CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Virtual Addresses

Programs typically reside on disk as binary executable files

A program must be placed in a process and loaded into memory before it can be run

Where that process resides in memory will not always be the same

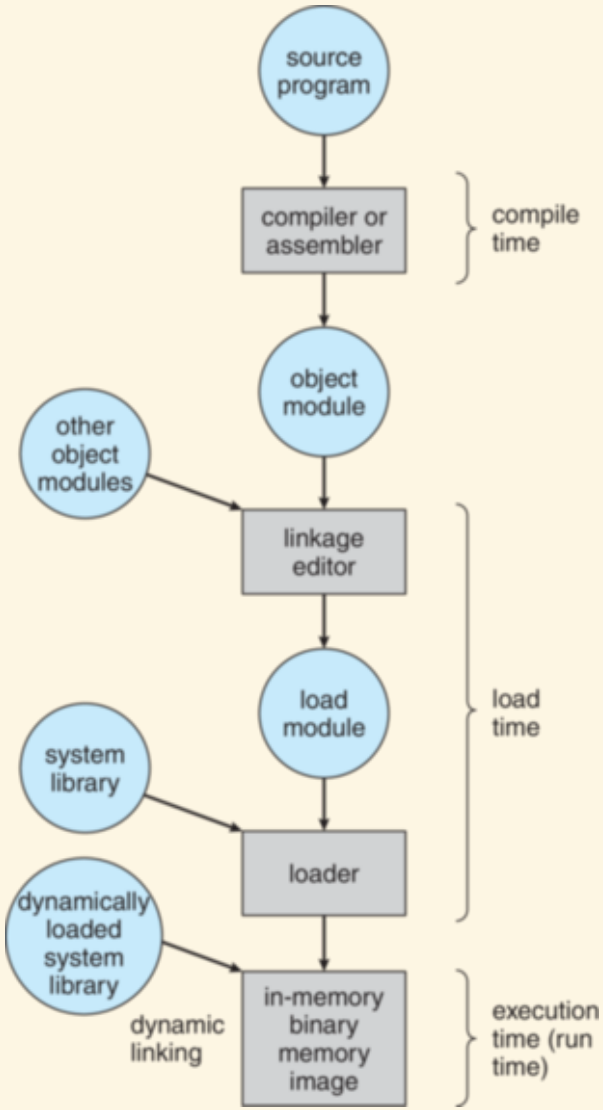
How do we know what memory addresses to access when writing code?

When going all the way from source code to a running process, addresses are represented several different ways

Programmers refer to memory by name (variables)

Compiler may **bind** variable to relocatable address ("32 bytes from start of this function")

Linker or loader may turn relocatable addresses into absolute addresses -- the ones actually seen by the machine


source program
compiler or assembler
compile time
object module
other object modules
linkage editor
load module
load time
system library
loader
dynamically loaded system library
dynamic linking
in-memory binary memory image
execution time (run time)

Not all of these steps necessarily occur

Binding to memory addresses can occur at various points

Compile time

If process always resides in same place in memory, compiler can generate **absolute code**

Changing were program is loaded will require recompiling code

Load time

When it is not known where process will reside, compiler generates **relocatable code** and final binding occurs at load time

Changing starting address requires simply reloading program

Execution time

What if process can be moved *during execution*?

Binding to memory addresses must be delayed until run time, requiring special hardware

More complicated, but used by most general-purpose OSes

Binding at execution time requires us to think more abstractly about addresses

Virtual addressing

Address generated by CPU (e.g., when requesting to load a variable from memory) is called **logical address**

Address seen by memory unit is **physical address**

For compile-time and load-time address binding,

logical address == physical address

To implement execution-time address binding, we allow logical address to be different from physical address

Logical address is also called **virtual address**

Logical address space is set of all logical addresses accessed by a program

Physical address space is set of all physical addresses corresponding to logical address space

For execution-time address binding,

logical address space != physical address space

How we manage separate logical and physical address spaces will be main question of memory management

