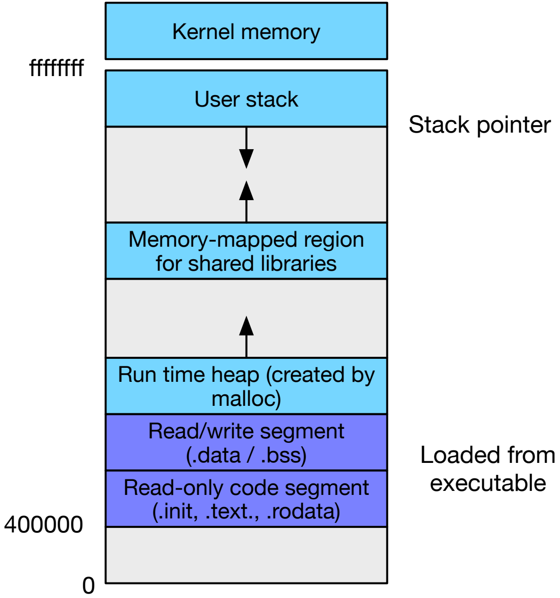
Lecture 22: Virtual Memory

Brennon Brimhall

30 June 2025

Recall: Process Address Space


Kernel memory
ffffffff
User stack
Stack pointer
Memory-mapped region for shared libraries
Run time heap (created by malloc)
Read/write segment (.data / .bss)
Loaded from executable
Read-only code segment (.init, .text., .rodata)
400000
0

Virtual Memory

- Abstraction of physical memory
- Purpose
  - appearance of more available memory than physically exists (DRAM)
  - handles disk caching / loading
  - insulates memory of each process

Virtual Memory

- Abstraction of physical memory
- Purpose
  - appearance of more available memory than physically exists (DRAM)
  - handles disk caching / loading
  - insulates memory of each process
- Page table: maps from virtual address to physical addresses

Virtual Memory

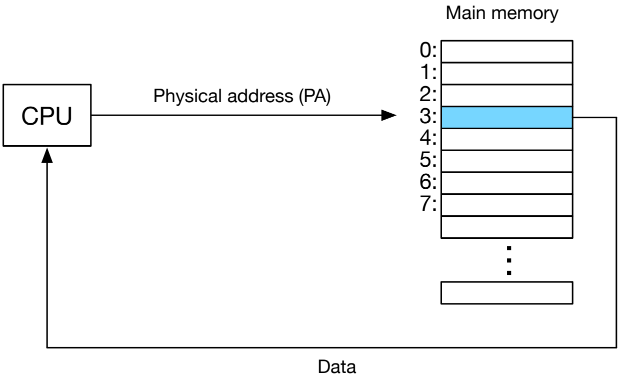
- Abstraction of physical memory
- Purpose
  - appearance of more available memory than physically exists (DRAM)
  - handles disk caching / loading
  - insulates memory of each process
- Page table: maps from virtual address to physical addresses
- Memory management unit (MMU):
  hardware implementation of address translation

Warning

- This is going to get very complex
- Closely tied with multi-tasking (multiple processes)
- Partly managed by hardware, partly managed by software

