Interlude

Object Oriented Design

Announcements for This Lecture

This Week

- Today is an **Interlude**
  - **Nothing today is on exam**
  - Another "Big Picture" talk
  - Relevant to Assignment 6
- Review for exam posted
- **New Review Session**
  - Saturday evening 5pm!
  - Here in Phillips 101
  - Slides posted tomorrow

Assignments

- Assignment 5 almost done
  - Should be graded by tonight
  - Grades looking okay so far
- Keep on Assignment 6
  - Helps with arrays (on exam)
  - Due next Thursday
- **Extra credit:**
  - It will be worth 5 points
  - Can make more than 100

The Challenge of Making Software

```
/** Simulate vignetting (corner darkening)
 *  characteristic of antique lenses. Darken
 *  each pixel in the image by the factor
 *            (d / hfD)^2
 *  where d is the distance from the pixel
 *  to the center of the image and hfD (for
 *  half diagonal) is the distance from the
 *  center of the image to the corners.
 * The alpha component is not changed.
 */
public void vignette() {
    int rows= currentIm.getRows();
    // FINISH ME
}
```

- We do a lot for you
  - Classes made ahead of time
  - Detailed specifications
  - You just "fill in blanks"
- The "Real World"
  - Vague specifications
  - Unknown # of classes
  - Everything from scratch
- Where do you start?

Software Patterns

- **Pattern**: reusable solution to a common problem
  - Template, not a single program
  - Tells you how to design your code
  - Made by someone who ran into problem first
- In many cases, a pattern gives you the interface
  - List of headers for the public methods
  - Specification for these public methods
  - Only thing missing is the implementation

Just like this course!

Example Pattern: I/O Streams

Challenge: want to get input from somewhere

- Are these cases different?
- Or do they have a pattern?

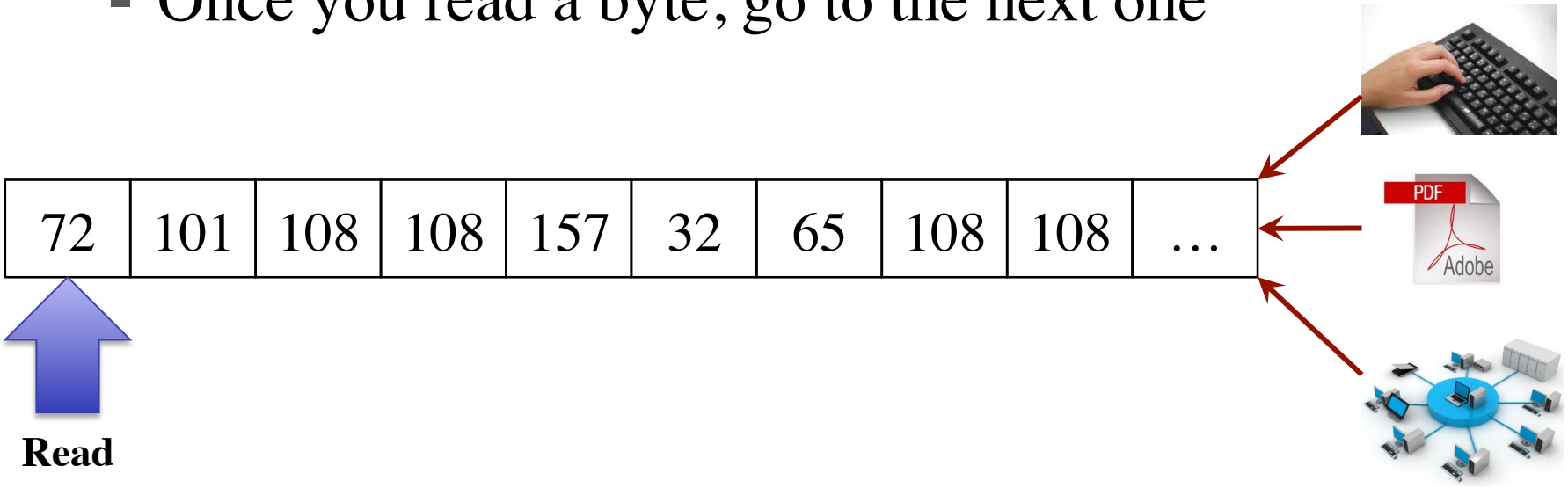
- From a file:

