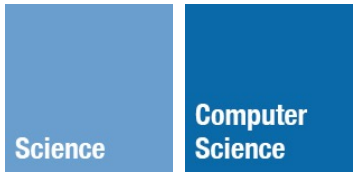
xv6 Overview



CS 450 : Operating Systems
Michael Saelee <lee@iit.edu>

IIT College of Science
ILLINOIS INSTITUTE OF TECHNOLOGY

Agenda

- Architectural overview
  - Features & limitations
- Hardware dependencies/features

Agenda

- Code review:
  - Headers and Process structures
  - Bootstrap procedure
  - Scheduling & Context switching
  - Sleep & Wakeup
  - Trap / Syscall mechanism

§Architectural Overview

xv6 is a *monolithic*, *preemptively-multitasked*, *multiprocessor-capable*, *32-bit*, *UNIX-like* operating system

some limitations:

- max addressable memory: 2GB
- few supported devices (e.g., no network)
- no support for kernel-level threading

limited syscall API:

System call	Description
fork()	Create process
exit()	Terminate current process
wait()	Wait for a child process to exit
kill(pid)	Terminate process pid
getpid()	Return current process's id
sleep(n)	Sleep for n seconds
exec(filename, *argv)	Load a file and execute it
sbrk(n)	Grow process's memory by n bytes
open(filename, flags)	Open a file; flags indicate read/write
read(fd, buf, n)	Read n byes from an open file into buf
write(fd, buf, n)	Write n bytes to an open file
close(fd)	Release open file fd
dup(fd)	Duplicate fd
pipe(p)	Create a pipe and return fd's in p
chdir(dirname)	Change the current directory
mkdir(dirname)	Create a new directory
mknod(name, major, minor)	Create a device file
fstat(fd)	Return info about an open file
link(f1, f2)	Create another name (f2) for the file f1
unlink(filename)	Remove a file

very limited set of user-level programs:

- shell, cat, echo, grep, kill, ln, ls, mkdir, rm, wc
- no compiler/debugger/editor
  - development (kernel/user) takes place on another platform!

§Hardware Dependencies / Features

xv6 runs on an x86 (Intel) processor, and relies on many of its hardware features

e.g., privilege levels (kernel/user mode), interrupt vector & procedure, segmentation & paging (VM)

Recall: 2-bit *current privilege level* (CPL) flag

- CPL=3 → "user" mode
- CPL=0 → "supervisor/kernel" mode
  - guards special instructions & hardware
  - also restricts access to interrupt & VM structures

CPL is actually part of the `%cs` register, which specifies the *code segment* address

`%cs` and `%eip` (x86 PC) identify an instruction to execute *and its privilege level*

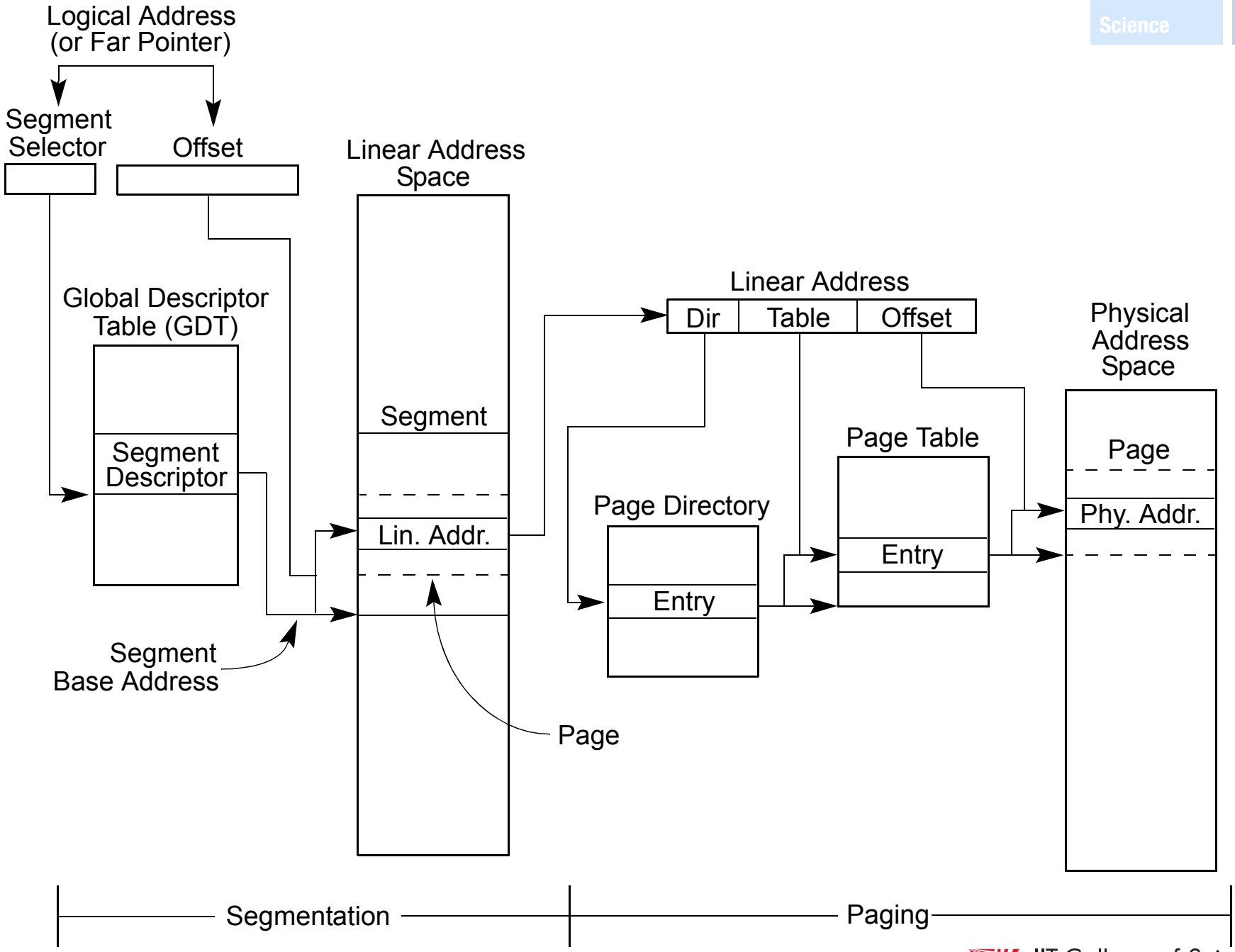
but CPL cannot be modified directly!

