Interface Builder & Mac UI Programming

Lecture 6

Model-View-Controller

The Model

- This is your data model; maintains state
- Has no idea about UI or how the data is presented
- Example: a `Fraction` class for a fraction calculator app

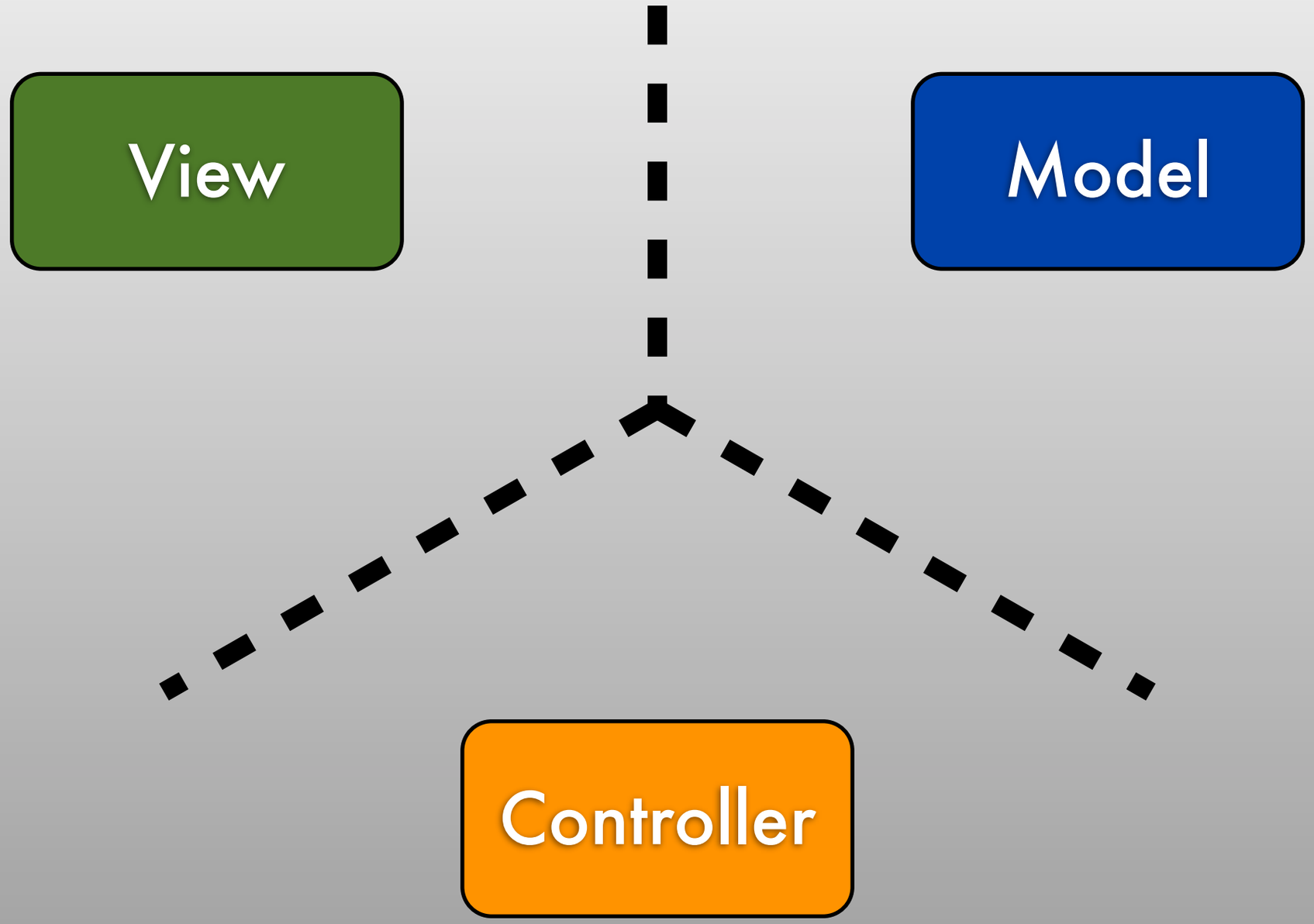
The View

- The part the user sees; the UI
- Displays the model
- Does not store data
  (except maybe caching its state)

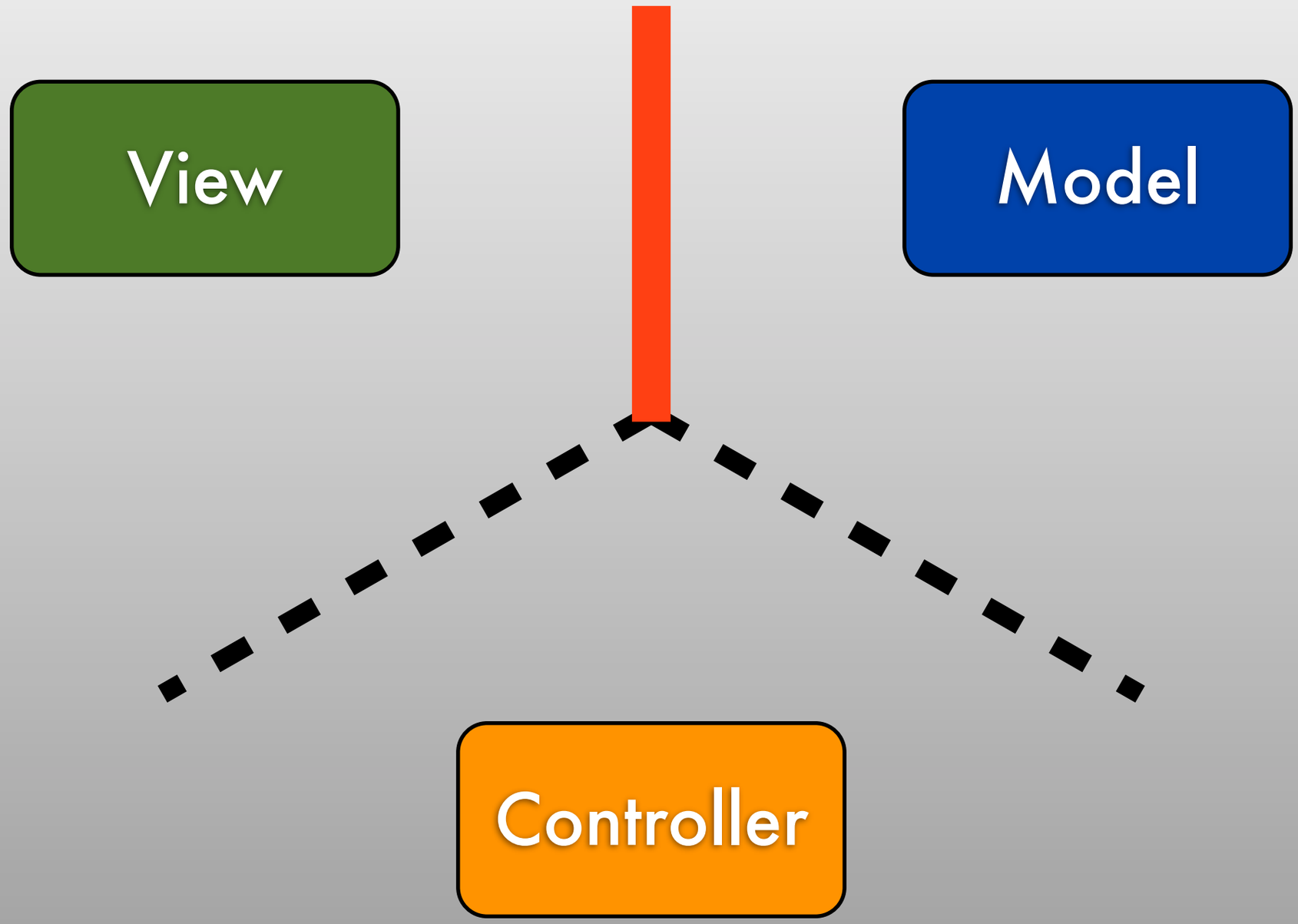
The Controller

- The middleman between the model and view
- Updates the view when the model changes
- Tells the model to update when the user manipulates the view

