THE ANATOMY OF

Backbone.js

Introduction

- LEVEL 1 -

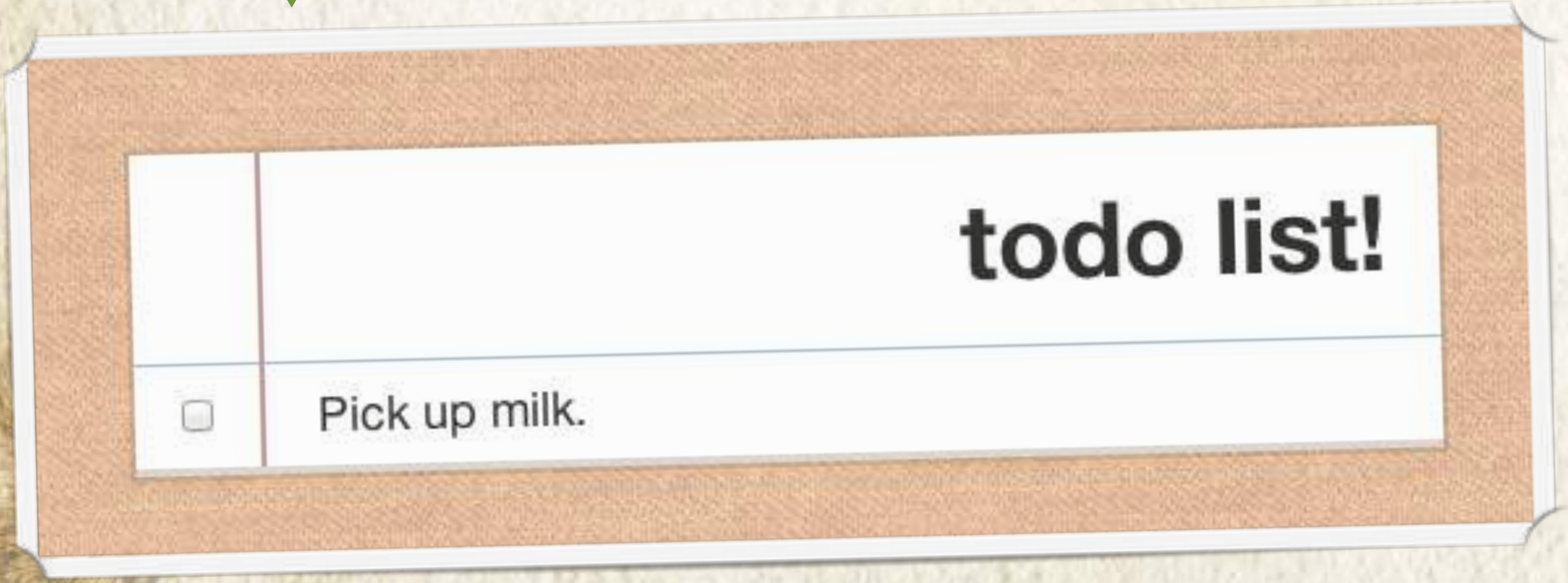
Introducing the Todo App

To get the todo data

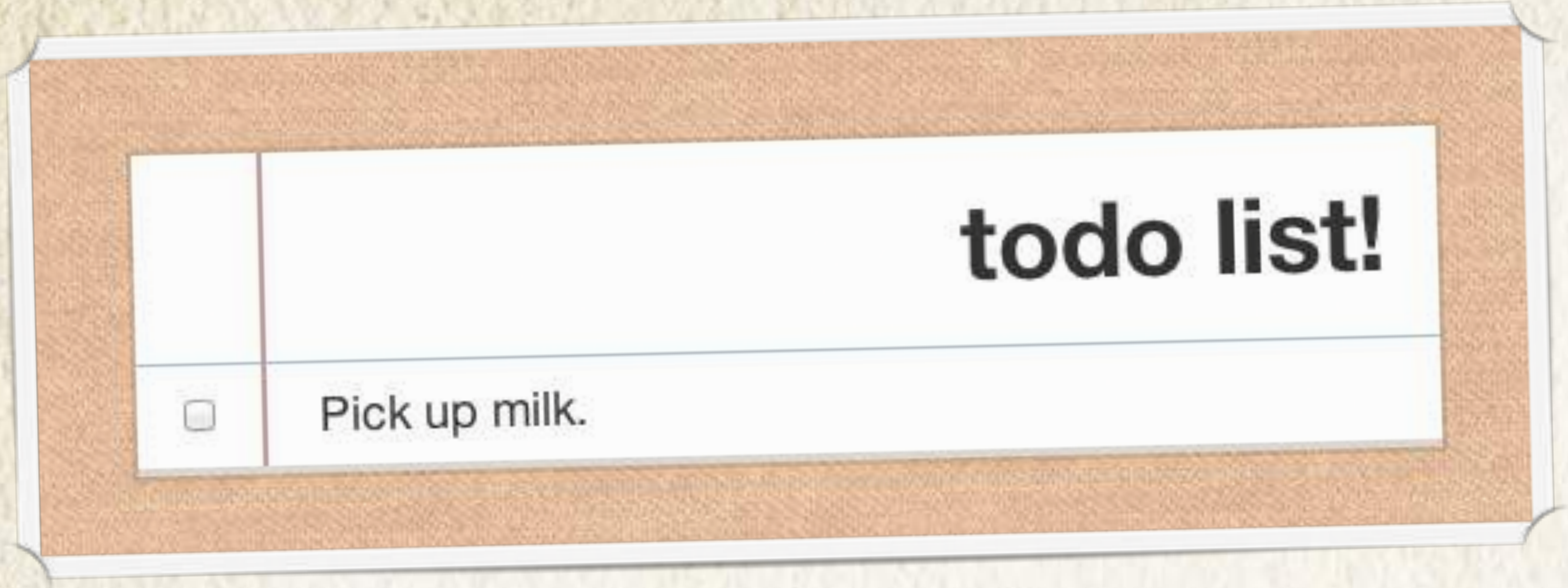
```
$.getJSON('/todo', function(data) {
```

The data returned

```
{ description: 'Pick up milk', status: 'incomplete', id: 1 }
```



Adding Functionality


todo list!

Pick up milk.

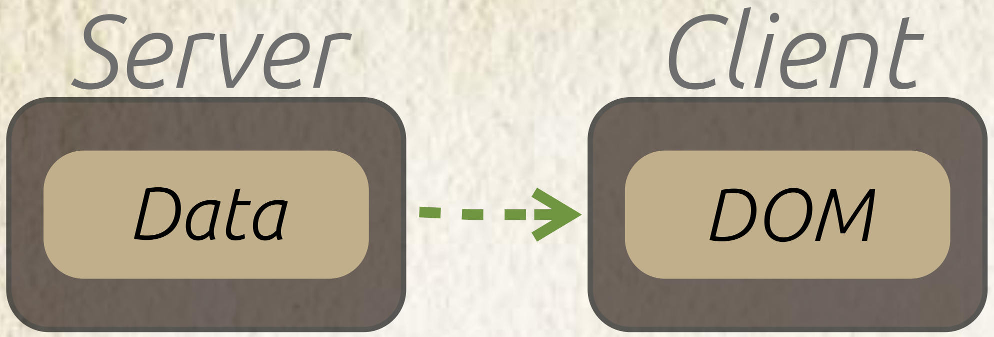
Checking off a todo item

Add deadlines

Reorder & sort

Methods can be disorganized
We lose the data structure

Without Backbone.js



```
{ description: 'Pick up milk', status: 'incomplete', id: 1 }
```

```
<h3 class='incomplete'>
  <input type='checkbox' data-id='1' />
  Pick up milk
</h3>
```

We need an object to maintain the data

Introducing Backbone.js

"Get your truth out of the DOM"

- Jeremy Ashkenas

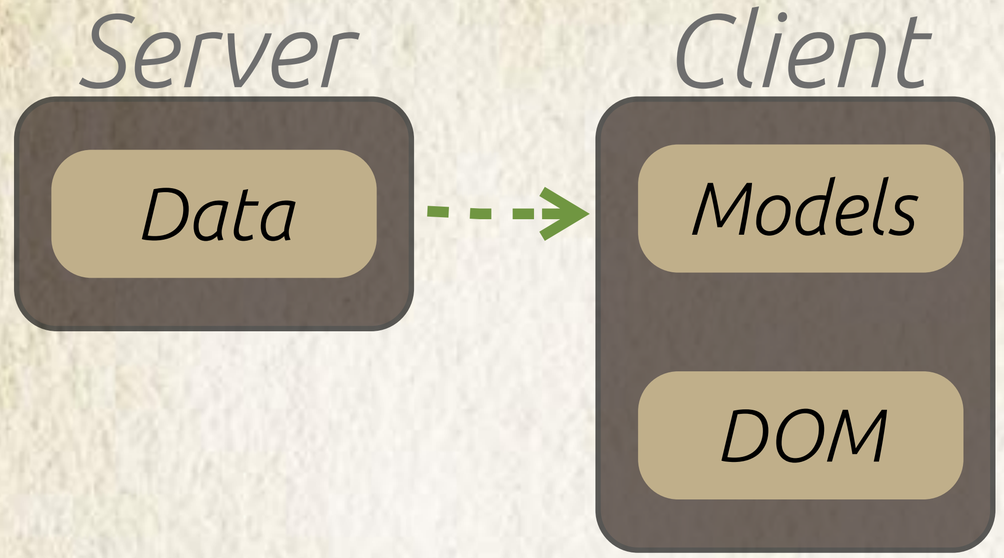
Provides client-side app structure

Models to represent data

Views to hook up models to the DOM

Synchronizes data to/from server

With Backbone.js



To create a model class

```
var TodoItem = Backbone.Model.extend({});
```

To create a model instance

```
var todoItem = new TodoItem(
  { description: 'Pick up milk', status: 'incomplete', id: 1 }
);
```

Backbone.js

Backbone Models

```
var todoItem = new TodoItem(
  { description: 'Pick up milk', status: 'incomplete', id: 1 }
);
```

To get an attribute

```
todoItem.get('description');
```

---> 'Pick up milk'

To set an attribute

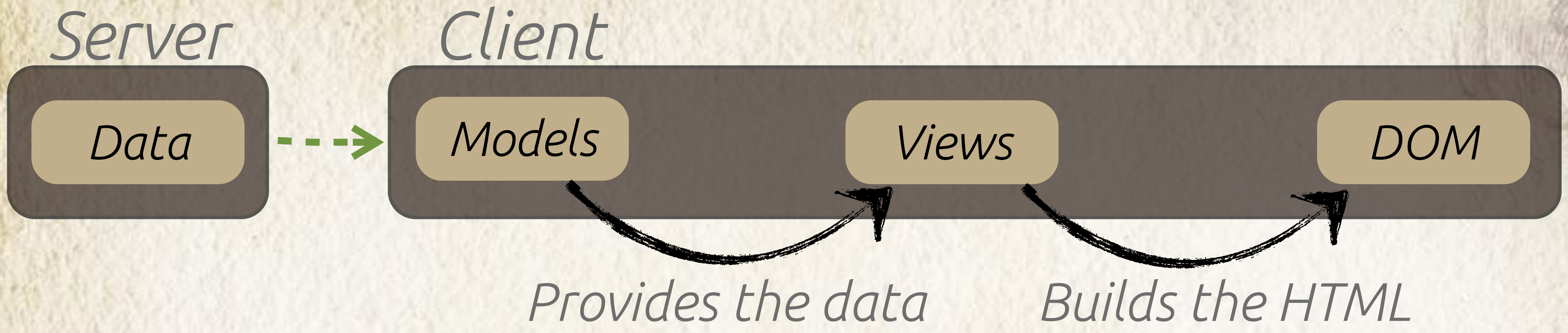
```
todoItem.set({status: 'complete'});
```

