Lecture 19

Object Oriented Design

Announcements for Today

Reading

- Today: See reading online
- Thursday: Chapter 7

- **Prelim, Nov 6th 7:30-9:30**
  - Review posted tonight
  - Review session Sunday
  - Recursion + Loops + Classes
- **Last day for conflicts!!!**
  - Submit conflict on CMS
  - Extra time: please submit too

Assignments

- A4 **still** being graded
  - Hope to be done by Thurs
  - Also looking at surveys
- A5 due tomorrow
  - Remember to upgrade your CornellExtensions
  - Extra consultants today
- A6 posted Thursday
  - Over two full weeks
  - Week and ½ after exam

Computer Game Development

Credits: Planetfall (1983)

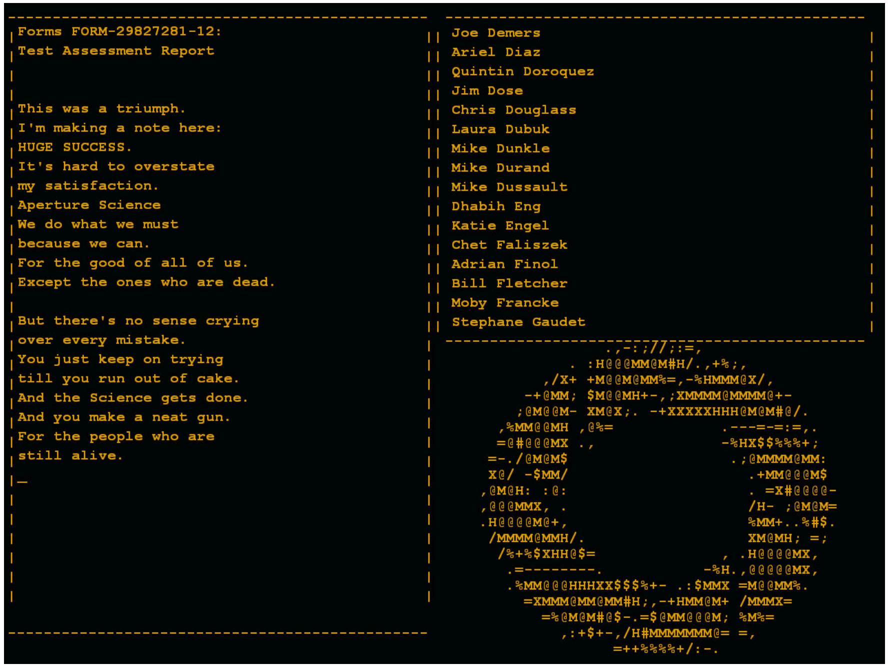
Steve Meretzky

Computer Game Development

Credits: Planetfall (1983)

Steve Meretzky

Credits: Portal (2007)

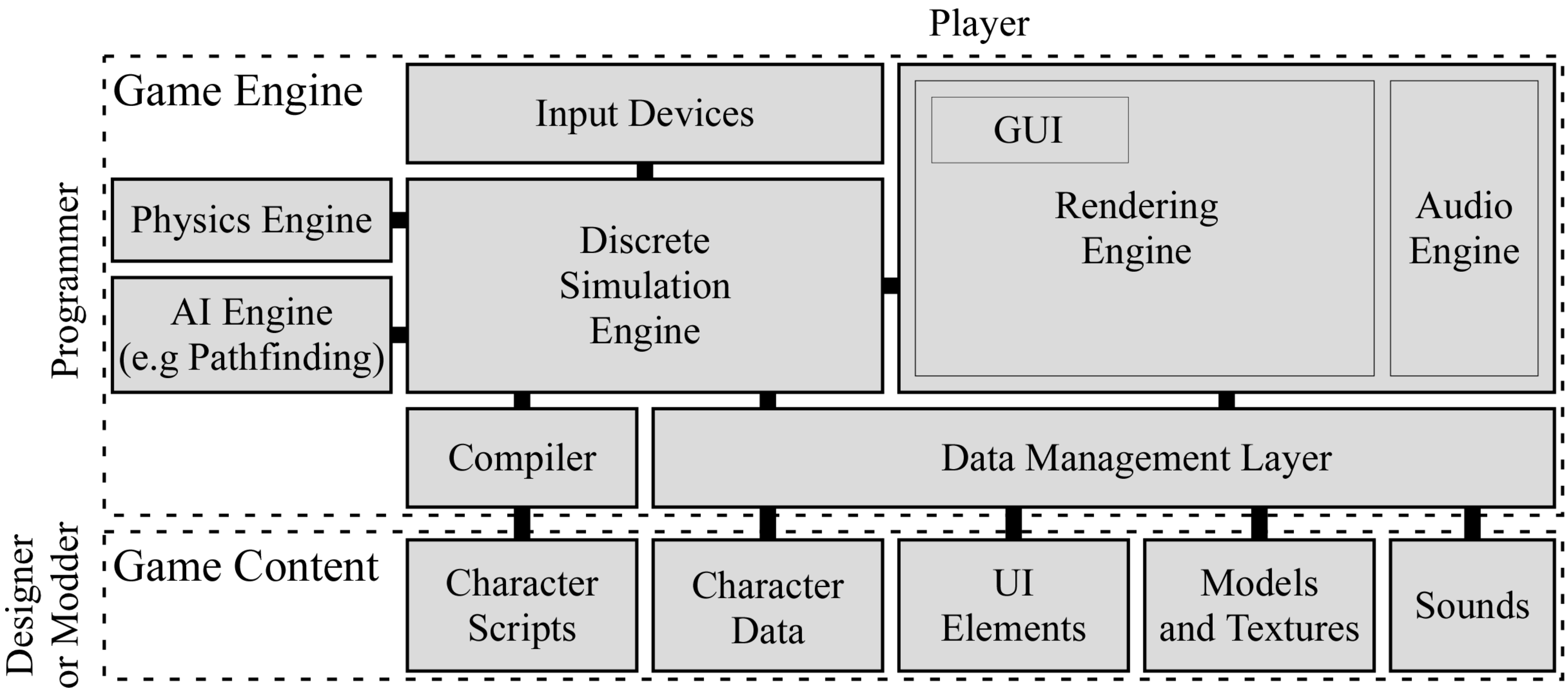


```
Forms FORM-29827281-12:
Test Assessment Report


This was a triumph.
I'm making a note here:
HUGE SUCCESS.
It's hard to overstate
my satisfaction.
Aperture Science
We do what we must
because we can.
For the good of all of us.
Except the ones who are dead.

But there's no sense crying
over every mistake.
You just keep on trying
till you run out of cake.
And the Science gets done.
And you make a neat gun.
For the people who are
still alive.
_
```