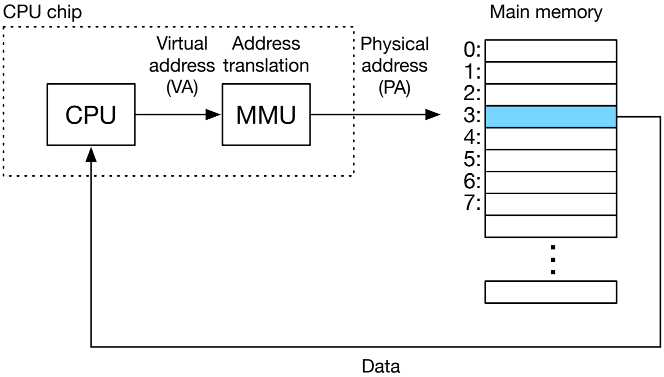
Virtual addressing

Physical Addressing



- So far, assumed CPU addresses physical memory

Virtual Addressing



- Memory management unit (MMU): maps virtual to physical addresses

Address Space

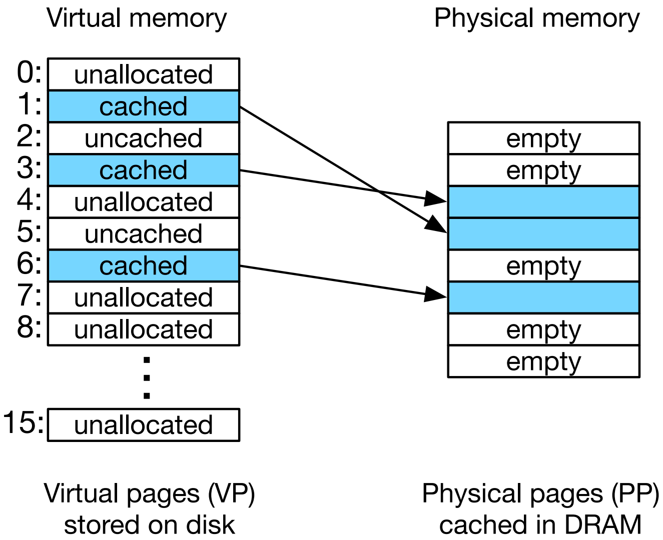
- Virtual memory size: $\mathsf{N} = 2^{\mathtt{n}}$ bytes, e.g., 256TB
- Physical memory size: $\mathsf{M} = 2^{\mathtt{m}}$ bytes, e.g., 16GB
- Page (block of memory): $\mathsf{P} = 2^{\mathtt{p}}$ bytes, e.g., 4KB
- A virtual address can be encoded in *n* bits

Caching

Caching... Again?

- Yes, we already discussed caching, but for on-chip cache of DRAM memory
- Now
  - caching between RAM and disk
  - driven by a large virtual memory address space
  - to avoid unnecessary and duplicate loading
- Jargon
  - previously “block”, now “page”
  - now: “swapping” or “paging”

Mapping



Virtual memory
Physical memory
0: unallocated
1: cached
2: uncached
3: cached
4: unallocated
5: uncached
6: cached
7: unallocated
8: unallocated
15: unallocated
empty
empty
empty
empty
empty
Virtual pages (VP) stored on disk
Physical pages (PP) cached in DRAM

State of Virtual Memory Page

- Cached
  - allocated page
  - stored in physical memory

State of Virtual Memory Page

- Cached
  - allocated page
  - stored in physical memory
- Uncached
  - allocated page
  - not in physical memory

State of Virtual Memory Page

- Cached
  - allocated page
  - stored in physical memory
- Uncached
  - allocated page
  - not in physical memory
- Unallocated
  - not used by virtual memory system so far

Page Table

- Array of page table entries (PTE)

Page Table

- Array of page table entries (PTE)
  (actually, a tree where the leaves store the page table entries)

Page Table

- Array of page table entries (PTE)
  (actually, a tree where the leaves store the page table entries)
- Each PTE maps a virtual page to a physical page