Sync to the server

```
todoItem.save();
```

! **Configuration needed**

Displaying the Data



To create a view class

```
var TodoView = Backbone.View.extend({});
```

To create a view instance

```
var todoView = new TodoView({ model: todoItem });
```

Rendering the View

```
var TodoView = Backbone.View.extend({
  render: function(){
    var html = '<h3>' + this.model.get('description') + '</h3>';
    $(this.el).html(html);
  }
});
```

Every view has a top level **EL**ement

<div>	<p>	<li>	<header>	<section>	...

default

Rendering the View

```
var TodoView = Backbone.View.extend({
  render: function(){
    var html = '<h3>' + this.model.get('description') + '</h3>';
    $(this.el).html(html);
  }
});
```

```
var todoView = new TodoView({ model: todoItem });
todoView.render();
console.log(todoView.el);
```

```
<div>
  <h3>Pick up milk</h3>
</div>
```

Models

- LEVEL 2 -

Reviewing Models

Generating a model class

```
var TodoItem = Backbone.Model.extend({});
```

Generating a model instance

```
var todoItem = new TodoItem(
  { description: 'Pick up milk', status: 'incomplete' }
);
```

To get an attribute

```
todoItem.get('description');
```

---> 'Pick up milk'

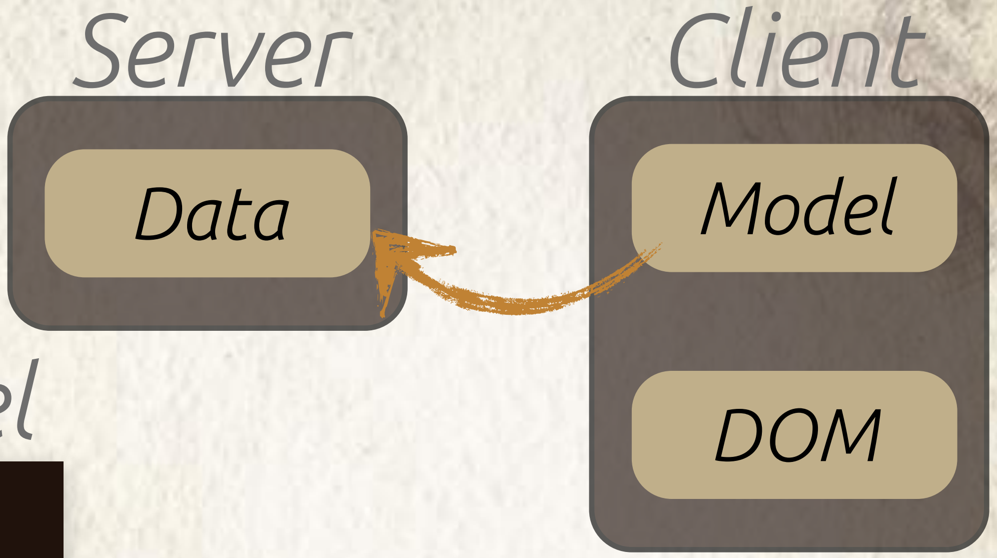
To set an attribute

```
todoItem.set({status: 'complete'});
```

Backbone.js

Fetching Data from the Server

```
var todoItem = new TodoItem();
```



URL to get JSON data for model

```
todoItem.url = '/todo';
```

To populate model from server

```
todoItem.fetch();
```

```
---> { id: 1, description: 'Pick up milk', status: 'incomplete' }
```

```
todoItem.get('description');
```

```
---> 'Pick up milk'
```

! /todo isn't a good URL

Backbone.js

Fetching Data from the Server

```
var TodoItem = Backbone.Model.extend({urlRoot: '/todos'});
```

RESTful web service (Rails flavor)

Fetch todo with id = 1

```
var todoItem = new TodoItem({id: 1})
```

```
todoItem.fetch();
```

GET /todos/1

```
---> { id: 1, description: 'Pick up milk', status: 'incomplete' }
```

Update the todo

```
todoItem.set({description: 'Pick up cookies.'});
```

```
todoItem.save();
```

PUT /todos/1
with JSON params

Creating & Destroying a New Todo

```
var todoItem = new TodoItem();
```

```
todoItem.set({description: 'Fill prescription.'});
```

```
todoItem.save();
```

POST /todos with JSON params

```
todoItem.get('id');
```

---> 2

```
todoItem.destroy();
```

DELETE /todos/2

Get JSON from model

```
todoItem.toJSON();
```

```
---> { id: 2, description: 'Fill prescription', status: 'incomplete' }
```

Default Values

```
var TodoItem = Backbone.Model.extend({
  defaults: {
    description: 'Empty todo...',
    status: 'incomplete'
  }
});
```

```
var todoItem = new TodoItem();
```

```
todoItem.get('description');
```

---> 'Empty todo...'

```
todoItem.get('status');
```

---> 'incomplete'

Models Can Have Events

To listen for an event on a model

```
todoItem.on('event-name', function(){
  alert('event-name happened!');
});
```

Run the event

```
todoItem.trigger('event-name');
```

Special Events

To listen for changes

```
todoItem.on('change', doThing);
```

```
var doThing = function() {
   ...
}
```

Event triggered on change

```
todoItem.set({description: 'Fill prescription.'});
```

Set without triggering event

```
todoItem.set({description: 'Fill prescription.'},
                                 {silent: true});
```

Remove event listener

```
todoItem.off('change', doThing);
```

Special Events

```
todoItem.on(<event>, <method>);
```

Built-in events

change	*When an attribute is modified*
change:<attr>	*When <attr> is modified*
destroy	*When a model is destroyed*
sync	*Whenever successfully synced*
error	*When model save or validation fails*
all	*Any triggered event*

Views

- LEVEL 3 -

More on the View Element

```
var SimpleView = Backbone.View.extend({});
var simpleView = new SimpleView();
```

```
console.log(simpleView.el);
```

```
---> <div></div>
```

```
var SimpleView = Backbone.View.extend({tagName: 'li'});
var simpleView = new SimpleView();
```

```
console.log(simpleView.el);
```

```
---> <li></li>
```

tagName can be any HTML tag

More on the View Element

```
var TodoView = Backbone.View.extend({
  tagName: 'article',
  id: 'todo-view',
  className: 'todo'
});
```

```
var todoView = new TodoView();
console.log(todoView.el);
```

---> `<article id="todo-view" class="todo"></article>`

More on the View Element

```
var todoView = new TodoView();
console.log(todoView.el);
```

```
---> <article id="todo-view" class="todo"></article>
```

I want to use a jQuery method

```
$('#todo-view').html();
```

!

el is a DOM Element

```
$(todoView.el).html();
```

Shortcut

```
todoView.$el.html();
```

✓

Good since the el's id may be dynamic

Back in Level 1

```
var TodoView = Backbone.View.extend({
  render: function(){
    var html = '<h3>' + this.model.get('description') + '</h3>';
    $(this.el).html(html);
  }
});
```

```
var todoView = new TodoView({ model: todoItem });
todoView.render();
console.log(todoView.el);
```

```
<div>
  <h3>Pick up milk</h3>
</div>
```

Adding the EL attributes

```
var TodoView = Backbone.View.extend({
  tagName: 'article',
  id: 'todo-view',
  className: 'todo',
  render: function(){
    var html = '<h3>' + this.model.get('description') + '</h3>';
    $(this.el).html(html);
  }
});
```

!

Fixing the EL

```
var TodoView = Backbone.View.extend({
  tagName: 'article',
  id: 'todo-view',
  className: 'todo',
  render: function(){
    var html = '<h3>' + this.model.get('description') + '</h3>';
    this.$el.html(html);
  }
});
```

```
$(this.el).html(html);
```

```
var todoView = new TodoView({ model: todoItem });
todoView.render();
console.log(todoView.el);
```

```
---> <article id="todo-view" class="todo">
       <h3>Pick up milk</h3>
     </article>
```

Using a Template

```
var TodoView = Backbone.View.extend({
  ...
  template: _.template('<h3><%= description %></h3>'),

  render: function(){
    var attributes = this.model.toJSON();
    this.$el.html(this.template(attributes));
  }
});
```

The underscore library

```
var todoView = new TodoView({ model: todoItem });
todoView.render();
console.log(todoView.el);
```

```
---> <article id="todo-view" class="todo">
       <h3>Pick up milk</h3>
     </article>
```

VIEWS

Backbone.js

Templating Engines

Underscore.js

```
<h3><%= description %></h3>
```

Mustache.js

```
<h3>{{description}}</h3>
```

Haml-js

```
%h3= description
```

Eco

```
<h3><%= @description %></h3>
```

Adding View Events

```
<h3><%= description %></h3>
```

In jQuery to add an alert on click

```
$("h3").click(alertStatus);

function alertStatus(e) {
  alert('Hey you clicked the h3!');
}
```

Not how we do things in Backbone

View Events

Views are responsible for responding to user interaction

```
var TodoView = Backbone.View.extend({
  events: {
    "click h3": "alertStatus"
  },

  alertStatus: function(e){
    alert('Hey you clicked the h3!');
  }

});
```

```
"<event> <selector>": "<method>"
```

Selector is scoped to the el

```
this.$el.delegate('h3', 'click', alertStatus);
```

Views Can Have Many Events

anywhere on EL

```
var DocumentView = Backbone.View.extend({

  events: {
    "dblclick"                    : "open",
    "click .icon.doc"             : "select",
    "click .show_notes"           : "toggleNotes",
    "click .title .lock"          : "editAccessLevel",
    "mouseover .title .date"      : "showTooltip"
  },
  ...

});
```

View Event Options

```
var SampleView = Backbone.View.extend({
  events: {
    "<event> <selector>": "<method>"
  },
  ...
});
```

Events

change	click	dblclick	focus	focusin
focusout	hover	keydown	keypress	load
mousedown	mouseenter	mouseleave	mousemove	mouseout
mouseover	mouseup	ready	resize	scroll
select	unload			

Models & Views

- LEVEL 4 -

Review our Model View

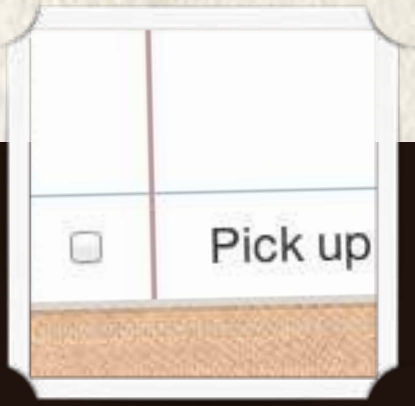
```
var TodoView = Backbone.View.extend({
  template: _.template('<h3><%= description %></h3>'),

  render: function(){
    this.$el.html(this.template(this.model.toJSON()));
  }
});
```

```
var todoView = new TodoView({ model: todoItem });
todoView.render();
console.log(todoView.el);
```

```
<div>
  <h3>Pick up milk</h3>
</div>
```