- From the keyboard:

- From the nework

Example Pattern: I/O Streams

- **InputStream**: Read-only list of bytes (0..255)
  - Like an array, but can only read once
  - Once you read a byte, go to the next one



- **OutputStream**: Like InputStream, but write-only

Example Pattern: I/O Streams

```
public class InputStream {
  /** Yields: next byte (0..255)
    *   in stream or -1 if empty  */
  public int read() throws IOE{
     ...
  }
 /** Shuts the input stream
   *   down (close file, disconnect
   *   network, etc.)  */
  public void close() throws IOE{
     ...
  }
}
```

```
public class OutputStream {
  /** Writes a byte to the stream
    *   Pre: b is in range 0..255 */
  public int write() throws IOE{
     ...
  }
  /** Shuts the input stream
   *   down (close file, disconnect
   *   network, etc.)  */
  public void close() throws IOE{
     ...
  }
}
```

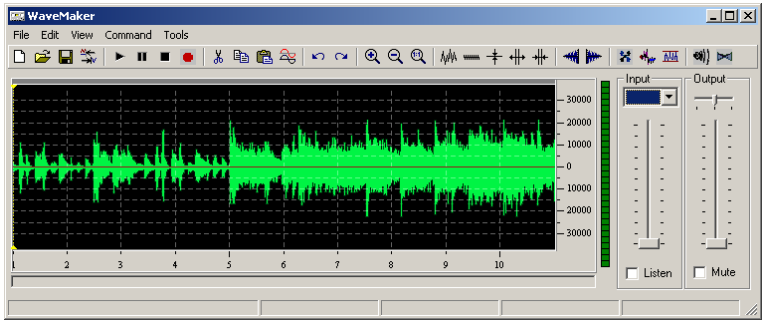
Example Pattern: I/O Streams

Challenge: want I/O stream for data other than bytes

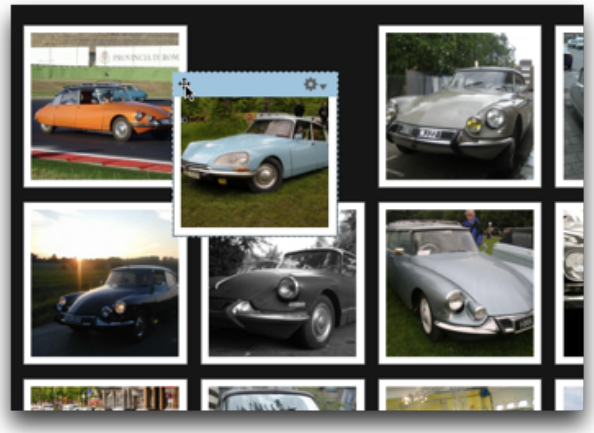
- Text:

ABCDEFGHIJKLMN
OPQRSTUVWXYZÀ
abcdefghijklmnopqr
stuvwxyzàåéîõøü&
1234567890($£€.,!?)

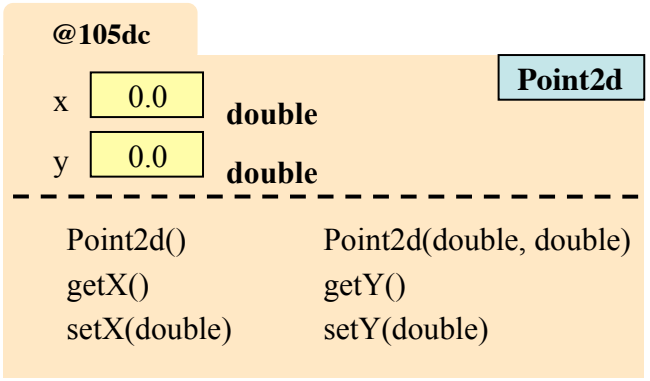
- Sound:



- Images



- General Objects



How Many Classes Do We Need?