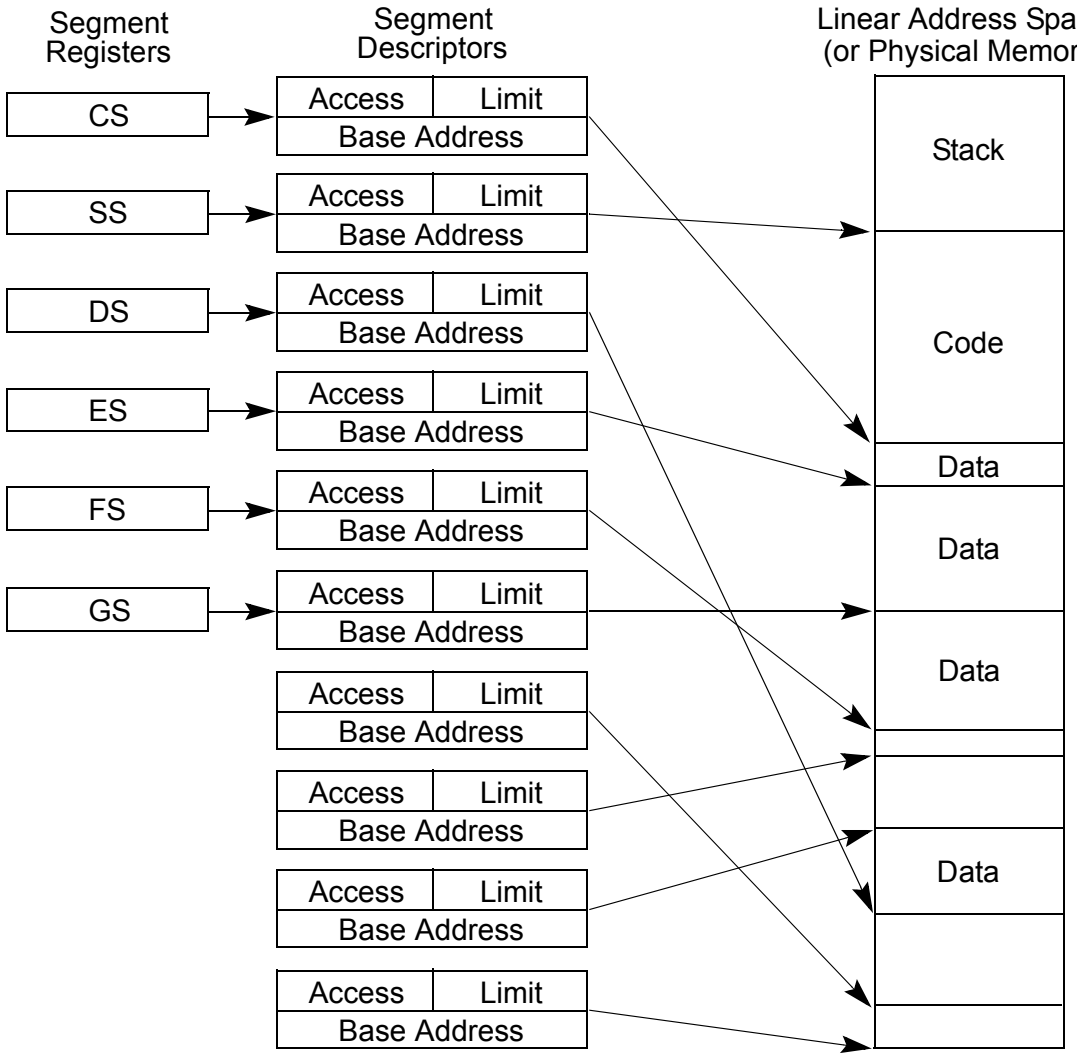
- lower (raise priority) via `int` instruction
- raise (lower priority) via `iret` instruction

