Getting Started with MVC in ColdFusion using FW/1

Carl Von Stetten / @cfvonner

About Me

- Geographic Information Systems (GIS) Analyst for a water resources/reclamation agency in Northern California
  - Working with ColdFusion since 2002 (CF 4.5)
  - Lots of spatial and non-spatial data modeling, data management
  - Intranet application development
- Love the outdoors (camping/hiking/paddleboarding/kayaking)
- Passionate about craft beer! (should become obvious in a few minutes)

My Tech Stack

- Tools I Use:
  - Adobe ColdFusion
  - JavaScript (incl. Bootstrap)
  - Microsoft SQL Server
  - Python, especially ArcPy (Python library for ArcGIS)
  - Esri ArcGIS Enterprise
  - Safe FME (spatial ETLs)
- Adobe Community Professional for ColdFusion *(if I re-certified yesterday)*

My Journey to MVC...

- Wrote/Maintained Procedural CFML apps 2002-2016
  - Mostly .CFM files
  - Custom tags
  - Some components (.CFCs)
- Complete overhaul of intranet map portal starting spring 2016
  - Rewrote CFML portion from scratch
  - "We're going to do it right" = Go MVC
  - Working prototype within 3 weeks

Agenda

- Prerequisites for this talk
- Brief review of procedural applications
  - Definition
  - Sample procedural code
  - Drawbacks
- Introduction to Model-View-Controller concept
  - MVC frameworks for ColdFusion
  - Sample MVC code
  - How MVC works
  - Pros/cons of MVC

What Do I Need to Know?

- CFML language (syntax, functions/tags, etc.)
- Basic understanding of components
  - How are they structured?
  - How do I use them
    - New
    - createObject()
    - / cfinvoke()
    - / cfobject()
  - ColdFusion request cycle

  - Also helpful:
    - CFScript syntax (similar to JavaScript syntax)

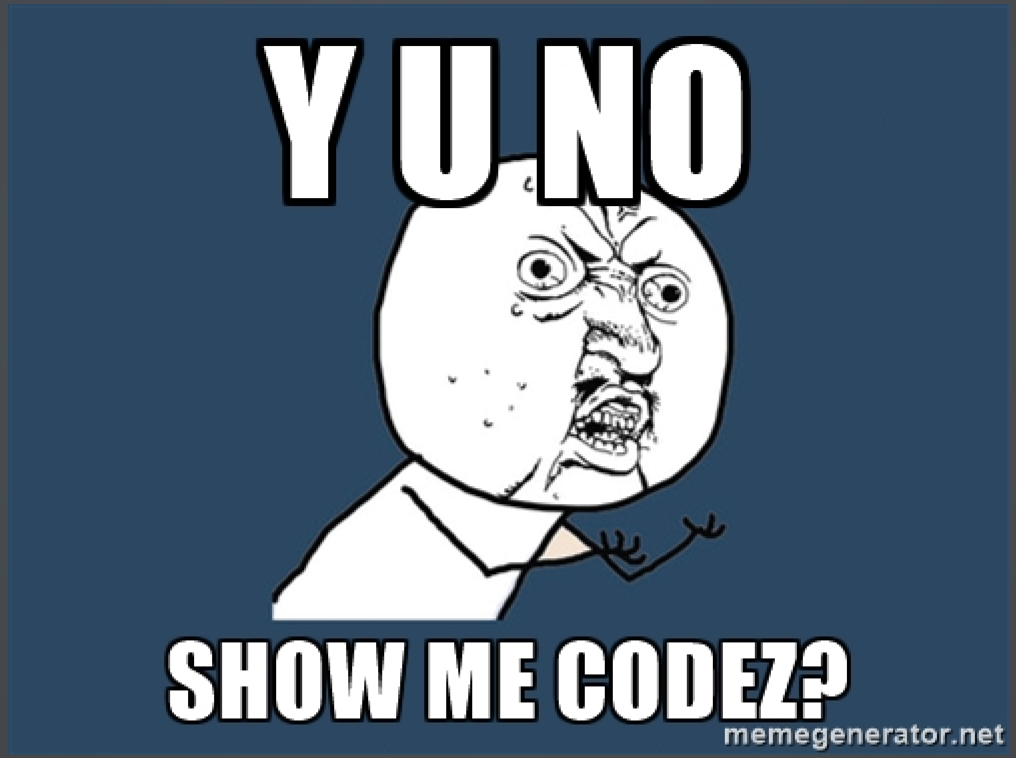
What Is a Procedural Application?

