



TRP.

407

try ember



Level 2.2

# Routing and Templating

Routes





# Driving With Data

Ember is built for delivering dynamic, data-driven applications.

```
app/templates/index.hbs
```

```
Hello from Index  
{{#link-to "orders"}}Orders{{/link-to}}
```

*This is all static content.*




WOODLAND WANDERER  
**WHATCHAMACALLITS**

MENU

ORDER

Order details for Blaise Blobfish (#1234)

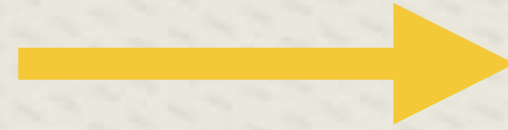
Item	Qty	Unit	Cost	Cost%
 Tent	1	\$10/ea	\$10	63%

*How do we put dynamic data into a template?*



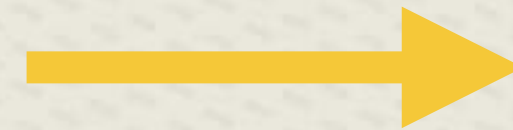
# Recalling What We Know

---



Manages state

*It does not manage templates.*



Defines HTML

*Often from dynamic data.*

*Where does it get the data?*





# Discovering a Hidden Layer

Ember has been automatically generating a hidden layer between the router and templates.

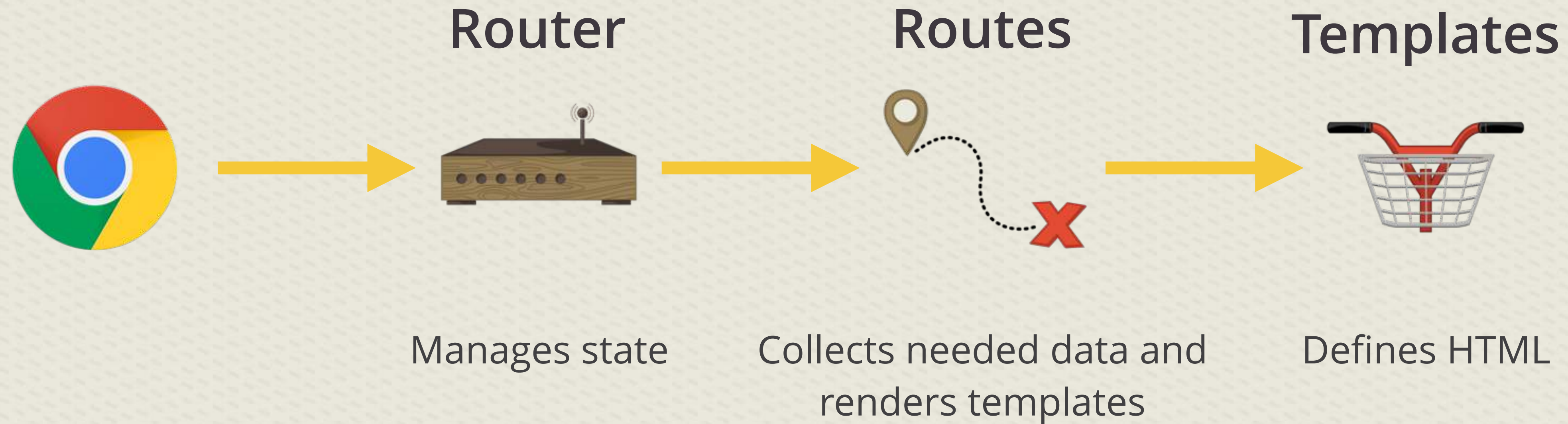




# Introducing Routes



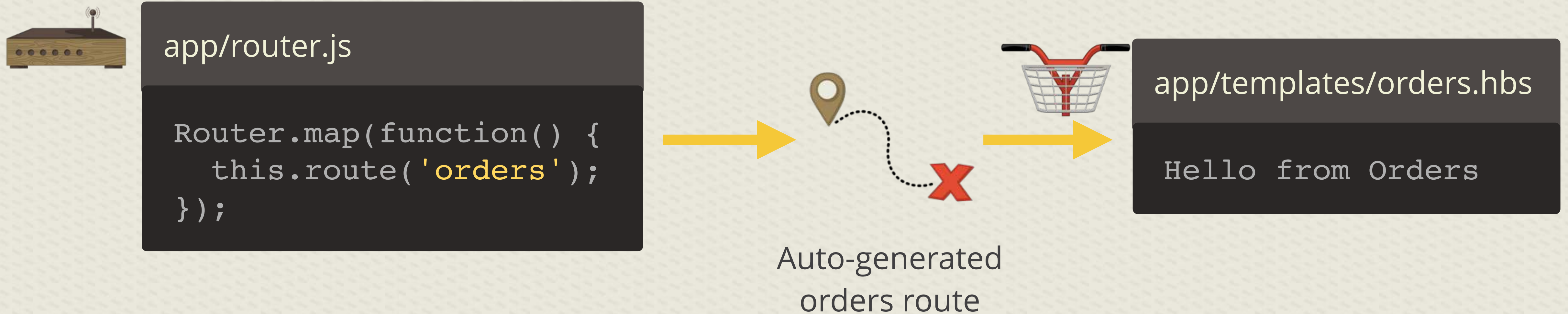
Routes are responsible for collecting data and rendering the proper templates.





# Auto-generated Routes

Auto-generated routes render the template of the same name.



The generated orders route renders the "orders" template.





# Generating a Route

Ember CLI provides a generator for creating a route and updating the router.

```
ember generate route <route-name>
```

## Console

```
$ ember generate route orders
installing route
  create app/routes/orders.js
  create app/templates/orders.hbs
updating router
  add route orders
...
```



Adds to the router

```
this.route('orders');
```

This is the orders route.







# Examining the Route

Routes are defined in app/routes with a file name matching their route name.

app/routes/orders.js



```
import Ember from 'ember';  
  
export default Ember.Route.extend({  
});
```



This is the “orders” route.  
Currently, it’s identical to the  
auto-generated default.

Using `extend` is creating a subclass of the  
`Ember.Route` object.





# Customizing the Route's Model

The model hook returns the data used in the route and its template.

app/routes/orders.js

```
import Ember from 'ember';

export default Ember.Route.extend({
  model() {
    return 'Nate';
  }
});
```

app/templates/orders.hbs

Order for {{model}}

The route's model is available to the template as "model".





# Working With an Object

The model hook may return objects.