`int` instruction (and h.w. interrupt) result in *interrupt descriptor table* (IDT) lookup

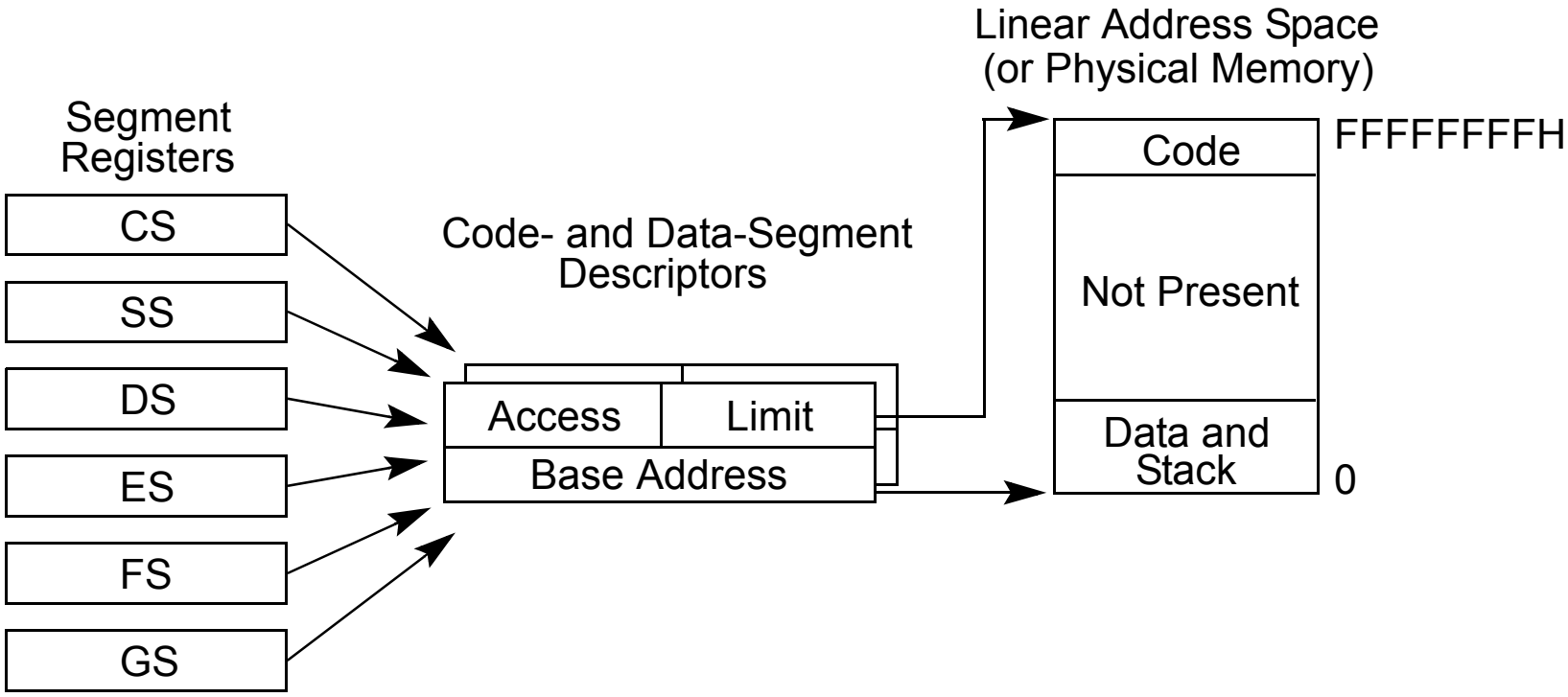
- fetches target `%cs` and `%eip` (aka "gate") for corresponding handler
- restricts entry points into kernel
- install base address of IDT with `lidt`

