22. Virtual Memory: Basics

EECS 370 – Introduction to Computer Organization – Winter 2023

EECS Department
University of Michigan in Ann Arbor, USA

ANNOUNCEMENTS

- Project 4 due Thursday, April 13th
- Homework 6 due Monday, April 17th
- Just 3 more lectures of material and one review session!
  - Notice, we have no class on April 13th

Go check cache organization in your machine!

- For Linux you can use following command:

```
$:sudo dmidecode -t cache
```

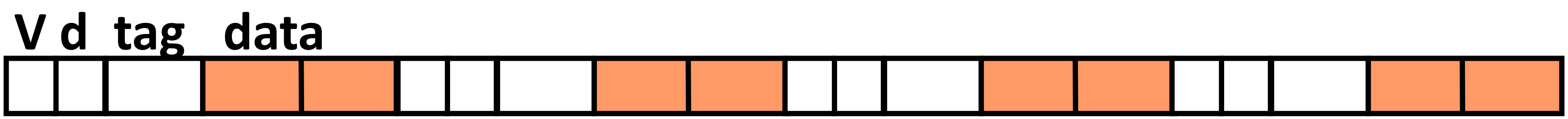
DMI—desktop management interface

(you may need to install **`dmidecode`** on your machine)

Cache Organization Comparison/Review

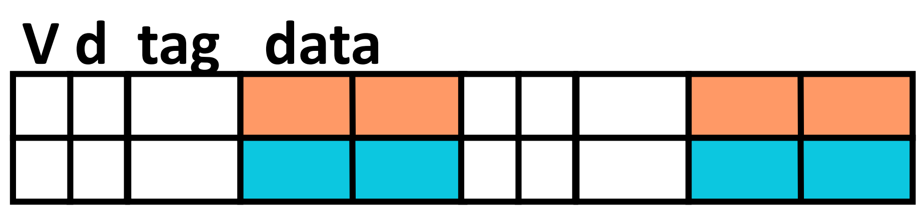
Block size = 2 bytes, total cache size = 8 bytes for all caches

1. Fully associative (4-way associative)



2. Direct mapped

3. 2-way associative



Quick review questions

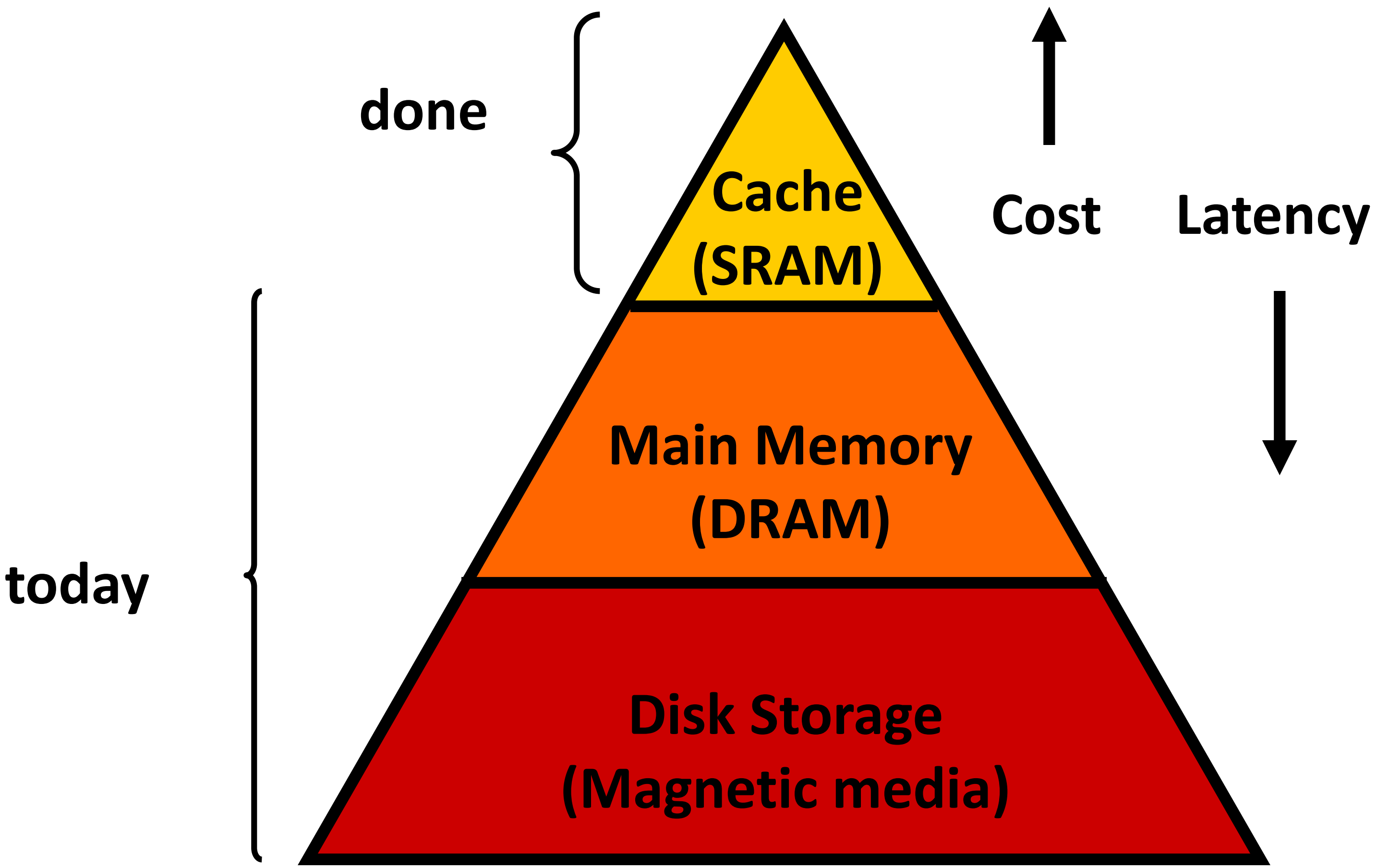
- If you have a 16KB cache with 32-byte cache lines that is four-way set-associative on a computer with 32-bit addresses:
  - How many sets do you have?
  - How many bits do you need for the Offset? Index? Tag?

