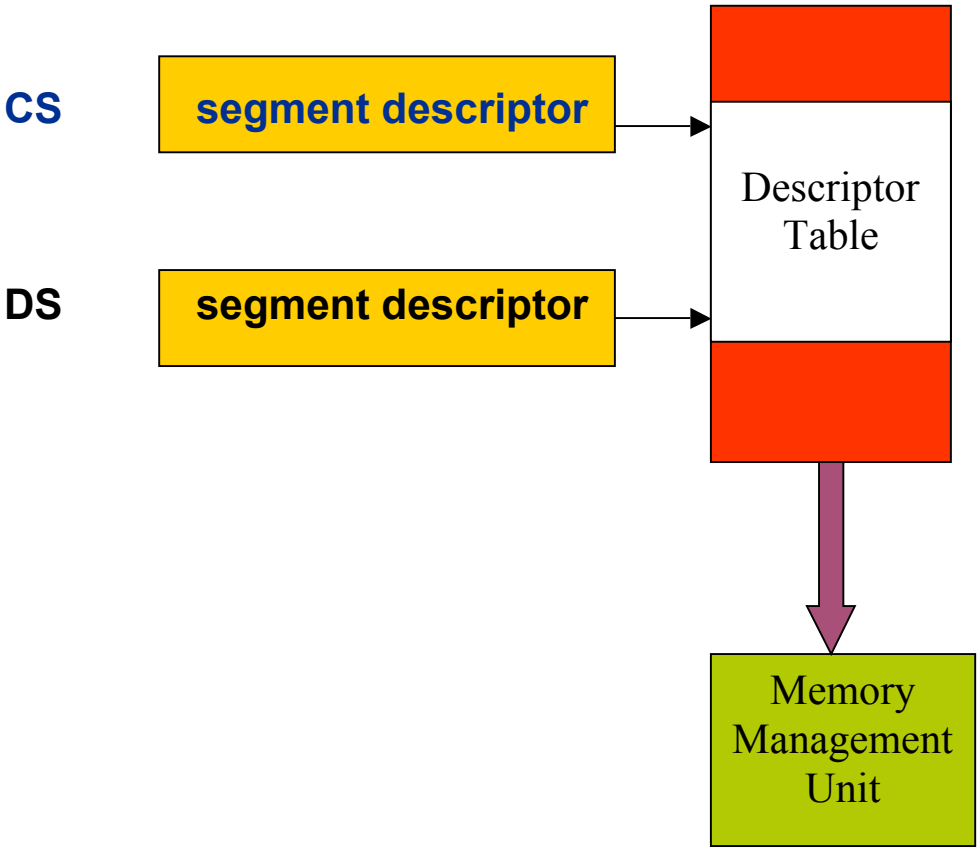
Segmented Virtual Memory of Pentium

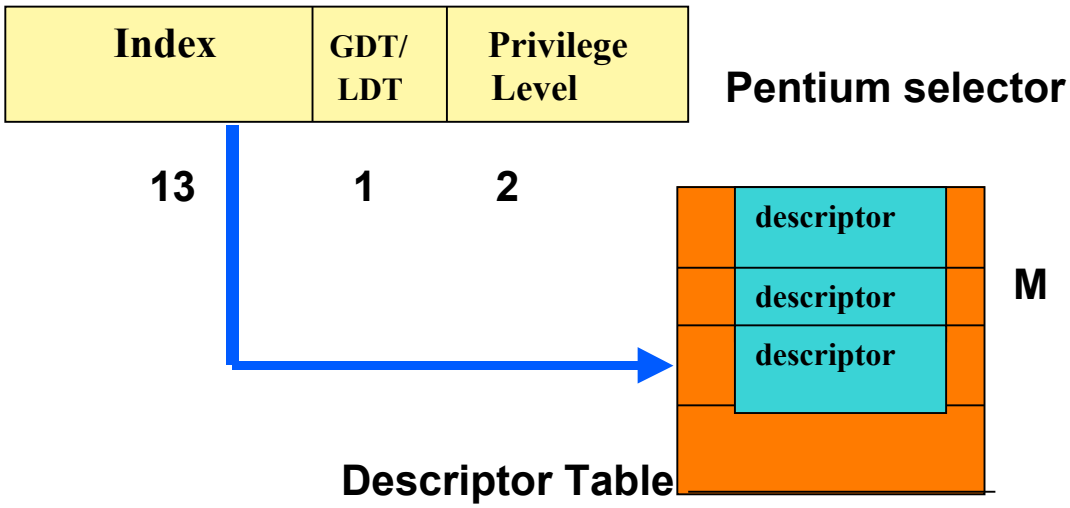
MMU translates the Virtual Address to Physical address.

