Vue.js

Reactive Components for Modern Web Interfaces

”Vue.js – Progressive enhancement workshop”

Progressive library

Progressive enhancement

Single Page Application

Universal Application (2.0)

Why use Vue over JQuery?

Solves different concerns

Reason with ***state***

Awesome ***developer tools***

Progressively ***adapt integration*** with ***scalability***

Why not use React or Angular?

React/Angular solutions for larger structure applications

Don’t tie well with Progressive Enhancement principles

Larger ***learning curves***

Angular 1, is dirty, it makes me cry at night. So don’t use it!

What does Vue deliver?

Library for the ***View*** layer

Components with ***reactivity***

Extendable via ***plugins***

Lightweight

Simplicity

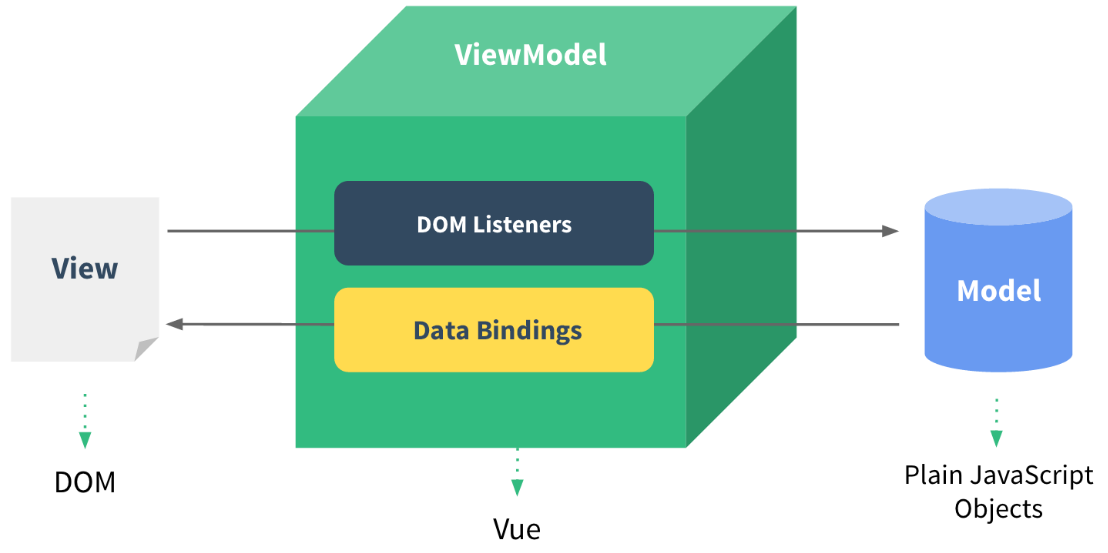
How does Vue.js work?

How does Vue.js work?

Reactive data-binding system for a ***Data-driven view***

DOM in sync with data

Object.defineProperty

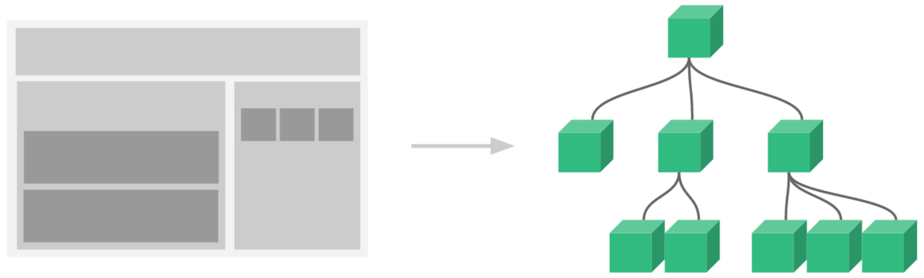


How does Vue.js work?