- Describe why we have an Icache and a Dcache on nearly all computers rather than just one unified cache.

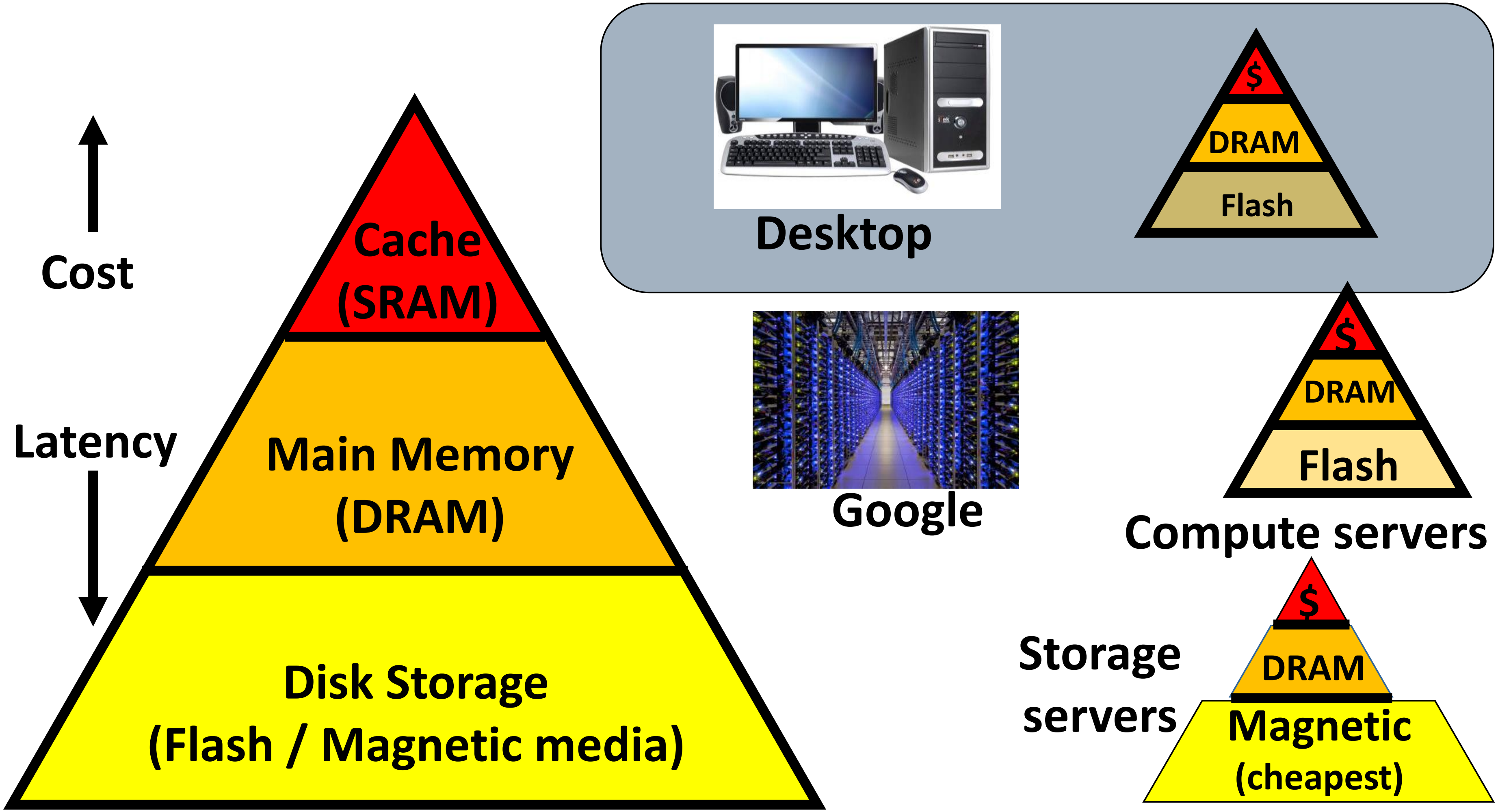
- Describe the primary advantage of write-back caches over write-through caches.
  - Give an example when that isn't the case (that is, where write-through caches do better than write-back caches on the advantage you identified.

VIRTUAL MEMORY

Storage Hierarchy



Memory and Storage Hierarchy



Memory: the issues(s)

- We run many programs on a same machine
  - Each of them may require GBs of storage
  - Unrelated programs must not have access to each other's storage

- DRAM is too expensive to buy 100s GB, but disk space is not…
  - We want our system to work even if it requires more DRAM than we bought.
  - We also don't want a program that works on a machine with 2048 MB of DRAM to stop working if we try to run it on a machine with only 512 MB of main memory.

- And, it would be nice to be able to enforce different policies on different portions of the memory (e.g.: read-only, etc)

Solution 1: User control

- Leave the problem to the programmer
  - Assume the programmer knows the exact configuration of the machine.
    - Programmer must either make sure the program fits in memory, or break the program up into pieces that do fit and load each other off the disk when necessary