- CFM page for each URL in application
- Pages execute from top to bottom
- Typically 100's of lines of code per page
- Mix of business logic and display code
- Maybe some code reuse via <cfinclude> or custom tags

Disclaimer

> *The code you are about to see is simplified for clarity. It does not represent best practices, does not include security measures, and does not include user input validation/sanitation.--Me*

What Does a Procedural App Look Like?



Downside of Procedural Code

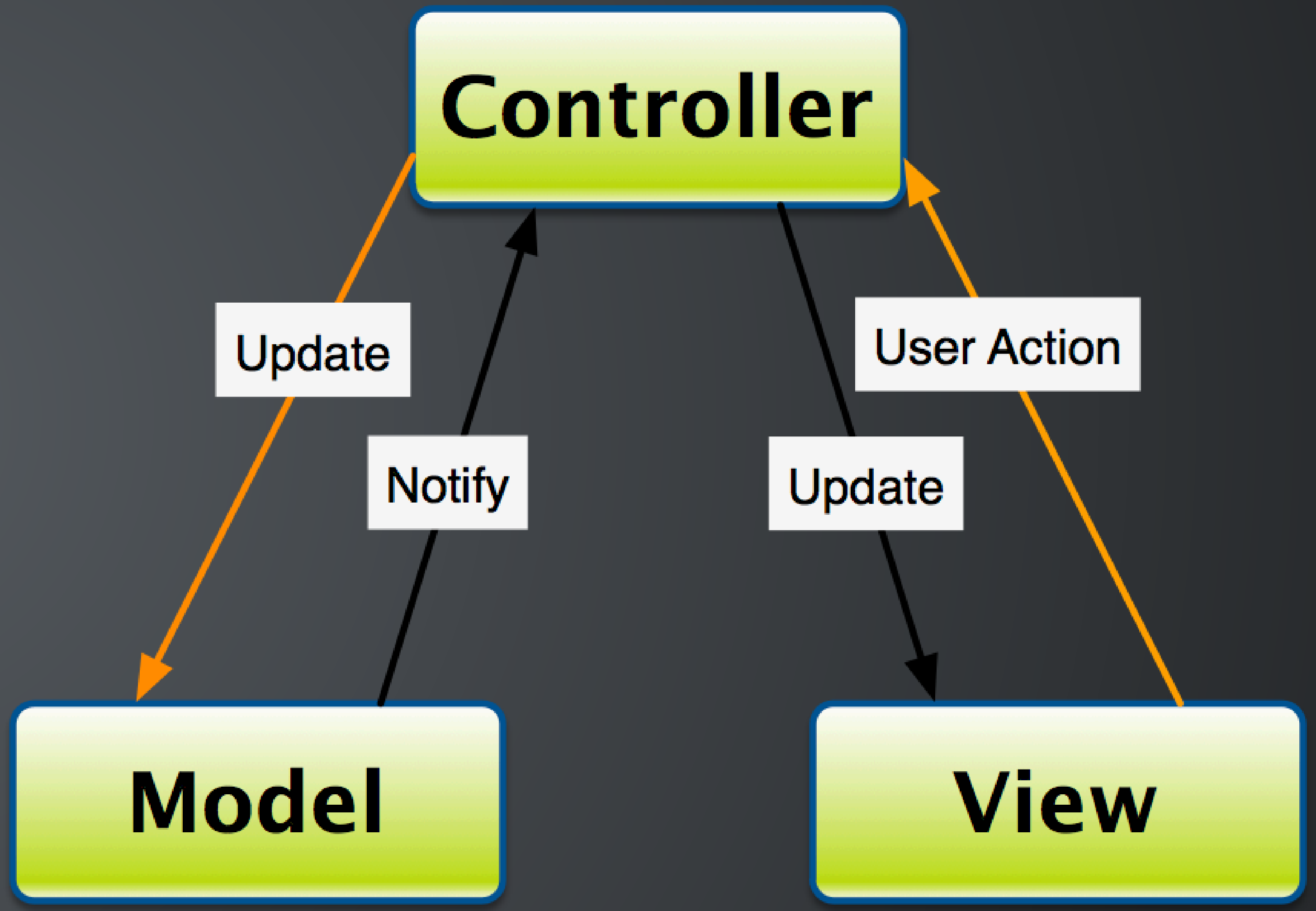
- Increasingly hard to maintain, especially as application evolves
- Hard to distribute work to teams
- Potentially lots of code duplication
  - Violates the DRY principle
- Can't build an API from the business logic
  - Intertwined with display code
- Can't perform automated unit tests on the business logic

