從 Vue.js 初探 Web Component 的世界

Kuro Hsu X 六角學院

Kuro Hsu

前端工程師
五倍紅寶石兼任講師
資策會兼任講師

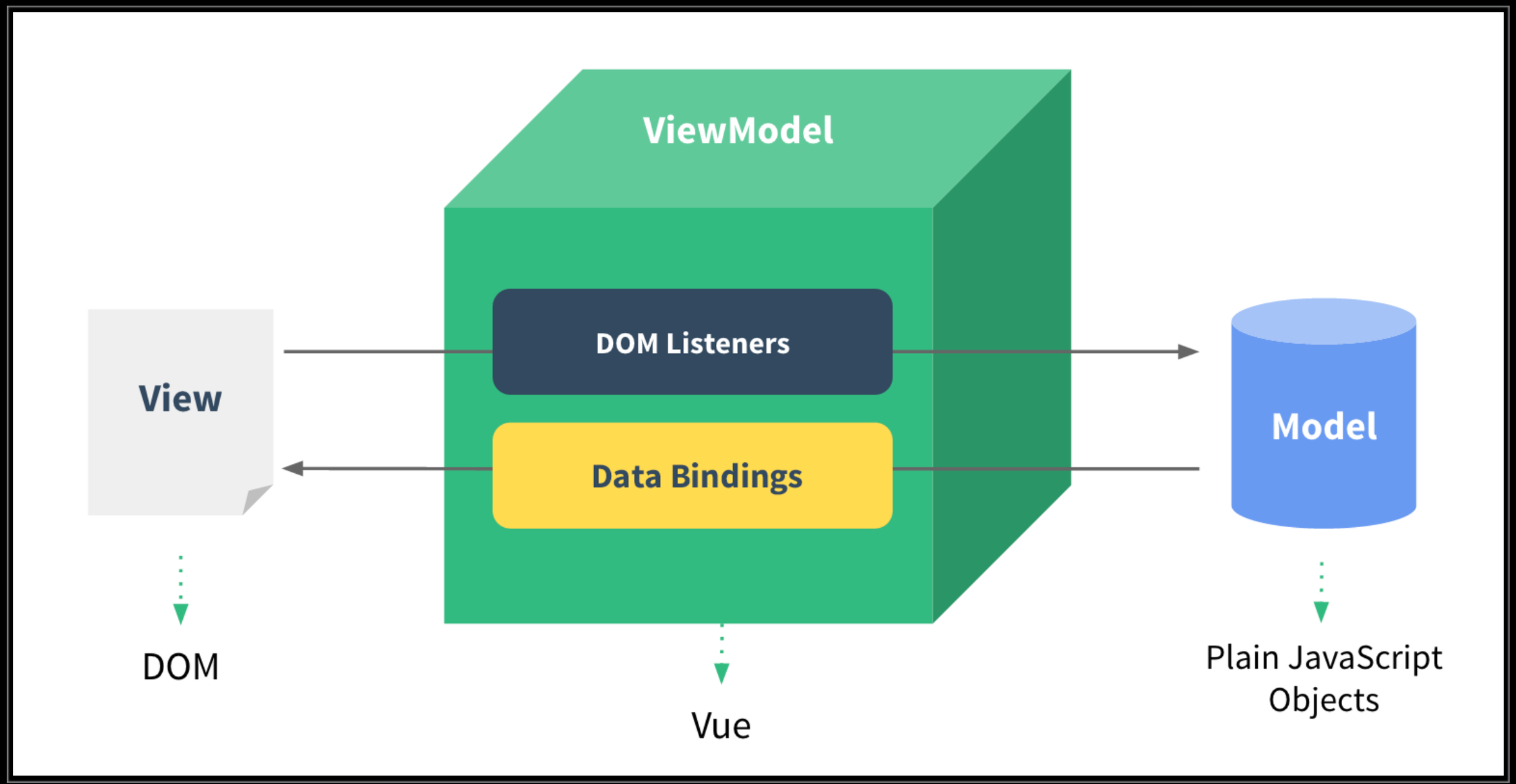
VueJS 台灣社團管理群

http://kuro.tw