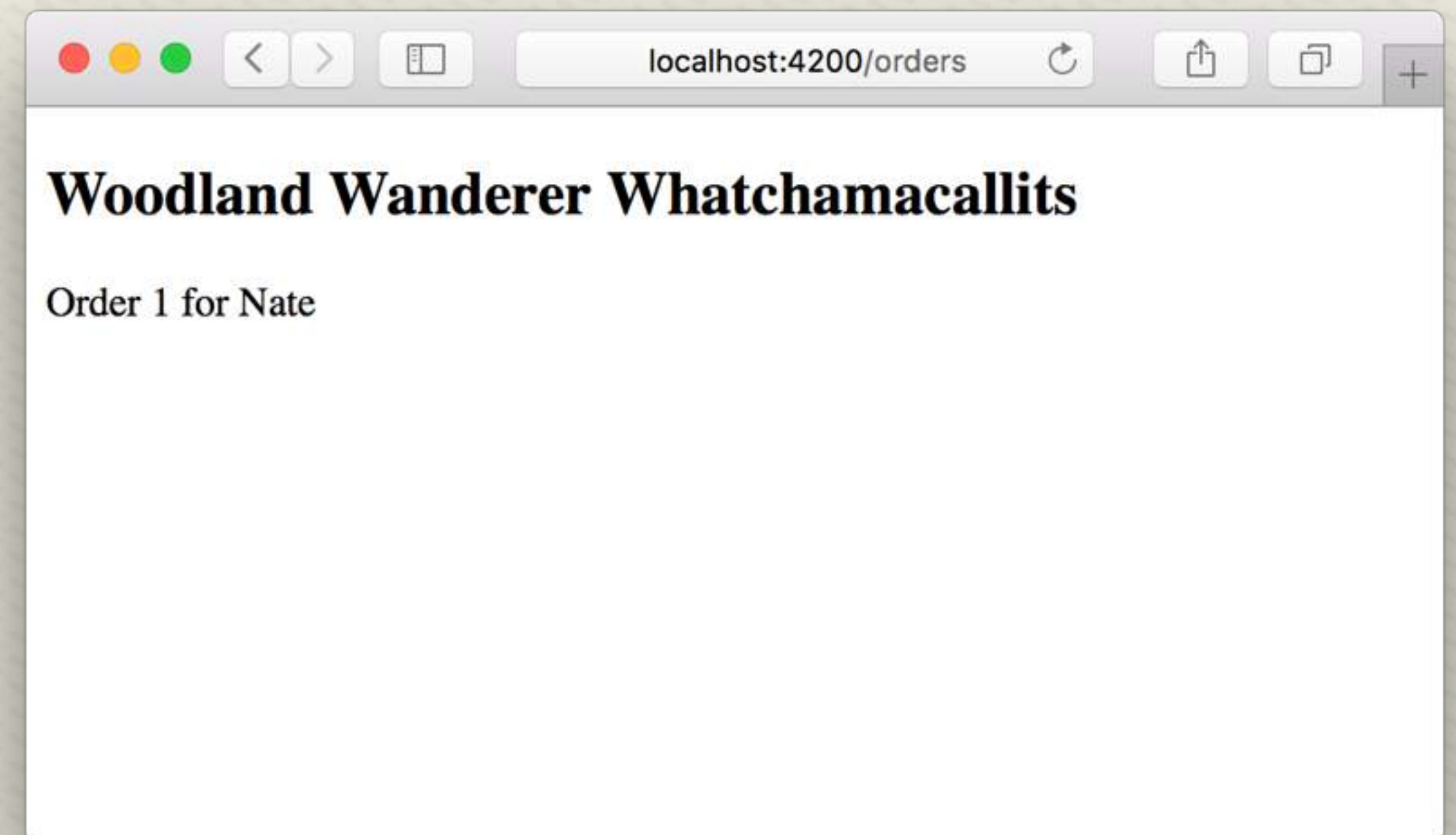
app/routes/orders.js

```
import Ember from 'ember';

export default Ember.Route.extend({
  model() {
    return { id: '1', name: 'Nate' };
  }
});
```

app/templates/orders.hbs

```
Order {{model.id}} for {{model.name}}
```



Here, *model* is an object and its properties are accessed with `{{model.property}}`.

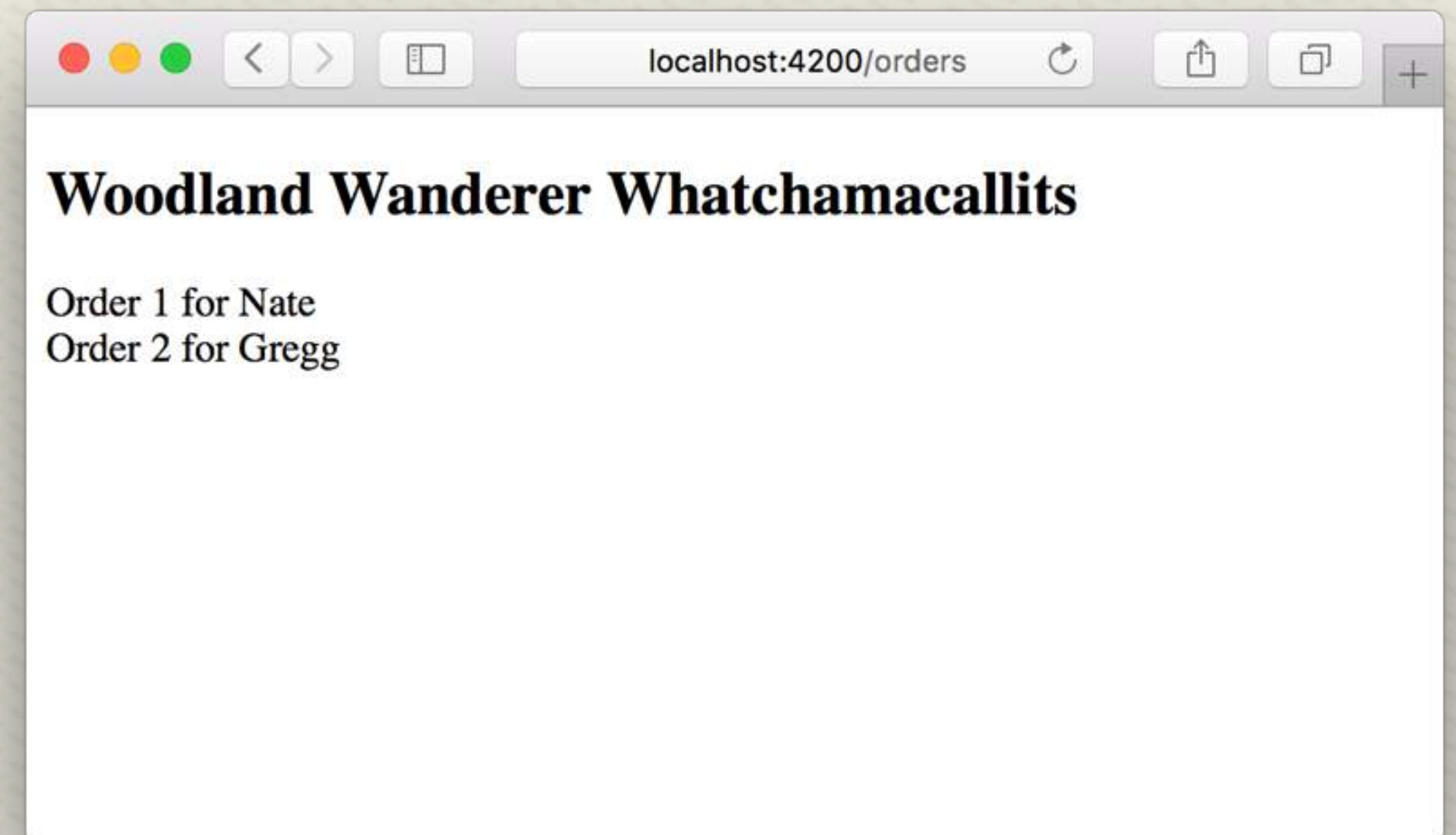


# Working With Collections

The `{{#each}}` helper iterates over a collection and renders the block once for each item.

```
import Ember from 'ember';

export default Ember.Route.extend({
  model() {
    return [
      { id: '1', name: 'Nate' },
      { id: '2', name: 'Gregg' }
    ];
  }
});
```



app/templates/orders.hbs

```
{{#each model as |order|}}
  Order {{order.id}} for {{order.name}}<br>
{{/each}}
```

“order” is populated with one successive order each iteration.



# Customizing a Route

---

Routes have several hooks for customizing their behavior.

## Commonly Used Route Hooks

Name	Usage
activate	Triggered when navigation enters the route.
deactivate	Triggered when navigation leaves the route.
model	Returns model data to use in the route and template.
redirect	Optionally used to send the user to a different route.






# Linking to a Single Item

Now that we're displaying the orders, we still need to link to each one.

app/templates/orders.hbs

```
{{#each model as |order|}}  
  {{#link-to ???}}  
    Order {{order.id}} for {{order.name}}<br>  
  {{/link-to}}  
{{/each}}
```



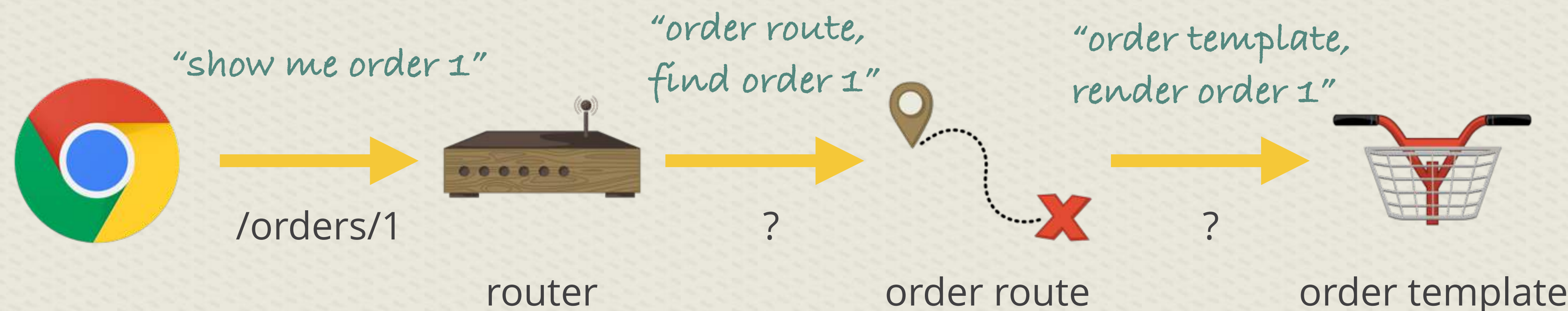
But how do we link to a single item?  
And where do we link it to?