- Source:
  - Keyboard
  - File
  - Network
- Data Type:
  - Text
  - Images
  - Sound
  - Objects

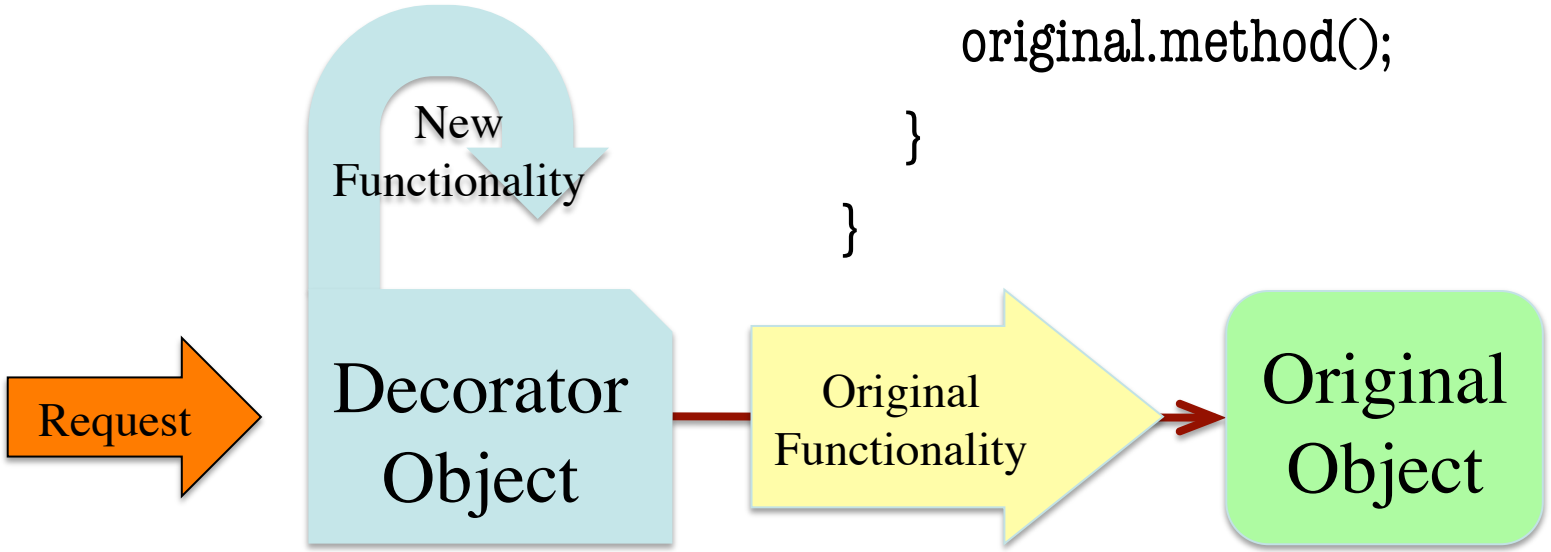
3x4 = **12 Classes**!

Need 3 more every time we add a new data type

Must be a better way!

Example Pattern: Decorators

```
public class Decorator {
    private Object original;
    public void method() {
        doSomethingNew();
        original.method();
    }
}
```



Decorators and Java I/O

- Java I/O works this way.
  - Start with basic Input/OutputStream
  - Determined by source (keyboard, file, etc.)
  - Add decorator for type (text, images, etc.)
- You did this in the lab on File I/O

