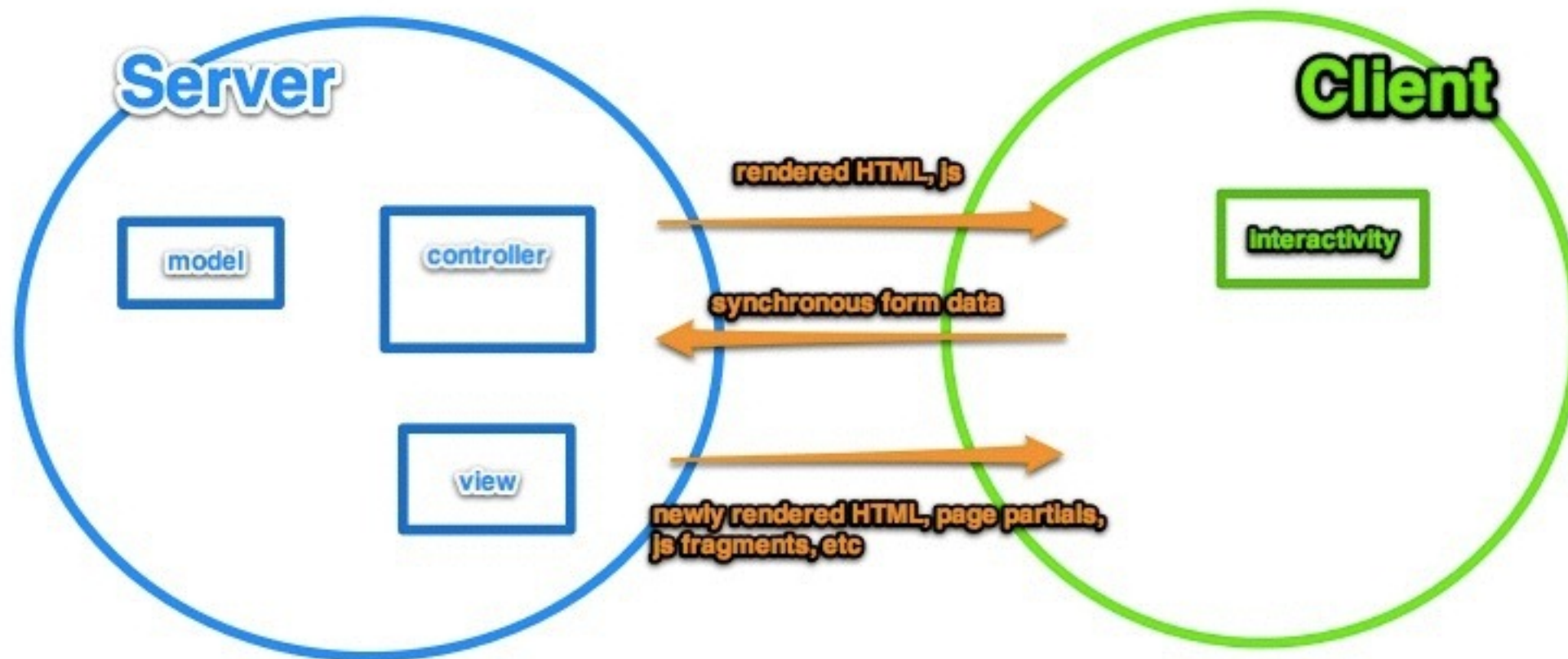




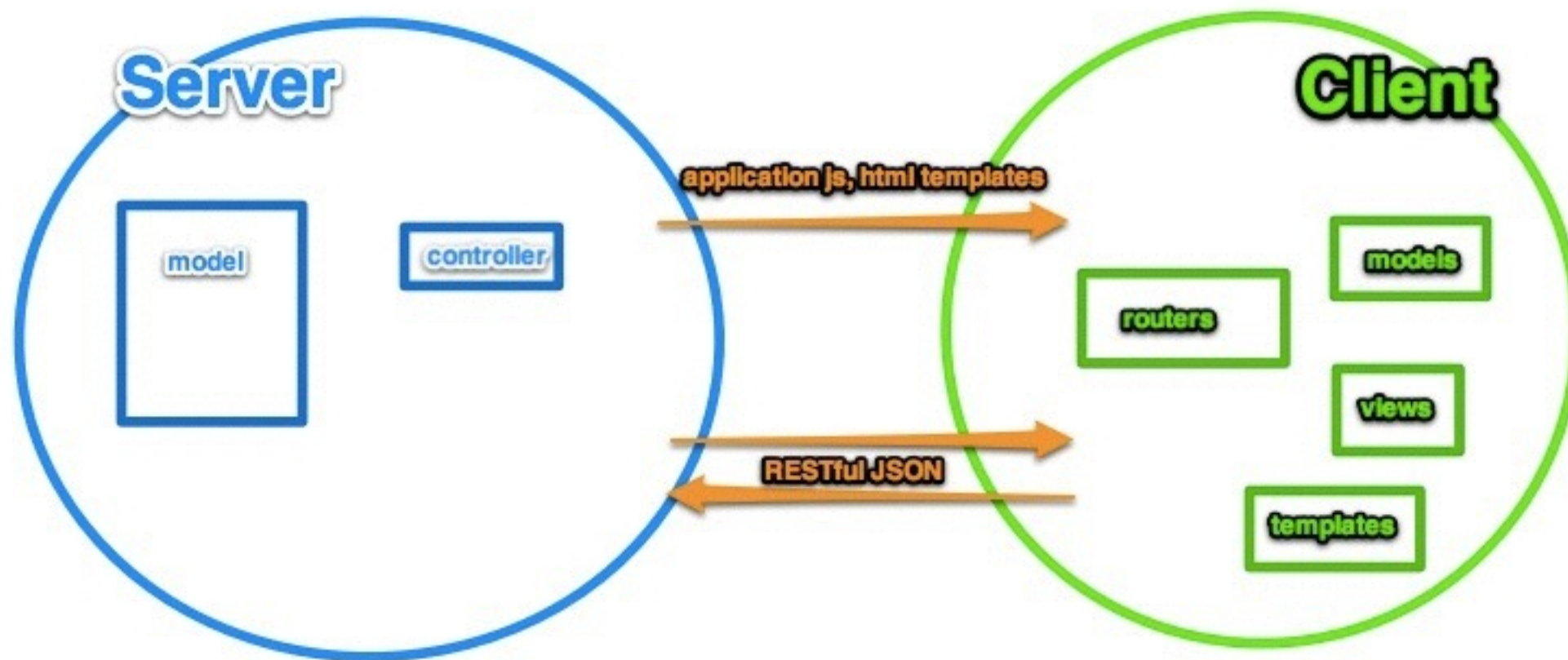
Welcome to CampNG!

Hi!

The old



The new



Angular Zen

Simplicity

Decouple all the things

Binding

Dependency Injection

Testing

HTML compiler

Angular expressions

{{ stuff }}

Our first angular app

You can make one, too!

- Make your page an angular app
- Add a very simple angular expression

Controllers

ng-controller

Manage \$scope

Adds models

Models = POJSOs

Responds to UI actions

Should be thin(ish)

~~Manipulates the DOM~~

Let's see a controller

You try it!

- Have your page say: “Hi, my name is _ and I am _ years old”
- Need 2 expressions and object(s) on scope
- Bonus point for single object on scope

View

POHTML*

Directives

- Custom attributes `<div thing="wut">`
- Custom element `<thing></thing>`
- CSS class `<div class="thing">`

ng-repeat

ng-repeat="item in items"

ng-repeat="(key, value) in object"

Show me some show
me some repeating!

track by expression

You try!

- Instead of just your name and age, display a list of students with names and ages

ng-model

- Establish a two-way binding between model and view
- Changes from either side are immediately reflected

Show me some model

You should model

- Edit the names of one or all of the students
- Feel free to do age too!

ng-click

- Binds to a function on scope
- Can pass things on scope as params

Show me the clicking!

You can click too

- Create a new student
- Enter name and age
- Add it to the list

Built-in directives

- input, textarea, select
- ng-app
- ng-blur
- ng-change
- ng-checked
- ng-class
- ng-click
- ng-cloak
- ng-controller
- ng-disabled
- ng-show, ng-hide
- ng-model
- ng-repeat

So many!

But wait, there's more...

More about \$scope

- Can be nested
 - nested scopes inherit prototypically
- All inherit from \$rootScope
- `angular.element("foo").scope()`

Beware of shadowing!

Let's see why!

Modules