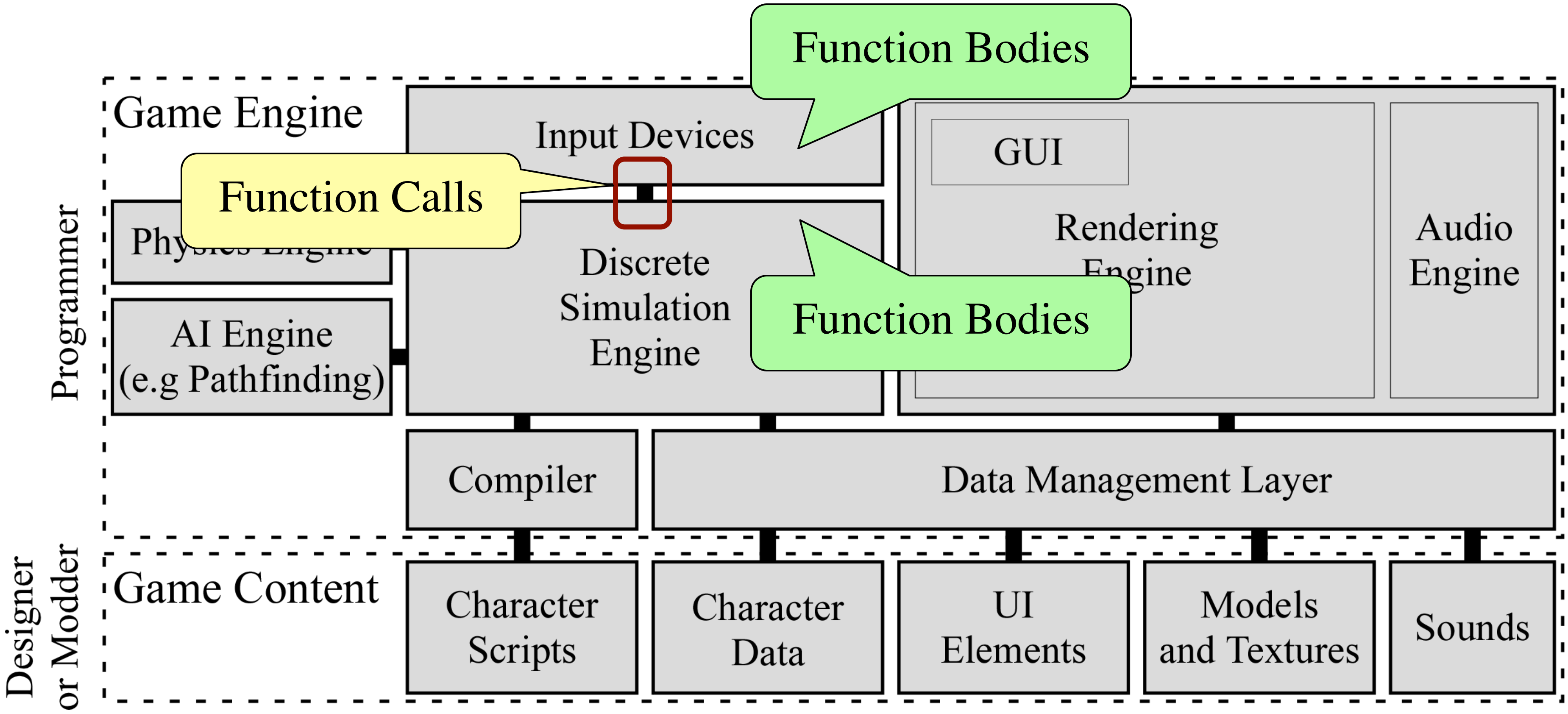
```
Joe Demers
Ariel Diaz
Quintin Doroquez
Jim Dose
Chris Douglass
Laura Dubuk
Mike Dunkle
Mike Durand
Mike Dussault
Dhabih Eng
Katie Engel
Chet Faliszek
Adrian Finol
Bill Fletcher
Moby Francke
Stephane Gaudet
```

```
              .,-:;//;:=,
          . :H@@@MM@M#H/.,+%;,
       ,/X+ +M@@M@MM%=,-%HMMM@X/,
     -+@MM; $M@@MH+-,;XMMMM@MMMM@+-
    ;@M@@M- XM@X;. -+XXXXXHHH@M@M#@/.
  ,%MM@@MH ,@%=            .---=-=:=,.
  =@#@@@MX .,              -%HX$$%%%+;
 =-./@M@M$                  .;@MMMM@MM:
 X@/ -$MM/                    .+MM@@@M$
,@M@H: :@:                    . =X#@@@@-
,@@@MMX, .                    /H- ;@M@M=
.H@@@@M@+,                    %MM+..%#$.
 /MMMM@MMH/.                  XM@MH; =;
  /%+%$XHH@$=              , .H@@@@MX,
   .=--------.           -%H.,@@@@@MX,
   .%MM@@@HHHXX$$$%+- .:$MMX =M@@MM%.
     =XMMM@MM@MM#H;,-+HMM@M+ /MMMX=
       =%@M@M#@$-.=$@MM@@@M; %M%=
         ,:+$+-,/H#MMMMMMM@= =,
               =++%%%%+/:-.
```