# Navigating to a Single Item

In order to link to a single item, we need a route that can load and render a single item.



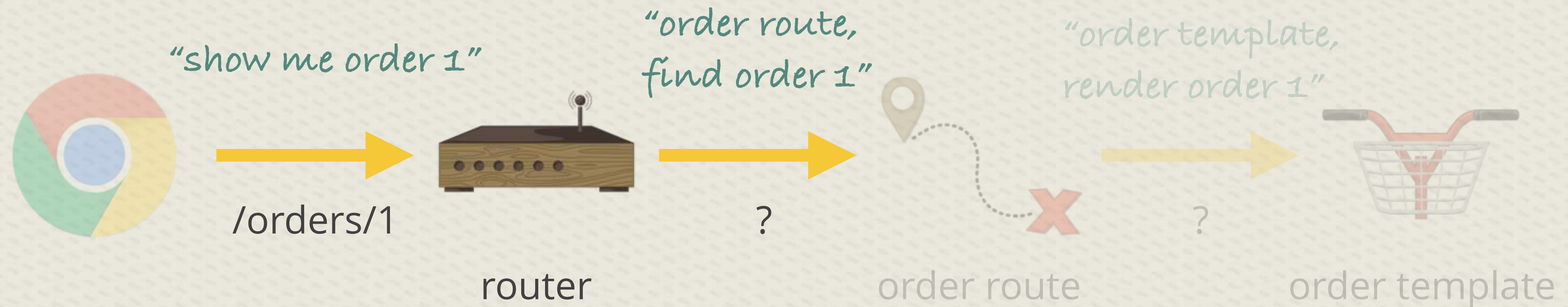
State	URL
View the menu	/
List and create orders	/orders
View a receipt for Order #	/orders/###

This route doesn't exist yet, but its functionality is described above.



# Defining Dynamic Segments in the Router

Dynamic segments are a part of the URL path that holds variable data, like identifiers.



app/router.js

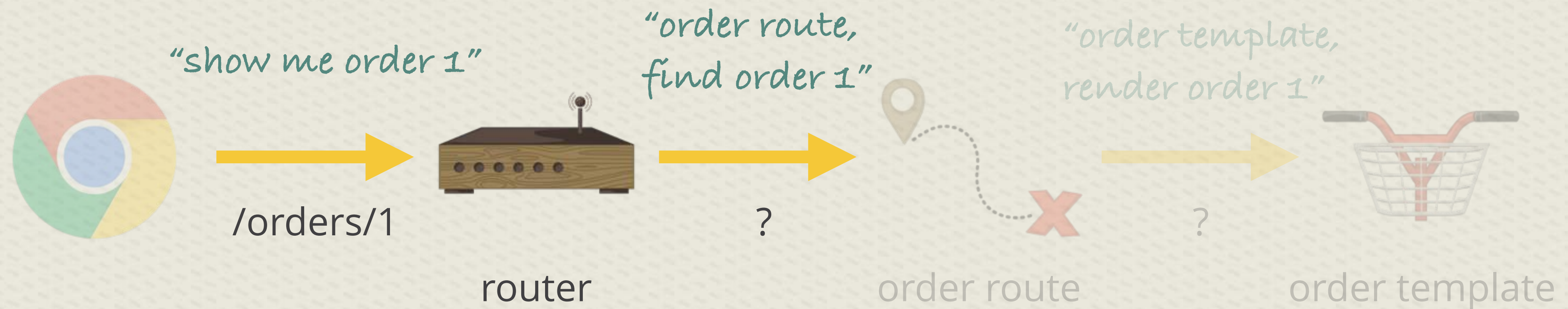
```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```

*Dynamic segments start with a colon.*



# Defining Dynamic Segments in the Router

Dynamic segments are a part of the URL path that holds variable data, like identifiers.



app/router.js

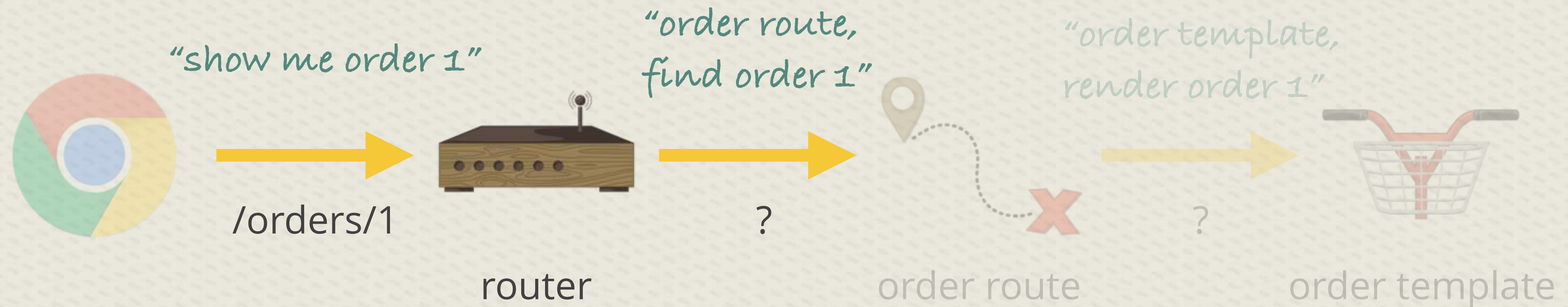
```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```

← /orders/1



# Defining Dynamic Segments in the Router

Dynamic segments are a part of the URL path that holds variable data, like identifiers.



app/router.js

```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```

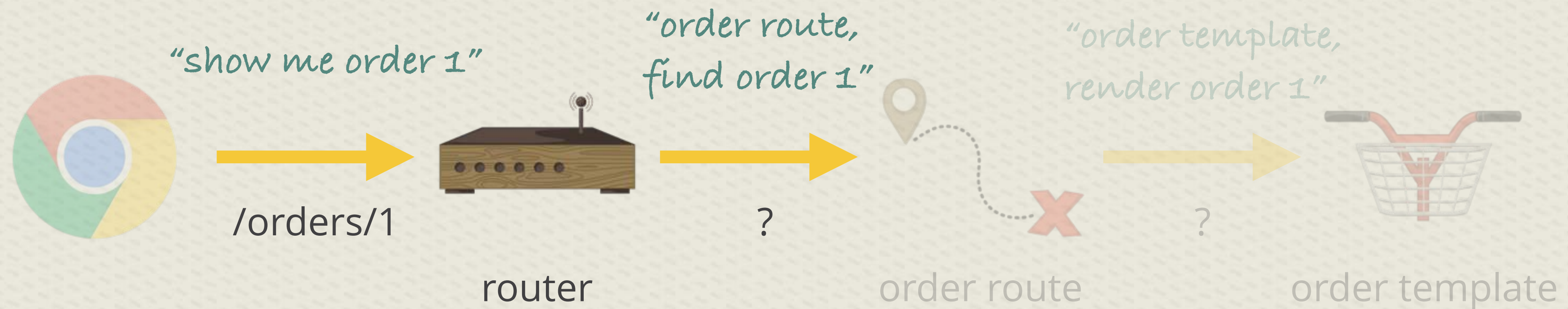
← /orders/1

"/orders" does not match "/orders/1".



# Defining Dynamic Segments in the Router

Dynamic segments are a part of the URL path that holds variable data, like identifiers.



app/router.js

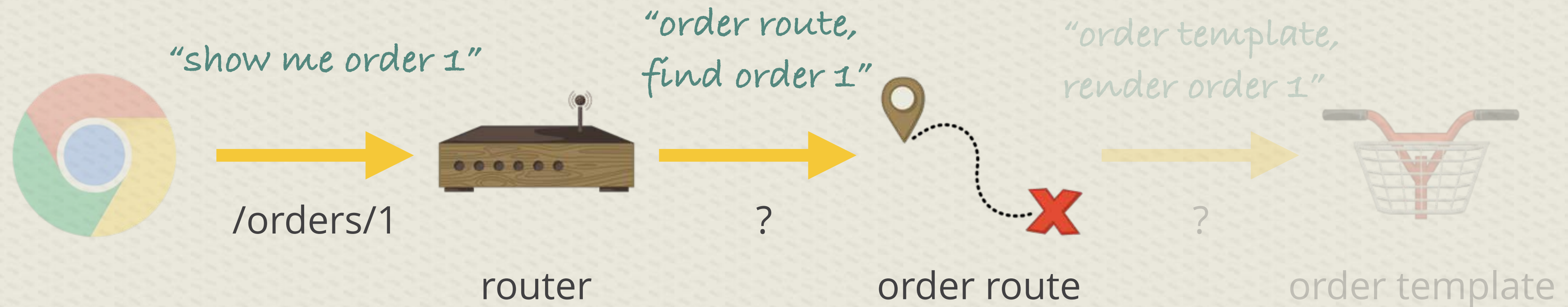
```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```

