Today

- Memory
- Memory Management

Abstraction

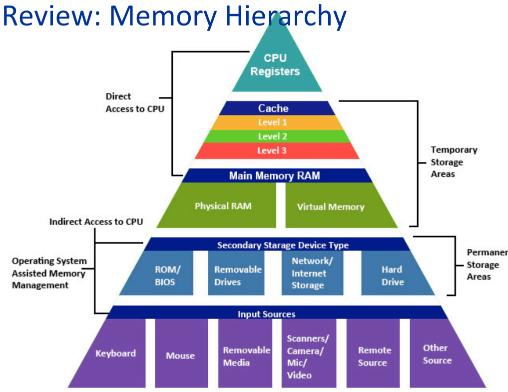
- Separate:
  - Interface from internals
  - Specification from implementation
- Abstraction is a double-edged sword
  - "Don't hide power."
- More than an interface…
  - It's a **contract** for how an object is to be used and how it is to behave, across many variations of its specific use and implementation.
  - We want abstractions that are simple, powerful, efficient to implement, and long-lasting.

Memory Management

- Basic hardware capabilities
  - Logical vs. Physical addresses
  - Address binding
- Multiprogramming and Memory
- Virtual Memory

Ties in with Project 5: Processes & Multiprogramming

Review: Memory Hierarchy



Memory Hierarchy

Hit Cost: cost to access

Cache	Hit Cost	Size
1st level cache/first level TLB	1 ns	64 KB
2nd level cache/second level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 μs	100 TB
Local non-volatile memory	100 μs	100 GB
Local disk	10 ms	1 TB
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 XB

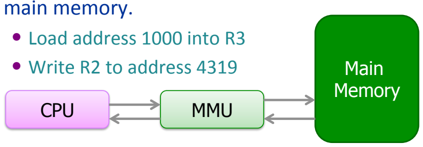
i7 has >10MB as shared 3rd level cache; 2nd level cache is per-core

Memory Management

- Processor can only directly use data from registers
  - Need to move data closer (memory)
- Ideally, programmers want memory that is large, fast, & non volatile
- Memory hierarchy
  - Small amount of fast, expensive memory – cache
  - Some medium-speed, medium-price – main memory
  - Gigabytes of slow, cheap disk storage – swap/virtual memory
- Multiprogramming makes memory management trickier

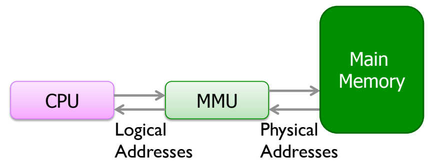
Basic Hardware Capabilities

- Assumptions about basic computer hardware:
  - For instructions to be executed or data to be accessed, they must be contained in main memory.
  - Program execution generates a stream of memory references that are sent from the CPU to the ***m**emory **m**anagement **u**nit*, controls access to the main memory.
    - Load address 1000 into R3
    - Write R2 to address 4319



Logical vs Physical Addresses

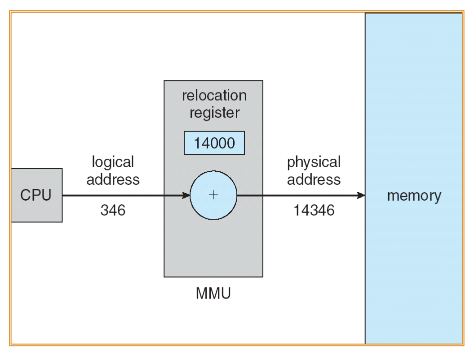
- Logical Addresses: The addresses issued by a program to the MMU.
- Physical Addresses: Addresses issued by the MMU to the main memory.



Memory-Management Unit (MMU)

- MMU = Hardware that maps virtual to physical addresses
- The user program deals with *logical* addresses
  - never sees the *real* physical addresses
- At runtime, relocation register added to every address generated by user process before being sent to memory

Dynamic Relocation Using Relocation Register



Absolute vs. Relative Programs

- In an *absolute program* logical addresses and physical addresses are the same.
  - i.e., program contains physical addresses
  - With respect to multiprogramming:
    - Limitations?
    - Benefits?
- In a *relative program* logical memory addresses in an executable program are relative to some base address
  - e.g., the start of the program, a segment register, etc.
  - Requires hardware support

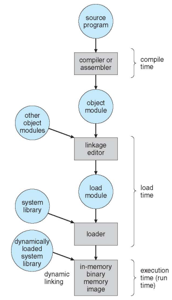
Address Binding

- The process of mapping the *logical* memory addresses contained in the executable program to the *physical* memory addresses where the program and data are actually located.
- Address binding techniques – differentiated by time
  - Compile time
  - Load time
  - Run time

Binding of Instructions and Data to Memory

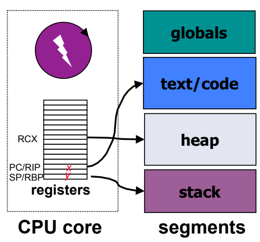
- Timing of when address binding can happen:
  - **Compile time**: If memory location known a priori, absolute code can be generated; must recompile code if starting location changes
  - **Load time**: Must generate relocatable code if memory location is not known at compile time
  - **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - Need hardware support for address maps (e.g., base and limit registers)

Multistep Processing of a User Program



Memory segments: a view from C

- Globals:
  - Fixed-size segment
  - Writable by user program
  - May have initial values
- Text (instructions)
  - Fixed-size segment
  - Executable
  - Not writable
- Heap and Stack
  - Variable-size segments
  - Writable
  - Zero-filled on demand



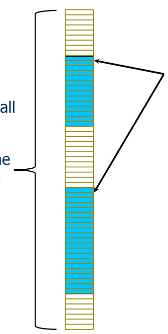
RCX – Counter
RIP – Instruction pointer register
RBP – Base pointer register

Heap: dynamic memory

Heap segment. A contiguous chunk of memory obtained from OS kernel.
E.g., with Unix *sbrk*() syscall

A runtime library obtains the block and manages it as a "heap" for use by the programming language environment, to store dynamic objects.