kurotanshi@gmail.com
@kurotanshi

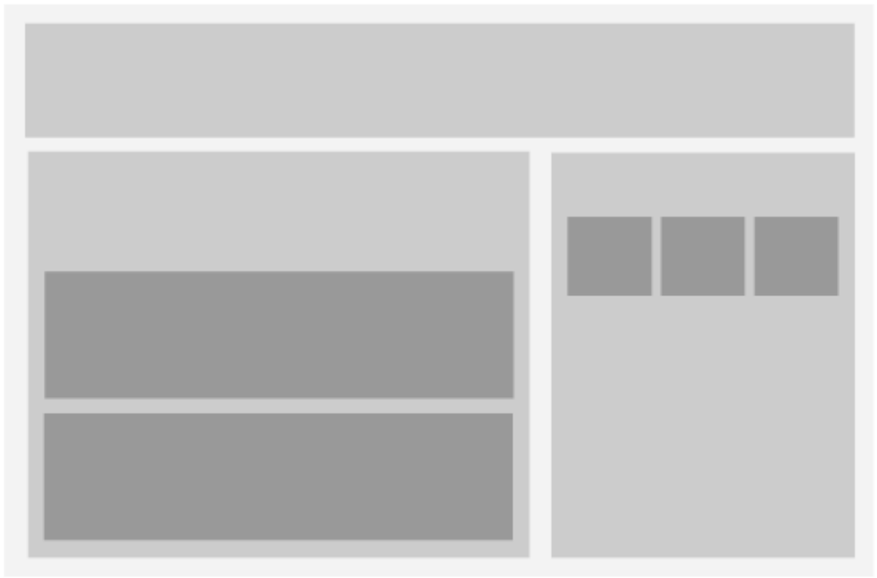
Overview

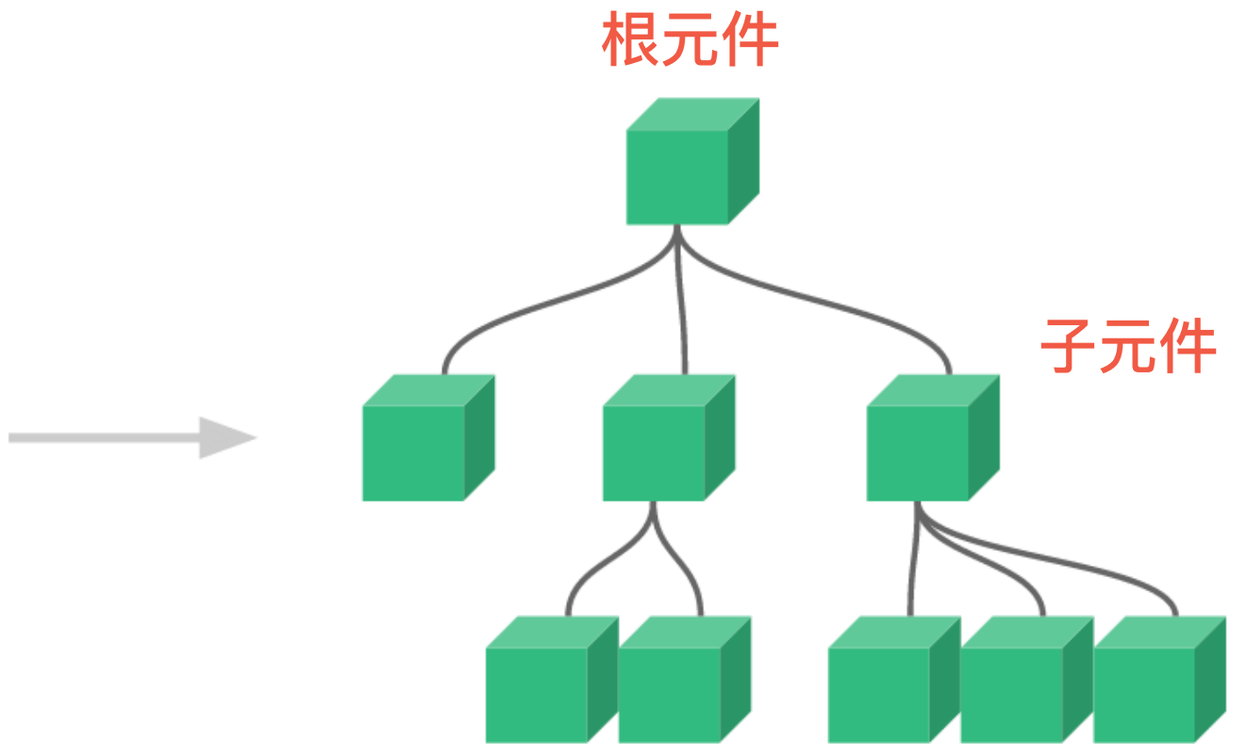
快速導覽

MVVM (Model-View-ViewModel)