Component System helps with ***small abstraction layer***

Component loosely modeled after the ***Web Components spec***

Implements the ***Slot API*** and the special attribute ***is***



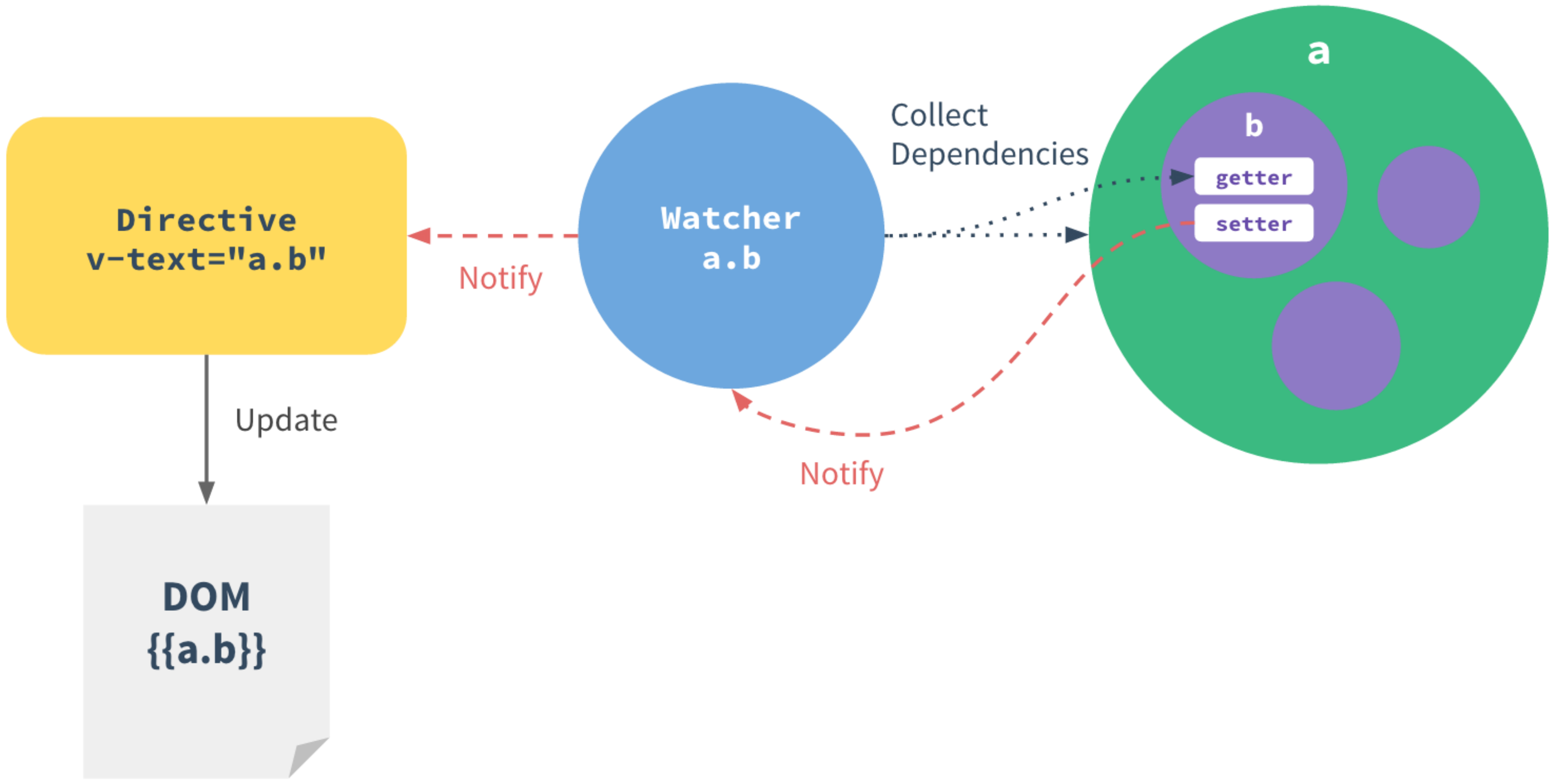
How does Vue.js work?

Object.defineProperty

Creates **getters** and **setters** which enables:

- **Dependency-tracking** and **change-notification** during access and modification

What does reactivity look like?



Reactivity – the caveats...

There are some caveats…

As there is a limitation of ES5 – it is not possible to detect detect **additions** and **deletions**

Fixed by: *vm.$set()* and *Vue.set()*

Best Practice: Declare data structures

Asynchronous DOM

DOM updates ***asynchronously***

When a change is detected ***Buffer*** is created

Further changes are processed into the buffer

Next “tick” – buffer flushed performing only the necessary DOM updates

$nextTick mechanism to wait until DOM data change

Computed properties

Keeps track of it's ***own reactive dependencies***

Caches it's evaluated result value

When one of it's dependencies changes, it revaluates otherwise it uses the cached value

Features

Features

- Data Binding Syntax
- Modifiers
- Components
- Dynamic components
- Asynchronous components
- Component lifecycle
- Directives
- Methods and event handling
- Filters
- Mixins
- Transitions
- Hot reloading
- Plugins
- Global & Local registration

Data Binding

Data-binding

One way binding by default!