Is There a Better Way?

- YES!
- MVC

What Is MVC?

- Model-View-Controller design pattern
- Separates concerns (business logic vs. view code)
- Common pattern in OO languages
- Usually leverage MVC framework
  - Rails (Ruby)
  - Django / Flask (Python)
  - ASP.Net MVC
  - Express / Sails (Node.js)
  - Laravel / CakePHP (PHP)
  - Spring / Struts (Java)
  - FW/1 / ColdBox (ColdFusion)



MVC in ColdFusion

- MVC can be done without a framework
  - Unless your hobby is reinventing the wheel, why would you?
- Frameworks offer lots of functionality you don't have to reinvent
  - Dependency Injection (DI)/Inversion of Control (IoC)
  - URL Routes
  - Layout/View templating
  - Data rendering (JSON, XML, text, custom)
  - Modules/Subsystems
- Frameworks often improve code organization

MVC Frameworks for ColdFusion

- Framework-One (FW/1)
- ColdBox
- Wheels (formerly CFWheels)
  - Follows Ruby on Rails paradigm
- Some older MVC frameworks still in use
  - FuseBox
  - Model-Glue
  - Mach II
  - FarCry Core

What is Framework-One (FW/1)

- Created by Sean Corfield in 2009
- Latest release is version 4.3
- Small, lightweight
    - MVC portion is one file, *one.cfc* (140KB)
    - Two related files:
        - *ioc.cfc* - (aka DI/1) Dependency Injection/Inversion of Control
        - *aop.cfc* - Aspect-Oriented Programming
- Convention-Over-Configuration