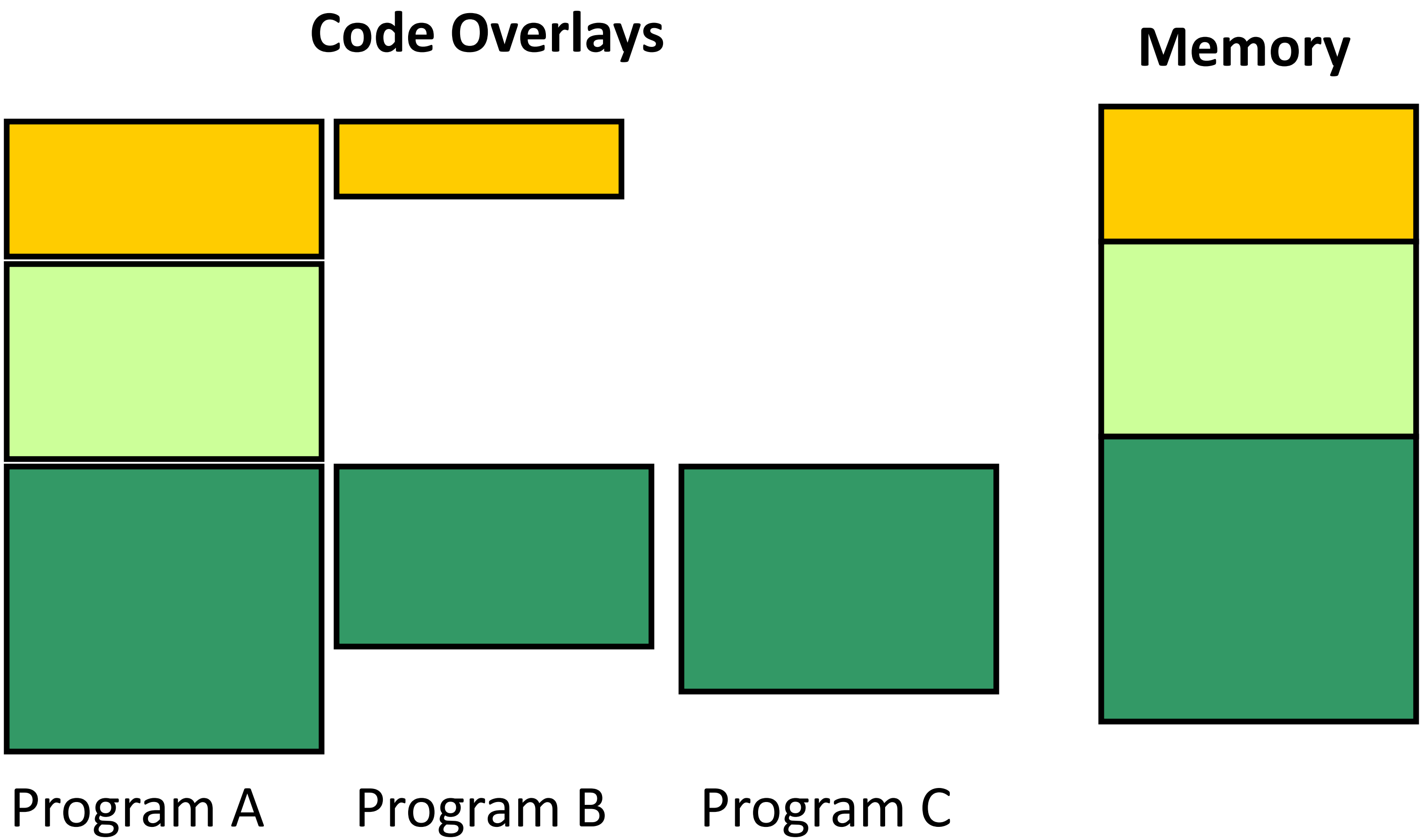
- Not a bad solution in some domains
  - The hardware design is simple
  - (Original) PlayStation, engine control unit in a car, etc.
  - Systems with severe design constraints (e.g. RTOS)

Solution 2: Overlays

- A little automation to help the programmer
  - build the application in overlays
    - Two pieces of code/data may be overlayed iff
      - They are not active at the same time
      - They are placed in the same memory region

- Managing overlays is performed by the compiler
  - Good compilers may determine overlay regions
  - Compiler adds code to read the required overlay memory off the disk when necessary
- The hardware design is still simple (most of the time)

Overlay example



Solution 3: Virtual memory

- Build new hardware and software that automatically translates each memory reference from a

virtual address

(which the programmer sees as an array of bytes)

to a

physical address

(which the hardware uses to either index DRAM or identify where the storage resides on disk)

Basics of Virtual Memory

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

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

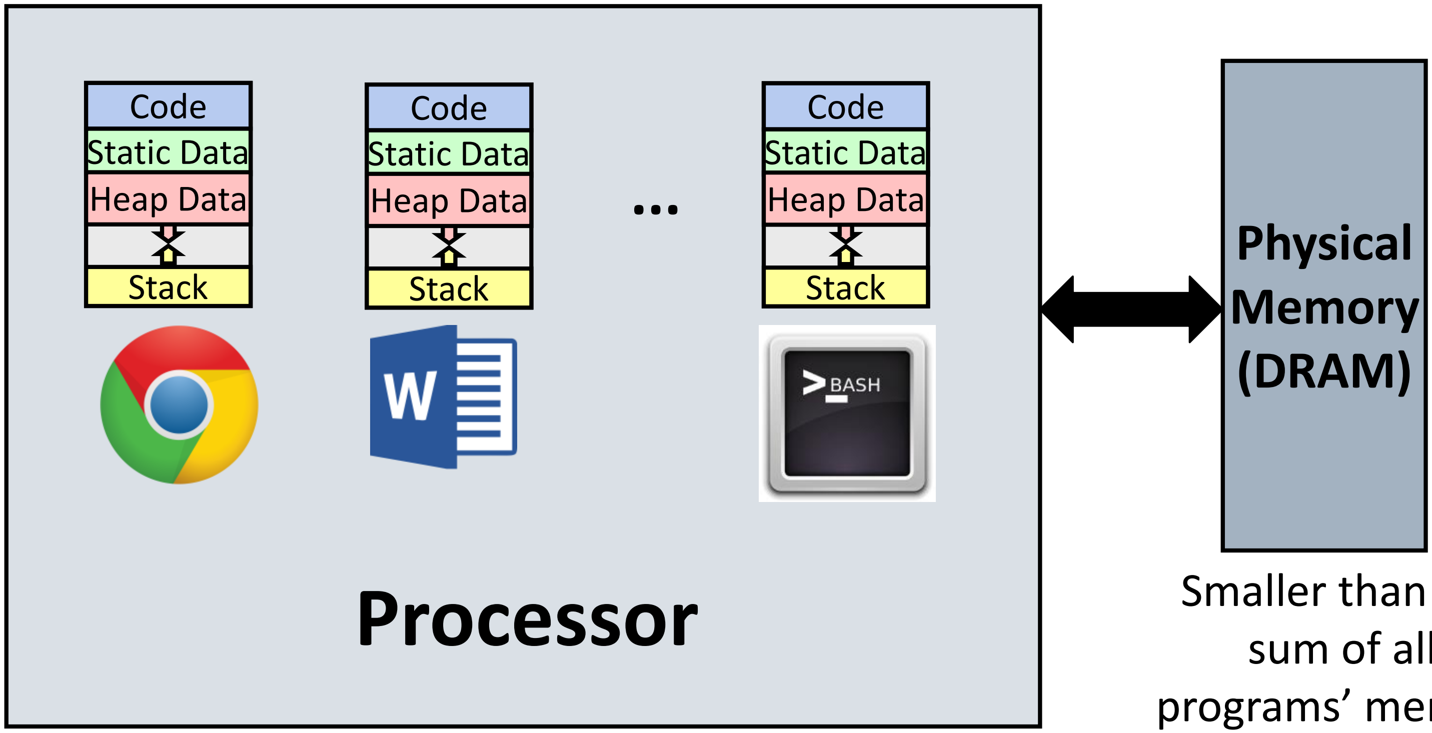
Virtual address		Physical address
0x800	→	**Disk ID 803C4**

