A Target Always Moving

New Rails Features

and how to use them

NOTCH8

Insert Self Deprecation Slide Here

Who is This Clown?

Rob Kaufman
@orangewolf
http://spkr8.com/s/7218

Founder of Notch8 - An App Development Consultancy since 2007

NOTCH8

It's the high level

Part 1 - Overview

NOTCH8

Timeline

- Rails 4.2 December 20, 2014
  - Active Job, ActionMailer #deliver_later
- Rails 5.0 June 30, 2016
  - Action Cable, Turbolinks 5, Attributes API, ApplicationRecord
- Rails 5.1 April 27, 2017
  - Webpacker

NOTCH8

ActiveJob

- A uniform instance for background work
- This is code the runs outside the typical request, response system
- Can be backed by many different background runners including: DelayedJob, Resque, Sidekiq Built in queue

ApplicationRecord

- Adds a parent object that all models inherit from
- Makes AR uniform with ApplicationControllers
- Applies to ActionMailer and ActiveJob as well

NOTCH8

Attributes API

- gives types to attr_accessor or AR attribute objects
- you can add your own custom types

NOTCH8

Webpacker in Rails

- Brings in two new tools to the asset pipeline flow
- The first is webpack a pre-compiler and packager
- The second is yarn, which is built on top of NPM for package management, similar to how Bundler works on gem files

ActionCable

- Websockets in Rails
- Can create realtime updating events and access them both on the server and on the client side
- Uses PubSub for clients

NOTCH8

It's almost over!

Questions?

Rob Kaufman
rob@notch8.com
http://spkr8.com/s/7218
@orangewolf

Matt Clark
matt@notch8.com
@winescout

NOTCH8

Let's get to the details

Part 2 - Backend

NOTCH8

ActiveJob

- A uniform instance for background work
- This is code the runs outside the typical request, response system
- Can be backed by many different background runners including: DelayedJob, Resque, Sidekiq Built in queue

ActiveJob

Adapters						
	Async	Queue	Delayed	Priorities	Timeout	Retries
Backburner	Yes	Yes	Yes	Yes	Job	Global
Delayed Job	Yes	Yes	Yes	Job	Global	Global
Qu	Yes	Yes	No	No	No	Global
Que	Yes	Yes	Yes	Job	No	Job
queue_classic	Yes	Yes	Yes*	Yes	No	No
Resque	Yes	Yes	Yes (gem)	Queue	Global	Yes
Sidekiq	Yes	Yes	Yes	Queue	No	Job
Sneakers	Yes	Yes	No	Queue	Queue	No
Sucker Punch	Yes	Yes	Yes	No	No	No
Active Job Async	Yes	Yes	Yes	No	No	No
Active Job Inline	No	Yes	N/A	N/A	N/A	N/A

http://api.rubyonrails.org/v5.1.4/classes/ActiveJob/QueueAdapters.html for adapters

NOTCH8

deliver_later

Lets you easily queue mail so your users never wait for SMTP handshakes

ActiveJob

```
bin/rails generate job guests_cleanup --queue urgent

class GuestsCleanupJob < ApplicationJob
  queue_as :default

  def perform(*guests)
    # Do something later
  end
end


# Configure
config.active_job.queue_adapter = :sidekiq
```

ActiveJob

```
# Enqueue a job to be performed as soon as the queuing system is free
GuestsCleanupJob.perform_later guest

# Enqueue a job to be performed tomorrow at noon.
GuestsCleanupJob.set(wait_until: Date.tomorrow.noon).perform_later(guest)

# Enqueue a job to be performed 1 week from now.
GuestsCleanupJob.set(wait: 1.week).perform_later(guest)

# `perform_now` and `perform_later` will call `perform` under the hood so
# you can pass as many arguments as defined in the latter.
GuestsCleanupJob.perform_later(guest1, guest2, filter: 'some_filter')
```

NOTCH8

ActiveJob

Callbacks

- before_enqueue
- around_enqueue
- after_enqueue
- before_perform
- around_perform
- after_perform

