Interacting within a Component

John Papa
DEVELOPER ADVOCATE

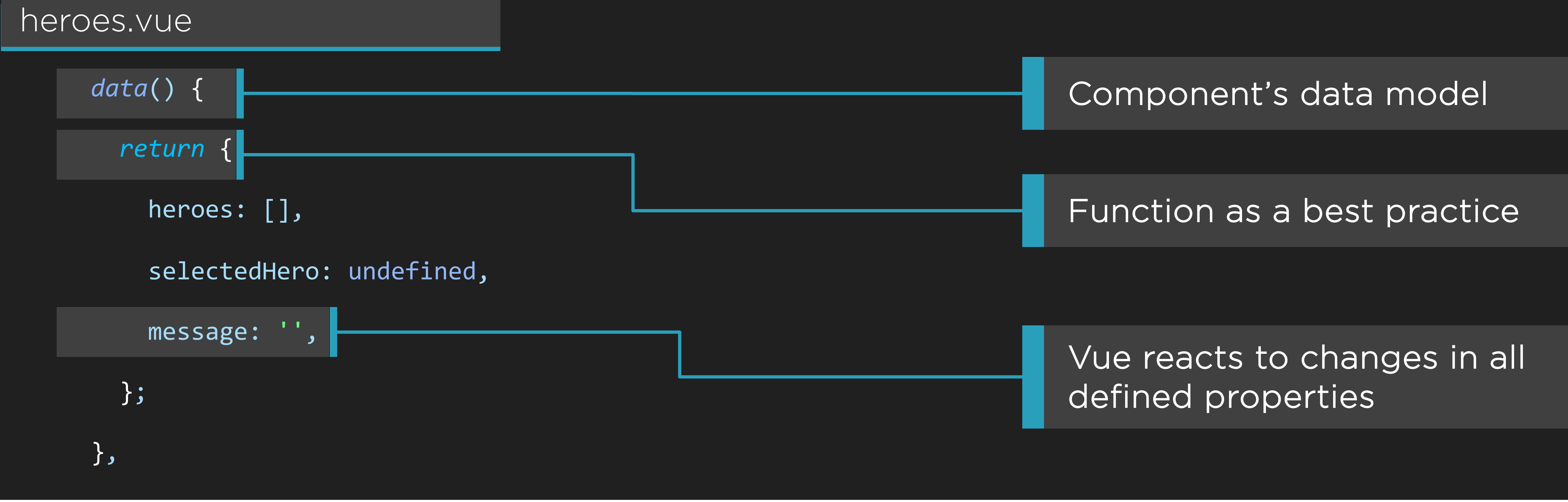
@john_papa www.johnpapa.net

How do we …

- define data models?
- define functions to for a component?
- tap into specific events in the component life cycle?
- define custom properties that react to model changes?
- run custom logic when model's change?
- format data for the users?

Demo

Data models



heroes.vue

```
data() {
  return {
    heroes: [],
    selectedHero: undefined,
    message: '',
  };
},
```

Component's data model

Function as a best practice

Vue reacts to changes in all defined properties

The data Function

The Vue instance adds all properties in data to Vue's reactivity system

When values change, the component reacts

Component updates to match the new values

Demo

Computed Properties

Computed Properties

Fire when any dependency value changes

Cached based on its reactive dependencies

Only re-evaluates when any of its reactive dependencies have changed

heroes.vue

```
computed: {
  fullName() {
    return `${this.hero.firstName} ${this.hero.lastName}`;
  },
},
```

