Separating Your Application from Rails

No Hexagons Required

Brian V. Hughes originate.com @brianvhughes

Why separate?

Why separate?

- Rails MVC encourages tight coupling
- Tight coupling inhibits long-term maintenance
- Tight coupling inhibits efficient TDD
- "Standard" Rails Controllers obliterate SRP

"Standard" Controller Responsibilities

- Receive HTTP request
- Authentication
- Authorization
- Parameter handling
- Coordinate models/ associations
- Control Persistence
- Perform business logic
- Call external APIs
- Render response data
- Return HTTP response

"Standard" Controller Responsibilities

- Receive HTTP request
- Authentication
- Authorization
- Parameter handling
- Coordinate models/ associations
- Control Persistence
- Perform business logic
- Call external APIs
- Render response data
- Return HTTP response

"Reduced" Controller Responsibilities

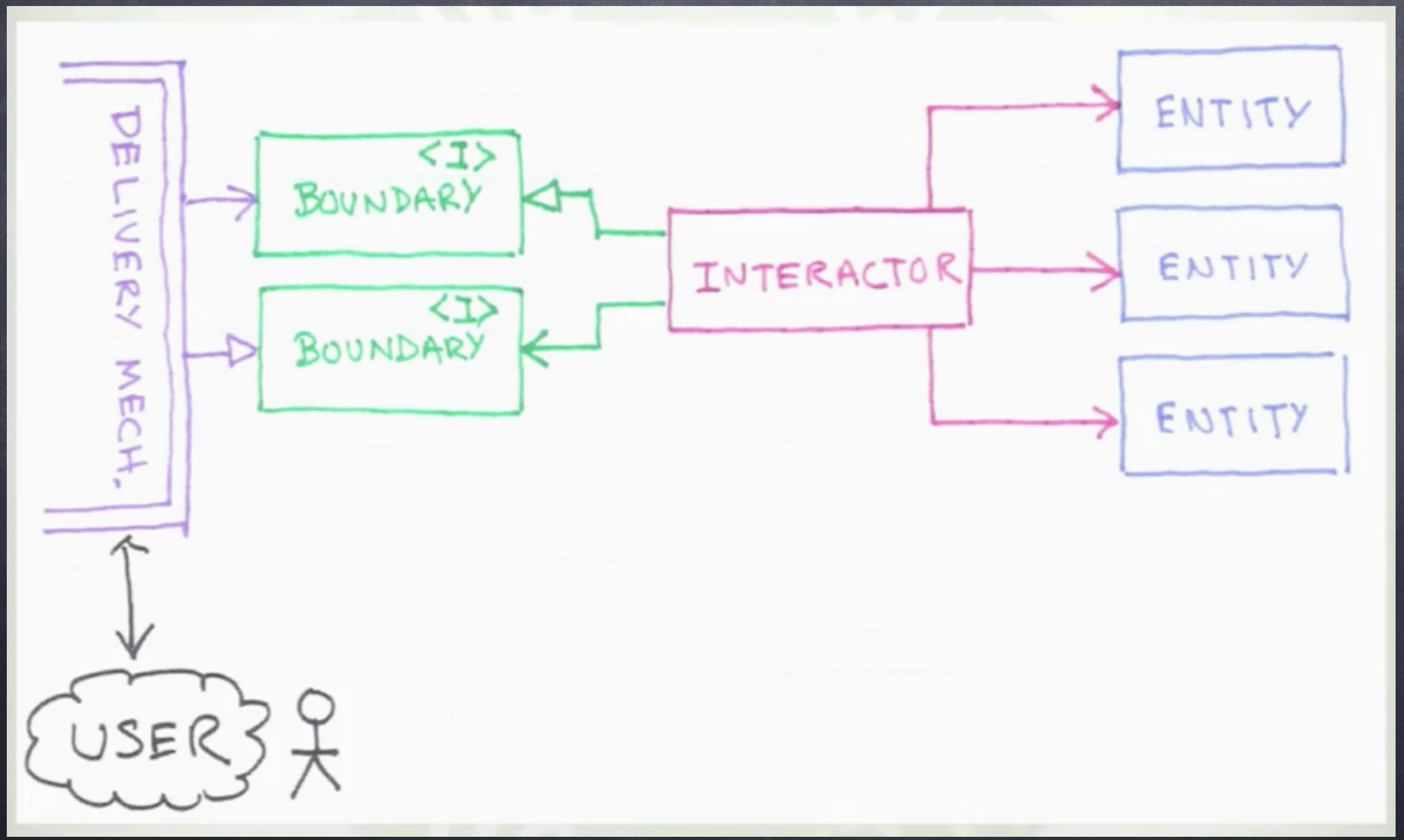
- Receive HTTP request
- Authentication
- Tell Application to perform work
- Render response data
- Return HTTP response

What does a "separated" application look like?