NOTCH8

Exercise

In app samvera-active-job in the VirtualBox image, move the following to be background tasks

- Sending an email report from SearchRecord once a day

- Creating a SearchRecord when a search is done

ApplicationRecord

- Adds a parent object that all models inherit from
- Makes AR uniform with ApplicationControllers
- Applies to ActionMailer and ActiveJob as well

NOTCH8

Attributes API

- gives types to attr_accessor or AR attribute objects
- you can add your own custom types

NOTCH8

```ruby
class Reservation < ApplicationRecord
  after_initialize :set_default_start_date
  after_initialize :set_default_end_date
  attr_accessor :end_date

  def price=(value)
    return super(0) if !value.to_s.include?('$')

    price_in_dollars = value.gsub(/\$/, '').to_d
    super(price_in_dollars * 100)
  end

  private

  def set_default_start_date
    self.start_date = 1.day.from_now if start_date.blank?
  end

  def set_default_end_date
    self.end_date = 8.days.from_now if end_date.blank?
  end
end
```

```
2.3.1 :001 > reservation = Reservation.new
 => #<Reservation id: nil, start_date: "2016-12-03",
                          price: nil, created_at: nil, updated_at: nil>
2.3.1 :002 > reservation.start_date
 => Sat, 03 Dec 2016
2.3.1 :003 > reservation.end_date
 => Sat, 10 Dec 2016
2.3.1 :004 > reservation = Reservation.new(start_date: 3.days.from_now)
 => #<Reservation id: nil, start_date: "2016-12-05",
                          end_date: “2016-12-10",
                          price: nil, created_at: nil, updated_at: nil>
2.3.1 :005 > reservation.start_date
 => Mon, 05 Dec 2016
```

```ruby
class Reservation < ApplicationRecord
  attribute :start_date, :date, default: -> { 1.day.from_now }
  attribute :end_date, :date, default: -> { 8.days.from_now }

  def price=(val)
    return super(0) if !value.to_s.include?('$')

    price_in_dollars = value.gsub(/\$/, '').to_d
    super(price_in_dollars * 100)
  end
end
```

app/types/price.rb

```
class PriceType < ActiveRecord::Type::Integer
  def cast(value)
    return super if value.kind_of?(Numeric)
    return super if !value.to_s.include?('$')

    price_in_dollars = BigDecimal.new(value.gsub(/\$/, ''))
    super(price_in_dollars * 100)
  end
end
```

config/initializers/types.rb

```
ActiveRecord::Type.register(:price, Price)
```

```ruby
class Reservation < ApplicationRecord
  attribute :start_date, :date, default: -> { 1.day.from_now }
  attribute :end_date, :date, default: -> { 8.days.from_now }
  attribute :price, :price
end
```

NOTCH8

Exercise

- Use a combination of ApplicationRecord and Attributes API to refactor the code found in samvera-attributes