Memory contains a number of program segments. Each segment has a segment descriptor stored in an appropriate segment register.



Segment descriptor

A segment register contains a 16-bit selector



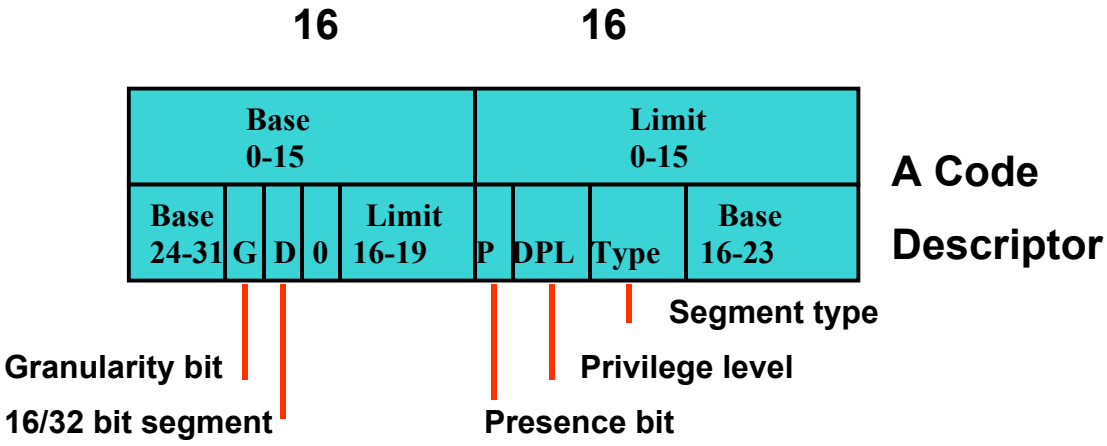
LDT (Local Descriptor Table): one per process

GDT (Global Descriptor Table): one for the system

Pentium can support (1) pure segmentation or (2) segmentation and paging

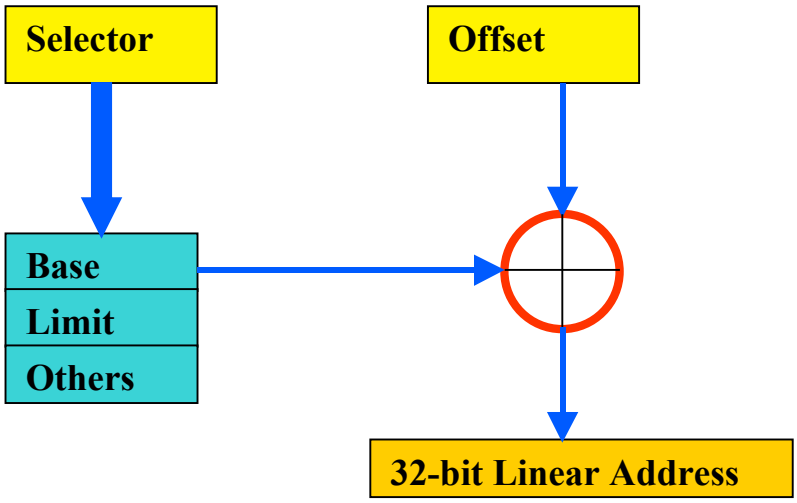
Example of a descriptor

Descriptor loaded into MMU.



