Abstractions and Reality

- Learning Objectives (i.e., after reviewing this presentation, you should be able to:)
  - Describe some ways that the abstraction provided by the C language is similar to and different from the abstraction provided by the underlying machine.
  - Explain the difference between a **program** and a **process**.
  - Visualize how memory is arranged in a process.
  - Write simple programs to help you answer questions about how memory is arranged in a process.

Getting Started

- The code examples used here can be found in the `cs61-videos` repository in the `abstractions` directory.
- You should already have your class appliance set up.
- You should then be able to clone the repository:

```
git clone git://code.seas.Harvard.edu/cs61/cs61-videos
```

Abstractions Everywhere

- Programming and computer science is chock full of abstractions.
- Abstractions are a way to:
  - Manage complexity
  - Make the unfamiliar familiar
  - Separate relevant from irrelevant details
- But, abstractions also have a cost:
  - Sometimes they incur overhead (e.g., speed, memory)
  - Sometimes they hide power

Some abstractions

- A web application framework
- A database
- Collections of objects
- The C language
- Assembly language
- A processor architecture

The Abstractions We'll Examine

- The C language & assembly language
  - Programming languages provide an abstraction that lets humans express the meaning of a program.
  - The language definition of C is higher level than that of assembly language, but both are still designed for humans.
  - Compilers transform C into assembly language.
  - Assemblers then transform assembly language into machine code, targeting a specific …
- A processor architecture
  - A machine implements some processor architecture
  - There can be multiple implementations of an architecture

Why bother?

- "I'm perfectly happy with my abstractions, why bother looking under the covers?"
- Understanding the real machine helps us understand why some programs are fast/slow.
- It helps us understand how things go wrong.
- The real machine is more powerful
  - with power comes responsibility – it is also in some ways more "dangerous"

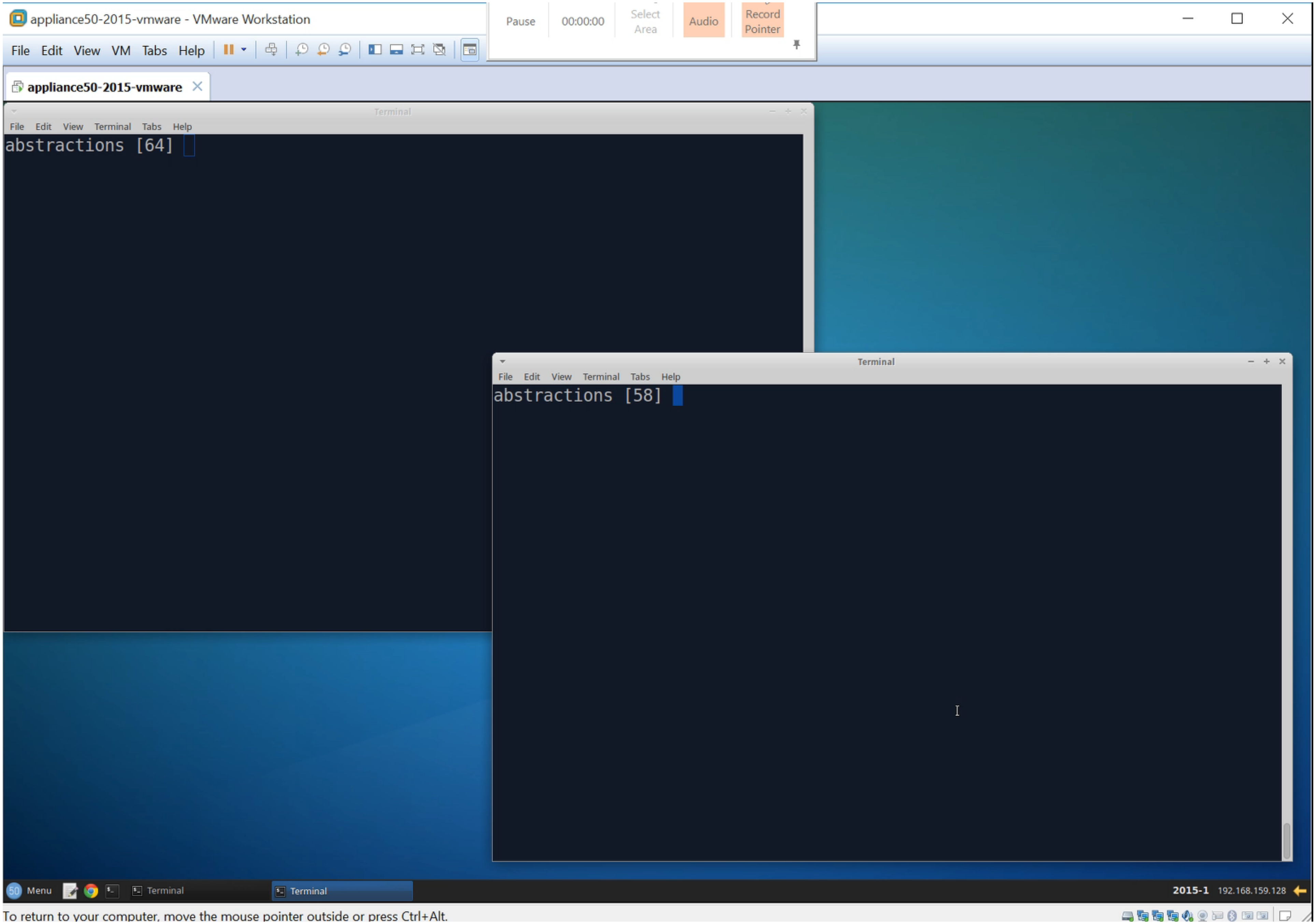
```
abstractions [51] more hello.c
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Hello World!\n");
}
abstractions [52]
```

