Virtual Memory

What happens when you want to use more memory than your computer actually has?

- Many programs in memory
- Large input files

Solution: use the next level of the memory hierarchy—the disk!

Virtual memory is for weenies – Seymour Cray

640K ought to be enough for anybody – Bill Gates, 1981

Virtual Memory

Goal: give the illusion of a large memory

Strategy:

- Break physical memory into blocks (*pages*)
- Page might be in physical memory or on disk

Addresses:

- Generated by loads/stores: virtual
- Actual memory locations: physical

Virtual Memory

Memory access:

- Load/store/pc computes virtual address
- Check virtual address

  Is the page in memory?

  Calculate physical address

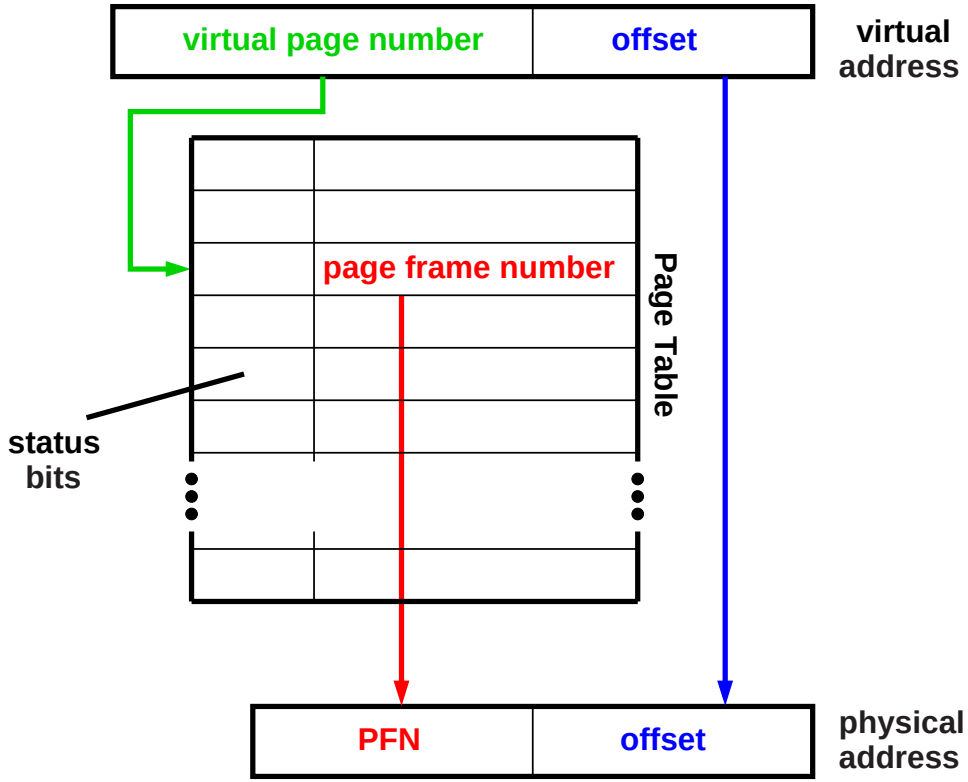