Why Virtual Memory?

- Virtual memory enables multiple programs to share the physical memory.

- Provides 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

Revisit real system view—multitasking



Smaller than the sum of all programs' memory

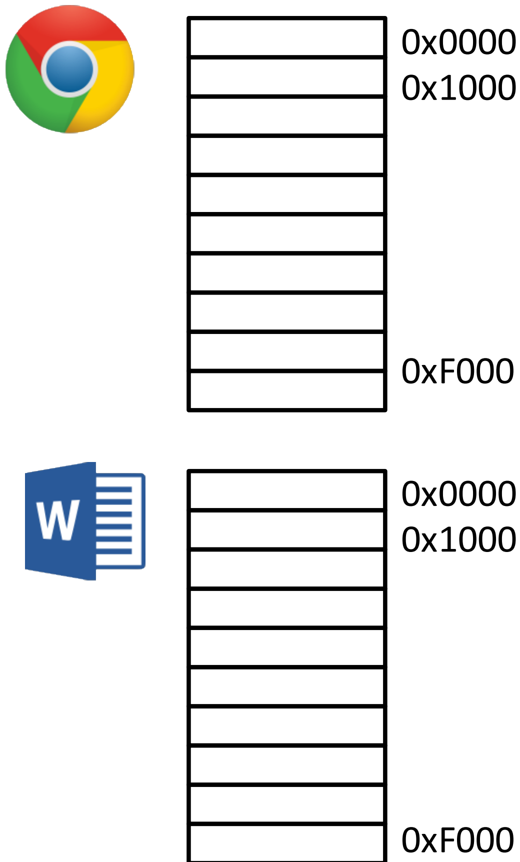
Managing Virtual Memory—VM

- Managed by hardware logic ***and*** operating system software.
  - Hardware for speed
  - Software for flexibility and because disk storage is controlled by the operating system

- The hardware must be designed to support VM*

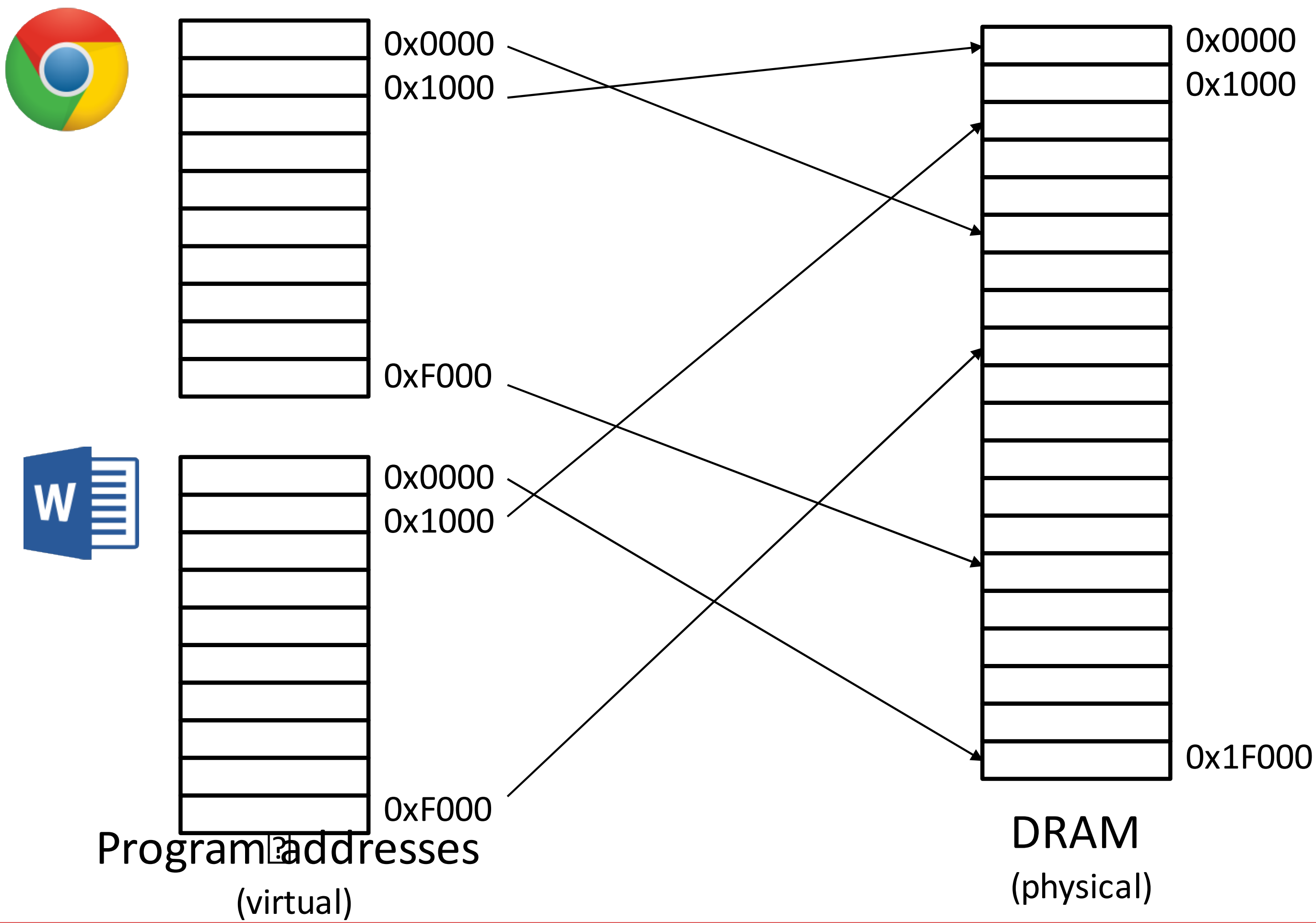
*(do not confuse VM for “Virtual Machine”, which is another concept entirely)