Entities and Interactors

Entities and Interactors

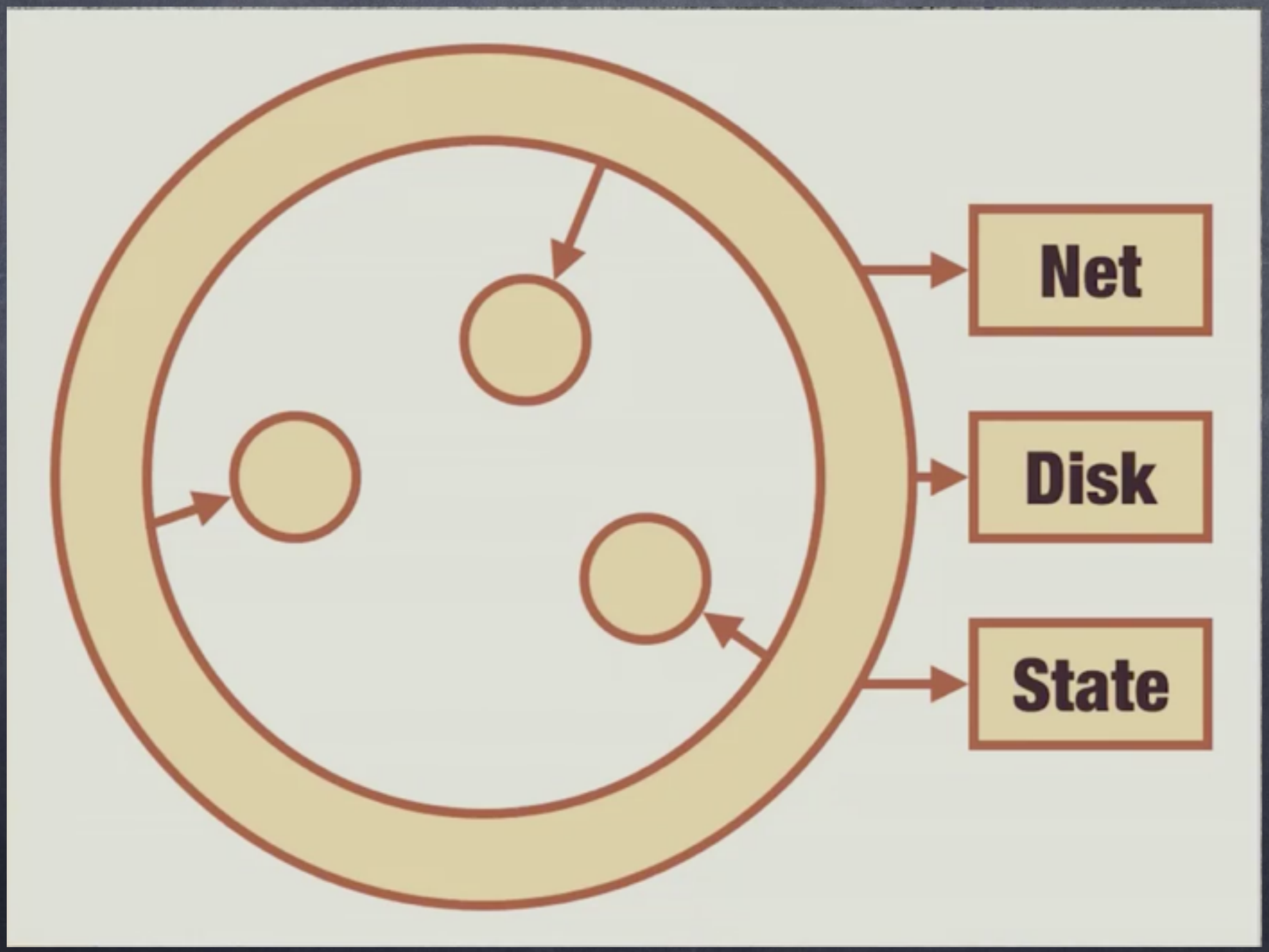
Architecture, the Lost Years, Bob Martin, Ruby Midwest '11
https://www.youtube.com/watch?v=WpkDN78P884