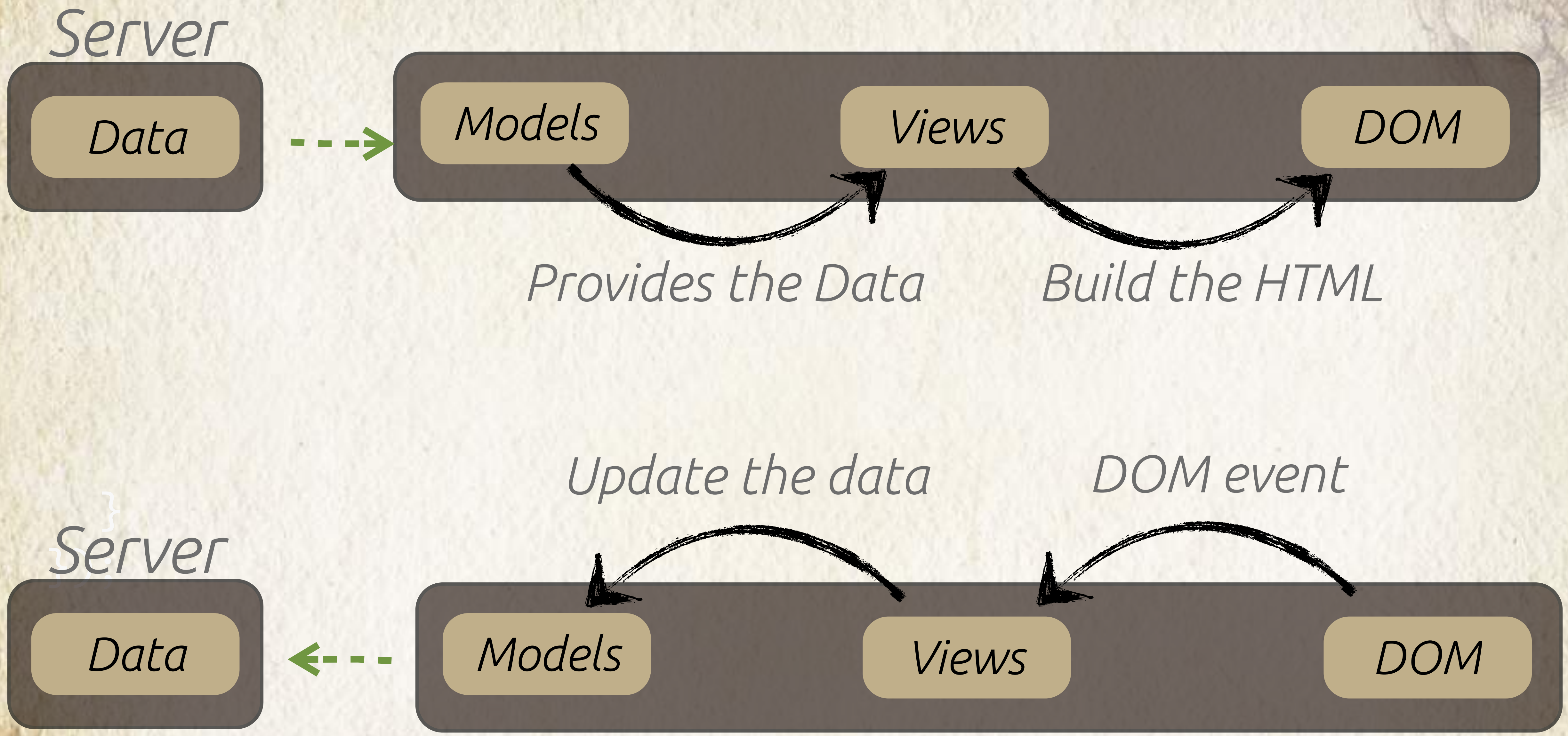
Adding a checkbox

```
var TodoView = Backbone.View.extend({
  template: _.template('<h3>' +
    '<input type=checkbox ' +
    '<% if(status === "complete") print("checked") %>/>' +
                                '<%= description %></h3>'),

  render: function(){
    this.$el.html(this.template(this.model.toJSON()));
  }
});
```



How do we update the model when checkbox changes?

View events update the Model



Backbone.js

Update model on UI event

```
var TodoView = Backbone.View.extend({

  events: {
    'change input': 'toggleStatus'
  },

  toggleStatus: function(){
    if(this.model.get('status') === 'incomplete'){
      this.model.set({'status': 'complete'});
    }else{
      this.model.set({'status': 'incomplete'});
    }
  }

});
```

Model logic in view

Refactor to the Model

```
var TodoView = Backbone.View.extend({
  events: {
    'change input': 'toggleStatus'
  },
  toggleStatus: function(){
   this.model.toggleStatus();
  }
});

var TodoItem = Backbone.Model.extend({
  toggleStatus: function(){
    if(this.get('status') === 'incomplete'){
      this.set({'status': 'complete'});
    }else{
      this.set({'status': 'incomplete'});
    }
  }
});
```

✓ Model logic in Model

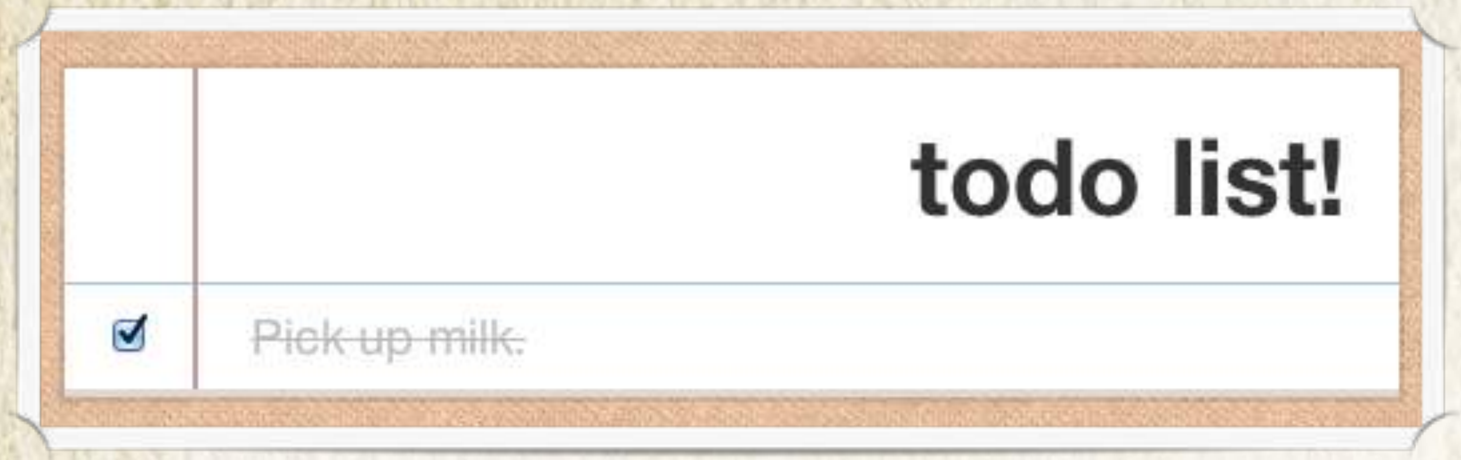
Sync changes to server

```
var TodoItem = Backbone.Model.extend({
  toggleStatus: function(){
    if(this.get('status') === 'incomplete'){
      this.set({'status': 'complete'});
    }else{
      this.set({'status': 'incomplete'});
    }

    this.save();
  }
});
```

```
PUT /todos/1
```

Backbone.js

Update view to reflect changes



todo list!

~~Pick up milk.~~

```
.complete {
  color: #bbb;
  text-decoration: line-through;
}
```

update TodoView template:

```
template: _.template('<h3 class="<%= status %>">' +
        '<% if(status === "complete") print("checked") %>/>' +
        ' <%= description %></h3>')
```

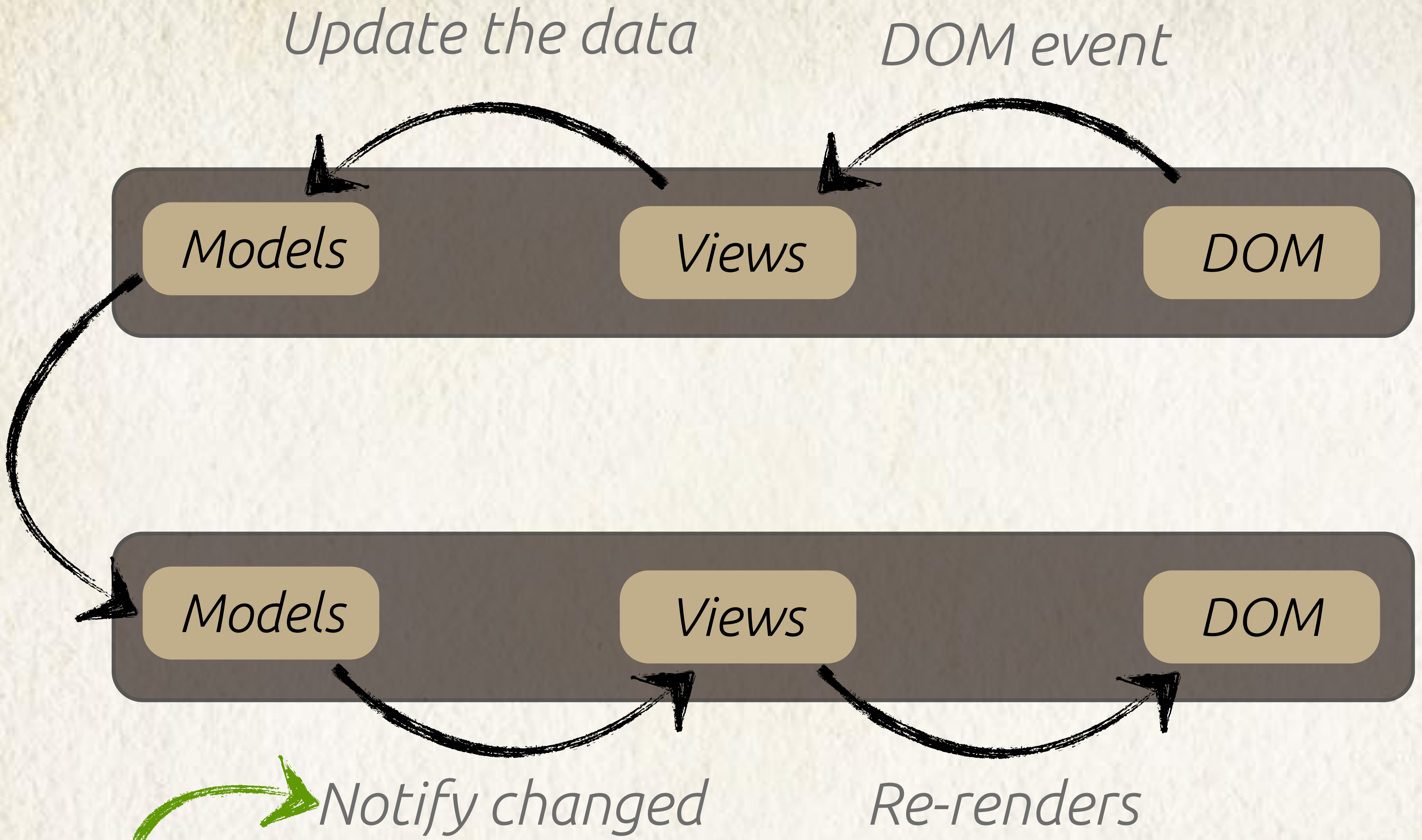
How should we update the view when the model changes?

