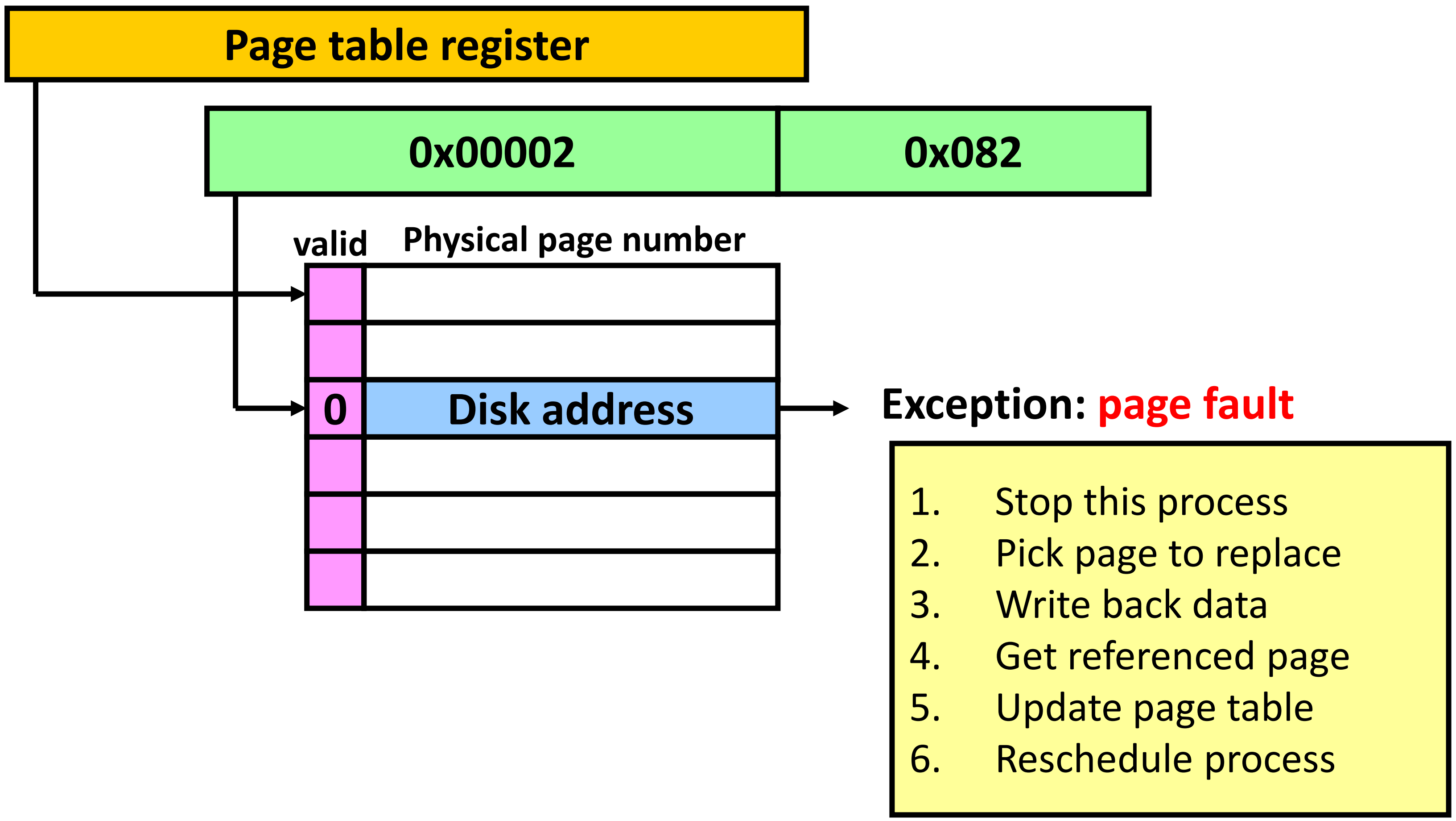
2. How to be not limited by DRAM capacity?

- 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



Page table register
0x00002
0x082
valid
Physical page number
0
Disk address
Exception: page fault
1. Stop this process
2. Pick page to replace
3. Write back data
4. Get referenced page
5. Update page table
6. Reschedule process

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.