E.g., with Unix *malloc* and *free* library calls.



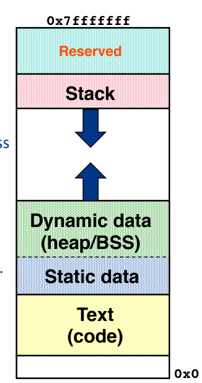
Allocated heap blocks for structs or objects.

Heap abstraction, simplified

1. User program calls heap manager to allocate a block of any desired size to store some dynamic data.
2. Heap manager returns a pointer to a block. The program uses that block for its purpose. The block's memory is reserved exclusively for that use.
3. Program calls heap manager to free (deallocate) the block when the program is done with it.
4. Once the program frees the block, the heap manager may reuse the memory in the block for another purpose.
5. User program is responsible for initializing the block, and deciding what to store in it. Initial contents could be old. Program must not try to use the block after freeing it.

VAS example (32-bit)

- The program uses virtual memory through its process' Virtual Address Space:
  - An addressable array of bytes…
  - Containing every instruction the process thread can execute…
  - And every piece of data those instructions can read/write…
    - i.e., read/write == load/store on memory
  - Partitioned into logical segments (regions) with distinct purpose and use.
  - Every memory reference by a thread is interpreted in the context of its VAS.
    - Resolves to a location in machine memory



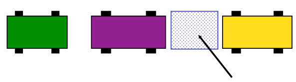
Heap blocks are contiguous

The storage in a heap block is contiguous in the Virtual Address Space. The term **block** always refers to a contiguous sequence of bytes suitable for **base+offset** addressing.

C and other PL environments require this. E.g., C compiler determines the offsets for named fields in a **struct** and "bakes" them into the code.

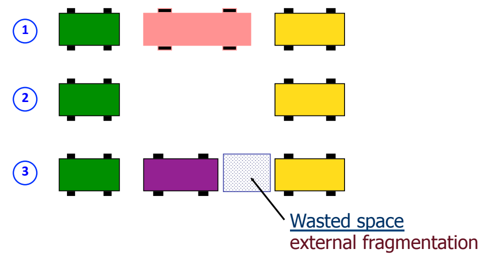
This requirement complicates the heap manager because the heap blocks may be different sizes (must use Variable Partitioning).

How to pack them into the available space in the heap region?



Variable Partitioning

Variable partitioning is the strategy of parking differently sized cars along a street with no marked parking space dividers.



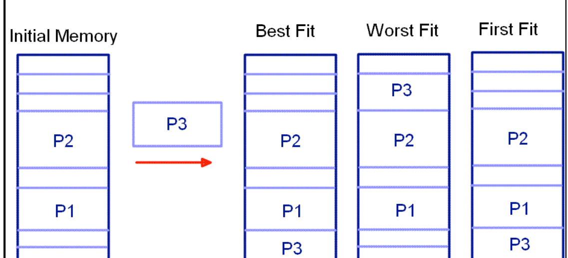
Heap manager policy

- The heap manager must find a suitable free block to return for each call to malloc().
  - No byte can be part of two simultaneously allocated heap blocks!
  - If any byte of memory is doubly allocated, programs will fail. We test for this!
- A heap manager has a policy algorithm to identify a suitable free block within the heap.
  - What should that policy be?

Heap manager policy

- The heap manager must find a suitable free block to return for each call to malloc().
  - No byte can be part of two simultaneously allocated heap blocks!
  - If any byte of memory is doubly allocated, programs will fail. We test for this!
- A heap manager has a policy algorithm to identify a suitable free block within the heap.
  - Last fit, first fit, best fit, worst fit
  - Choose your favorite!
  - Goals:
    - be quick (first-fit)
    - use memory efficiently (others)
  - Behavior depends on **workload**: pattern of malloc/free requests
- This is an old problem in computer science, and it occurs in many settings: variable partitioning.

Best fit, worst fit, first fit



[http://www.r9paul.org/blog/2008/managing-your-memory/]

Policy Tradeoff Discussion

- First
- Best
- Worst

Metrics: Speed, Fragmentation

Best fit, worst fit, first fit

- People used to study the relative merits of these algorithms for variable partitioning. Let's not.
- Which is best at reducing fragmentation?
  - "It depends."
  - Depends on workload:
    - the particular pattern of requests (e.g., malloc/free) that we receive.
    - Sizes requested
    - Order of malloc/free
- In general, we won't know the workload in advance, and we avoid assumptions about it that limit generality.
- But if we do know in advance, then we can optimize.

Looking Ahead

- Project 5
  - Due Friday, Dec