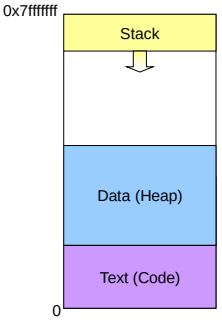
Virtual Memory

Process's Address Space

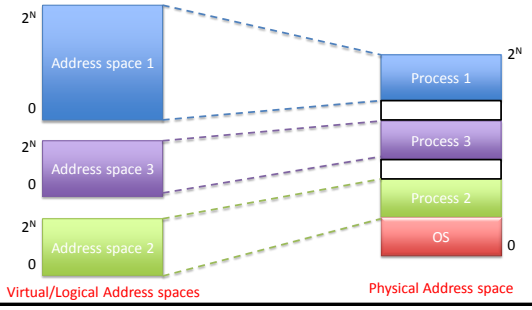


Goals for Multiprogramming

- Sharing
  - Several processes coexist in main memory
- Transparency
  - Processes not aware memory is shared
  - Run regardless of number and/or locations of processes
- Protection
  - Cannot corrupt OS or other processes
  - Privacy: Cannot read data of other processes
- Efficiency should not be severely degraded
  - Purpose of sharing is to increase efficiency
  - CPU and memory resources not wasted

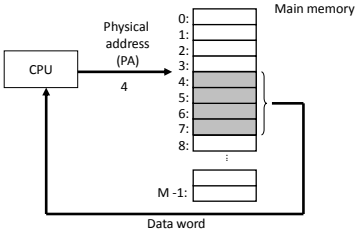
Address Spaces

- Translation from logical to physical addresses



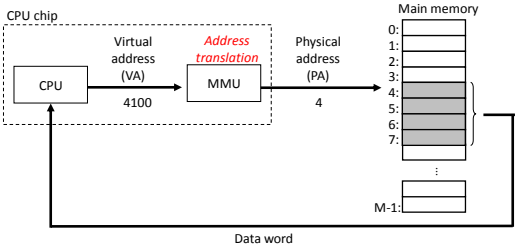
A system with physical addressing

- Main memory - An array of M contiguous byte-sized cells, each with a unique physical address
- Physical addressing
  - Most natural way to access it – Addresses generated by the CPU correspond to bytes in it
  - Used in simple systems like early PCs and embedded microcontrollers (e.g. cars and elevators)



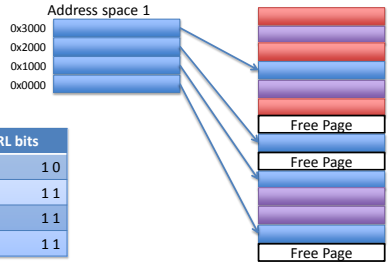
A system with virtual addressing

- Modern processors use virtual addresses
- CPU generates virtual address and address translation is done by dedicated hardware (*memory management unit*) via OS-managed lookup table



Virtual Memory Example

- Mapping of virtual addresses to physical memory



Page Table for process 1

Base Address	CTRL bits
1	1 0
4	1 1
6	1 1
10	1 1

Pages

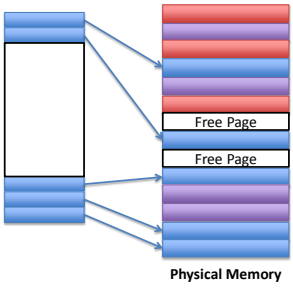
- Break memory up into fixed sized chunks
- Fast to allocate and free
- Easier to translate, less translations

- Mostly 4KB, but not always
  - 32 bit x86 supports 4KB and 4MB
  - 64 bit x86 supports 4KB, 2MB and 1GB

Paging with Large Address Spaces

- Mapping of logical addresses to physical memory
- Page table for process

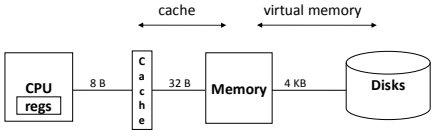
Base Address	CTRL bits
0	1 0
1	1 0
4	1 1
… skipped entries..	0 0
6	1 1
10	1 1



Motivations for virtual memory

- Use physical DRAM as a cache for the disk
  - Address space of a process can exceed physical memory
  - Sum of address spaces of multiple processes can exceed physical memory
- Simplify memory management
  - Multiple processes resident in main memory
    - Each process with its own address space
  - Only "active" code and data is actually in memory
    - Allocate more memory to process as needed
- Provide protection
  - One process can't interfere with another
    - Because they operate in different address spaces
  - User process cannot access privileged information
    - Different sections of address spaces have different permissions.