← `/orders/1`



# Defining Dynamic Segments in the Router

Dynamic segments are a part of the URL path that holds variable data, like identifiers.



app/router.js

```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```

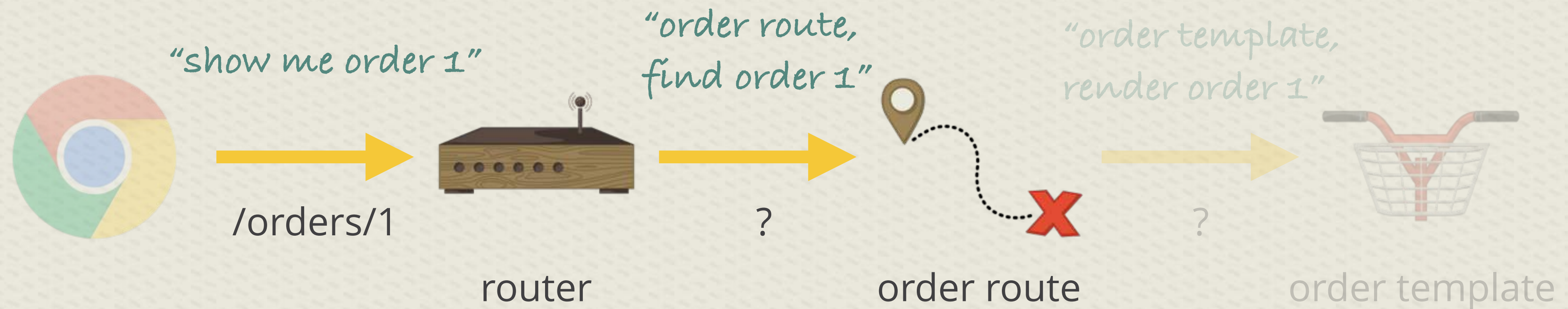
← /orders/1

"/orders/:order\_id" matches "/orders/1"  
because dynamic segments are placeholders.



# Defining Dynamic Segments in the Router

Dynamic segments are a part of the URL path that holds variable data, like identifiers.



app/router.js

```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```

/orders/:order\_id

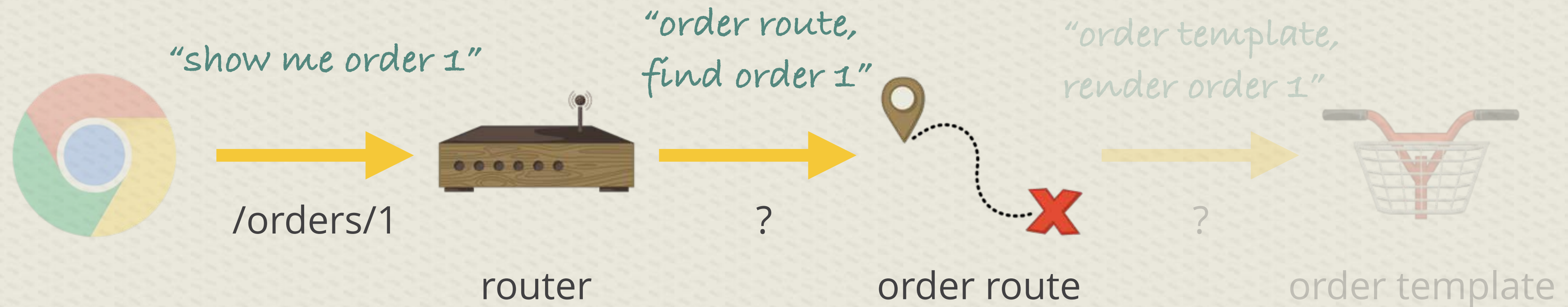
← /orders/1

Dynamic portion is extracted.



# Defining Dynamic Segments in the Router

Dynamic segments are a part of the URL path that holds variable data, like identifiers.



app/router.js

```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```

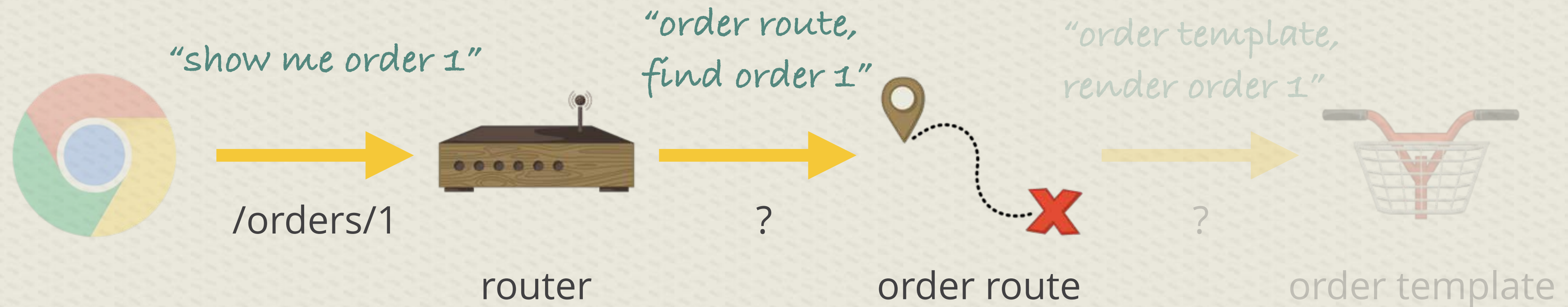
:order\_id

1



# Defining Dynamic Segments in the Router

Dynamic segments are a part of the URL path that holds variable data, like identifiers.



app/router.js

```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```

{:order\_id 1}



# Defining Dynamic Segments in the Router

Dynamic segments are a part of the URL path that holds variable data, like identifiers.



app/router.js

```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```

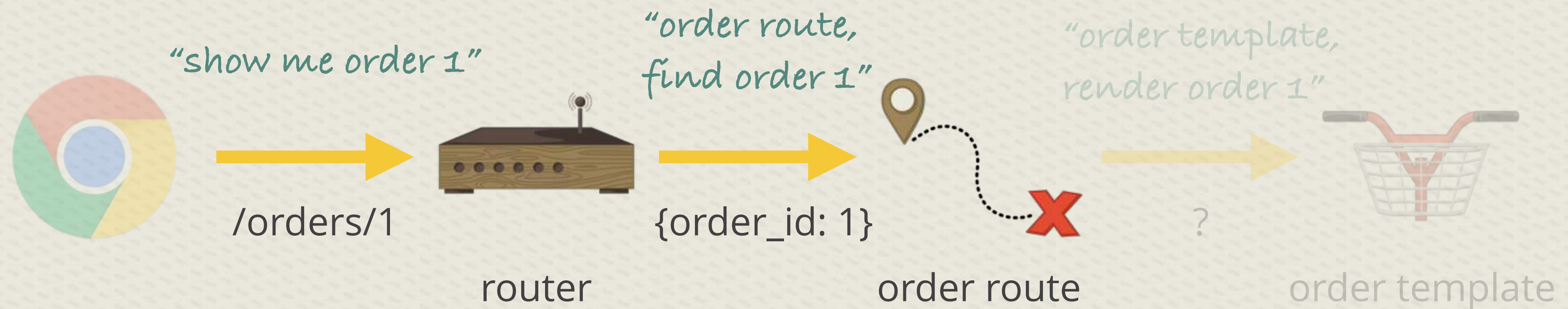
{order\_id: 1}

↑  
Data sent to the  
route



# Defining Dynamic Segments in the Router

Dynamic segments are a part of the URL path that holds variable data, like identifiers.



app/router.js