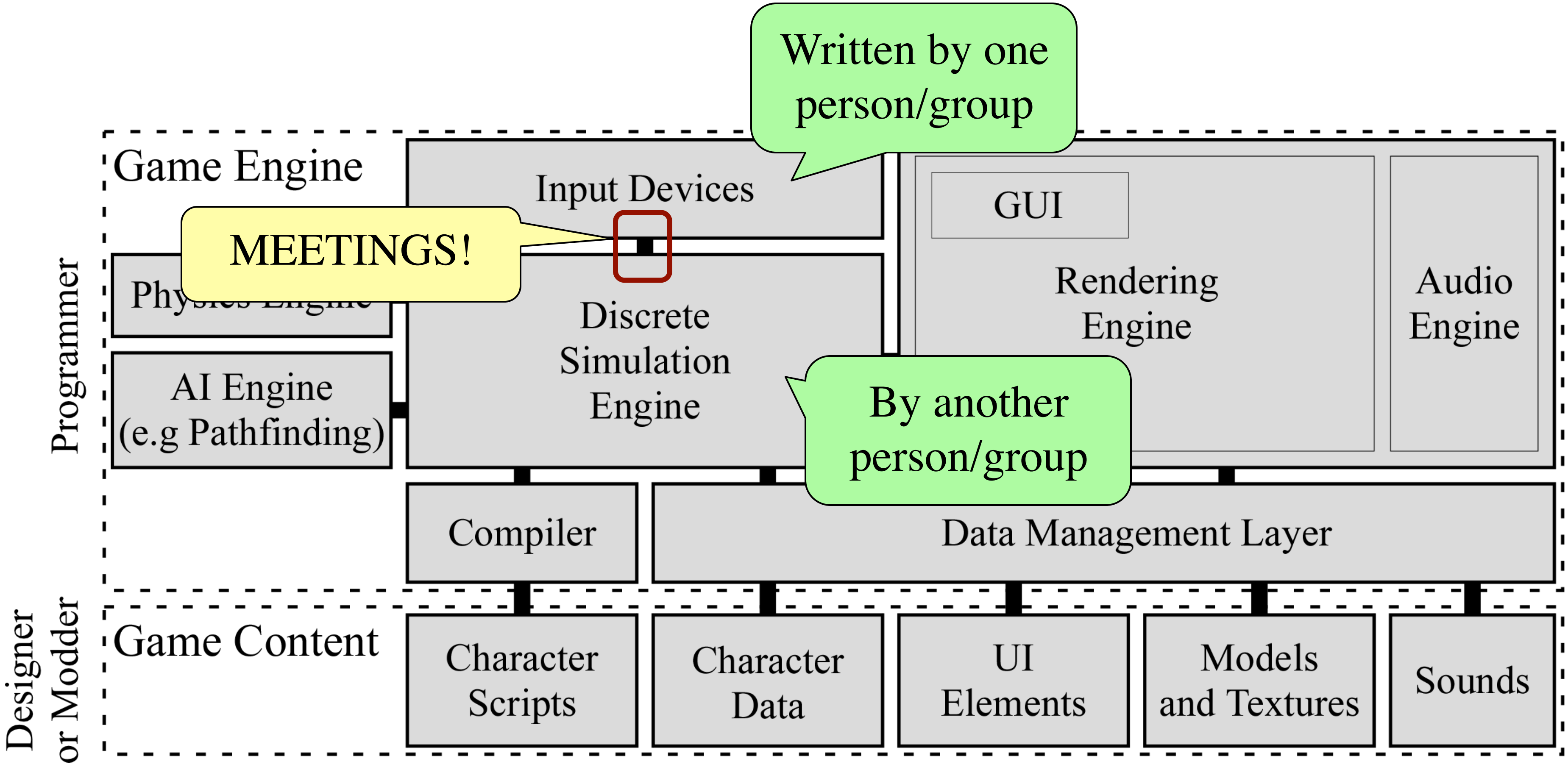
Challenge: Breaking Up Software



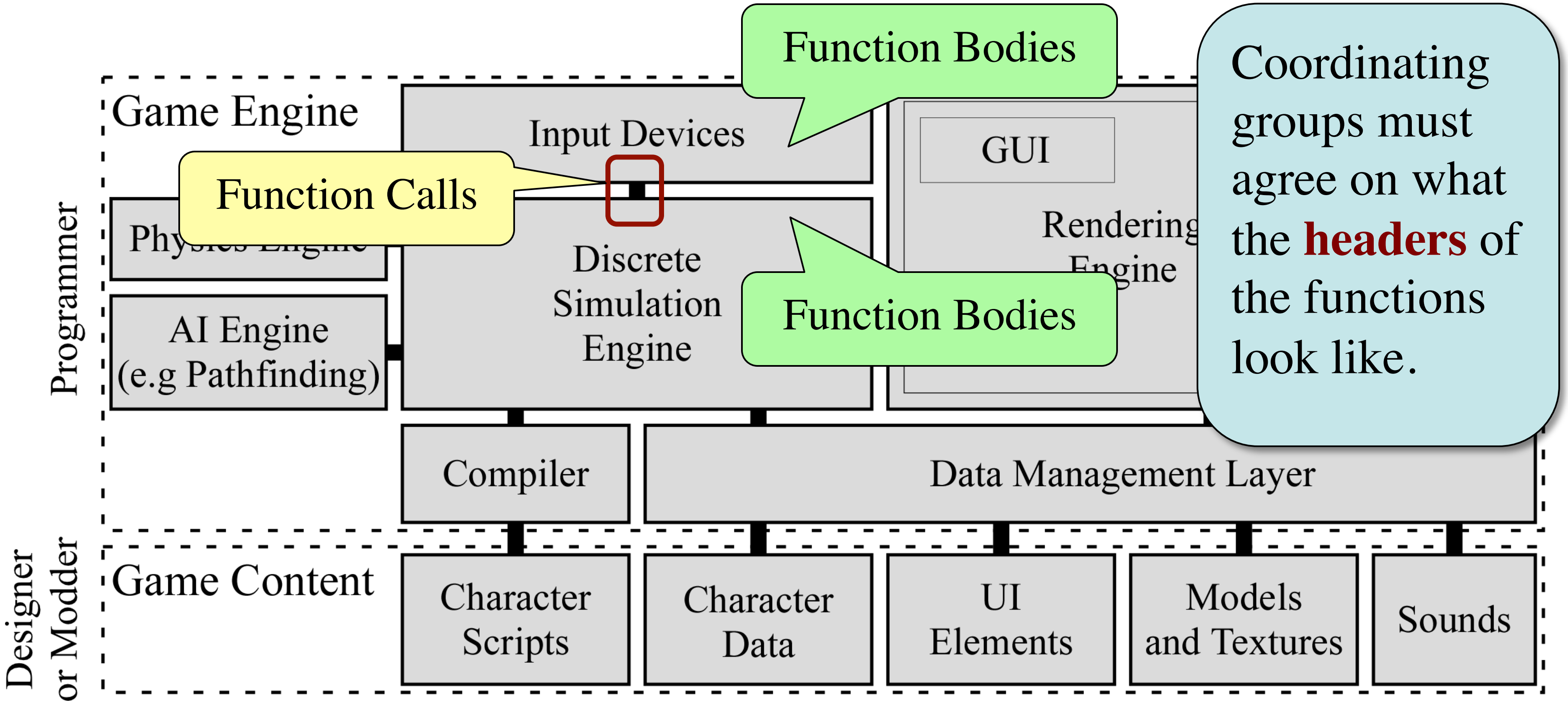
Challenge: Breaking Up Software



Challenge: Breaking Up Software



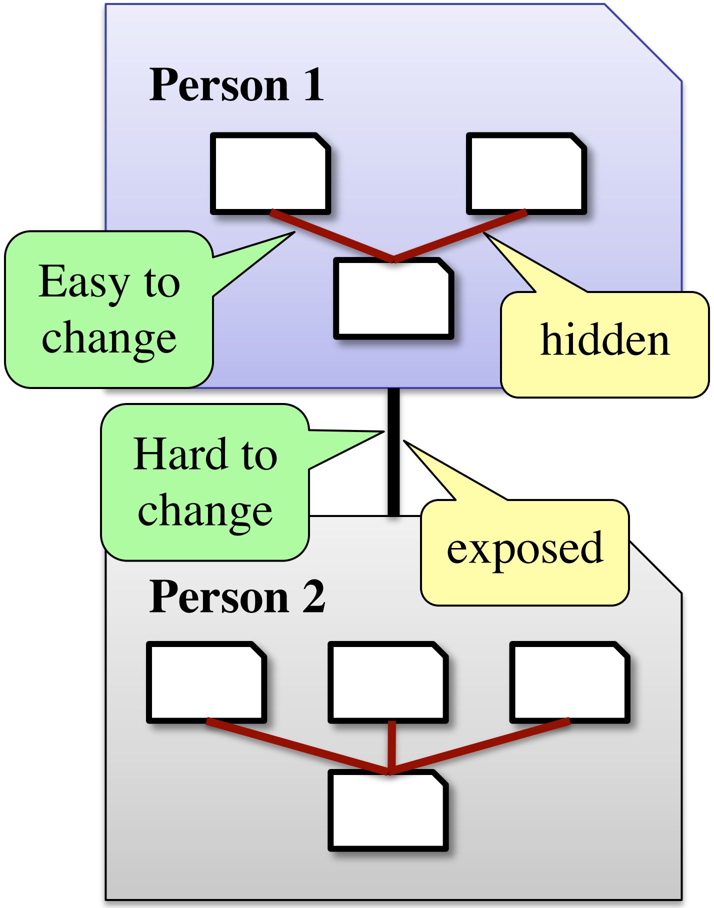
Challenge: Breaking Up Software



Coordinating groups must agree on what the **headers** of the functions look like.

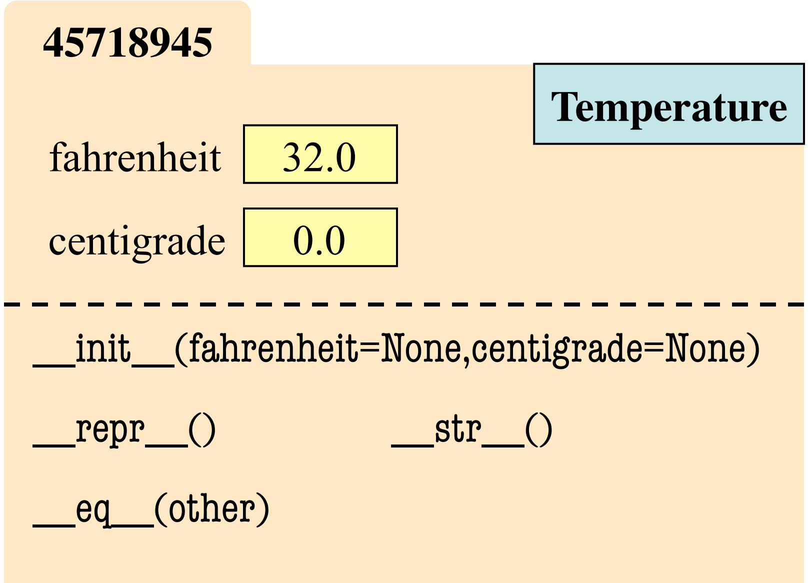
Encapsulation: Reducing Dependencies

- Development is iterative
  - You are always making changes (to improve your software)
- Coordination hurts iteration
  - Others are calling your functions
  - If you change how functions work, their code may no longer work
  - **Example**: Our test code in A1
- **Encapsulation**: limit what the other programmers can access in your code
  - If cannot access, changes are okay