元件系統





大多數的屬性與實體都相同，
唯一要注意的是 data 屬性。

```
var vm = new Vue({
  el: '#app',           //         el: 用來掛載 Vue 實體的元素
  data:  {  },          //       data: 要綁定的資料
  props: {  },          //      props: 用來接收外部資料的屬性
  methods: {  },        //    methods: 用來定義在 Vue 實體內使用的函數（方法）
  watch: {  },          //      watch: 用來觀察 Vue 實體內資料的變動
  computed: {  },       //   computed: 自動為內部資料計算過的屬性
  template: "...",      //   template: 提供 Vue 實體編譯後的樣板
  components: {   }     // components: 用來定義子元件
});
```

Vue Component 註冊

(Scoped)

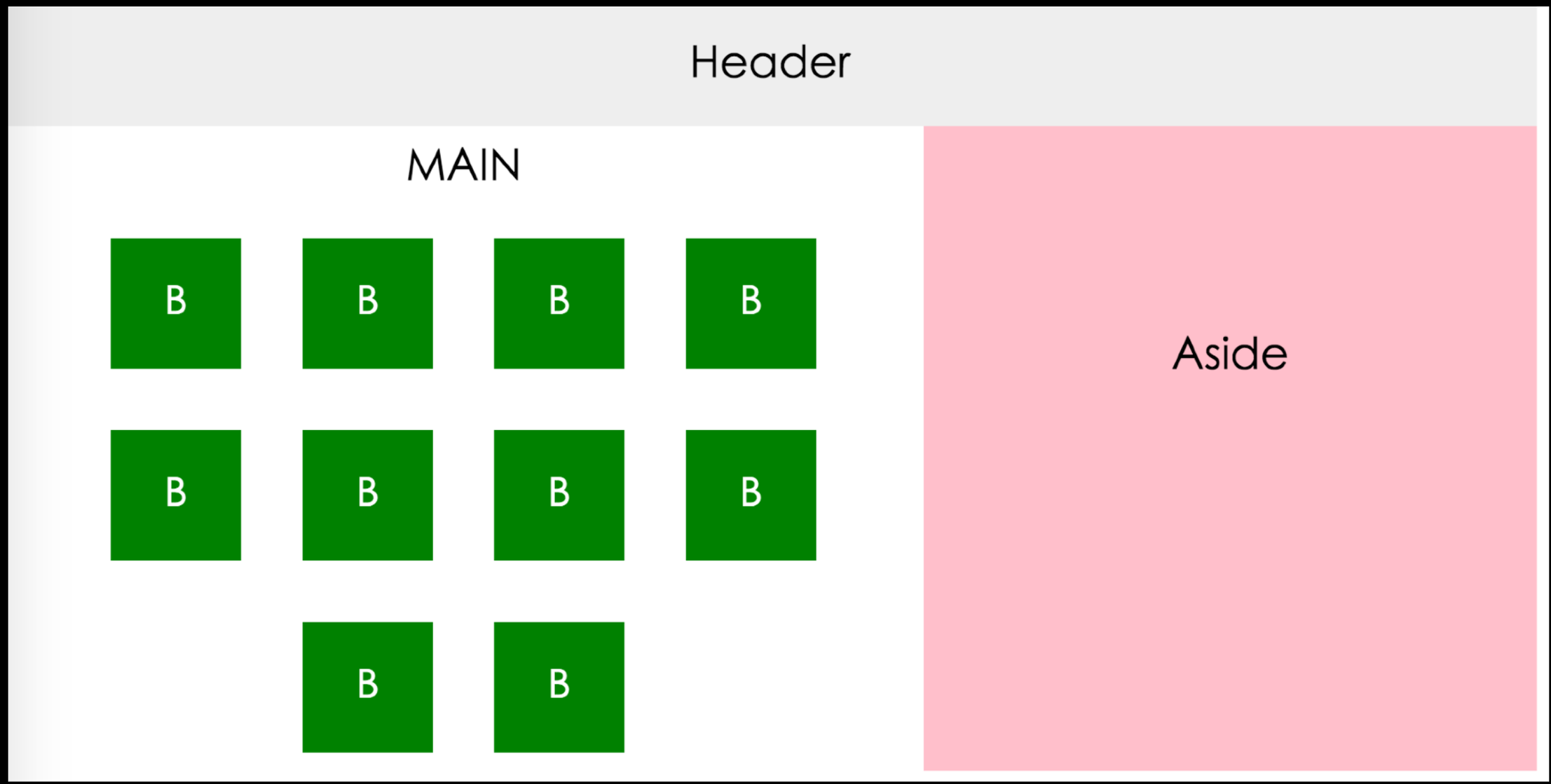
```
<div id="app">
  <my-component></my-component>
</div>
```

```
// create a root instance
new Vue({
  el: '#app',
  components: {
    // register
    'my-component': {
      template: '<div class="component">A custom component of Vue!</div>'
    }
  }
});
```

