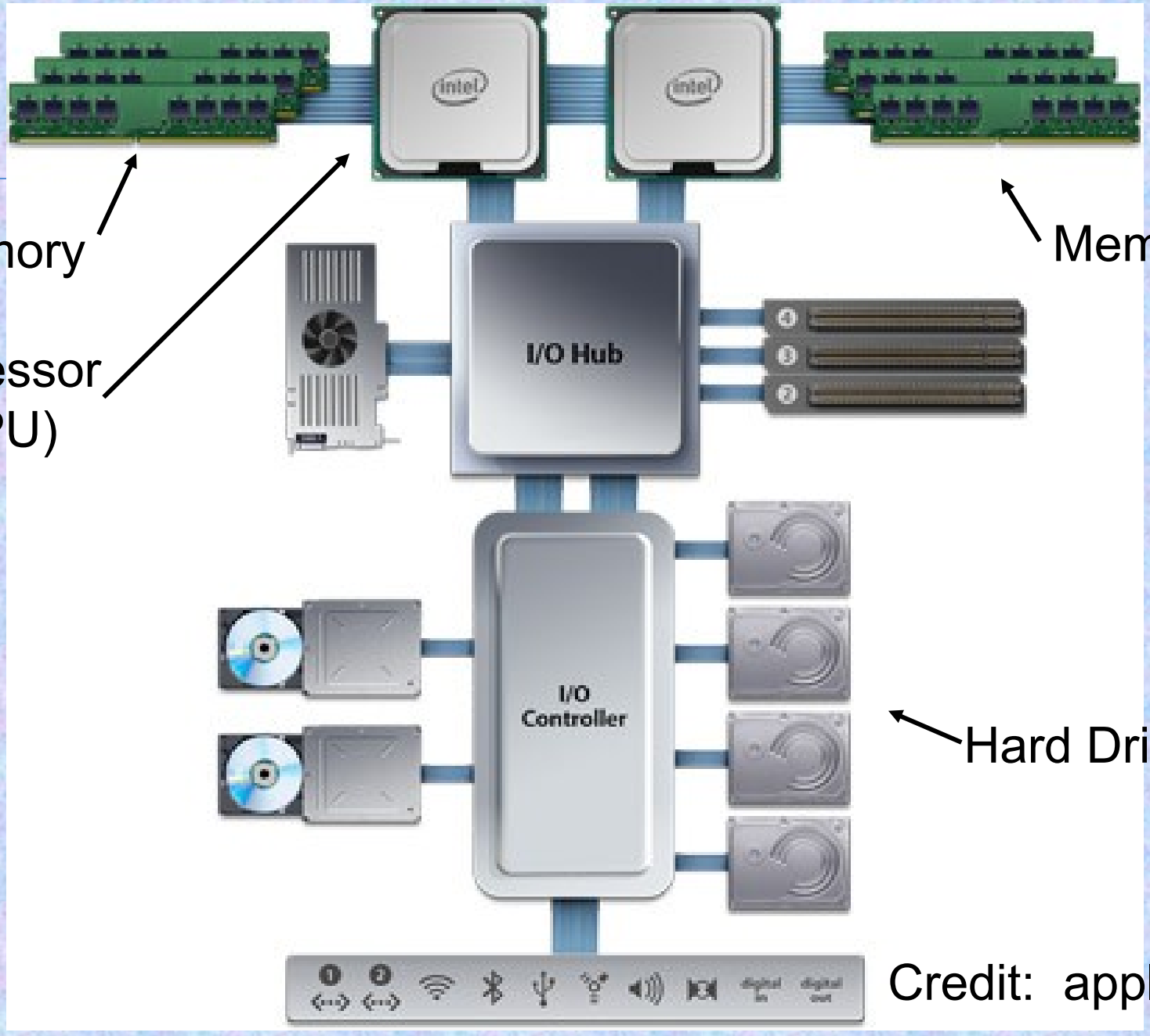
Value Types vs. Reference Types

- Master the pointers in C++/C
- Importance of memory managment


Memory
Processor
(CPU)
Memory
I/O Hub
I/O
Controller
Hard Drives
Credit: apple.com

CPU + Memory

- Note that the CPU
  - the “central *processing* unit” is separate from memory, where data are *stored*.

CPU + Memory

- Any data being directly used by a program at any given moment is placed within the CPU on what is called a *register*.

CPU + Memory

- While it's actually more complex than this…
  - Registers are *like* local variables
    - temporary placeholders for values.
  - Memory is *like* a giant, global array.

CPU + Memory

- For now, we'll settle for this simplification.
  - More details will come in CDA 3101
  - Introduction to Computer Organization.

Memory

- Memory is like a giant set of lockers, where each such locker can hold a set, limited amount of data.
- Each locker has a very precise number assigned to it – in computer terms, it has a unique memory *address*. (Like a mailbox.)

Memory

The address of each “locker” is permanently assigned to it.

- You cannot move (or “give a new address to”) an already existing locker.
- Thus, to change the address of data being stored, you must move it from one “locker” to another.

Memory

- It is possible to use multiple adjacent "lockers" to store a large data structure.
  - Another name for a "data structure" would be an *object*, in C++ terms.
  - In this sense, the idea of an "array" is *also* a "data structure."

Memory + Arrays

- An *array*, when actually utilized during execution, is a large, contiguous (undivided) block of memory.
- The array's starting location – its *address* within memory – is then stored for future *reference*.
  - All of its data can be found given this starting reference and indices.

Memory + Arrays

The “first” (typically, index “0”) element of the array is stored directly at the starting address of the array.

- Each subsequent element is then stored at a constant offset from this address.

Working with Data in C++

- Note: while the information on the next few slides is written in a C++ fashion, the underlying principles apply to most computer languages.
- Remember the learning experience from Java to C++/C. You will need to learn new languages in your CS careers.

Working with Data in C++

- At its core, all data within a program are stored as a binary number.
  - These are the famous 0's and 1's.
  - Each 0 and 1 is known as a *bit*, or *bi*nary dig*it*.
  - Eight of these make a *byte*, 1024 bytes make a *kilo*byte, and so forth.

Working with Data in C++

- This “binary number” may be thought of as a ***value***.
- Data which are *directly* represented on a CPU, within a programming language, by a binary number is considered a ***value type***.

Working with Data in C++

- *(Primitive) value types* within C++:
  - int
  - short
  - long
  - char
  - double
  - float
  - bool

This list is not exhaustive.

Working with Data in C++

- Other values are instead handled *through their memory address*.
- Data which are *referenced* on a CPU, within

  a programming language, through its address is considered a ***reference type***.
  - Note that the *address itself*, while unseen by the programmer, is also a *value*.
    - We call this value… a ***pointer***.

Working with Data in *Java*

- *Reference types* within Java are ***all*** "classes"/"objects," and vice-versa.
  - Only the primitive value types are treated "by value" in Java.
- Note that these objects, or classes, may be composed of multiple value types.
  - These values must be obtained *through* the whole object's **reference**.

Working with Data in *C++*

- In C++, the programmer may choose which way to handle data.
  - Objects and arrays may be handled by value (within a function) *or* through a pointer.
  - While primitive types default to “by value,” they may be handled by reference!