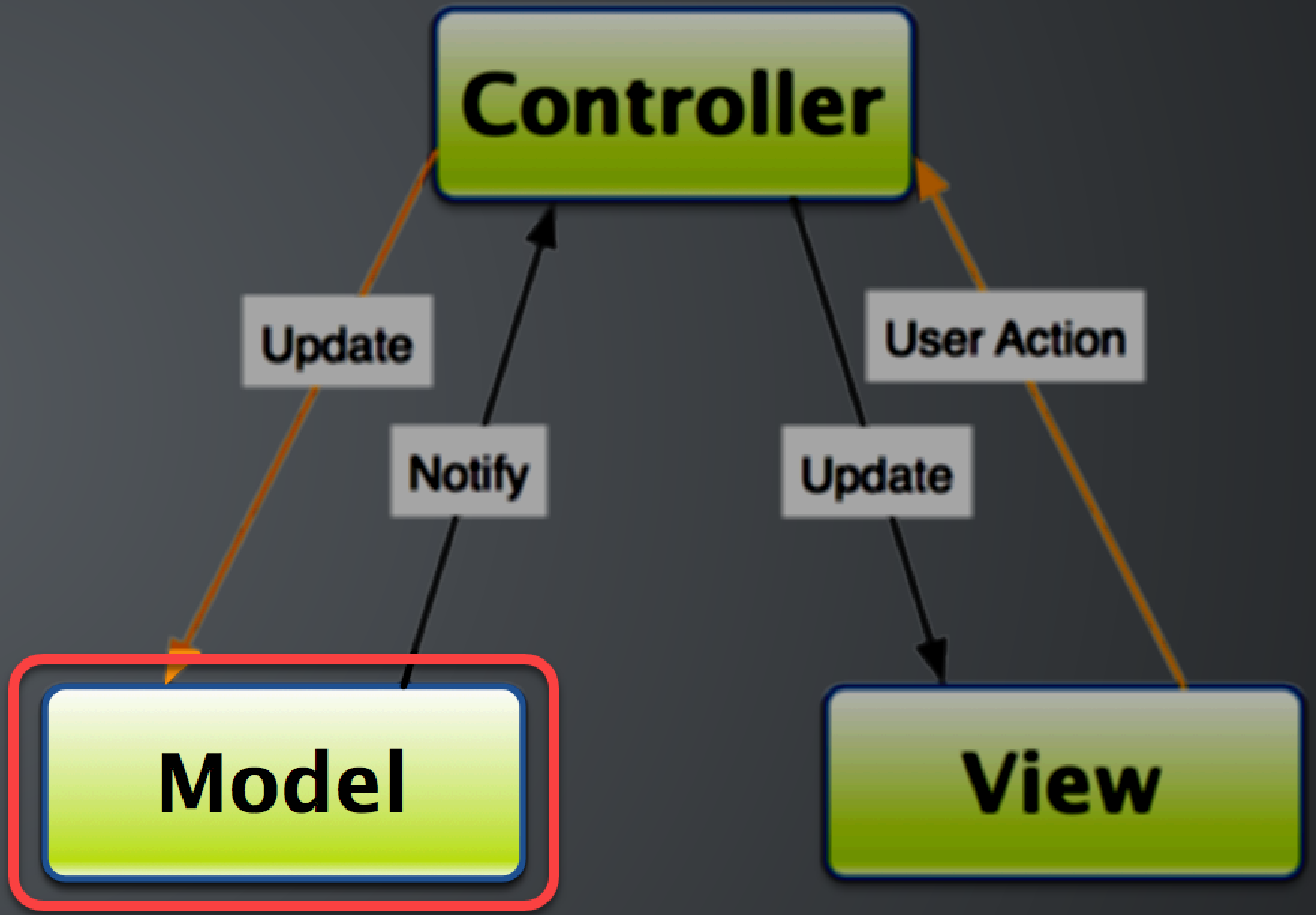
What is "Convention Over Configuration"

- Structure application folders per recommendations and the framework figures out the rest
- Only need to specify configuration settings to:
  - Override defaults
  - Deviate from the recommended structure
  - Deviate from the file naming conventions

Model Layer

- Business logic
  - Database interactions (CRUD)
  - SOAP/REST service calls
  - File system interactions
  - Helper Services
- Validation
- Responds to requests from controllers
- Should not know anything about the framework (including controllers and views)*
- Not accessible from web browser

