Lecture 23

Controllers and Object Oriented Design

Announcements for This Lecture

Exams

- Not so top heavy this time
  - Mean: 74, **Median**: 77
  - For-loop, not recursion hard
- Good grade distribution
  - **A**: Mid 80s up
  - **B**: Mid-low 60s to mid 80s
  - **C**: 35 to mid-low 60s
- Final should be similar
  - More time, more questions

Assignment & Lab

- A5 is due tonight!
  - Grading will be curved
  - Parts A-F get at least A-
- Finish Survey by Friday
  - New questions this time!
- Today's lab is on invariants
  - Due before Thanksgiving
  - No official lab next week
  - But will be there on Tues

Computer Game Development

Credits: Planetfall (1983)

Steve Meretzky

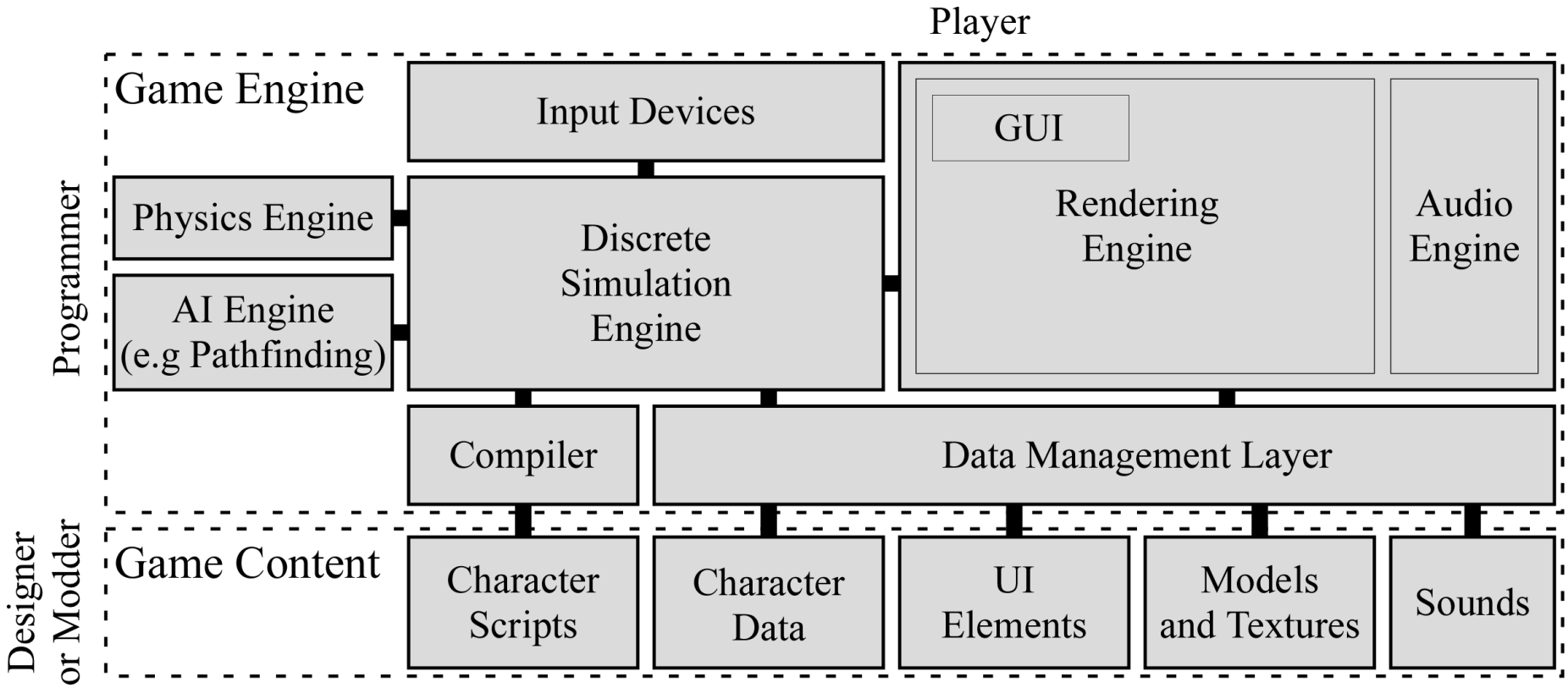
Computer Game Development

Credits: Planetfall (1983)