(global)

```html
<div id="app">
  <my-component></my-component>
</div>
```

```js
// register
Vue.component('my-component', {
  template: '<div class="component">A custom component of Vue!</div>'
});

// create a root instance
new Vue({
  el: '#app'
});
```


Header
MAIN
B
B
B
B
B
B
B
B
B
Aside

```
new Vue({
  el: '#app',
  components: {

    CustomHeader: Vue.extend({
      template: '<div class="header">Header</div>',
    }),

    CustomMain: Vue.extend({
      template:
      '<div class="main">' +
        '<div style="margin: 10px;">MAIN</div>' +
        '<custom-block></custom-block>' +
        '<custom-block></custom-block>' +
        '<custom-block></custom-block>' +
        '<custom-block></custom-block>' +
        '<custom-block></custom-block>' +
        '<custom-block></custom-block>' +
        '<custom-block></custom-block>' +
        '<custom-block></custom-block>' +
      '</div>',
      components: {
        CustomBlock: Vue.extend({
          template: '<div class="block">B</div>',
        })
      }
    }),

    CustomAside: Vue.extend({
      template: '<div class="aside">Aside</div>',
    }),
  }
});
```

Header

Main

Block

Aside

```
var CustomBlock =  Vue.extend({
  template: '<div class="block">B</div>',
});

var CustomMain =  Vue.extend({
  template:
  '<div class="main">' +
    '<div style="margin: 10px;">MAIN</div>' +
    '<custom-block></custom-block>' +
    '<custom-block></custom-block>' +
    '<custom-block></custom-block>' +
  '</div>',
  components: {
    CustomBlock
  }
});

var CustomHeader = Vue.extend({
  template: '<div class="header">Header</div>',
});

var CustomAside = Vue.extend({
  template: '<div class="aside">Aside</div>',
});

new Vue({
  el: '#app',
  components: {
    CustomMain,
    CustomHeader,
    CustomAside
  }
});
```

子元件的 data 必須是 function

```
// wrong
Vue.component('my-component', {
  template: '<div class="component">{{ msg }}</div>',
  data: {
    msg: 'A custom component of Vue!'
  }
});

// right
Vue.component('my-component', {
  template: '<div class="component">{{ msg }}</div>',
  data: function() {
    return {
      msg: 'A custom component of Vue!'
    }
  }
});
```

將網頁模板封裝成 Component

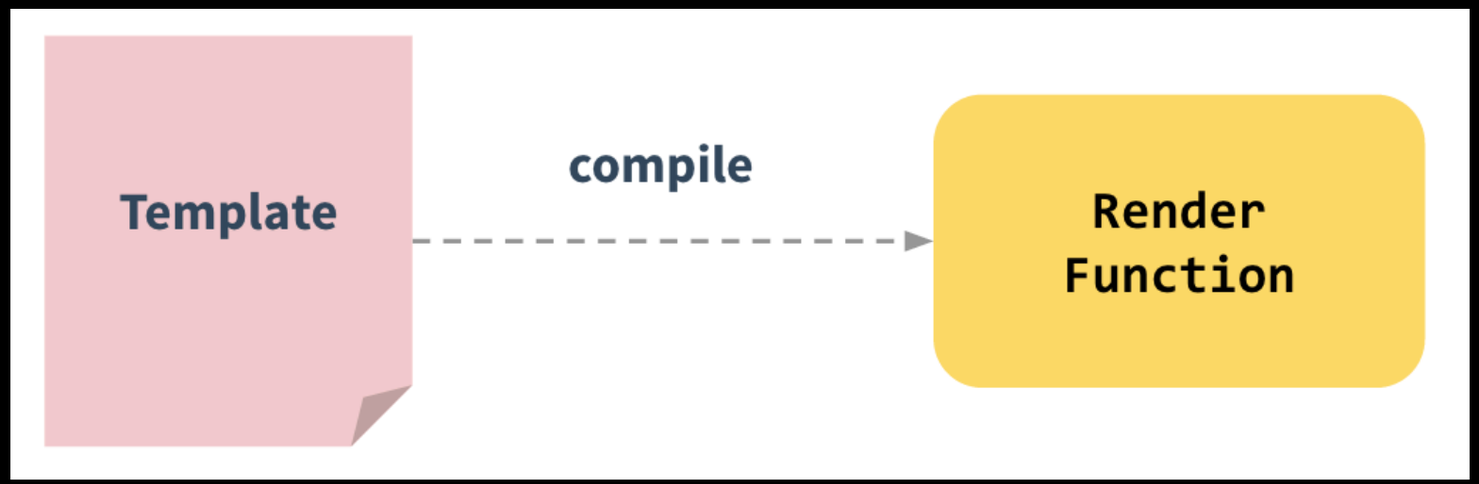
HTML 標籤

```
var CustomMain = Vue.extend({
  template: '<div class="main">' +
    '<div style="margin: 10px;">MAIN</div>' +
    '<custom-block></custom-block>' +
    '<custom-block></custom-block>' +
    '<custom-block></custom-block>' +
    '</div>',
  components: {
    CustomBlock
  }
});
```