Re-render the view

```
var TodoView = Backbone.View.extend({
  events: {
    'change input': 'toggleStatus'
  },
  toggleStatus: function(){
   this.model.toggleStatus();
   this.render();
  },
  render: function(){
   this.$el.html(this.template(this.model.toJSON()));
  }
});
```

✖ Doesn't work for other model changes

Model updates change the View



✓ Use Model Events

Backbone.js

Re-render the view

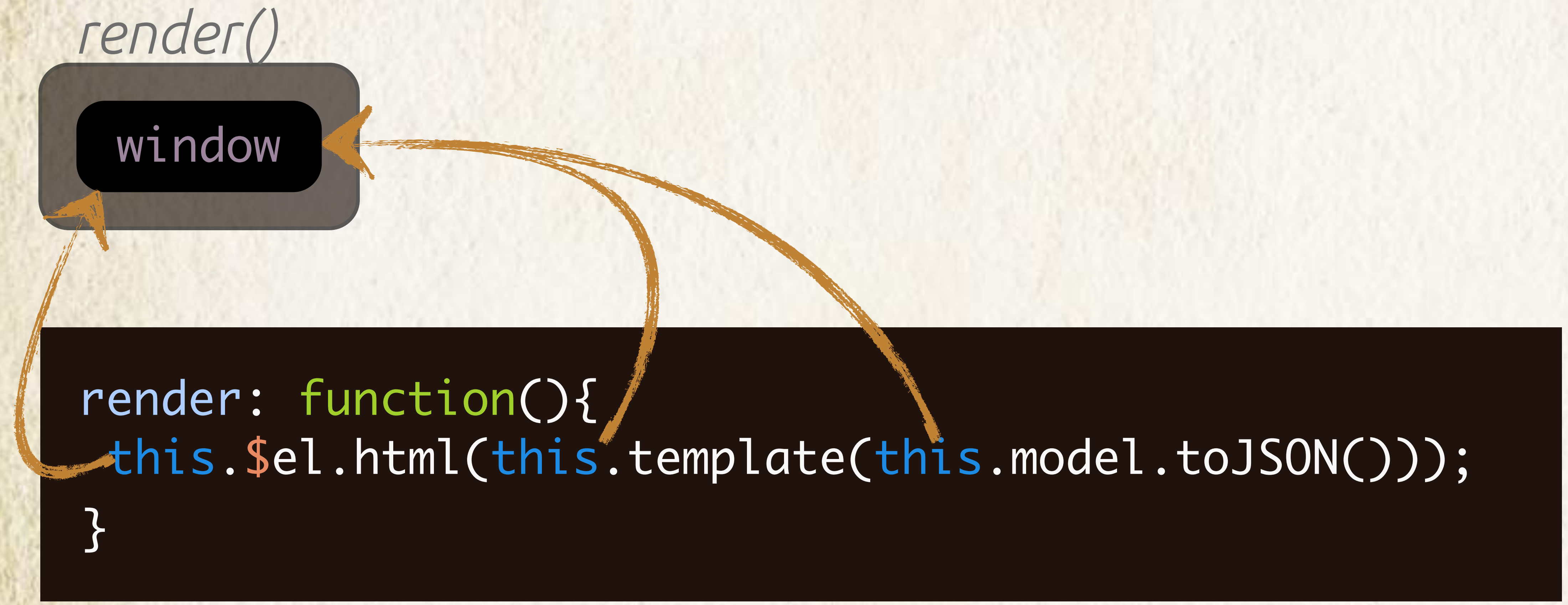
```
var TodoView = Backbone.View.extend({
  events: {
    'change input': 'toggleStatus'
  },
  initialize: function(){
    this.model.on('change', this.render, this);
  },
  toggleStatus: function(){
   this.model.toggleStatus();
  },

  render: function(){
   this.$el.html(this.template(this.model.toJSON()));
  }
});
```

✓

Why the third argument?

What is this?

```
this.model.on('change', this.render);
```

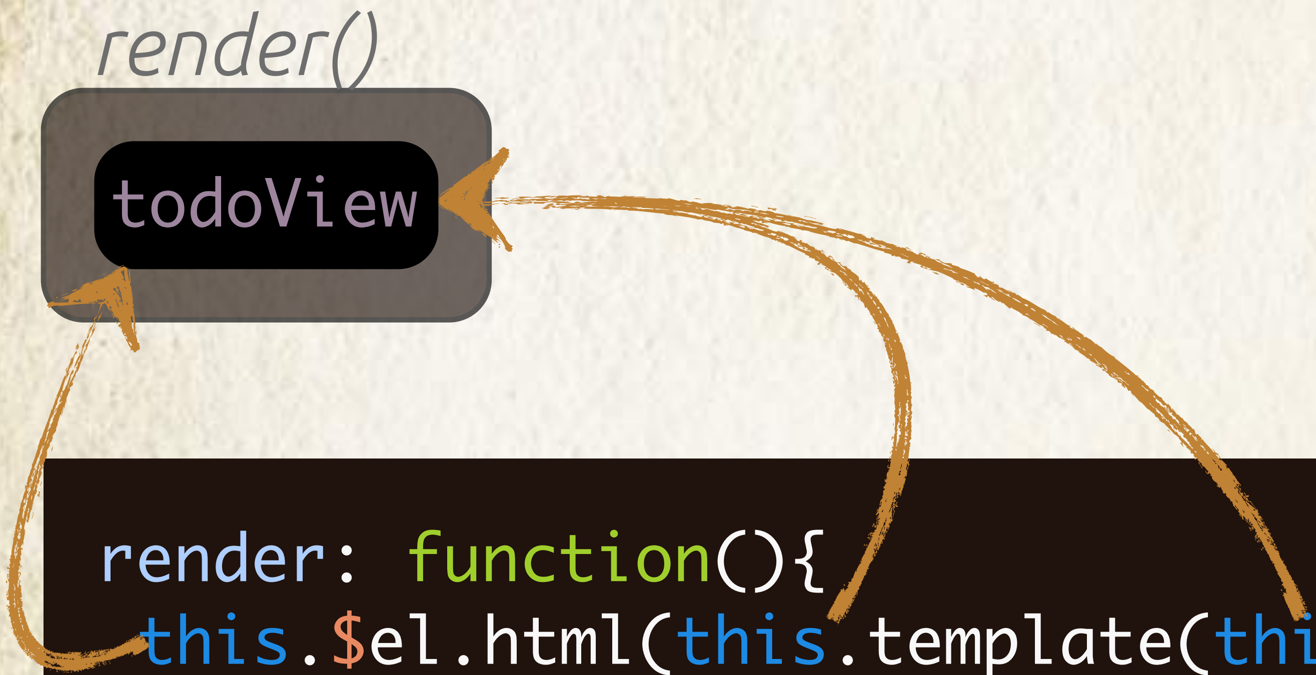


```
render: function(){
 this.$el.html(this.template(this.model.toJSON()));
}
```

render context is not the view

What is this?

```
this.model.on('change', this.render, this);
```



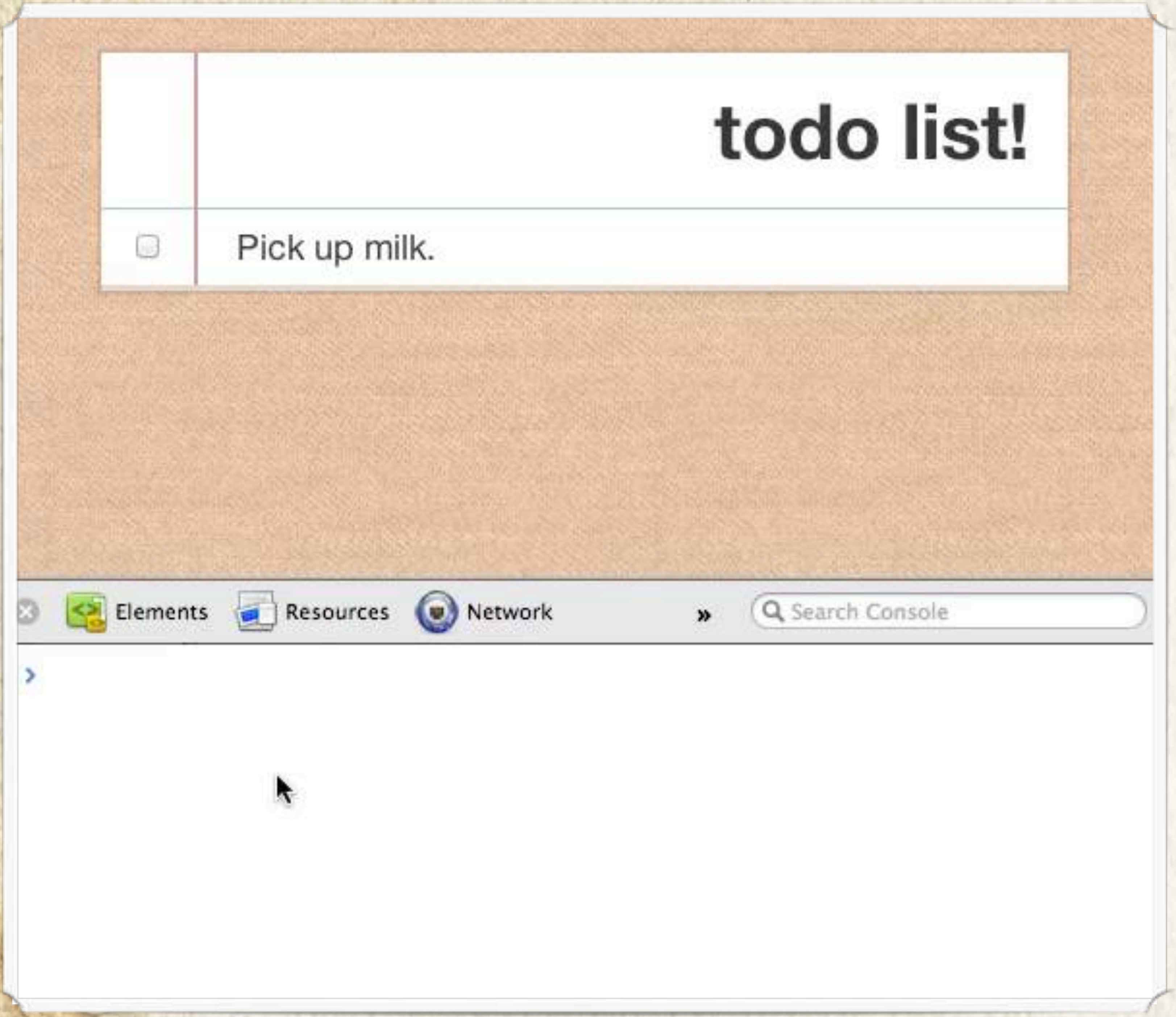
```
render: function(){
 this.$el.html(this.template(this.model.toJSON()));
}
```

✓ **render context is bound to the view**

Remove view on model destroy

```
var TodoView = Backbone.View.extend({
  initialize: function(){
    this.model.on('change', this.render, this);
    this.model.on('destroy', this.remove, this);
  },

  render: function(){
   this.$el.html(this.template(this.model.toJSON()));
  },

  remove: function(){
    this.$el.remove();
  }

});
```

Watch it in action


todo list!

Pick up milk.

Backbone.js

Collections

- LEVEL 5 -

Set of Models

```
var todoItem1 = new TodoItem();
var todoItem2 = new TodoItem();
var todoList = [todoitem1, todoItem2];
```

✗

```
var TodoList = Backbone.Collection.extend({
  model: TodoItem
});
var todoList = new TodoList();
```

✓

TodoList manages a set of TodoItem model instances

Add/Remove/Get

get number of models

```
todoList.length;
```

--→ 2

add a model instance

```
todoList.add(todoItem1);
```

get model instance at index 0

```
todoList.at(0);
```

--→ todoItem1

get by id 1

```
todoList.get(1);
```

--→ todoItem1

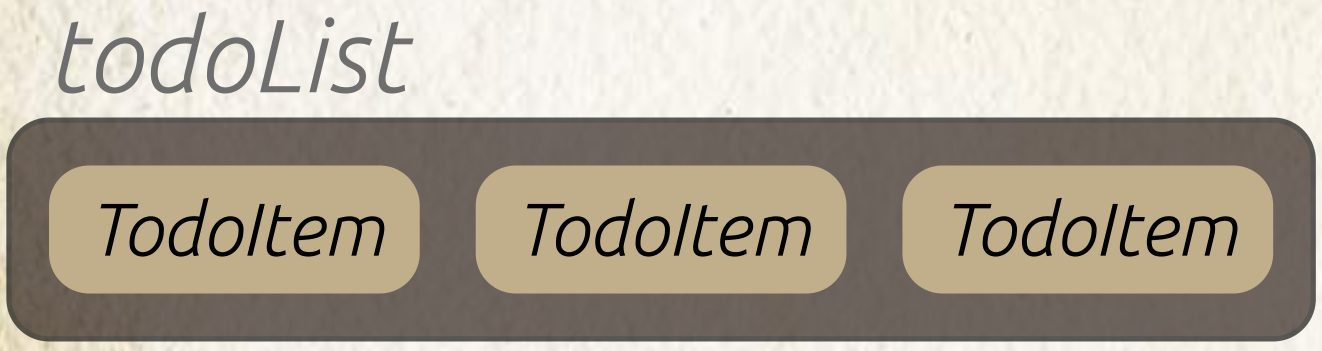
removing a model instance

```
todoList.remove(todoItem1);
```