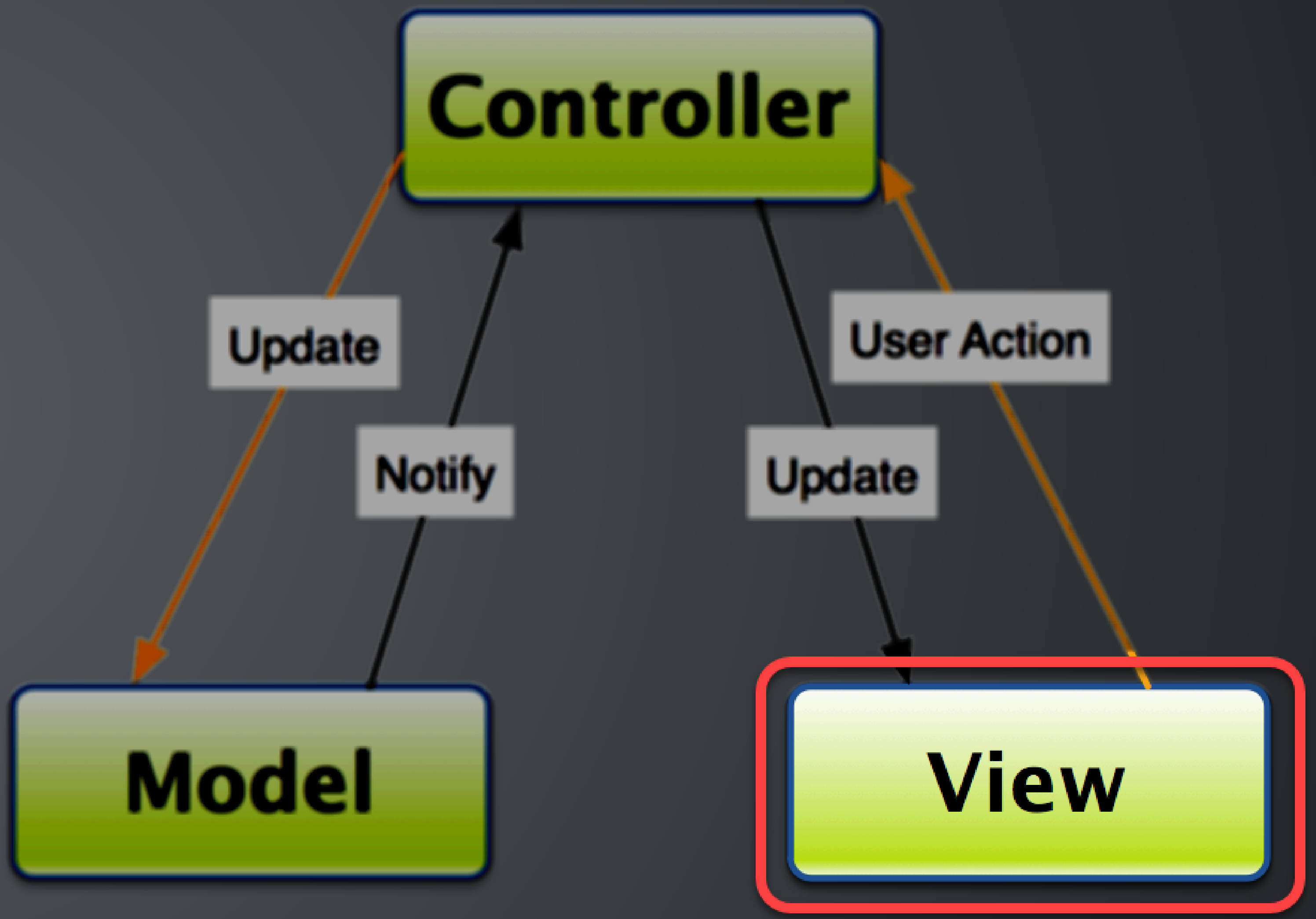
* Some exceptions: DI/IOC framework methods and helper modules loaded by DI/IOC.



View Layer

- User interface
- HTML/CSS/JavaScript
- Minimal CFML logic for display control
- NO business logic!!!
  - No SQL code here!!!
- Layouts and views