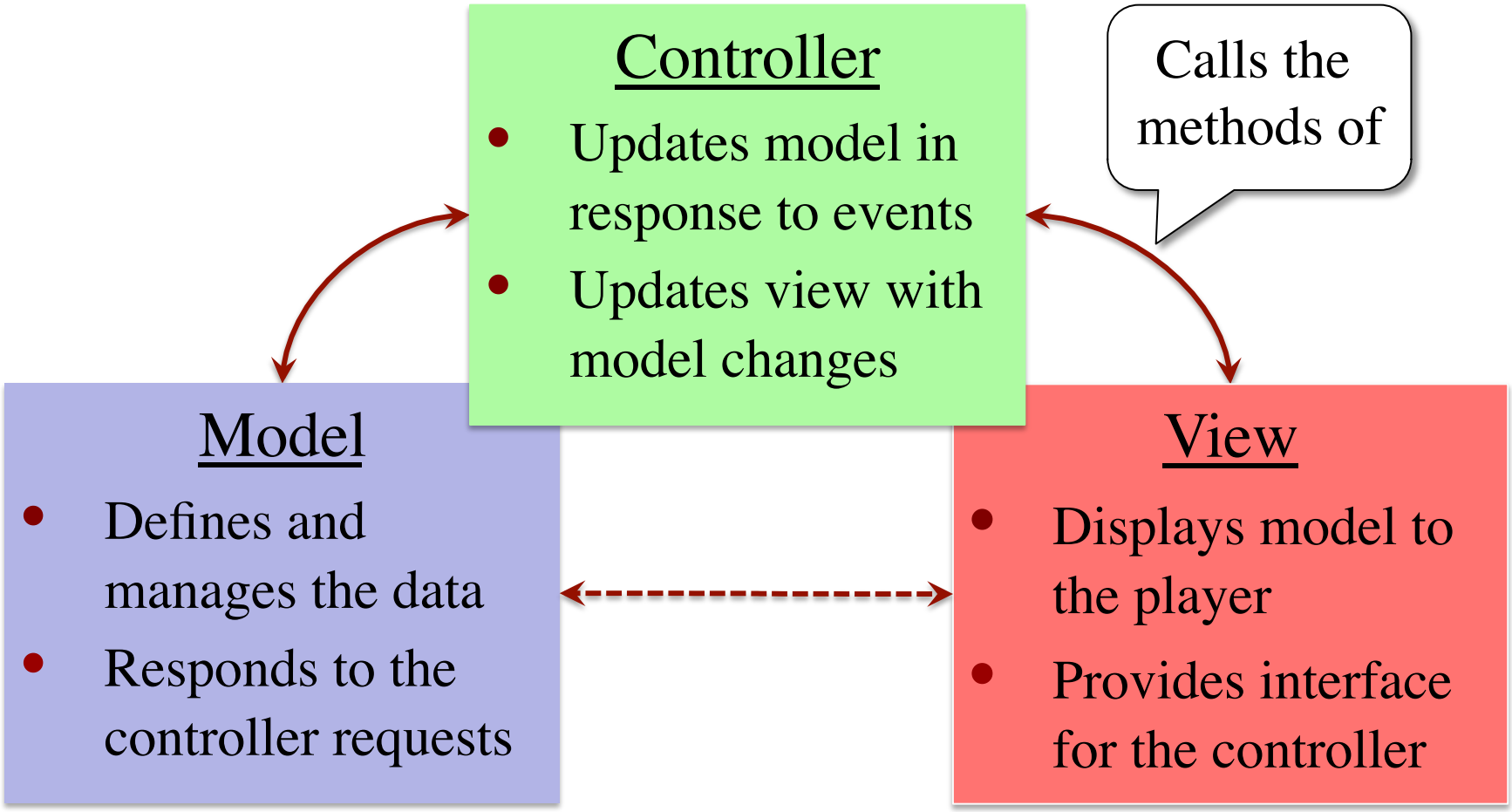
```
FileInputStream input = new FileInputStream("myfile.txt");
BufferedReader reader = new BufferedReader(input);


// Read a line of text
String line = reader.readLine()
```

Architecture Patterns

- Essentially same idea as **software pattern**
  - Template showing how to organize code
  - But does not contain any code itself
- Only difference is **scope**
  - **Software pattern**: simple functionality
  - **Architecture pattern**: complete application
- Large part of the job of a **software architect**
  - Know the best patterns to use in each case
  - Use these patterns to distribute work to your team