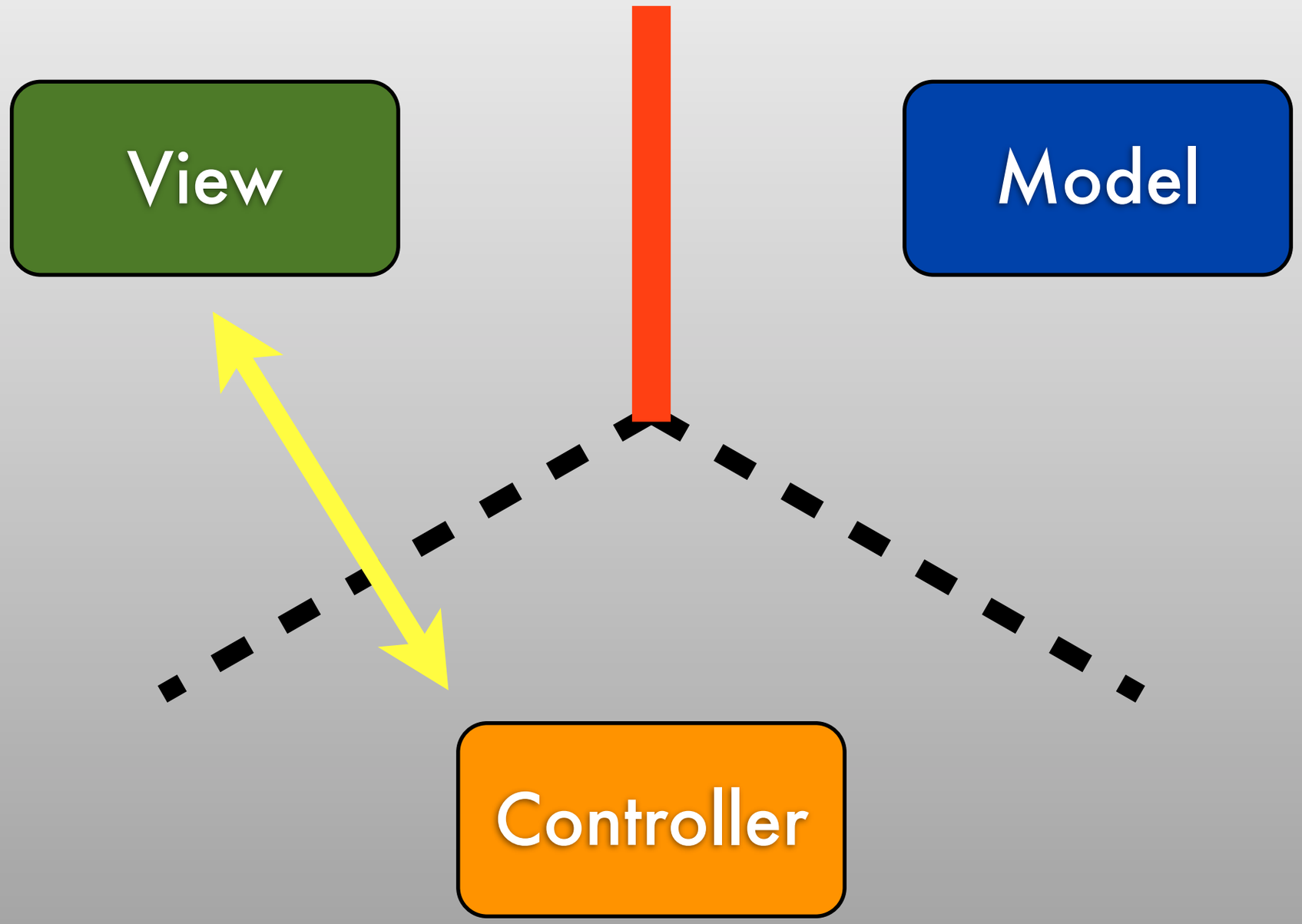
Model, View, Controller



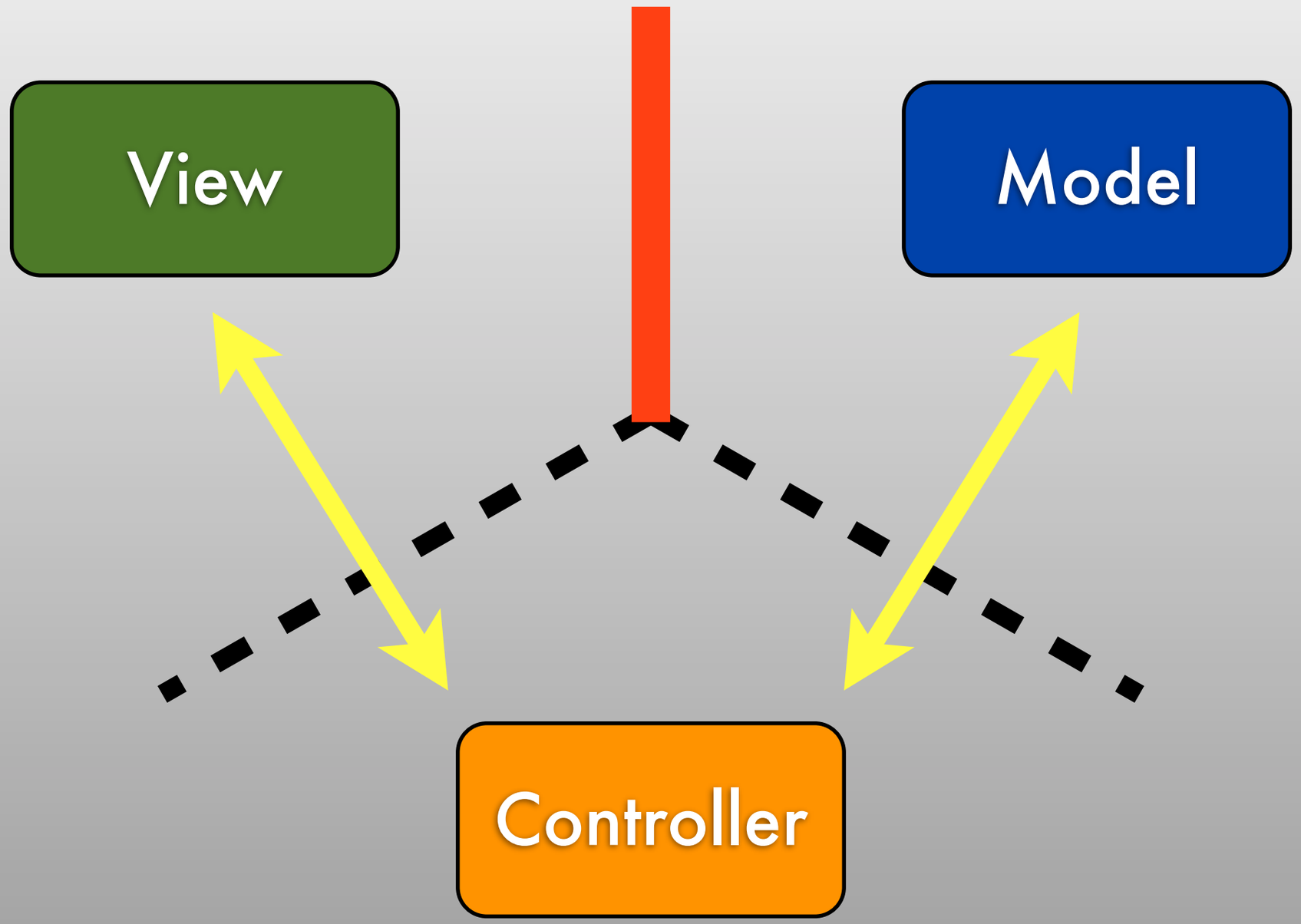
Model, View, Controller



Model, View, Controller



Model, View, Controller

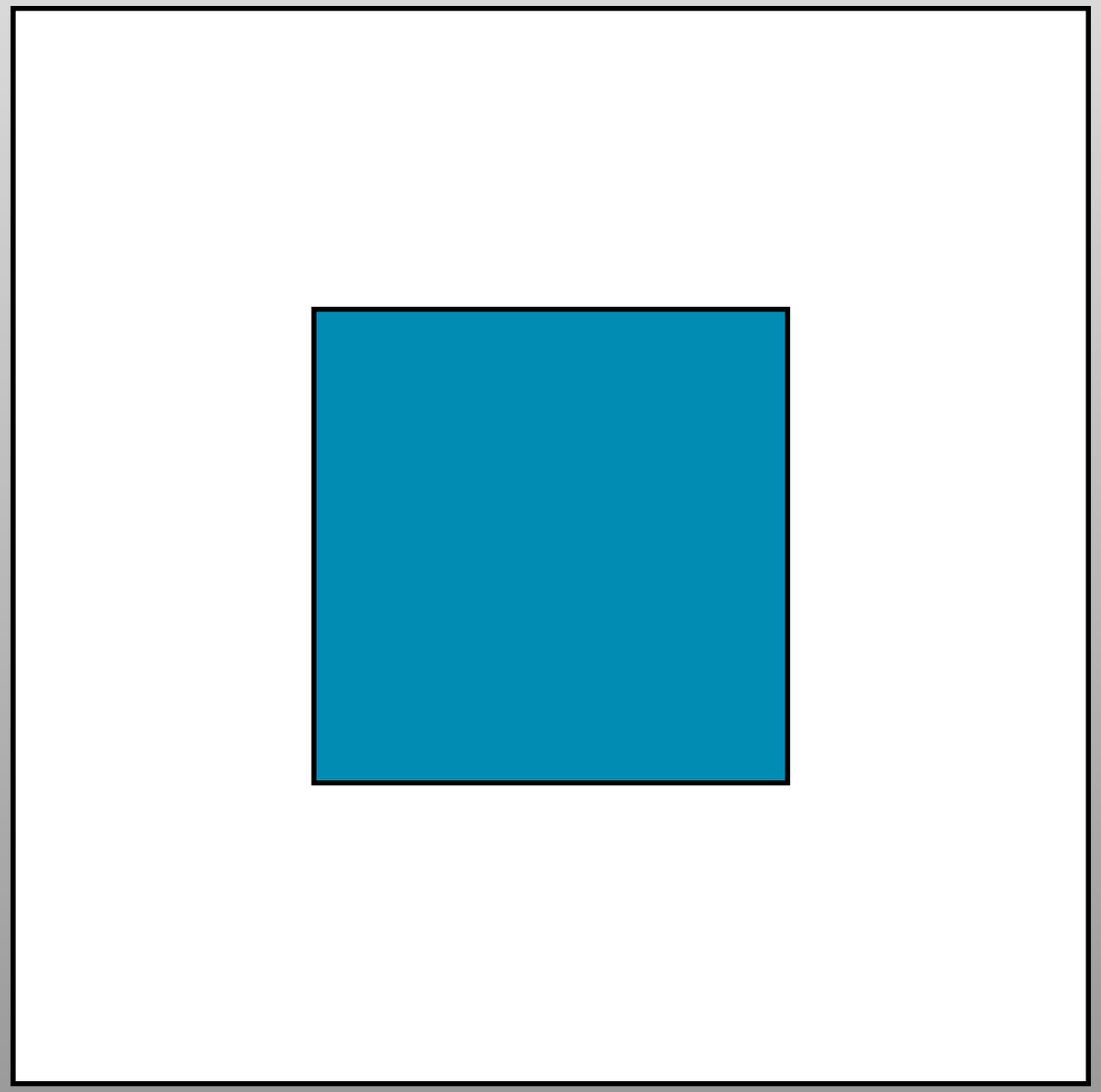


Demo: Making a UI