Functional Core, Imperative Shell

Functional Core, Imperative Shell

Boundaries, Gary Bernhardt, RubyConf '12
https://www.youtube.com/watch?v=yTkzNHF6rMs

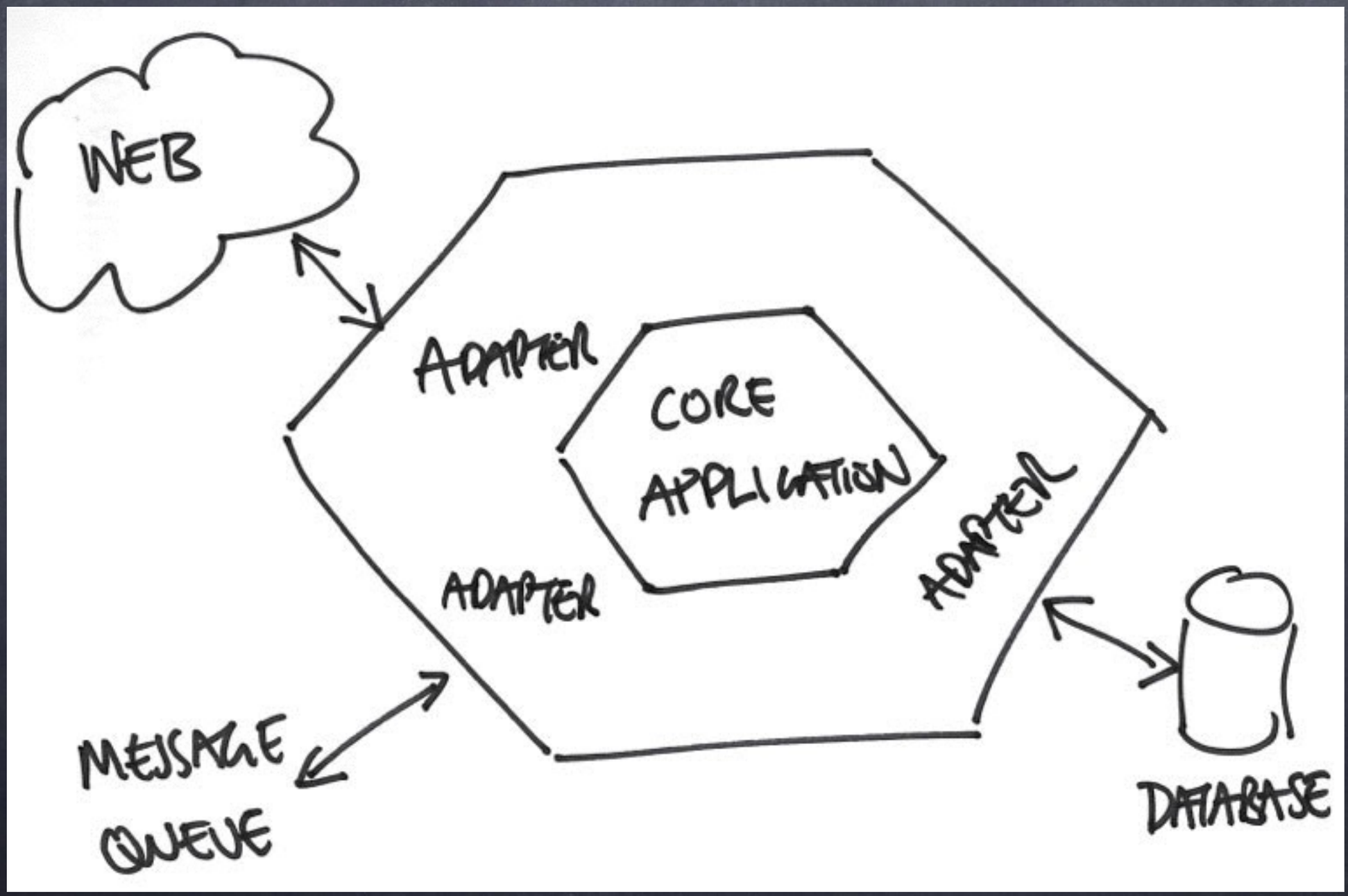


