Summary so far

- The speed gap between CPU, memory and mass storage continues to widen.

- Well-written programs exhibit a property called locality.

- Memory hierarchies based on caching close the gap by exploiting locality.
Sharing Memory

- Rewind to the days of “second-generation” computers
  - Programs use physical addresses directly
  - OS loads job, runs it, unloads it

- Multiprogramming changes all of this
  - Want multiple processes in memory at once
    » Overlap I/O and CPU of multiple jobs
  - How to share physical memory across multiple processes?
    » Many programs do not need all of their code and data at once (or ever) – no need to allocate memory for it
    » A program can run on machine with less memory than it “needs”
Virtual Addresses

- To make it easier to manage the memory of processes running in the system, we’re going to make them use **virtual addresses** (logical addresses)
  - Virtual addresses are independent of the actual physical location of the data referenced
  - OS determines location of data in physical memory
- Instructions executed by the CPU issue virtual addresses
  - Virtual addresses are translated by hardware into physical addresses (with help from OS)
  - The set of virtual addresses that can be used by a process comprises its **virtual address space**
Virtual Addresses

- Many ways to do this translation…
  - Need hardware support and OS management algorithms

- Requirements
  - Need protection – restrict which addresses jobs can use
  - Fast translation – lookups need to be fast
  - Fast change – updating memory hardware on context switch
Fixed Partitions

- Physical memory is broken up into fixed partitions
  - Size of each partition is the same and fixed
  - Hardware requirements: base register
  - Physical address = virtual address + base register
  - Base register loaded by OS when it switches to a process
Fixed Partitions

Base Register
P4’s Base

Virtual Address
Offset

How do we provide protection?

Physical Memory

P1
P2
P3
P4
P5
Fixed Partitions

● Advantages
  ◆ Easy to implement
    » Need base register
    » Verify that offset is less than fixed partition size
  ◆ Fast context switch

● Problems?
  ◆ Internal fragmentation: memory in a partition not used by a process is not available to other processes
  ◆ Partition size: one size does not fit all (very large processes?)
Variable Partitions

- Natural extension – physical memory is broken up into variable sized partitions
  - Hardware requirements: base register and limit register
  - Physical address = virtual address + base register

- Why do we need the limit register?
  - Protection: if (virtual address > limit) then fault
Variable Partitions

Virtual Address
Offset

Base Register
P3’s Base

Limit Register
P3’s Limit

Yes?

No?

Protection Fault

Yes?

No?
Variable Partitions

● Advantages
  ◆ No internal fragmentation: allocate just enough for process

● Problems?
  ◆ External fragmentation: job loading and unloading produces empty holes scattered throughout memory
Paging

- Paging solves the external fragmentation problem by using fixed sized units in both physical and virtual memory.
Process Perspective

- Processes view memory as one contiguous address space from 0 through N
  - Virtual address space (VAS)
- In reality, pages are scattered throughout physical storage
- The mapping is invisible to the program
- Protection is provided because a program cannot reference memory outside of its VAS
  - The address “0x1000” maps to different physical addresses in different processes
Paging

- Translating addresses
  - Virtual address has two parts: virtual page number and offset
  - Virtual page number (VPN) is an index into a page table
  - Page table determines page frame number (PFN)
  - Physical address is PFN::offset

- Page tables
  - Map virtual page number (VPN) to page frame number (PFN)
    - VPN is the index into the table that determines PFN
  - One page table entry (PTE) per page in virtual address space
    - Or, one PTE per VPN
Page Lookups

Virtual Address

Page number

Offset

Page Table

Page frame

Physical Address

Page frame

Offset

Physical Memory
Paging Example

- Pages are 4KB
  - Offset is 12 bits (because 4KB = $2^{12}$ bytes)
  - VPN is 20 bits (32 bits is the length of every virtual address)
- Virtual address is 0x7468
  - Virtual page is 0x7, offset is 0x468
- Page table entry 0x7 contains 0x2000
  - Page frame number is 0x2000
  - Seventh virtual page is at address 0x2000 (2nd physical page)
- Physical address = 0x2000 + 0x468 = 0x2468
Page Table Entries (PTEs)

- Page table entries control mapping
  - The **Modify** bit says whether or not the page has been written
    » It is set when a write to the page occurs
  - The **Reference** bit says whether the page has been accessed
    » It is set when a read or write to the page occurs
  - The **Valid** bit says whether or not the PTE can be used
    » It is checked each time the virtual address is used (**Why?**)
  - The **Protection** bits say what operations are allowed on page
    » Read, write, execute (**Why do we need these?**)
  - The **page frame number** (PFN) determines physical page
Paging Advantages

- Easy to allocate memory
  - Memory comes from a free list of fixed size chunks
  - Allocating a page is just removing it from the list
  - External fragmentation not a problem
    - All pages of the same size

- Simplifies protection
  - All chunks are the same size
  - Like fixed partitions, don’t need a limit register

- Simplifies virtual memory – later
Paging Limitations

- Can still have internal fragmentation
  - Process may not use memory in multiples of a page

- Memory reference overhead
  - 2 references per address lookup (page table, then memory)
  - What can we do?

- Memory required to hold page table can be significant
  - Need one PTE per page
  - 32 bit address space w/ 4KB pages = \(2^{20}\) PTEs
  - 4 bytes/PTE = 4MB/page table
  - 25 processes = 100MB just for page tables!
  - What can we do?
Segmentation

- Segmentation is a technique that partitions memory into logically related data units
  - Module, procedure, stack, data, file, etc.
  - Units of memory from user’s perspective

- Natural extension of variable-sized partitions
  - Variable-sized partitions = 1 segment/process
  - Segmentation = many segments/process
  - Fixed partition: Paging :: Variable partition: Segmentation

- Hardware support
  - Multiple base/limit pairs, one per segment (segment table)
  - Segments named by #, used to index into table
  - Virtual addresses become <segment #, offset>
Segment Lookups

Virtual Address → Segment Table

- Segment #
- Offset

Segment Table:
- limit
- base

Yes? → + → Physical Memory

No? → Protection Fault

≥ Segments Table
≤ Base

Protection Fault