X-Templates

```
<script type="text/x-template" id="my-component">
  <div class="component">A custom component of Vue!</div>
</script>

<script>
  // register
  Vue.component('my-component', {
    template: '#my-component'
  });
```

render function

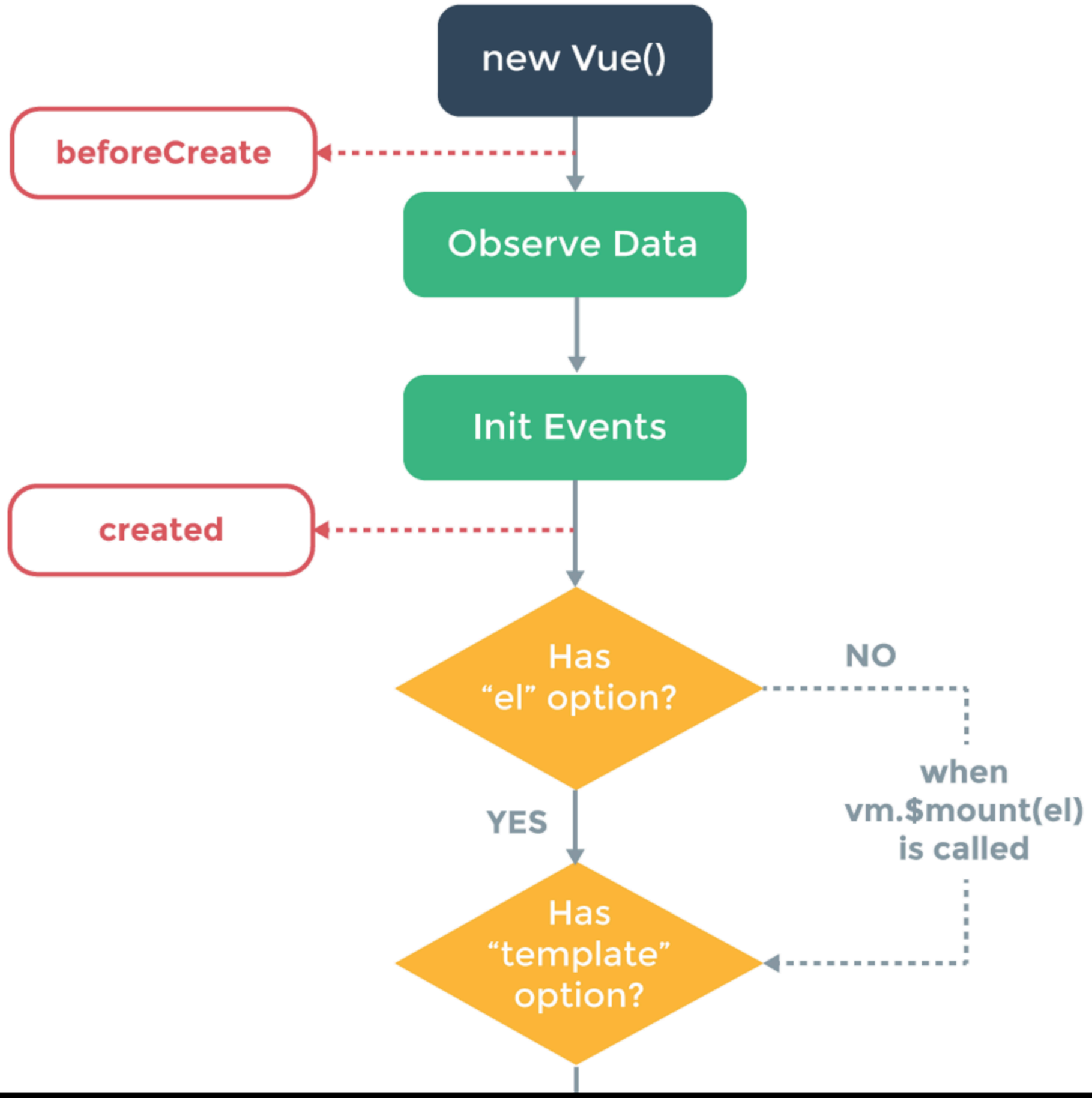


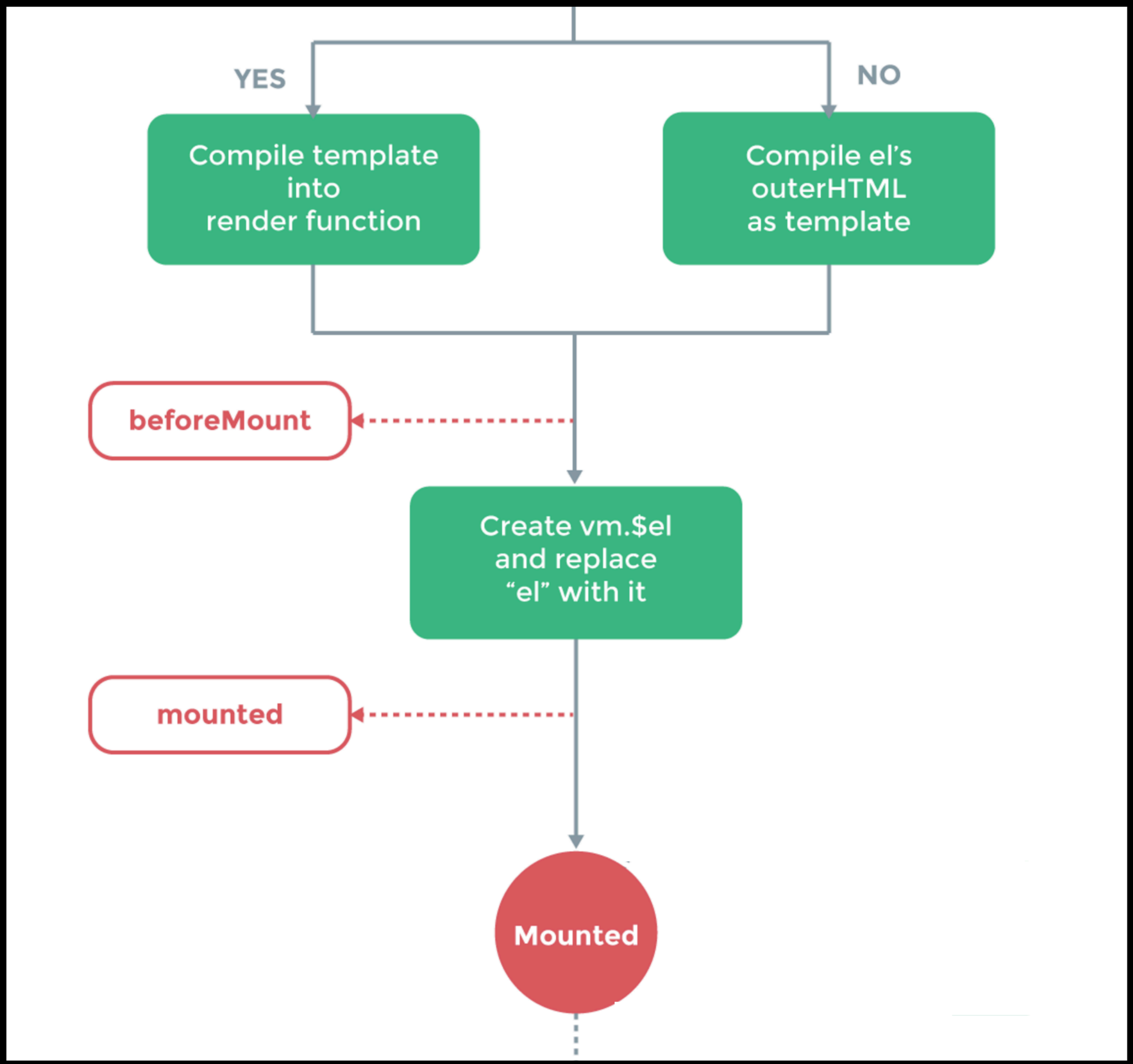

```
<div id="demo">
  <h1>{{msg}}</h1>
</div>
```