```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```

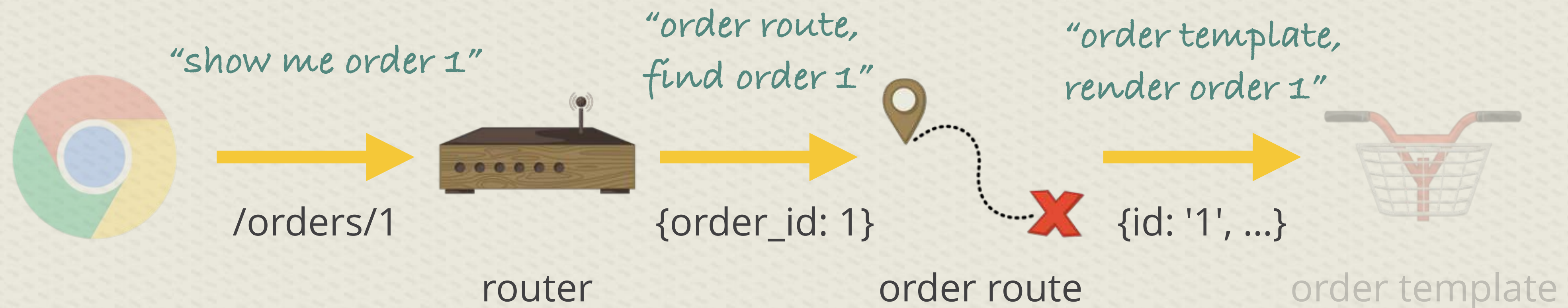
`{order_id: 1}`

↑  
Data sent to the  
route



# Using the Dynamic Segment Values

The router passes the dynamic segment values to the route's model hook.



app/routes/order.js

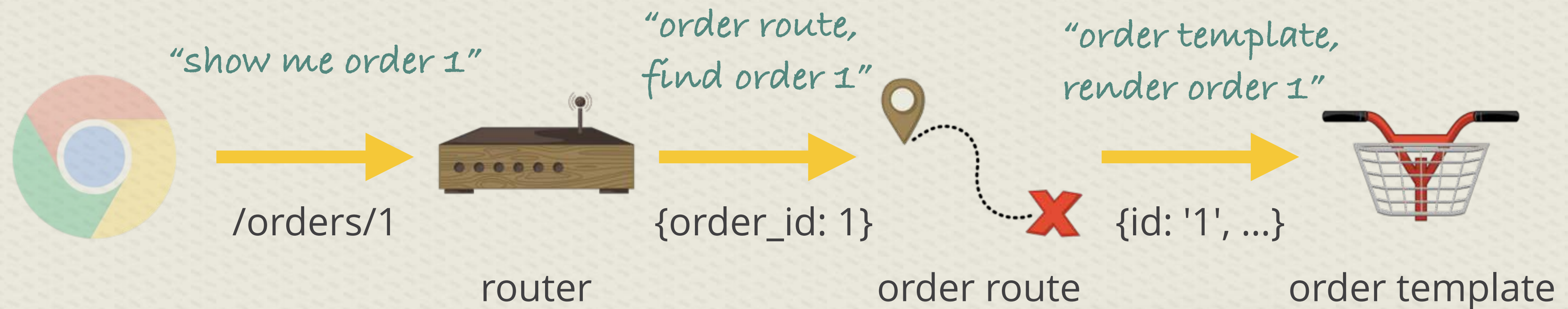
```
export default Ember.Route.extend({
  model(params) {
    return [
      { id: '1', name: 'Nate' },
      { id: '2', name: 'Gregg' }
    ].findBy('id', params.order_id);
  }
});
```

`findBy(property, value)` is provided by Ember and returns the first matching item.



# Displaying the Order

The order template displays the given model data as we've done before.



app/routes/order.hbs

```
<p>Order {{model.id}} for {{model.name}}</p>
<p>The order is ready!</p>
```





# Linking to a Single Item

---

Now that we have a route to link to, how do we use it?

```
app/templates/orders.hbs
```

```
{{#each model as |order|}}  
  {{#link-to ???}}  
    Order {{order.id}} for {{order.name}}<br>  
  {{/link-to}}  
{{/each}}
```





# Linking to a Single Item

The `{{#link-to}}` helper accepts one or more objects to populate the dynamic segments.

*"order" route name*

*order object*

app/templates/orders.hbs

```
{{#each model as |order|}}  
  {{#link-to "order" order}}  
    Order {{order.id}} for {{order.name}}<br>  
  {{/link-to}}  
{{/each}}
```

*This links to the "order" route and passes the order that we'd like to view.*

```
{{#link-to "<route-name>" <object-to-view>}}
```





# Linking to Objects With Their ID

The `{{link-to}}` helper automatically uses the given object's ID as the dynamic segment value.

app/templates/orders.hbs

```
{{#each model as |order|}}  
  {{#link-to "order" order}}  
    Order {{order.id}} for {{order.name}}<br>  
  {{/link-to}}  
{{/each}}
```

Rendered HTML

```
<a href="/orders/1">Order 1 for Nate<br></a>  
<a href="/orders/2">Order 2 for Gregg<br></a>
```

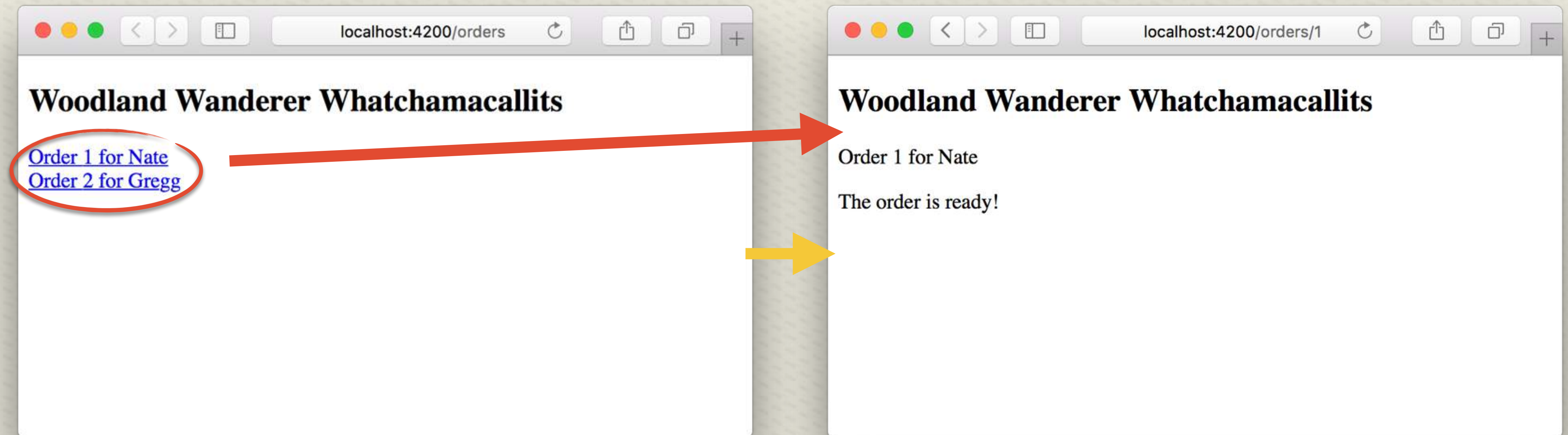


Automatically set from the ID property of the order instance.





# Navigating to an Order



viewing an order detail loses the order listing.



# Visualizing the Current Structure

---

The “flat” route mappings create a flat hierarchy where each route replaces the other.

```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```

application.hbs

*page header*

*page content*  
**{{outlet}}**

*page footer*

index.hbs

orders.hbs

order.hbs



# Visualizing the Current Structure

Navigating to the root path renders the index template into the application outlet.