Hexagonal Architecture

Brian V. Hughes originate.com @brianvhughes

Hexagonal Architecture

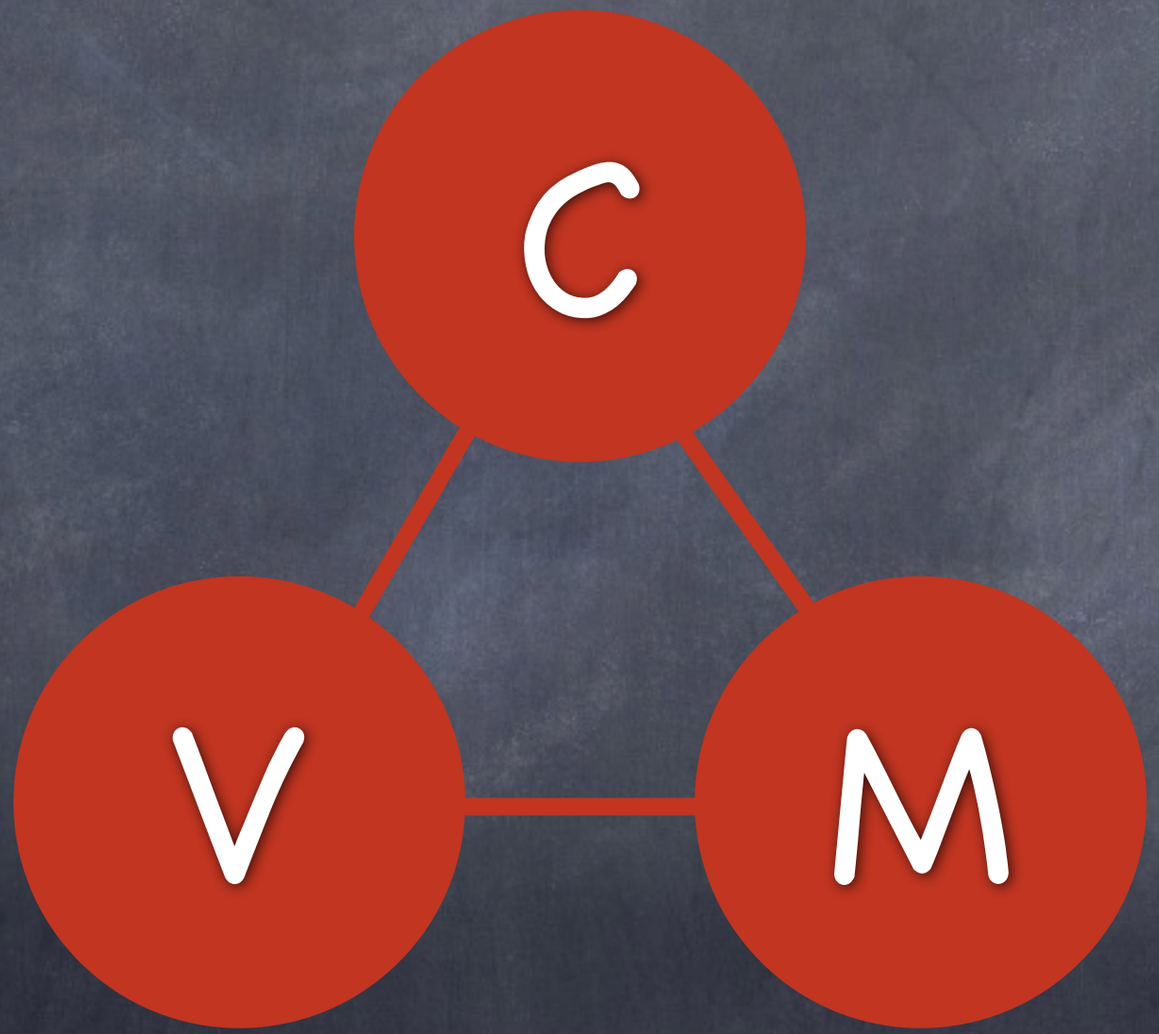
Presented by Matt Wynn at GORUCO '12
https://www.youtube.com/watch?v=CGN4RFkhH2M