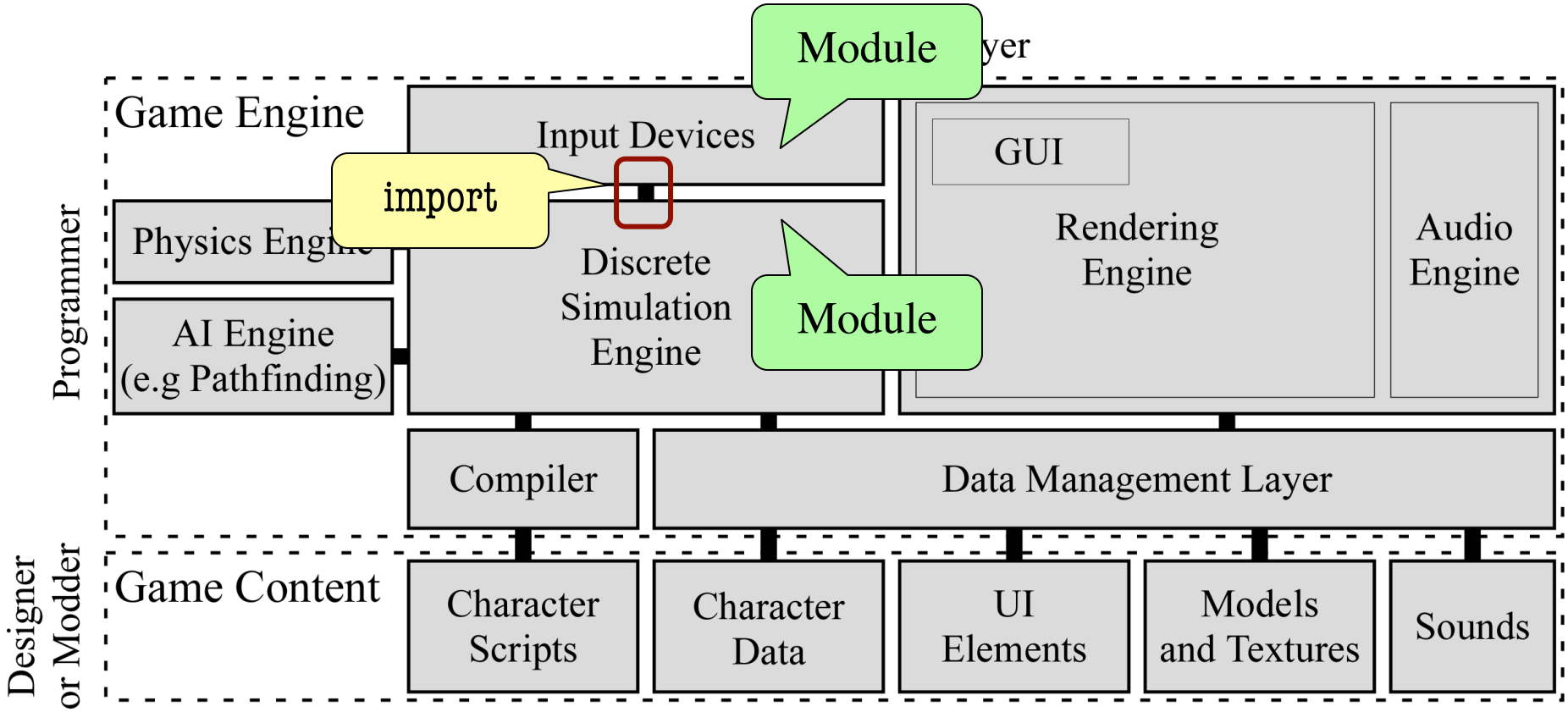
Steve Meretzky

Credits: Portal (2007)