1. How to achieve **transparency & protection**?

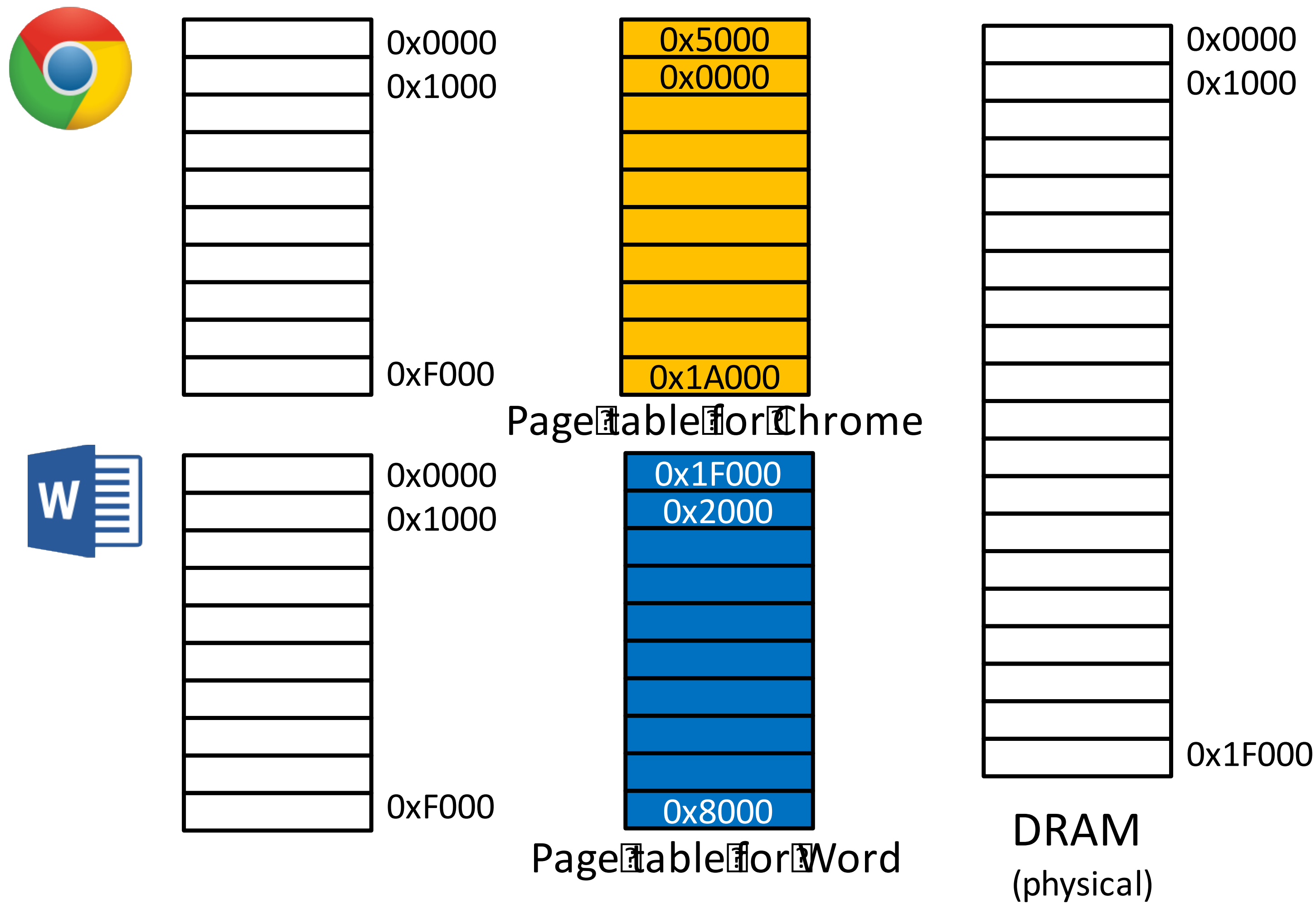


Program addresses
(virtual)

1. How to achieve transparency & protection?



1. How to achieve **transparency & protection**?



Page Table

- Page tables are maintained by the operating system
- Each process has its own page table
- Contains address translation information i.e., virtual address ➔ physical address
- 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