xv6 also relies on x86 *segmentation* and *paging* to implement virtual memory

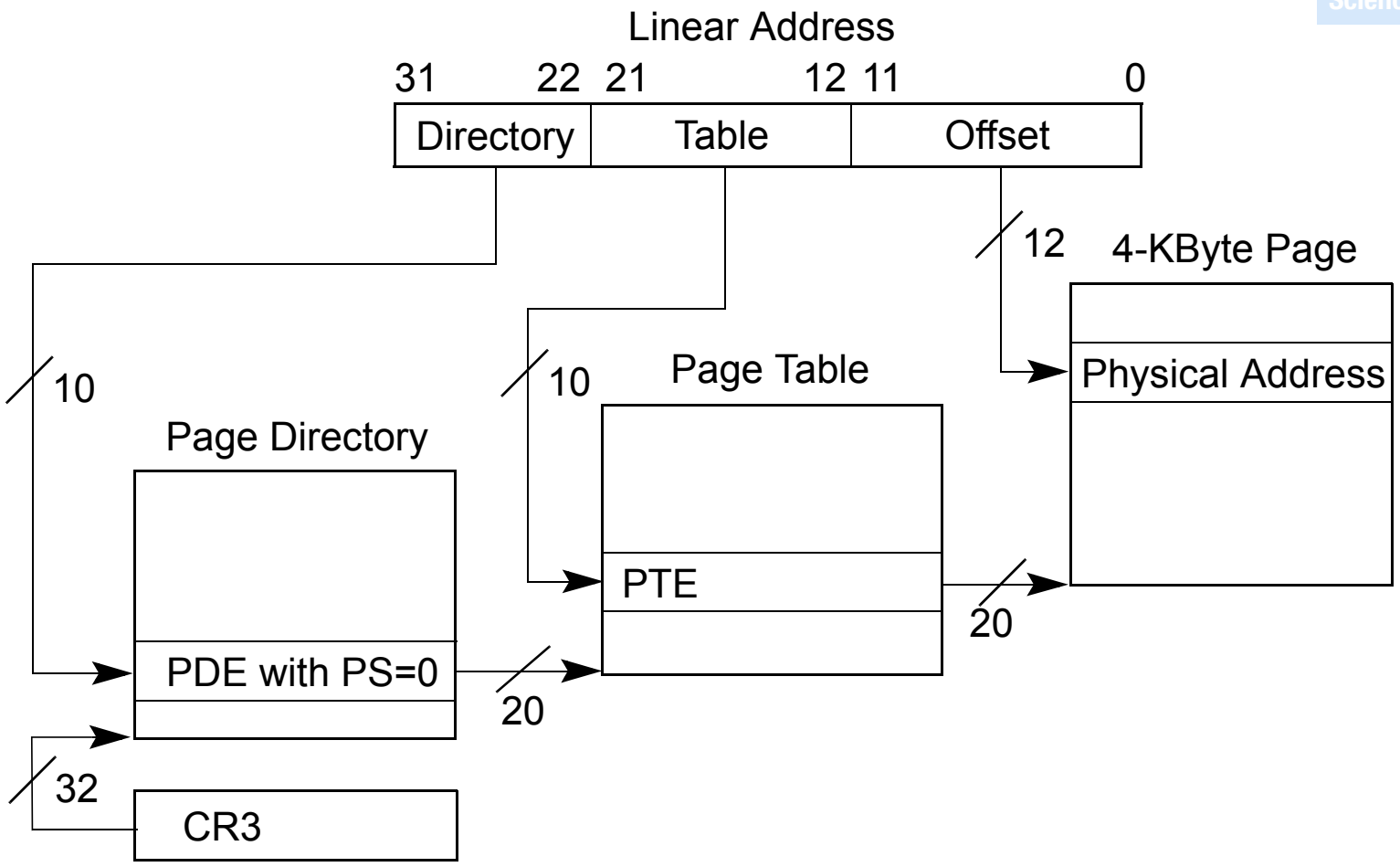

Logical Address
(or Far Pointer)
Segment Selector
Offset
Linear Address Space
Global Descriptor Table (GDT)
Segment Descriptor
Segment
Lin. Addr.
Segment Base Address
Page
Linear Address
Dir
Table
Offset
Page Directory
Entry
Page Table
Entry
Physical Address Space
Page
Phy. Addr.
Segmentation
Paging