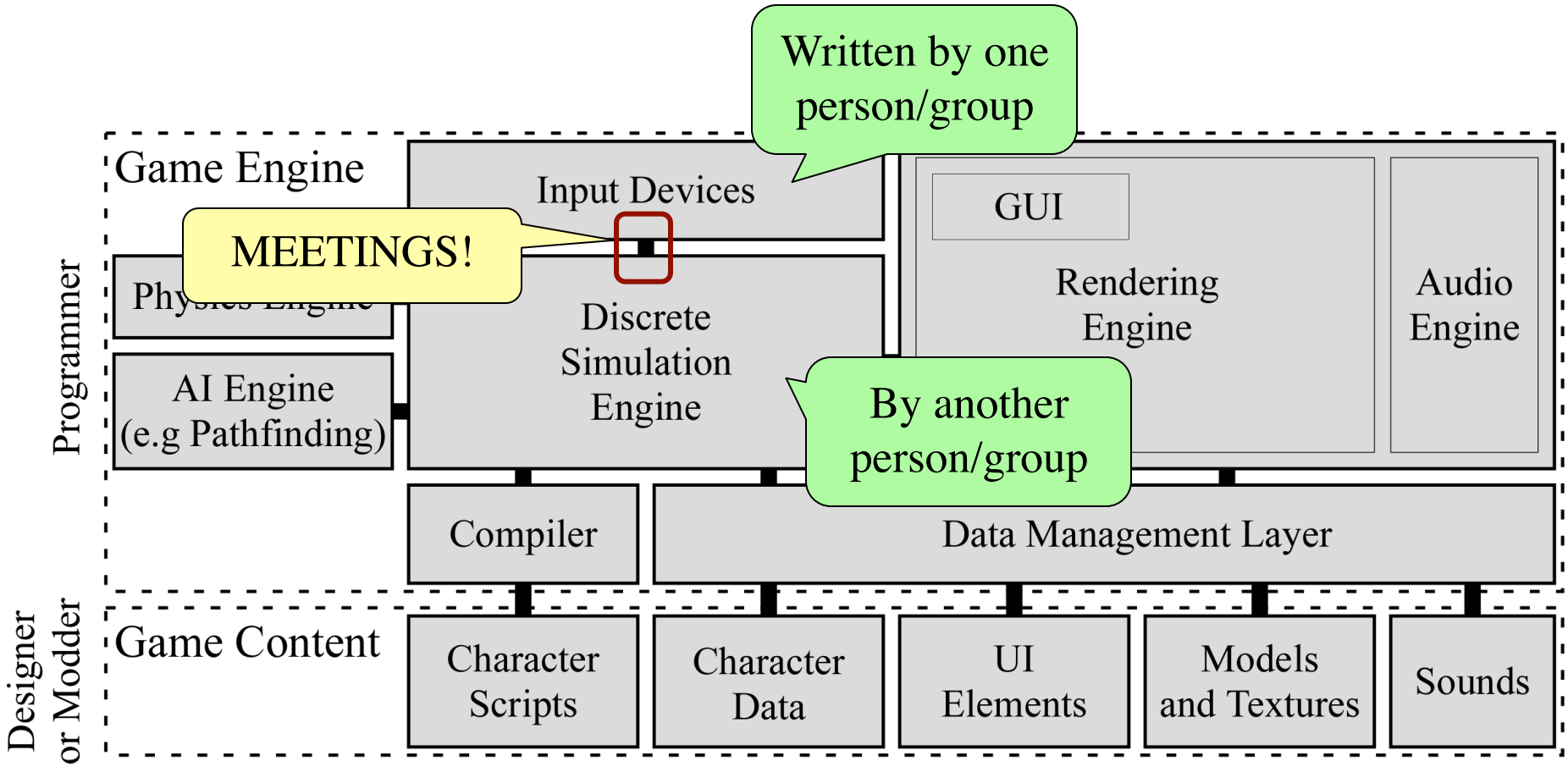
Challenge: Breaking Up Software



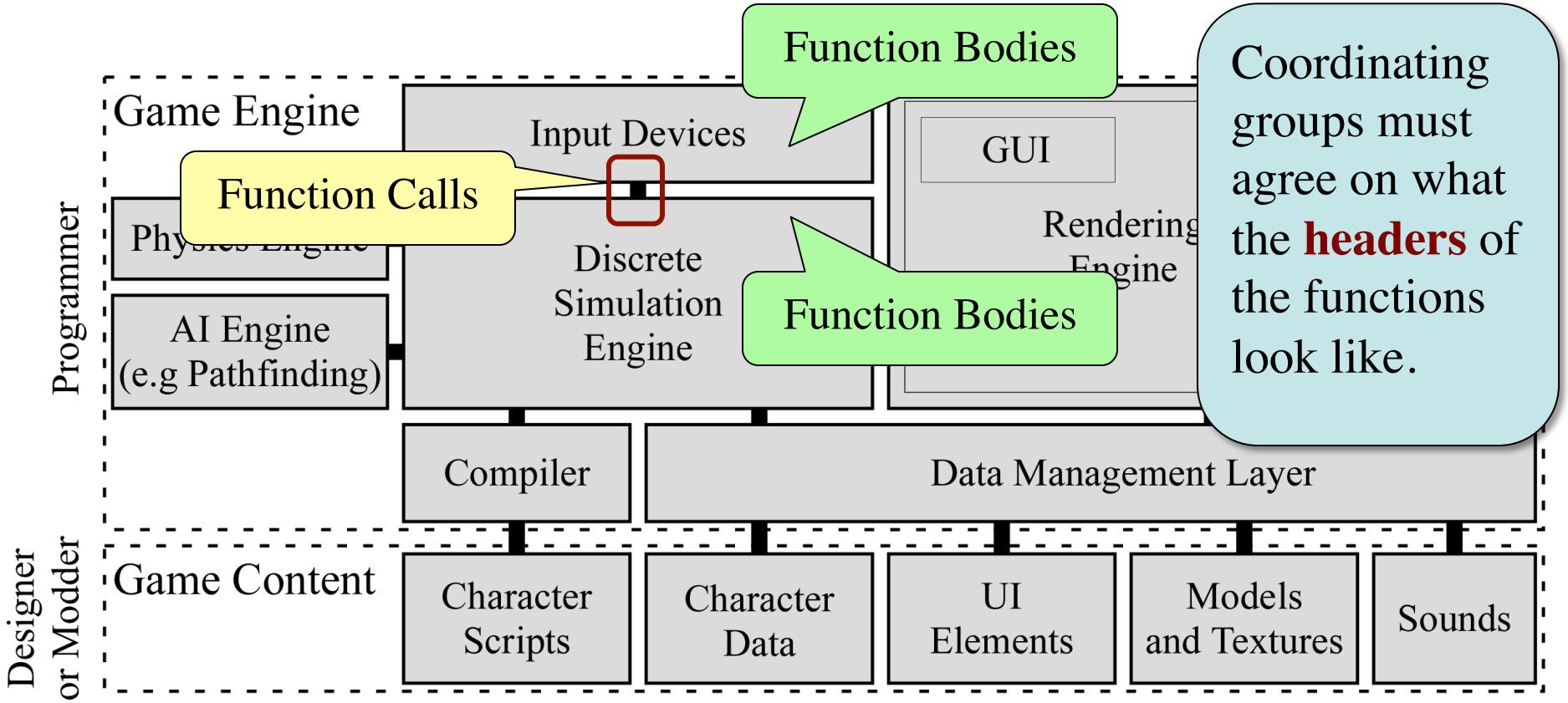
Challenge: Breaking Up Software



Challenge: Breaking Up Software



Challenge: Breaking Up Software

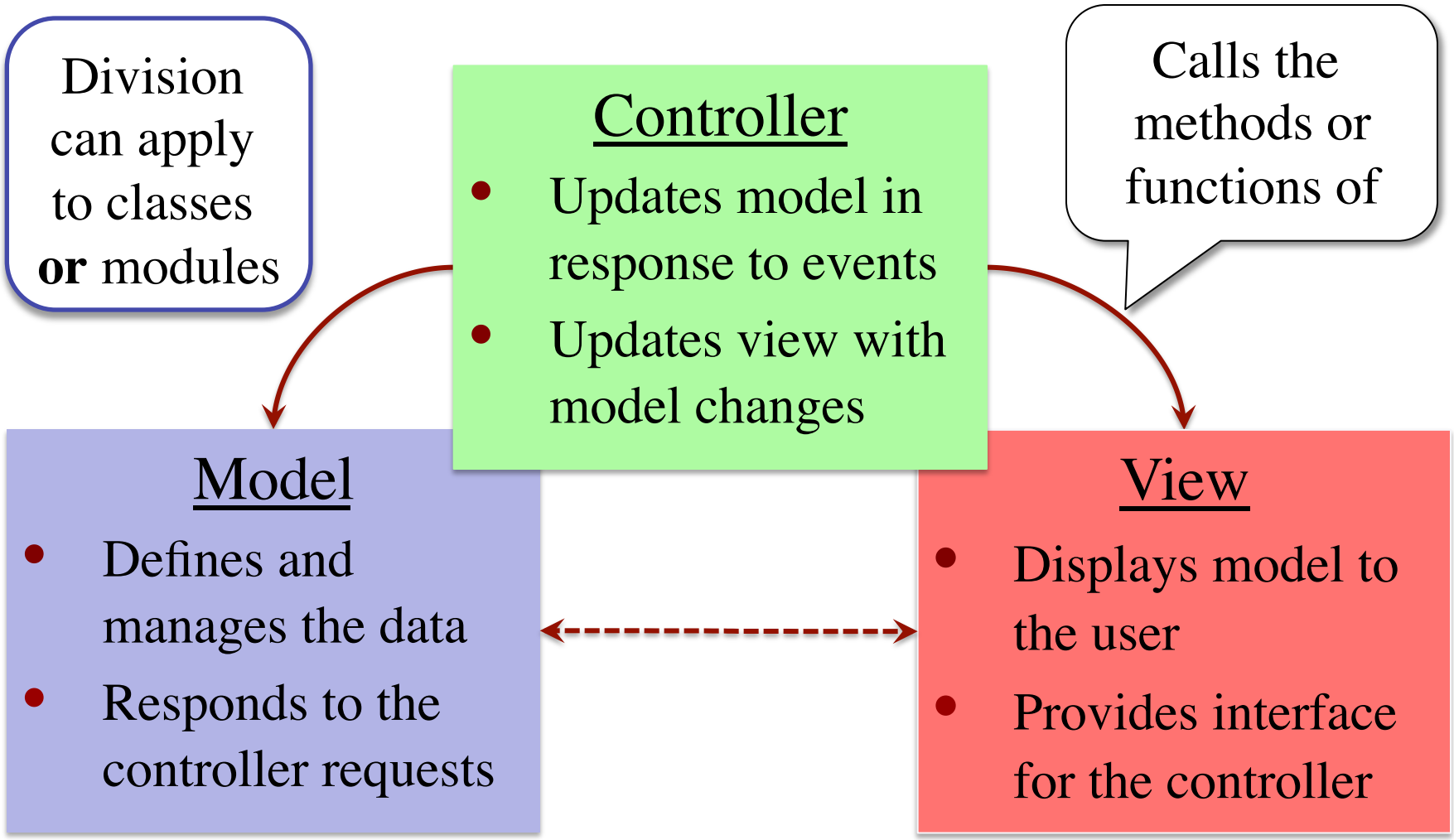


Software Patterns

- **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



MVC in this Course

Model

- **A3**: Color classes
  - RGB, CMYK & HSV
- **A4**: Turtle, Pen
  - Window is **View**
- **A5**: Database, Cluster
  - Data is always in model
- **A6**: Ball, Brick, etc..
  - All shapes/geometry

Controller

- **A3**: a3app.py
  - Hidden classes