NOTCH8

But really, mostly Javascript

Frontend

NOTCH8

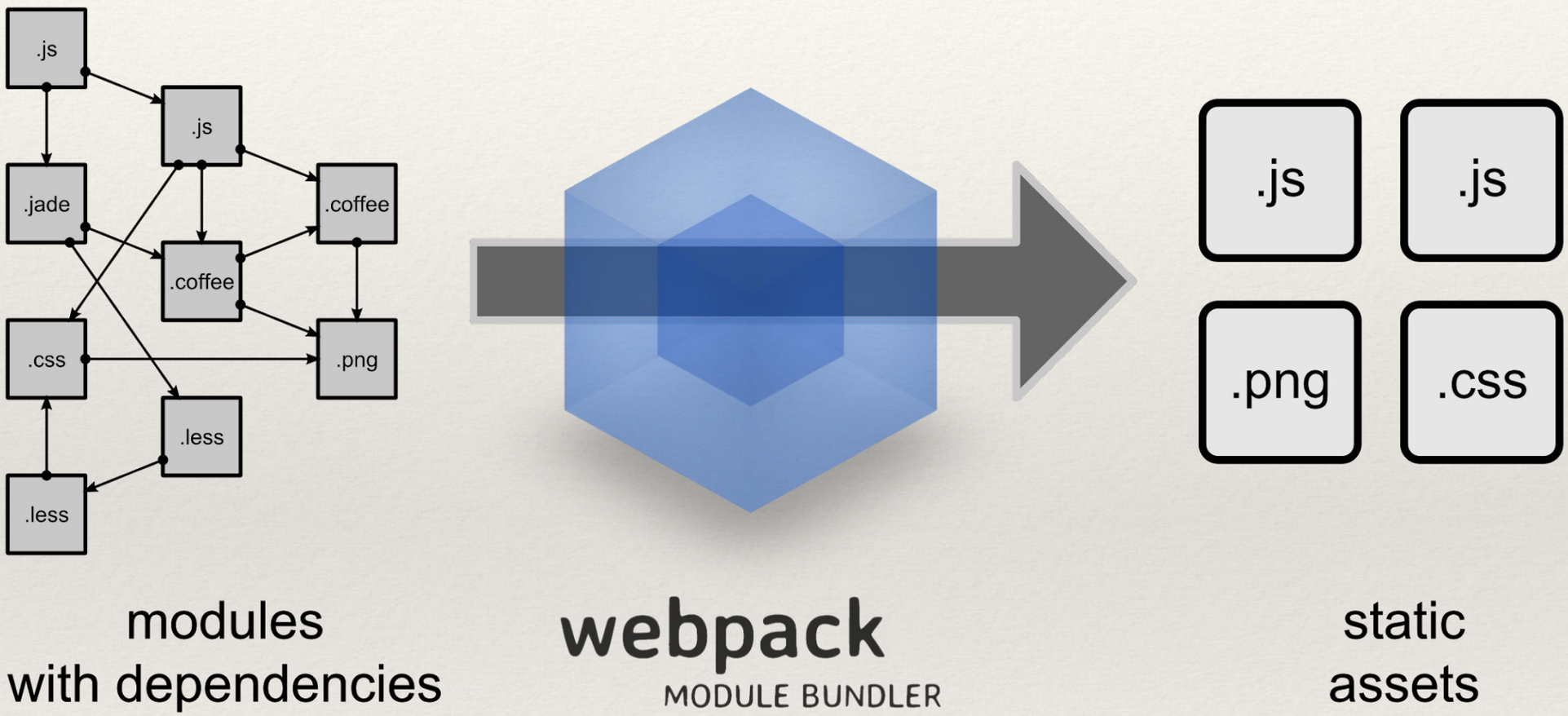
Webpacker in Rails

- Brings in two new tools to the asset pipeline flow
- The first is webpack a pre-compiler and packager (also babel)
- The second is yarn, which is built on top of NPM for package management, similar to how Bundler works on gem files