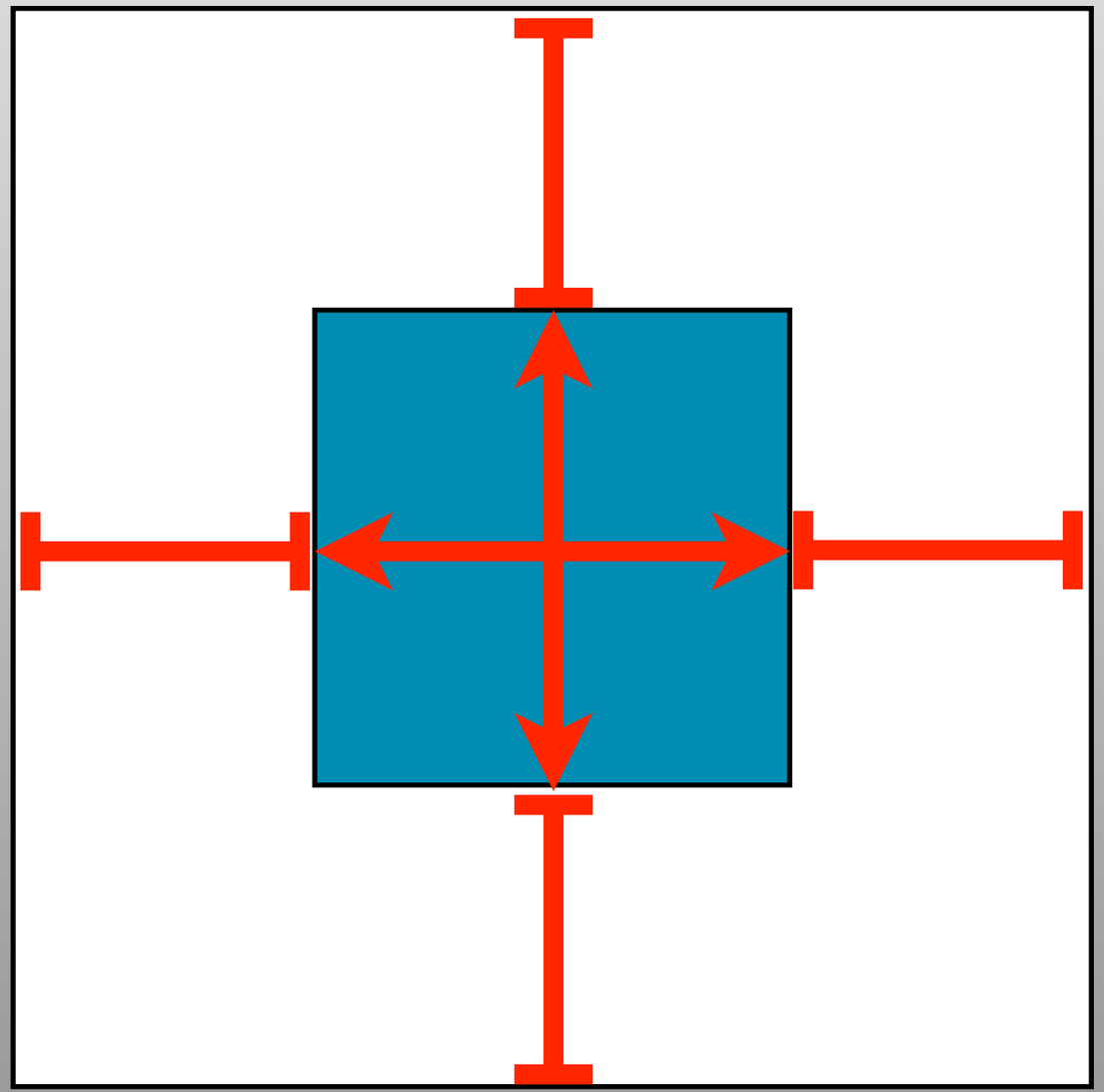
Autoresizing Views

- By default, views don't resize or move as its parent view resizes
- Views can set several properties to customize their resizing behavior
- Then Cocoa takes over and computes the appropriate size and location for the view