Levels in memory hierarchy



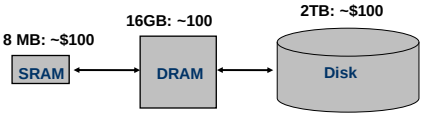
	Register	Cache	Memory	Disk Memory
size:	32 B	32KB-4MB	1024 MB	100 GB
speed:	1 ns	2 ns	30 ns	8 ms
$/Mbyte:		$125/MB	$0.20/MB	$0.001/MB
line size:	8 B	32 B	4 KB	

10x slower *100,000x slower*

larger, slower, cheaper

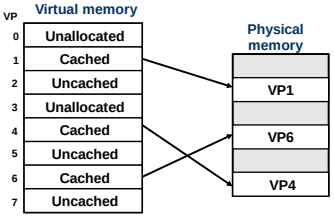
Motivation: DRAM a "cache" for disk

- Full address space is quite large:
  - 32-bit addresses: ~4 Gigabytes (4 billion) bytes
  - 64-bit addresses: ~16 Exabytes (16 quintillion) bytes
- Disk storage is ~100X cheaper than DRAM storage
  - 2TB of DRAM: ~ $10,000
  - 2TB GB of disk: ~$100
- To access memory in a reasonable amount of time it must be stored in memory
- To access large amounts of data in a cost-effective manner, the bulk of the data must be stored on disk



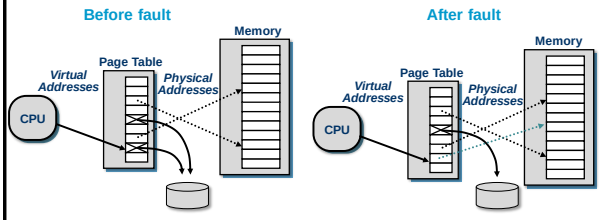
Virtual and physical pages

- Data on disk is partitioned into fixed sized blocks – virtual pages
- Physical memory into blocks of equal size – physical pages or page frames
- At any point, VM pages are partitioned into three sets
  - Unallocated
  - Allocated
  - Cached



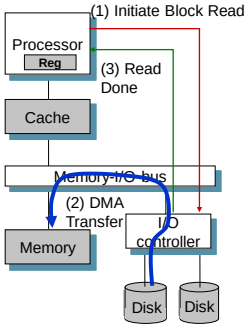
Page faults (like "cache misses")

- What if an object is on disk rather than in memory?
  - Page table entry indicates virtual address not in memory
  - OS exception handler invoked to move data from disk into memory
    - Current process suspends, others can resume
    - OS has full control over placement, etc.



Servicing a page fault

- Processor signals controller
  - Read block of length P starting at disk address X and store starting at memory address Y
- Read occurs
  - Direct Memory Access (DMA)
  - Under control of I/O controller
- I / O controller signals completion
  - Interrupt processor
  - OS resumes suspended process

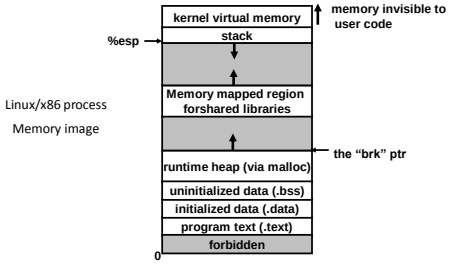


Motivations for virtual memory

- Use physical DRAM as a cache for the disk
  - Address space of a process can exceed physical memory size
  - Sum of address spaces of multiple processes can exceed physical memory
- **Simplify memory management**
  - **Multiple processes resident in main memory**
    - **Each process with its own address space**
  - **Only "active" code and data is actually in memory**
    - **Allocate more memory to process as needed**
- Provide protection
  - One process can't interfere with another
    - Because they operate in different address spaces
  - User process cannot access privileged information
    - Different sections of address spaces have different permissions.

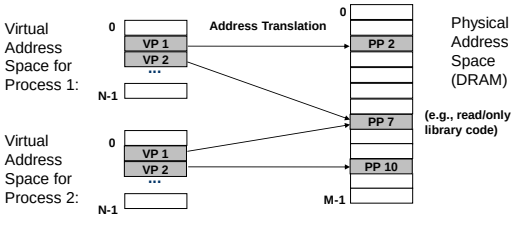
Motivation #2: Memory management

- Multiple processes can reside in physical memory.
- How do we resolve address conflicts?
  - what if two processes access something at the same address?



Solution: Separate virtual addr. spaces

- Each process has its own virtual address space
  - OS controls how virtual pages are assigned to physical mem.