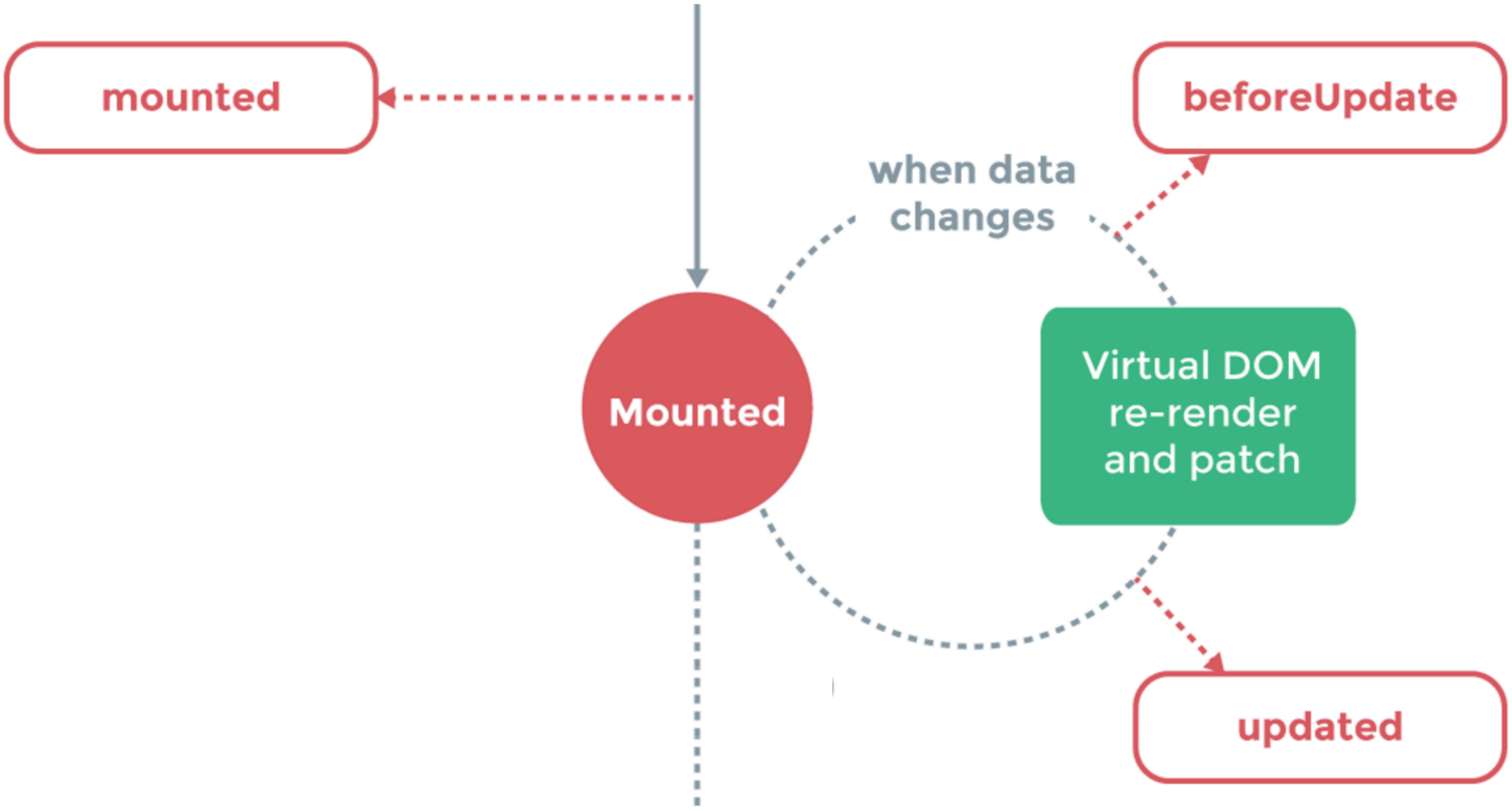
```
render (createElement) {
    return createElement(
        'div',
        { attrs: { id: 'demo' }},
        [createElement('h1', this.msg)]
    )
}
```