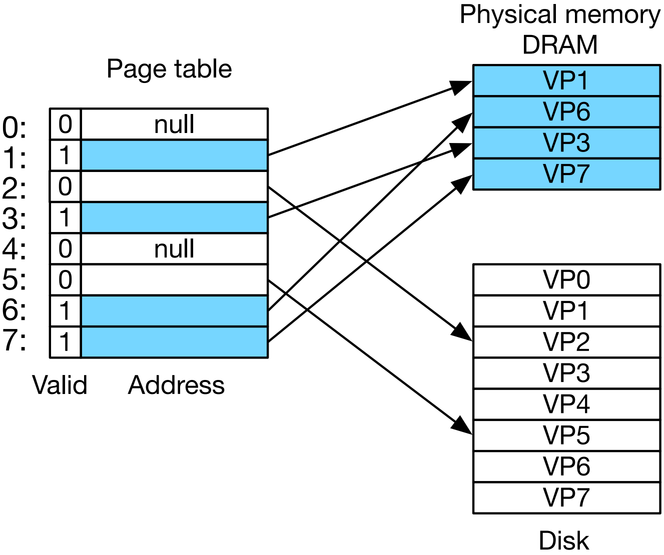
Page Table

- Array of page table entries (PTE)
  (actually, a tree where the leaves store the page table entries)
- Each PTE maps a virtual page to a physical page
- Valid bit
  - set if PTE currently maps to physical address (cached)
  - not set otherwise (uncached or unallocated)

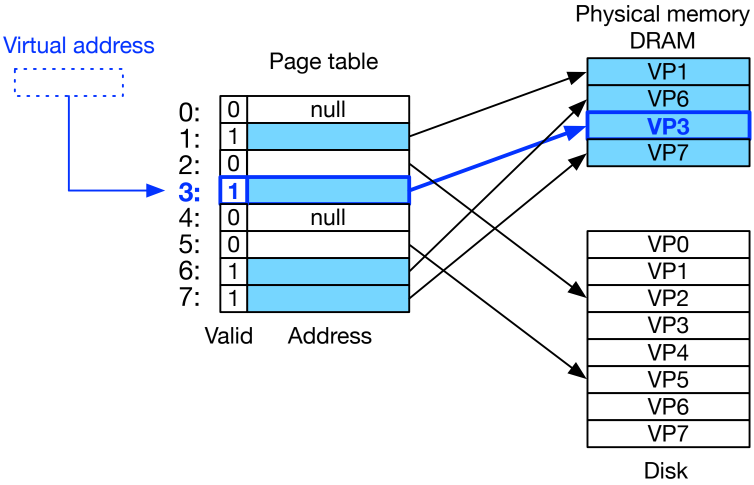
Page Table

- Array of page table entries (PTE)
  (actually, a tree where the leaves store the page table entries)
- Each PTE maps a virtual page to a physical page
- Valid bit
  - set if PTE currently maps to physical address (cached)
  - not set otherwise (uncached or unallocated)
- Mapped address
  - if cached: physical address in DRAM
  - if not cached: physical address on disk

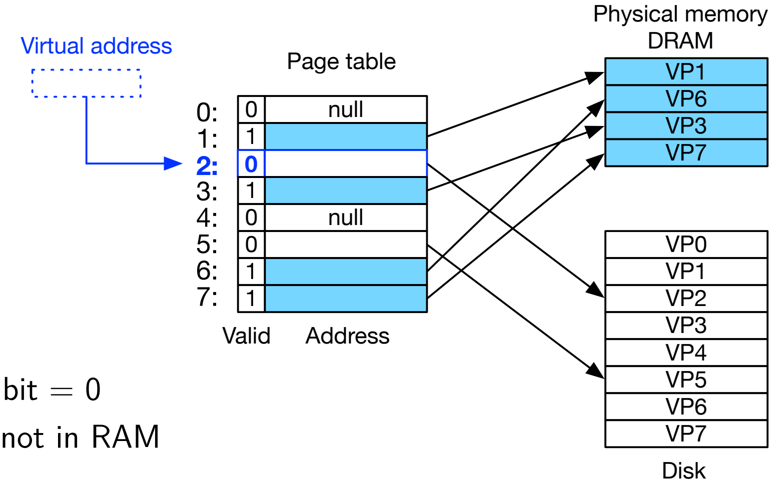
Page Table


Page table
Physical memory
DRAM
0:
1:
2:
3:
4:
5:
6:
7:
0
null
1
0
1
0
null
0
1
1
Valid
Address
VP1
VP6
VP3
VP7
VP0
VP1
VP2
VP3
VP4
VP5
VP6
VP7
Disk