2. How to be not limited by DRAM capacity?

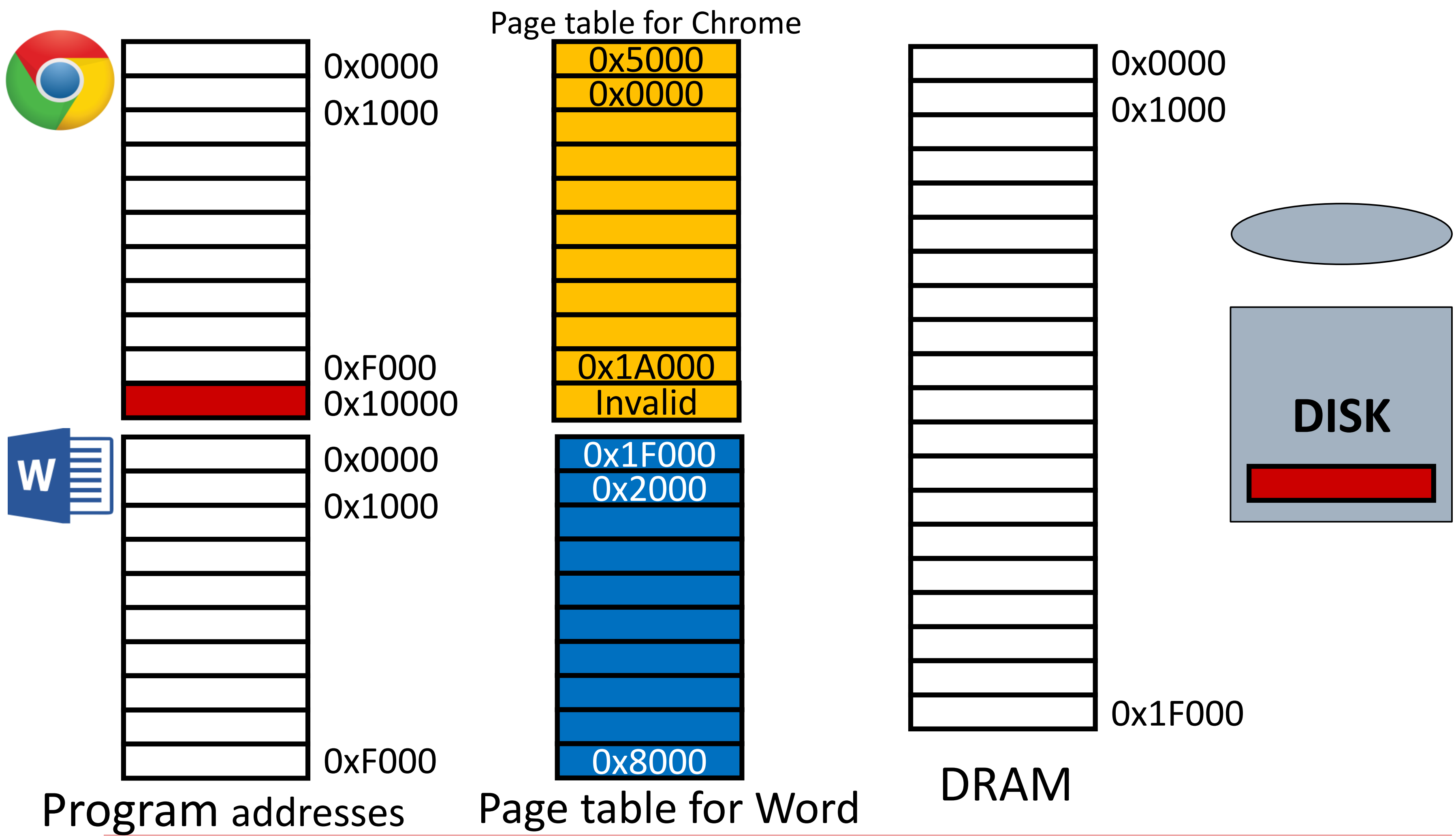
- Use disk as temporary space in case memory capacity is exhausted
  - This temporary space in disk is called swap partition in Linux-based systems
  - For fun check swap space in a linux system by:

$: 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**?



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)