Struts & Springs

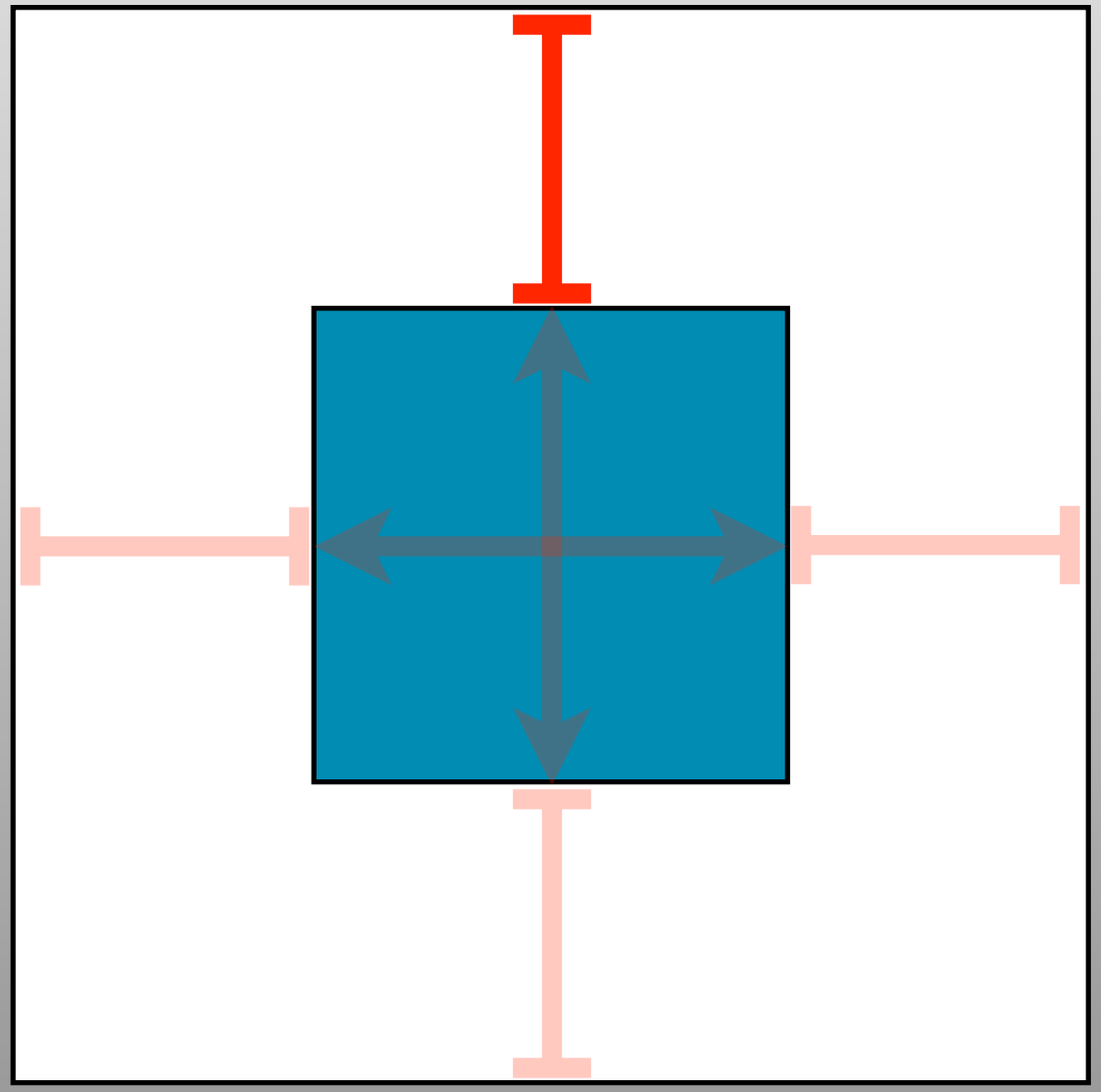


Struts & Springs



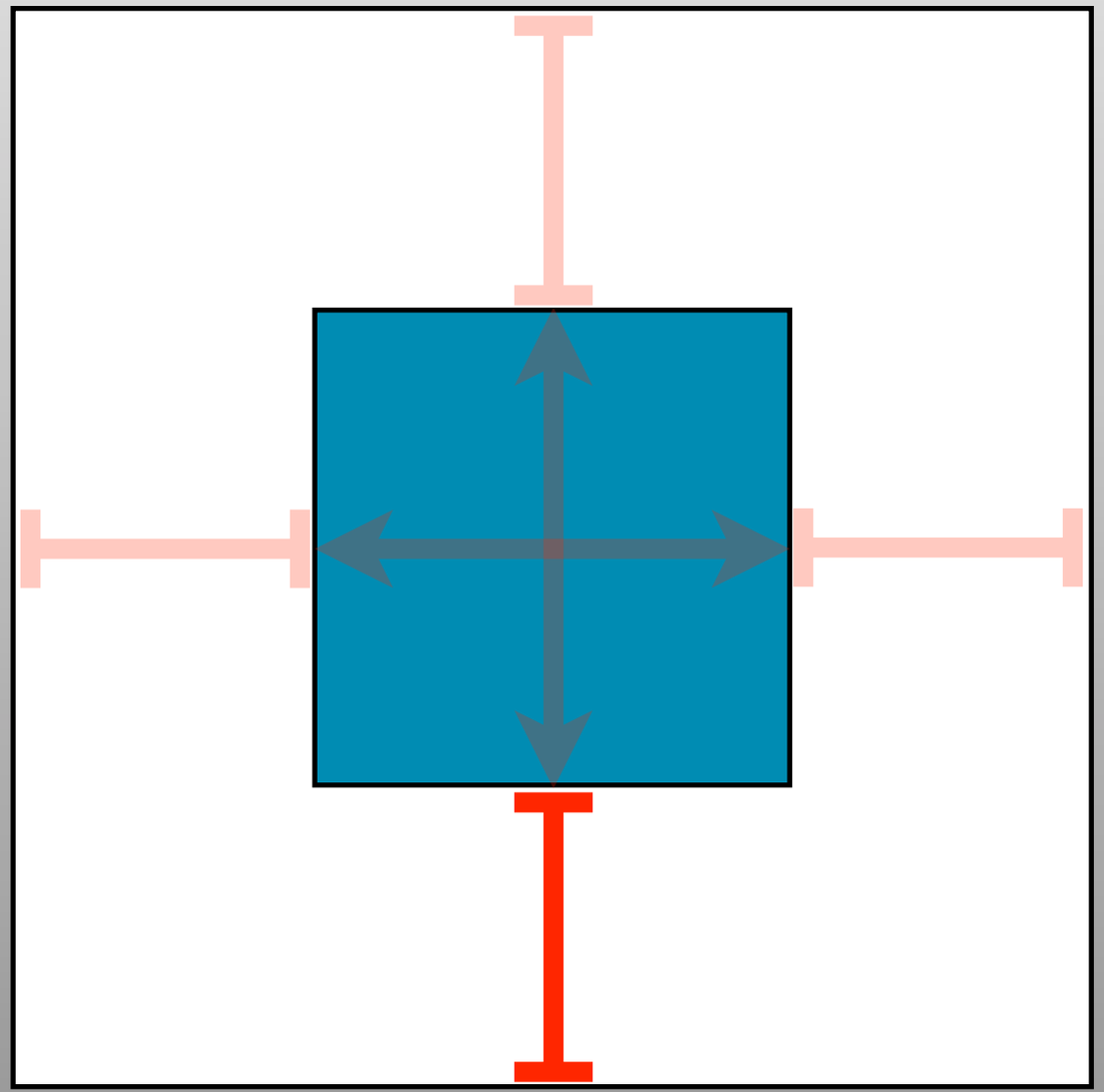
Struts & Springs

Anchor to top



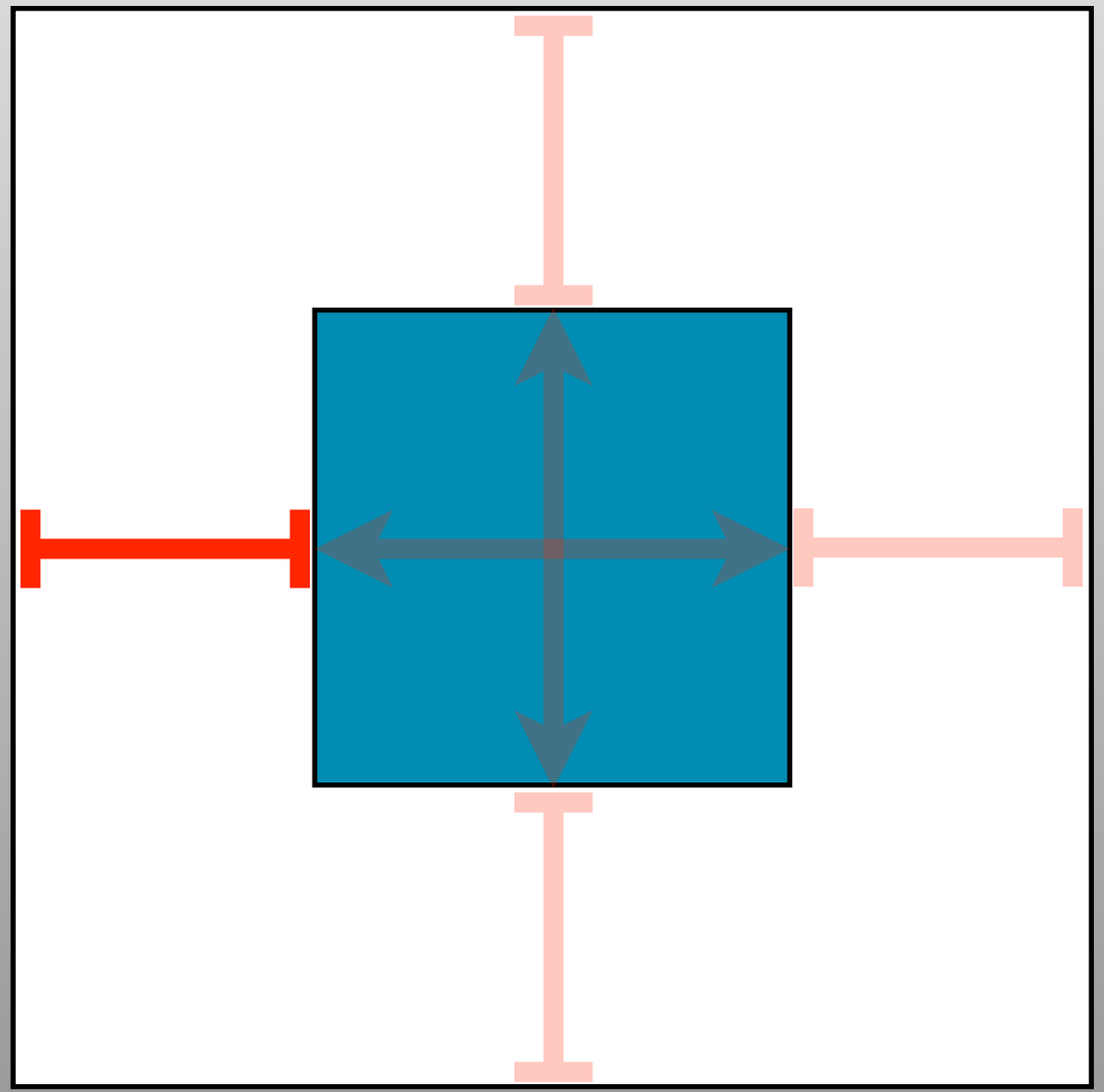
Struts & Springs

Anchor to bottom



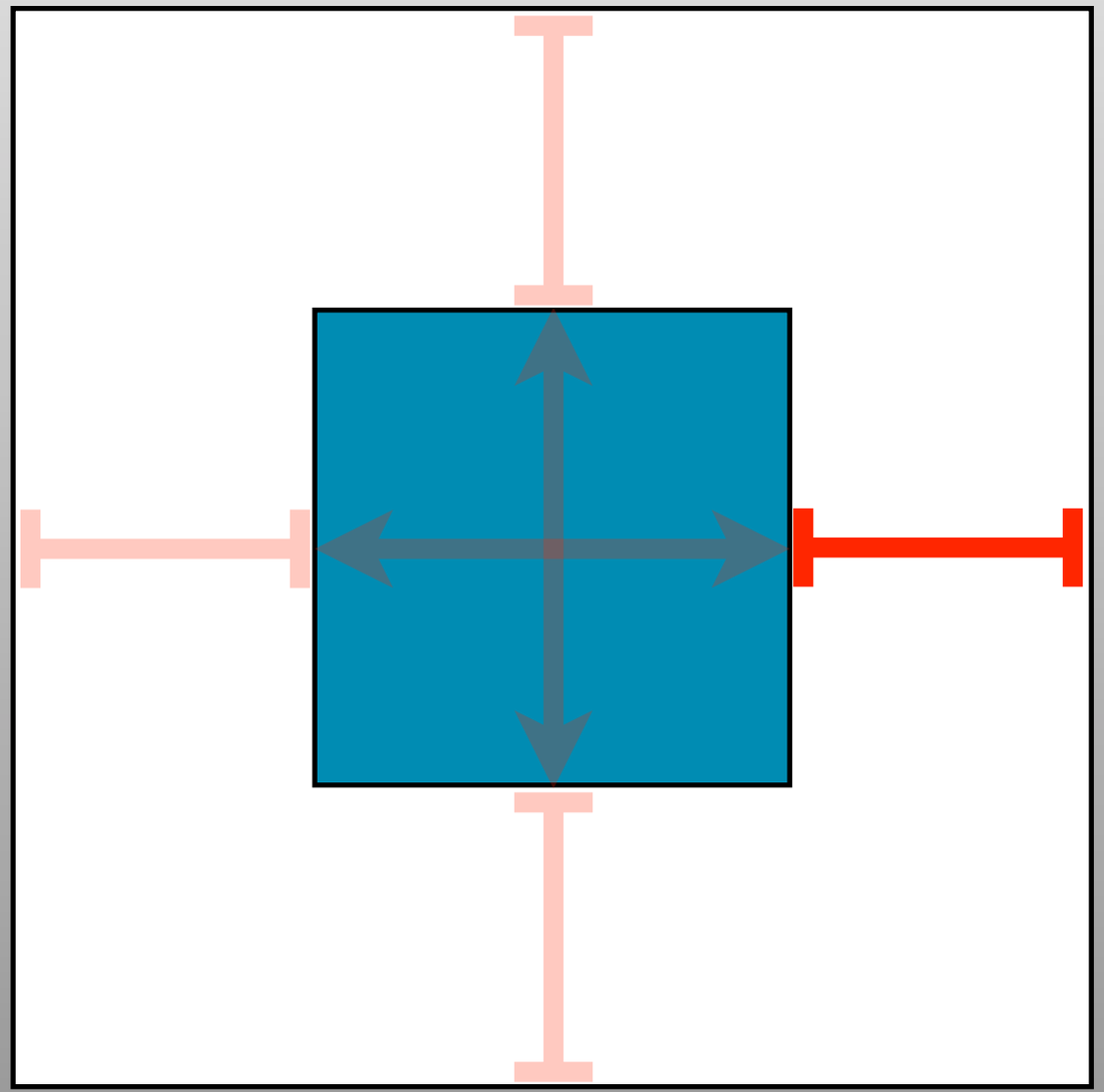
Struts & Springs

Anchor to left



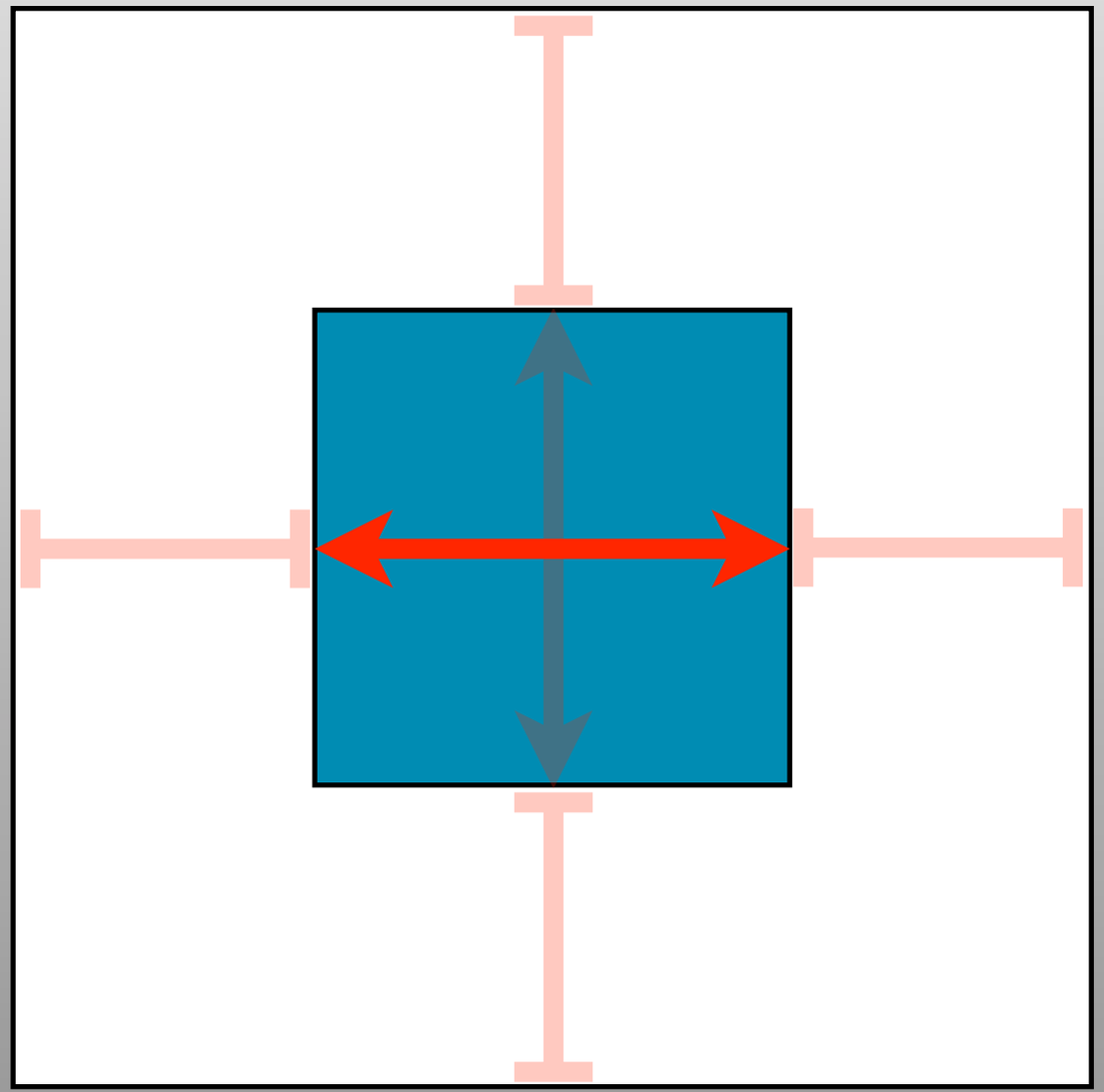
Struts & Springs

Anchor to right



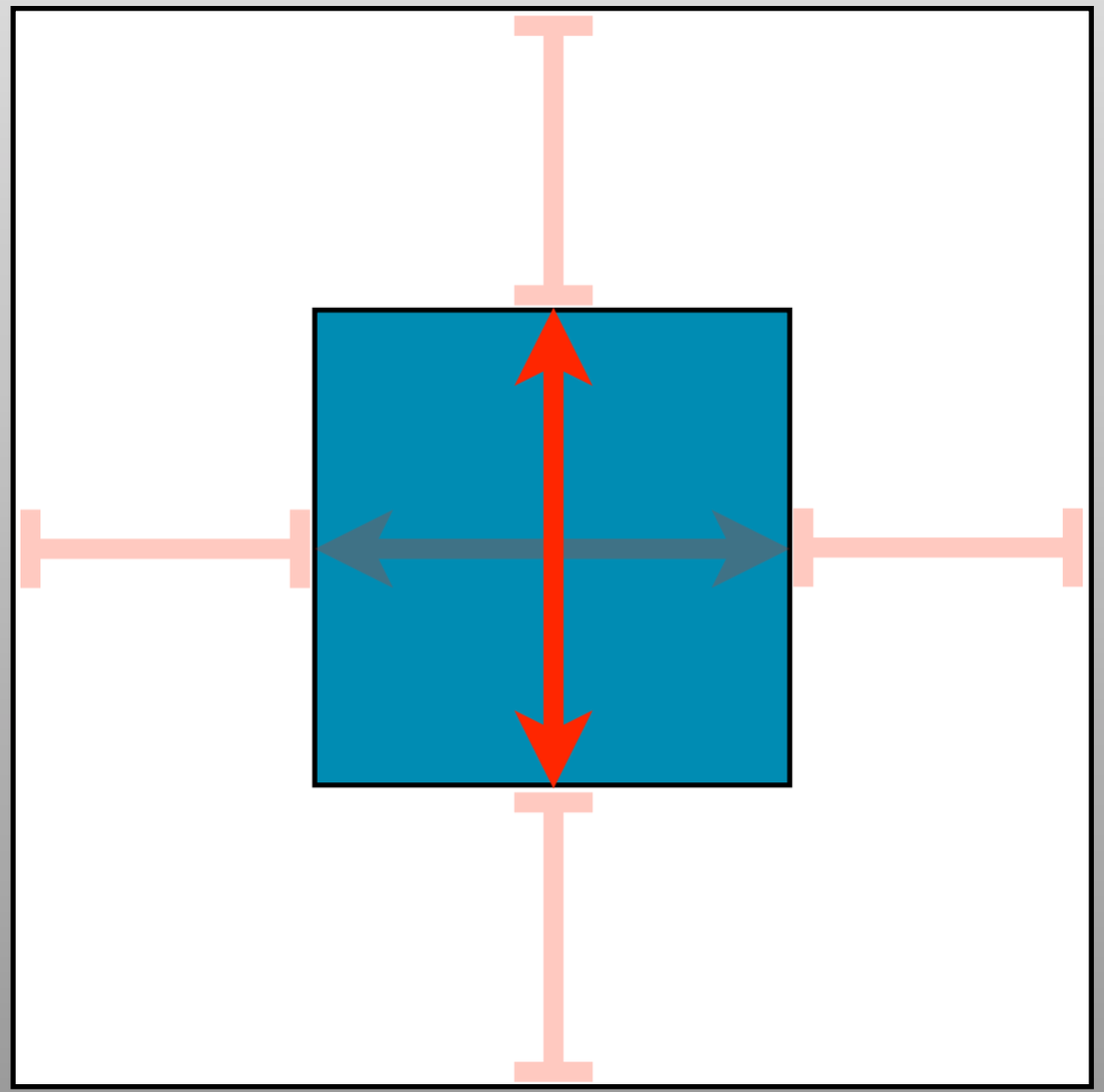
Struts & Springs

Flexible width



Struts & Springs

Flexible height



Autolayout

- New (10.7+) system for arranging views
- "Constraint" based system
  - Interface Builder automatically turns the blue guidelines it shows into constraints

Benefits

- Reduced need for in-code layout
  - Constraints accommodate dynamic content