Page Hit

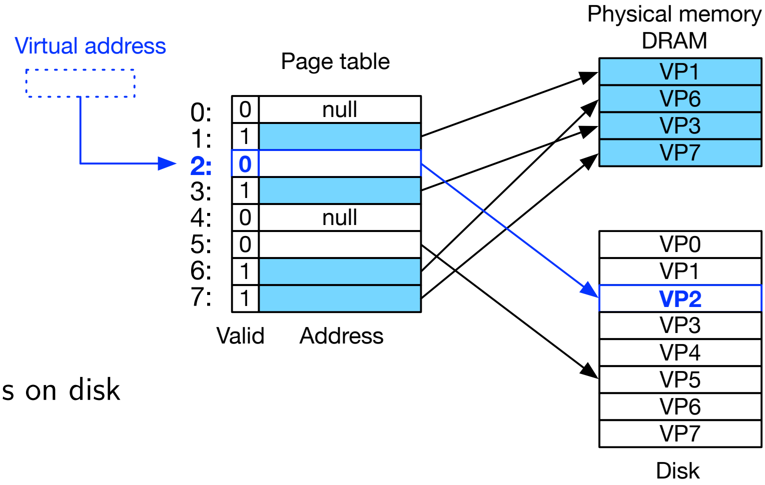

Virtual address
Page table
0: 0 null
1: 1
2: 0
3: 1
4: 0 null
5: 0
6: 1
7: 1
Valid
Address
Physical memory
DRAM
VP1
VP6
VP3
VP7
VP0
VP1
VP2
VP3
VP4
VP5
VP6
VP7
Disk