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.

Virt addr	Virt page	Page fault?	Phys addr
0x00F0C			
0x01F0C			
0x20F0C			
0x00100			
0x00200			
0x30000			
0x01FFF			
0x00200			

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 0 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			
0x20F0C			
0x00100			
0x00200			
0x30000			
0x01FFF			
0x00200			

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.

Virt addr	Virt page	Page fault?	Phys addr
0x00F0C	0x0	N	0x1F0C
0x01F0C	0x1	N	0x2F0C
0x20F0C			
0x00100			
0x00200			
0x30000			
0x01FFF			
0x00200			

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.

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

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.

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

Next time

- Improving Virtual Memory Design
  - Multi-level page-tables
- Lingering questions / feedback? I'll include an anonymous form at the end of every lecture: https://bit.ly/3oXr4Ah

Extra Slides

Size of the page table

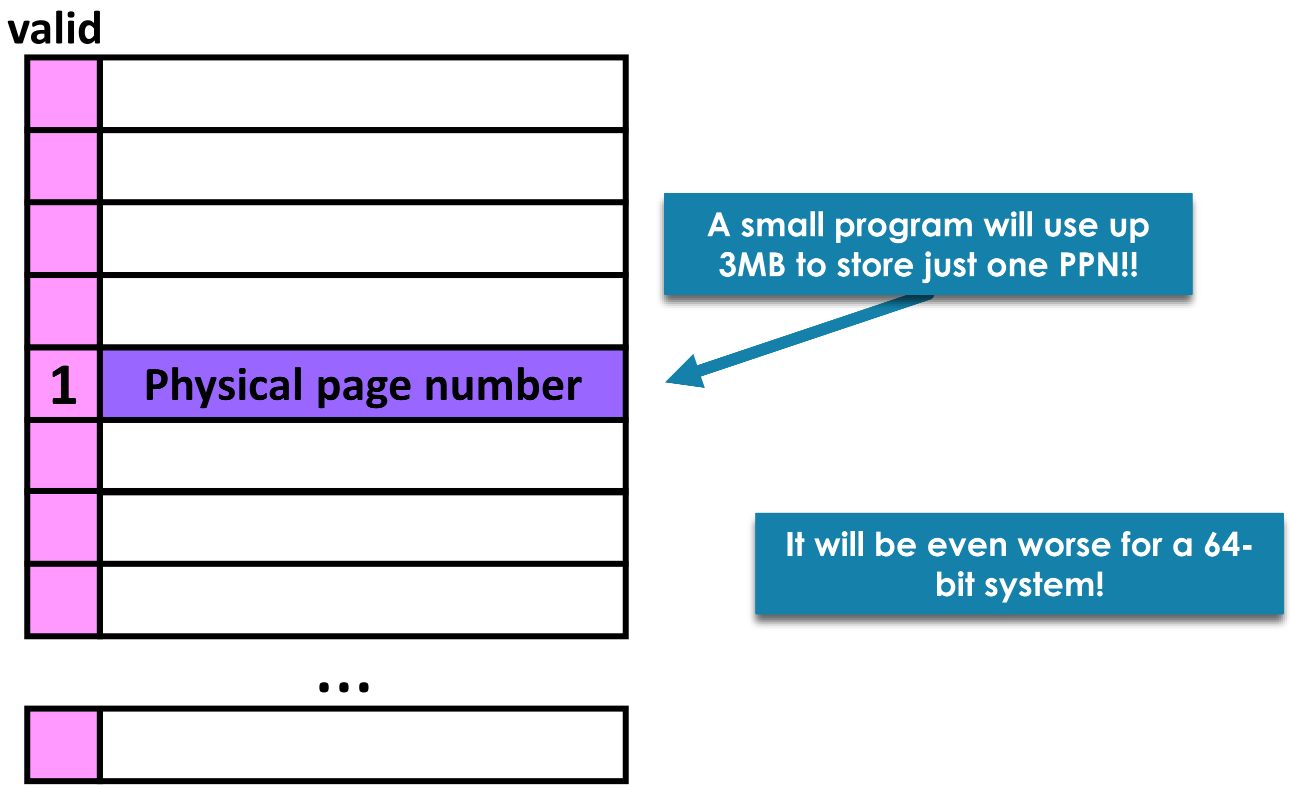
- How big is a page table entry?
  - For 32-bit virtual address:
    - If the machine can support 1GB = 2^{30} bytes of <u>physical</u> memory and we use pages of size 4KB = 2^{12},
    - then the physical page number is 30-12 = 18 bits.
      Plus another valid bit + other useful stuff (read only, dirty, 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

How can you organize the page table?