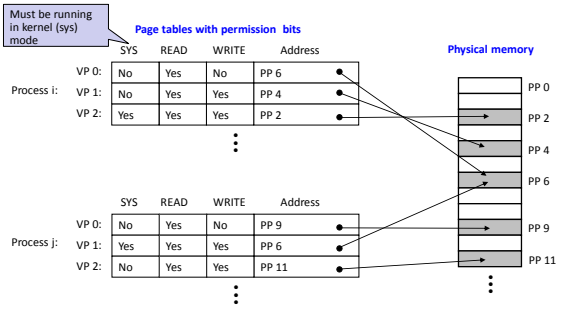


Motivations for virtual memory

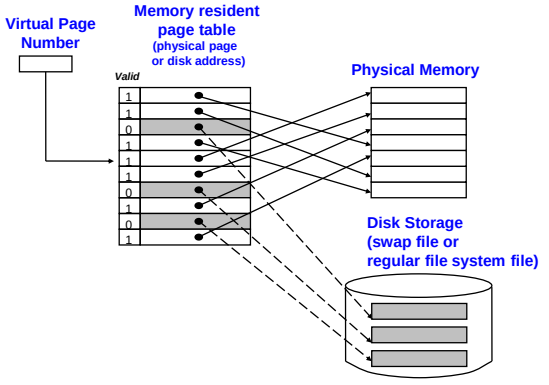
- Use physical DRAM as a cache for the disk
  - Address space of a process can exceed physical memory size
  - Sum of address spaces of multiple processes can exceed physical memory
- Simplify memory management
  - Multiple processes resident in main memory
    - Each process with its own address space
  - Only "active" code and data is actually in memory
    - Allocate more memory to process as needed
- Provide protection
  - One process can't interfere with another
    - Because they operate in different address spaces
  - User process cannot access privileged information
    - Different sections of address spaces have different permissions

Motivation #3: Protection

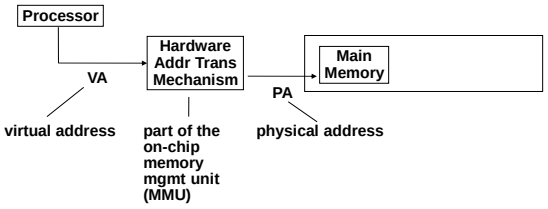
- Page table entry contains access rights information
  - HW enforces this protection (trap into OS if violation occurs)



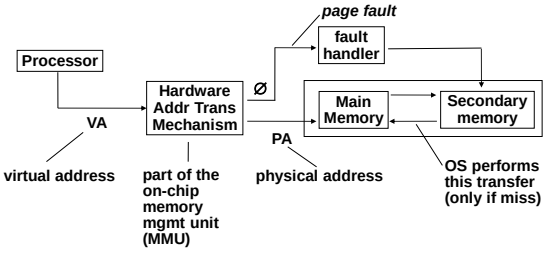
Page tables



VM address translation: Hit

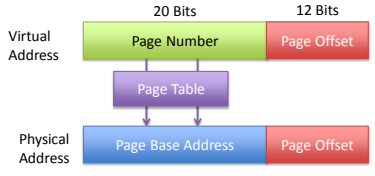


VM address translation: Miss



Page Translation

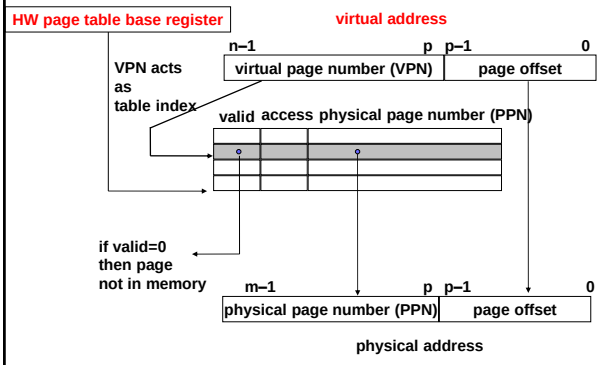
- How are virtual addresses translated to physical addresses
  - Upper bits of address designate page number



4K Pages

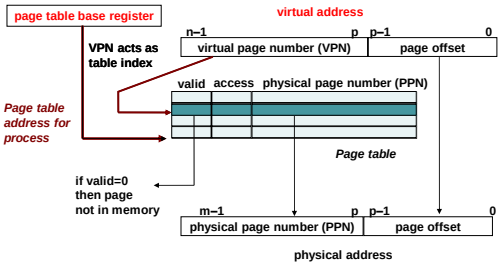
- 1 page table per process: One entry per page in address space
  - Base address of each page in physical memory
  - Read/Write protection bits

Address translation via page table



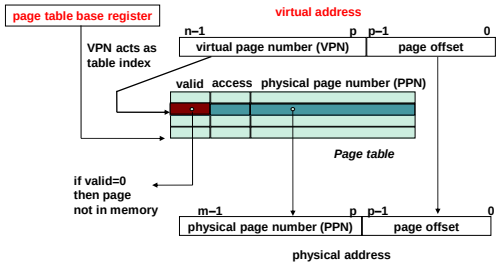
Page table operation

- Translation
  - Separate (set of) page table(s) per process
  - VPN forms index into page table (points to a page table entry)



Page table operation

- Computing physical address
  - Page Table Entry (PTE) provides info about page
    - if (valid bit = 1) then the page is in memory.
      - Use physical page number (PPN) to construct address
    - if (valid bit = 0) then the page is on disk - page fault



Page table operation

- Checking protection
  - Access rights field indicate allowable access
    - e.g., read-only, read-write, execute-only
    - typically support multiple protection modes
  - Protection violation fault if user doesn't have necessary permission