Page Fault



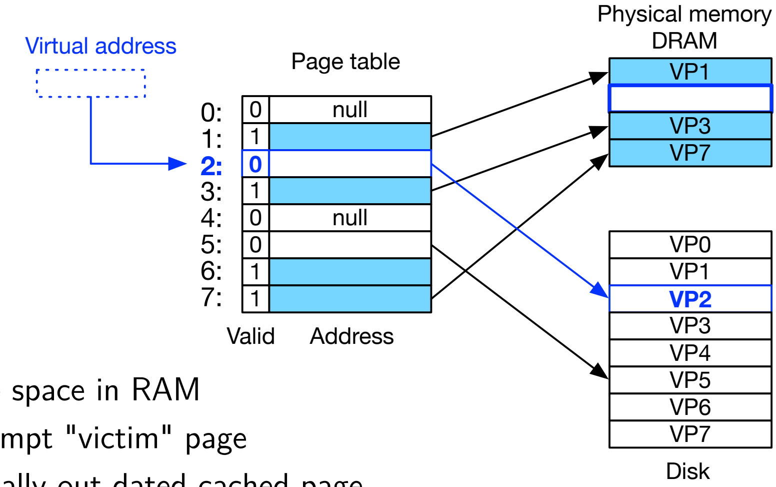
- Valid bit $= 0$
- Page not in RAM

Page Fault



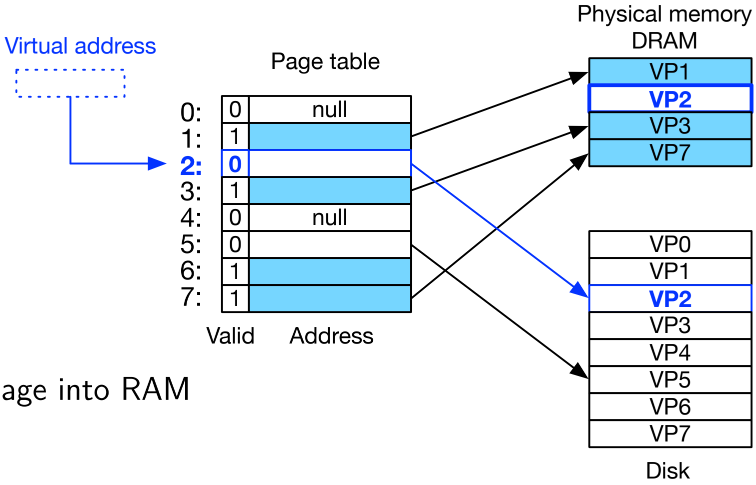
- Page is on disk

Page Fault



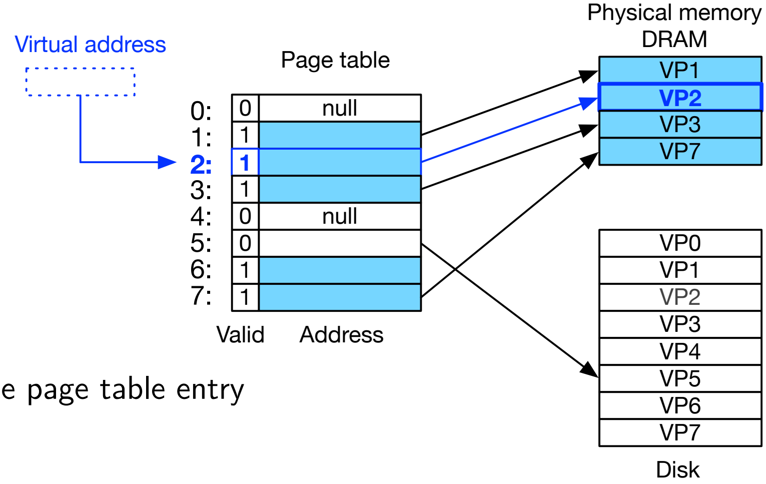
- Make space in RAM
- Pre-empt "victim" page
- Typically out-dated cached page

Page Fault



- Load page into RAM

Page Fault

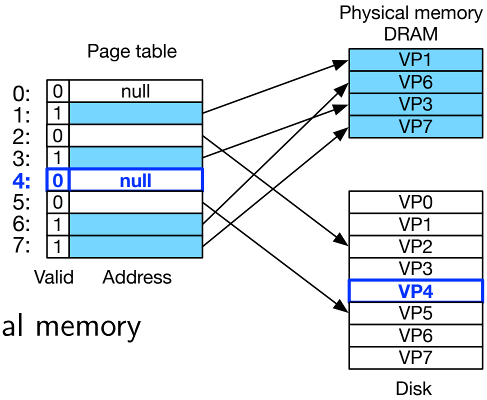


- Update page table entry

Allocating Pages

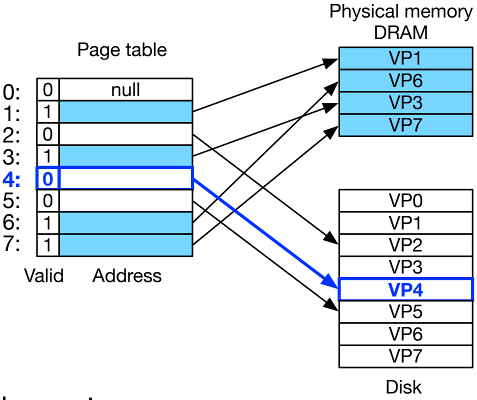
- What happens when we load a program?
- We need to load its executable into memory
- Similar: create data objects when program is running ("allocating" memory)

Allocating Page



- Identify space in virtual memory

Allocating Page



- Map to data on disk
  - do not actual load
  - just create page table entries
  - let virtual memory system handle loading

⇒ On-demand loading

Clicker quiz!

Clicker quiz omitted from public slides

Process Memory

- Nothing loaded at startup

Process Memory

- Nothing loaded at startup
- Working set (or resident set)
  - pages of a process that are currently in DRAM
  - loaded by virtual memory system on demand

Process Memory

- Nothing loaded at startup
- Working set (or resident set)
  - pages of a process that are currently in DRAM
  - loaded by virtual memory system on demand
- Thrashing
  - memory actively required by all processes larger than physically available
  - frequent swapping of memory to/from disk
  - very bad: slows down machine dramatically

Acknowledgements

Slides adapted from materials provided by David Hovemeyer.