Simple interpolation

v-bind and ***v-on***

Style property auto prefixing

Data-binding - Interpolation

```
{{ msg }}   //Double curly braces:

{{ * msg }}   //Never change from first value

<div id="item-{{ id }}"> </div>    // use inside attributes:
```

Data-binding – v-bind

Reactively bind data to attributes

```
<a v-bind:href="url"> </a>

<a :href="url"> </a>
```

Data-binding – v-on

Listen to DOM events calling component methods with v-on

```
<a v-on:click="method"></a>

<a @click="method"></a>
```

Data-binding – Modifiers

Directives can have modifiers

```
<a v-on:click.prevent="method"></a>

<a @click.prevent="method"></a>
```

Components

Components – Basics

High-level they are simply ***custom elements***

Internal lifecycle

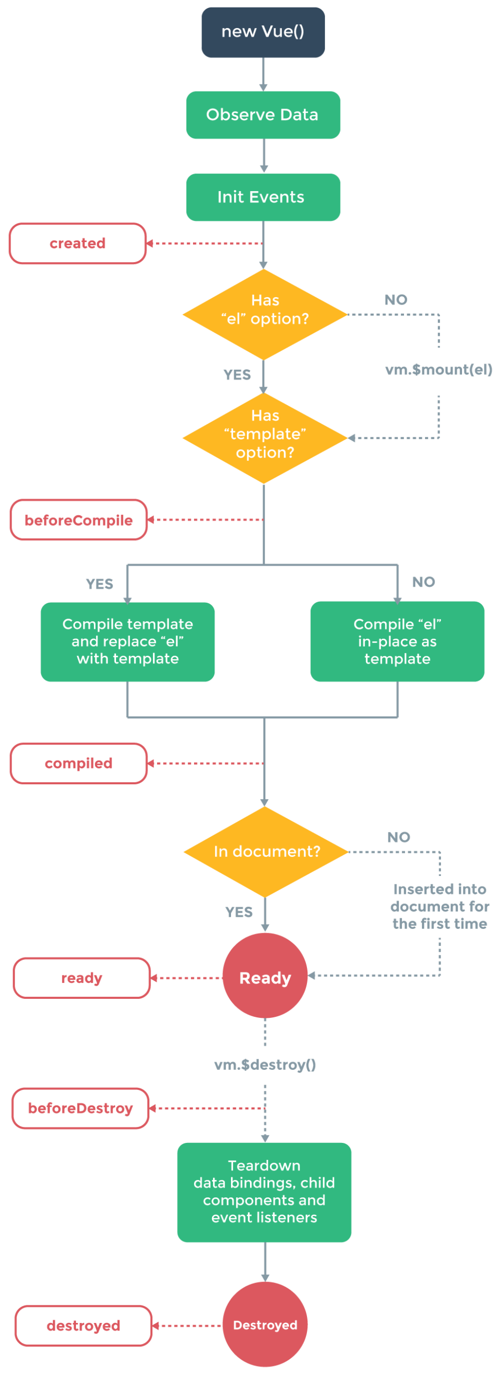
Isolated scope

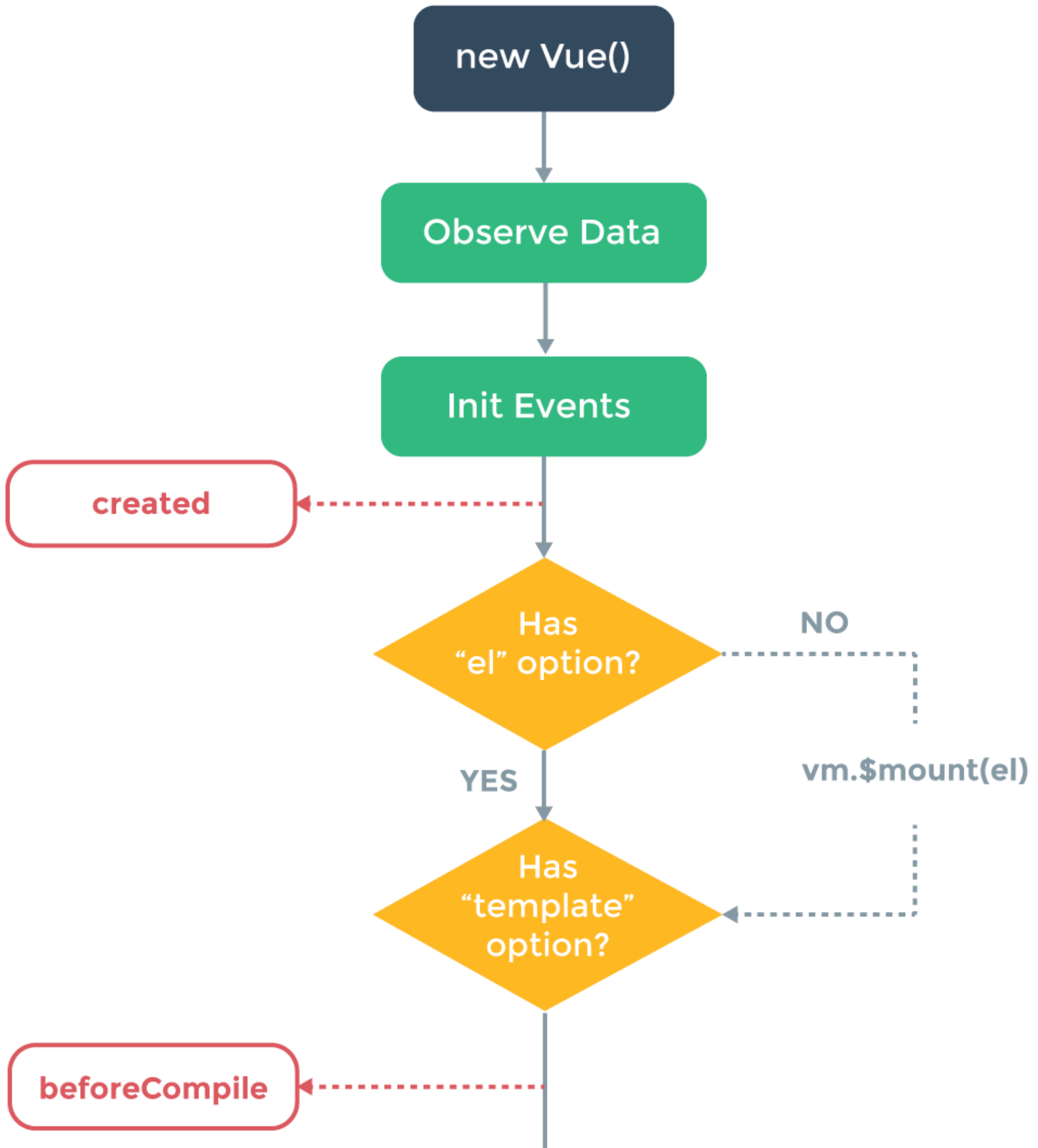
Encapsulate ***reusable*** code

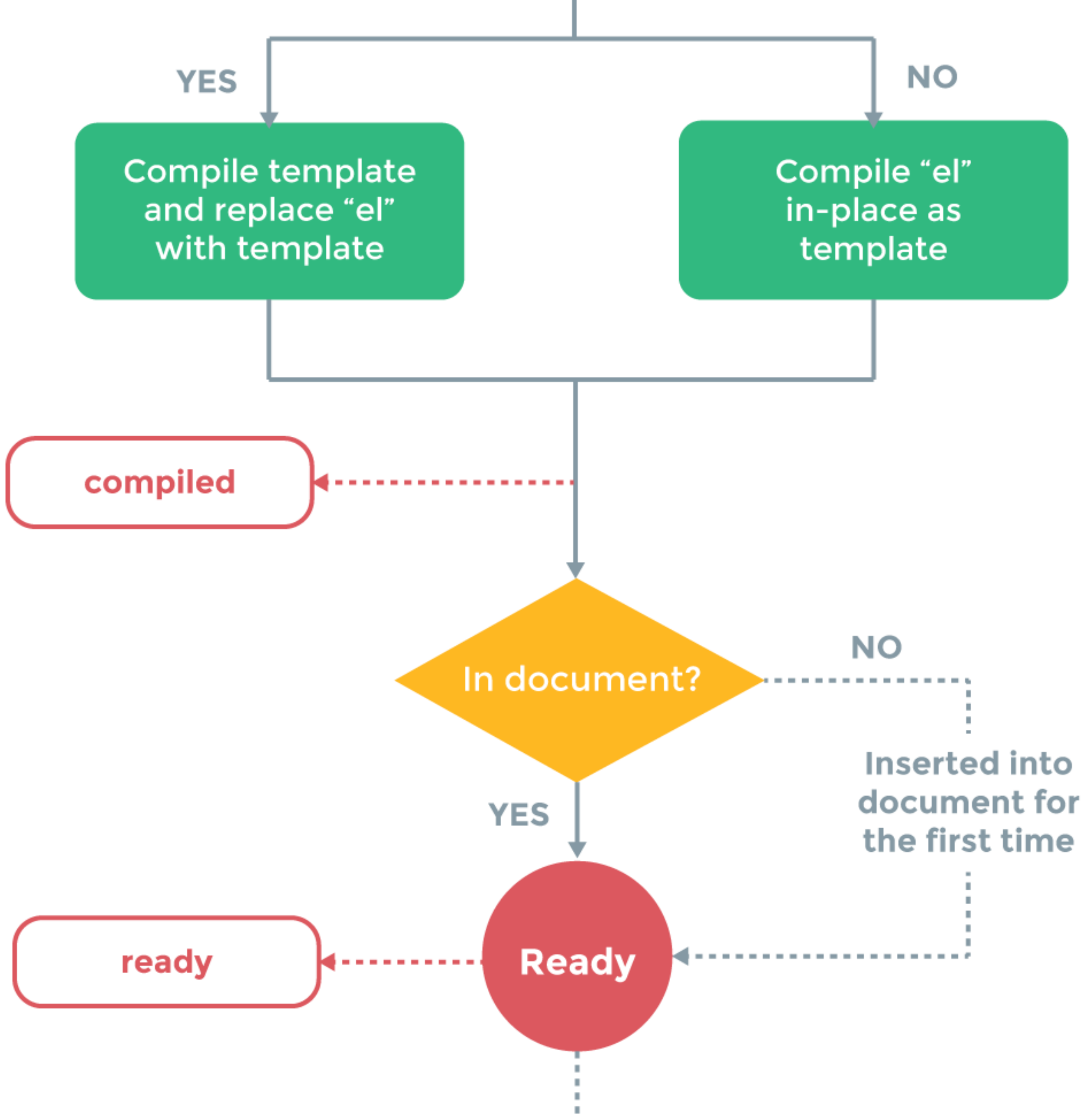
Components – lifecycle

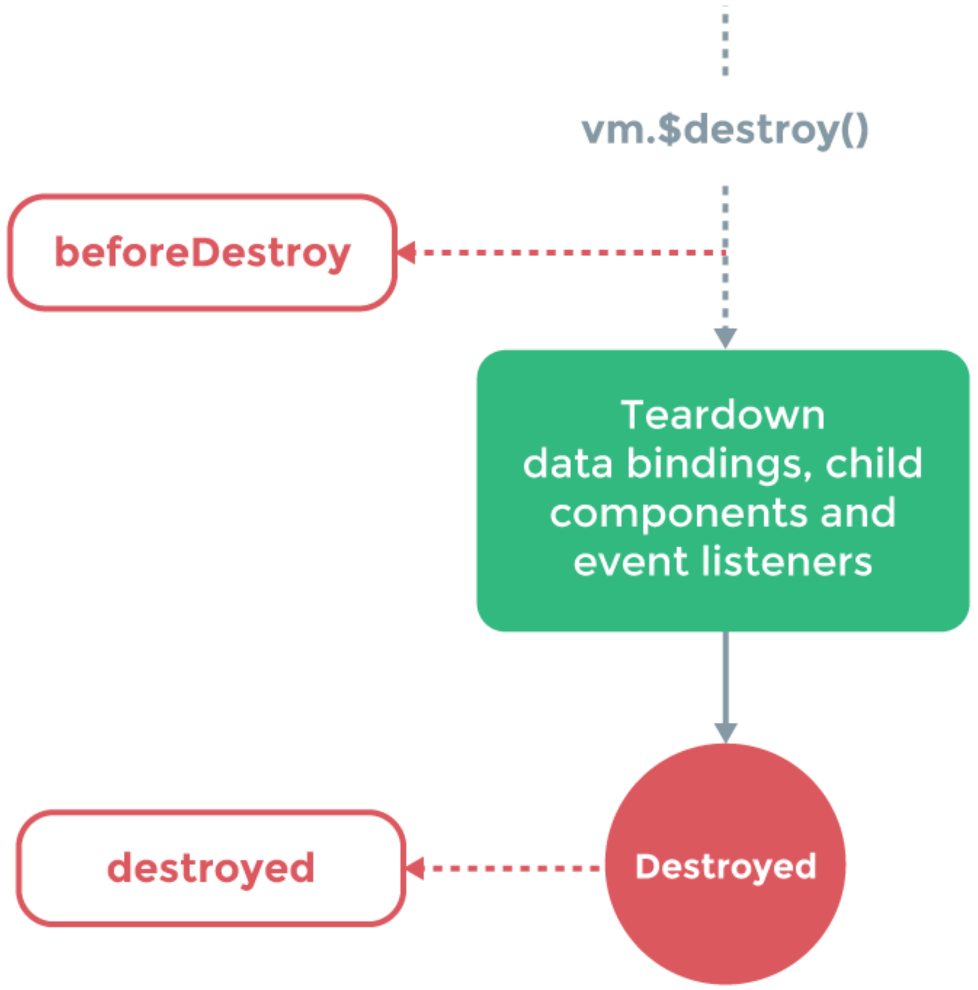
We can hook in at the lifecycle during:

- Data observation
- ***init***
- ***created***
- ***beforeCompile***
- ***compiled***
- ***ready***
- ***beforeDestroy***
- ***destroyed***




new Vue()
Observe Data
Init Events
created
Has “el” option?
NO
vm.$mount(el)
YES
Has “template” option?
beforeCompile


YES
NO
Compile template and replace “el” with template
Compile “el” in-place as template
compiled
In document?
NO
Inserted into document for the first time
YES
ready
Ready


vm.$destroy()
beforeDestroy
Teardown data bindings, child components and event listeners
Destroyed
destroyed

Component – Put together

```
Vue.component('demo-component', {
        template: '<div> {{ msg }} </div>',
        // Data initialization
        data() {
                return {
                        msg: 'Vue is cool'
                };
        },
        computed: {   },
        methods: {   },
        events: {   },
        ready() {
                // Execute logic on ready hook
        },
        // ...
});
```

Component – Into the DOM

Components are used with your HTML like so

```
<div>
    <demo-component></demo-component>
</div>
```

Components – Dynamic Components

Same entry point for loading components: ***component tag*** and ***is*** attribute.

```
<component :is="currentView"> </component>
```

Components – Dynamic Components

Reuse components to keep state and prevent re-rendering with the ***keep-alive*** option

```
<component :is="currentView"  keep-alive></component>
```

Directives

Directives – What are they?

Clear distinction from components

Directives are meant to ***encapsulate DOM manipulations only***

Components stand for a self-contained unit that has its own view and data logic

Directives – Custom directives

```
Vue.directive('demo-directive', {
        bind() {
                // Preparation work – adding event listeners or expensive stuff
        },
        update(newValue, oldValue) {
                // do something based on the initial and updated statement
        },
        unbind() {
                // Clean up - remove event listeners added in bind()
        }
});
```

Methods and Event Handling

Methods and Event handling

Easily locate the handler functions within the View.

No manual attachment of event listeners, JS can be ***pure logic*** and importantly ***DOM free***

All event listeners are automatically removed, upon destroying

Easily testable

Workflow

A brief introduction

Workflow

Vue.js + development = WIN

vue-cli

Boiler plate templates for aiding your development workflow:
https://github.com/vuejs-templates

Works great with ***Webpack*** or ***Browserify***

Enables ***Hot reloading*** of your components HTML/CSS/JS internally

Workflow – Component Hot Reloading

Changes detected to your Template, JS, or Styles prorogate up to components

Components are reloaded keeping any state

.vue files are essential for this, to act as a dependency tree

Resolving a bug? How annoying is it to have to fully refresh your pages?