- Improved in-code layout
  - ASCII art, improved layering
- Design flexibility
  - Baselines, alignment rect, intrinsic content size
- Localization
  - Dynamic content, intrinsic content size
- Expressiveness
  - Peer to peer relationships, inequalities, priority

Outlets & Actions

- How do we connect our code to the user interface (and vice-versa)?
- Outlets: create references in code to UI controls
- Actions: allow UI controls to send messages to objects
- Control-drag to attach

Outlets

- `IBOutlet NSButton* button;`
- `IBOutlet` is completely ignored by the compiler
  - It's simply an indicator for Interface Builder
- Caution! Outlets may not be loaded (connected) until `- (void) awakeFromNib`

Actions

- `- (IBAction) buttonClicked: (NSButton*) sender;`
- `IBAction` is `#define`'d as `void`
- The `sender` argument is required

Target-Action

- Actions can be set manually on controls
- On Mac, controls have one target-action

```
- (void) setTarget: (id) target;
- (void) setAction: (SEL) action;
```

Nib Loading

- A nib is *not* generated code.
- Rather, it is a collection of *serialized* objects
- Connects outlets & actions automatically
  - Then calls `- (void) awakeFromNib` on each object

MVC Example

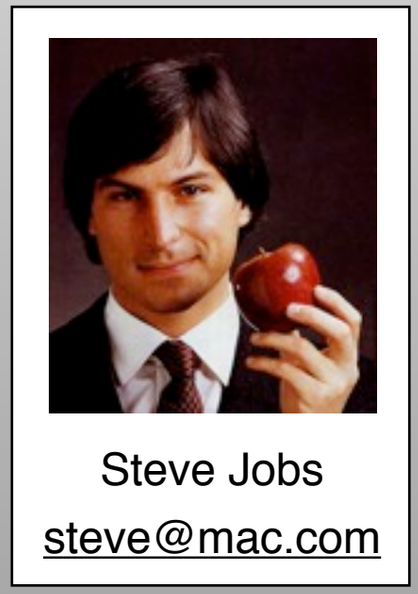
```
- (NSImage*)photo;
- (NSString*)name;
- (NSString*)email;
```

Model (Person)

MVC Example

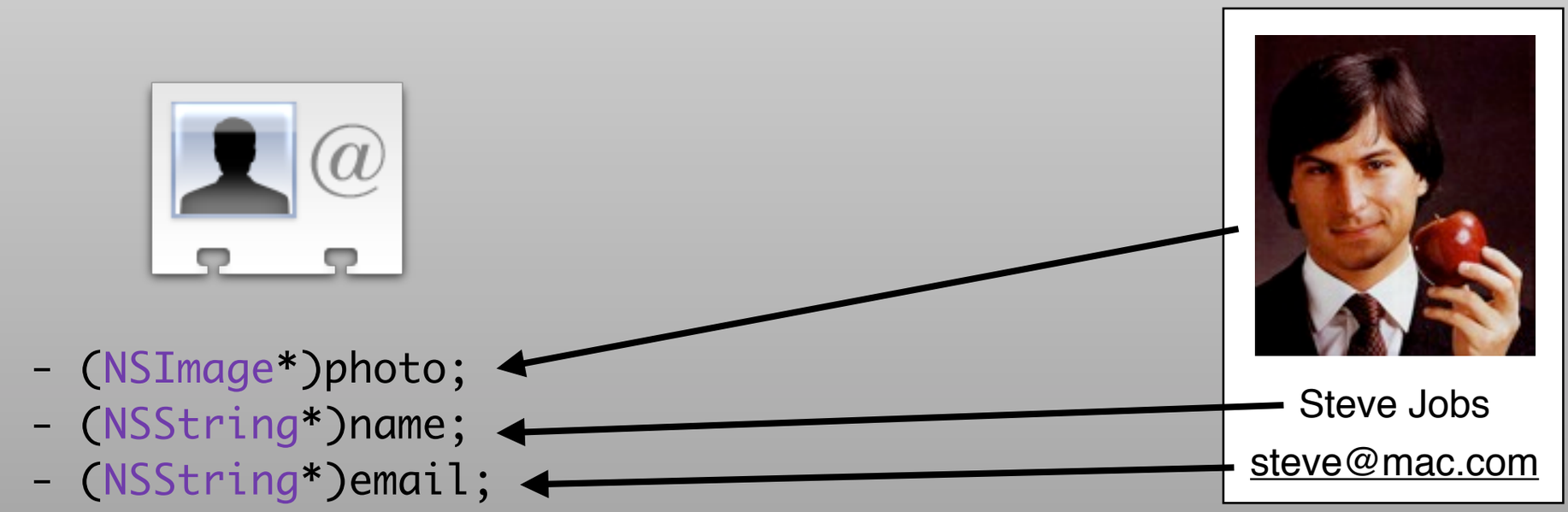
```
- (NSImage*)photo;
- (NSString*)name;
- (NSString*)email;
```