- An app or package
- Contain:
 - controllers
 - services
 - filters
 - directives
 - configuration

Modules

- create:
 - `angular.module("mod", [deps])`
- reference:
 - `angular.module("mod")`

Show me the module!

Let's module!

- Put your app in it's own module
- Call controller on that module to make your controller

We got both kinds of injection

- By argument name
- Inline (array syntax)
- Let's see some code

Dependencies

- You can define your own!
- Best way to share between things (controllers, directives, etc) in your app
- Singletons

Types of Dependencies

- Factory
 - function where return value is injected
- Service
 - returns a function called with new
- Provider
- value
- constant

Meet Fruit Factory

Make Student Factory

- Pull your students out of your controller into a factory
- For bonus points, give your factory a method to access a student by name

Service and Provider

- Service is like a factory, but calls defined function as constructor
- Provider: lowest level way to define a dependency
 - can be injected into config

Examples!

ng-router

- Broken out from angular core in 1.2
- Client side routing for SPAs
- Need to add script and depend on ngRoute module

Routing

- `$routeProvider`
 - Used in a config block to define routes
- `$routeParams`
 - Can be injected in controller to access params
- `$location`
 - manually trigger route change with `$location.path()`

Defining a route

- `$routeProvider.when("route", options)`
- everything after “#”
- can have params “/things/:id/:name”
 - these become properties of `$routeParams`
- `$routeProvider.otherwise({redirectTo: “/route”});`

Route options

- controller
 - Invoked when the route is triggered
- template
 - String of markup
- templateUrl
 - External url to load template
 - script of type “text/ng-template” with an id

\$routeParams

- injectable into controller
- properties for each param

Fruit routes

You can route too!