```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```

application.hbs

*page header*

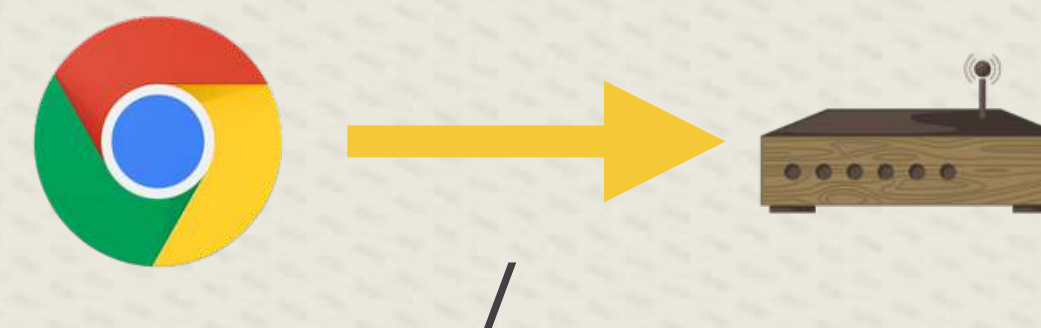
*page content*  
**{{outlet}}**

*page footer*

index.hbs

orders.hbs

order.hbs



URL path	/
Route name	index
Route	index.js
Template	index.hbs



# Visualizing the Current Structure

Navigating to the /orders path replaces the index template with the rendered orders template.

```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```

application.hbs

*page header*

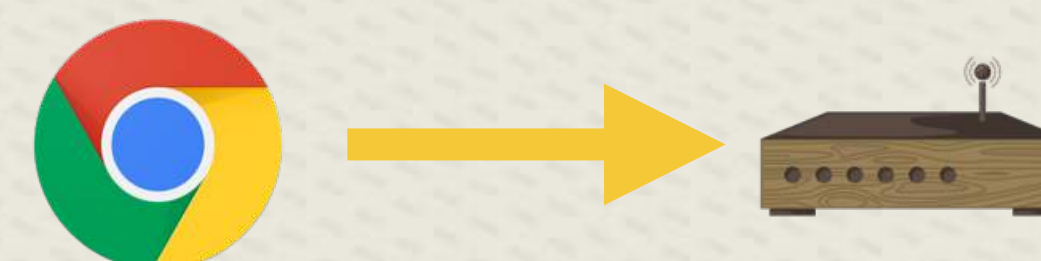
*page content*  
**{{outlet}}**

*page footer*

index.hbs

orders.hbs

order.hbs



/orders

URL path	/orders
Route name	orders
Route	orders.js
Template	orders.hbs



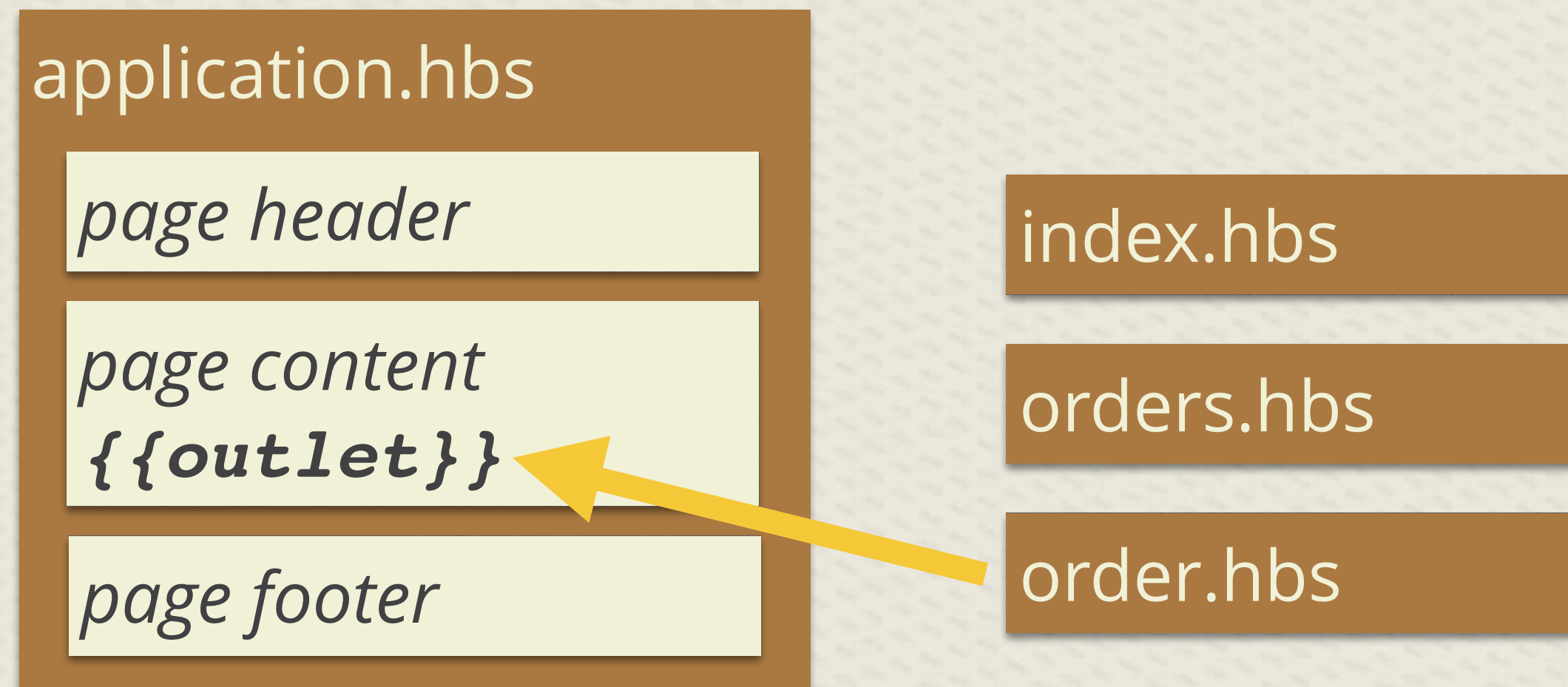
# Visualizing the Current Structure

Navigating to the `/orders/1` path replaces the orders template with the order template.

```
Router.map(function() {  
  this.route('orders');  
  this.route('order', { path: '/orders/:order_id' });  
});
```



URL path	/orders/1
Route name	order
Route	order.js
Template	order.hbs



The order template replaced the orders list!



# Nesting the Order Routes

A nested mapping allows multiple routes and templates to be displayed.

```
Router.map(function() {  
  this.route('orders', function() {  
    this.route('order', { path: '/:order_id' });  
  });  
});
```

Adding an anonymous function defines a parent route.

application.hbs

*page header*

*page content*  
**{{outlet}}**

*page footer*

index.hbs

orders.hbs

*list*

*order detail*  
**{{outlet}}**

orders/index.hbs

orders/order.hbs

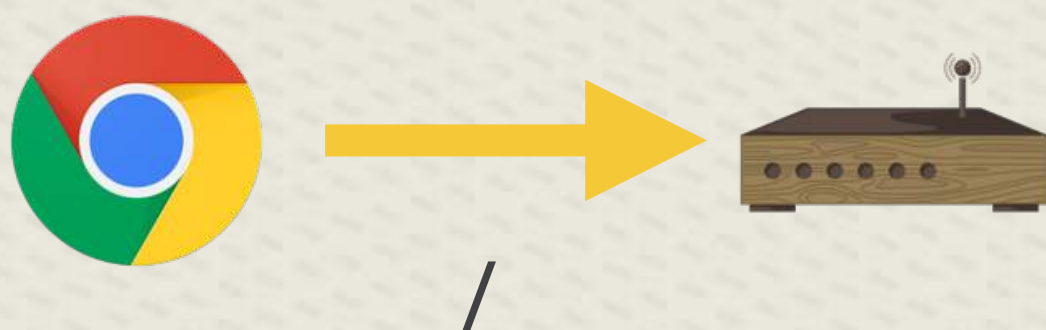
Just like application, the orders template needs an {{outlet}} now!



# Nesting the Order Routes

Navigating to the root path renders the index template into the application outlet.

```
Router.map(function() {  
  this.route('orders', function() {  
    this.route('order', { path: '/:order_id' });  
  });  
});
```



URL path	/
Route name	index
Route	index.js
Template	index.hbs

The same as before.

application.hbs

page header

page content  
**{{outlet}}**

page footer

index.hbs

orders.hbs

list

order detail  
**{{outlet}}**

orders/index.hbs

orders/order.hbs



# Nesting the Order Routes

A nested mapping allows multiple routes and templates to be displayed.

```
Router.map(function() {  
  this.route('orders', function() {  
    this.route('order', { path: '/:order_id' });  
  });  
});
```



URL path	/orders
Route name	orders.index
Route	orders/index.js
Template	orders/index.hbs

application.hbs

page header

page content  
**{{outlet}}**

page footer

index.hbs

orders.hbs

list

order detail  
**{{outlet}}**

orders/index.hbs

orders/order.hbs

A new "orders.index"  
route is automatically  
created.

Index routes are always created for parent routes.  
They're what's rendered when you go to the parent.



# Nesting the Order Routes

A nested mapping allows multiple routes and templates to be displayed.