Language and Machine Abstractions

1. I want to print “Hello World!” to the screen.
2. I write a C program.
3. The compiler translated the C into assembly
4. An assembler translated the assembly into machine code
5. A linker combined the machine code with library information to create an executable file.
6. The OS created a process in which to execute that file.
7. “Hello World!” appeared.

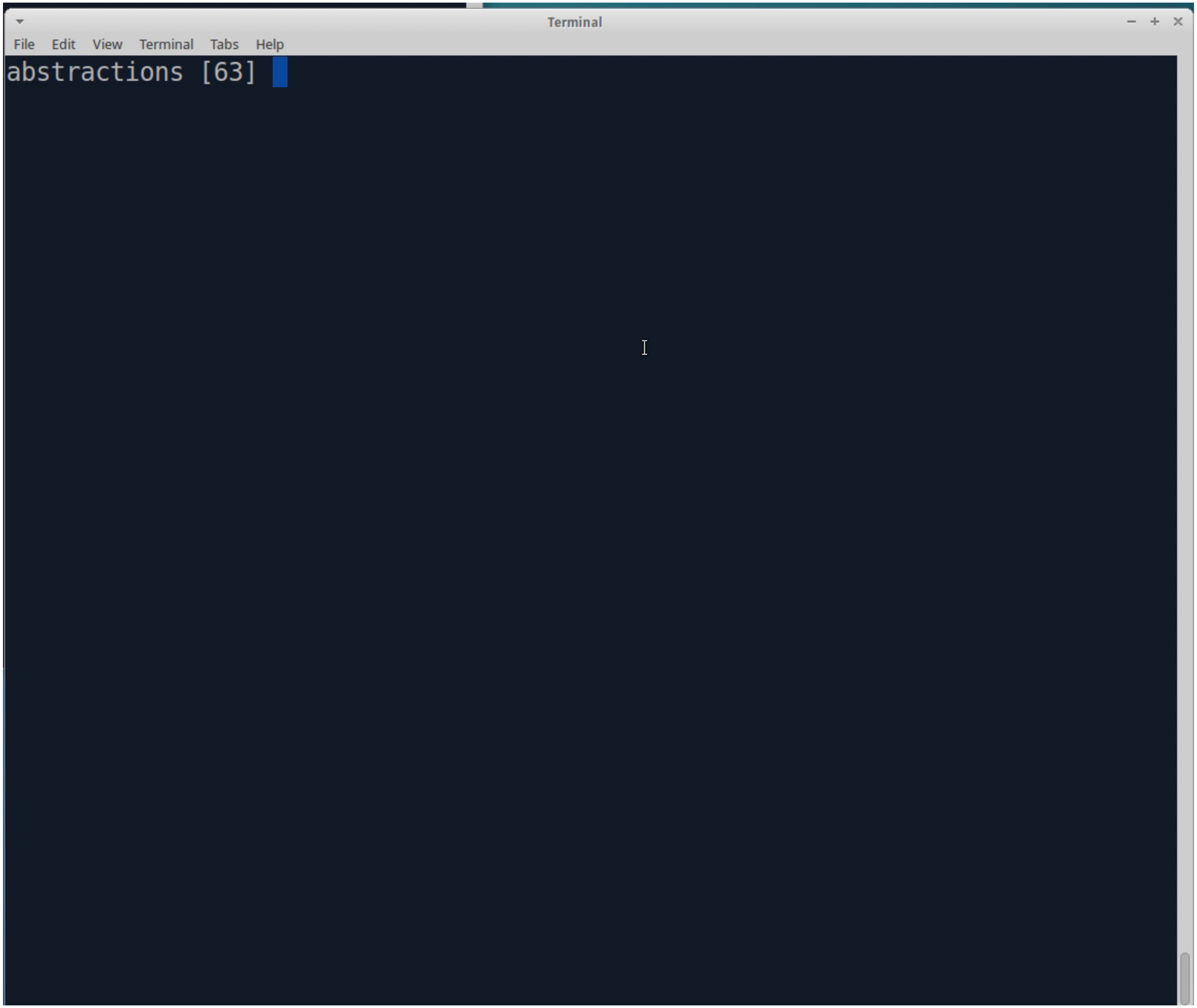
From Program to Process

- A process is the realization of a program executing on a machine.
- It is an abstraction, provided by the operating system.
  - Provides isolation (you and I can both run things and they don't interfere with each other).
  - Makes it look like nothing else is running except the process.
  - Makes it appear as if the process runs from start to end without interruption.
- But this is all an illusion!
  - Many processes might be running.
  - A process can be interrupted.