  Access memory with physical address

Called *address translation*

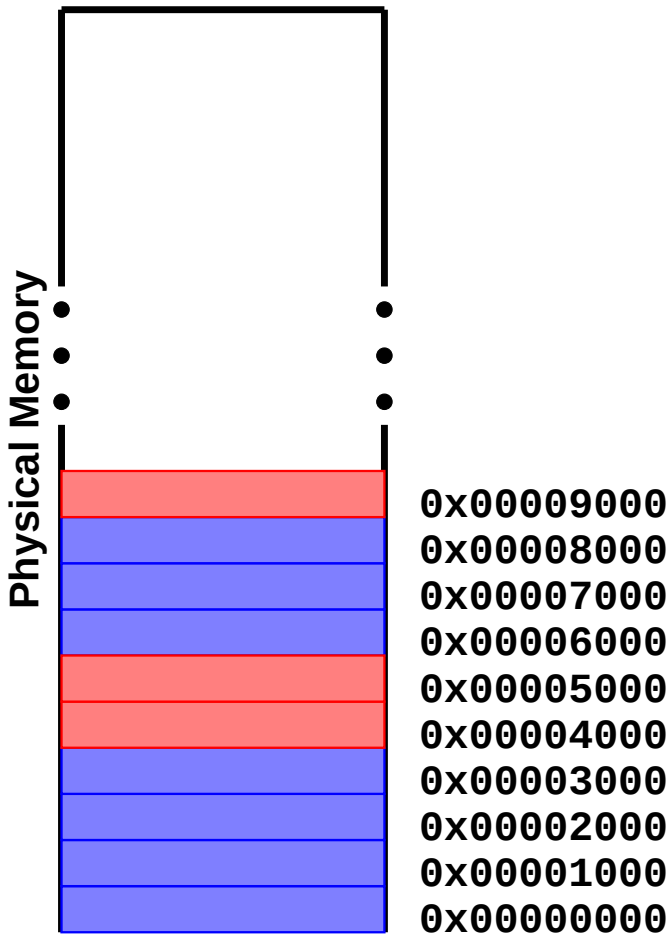
virtual addr to physical addr mapping

Page Tables

Page table: data structure used to store mapping from virtual to physical addresses.



Page Tables: Protection



Page Table:

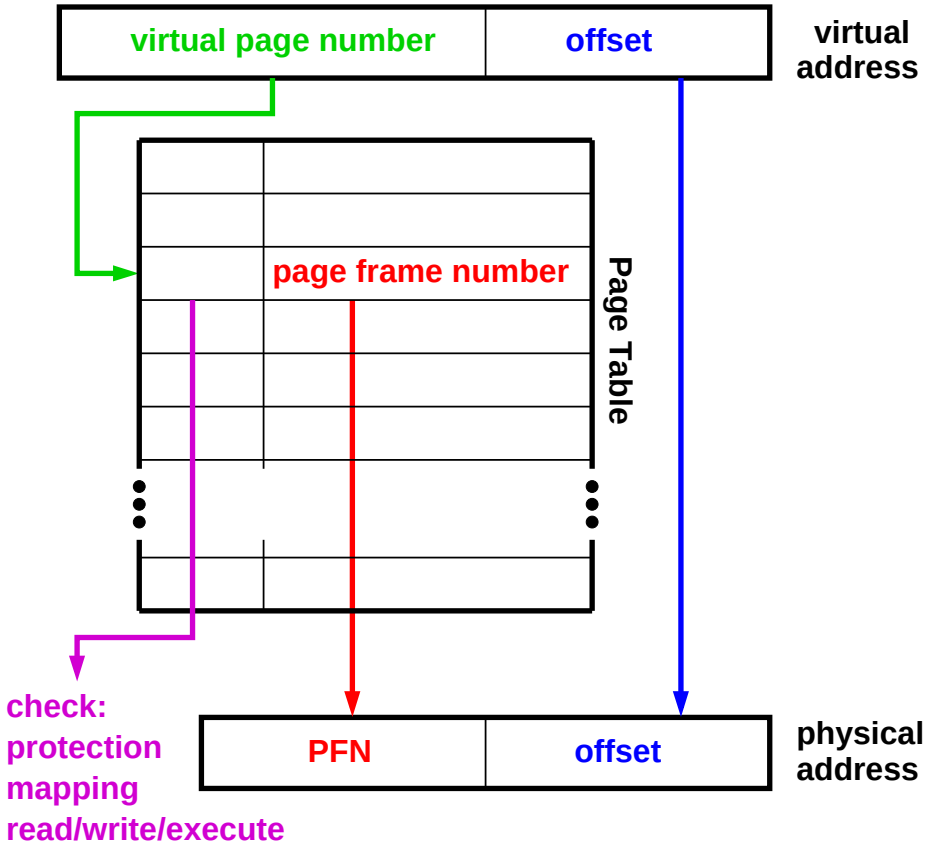
```
map=1, pid=0, 0x00400000 -> 0x00000000
map=1, pid=0, 0x00401000 -> 0x00001000
map=1, pid=0, 0x00402000 -> 0x00002000
map=1, pid=0, 0x00403000 -> 0x00003000
map=1, pid=0, 0x00404000 -> 0x00006000
map=1, pid=0, 0x00405000 -> 0x00007000
map=1, pid=0, 0x00406000 -> 0x00008000

map=1, pid=1, 0x00400000 -> 0x00004000
map=1, pid=1, 0x00401000 -> 0x00005000
map=1, pid=1, 0x00402000 -> 0x00009000
```

If the page is not present in memory, access disk.