- **A4**: Functions in a4.py
  - No need for classes
- **A5**: best_cluster
  - But visualizer is a class
- **A6**: Breakout
  - Controller class for you!

MVC in this Course

Model

- **A3**: Color classes
  - RGB, CMYK & HSV
- **A4**: Turtle, Pen
  - Window is **View**
- **A**
- **A6**: Ball, Brick, etc..
  - All shapes/geometry

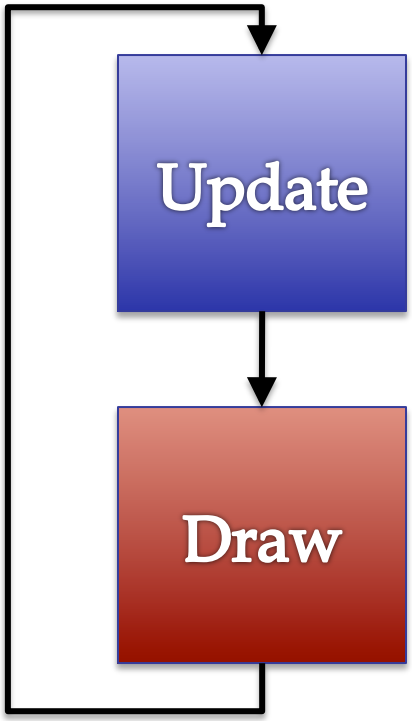
Why **classes** sometimes and **functions** others?

Controller

- **A3**: a3app.py
  - Hidden classes
- **A4**: Functions in a4.py
  - No need for classes
- **A5**: best_cluster
  - But visualizer is a class
- **A6**: Breakout
  - Controller class for you!