Segment table entry

Index	Base	Limit	P	G	Others



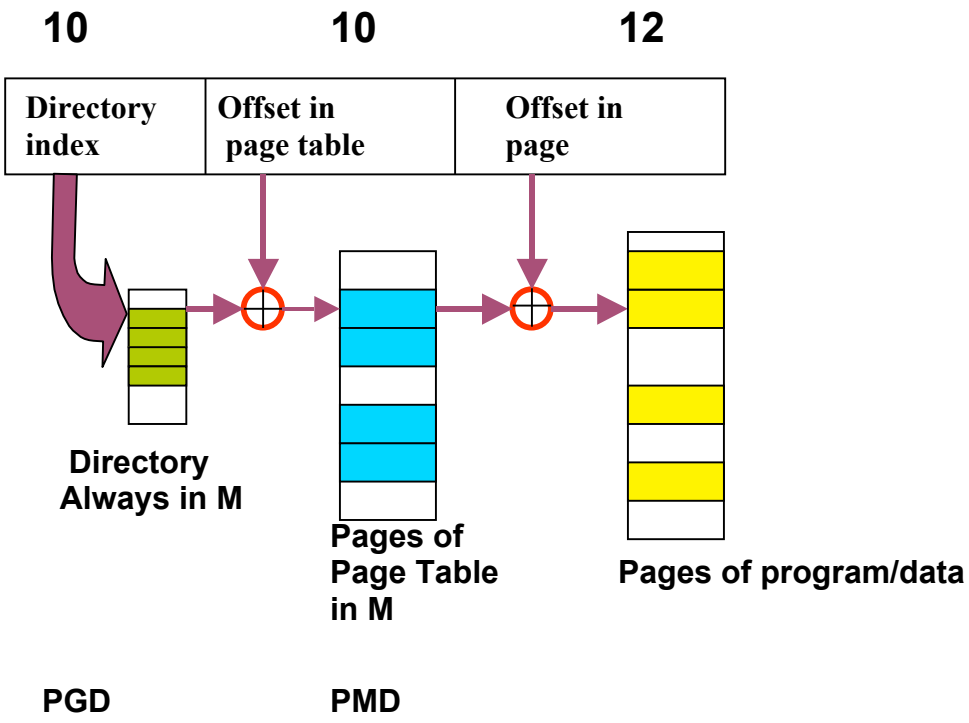
Page mode off

32-bit address is the physical address.

Page mode on

32-bit address is a virtual address, and the segment is divided into pages.

Physical address computed by a 2-level translation.



Implementing Protection through VM

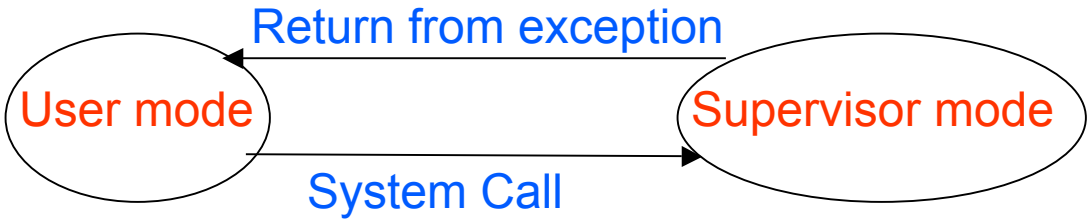
User mode

User cannot modify some system states, or execute certain instructions. The restriction is important so that user programs cannot cheat