```
Router.map(function() {  
  this.route('orders', function() {  
    this.route('order', { path: '/:order_id' });  
  });  
});
```



URL path	/orders/1
Route name	orders.order
Route	orders/order.js
Template	orders/order.hbs

application.hbs

page header

page content  
**{{outlet}}**

page footer

index.hbs

orders.hbs

list

order detail  
**{{outlet}}**

orders/index.hbs

orders/order.hbs

An "orders" namespace is added to the order route.



# Fixing Broken Links

Nesting introduced the “orders.” route namespace. Links need to get updated to match.

app/templates/orders.hbs

```
{{#each model as |order|}}  
  {{#link-to "orders.order" order}}  
    Order {{order.id}} for {{order.name}}<br>  
  {{/link-to}}  
{{/each}}  
  
{{outlet}}
```

Change “order” to “orders.order”.

Add the {{outlet}} so child templates may be displayed.

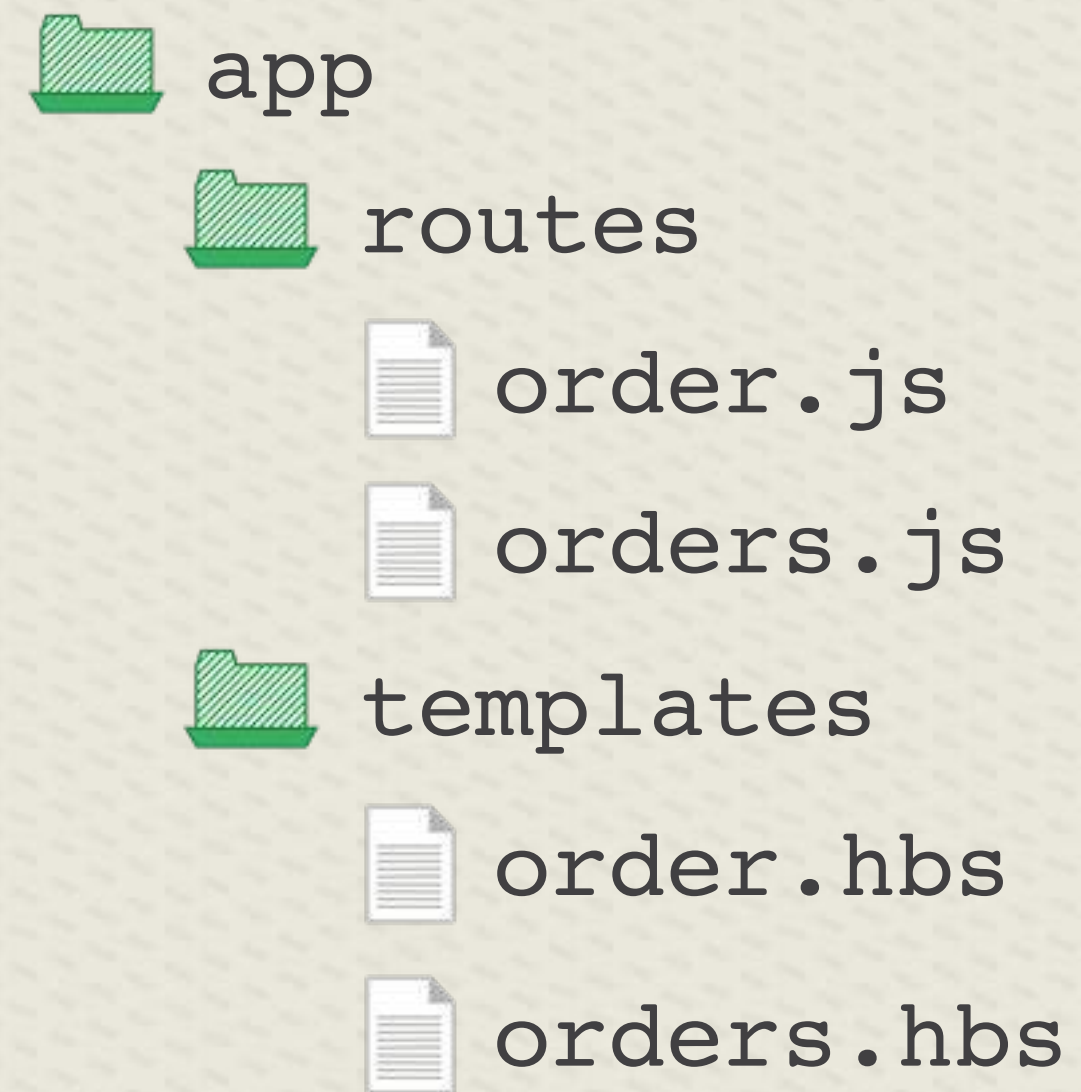




# Relocating Nested Files

---

Nesting introduced an “orders/” directory namespace. Files must move to match.

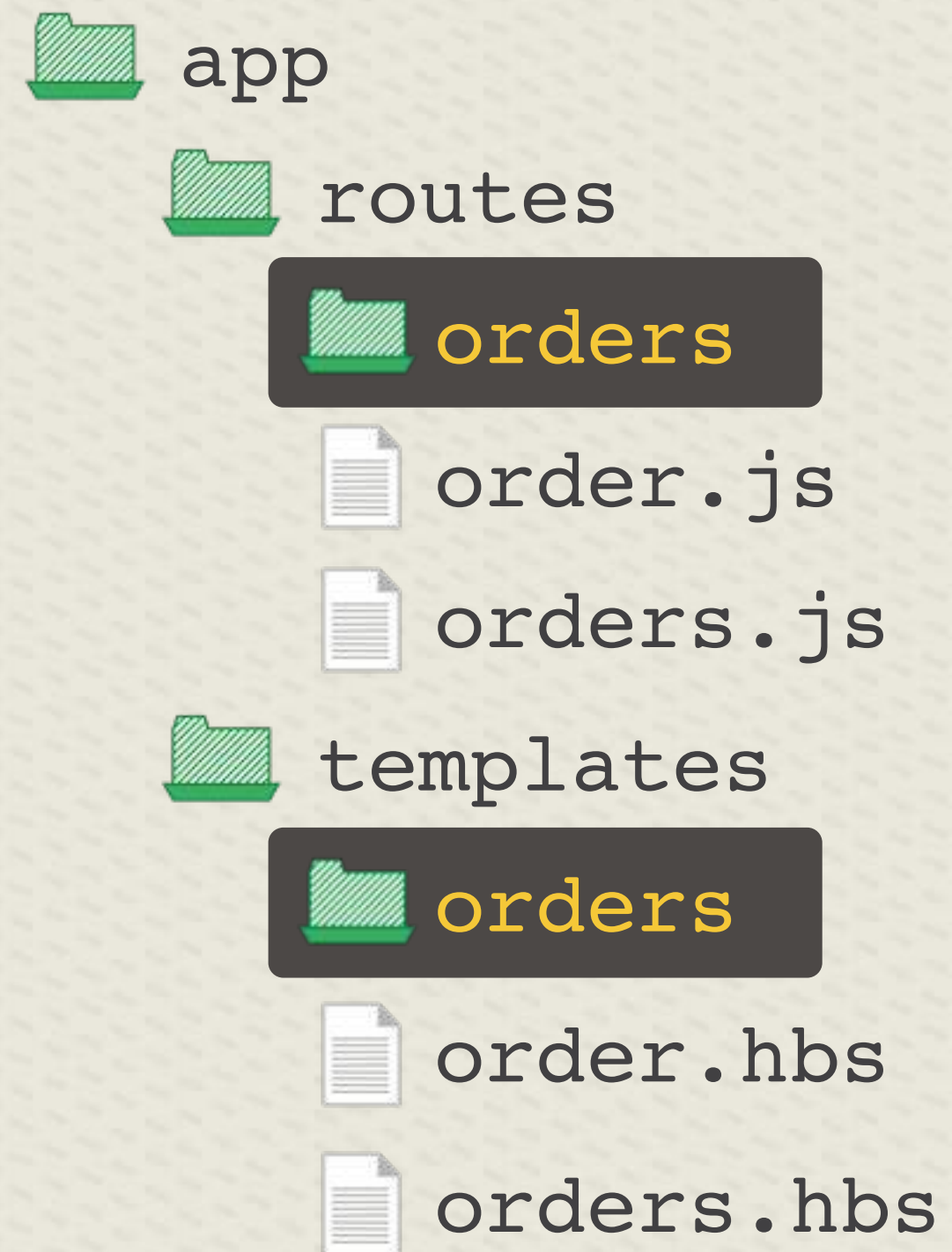




# Relocating Nested Files

---

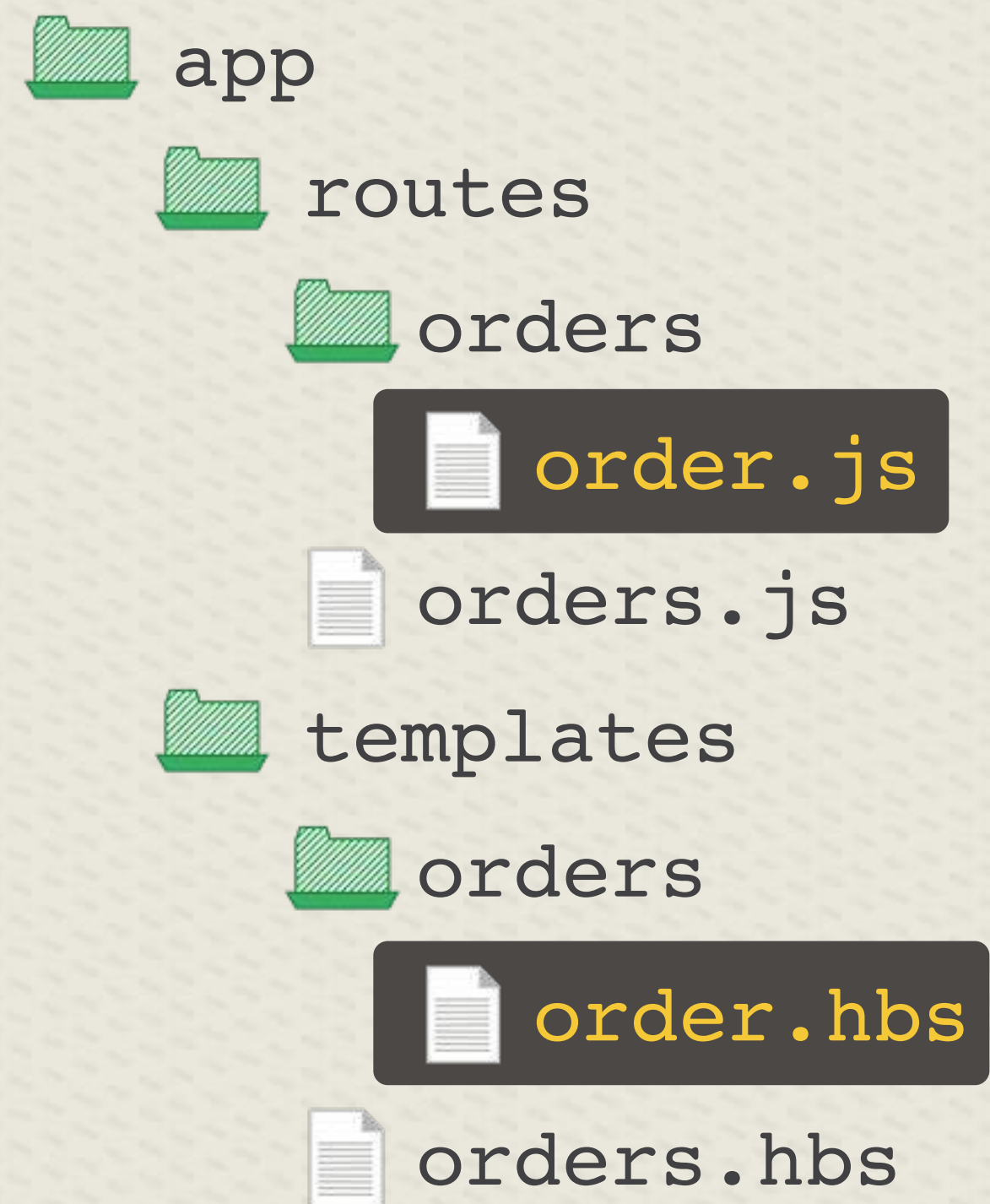
Nesting introduced an “orders/” directory namespace. Files must move to match.





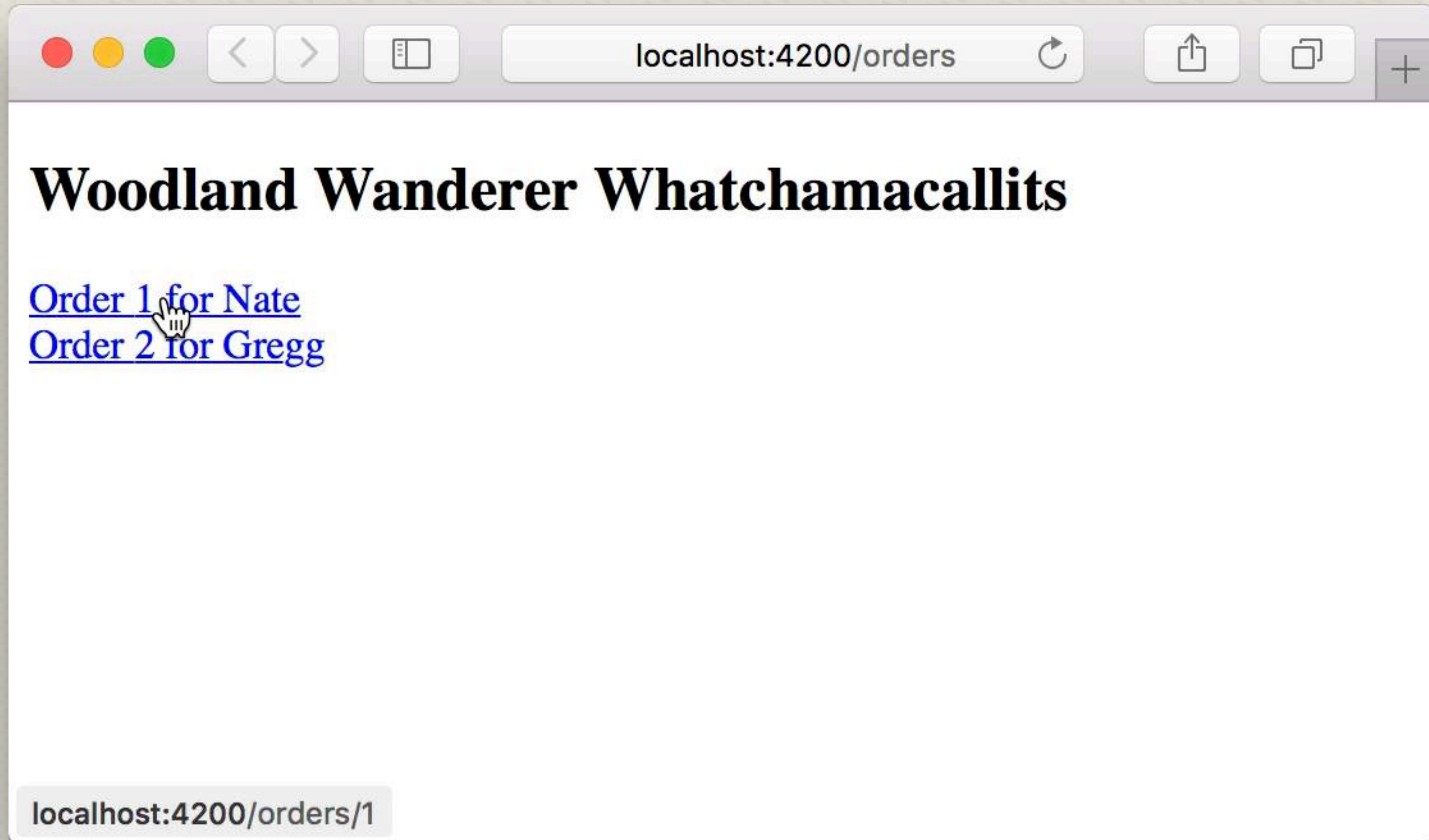
# Relocating Nested Files

Nesting introduced an “orders/” directory namespace. Files must move to match.





# Retaining the Listing





Level 2.2

# Routing and Templating

Routes