Bulk Population

reset

```
var todos = [
  {description: 'Pick up milk.', status: 'incomplete'},
  {description: 'Get a car wash', status: 'incomplete'},
  {description: 'Learn Backbone', status: 'incomplete'}
];

todoList.reset(todos);
```



Each object in Array becomes a TodoItem

Fetching Data from the Server

URL to get JSON data from

```
var TodoList = Backbone.Collection.extend({
  url: '/todos'
});
```

populate collection from server

```
todoList.fetch();
```

GET /todos

```
[
  {description: 'Pick up milk.', status: 'incomplete', id: 1},
  {description: 'Get a car wash', status: 'incomplete', id: 2}
]
```

```
todoList.length;
```

---> 2

Collections Can Have Events

To listen for an event on a collection

```
todoList.on('event-name', function(){
  alert('event-name happened!');
});
```

Run the event

```
todoList.trigger('event-name');
```

✓ **Works just like models!**

Special Events

listen for reset

```
todoList.on('reset', doThing);
```

```
var doThing = function() {
  ...
}
```

Event triggered on reset & fetch

```
todoList.fetch();
todoList.reset();
```

without notification

```
todoList.fetch({silent: true});
todoList.reset({silent: true});
```

Remove event listener

```
todoList.off('reset', doThing);
```

Special Events on Collection

```
todoList.on(<event>, <function>);
```

Built-in Events

add	*When a model is added*
remove	*When a model is removed*
reset	*When reset or fetched*

```
todoList.on('add', function(todoItem){
  ...
});
```

todoItem is the model being added

Backbone.js

Model Events

Models in collection

change	*When an attribute is modified*
change:<attr>	*When <attr> is modified*
destroy	*When a model is destroyed*
sync	*Whenever successfully synced*
error	*When an attribute is modified*
all	*When an attribute is modified*

Events triggered on a model in a collection will also be triggered on the collection

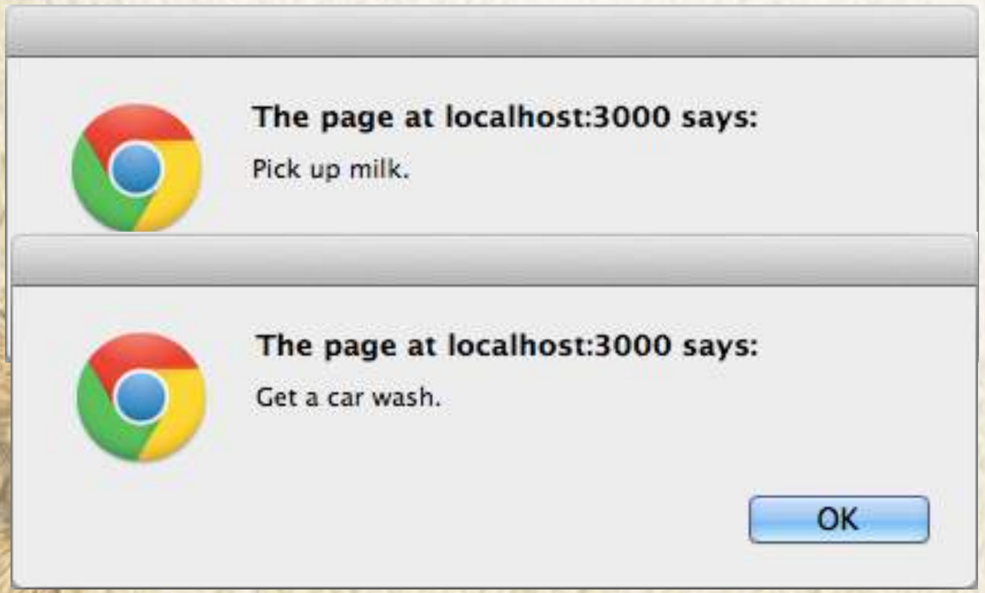
Iteration

Setup our collection

```
todoList.reset([
  {description: 'Pick up milk.', status: 'incomplete', id: 1},
  {description: 'Get a car wash.', status: 'complete', id: 2}
]);
```

Alert each model's description

```
todoList.forEach(function(todoItem){
  alert(todoItem.get('description'));
});
```



The page at localhost:3000 says:

Pick up milk.

The page at localhost:3000 says:

Get a car wash.

Iteration continued

Build an array of descriptions

```
todoList.map(function(todoItem){
  return todoItem.get('description');
});
```

```
---> ['Pick up milk.', 'Get a car wash']
```

Filter models by some criteria

```
todoList.filter(function(todoItem){
  return todoItem.get('status') === "incomplete";
});
```

Returns array of items that are incomplete

Other Iteration functions

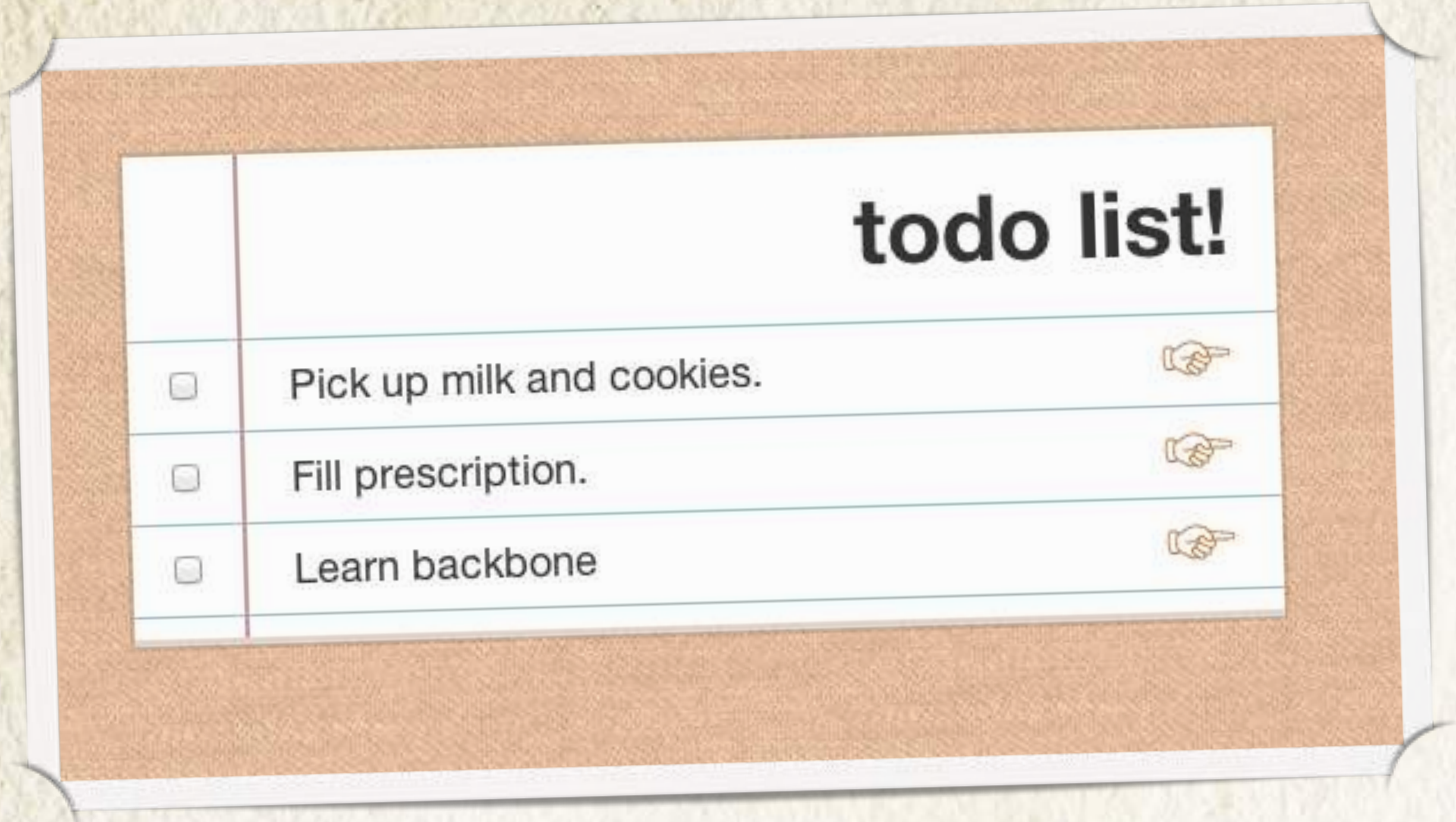
forEach	reduce	reduceRight
find	filter	reject
every	all	some
include	invoke	max
min	sortBy	groupBy
sortedIndex	shuffle	toArray
size	first	initial
rest	last	without
indexOf	lastIndexOf	isEmpty
chain		

http://documentcloud.github.com/backbone/#Collection-Underscore-Methods

Collections & Views

- LEVEL 6 -

Todo List!



todo list!

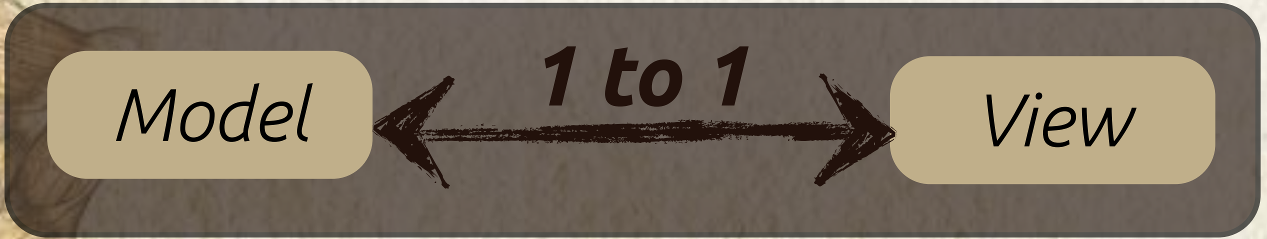
- Pick up milk and cookies.
- Fill prescription.
- Learn backbone

Collection + View == Collection View!