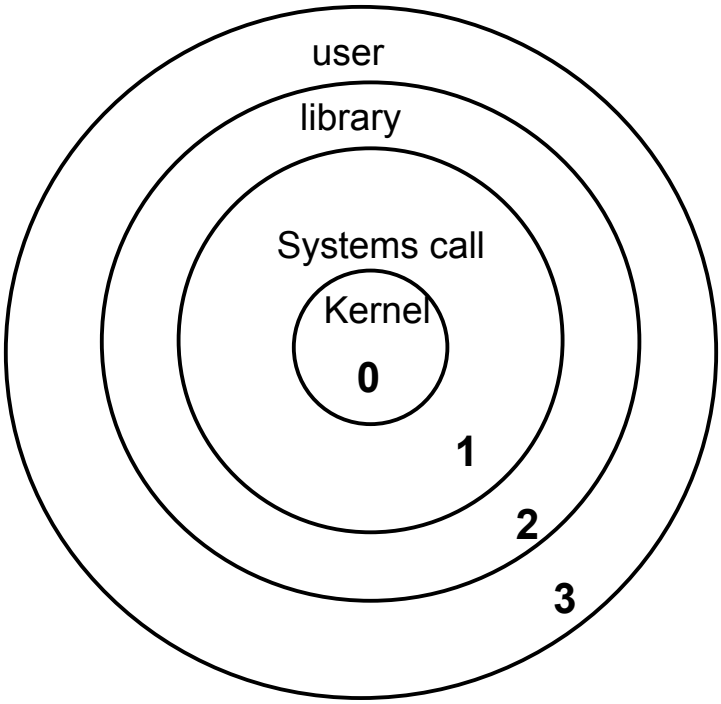
Supervisor mode (or kernel mode or privileged mode)

The OS takes control

A single bit distinguished between the two modes.



Memory protection in Pentium

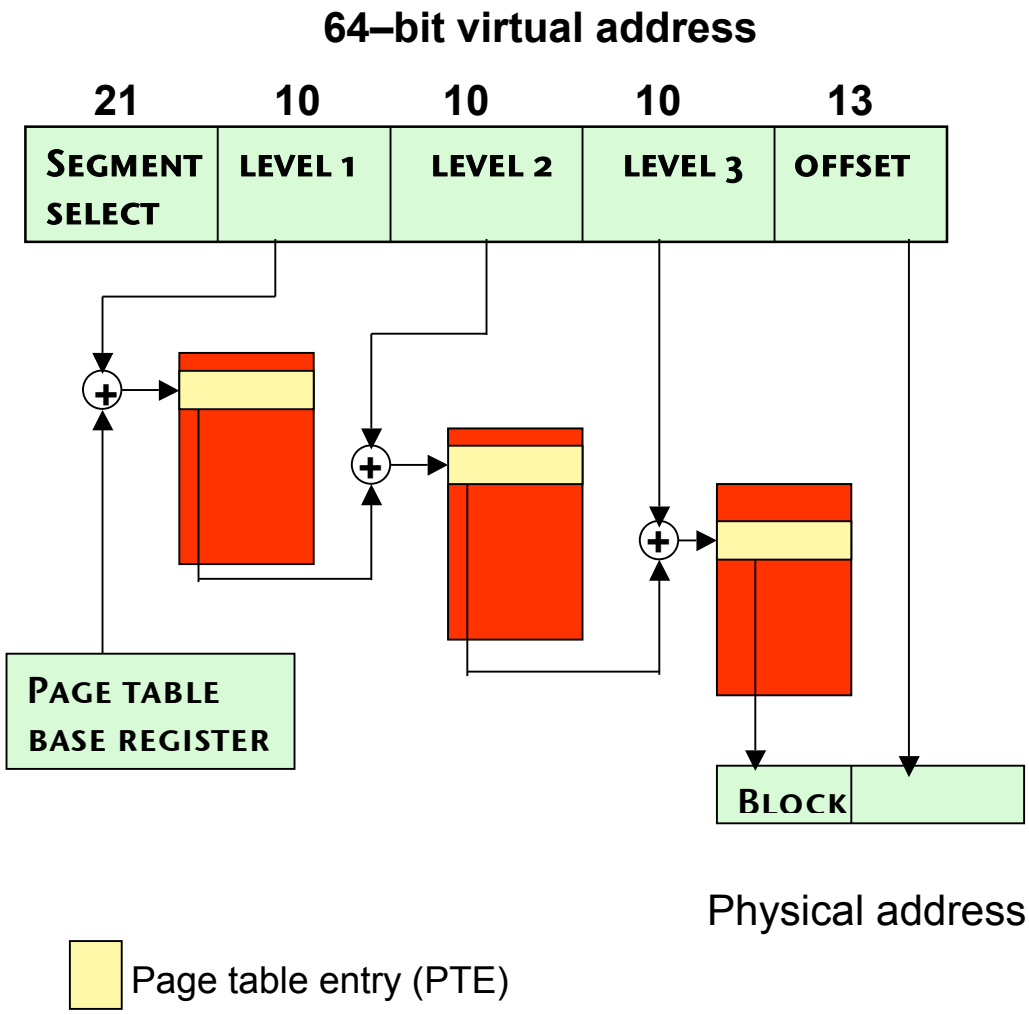


Each segment belongs to a ring with **a designated privilege level**. **Lower numbered** rings have **higher security**.