1. Single-level page table occupies continuous region in physical memory
   - Previous example always takes 3MB regardless of how much virtual memory is used



How can you organize the page table?

2. Option 2: Use a multi-level page table
   - 1st level page table (much smaller!) holds addresses 2nd level page tables
     - 2nd level page tables hold translation info, or 3rd level page tables if we wanna go deeper
     - Only allocate space for 2nd level page tables that are used

valid	PPN
1	0x1
1	0x2
1	0x3

Single-level: Tons of wasted space!

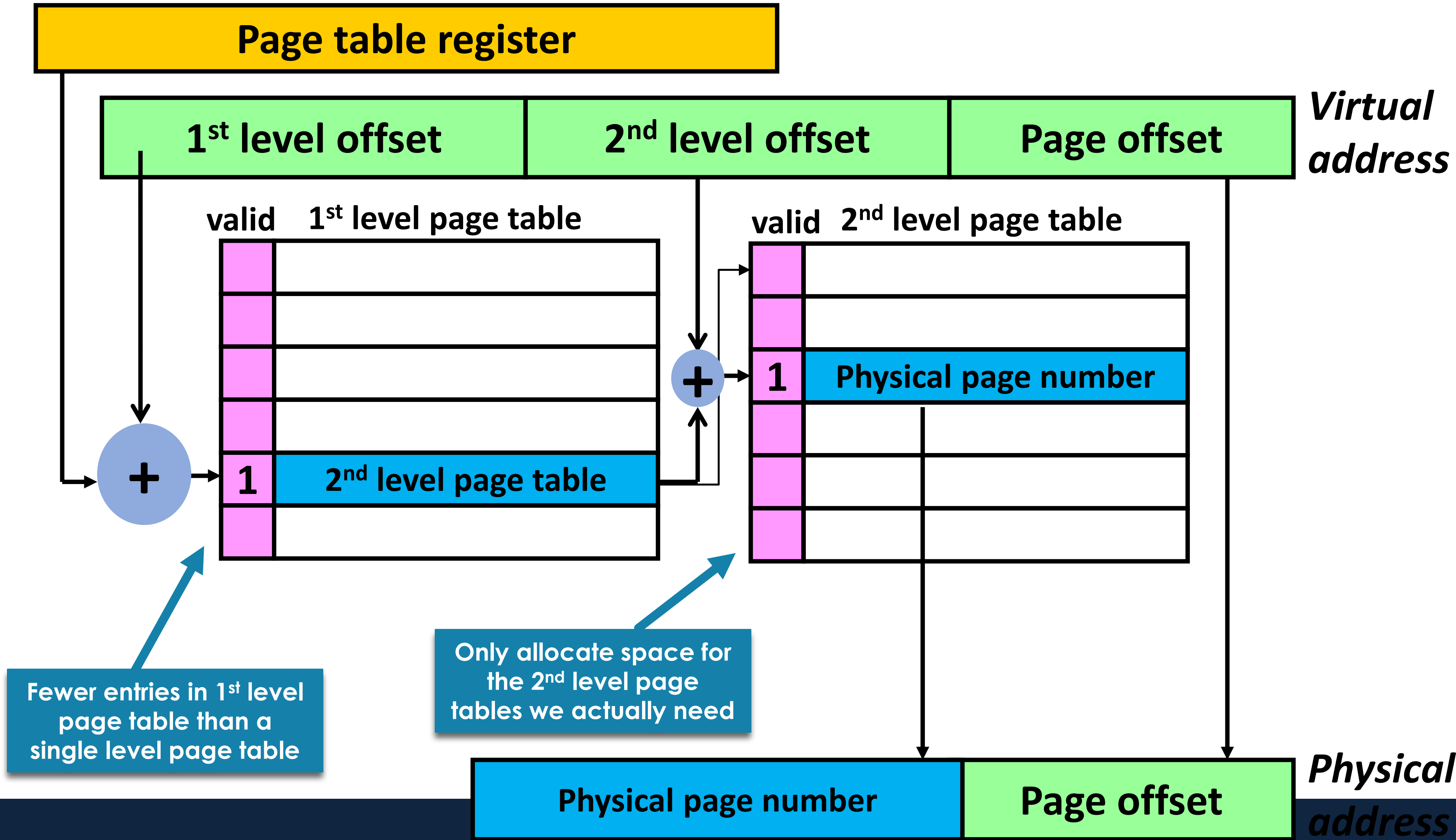
valid	2nd level page table
1	0x1000

valid	PPN
1	0x1
1	0x2
1	0x3

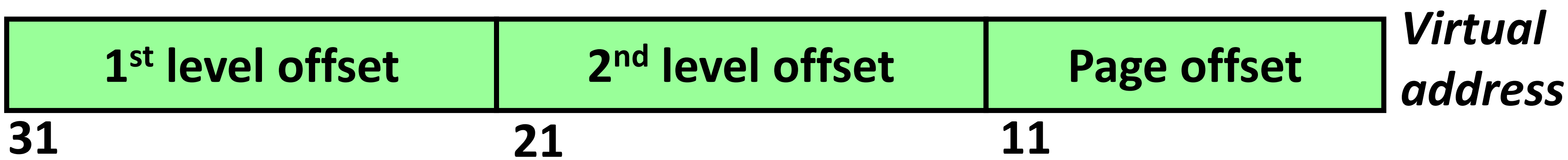
Multi-level: Size is proportional to amount of memory used