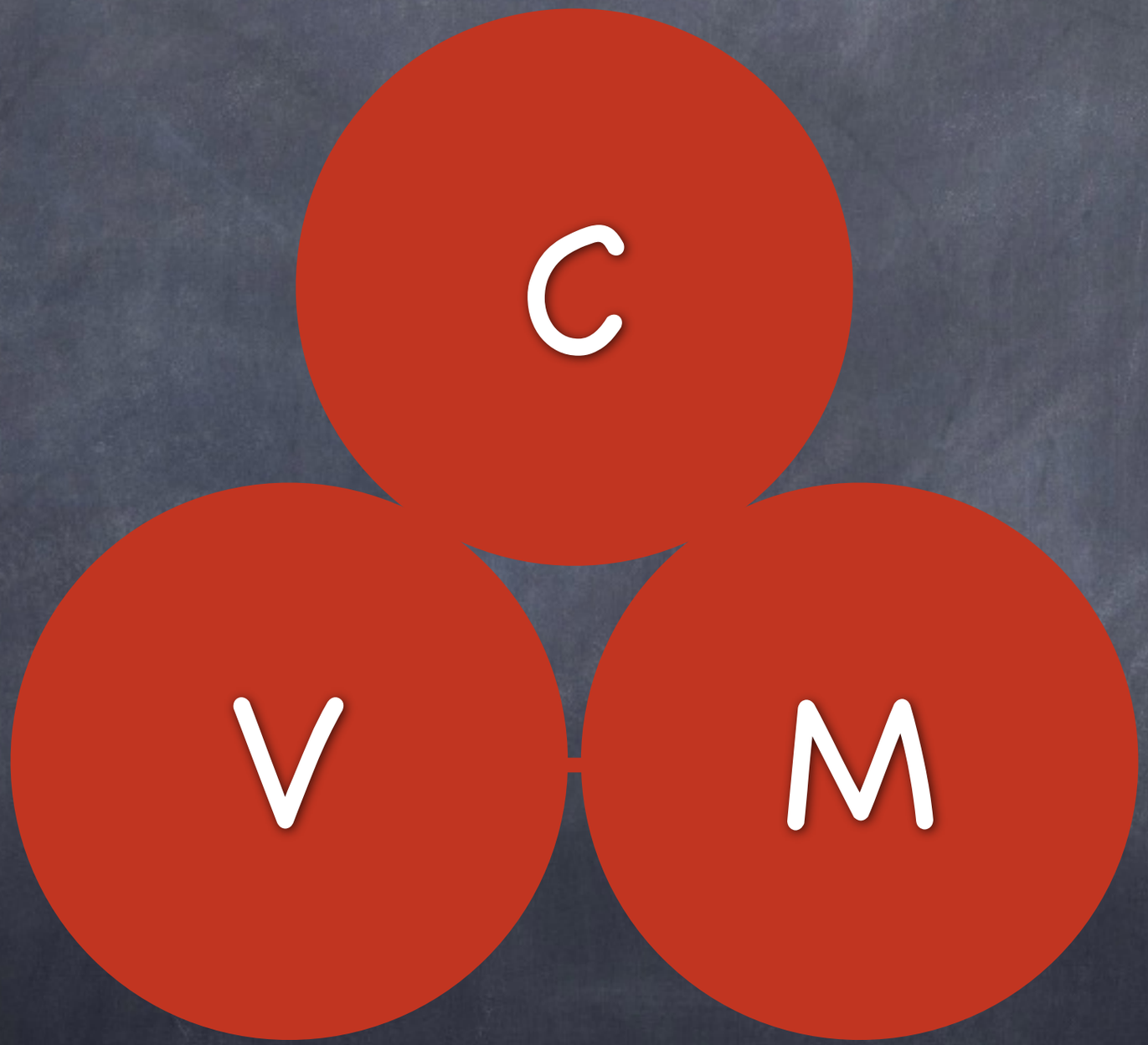
Response State

Response State