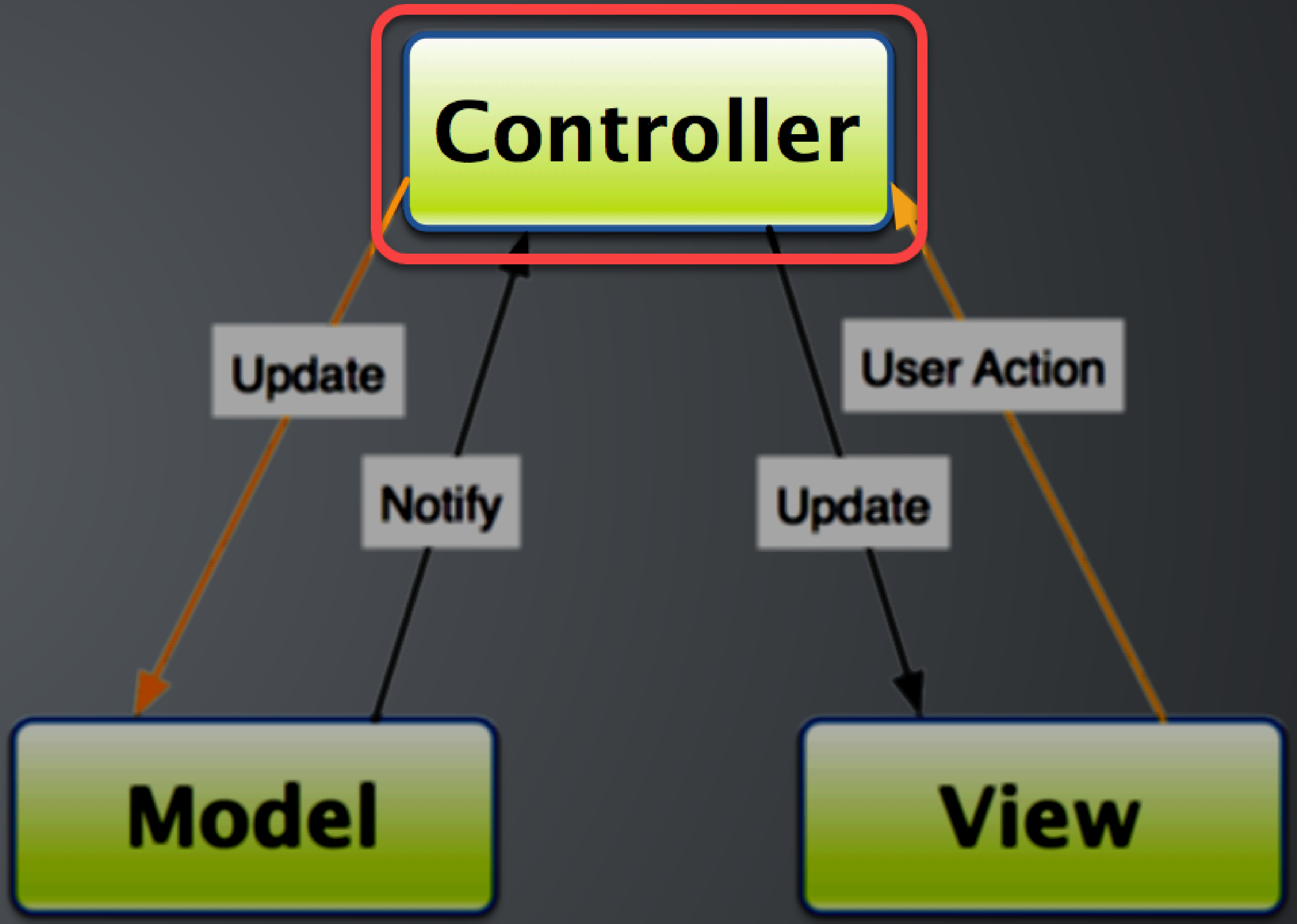
- Should not know anything about the model
- Relies on controller to put required data in "RC"



Controller Layer

- Controls the flow of the application
- Examines each incoming request (URL/FORM scope variables or path)
- Calls relevant business logic (model objects/components)
- Places results from business logic into "RC" scope for layouts/views

- Minimal validation
- Short controller functions



What the heck is "RC"?

- RC struct = alias for "request context"
- Sort of another scope (like Application, Session, Request, etc.)
- Mechanism for passing data between framework, controller, and views
- FW/1 automatically populates RC with URL and Form variables or path variables (SES URLs)
- Write controller methods to insert anything else your layouts/views need

Anatomy of a URL

https://localhost/index.cfm?action=section.item&somevariable.foo

action	• FW/1 looks for this URL parameter to decide what to do
section	• Name of controller to call (/controllers/*section*.cfc) • If a *section*.cfm file is found in /layouts, it will be applied before the default layout.
item	• Name of method (function) to call in /controllers/*section*.cfc • FW/1 will also look for this view: /views/*section*/*item*.cfm
somevariable.foo	• Any additional key/value data you want to pass in the URL (you can define this using the FW/1 `BuildLink()` function) • `somevariable` is the key, and `foo` is the value.