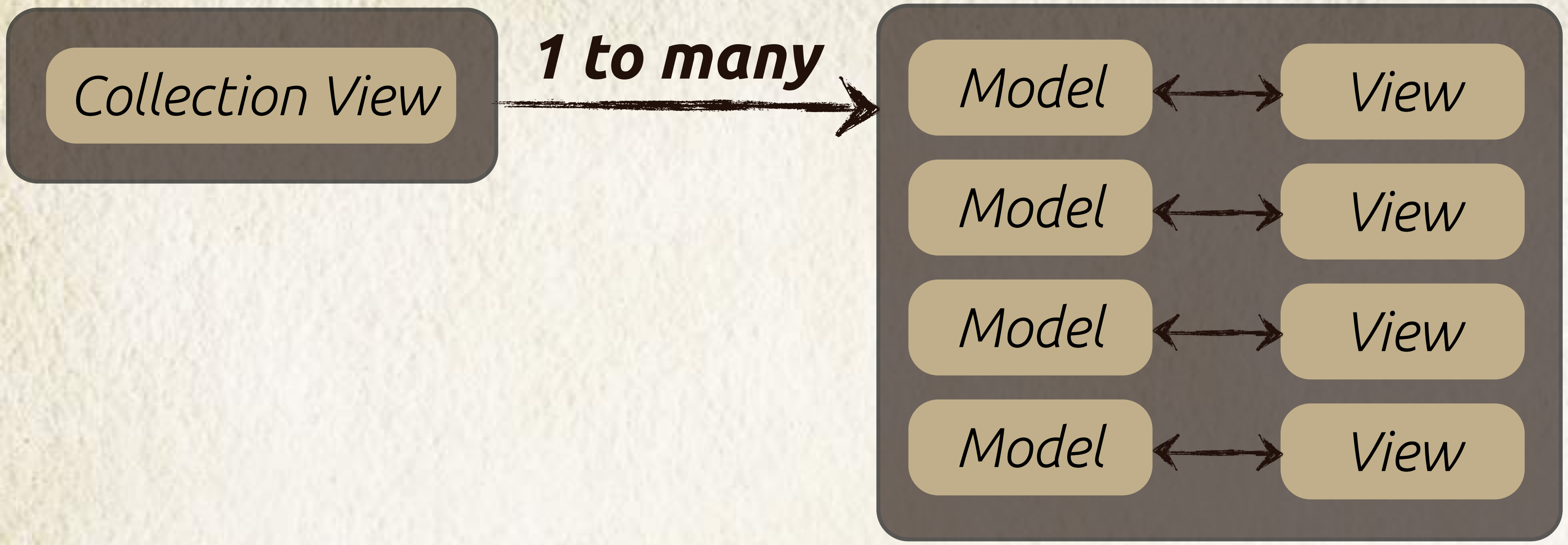
Review our Model View

```
var TodoView = Backbone.View.extend({
  render: function(){
    this.$el.html(this.template(this.model.toJSON()));
    return this;
  }
  ...
});
var todoItem = new TodoItem();
var todoView = new TodoView({model: todoItem});
console.log(todoView.render().el);
```

```
<div>
  <h3>Pick up milk</h3>
</div>
```



Collection Views



A Collection View doesn't render any of it's own HTML. It delegates that responsibility to the model views.

Define and Render

```
var TodoListView = Backbone.View.extend({});
var todoListView = new TodoListView({collection: todoList});
```

first crack at render

```
render: function(){
  this.collection.forEach(function(todoItem){
    var todoView = new TodoView({model: todoItem});
    this.$el.append(todoView.render().el);
  });
}
```

✗ **forEach changes context**

Rendering

addOne

```
render: function(){
  this.collection.forEach(this.addOne, this);
}

addOne: function(todoItem){
  var todoView = new TodoView({model: todoItem});
  this.$el.append(todoView.render().el);
}
```

✓ **forEach saves context**

Render continued

```
var todoListView = new TodoListView({collection: todoList});
todoListView.render();
console.log(todoListView.el);
```

```
<div>
  <h3 class="incomplete">
    <input type=checkbox />
    Pick up milk.
  </h3>

  <h3 class="complete">
    <input type=checkbox checked/>
    Learn backbone.
  </h3>
</div>
```

Adding new Models

```
var TodoListView = Backbone.View.extend({
  addOne: function(todoItem){
    var todoView = new TodoView({model: todoItem});
    this.$el.append(todoView.render().el);
  },
  render: function(){
    this.collection.forEach(this.addOne, this);
  }
});
```

```
var newTodoItem = new TodoItem({
  description: 'Take out trash.',
  status: 'incomplete'
});
todoList.add(newTodoItem);
```

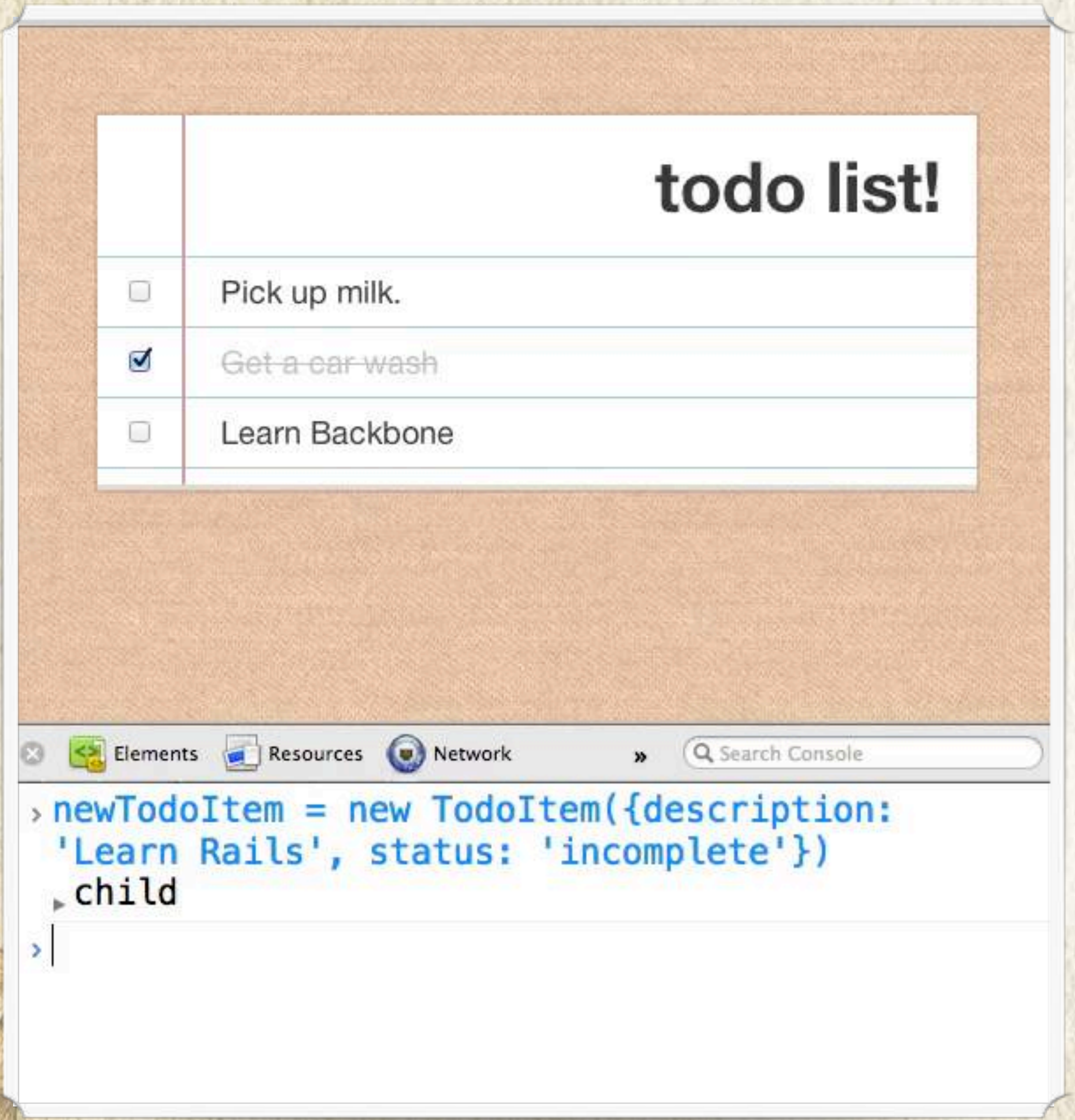
✗ **newTodoItem not in DOM**

Listen to the add Event

```
var TodoListView = Backbone.View.extend({
  initialize: function(){
    this.collection.on('add', this.addOne, this);
  },
  addOne: function(todoItem){
    var todoView = new TodoView({model: todoItem});
    this.$el.append(todoView.render().el);
  },
  render: function(){
    this.collection.forEach(this.addOne, this);
  }
});
```

```
var newTodoItem = new TodoItem({
  description: 'Take out trash.',
  status: 'incomplete'
});
todoList.add(newTodoItem);
```

Add Event in Action

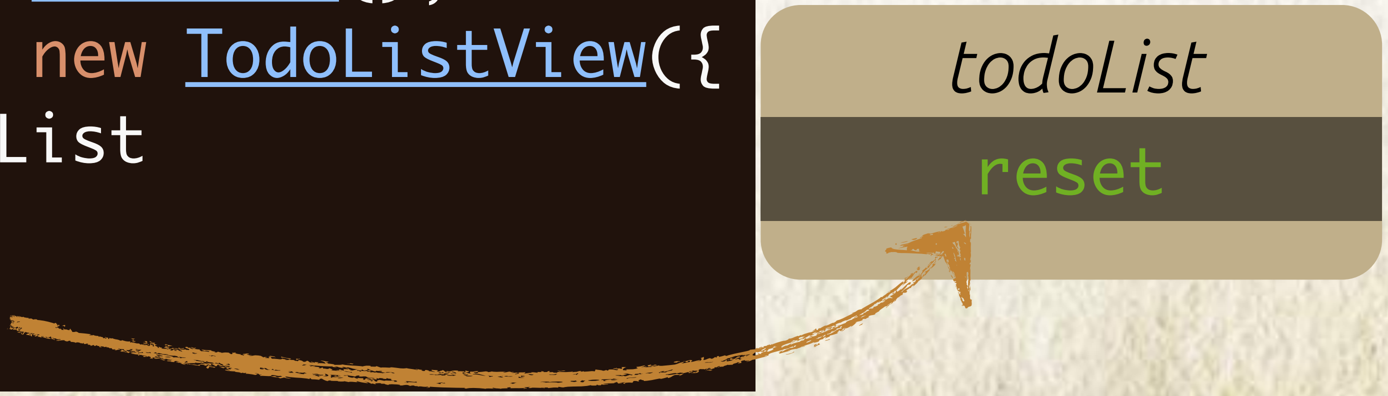


```
› newTodoItem = new TodoItem({description:
'Learn Rails', status: 'incomplete'})
  ▸ child
›
```

Reset Event

```
var TodoListView = Backbone.View.extend({
  initialize: function(){
    this.collection.on('add', this.addOne, this);
  },
  addOne: function(todoItem){
    var todoView = new TodoView({model: todoItem});
    this.$el.append(todoView.render().el);
  },
  render: function(){
    this.collection.forEach(this.addOne, this);
  }
});
```

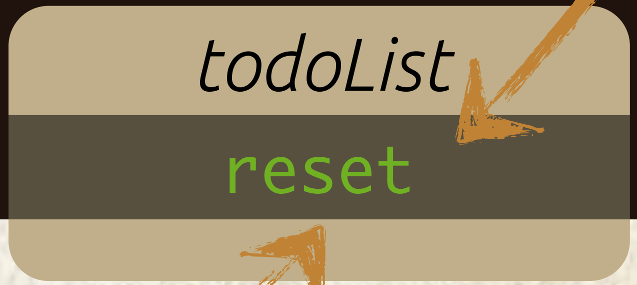
```
var todoList = new TodoList();
var todoListView = new TodoListView({
  collection: todoList
});
todoList.fetch();
```



Reset Event

```
var TodoListView = Backbone.View.extend({
  initialize: function(){
    this.collection.on('add', this.addOne, this);
    this.collection.on('reset', this.addAll, this);
  },
  addOne: function(todoItem){
    var todoView = new TodoView({model: todoItem});
    this.$el.append(todoView.render().el);
  },
  addAll: function(){
   this.collection.forEach(this.addOne, this);
  },
  render: function(){
    this.addAll();
  }
});
```

```
todoList.fetch();
```



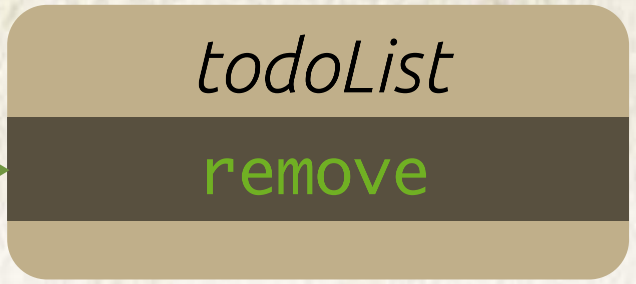
Reset Event in Action

!