Model (Person)



View (Address Card)

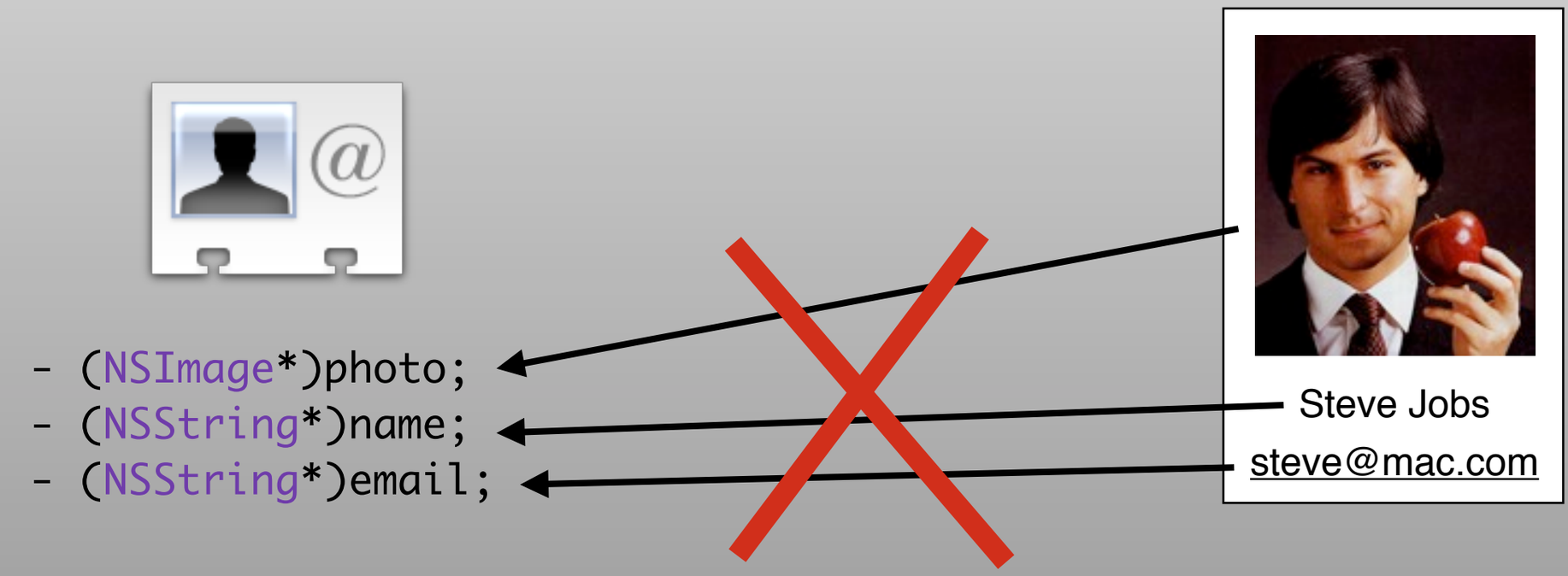
MVC Example



Model (Person)

View (Address Card)

MVC Example



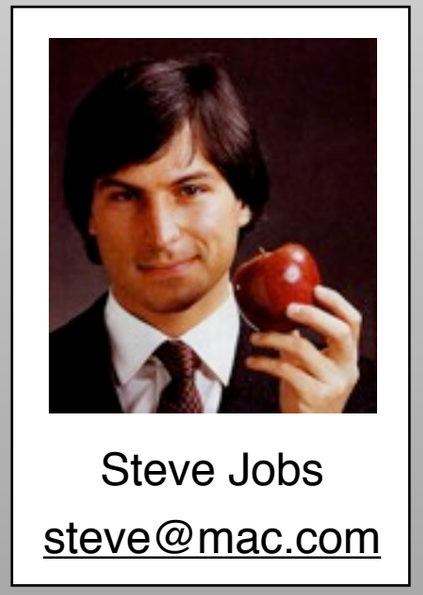
Model (Person)

View (Address Card)

MVC Example

Controller

```
- (NSImage*)photo;
- (NSString*)name;
- (NSString*)email;
```

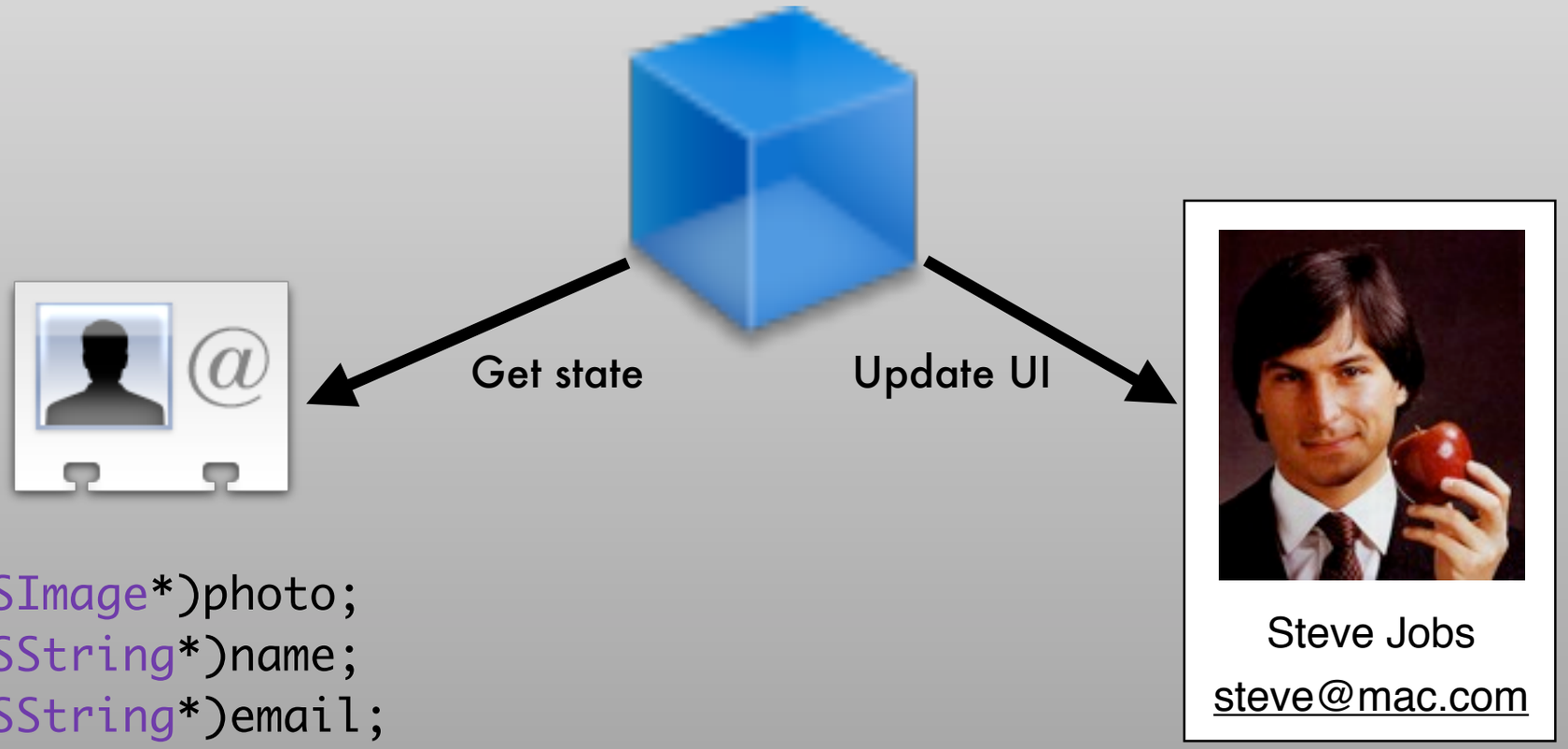


Model (Person)

View (Address Card)