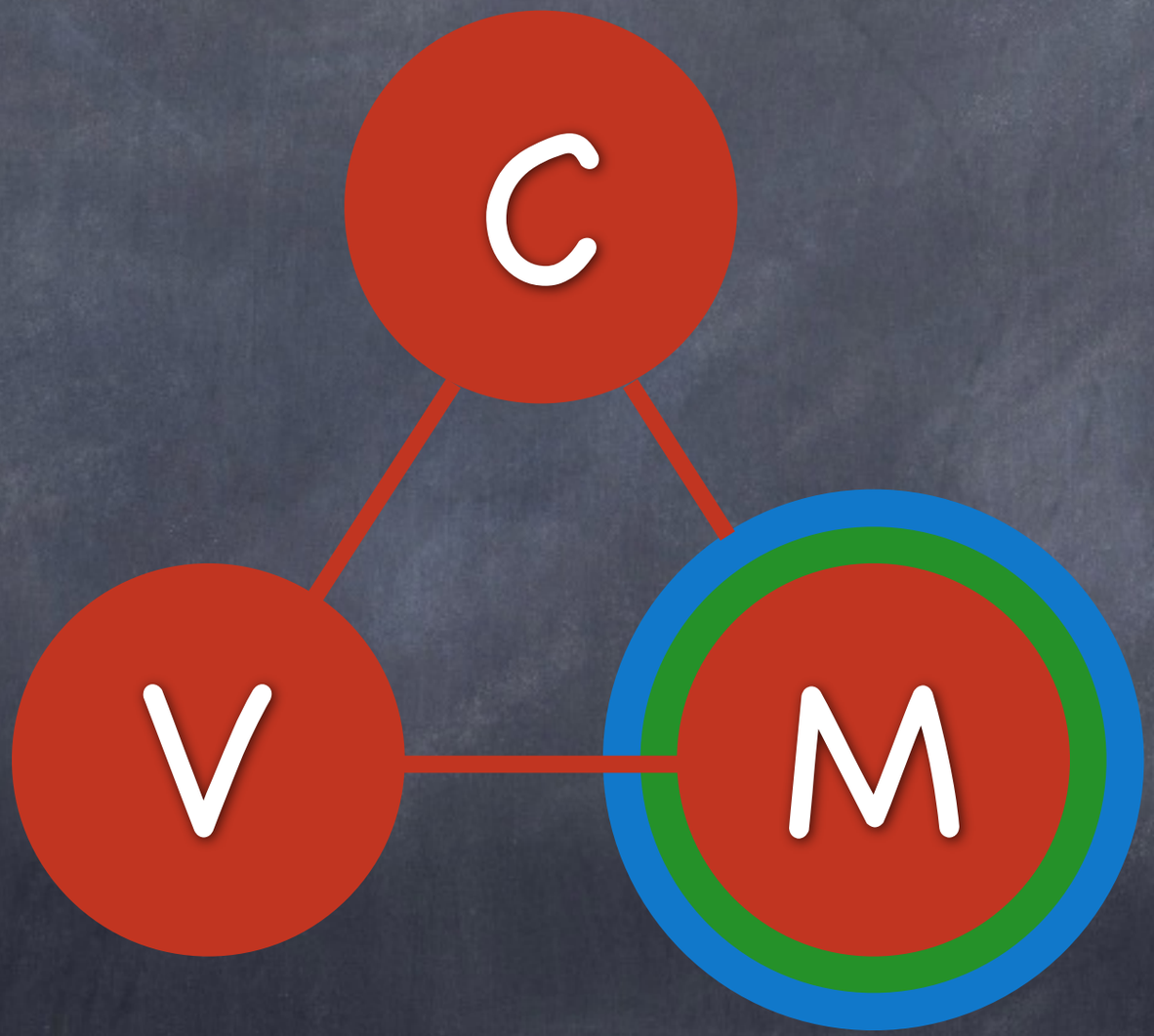
Presented by Brian V. Hughes, LVRUG 12/10/14