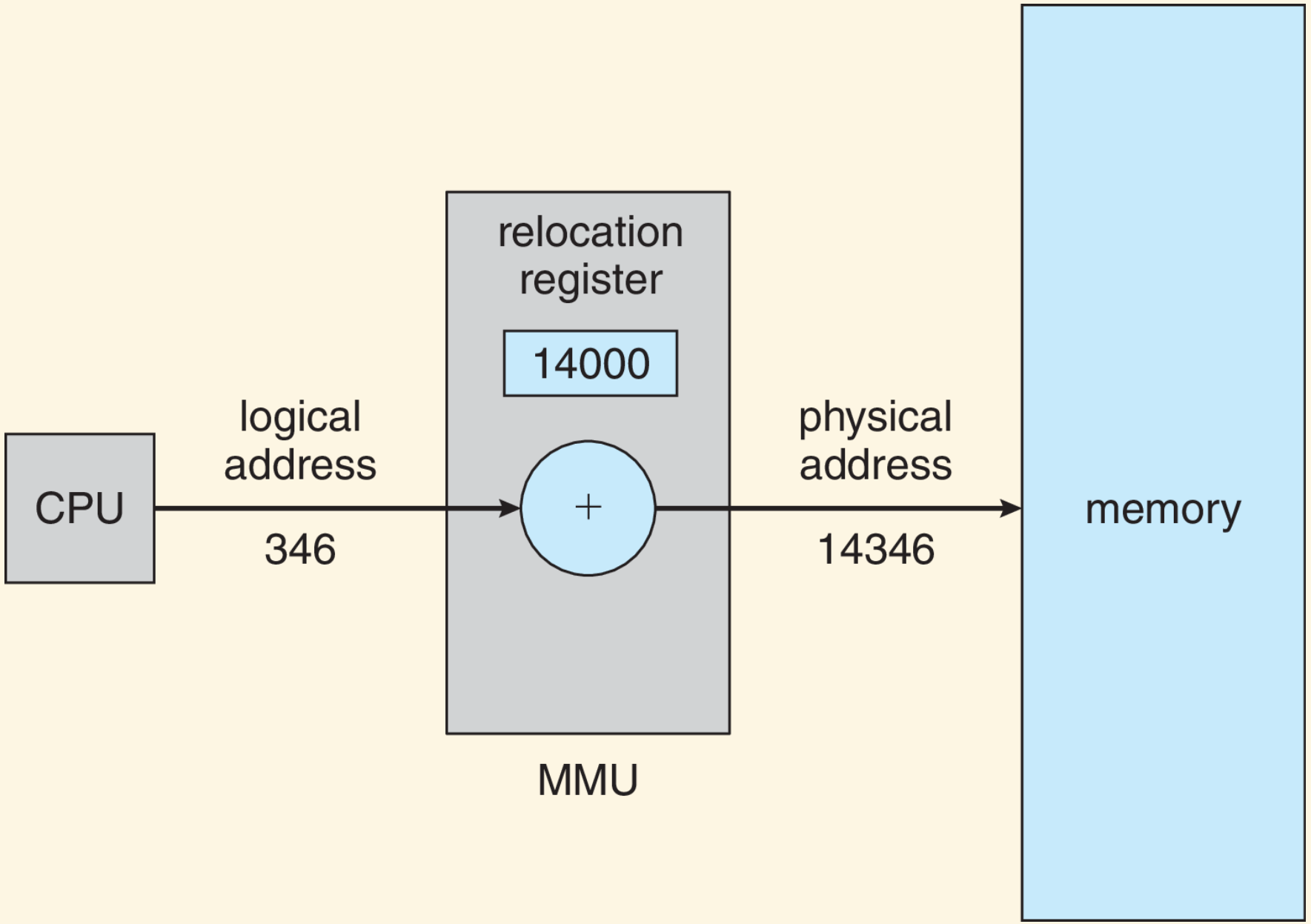
Memory-management unit (MMU) is hardware device for run-time mapping of virtual to physical addresses

We will see many different ways the MMU can do its job

One simple mapping is similar to what we saw previously used for memory protection

Keep a **relocation register** that stores first address of memory segment of running process

Every logical address generated by process is added to value in relocation register to get physical address


relocation
register
14000
logical
address
346
CPU
+
physical
address
14346
memory
MMU

MMU must convert logical addresses to physical addresses before they can be used

Notice that the physical address of an access is not determined until the access is actually made

Logical address space is always `[0, max]` for some `max`

Physical address space in previous example was `[R, R + max]` for value R in relocation register

Other (much more complicated) physical address spaces are possible

User program knows *only about logical addresses* and is *completely unaware* that translation is happening

User program has no idea what physical memory it runs on

Work of MMU to map between address spaces is transparent to user program