Virtual address

Virtual page number	Page offset
31	11 0

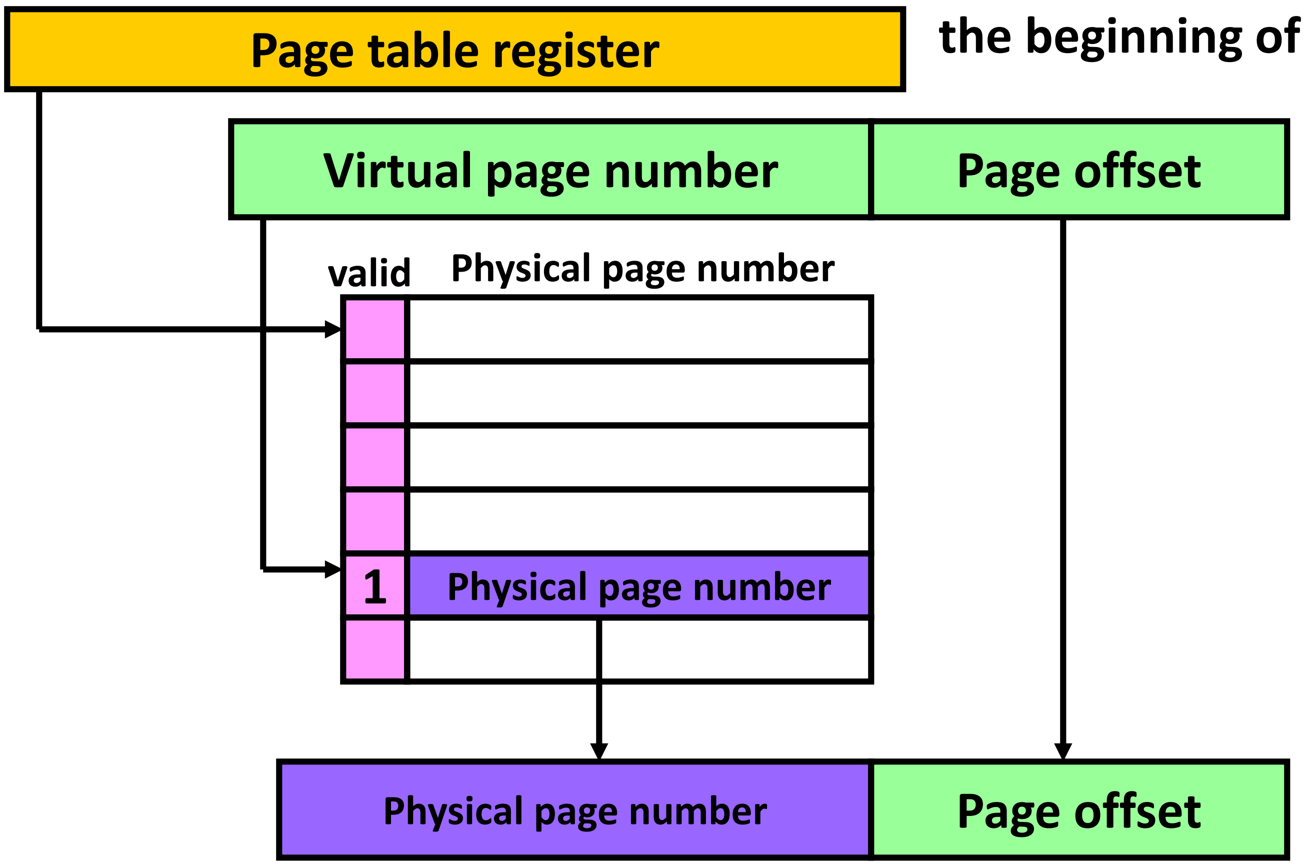
Physical address

Physical page number	Page offset
31	11 0

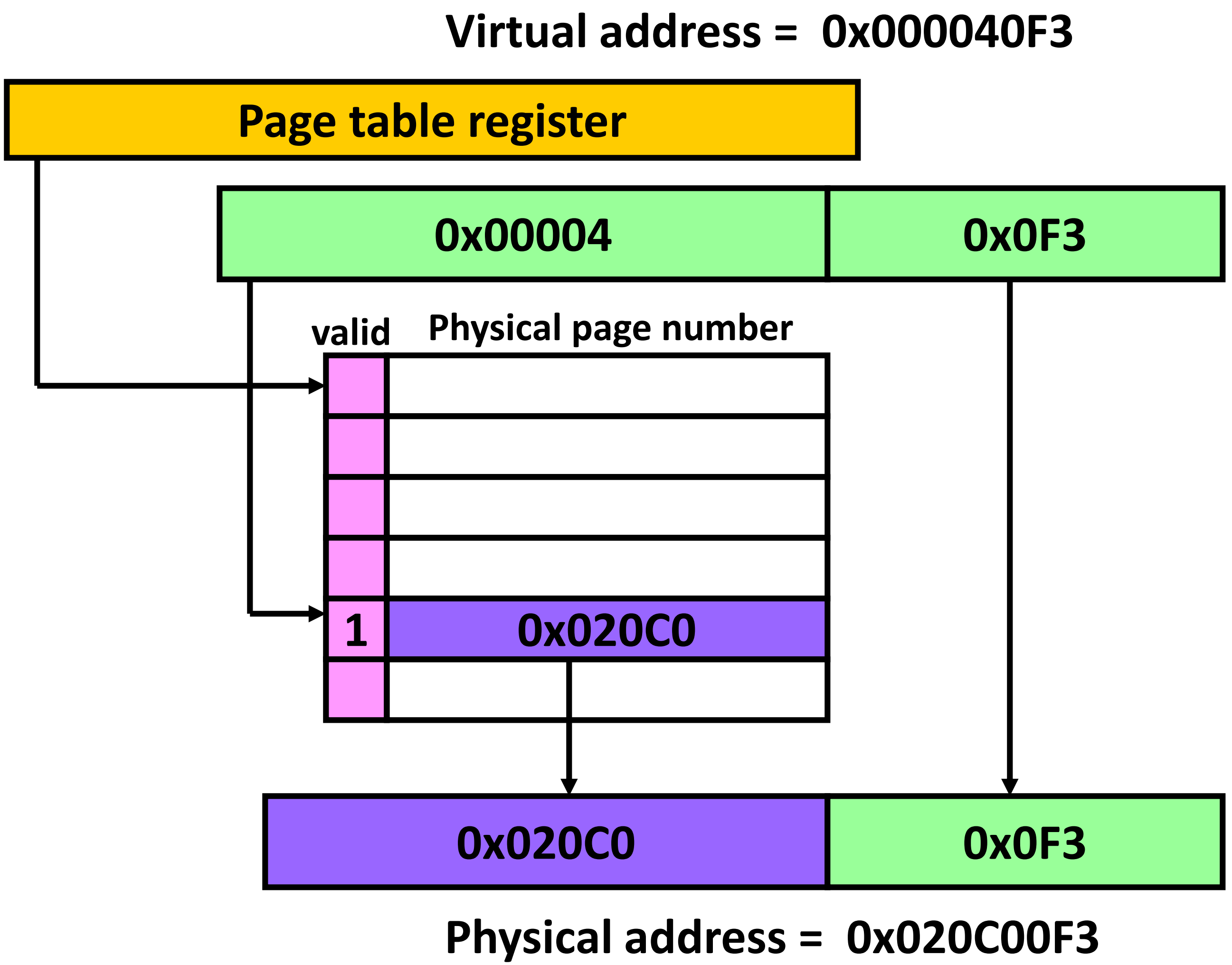
- **Virtual Page** accesses that are not found in physical memory (DRAM) are called **Page Faults**