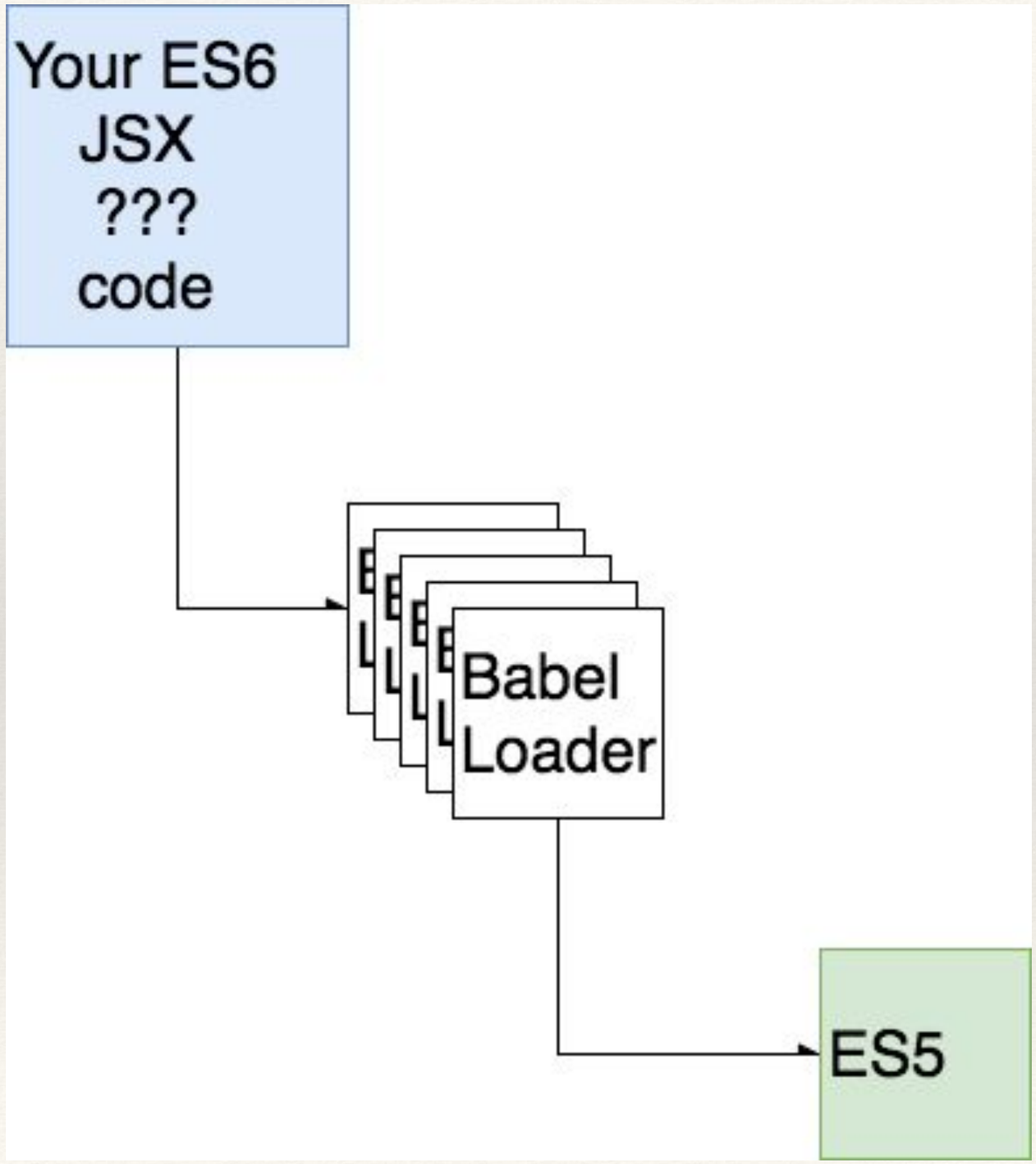
Webpacker in Rails

- Webpack is Javascript preprocessor and bundler that has wide adoption among current JS tools like React, Vue and and others.
- Webpacker, now built in to Rails, makes building React components in Rails applications first class citizens in the asset pipeline
- Doesn't require complex manipulation by Sprockets (the existing asset packager in Rails) but instead uses the same tool chain (Webpack and Bable) these communities use in other places.

Webpack


.js
.js
.jade
.coffee
.coffee
.css
.png
.less
.less
modules
with dependencies
webpack
MODULE BUNDLER
.js
.js
.png
.css
static
assets

What is Babel



Webpacker in Rails

- Javascript / CSS / Image sets can be built in to “packs” which are individually packaged up files that all get loaded together.
- A dashboard with several React widgets would be a good example of a pack.
- Packs live alongside your existing JS, but should not be intermixed
- Pack files live in app/javascript as opposed to app/assets/javascripts
- ./bin/webpacker-dev-server is only needed for hot reloading, we’re going to skip it for now and focus on what’s built in