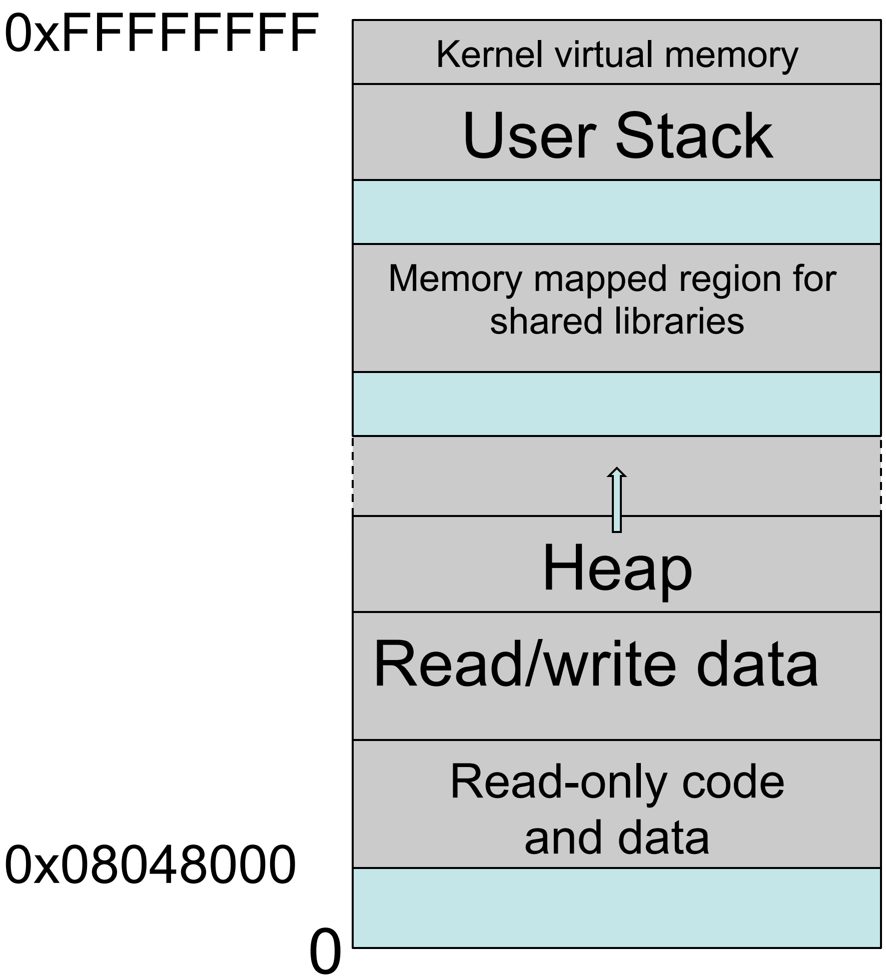

appliance50-2015-vmware - VMware Workstation
Pause
00:00:00
Select Area
Audio
Record Pointer
File Edit View VM Tabs Help
appliance50-2015-vmware
Terminal
File Edit View Terminal Tabs Help
abstractions [64]
Terminal
File Edit View Terminal Tabs Help
abstractions [58]
Menu
Terminal
Terminal
2015-1 192.168.159.128
To return to your computer, move the mouse pointer outside or press Ctrl+Alt.

But what is a process?

- A process is composed of two parts:
  - A part that keeps track of “stuff”: Address space
  - A dynamic part: Thread
- Address space:
  - A “place” in which execution happens.
  - The set of addresses (e.g., memory locations) to which a running computation has access.
  - An address space can be physical (addresses map directly to locations in the hardware) or virtual (addresses are “make believe” but get translated into locations in hardware).
  - Address spaces provide protection boundaries.


Terminal
File Edit View Terminal Tabs Help
abstractions [63]

The Address Space



Local: 0xbf------
Global: 0x0804a024
Const Global: 0x08048870
Heap: 0x08------ (> Global)
Main: 0x080484a0
Printf: 0xb7e674a0

Summing in up

- The C language presents an abstract machine that lets a human express a computation.
- Tools (system programs) transform that expression into an executable that the operating system knows how to execute.
- The operating system creates a process to execute that program.
- The process lives in the memory of a real machine.
- The real machine reads instructions and data from that memory and executes the instructions.