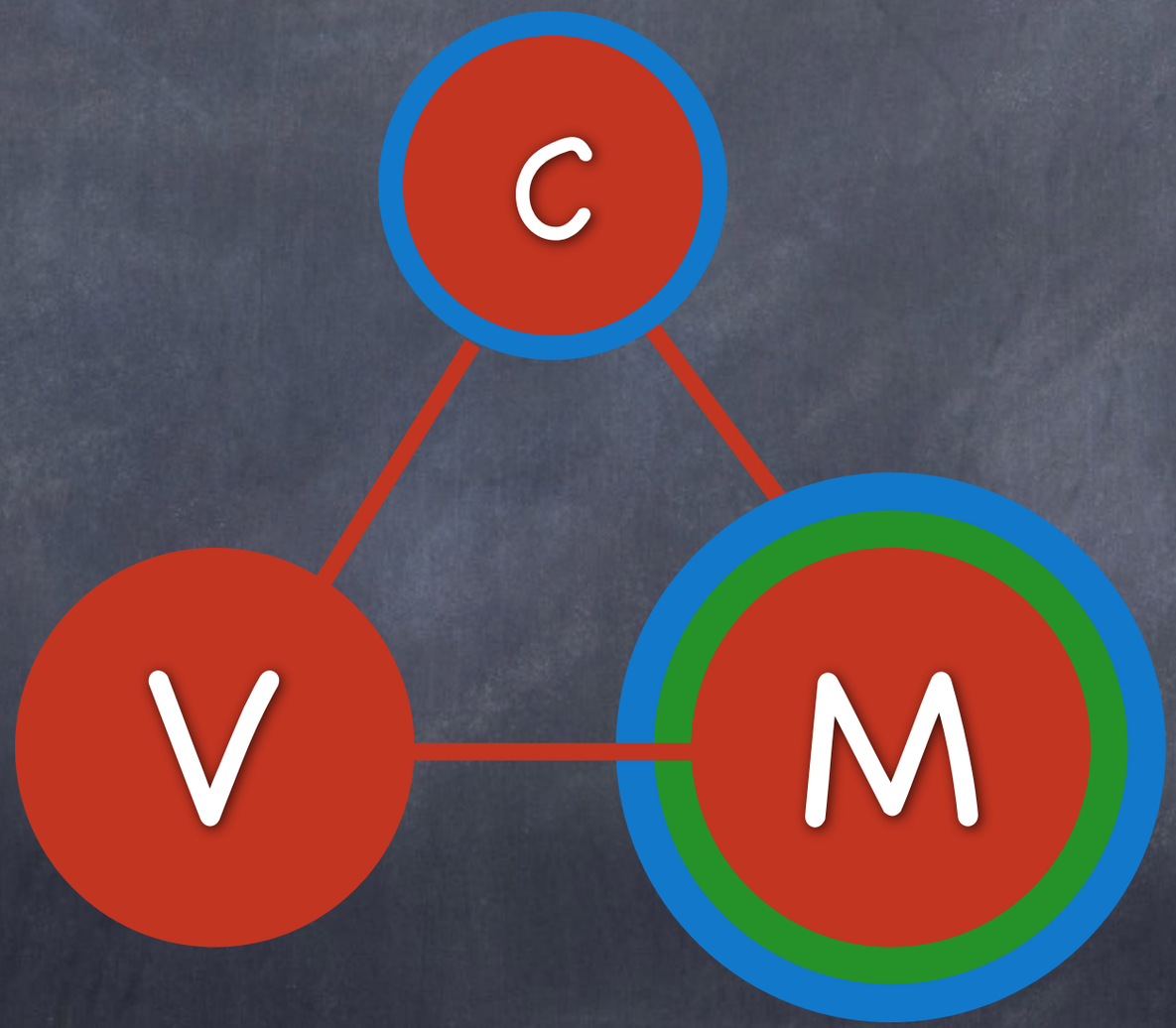
Response State



Response State



Response State



Response State

- A new interface in controller actions, aka. Response State Pattern
- Response State service classes, that provide the new controller interface
- Response Object instances, passed from the service class, to the controller. Drives the pattern.

Response State

The so-called "Design Pattern"

- Replace the body of a controller action with a call to a Response State service class.
- The class "responds", via yield, with an object representing the new app state
- Action doesn't inspect response object, but sends directed calls, to named states.
- Only 1 state can be active; that's the path the controller action follows.

Response State

The Service Class

- Response State service class exposes only a :call method, as its public API.
- Ideally, that :call method returns nil, in the process of yielding a Response Object
- Service class responsible for defining the allowed states of the Response Object.
- Handles all high-level application logic, delegates to services and models.

Response State

The Response Object

- Response Object always instantiated with 3 parameters: :state, :message, :context.
- The value of :state determines which methods Response Object responds to.
- When called with the current :state method value, it yields, otherwise returns nil.
- :message and :context can both be nil.

Response State

Presented by Brian V. Hughes, Originate `14
Ruby Gem developed by Alex Peachey
https://github.com/Originate/response_state