<u>Common case</u>: most programs use small portion of virtual memory space

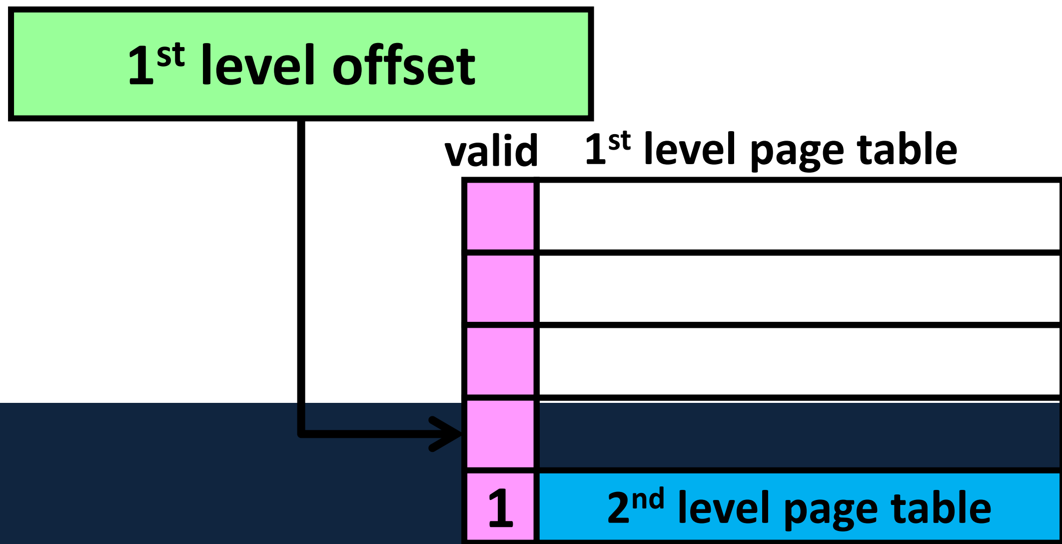
Hierarchical page table



Hierarchical page table – 32bit Intel x86

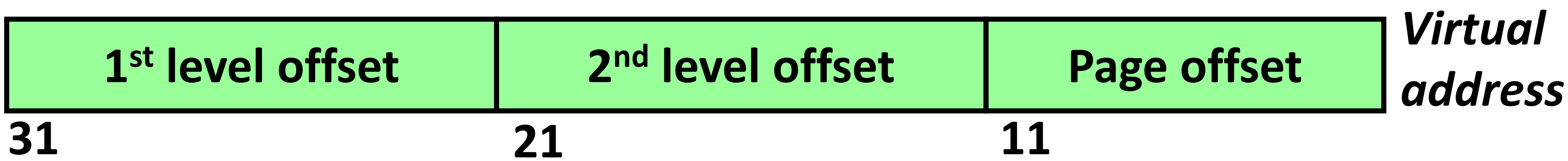


- How many bits in the virtual 1st level offset field? 10
- How many bits in the virtual 2nd level offset field? 10
- How many bits in the page offset? 12
- How many entries in the 1st level page table? $2^{10}=1024$