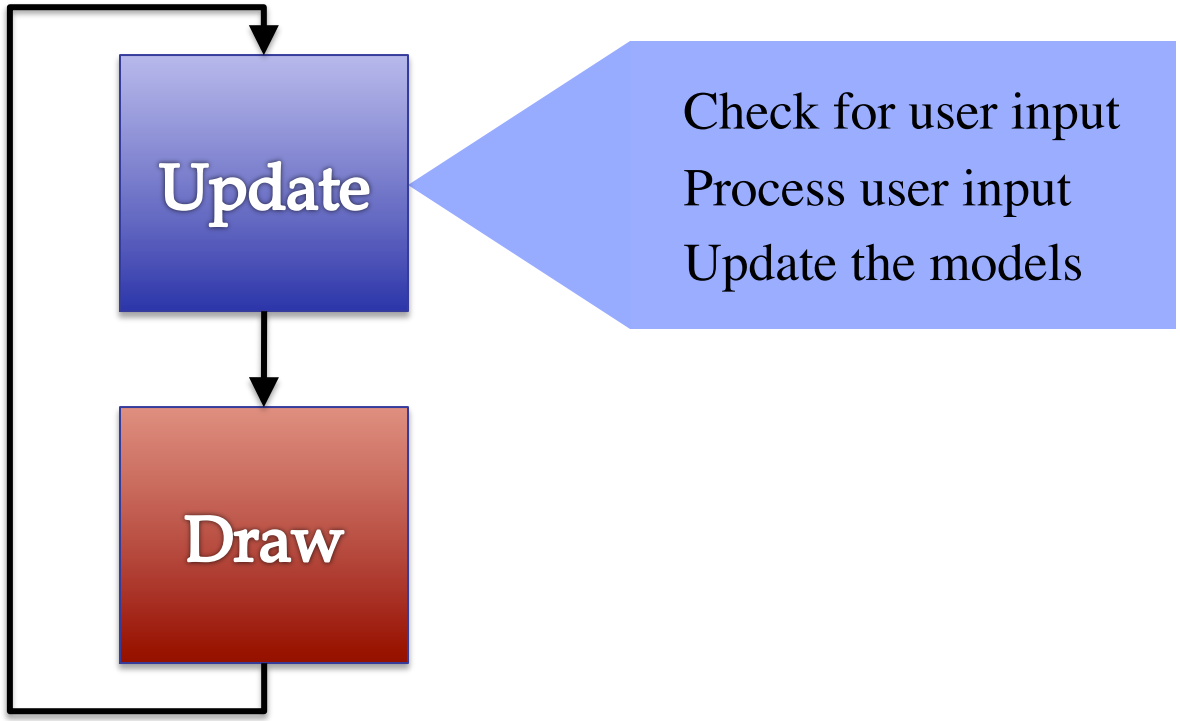
A Standard GUI Application

Animates the application, like a movie



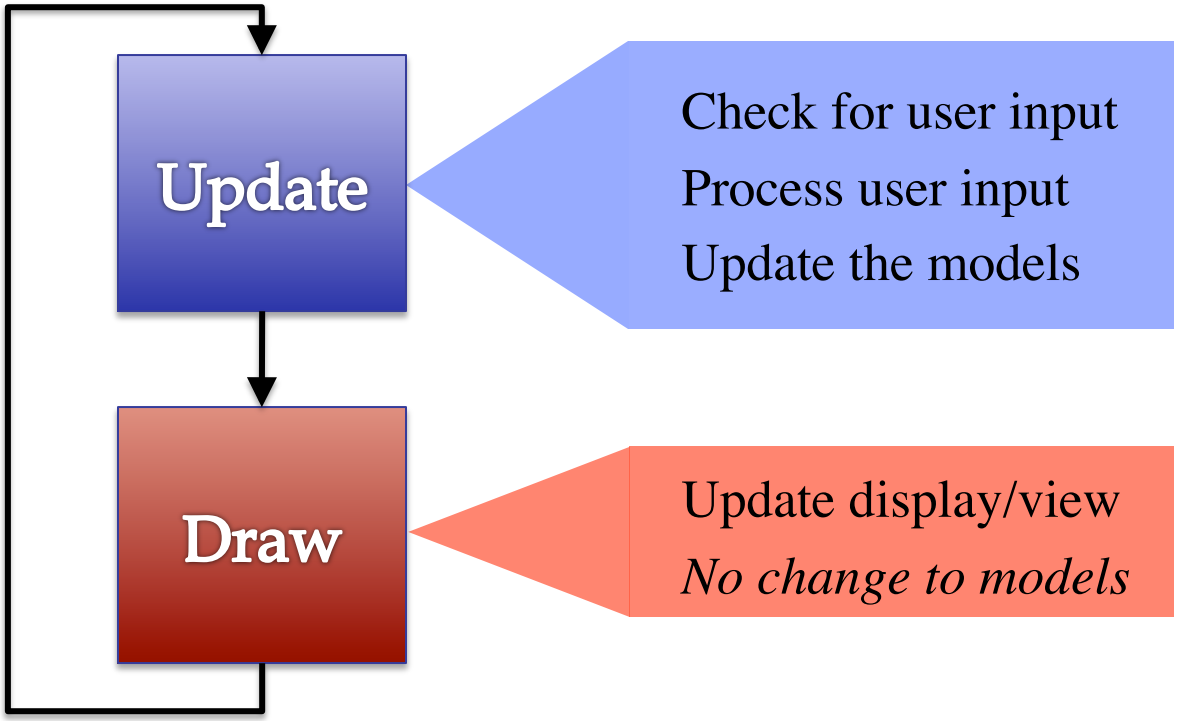
A Standard GUI Application

Animates the application, like a movie

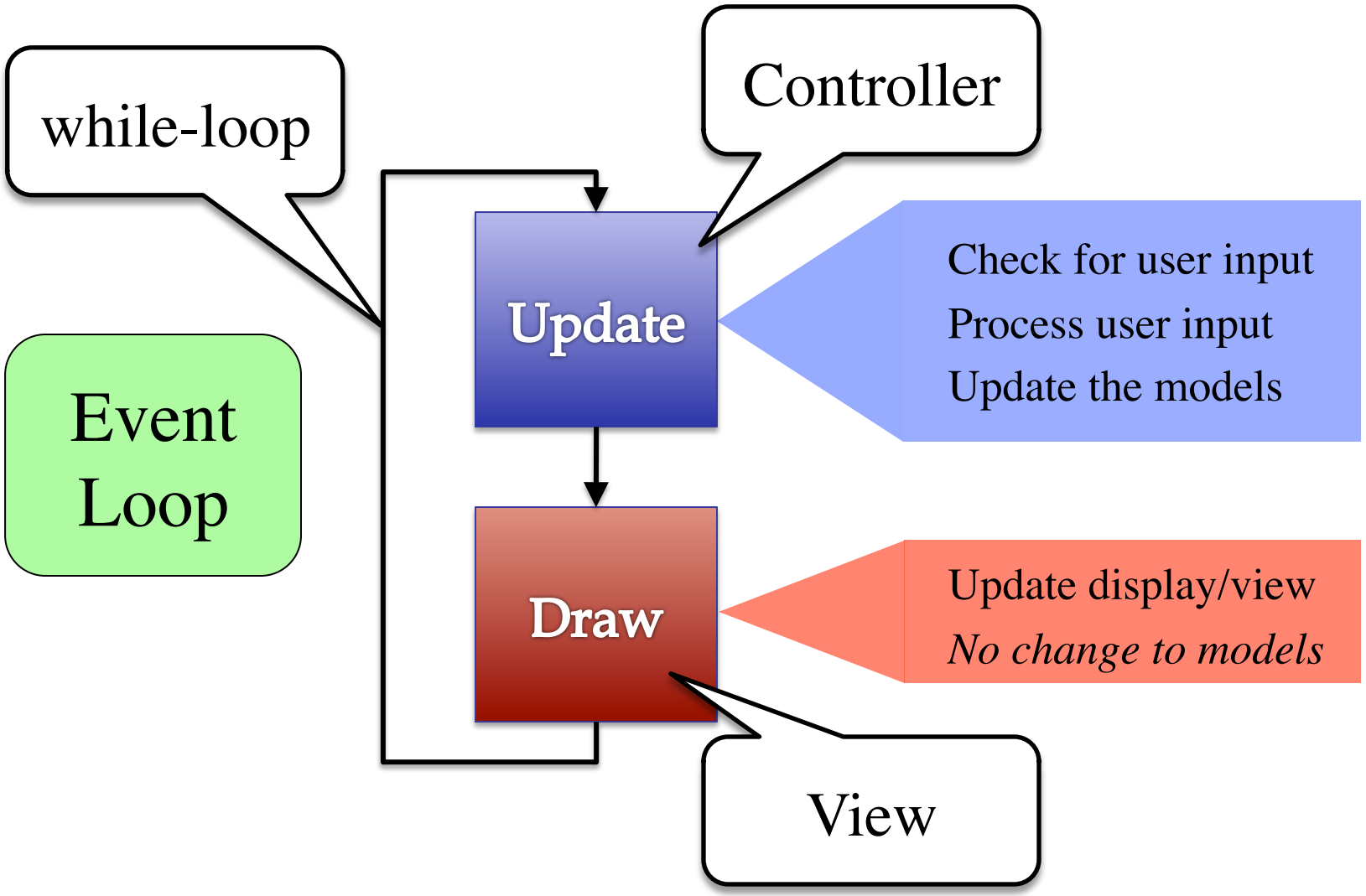


A Standard GUI Application

Animates the application, like a movie



A Standard GUI Application



Must We Write this Loop Each Time?

```
while program_is_running:
    # Get information from mouse/keyboard
    # Handled by OS/GUI libraries

    # Your code goes here



    # Draw stuff on the screen
    # Handled by OS/GUI libraries
```

Must We Write this Loop Each Time?

```
while program_is_running:
    # Get information from mouse/keyboard
    # Handled by OS/GUI libraries

    # Your code goes here



    # Draw stuff on the screen
    # Handled by OS/GUI libraries
```

Would like to "plug in" code

Why do we need to write this each time?

Must We Write this Loop Each Time?

```
while program_is_running:
    # Get information from mouse/keyboard
    # Handled by OS/GUI libraries

    # Your code go
    controller.doloop()

    n the screen
    # Handled by OS/GUI libraries
```

Method call (for loop body)

Controller object

- Write loop body in a controller.
- OS/GUI handles everything else.

Loop Invariants Revisited

Normal Loops

Properties of "external" vars

```
x = 0
i = 2
# x = sum of squares of 2..i
while i <= 5:
    x = x + i*i
    i = i + 1
# x = sum of squares of 2..5
```

Controller

What are the "external" vars?

```
while program_running:
    # Get input
    # Your code called here
    controller.doloop()
    # Draw
```

Loop Invariants Revisited

Normal Loops

```
x = 0
i = 2
# x = sum of squares of 2..i
while i <= 5:
    x = x + i*i
    i = i +1
# x = sum of squares of 2..5
```