Page table components

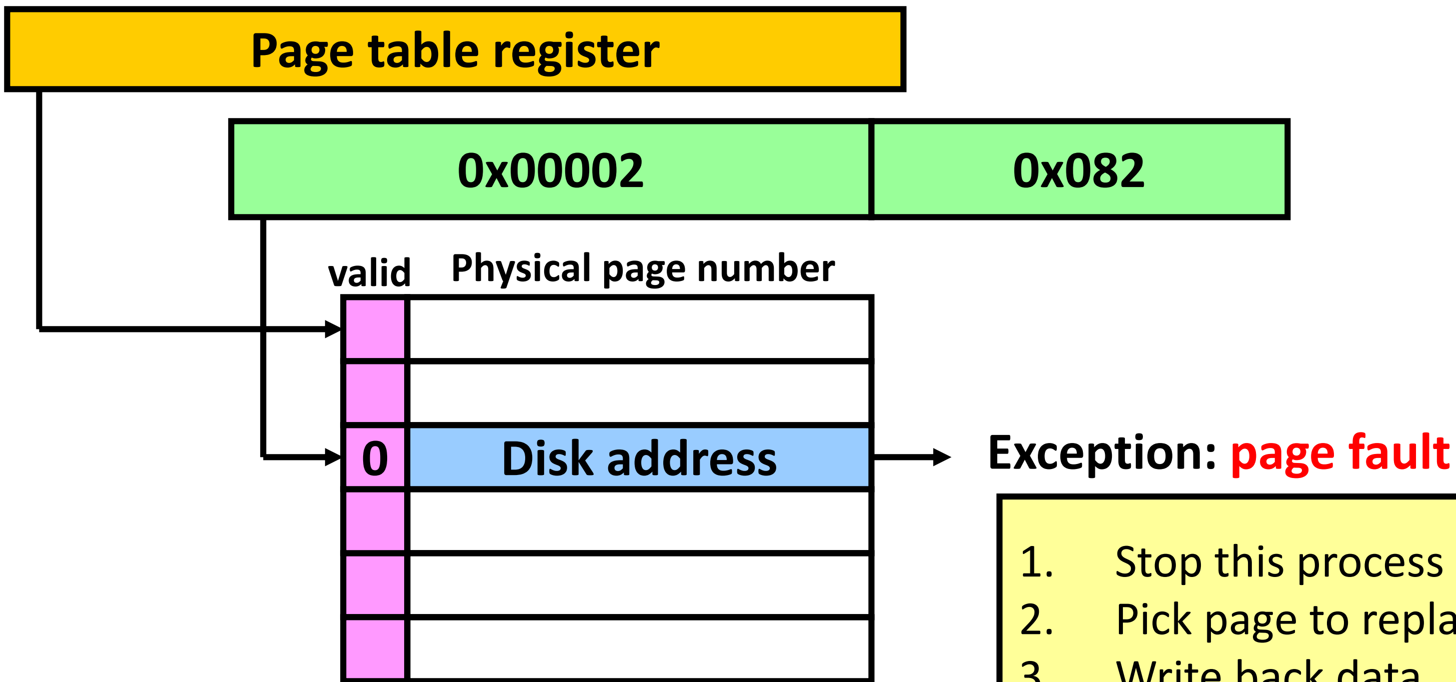
The *Page table register* points to the beginning of the page table



Page table components - Example



Page faults



1. Stop this process
2. Pick page to replace
3. Write back data
4. Get referenced page
5. Update page table
6. Reschedule process

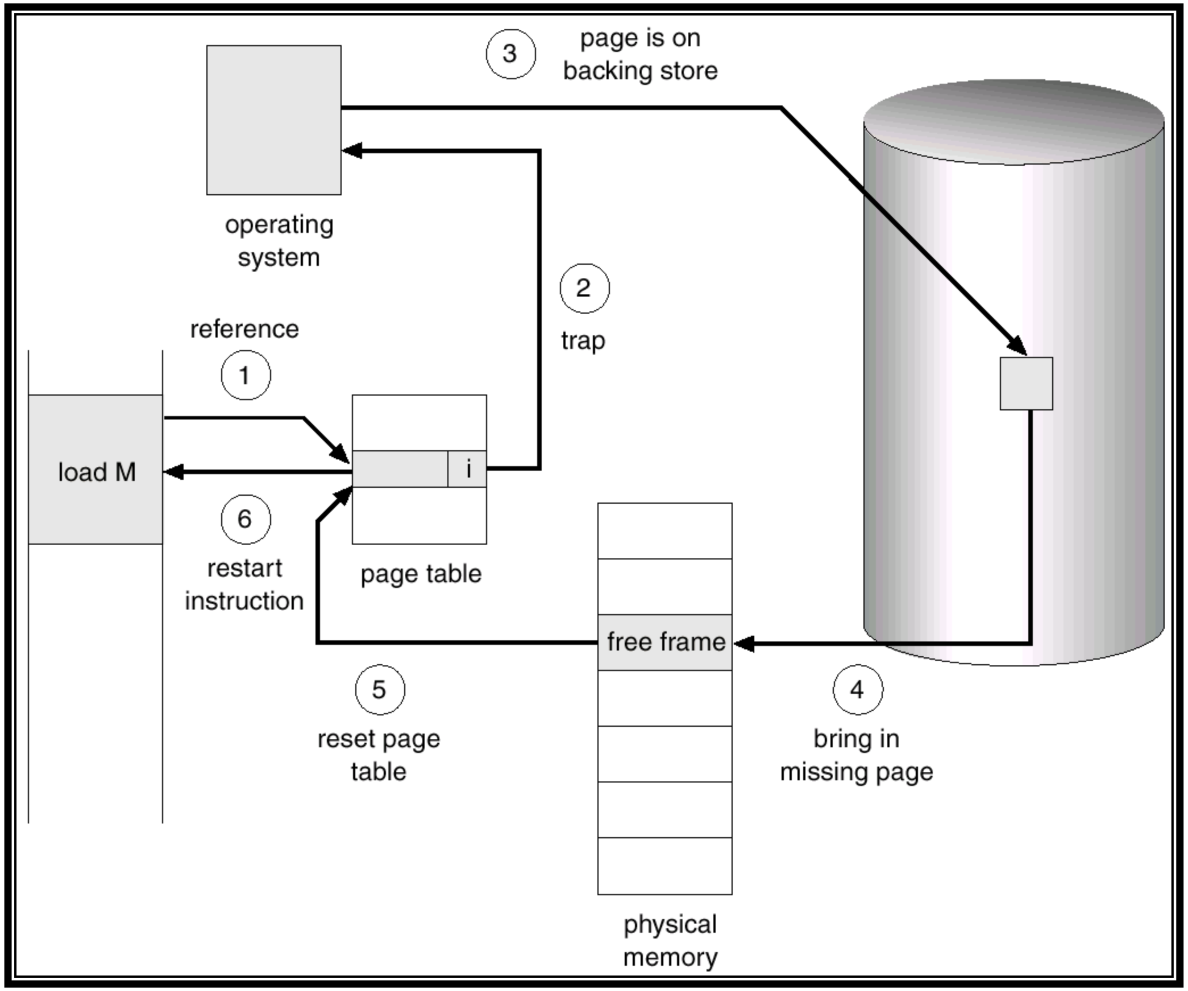
How do we find it on disk?

- 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.

Page faults



3
page is on
backing store
operating
system
2
trap
reference
1
load M
i
6
restart
instruction
page table
free frame
5
reset page
table
4
bring in
missing page
physical
memory

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			

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.

Class Problem (continued)

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