Webpack Plus Babel

.babelrc

```
{
  "presets": [
    ["env", {
      "modules": false,
      "targets": {
        "browsers": "> 1%",
        "uglify": true
      },
      "useBuiltIns": true
    }],
    "stage-0",
    "react"
  ],
  "plugins": [
    "syntax-dynamic-import",
    "transform-object-rest-spread",
    ["transform-class-properties", { "spec": true }]
  ]
}
```

Yarn in Rails

- Webpacker packs have access to NPM packages which are installed via yarn. Adding a NPM dependency to your application is as easy as typing yarn add DEPNAME
- You can then import that package in your JS pack as normal

Autocomplete Search Field

- Given an existing hyrax app, lets make the search field an auto completing React component

NOTCH8

Create a React Component

app/javascript/components/search/index.js

```
import React from 'react'
import {Component} from 'react'

export default class Search extends Component {
  render() {
    return(<h1>Hello from Search Component</h1>)
  }
}
```

NOTCH8

Create a Pack

app/javascript/packs/search.js

```
import Search from 'components/search'
import WebpackerReact from 'webpacker-react'

WebpackerReact.setup({Search})
```

Add Component To Form

```
<!-- Add the pack to the document head -->
<% content_for :head do %>
  <%= javascript_pack_tag 'search' %>
<% end %>

...
    <!-- Replace Text element ith our React component -->
    <%= react_component('Search', {query: params[:q], placeholder:
t('blacklight.search.form.search.placeholder')}) %>

...
```

NOTCH8

Expanded Component

```javascript
import React from 'react'
import {Component} from 'react'

import Autocomplete from 'react-autocomplete'

export default class Search extends Component {
  constructor(props){
    super(props)
    this.state = {
      value: props.value,
      autocompleteOptions: [
        "John",
        "Paul",
        "George",
        "Ringo"
      ]
    }
  }
```

NOTCH8

Expanded Component

```
render() {
  return(
    <Autocomplete
      wrapperStyle={{width: "100%"}}
      inputProps = {{
        name: "q",
        type: 'text',
        placeholder: this.props.placeholder,
        id: "search-field-header",
        class: "q form-control"
      }}
      items={this.state.autocompleteOptions}
      value={this.state.value}
      onChange={(e) => this.setState({value: e.target.value})}
      onSelect={(val) => this.setState({value: val})}
      getItemValue={(item) => item}
      renderItem={(item, isHighlighted) =>
        <div
          key={item}
          style={{ background: isHighlighted ? 'lightgray' : 'white' }}
        >
          {item}
        </div>
        }
      />
  )
}
```

Exercise

- Using the README in samvera-react as a guide follow the steps to get your first React component showing up in a Samvera app

NOTCH8

ActionCable

- Websockets in Rails
- Can create realtime updating events and access them both on the server and on the client side
- Uses PubSub for clients
- Connections
  - Handles Authentication
- Channels
  - Kind of like a controller