Properties of "external" vars

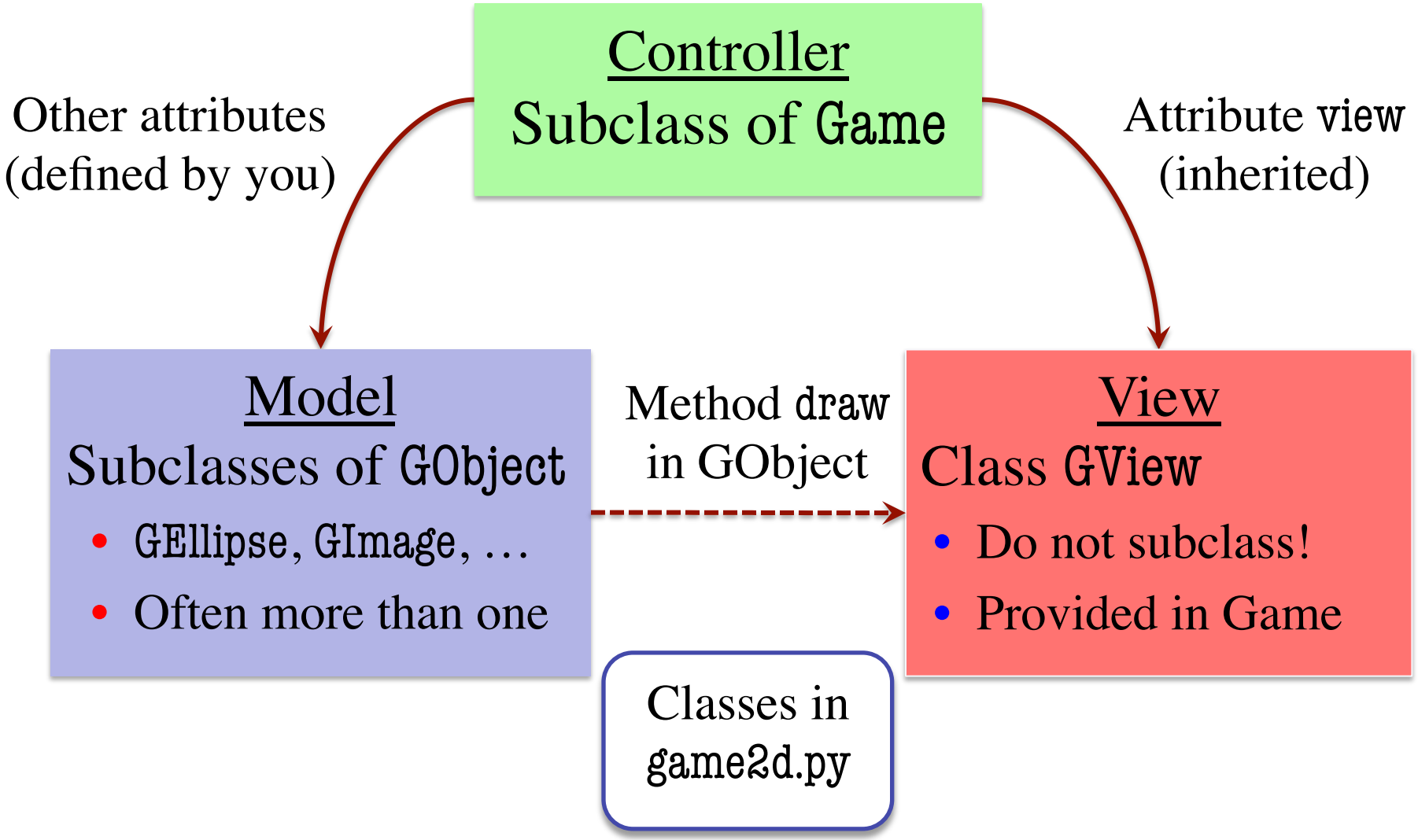
Controller

What are the "external" vars?

```
while program_running:
    # Get input
    # Your code called here
    controller.doloop()
    # Draw
```

controller is an object. It will have **attributes**!

Model-View-Controller in CS 1110



Attribute Invariants = Loop Invariants

- Attributes are a way to store value between calls
  - Not part of call frame
  - Variables outside loop
- A controller needs
  - Loop attributes
  - Initialization method (for loop, not __init__)
  - Method for body of loop
- Attribute descriptions, invariants are important

```
game = Game(...) #constructor
...
game.init() #Loop initialization
# inv: game attributes are ...
while program_running:
    # Get input
    # Your code goes here
    game.update(time_elapsed)
    game.draw()
# post: game attributes are ...
```

Example: Animation

```
class Animation(Game):
    """Application to an ellipse in a circle."""

    def init(self):
        """Special loop initialization method."""
        ...

    def update(self,dt):
        """Change the ellipse position."""
        ...

    def draw(self):
        """Draw the ellipse"""
        ...
```

See animation.py

Loop initialization
Do NOT use __init__

Loop body

Use method draw()
defined in GObject

Example: Animation

```
class Animation(Game):
    """Application to an el
    def init(self):
        """Special loop initialization method."""
        ...
    def update(self,dt):
        """Change the ellipse position."""
        ...
    def draw(self):
        """Draw the ellipse"""
        ...
```

Parent class that does hard stuff

See animation.py

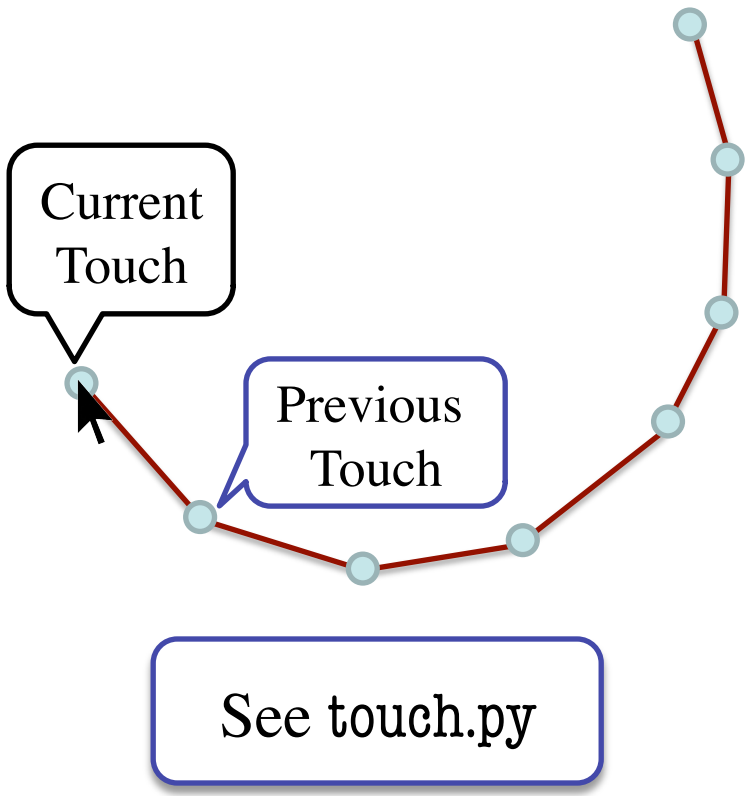
Loop initialization
Do NOT use __init__

Loop body

Use method draw() defined in GObject

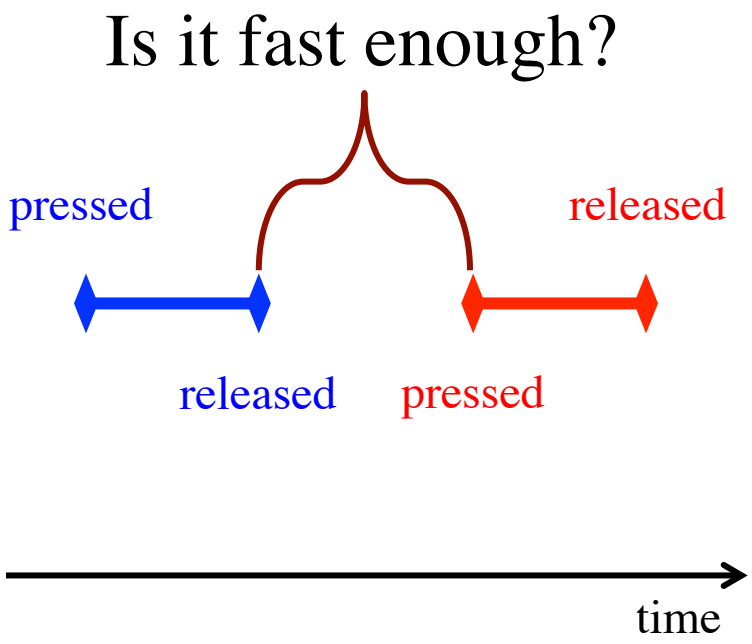
What Attributes to Keep: Touch

- Attribute touch in GView
  - The mouse press position
  - Or None if not pressed
  - Use self.view.touch inside controller (Game) methods
- Compare touch, last position
  - last None, touch not None: Mouse button **pressed**
  - last not None, touch None: Mouse button **released**
  - last and touch not None: Mouse **dragged** (button down)