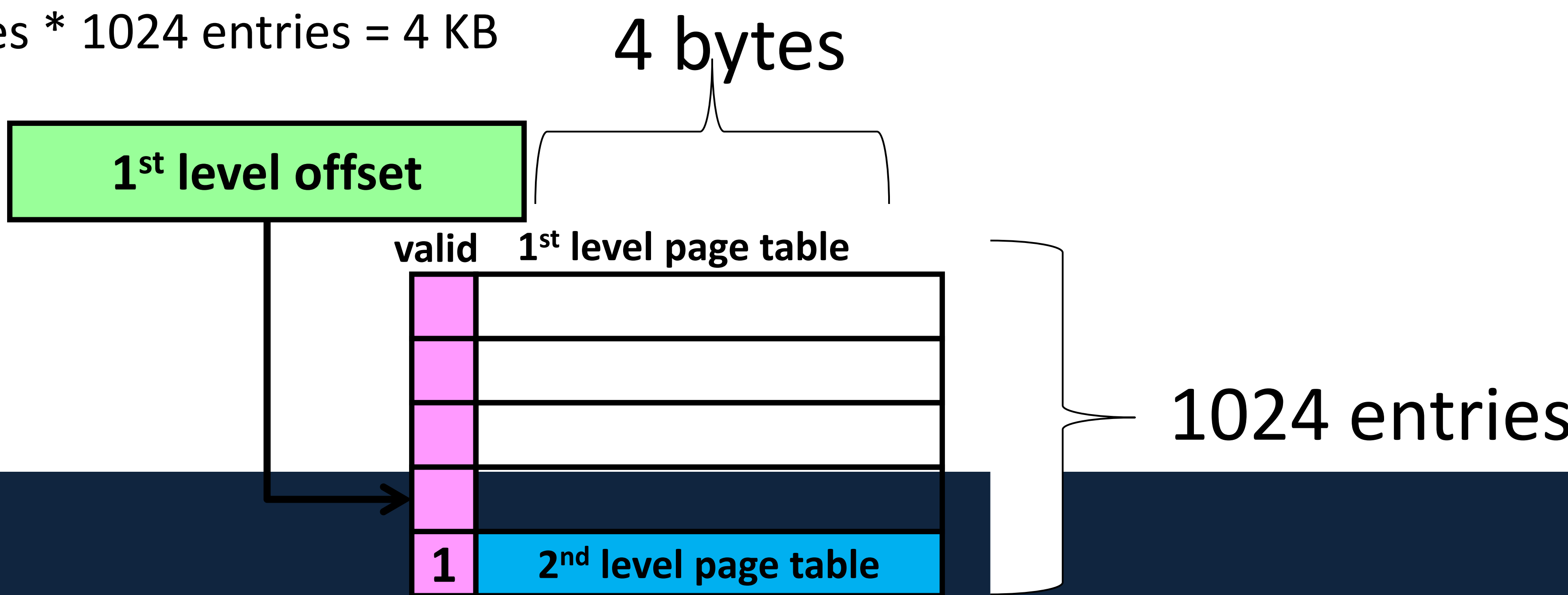


Poll

Hierarchical page table – 32bit Intel x86

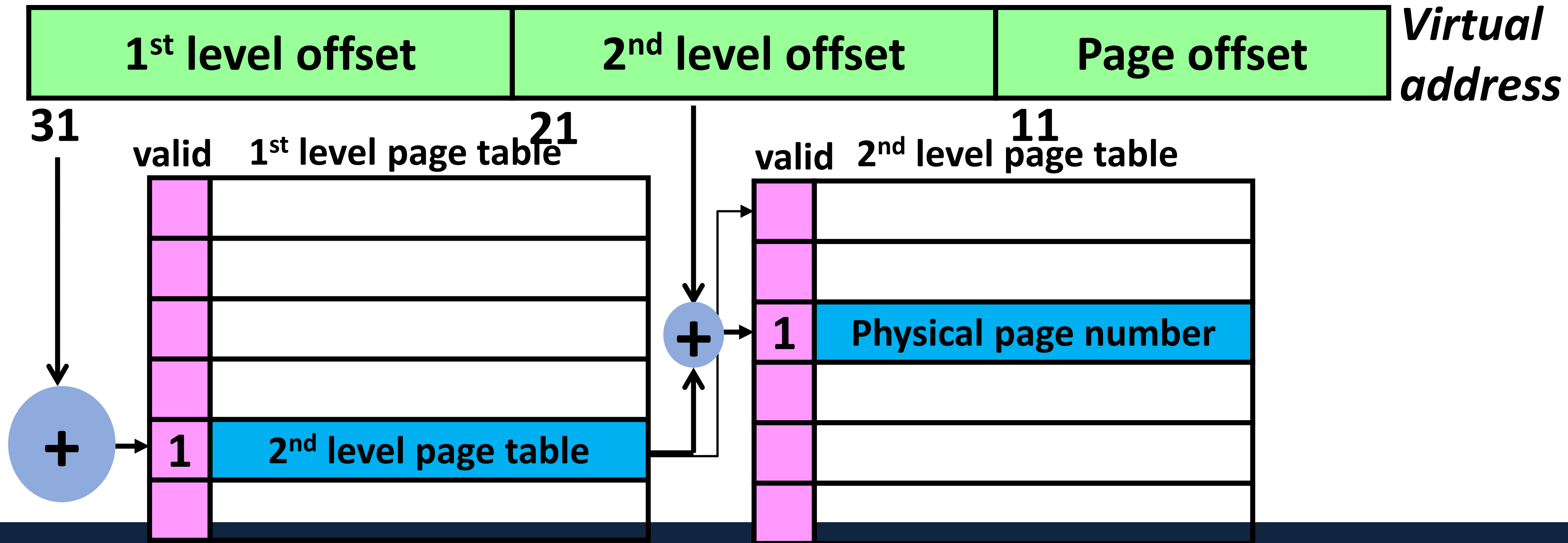


- How many bytes for each entry in the 1st level page table?
  - 4 bytes (address of 2nd level page)
- Total size of 1st level page table
  - 4 bytes * 1024 entries = 4 KB



Hierarchical page table

- How many entries in the 2nd level of the page table?
  - $2^{10} = 1024$
- How many bytes for each VPN in a 2nd level table?
  - Let's round up to 4 bytes



Hierarchical page table – 32bit Intel x86

1st level offset	2nd level offset	Page offset
31	21	11

Virtual address

- How many bits in the virtual 1st level offset field?
- How many bits in the virtual 2nd level offset field?
- How many bits in the page offset?
- How many entries in the 1st level page table?
- How many bytes for each entry in the 1st level page table?
- How many entries in the 2nd level of the page table?
- How many bytes for each entry in a 2nd level table?
- **What is the total size of the page table?**

  (here *n* is number of valid entries in the 1st level page table)

Hierarchical page table – 32bit Intel x86

1st level offset	2nd level offset	Page offset	*Virtual address*
31	21	11	

• How many bits in the virtual 1st level offset field?	10
• How many bits in the virtual 2nd level offset field?	10
• How many bits in the page offset?	12
• How many entries in the 1st level page table?	$2^{10}=1024$
• How many bytes for each entry in the 1st level page table?	4
• How many entries in the 2nd level of the page table?	$2^{10}=1024$
• How many bytes for each entry in a 2nd level table?	~4
• **What is the total size of the page table?** **(here *n* is number of valid entries in the 1st level page table)**	**4K+n*4K**