SES-friendly URLs

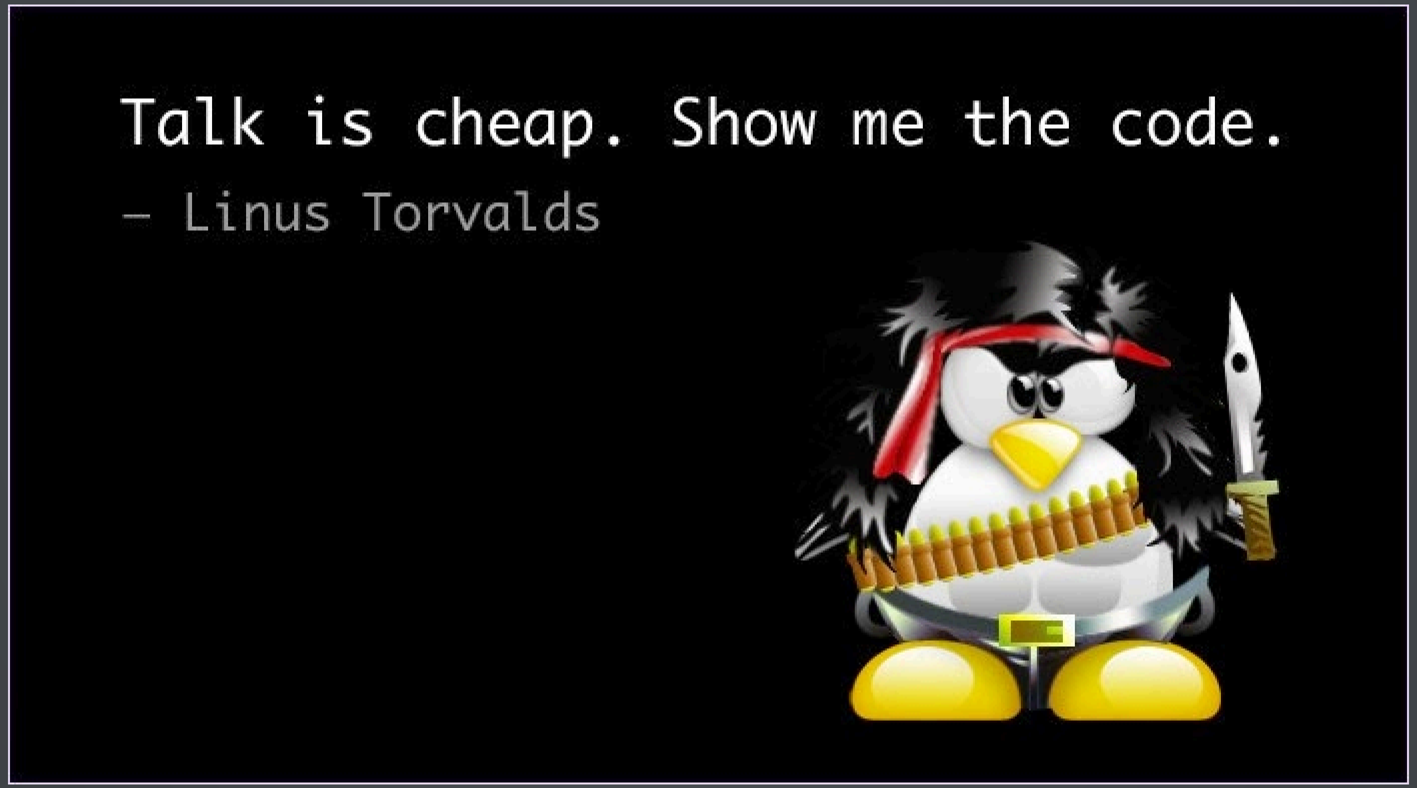
If you set FW/1's `generateSES` and `SESOmitIndex`* properties to true, you can use SES-friendly URLs like this:

https://localhost/section/item/somevariables/foo

action	• FW/1 doesn't need "action" to be explicitly stated in the URL.
section	• Name of controller to call (/controllers/*section*.cfc) • If a *section*.cfm file is found in /layouts, it will be applied before the default layout.
item	• Name of method (function) to call in /controllers/*section*.cfc • FW/1 will also look for this view: /views/*section*/*item*.cfm
somevariable/foo	• Any additional key/value data you want to pass in the URL (you can define this using the FW/1 `BuildLink()` function) • `somevariable` is the key, and `foo` is the value.

* You will need a URL rewrite rule in your web server configuration if you omit index.cfm from the URLs.

What Does a FW/1 MVC App Look Like?


Talk is cheap. Show me the code.
– Linus Torvalds

A Note About Dependency Injection (DI)

Dependency Injection simplifies managing dependencies

```
// This component depends on object "foo", which
// depends on objects "bar" and "baz"
component {
    public myObject function init( ) {
        var bar = new model.services.bar();
        var baz = new model.services.baz();
        variables.fooService = new model.services.foo( bar, baz );
    }
    // ...other functions/methods
}
```

```
// This is foo.cfc
component {
    public foo function init ( bar, baz ) {
        variables.barService = arguments.bar;
        variables.bazService = arguments.baz;
    }
    // ...other functions/methods
}
```

```
// This component depends on object "foo", which
// depends on objects "bar" and "baz"
component {
    property fooService;
    ...
}
```

```
// This is foo.cfc
component {
    property barService;
    property bazService;
    ...
}
```

MVC - The Good and The Bad

Pros

- Promotes DRY (code reuse)
- Assists team efforts
  - Different parts of model, controllers, views can be worked on simultaneously
- Common pattern/terminology
- Enforces better code organization
- Business logic is now testable (Unit Tests/TDD/BDD)

Cons

- Have to learn some new concepts, even if already an excellent procedural programmer
- Might seem to end up with more files to manage
  - But each file will be more focused and on-point

Migrating Legacy Apps

- Don't have to "eat the elephant" all at once:
  - Choose logical sections/modules of an app to migrate
  - Start new work in MVC pattern
  - Move code in phases
    - Move business logic into "fat controllers"
    - Move display code into layouts or layouts and views
    - Move business logic into the model and have "thin controllers"
- MVC code can coexist with procedural code
  - With combination of configuration settings and web server rewrite rules

Key Takeaways

- MVC isn't as hard as it seems
- It will make application code maintenance easier for the future

MVC Framework Resources

- Docmentation and Downloads
  - Framework-One
    - Docs: https://framework-one.github.io/
    - Download: https://github.com/framework-one/fw1/
    - CommandBox installation:

```
install fw1-commands // Only have to do this once
fw1 create app MyApp basic
```

  - ColdBox
    - Docs/Download: https://www.coldbox.org
    - CommandBox installation:

```
coldbox create app myApp
```

- Discussion Groups
  - Framework-One Google Group
  - ColdBox Community
- Real-Time Assistance
  - CFML Slack Team
    - FW/1 Channel
    - Box-Products Channel

- FW/1 project has been forked to CFMVC!
- New project stewards: Mark Takata, Brian Sappey, Ben Nadel
- Other CF community members will be contributors, along with some Adobe CF engineers
- Potential improvements to the developer experience
- Stay tuned - more information coming soon

Thank You!!!

- Contact Me:
  - Email: carl.vonstetten1@gmail.com
  - X (formerly Twitter): @cfvonner
  - GitHub: @cfvonner
  - CFML Team on Slack: @cfvonner

- Code/Slides: https://github.com/cfvonner/Intro-To-MVC-ColdFusion