removing from collection

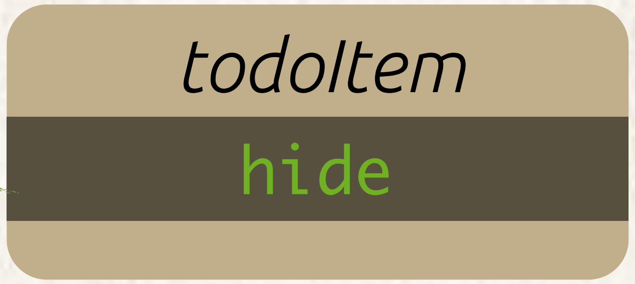
Fixing remove with Custom Events

```
todoList.remove(todoItem);
```



TodoList Collection

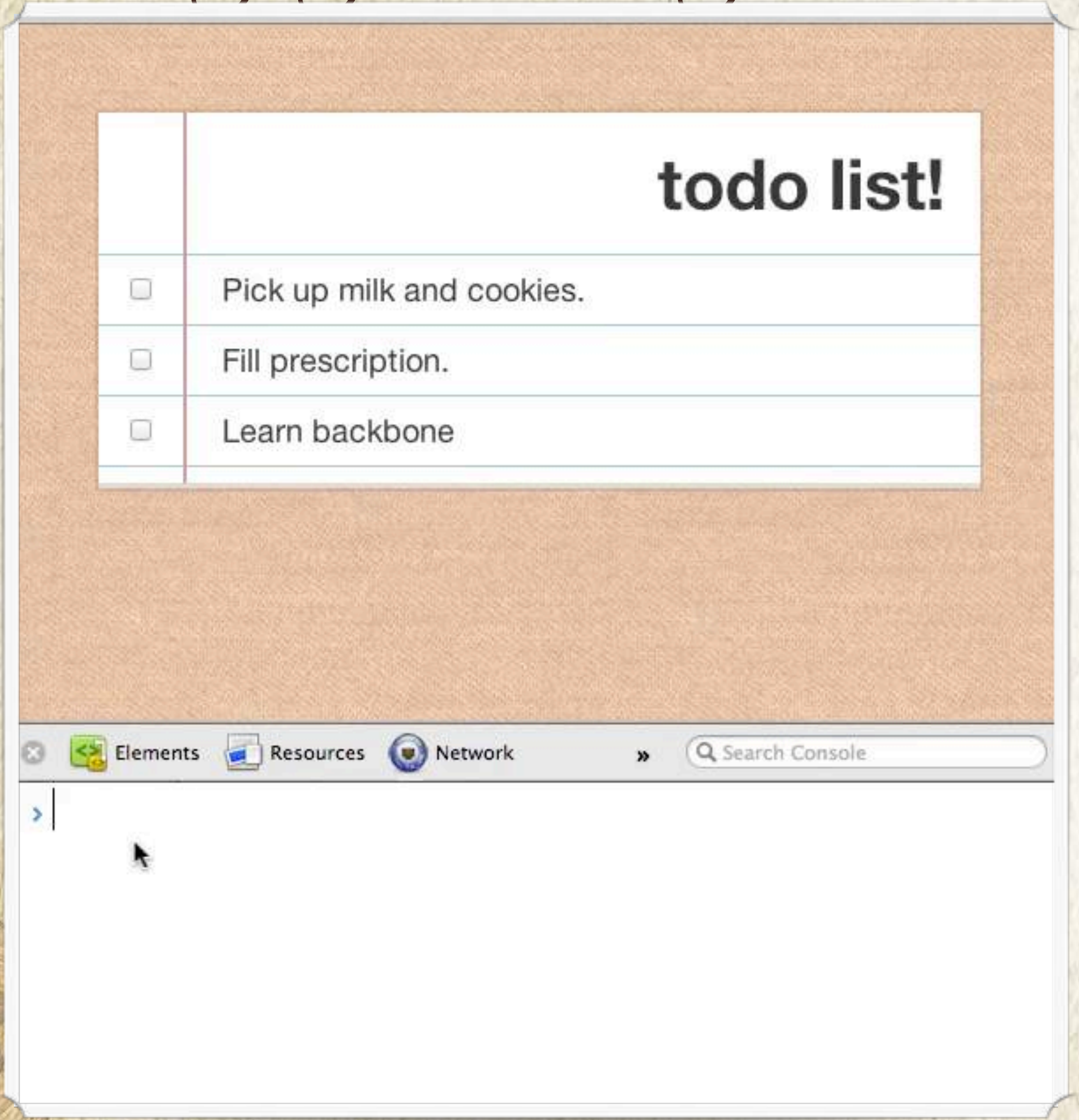
```
initialize: function(){
  this.on('remove', this.hideModel);
},
hideModel: function(model){
  model.trigger('hide');
}
```



TodoItem View

```
initialize: function(){
  this.model.on('hide', this.remove, this);
}
```

Bringing it all together

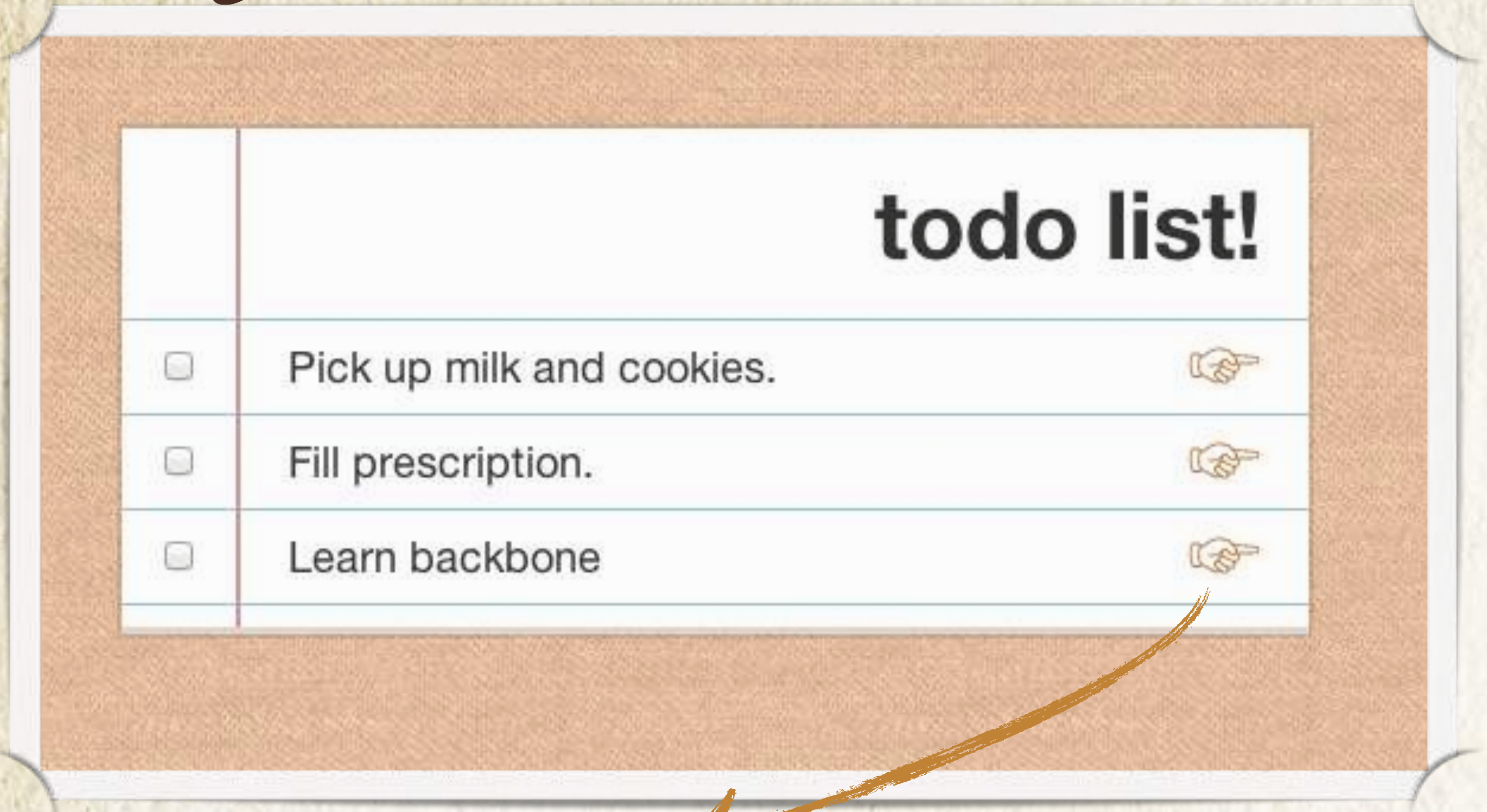

todo list!

- Pick up milk and cookies.
- Fill prescription.
- Learn backbone

Router and History

- LEVEL 7 -

The Problem


todo list!

Pick up milk and cookies.

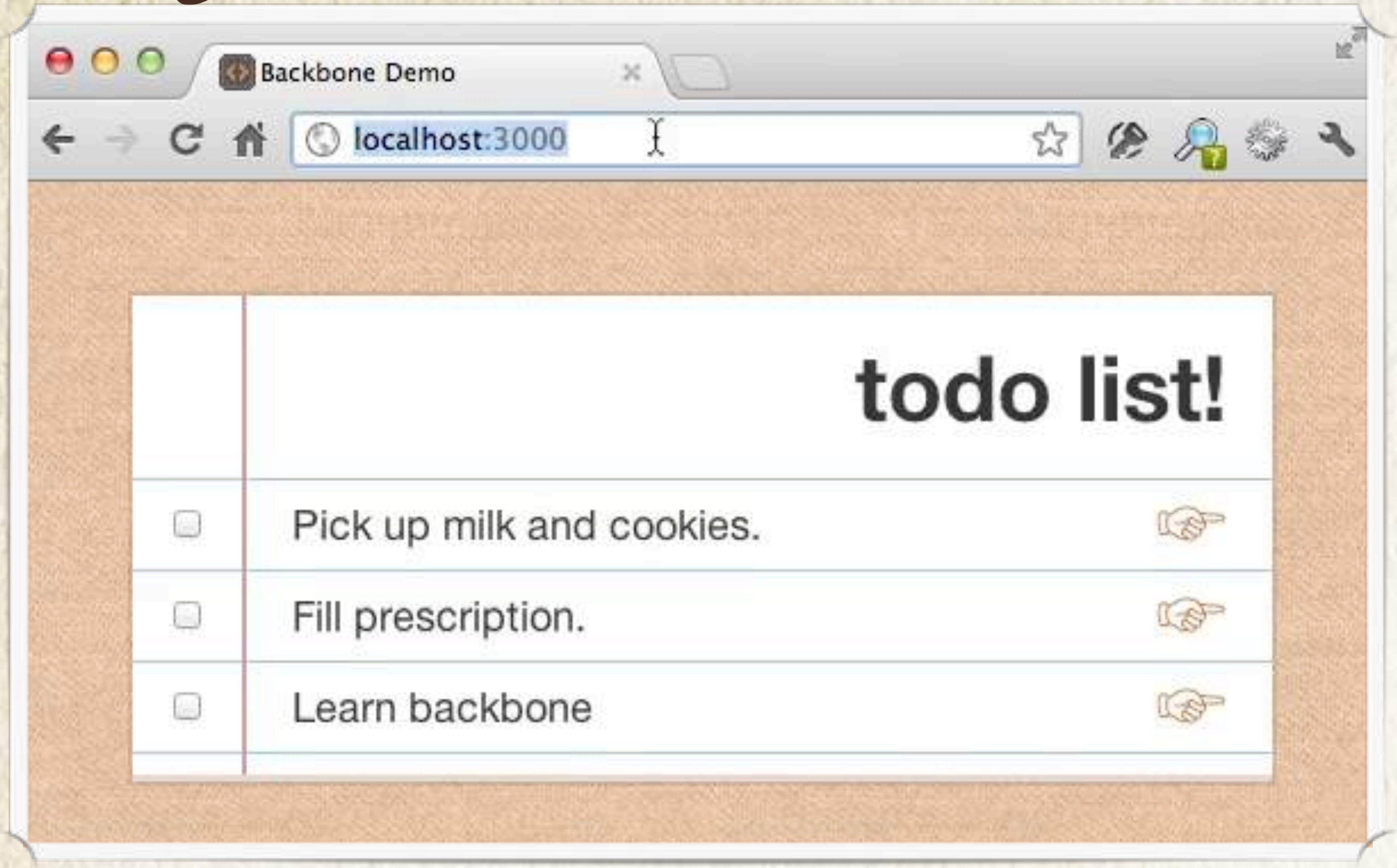
Fill prescription.