When either dependency changes, the computed evaluates

Computed get/set

Use get() to compute the value

Create getter and setter functions

Use set() to modify computed dependencies

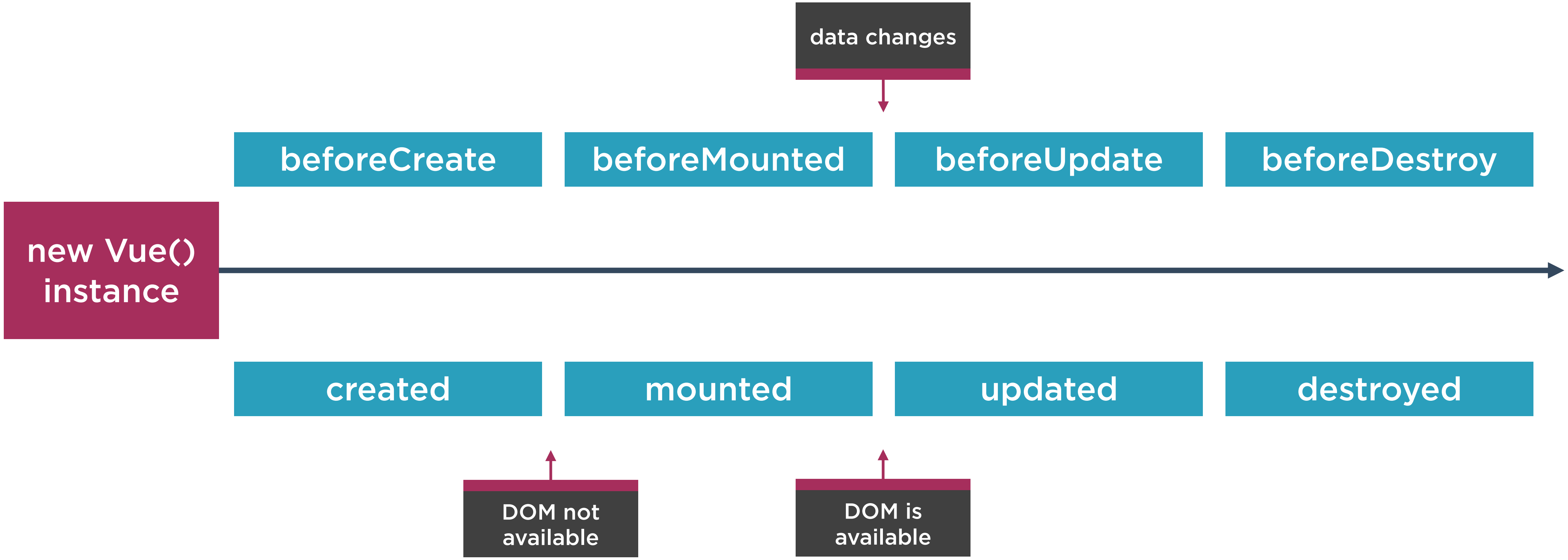
heroes.vue

```
computed: {
  fullName: {
    get() {
      let value = this.first;
      value += this.last ? ` ${this.last}` : '';
      return value;
    },
    set(value) {
      let names = value.split(' ');
      this.first = names[0];
      this.last = names.length === 1
        ? '' : names[names.length - 1];
    },
  },
}
```

Demo

Lifecycle Hooks

Component Lifecycle Hooks



created

Invoked when the component is created

Frequently used to fetch data for your component

Templates and virtual DOM are not yet mounted nor rendered

heroes.vue

```
created() {
  this.getHeroes().then(heroes => {
    this.heroes = heroes;
  });
},
```

Demo

Watched Properties

heroes.vue

```
watch: {
  hero(newValue, oldValue) {
    console.log(`old=${oldValue}, new=${newValue}`);
    // execute logic
  },
  'selectedHero.capeCounter': {
    immediate: true,
    handler(newValue, oldValue) {
      // execute logic
    },
  },
},
```

Watchers

React when this model changes

React to data changes

Named same as reactive value

Accepts new and old values

Ideal for asynchronous operations

Use quotes when needed

Demo

Filters

heroes.vue

```
filters: {
  capitalize: function(value) {
    if (!value) return '';
    value = value.toString();
    return value.charAt(0).toUpperCase()
      + value.slice(1);
  },
},
```

Filters

Local filters are defined in a component

Apply in the template

{{ firstName | capitalize }}

Filters can be chained

{{ firstName | capitalize | reverse }}

Optionally pass arguments

{{ someDate | myDateFilter('MM-DD-YY') }} app.vue

```
import Vue from 'vue';

Vue.filter('capitalize', function(value) {
  if (!value) return '';
  value = value.toString();
  return value.charAt(0).toUpperCase() + value.slice(1);
});
```

Global Filters

Define once, use everywhere

When defining a global filter, it must come before the Vue instance

Your Tools Inside a Component

data()	Define your component's models
methods	Execute custom logic
lifecycle hooks	Tap into when a specific component event occurs
computed	Property that fires when any dependency value changes
watch	Execute custom logic when a specific data model changes
filters	Transform the output (not the model) that the user sees

Summary

Create data models

Perform operations when data changes

Tap into lifecycle events

Format data