Model-View-Controller Pattern



TemperatureConverter Example

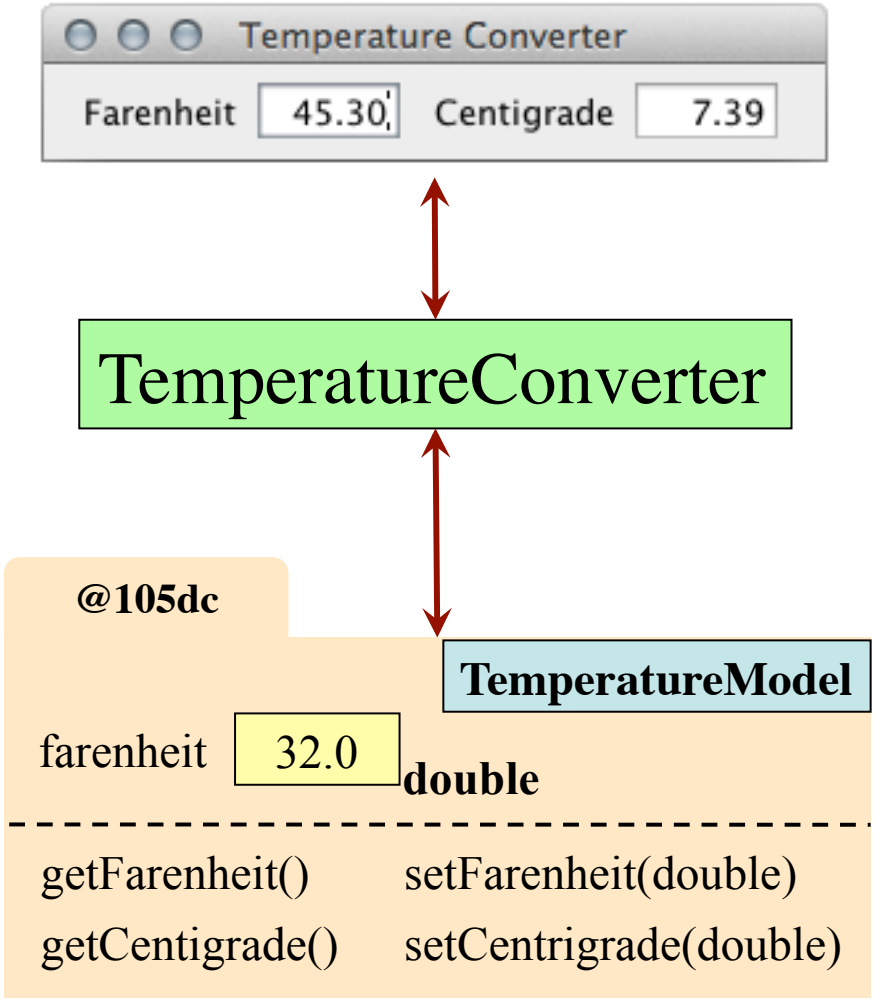
- Model: (TemperatureModel.java)
  - Stores one value: fahrenheit
  - But the methods present two values
- View: (TemperatureView.java)
  - Constructor creates GUI components
  - Recieves user input but does not “do anything”
- Controller: (TemperatureConverter.java)
  - **Main class**: instantiates all of the objects
  - “Communicates” between model and view