More Attributes: Checking Click Types

- Double click = 2 fast clicks
- Count number of fast clicks
  - Add an attribute `clicks`
  - Reset to 0 if not fast enough
- Time click speed
  - Add an attribute `time`
  - Set to 0 when mouse released
  - Increment when not pressed (e.g. in loop method `update()`)
  - Check `time` when next pressed

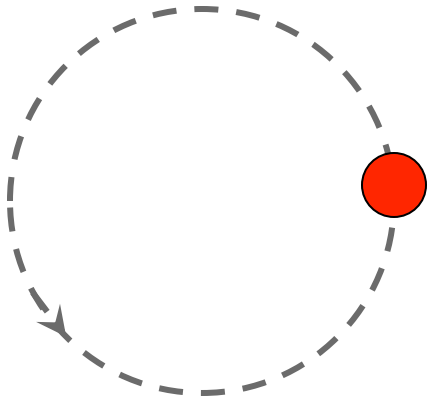


See `touch.py`

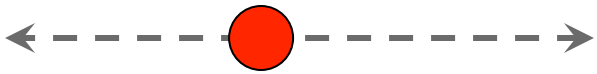
State: Changing What the Loop Does

- **State**: Current loop activity
  - Playing game vs. pausing
  - Ball countdown vs. serve
- Add an attribute `state`
  - Method `update()` checks state
  - Executes correct helper
- How do we store state?
  - State is an *enumeration*; one of several fixed values
  - Implemented as an int
  - Global **constants** are values

State `ANIMATE_CIRCLE`



State `ANIMATE_HORIZONTAL`

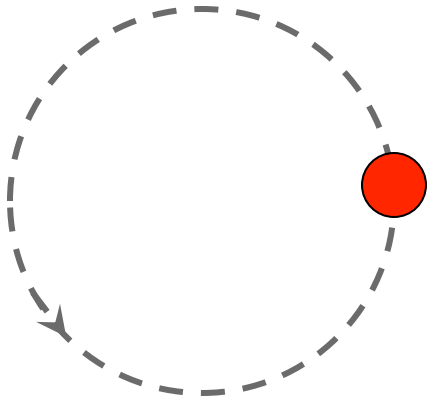


See state.py

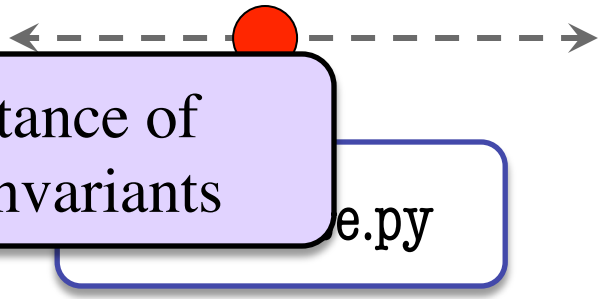
State: Changing What the Loop Does

- **State**: Current loop activity
  - Playing game vs. pausing
  - Ball countdown vs. serve
- Add an attribute `state`
  - Method `update()` checks state
  - Executes correct helper
- How do we store state?
  - State is an *enumeration*; one of several fixed values
  - Implemented as an int
  - Global **constants** are values

State `ANIMATE_CIRCLE`



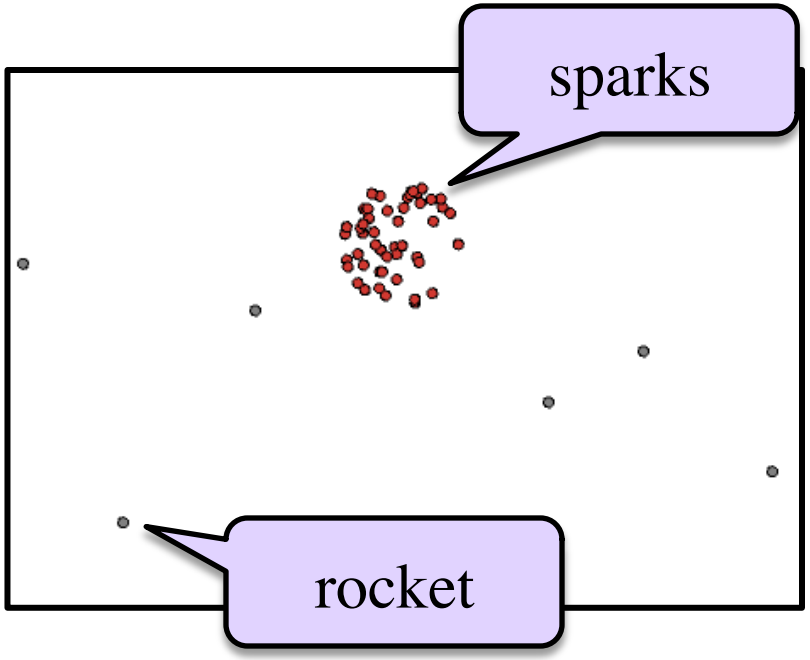
State `ANIMATE_HORIZONTAL`



Importance of class invariants

Types of Models for Assignment 6

- Often subclass of `GObject`
  - Has built-in draw method
  - See documentation in A6
- Includes groups of models
  - **Example**: rockets in `pyro.py`
  - Each rocket is a model
  - But so is the entire list!
  - `update()` will change both
- **A6**: `Model` class
  - Container, like `Database`
  - Holds bricks, ball, paddle



See `pyro.py`