Before Action Cable

Polling

Faye

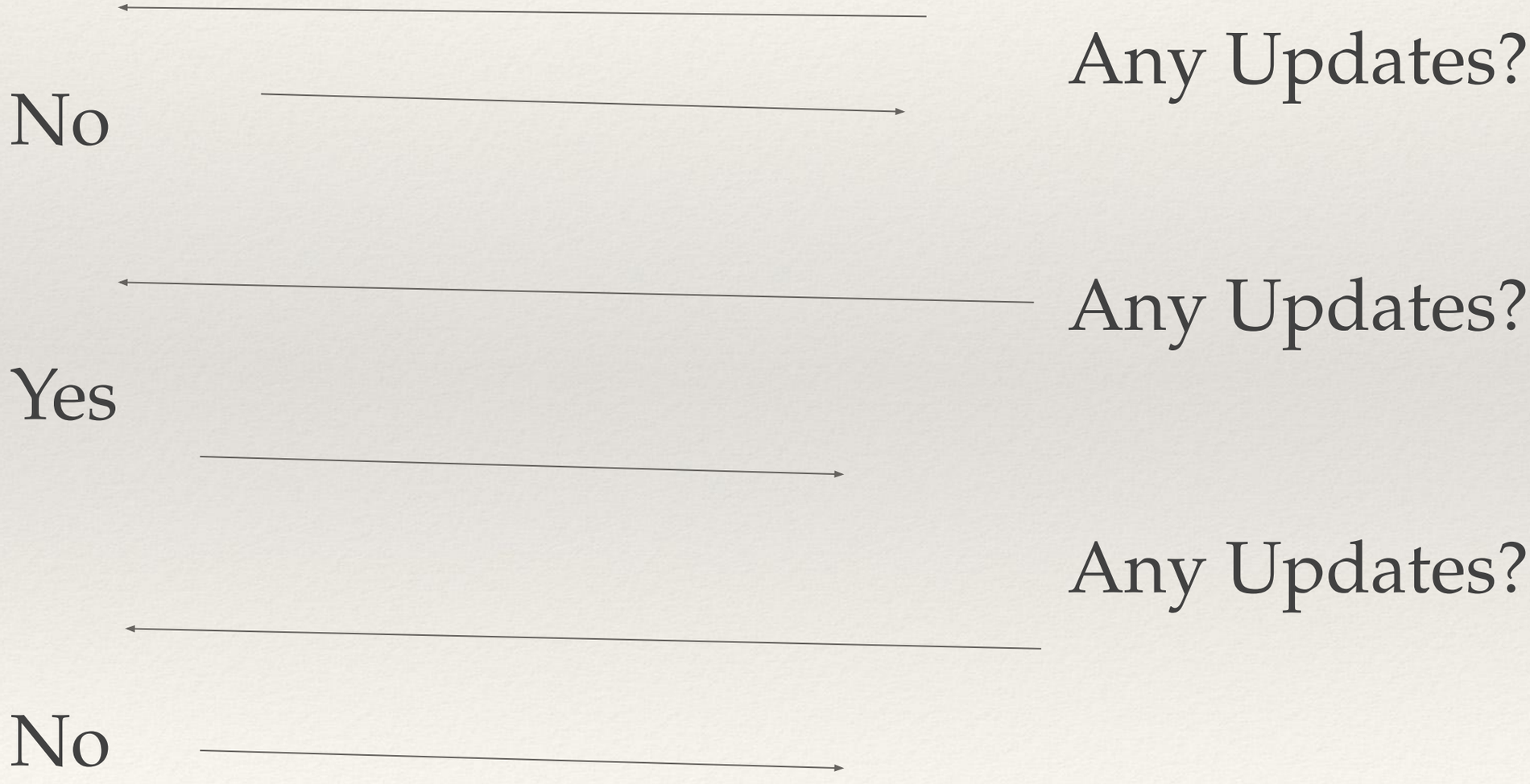
????