TemperatureConverter Example

View

Controller

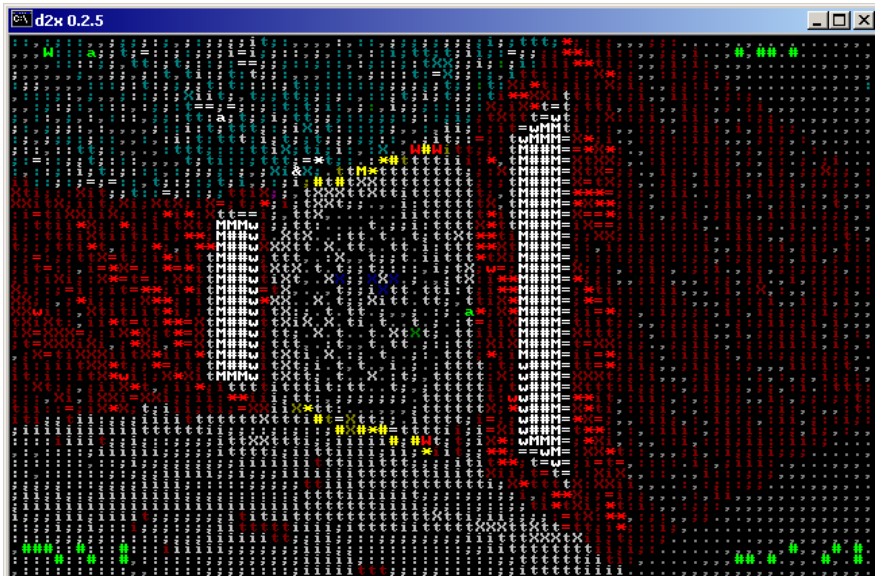
Model



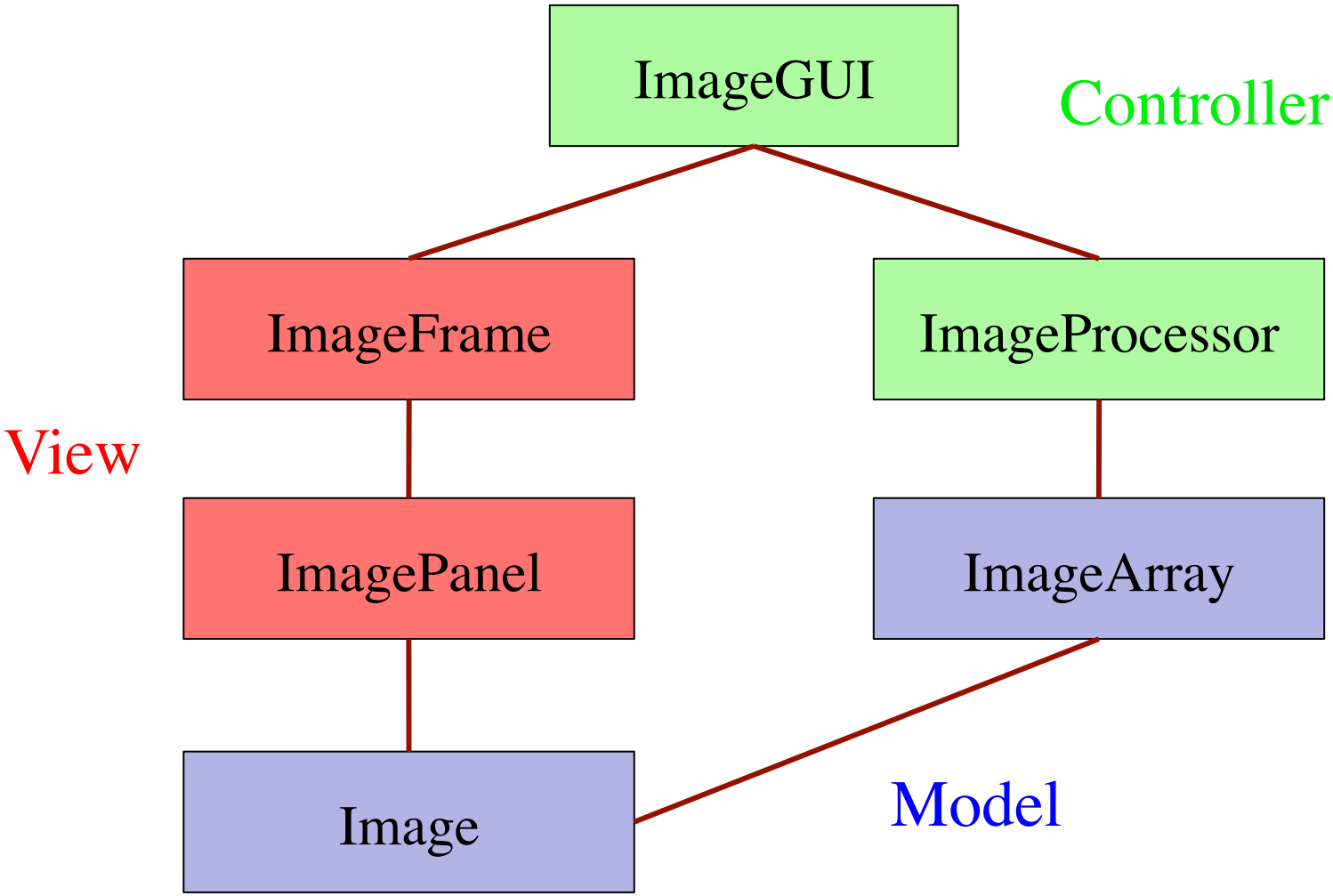
Advantages of This Approach

View

Another View

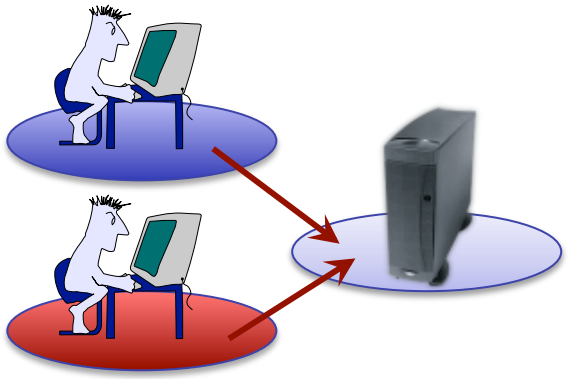


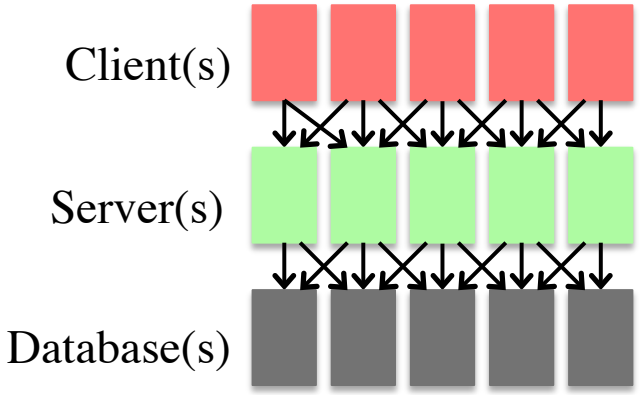
MVC and Assignment 6



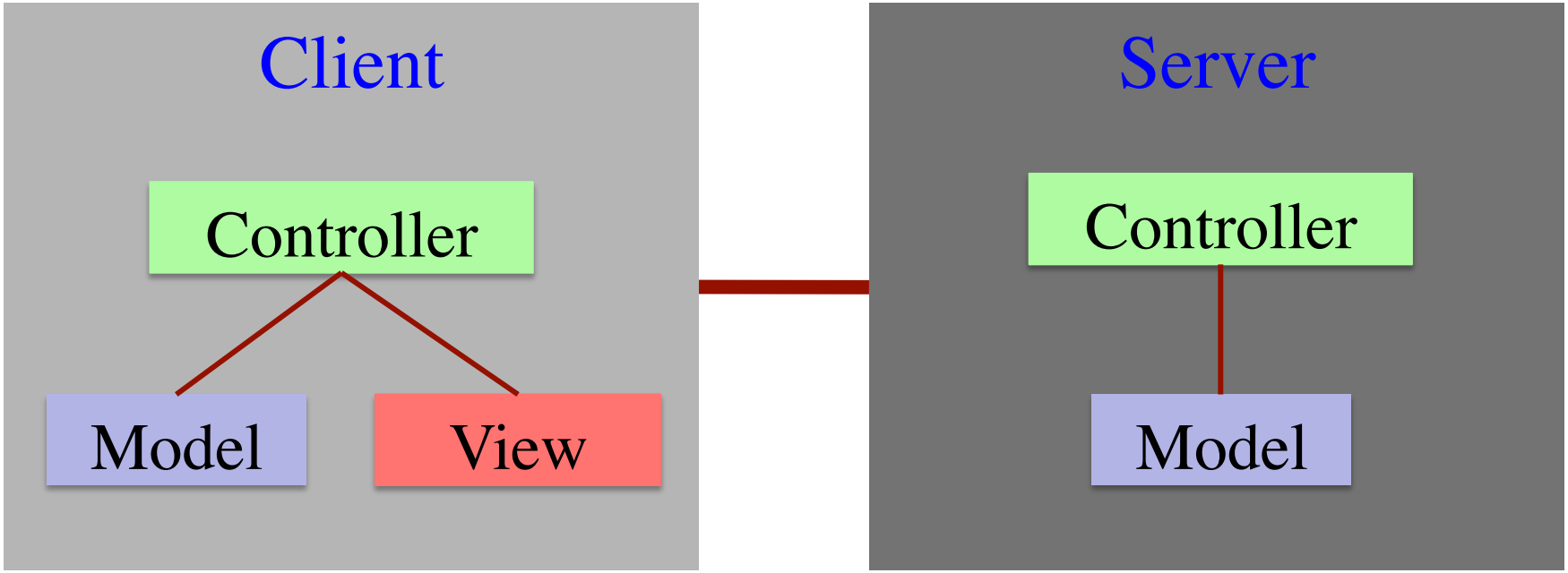
Beyond Model-View-Controller

- MVC is best pattern for offline programs
  - Networked get more complex
- Client-Server
  - Client runs on your computer
  - Client connects to remoter server
- Three-Tier Applications
  - Client-Server-Database
  - Standard for web applications
- … and many others





You Can Even Mix and Match



Software Patterns and Computer Science

- Patterns are part of **Software Engineering**
  - At Cornell that is part of the CS department
  - But also part of information science
- Very important in the "Systems" courses
  - Courses focused on building big applications
  - Examples: databases, operating systems, etc…
  - Interested in systems? Take 2110, then 3410
- Also a big part of the game design courses
  - Course is being renumbered CS 3152