MVC Example

Controller




```
- (NSImage*)photo;
- (NSString*)name;
- (NSString*)email;
```

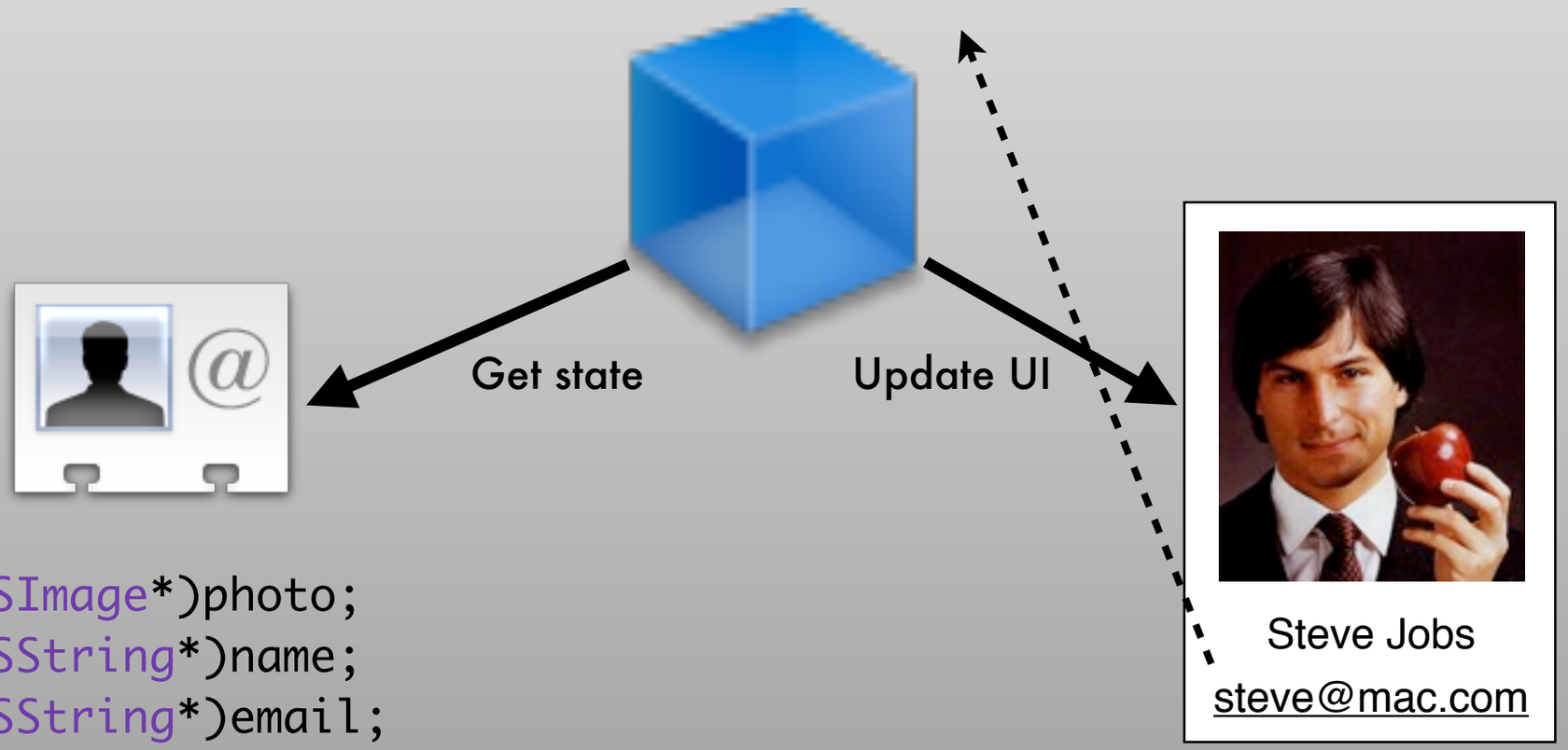
Model (Person)

View (Address Card)

MVC Example

Controller

```
- (IBAction) emailClicked: (id) sender
```




```
- (NSImage*)photo;
- (NSString*)name;
- (NSString*)email;
```

Model (Person)

View (Address Card)

Demo: Slider

NSApplication

- Manages the main *run loop* of your application:
  - Waits for *events* from the mouse, keyboard
  - *Dispatches* events to the relevant objects (of class NSResponder)
- *Owner* of MainMenu.xib

NSView

- Superclass for all views in Cocoa
- Position and size in `[view frame]`
- `[view addSubview:]` and friends used for *programmatic* layout
- Often subclassed; we'll talk about making your own views later

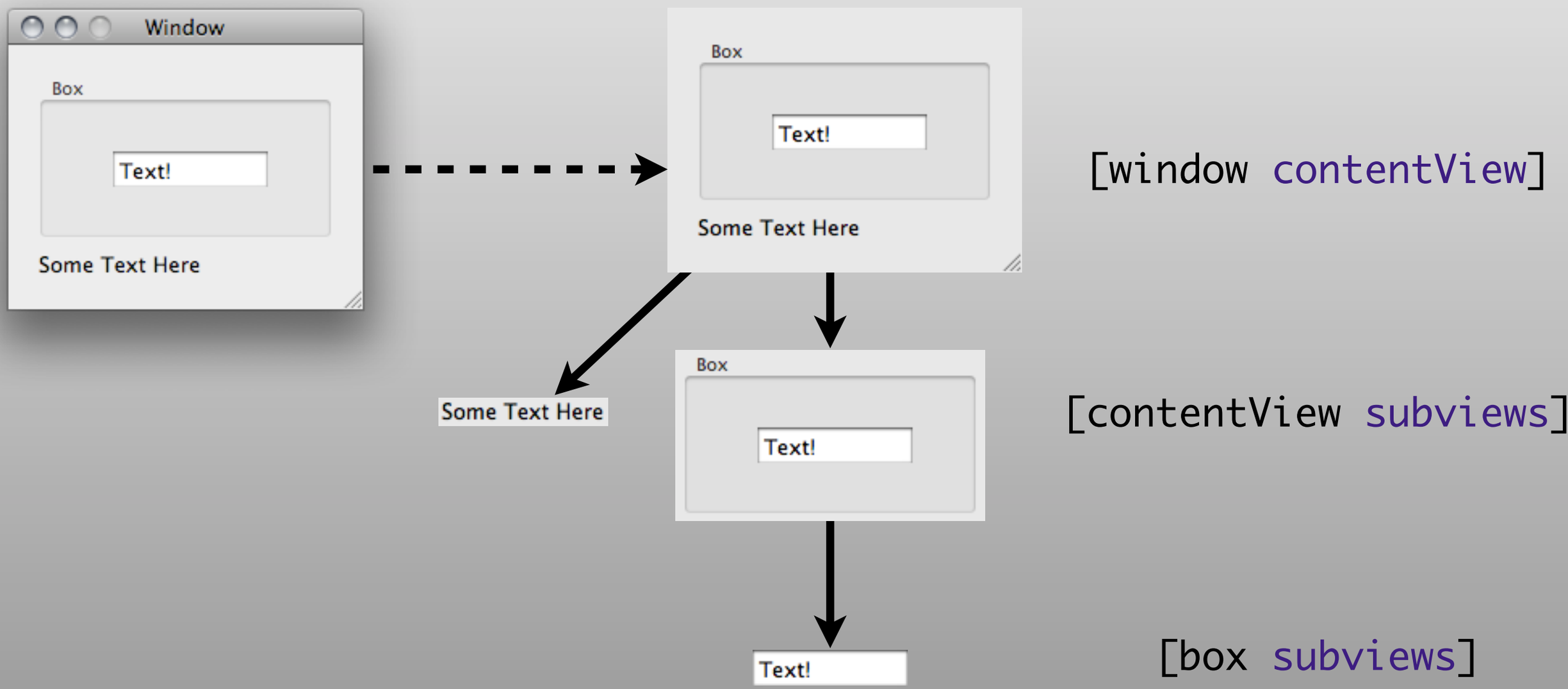
NSWindow

- Manages window frame
- Views in window are *subviews* of `[window contentView]`

The View Hierarchy

NSWindow

NSView



[window contentView]

[contentView subviews]

[box subviews]

NSControl

- Subclass of `NSView`
- Concept of value: `[control intValue]`, `[control floatValue]`, `stringValue`, and so on
- Superclass for text fields, buttons, etc.

Delegation

- Common pattern in Cocoa; avoids need to subclass in simple cases
- Objects notify their *delegates* when something happens through *delegate methods*
- Application delegate: start, terminate
- Window delegate: resize, close

Application Life Cycle

- Application starts
- MainMenu.xib loads
- Run loop:
  - Wait for event
  - Handle event
- Application terminates

Application Life Cycle

```
[delegate applicationDidFinishLaunching:notification];
```

- Application starts
- MainMenu.nib loads
- Event loop
- Application terminates

Application Life Cycle

```
[delegate applicationWillTerminate:notification];
```

- Application starts
- MainMenu.nib loads
- Event loop
- Application terminates