Encapsulation is the Primary Purpose of Object Oriented Programming

- Applies to both code and data!
  - Turtles have a lot of data that you never, ever saw
  - Did you need to see it
  - Would it have been a good idea if you could have seen it?
- Encapsulation in Python
  - Make all data hidden
  - Force data access through the properties (getters/setters)
  - Or through the methods (see Assignment 6)



Invariants

fahrenheit=9*centigrade/5.0+32

centigrade=5*(farenheit-32)/9.0

Encapsulation is the Primary Purpose of Object Oriented Programming

- Applies to both code and data!
  - Turtles have a lot of data that you never, ever saw
  - Did you need to see it
  - Would it have been a good idea if you could have seen it?
- Encapsulation in Python
  - Make all data hidden
  - Force data access through the properties (getters/setters)
  - Or through the methods (see Assignment 6)

```
class Temperature(object):
    _fahrenheit = 32.0
    _centigrade = 0.0

    @property
    def fahrenheit(self):
        """Temp value in fahrenheit"""
        return self._farenheit

    @fahrenheit.setter
    def fahrenheit(self,value):
         self._fahrenheit = float(value)
        # Enforce the invariant
        self._centigrade =5*(value-32)/9.0
```

Encapsulation is the Primary Purpose of Object Oriented Programming

- Applies to both code and data!
    - Turtles have a lot of data that you never, ever saw
    - Did you need to see it
    - Would it have been a good idea if you could have seen it?
- Encapsulation in Python
    - Make all data hidden
    - Force data access through the properties (getters/setters)
    - Or through the methods (see Assignment 6)

```
class Temperature(object):
    _fahrenheit = 32.0
    # _centigrade = 0.0  NOT NEEDED!

    @property
    def centigrade(self):
        """Temp value in centigrade"""
        return 5*(self._fahrenheit-32)/9.0

    @centigrade.setter
    def centigrade(self,value):
        # Change fahrenheit instead
        self. _fahrenheit=9*value/5.0+32
```

Interface vs. Implementation

Interface

- Unhidden methods/properties
- Specifications of the above

```
@property
def centigrade(self):
    """Temp value in centigrade"""
    return 5*(self._fahrenheit-32)/9.0
```

Difficult to change!

Implementation

- Hidden fields and methods
- Bodies of methods/properties

```
@property
def centigrade(self):
    """Temp value in centigrade"""
    return 5*(self._fahrenheit-32)/9.0
```

Easy to change

The Challenge of Making Software

```
def vignette(self):
    """Simulate antique lenses.

    Antique lenses had vignetting or corner
    darkening. This method darkens 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.""”
    rows = self.current.rows
    cols = self.current.cols
    # FINISH ME
```

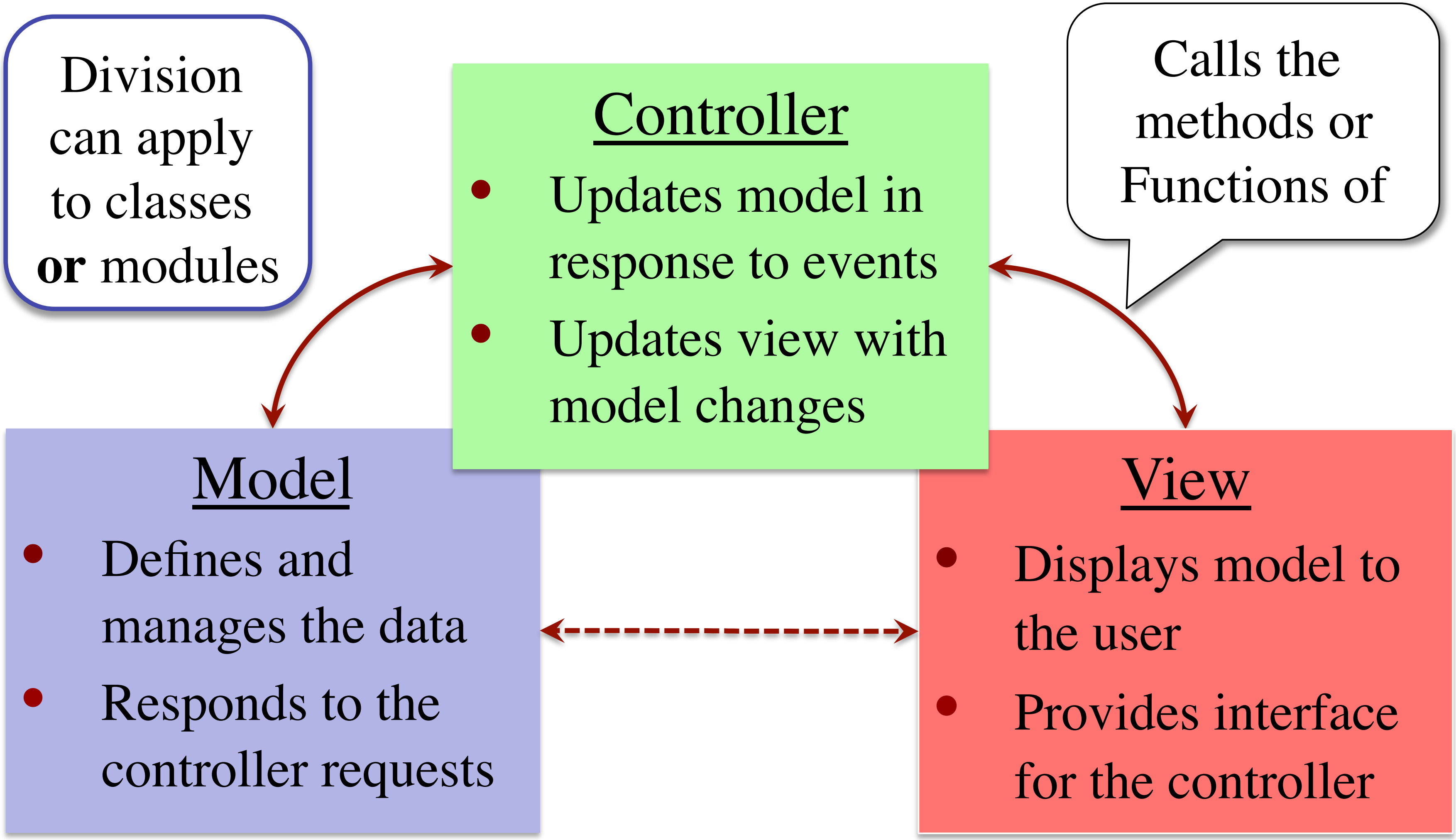
- We do a lot for you
  - Classes made ahead of time
  - Detailed specifications
  - You just “fill in blanks”
- The “Real World”
  - Vague specifications
  - Unknown # of classes
  - Everything from scratch
- Where do you start?