Learn backbone

```
<a href='#' class='todo'></a>
```

```
$('a.todo').click(function(e){
  e.preventDefault();
  // show single todo
})
```

The Problem



Backbone Demo

localhost:3000

todo list!

Pick up milk and cookies.

Fill prescription.

Learn backbone

Router & History to the rescue!

The Router

Router's map URLs to actions

```
var router = new Backbone.Router({
  routes: { "todos": 'index' },

  index: function(){
    ...
  }
});
```

when the url path is

/todos *or* #todos

```
index: function(){ ... }
```

Backbone.js

The Router

Routes match parameter parts

```
var router = new Backbone.Router({
  routes: { "todos/:id": 'show' }

  show: function(id){ ... }
})
```

Matches

URL	params
/todos/1	*id = 1*
/todos/2	*id = 2*
/todos/hello	*id = 'hello'*
/todos/foo-bar	*id = 'foo-bar'*

More Route Matchers

matcher	URL	params
search/:query	search/ruby	*query = 'ruby'*
search/:query/p:page	search/ruby/p2	*query = 'ruby', page = 2*
folder/:name-:mode	folder/foo-r	*name = 'foo', mode = 'r'*
file/*path	file/hello/world.txt	*path = 'hello/world.txt'*

*** Wildcard matches everything after file/**

Triggering Routes

Using navigate

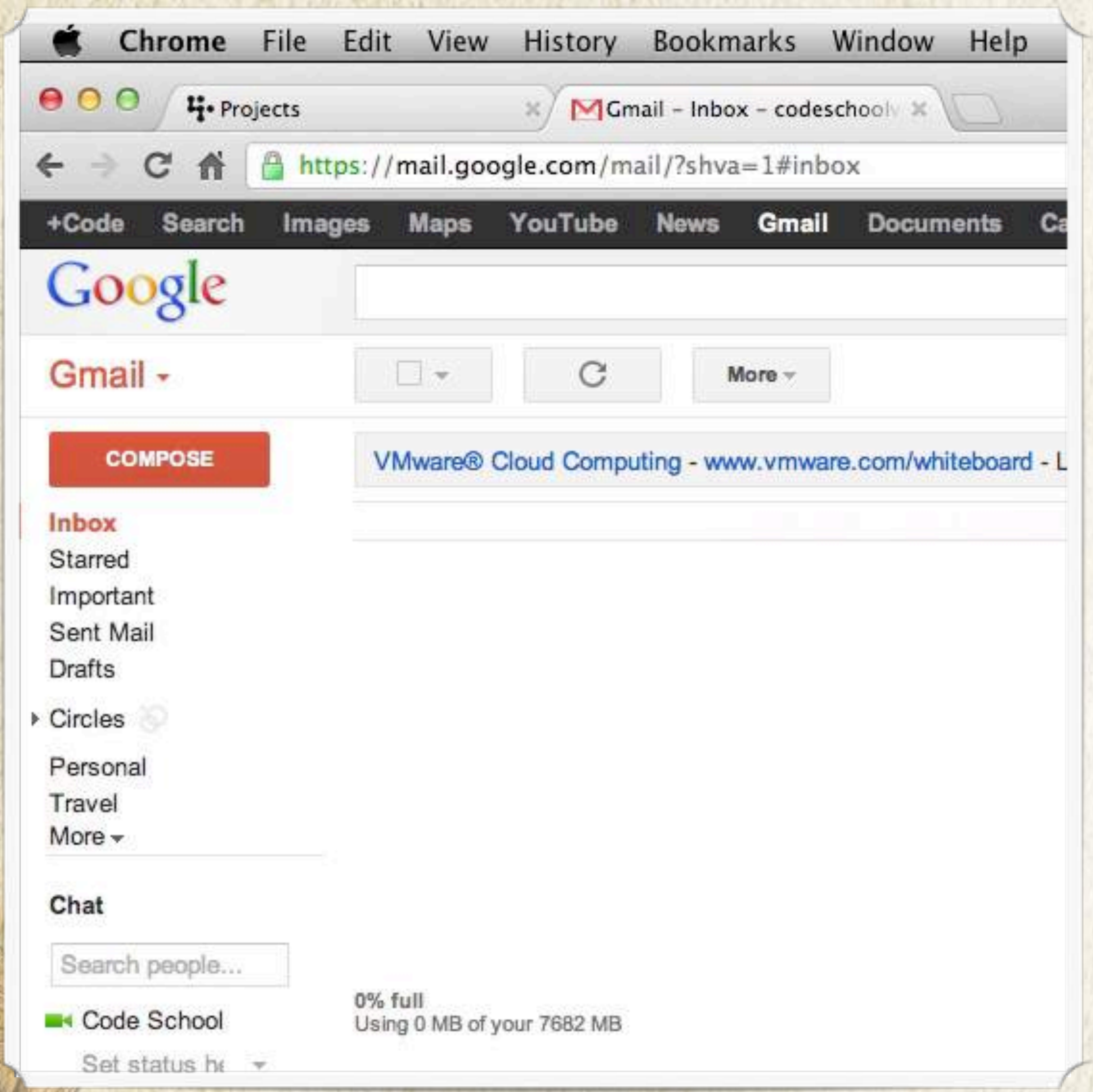
```
router.navigate("todos/1", {
  trigger: true
});
```

Using links

```
<a href='#todos/1'> ☞ </a>
```

! **These won't work yet**

History



https://mail.google.com/mail/?shva=1#inbox

VMware® Cloud Computing - www.vmware.com/whiteboard

0% full
Using 0 MB of your 7682 MB

$$\begin{tikzcd}\text{Hashbangs (\#)} \arrow[d, dashed] \\ \text{HTML5 pushState}\end{tikzcd}$$

Backbone.history

pushState off

```
Backbone.history.start();
```

```
router.navigate("todos/1")
```

✗ #todos/1

pushState on

```
Backbone.history.start({pushState: true});
```

```
router.navigate("todos/1")
```

✓ /todos/1

Show Action

Define router class

```
var TodoRouter = Backbone.Router.extend({
  routes: { "todos/:id": "show" },
  show: function(id){
    this.todoList.focusOnTodoItem(id);
  },
  initialize: function(options){
    this.todoList = options.todoList;
  }
});
```

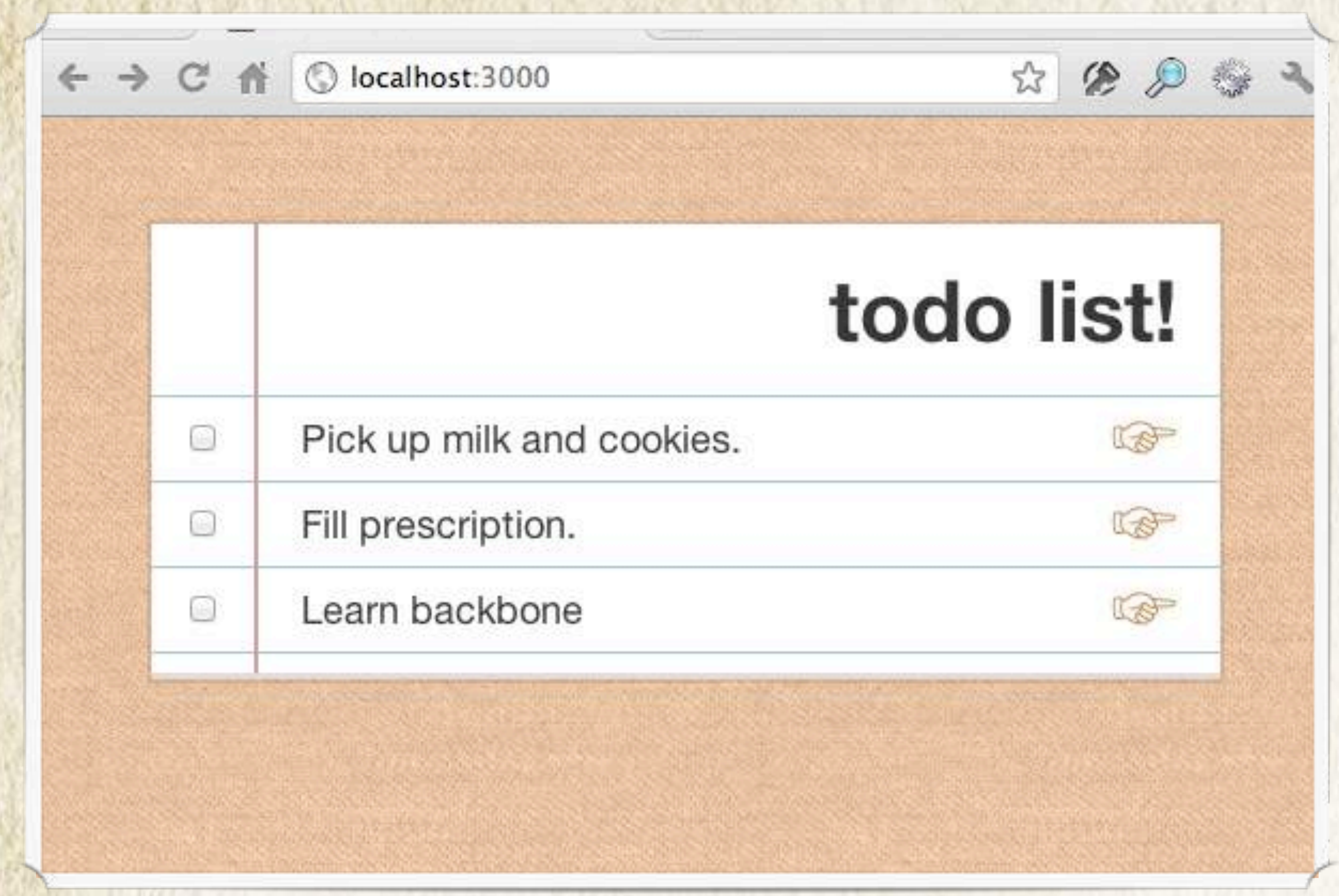
Instantiate router instance

```
var todoList = new TodoList();
var TodoApp = new TodoRouter({todoList: todoList});
```

Index Action

```
var TodoRouter = Backbone.Router.extend({
  routes: { "": "index",
            "todos/:id": "show" },

  index: function(){
    this.todoList.fetch();
  },
  show: function(id){
    this.todoList.focusOnTodoItem(id);
  },
  initialize: function(options){
    this.todoList = options.todoList;
  }
});
```

In Action


localhost:3000

todo list!

- Pick up milk and cookies.
- Fill prescription.
- Learn backbone

App Organization

```
var TodoApp = new (Backbone.Router.extend({
  routes: { "": "index", "todos/:id": "show" },
  initialize: function(){
    this.todoList = new TodoList();
    this.todosView = new TodoListView({collection: this.todoList});
    $('#app').append(this.todosView.el);
  },
  start: function(){
    Backbone.history.start({pushState: true});
  },
  index: function(){
    this.todoList.fetch();
  },
  show: function(id){
    this.todoList.focusOnTodoItem(id);
  }
}));
```

```
$(function(){ TodoApp.start() })
```