NOTCH8

Pubsub

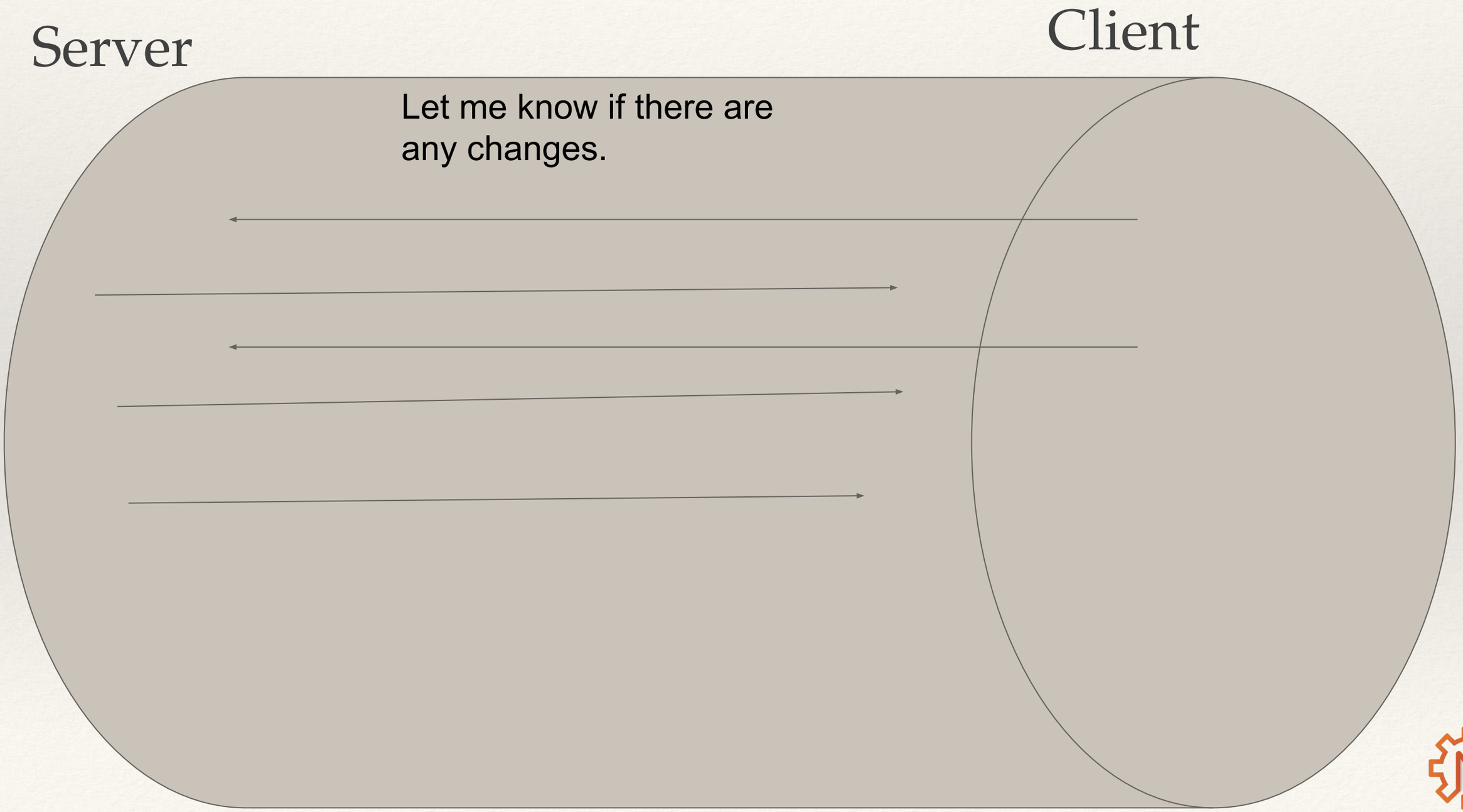
NOTCH8

Before Pubsub we had polling



NOTCH8

Websockets



NOTCH8

Server Side Connection

```ruby
module Hyrax
  module ApplicationCable
    class Connection < ActionCable::Connection::Base
      identified_by :current_user

      def connect
        self.current_user = find_verified_user
      end

      private

        def find_verified_user
          user = ::User.find_by(id: user_id)
          if user
            user
          else
            reject_unauthorized_connection
          end
        end

        def user_id
          session['warden.user.user.key'][0][0]
        rescue NoMethodError
          nil
        end

        def session
          cookies.encrypted[Rails.application.config.session_options[:key]]
        end
    end
  end
end
```

NOTCH8

Client Side Connection

```
// Action Cable provides the framework to deal with WebSockets in Rails.
// You can generate new channels where WebSocket features live using the rails generate channel command.
//
//= require action_cable
//= require_self
//= require_tree ./channels

(function() {
  this.App || (this.App = {});

  App.cable = ActionCable.createConsumer();

}).call(this);
```