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 non-hidden methods
  - Specification for non-hidden methods
  - Only thing missing is the implementation

Just like
this course!

Model-View-Controller Pattern



Division can apply to classes **or** modules

Calls the methods or Functions of

Controller

- Updates model in response to events
- Updates view with model changes

Model

- Defines and manages the data
- Responds to the controller requests

View

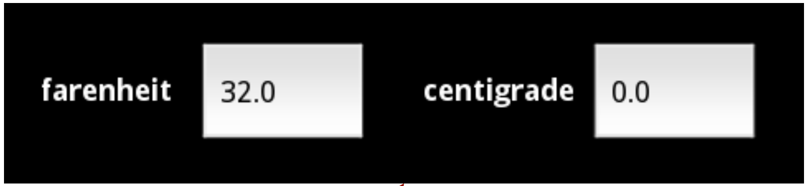
- Displays model to the user
- Provides interface for the controller

TemperatureConverter Example

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

TemperatureConverter Example

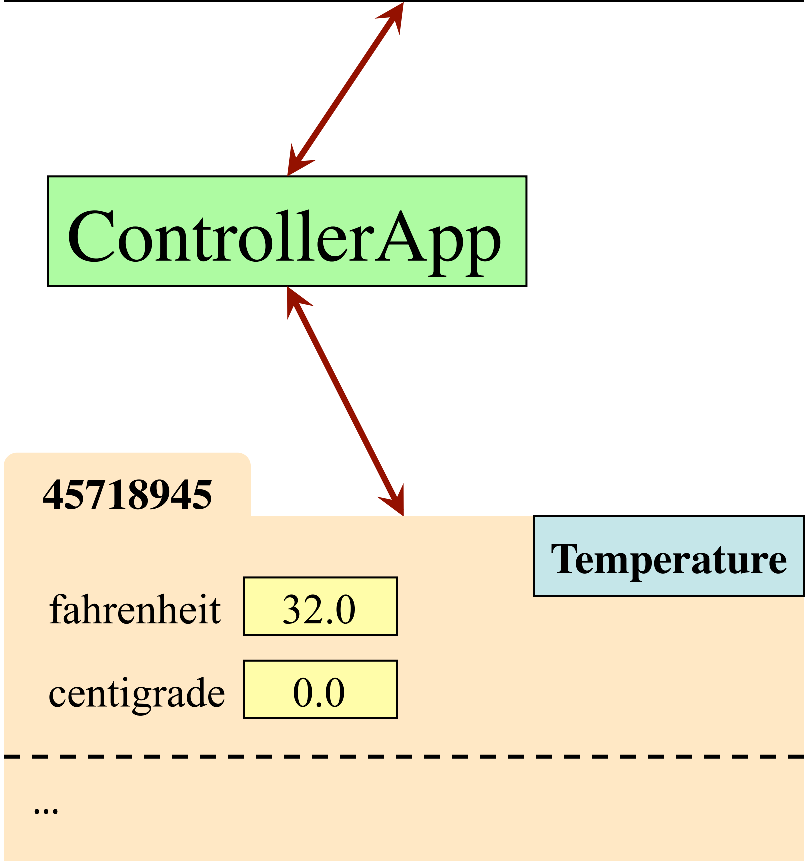
View



farenheit 32.0 centigrade 0.0

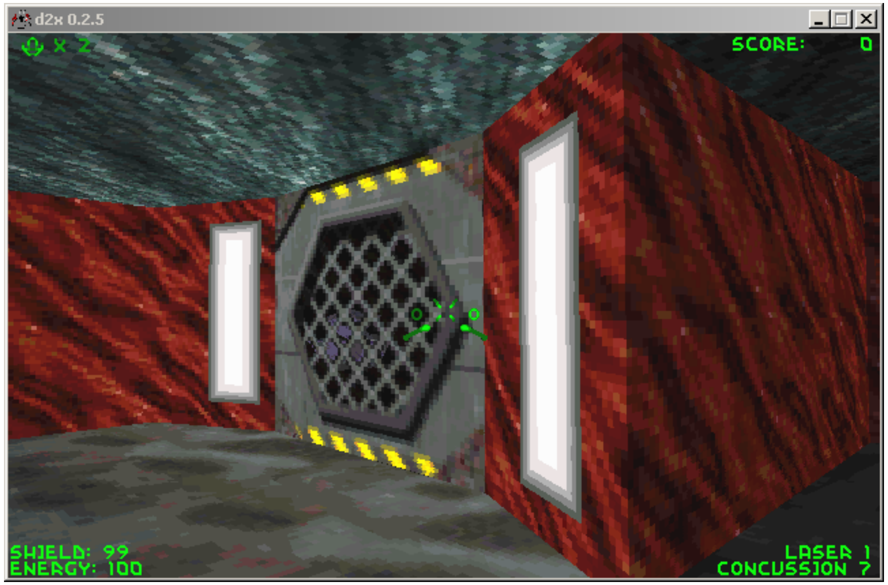
Controller

Model