Let's build something

Modals are generally difficult to produce, without some sort of JS implementation

We'll build a ***uni-flow*** modal with ***distributable content***

First let's improve our development experience…

Install developer tools

Developer tools: ***https://goo.gl/T6Fvtu***

Enable local files: ***chrome://extensions***

Creating a modal component

What do we need:

- Template
- Distributable content (Slot API)
- ’show’ state
- Open method
- Close method

Creating a modal component

Register a global component (modal.js)

```
Vue.component('modal', {

});
```

Creating a modal component

Register a template to modal component (modal.js)

```
Vue.component('modal', {
    template: ‘#modal-template’
});
```

Create a modal component

Adding state to component (modal.js)

```
Vue.component('modal', {
        // ...
        data: function() {
                return {
                        show: false
                };
        }
});
```

Create a modal component

Adding open method to component (modal.js)

```
Vue.component('modal', {
    // ...
    methods: {
        open: function() {
            this.show = true;
        }
    }
});
```

Create a modal component

Adding open method to component (modal.js)

```
// ...
methods: {
        // ...
        close: function() {
                this.show = false;
        }
}
```

Create a modal component

Adding distributable content (index.html – line 21)

```
<div class="modal-header">
        <slot name="header">default  header</slot>
</div>
<div class="modal-body">
        <slot name="body">default  body</slot>
</div>
<div class="modal-footer">
        <slot name="footer">
                <button  class="modal-button"  @click="close">Close</button>
        </slot>
</div>
```

Created a modal component!

We have made the modal component

Lets make sure that we are all up to date with consistent code

```
git reset HEAD --hard

git checkout component
```

Creating modal instances

What do we need:

- Modal instance
- Distributed content
- Content hiding until initialization
- Child to parent references
- Parent to child communication

Creating modal instances

Adding distributable content (index.html – line 40)

```
<div id="app">
    <modal>
        <h3 slot="header">Modal 1</h3>
        <div slot="body">Click `Close`!</div>
    </modal>
    <modal>
        <h3 slot="header">Modal 2</h3>
        <div slot="body">Please click `Close`!</div>
    </modal>
</div>
```

Creating modal instances

Hiding markup until initialization (index.html – line 40)

```
<div id="app">
    <modal v-cloak>
        // ...
    </modal>
    <modal v-cloak>
        // ...
    </modal>
</div>
```

Creating modal instances

Adding parent to child references (index.html – line 40)

```
<div id="app">
    <modal v-ref:modal1 v-cloak>
        // ...
    </modal>
    <modal v-ref:modal2 v-cloak>
        // ...
    </modal>
</div>
```

Opening of the modals

Adding buttons to open modals (index.html – line 40)

```
<div id="app">
    <button @click="$refs.modal1.open">Show Modal 1</button>
    <button @click="$refs.modal2.open">Show Modal 2</button>
    <modal v-ref:modal1 v-cloak>
        // . . .
    </modal>
    <modal v-ref:modal2 v-cloak>
        // . . .
    </modal>
</div>
```

Creating modal instances

We have two modal instances

Lets make sure that we are all up to date with consistent code

```
git reset HEAD --hard

git checkout instances
```

Modal transitions

What do we need:

- Some css transitions
- Transiton control

Modal transitions

Adding basic transition css (modal-animation.css)

```
.modal-enter, .modal-leave {
        opacity: 0;
}

.modal-enter .modal-container,
.modal-leave .modal-container {
        -webkit-transform: scale(1.1);
        transform: scale(1.1);
}
```

Modal transitions

Adding basic transition control (index.html – line 18)

```
<div class="modal-mask" v-show="show" transition="modal">
    // ...
</div>
```

Modal transitions

We now have animated modals

Lets make sure that we are all up to date with consistent code

```
git reset HEAD --hard

git checkout transition
```

Advanced parent flow

What do we need:

- Modal 1 to open modal 2
- Add no complexity to the modal component

Advanced parent control

Adding parent functionality (modal.js – line 20)

```
new Vue({
    el: '#app',
    methods: {
        parentHandler: function () {
            this.$refs.modal1.close();
            this.$refs.modal2.open();
        }
    }
});
```

Advanced parent control

Distributed content with parent flow (index.html – line 40)

```
<div id="app">
    // . . .
    <modal v-ref:modal1 v-cloak>
        <h3 slot="header">Modal 1</h3>
        <div slot="body">Click `Continue`!</div>
        <div slot="footer">
            <button class="modal-button" @click="parentHandler">Continue</button>
        </div>
    </modal
    // . . .
</div>
```

Advanced parent flow

We now have parent instance controlling flow of children

Lets make sure that we are all up to date with consistent code

```
git reset HEAD --hard

git checkout final
```

We are done!

Simple modal component!
Flexible content distribution!
Simple animation integration!
Uni-flow patterns!
No two way data binding!
Simple parent to child communication!