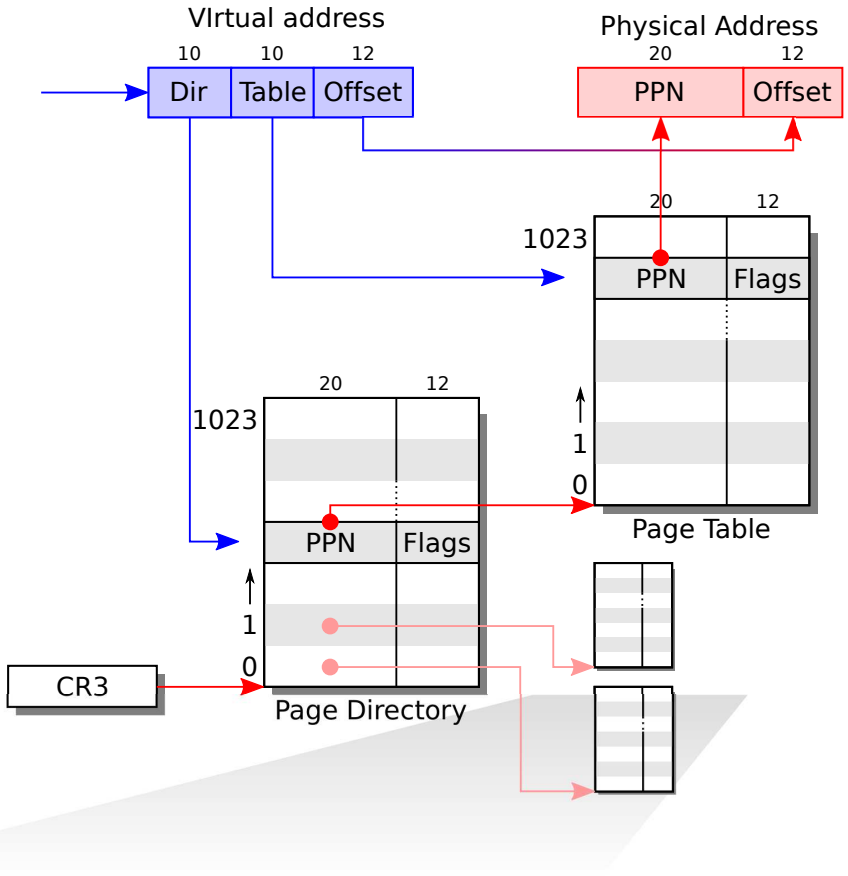
Segment descriptors

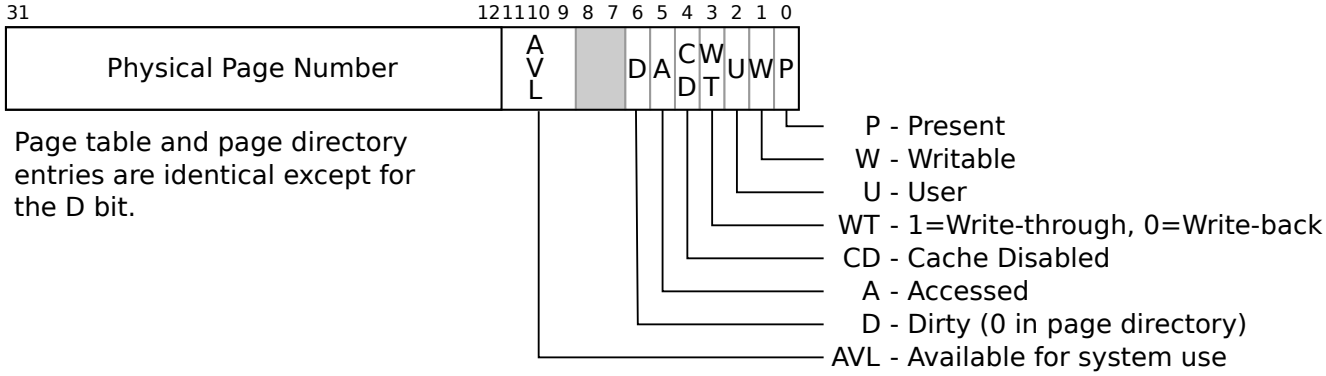

Linear Address Space
(or Physical Memory)
Segment
Registers
CS
SS
DS
ES
FS
GS
Code- and Data-Segment
Descriptors
Access
Limit
Base Address
Code
Not Present
Data and
Stack
FFFFFFFFH
0

"Flat" model



IA-32 paging (4KB pages)





Page table and page directory entries are identical except for the D bit.

§Demo & Code Review