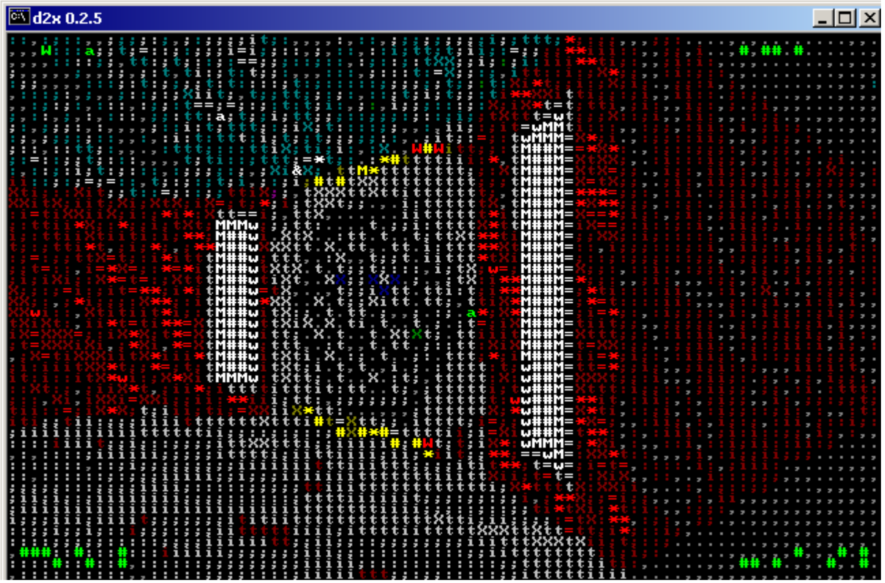


Advantages of This Approach

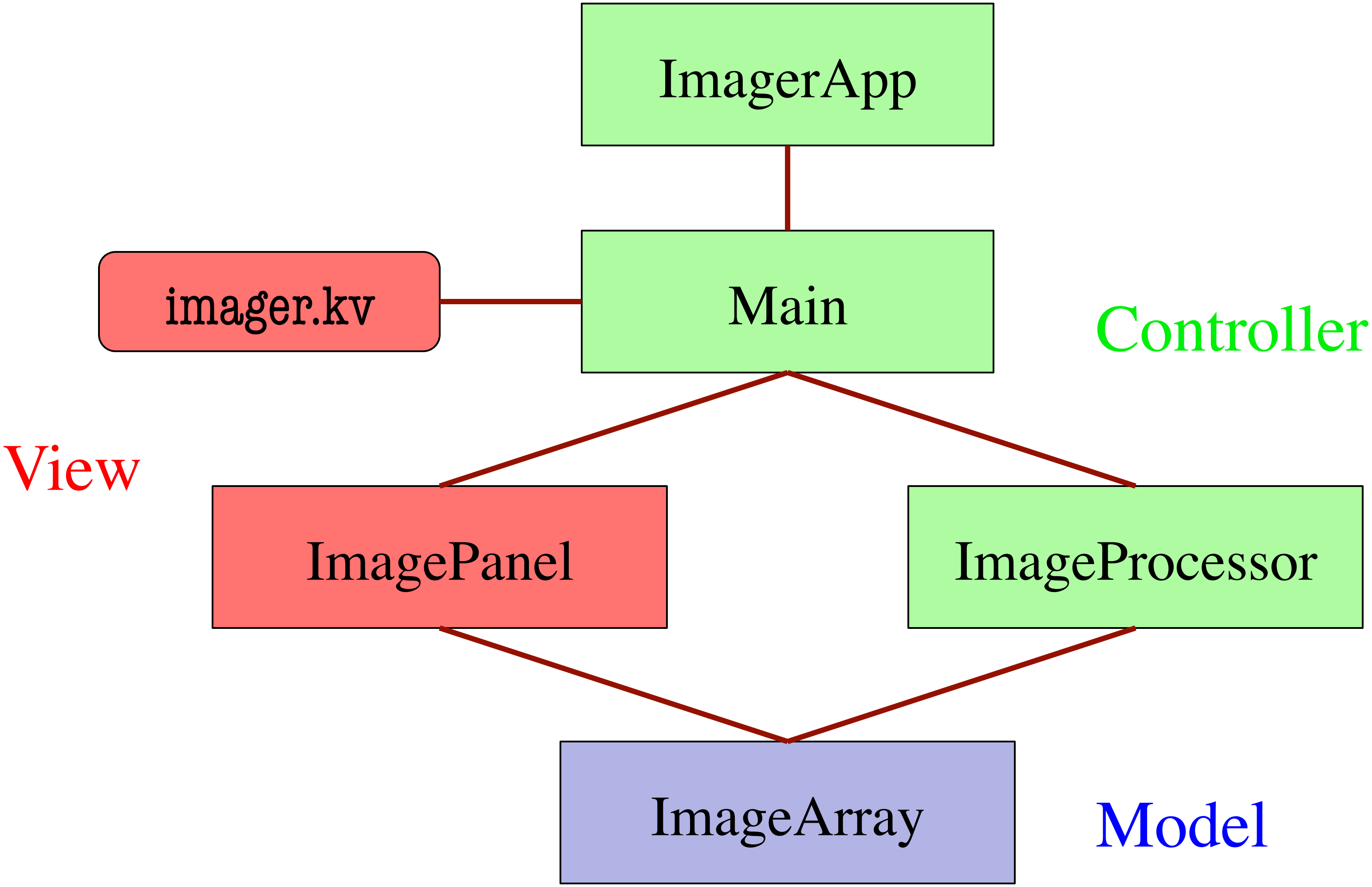
View



Another View

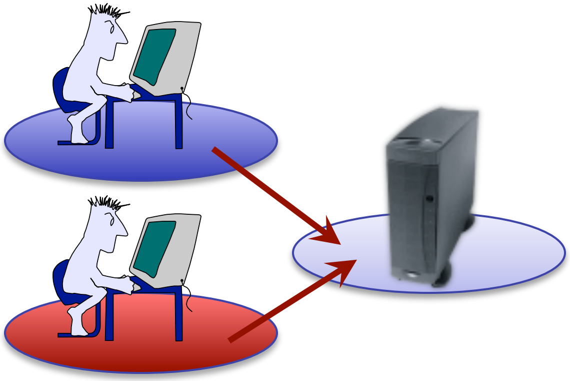


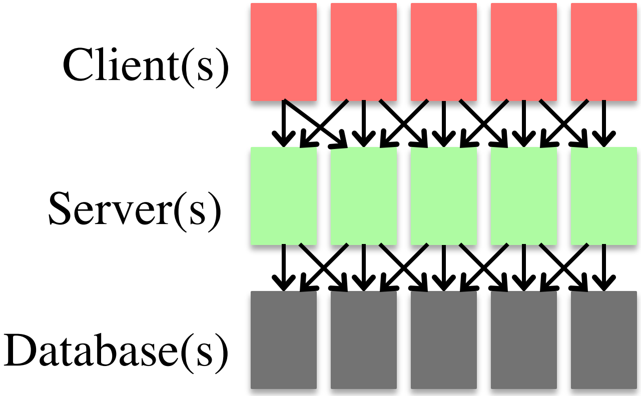
MVC and Assignment 6



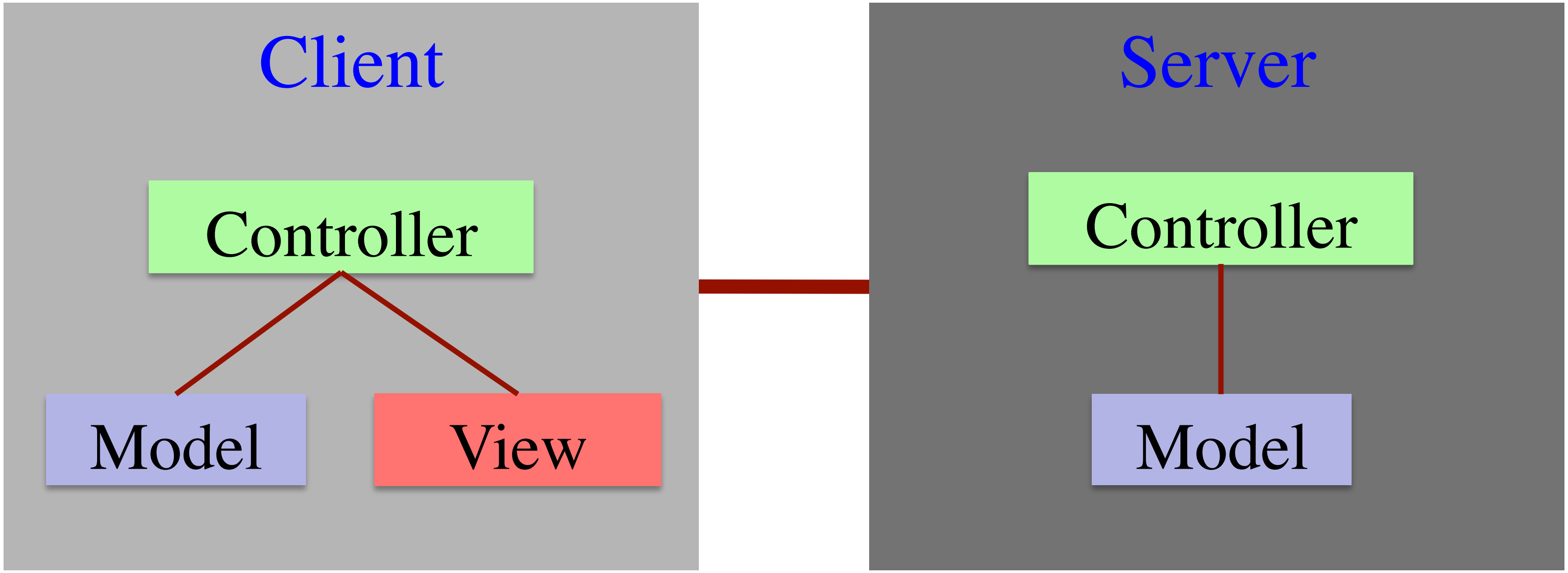
Beyond Model-View-Controller

- MVC is best 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
  - Recently renumbered as CS 3152

Software Engineering