- Make students clickable
- Display their details when you click 'em
- Be sure to add angular-route.js
 - Match versions!
- Also need module to depend on ngRoute

ui-router

Meet our recipe app

We got both kinds of
testing

unit-testing with Jasmine

angular-mocks.js

Lab #1

End-to-end testing

Lab #2

- Make the scenario spec pass

Lab #3

- Move recipes to a service
- Make the jasmine spec pass

Lab #4

- Make scenario spec pass
- You'll need some routing
- Use your shiny new service in the controller
- Make sure links work too

Forms

- Don't submit by default
- bind their values using ng-model
- form and inputs are available on scope by name

Form state

- `<form name="foo">`
 - `$scope.foo` is the form NOT the model
- `<input name="bar">`
 - `$scope.foo.bar`

State properties

- `$dirty`
- `$pristine`
- `$invalid`
- `$error`
- Also available as CSS classes

Fruit form

Lab #6

- Add edit route
- Add edit controller
- Wire up the edit form
- Make the scenario spec pass

```
$location.path("/foos")
```

Lab #7

- “saving” a recipe will just transition to show recipe
- use \$location
- Make the scenario pass

Bonus!

- Edit a copy to prevent immediate updating
- lab7_edit_copy

Form validation

- All inputs:
 - ng-required
 - ng-minlength
 - ng-maxlength
 - ng-pattern
- Number:
 - min, max

Moar validation

- magic properties
 - \$valid
 - \$invalid
 - \$error
 - keys for each validation
- on form and inputs

Lab #8

- Make title capitalization mandatory
- ng-pattern is your friend
- make the scenario pass

Lab 9

- Make the create recipes scenario pass
- Add a create method to Recipe to create a new recipe with an id
 - With jasmine spec, please!
- Add a new route
- Add new controller

selects in angular

- select decorates HTML select tag
- ng-options for building options

ng-options syntaxes

- label for value in array
 - value is what gets bound to ng-model
- selected as label for value in array
 - selected is what is bound to ng-model

select example

Talking to the server

- \$http
 - get(url)
 - post(url, data)
 - put(url, data)
 - delete(url)
- returns a promise
 - success(func)
 - error(func)
 - then(successFunc, errorFunc)

Let's see some httping

\$scope.\$apply

- kicks off the event loop
- Makes sure changes have propagated
- useful in tests
- Or non-angular aware code

\$httpBackEnd

- Given to us by angular mocks
- Simulating requests in unit tests
- whenGET, whenPOST, etc
- expectGET, expectPOST, etc
- returns an object with respond method

Lab #10

- Implement an ingredient factory to make the jasmine spec pass
- Implement add ingredient on show recipe to make scenarios pass

\$resource

- For talking to RESTful resources
- `$resource("path", options)`
 - returns a class for the resource
 - `get` and `query` on class
 - `$save`, `$remove`, `$delete` method on instances

LetsGitLunch

Filters

- Transform an expression
- Are called like “foo | bar”
 - bar is a filter which transforms foo
- Can take parameters “foo | bar:baz”
 - bar is the filter, baz is the param

Built-in filters

- date
- currency
- json
- lowercase, uppercase
- orderBy
- number
- limitTo

The filter filter

- transforms a collection by filtering
- param is what to filter by
- strings that match or objects where any property matches
- eg stuff | filter:search

Fruit filtering

Lab #10a

- Make scenario spec pass
- Add a search box and filter recipes

Creating filters

- `module.filter("name", function() {...})`
- need to return filter function
- See example!

Lab #11

- Make spec pass by creating a markdown filter
- Use ng-bind-html to add description to show page

Creating directives

The ~~easy~~ deprecated way

- `module.directive("fooBar", function() {...})`
- Return a function(scope, element, attrs)
- element is jQuery wrapped if you jQuery
 - handy for wrapping plugins

version directive

The ~~hard~~ recommended way

- return a directive specification object
 - specifies options to specify how directive works
 - there are many

The highlights

- restrict:
 - E for element
 - A for attribute
 - C for class
 - M for comment (no one uses that)

link

- a function(scope, element, attrs)
- Earlier directive form is a simplified syntax

Other things

- template
- templateUrl
- replace
- must have a single root element

Directive scope

- true = new scope
- {} = “isolate” scope
 - Maps attributes on directive to scope
 - @ maps attribute
 - = bind to parent scope
 - & pass in an expression

\$scope.\$watch

- Two args
 - An expression to watch
 - A function to execute on change
 - receives newval, oldval as params

upperCase directive

Directive controllers

- Specified with controller attribute
- Can be name or function

Directive with
controller

Transclusion

- The inclusion of one thing in something else
- `transclude: true`
- Allows directive to wrap arbitrary angular template content
- Use `ng-transclude` to specify where the original body content goes

Transclusion example

Lab #12

- Create a directive to live preview markdown
- Make the jasmine spec pass
- Add it to recipe edit view

Bonus material

- animations
- ui-router