Mode switch examples

Calls towards segments of lower security are unrestricted. However, a call to a segment of higher security is controlled using **call gates**. When such a call is allowed, the calling process temporarily acquires the privilege level of the called program. This can open the gates for Trojan horse attacks, but there are safeguards against such attacks.

Alpha AXP 21264 Virtual Memory



Notes

Each page table entry is of 64 bits, i.e. 8 bytes.

Page size = 8 KB, so we need 13 bits for the offset.

The size of the page table at each level is 1 page.

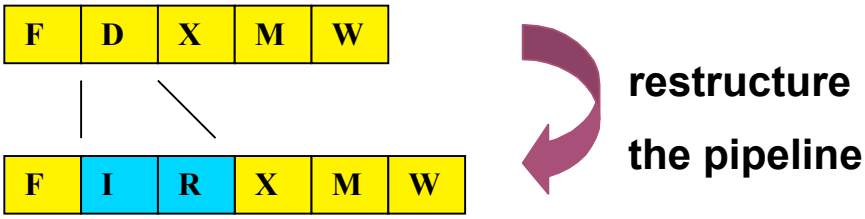
A TLB miss can cost up to three levels of memory access.

Linux on IA-64 uses three-level page tables

Advanced Pipelining Techniques

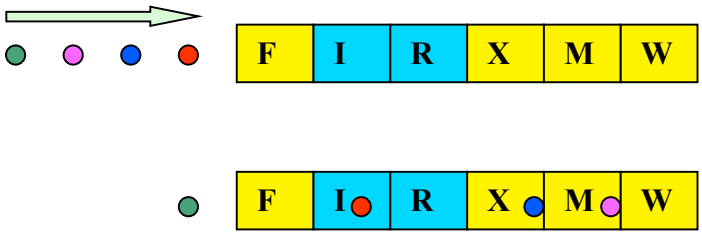
1. Dynamic Scheduling
2. Loop Unrolling
3. Software Pipelining
4. Dynamic Branch Prediction Units
5. Register Renaming
6. Superscalar Processors
7. VLIW (Very Large Instruction Word) Processors
8. EPIC (Explicitly Parallel Instruction Computers)
9. IA-64 Features

Dynamic Scheduling: CDC 6600 Style



Issue (I) **=** decode & wait for all
structural hazards to clear

Read (R) = read operands



In a **dynamic pipeline,** instructions can be **issued out-of-order** and they can **complete out-of-order. First used in CDC 6600.**

Dynamic pipelining will need **additional buffer space between stages**, but will also **speedup computation.**

The Impact of Dynamic Scheduling

Assume that the processor has an add/subtract unit (2 cycles), a multiplier (3 cycles) and a division unit (5 cycles).

Example 1 (speedup)

With static scheduling

1. **F2 := F4 / F6** — **F I R X X X X X M W**
2. **F10 := F2 + F8** — **F I o o o o o o o R X X M W**
3. **F12 := F6 – F14** — **F o o o o o o o o o o I R X X**

With dynamic scheduling

1. **F2 := F4 / F6** — **F I R X X X X X M W**
2. **F10 := F2 + F8** — **F I o o o o o o o R X X M W**
3. **F12 := F6 – F14** — **F I R X X M W**

Example 2 (Possibility of new hazards)

1. **F2 := F4 * F6** F I R X X X M W
2. **F8 := F10 * F12** F o o o o I R X X X M W
3. **F8 := F14 / F6** F I R X X X X X M W*

Note. Verify that this WAW hazard was *not possible* with static scheduling.