Page Tables: Protection

Problem: disk access takes a very long time

⇒ run another program!



CSL

Address Translation

Virtual to Physical translation:

- Read page table entry
- If page mapped (valid)
  - check access permissions
  - if not allowed, *protection fault*
- If page not mapped, *page fault*

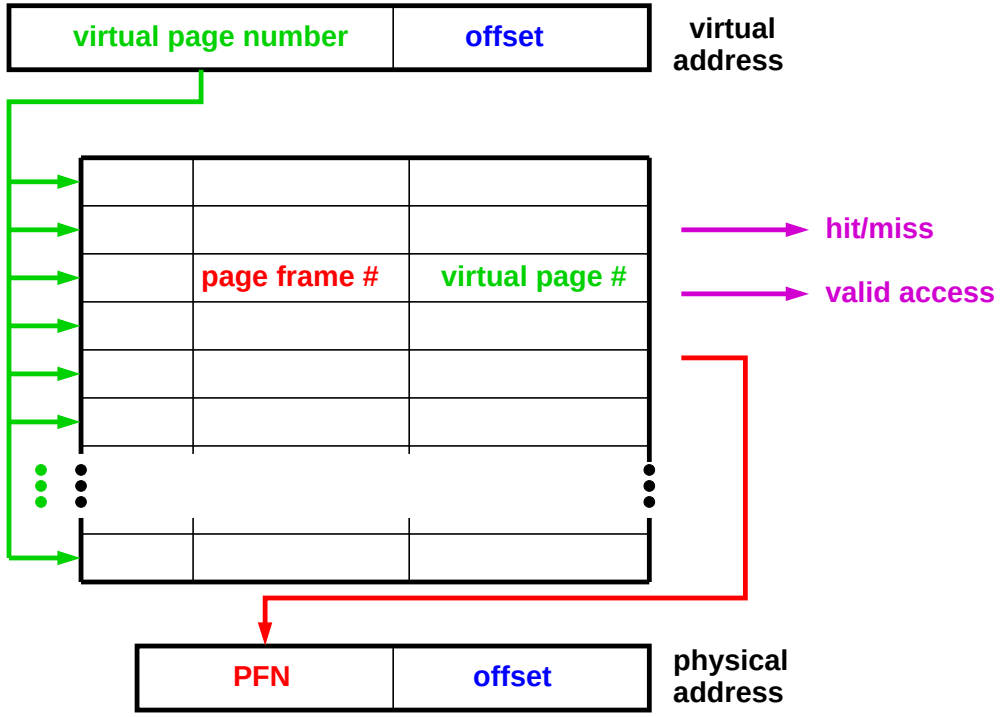
Software translation would be slow! (compiler can help here)

Hey look! It's not my fault! It's some guy named "General Protection." – Ratbert, 12/18/1996

Translation Lookaside Buffer (TLB)

TLB: a hardware cache for the page table.

A small, fully-associative cache.



CSL

TLB Miss Handling

Miss can be handled by software or hardware

- Software: operating system code reads page table
- Hardware: dedicated FSM

What happens on an error?

- add/sub overflow
- load/store to an unaligned address
- tlb miss
- protection fault
- page fault

Exceptions

Need some hardware mechanism that detects errors, and notifies software when errors occur.

- Detect type of error
- Detect where the error occurred

Various options: store error condition in special registers, have software check.

But exceptions are infrequent...

Hardware Exceptions

On an error, the hardware does the following:

- save the `PC` of the exception-causing instruction
- save the error type
- set `PC` to a pre-determined value
  the exception address or exception vector
  (sometimes a function of error type)
- Software exception handler located at the pre-determined `PC` value

Software handles the exception, potentially returning control to the program that raised the exception.

Precise Exceptions

What about instructions in the pipeline when an error is detected?

Simple solution (for software):

- Cancel result of instructions until we start executing exception handler
- Known as precise exceptions
- State at the beginning of exception handler matches state before the execution of the exception-causing instruction

Protection

Access to shared resources protected by the TLB.

How do we support TLB modifications?

new instructions

What prevents my program from modifying the TLB?

- Hardware modes (access control)
- User mode and kernel mode (can have more)
- Cannot modify TLB in user mode

How/when does the OS modify the TLB?

Protection

Uses exception mechanism.

- Exception address is fixed
- Store operating system code at that location
- Enter kernel mode on an exception
- Return to user mode when the OS is done