Boilerplate. Rails set this up for us

A Channel

```
# app/channels/chat_channel.rb
class ChatChannel < ApplicationCable::Channel
  def subscribed
    stream_from "chat_#{params[:room]}"
  end
end
```

Broadcasting to a Channel

```
#  app/controllers/messages_controller.rb

class MessagesController < ApplicationController

  def create
    message = Message.new(message_params)
    message.user = current_user
    if message.save
      ActionCable.server.broadcast “chat_#{params[:room]},
        message: message.content,
        user: message.user.username
      head :ok
    end
  end
end
```

Client Side

```
// app/assets/javascripts/channels/messages.js

App.messages = App.cable.subscriptions.create({channel: 'ChatChannel', room: ‘Samvera’}, {
  received: function(data) {
    return $('#messages').append(this.renderMessage(data));
  },

  renderMessage: function(data) {
    return "<p> <b>" + data.user + ": </b>" + data.message + "</p>";
  }
});
```

React?

```
import React, { Component } from 'react';

export default class LiveSearch extends Component {
  constructor(props){
    super(props)
    this.state = {
      searches: []
    }
  }

componentWillMount(){
  App.cable.subscriptions.create('LiveSearchChannel',
  {
     received: function(data){
       const newSearches = this.state.searches.slice(0)
       newSearches.push(data)
       this.setState({searches: newSearches})
     }.bind(this)
   })
}
```

Live Coding

Create a LiveSearch Component

/app/javascript/components/LiveSearch.js

```
import React, { Component } from 'react';

export default class LiveSearch extends Component {
  render() {
    return (
      <h3>Live Search</h3>
    );
  }
}
```

LiveSeach Pack

/app/javascript/packs/LiveSearch.js

```
import LiveSearch from '../components/LiveSearch'
import WebpackerReact from 'webpacker-react'

WebpackerReact.setup({LiveSearch})
```

NOTCH8

Copy over layout for head

Create a new file: /app/views/catalog/_search_sidebar.html.erb

```
<% content_for :head do %>
  <%= javascript_pack_tag 'LiveSearch' %>
<% end %>

<h4>Sidebar</h4>
<%= react_component(“LiveSearch”) %>
```

Working with Action Cable

$ rails g channel live_search

/app/channels/live_search_channel.rb

```
0 class LiveSearchChannel < ApplicationCable::Channel
1   def subscribed
2     stream_from "live_search"
3   end
4
5   def unsubscribed
6     # Any cleanup needed when channel is unsubscribed
7   end
8 end
~
~
~
```

NOTCH8

Live Search Component - Subscribe

/app/javascript/components/LiveSearch.js

```
import React, { Component } from 'react';

export default class LiveSearch extends Component {
  constructor(props){
    super(props)
    this.state = {
      searches: []
    }
}

componentWillMount(){
  App.cable.subscriptions.create('LiveSearchChannel',
  {
     received: function(data){
       const newSearches = this.state.searches.slice(0)
       newSearches.push(data)
       this.setState({searches: newSearches})
     }.bind(this)
   })
}
```

NOTCH8

Live Search -Render

```
render() {
  return (
    <div>
      <h3>Live Search</h3>
      <h5>Check out some of these search by other users</h5>
      {this.state.searches.map((search, index)=>{
        return(
          <div key={index}>
            <div class='card-body'>
              <a href={`/catalog?utf8=✓&locale=en&search_field=all_fields&q=${search}`}>
                {search}
              </a>
            </div>
          </div>
        )
      })}
    </div>
  );
}
```

NOTCH8

Broadcast Searches

```
before_action :broadcast_live_search

def broadcast_live_search
    if params[:q]
      ActionCable.server.broadcast "live_search",
params[:q]
    end
  end
```

Exercise

Use the samvera-action-cable repo to create your own live search component

NOTCH8

It's OVER!!

Thank You!

Rob Kaufman
rob@notch8.com
http://spkr8.com/s/7218
@orangewolf

Matt Clark
matt@notch8.com
@winescout

NOTCH8