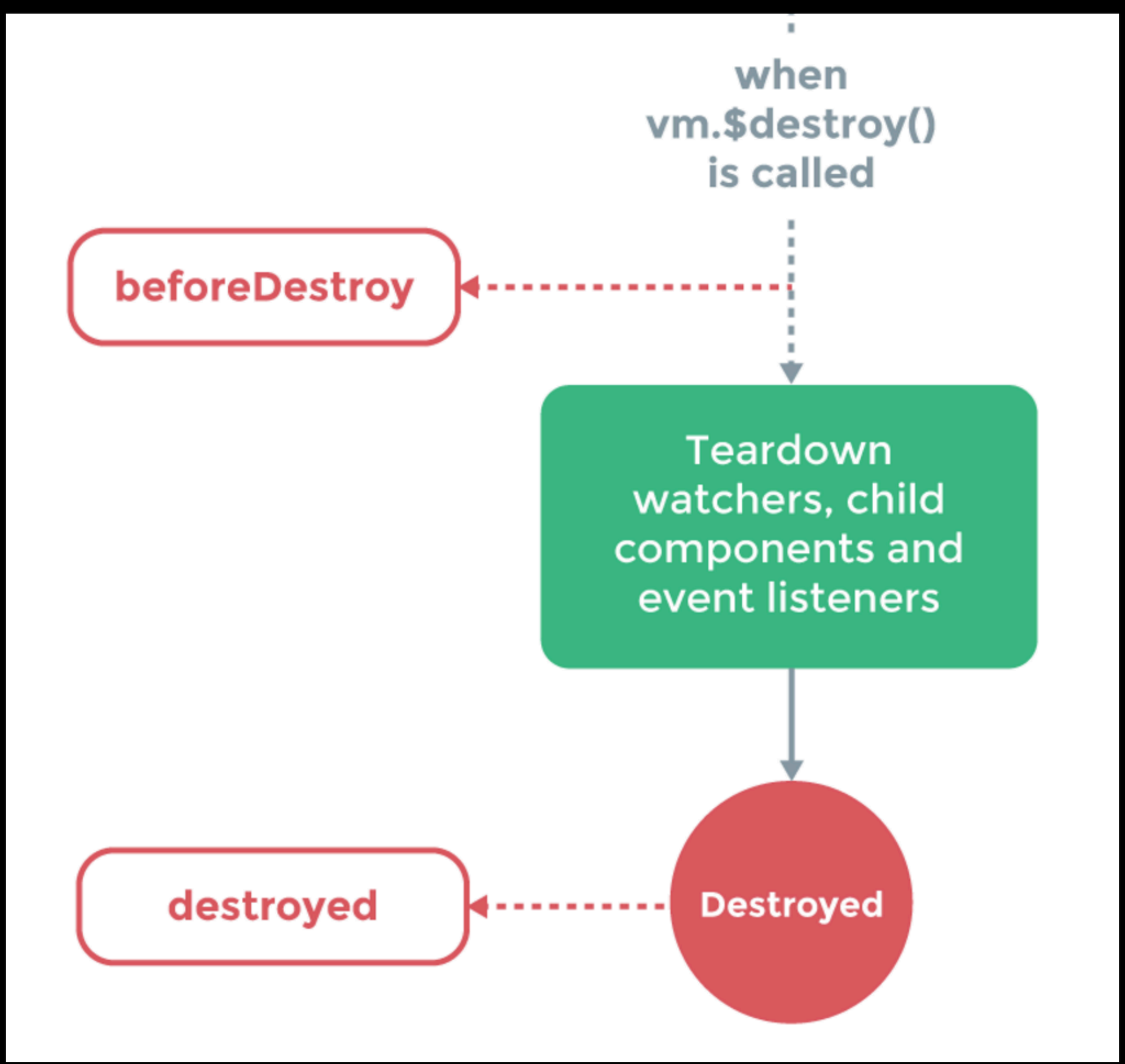
Lifecycle

Vue 元件實體生命週期


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


YES
NO
Compile template into render function
Compile el's outerHTML as template
beforeMount
Create vm.$el and replace "el" with it
mounted
Mounted

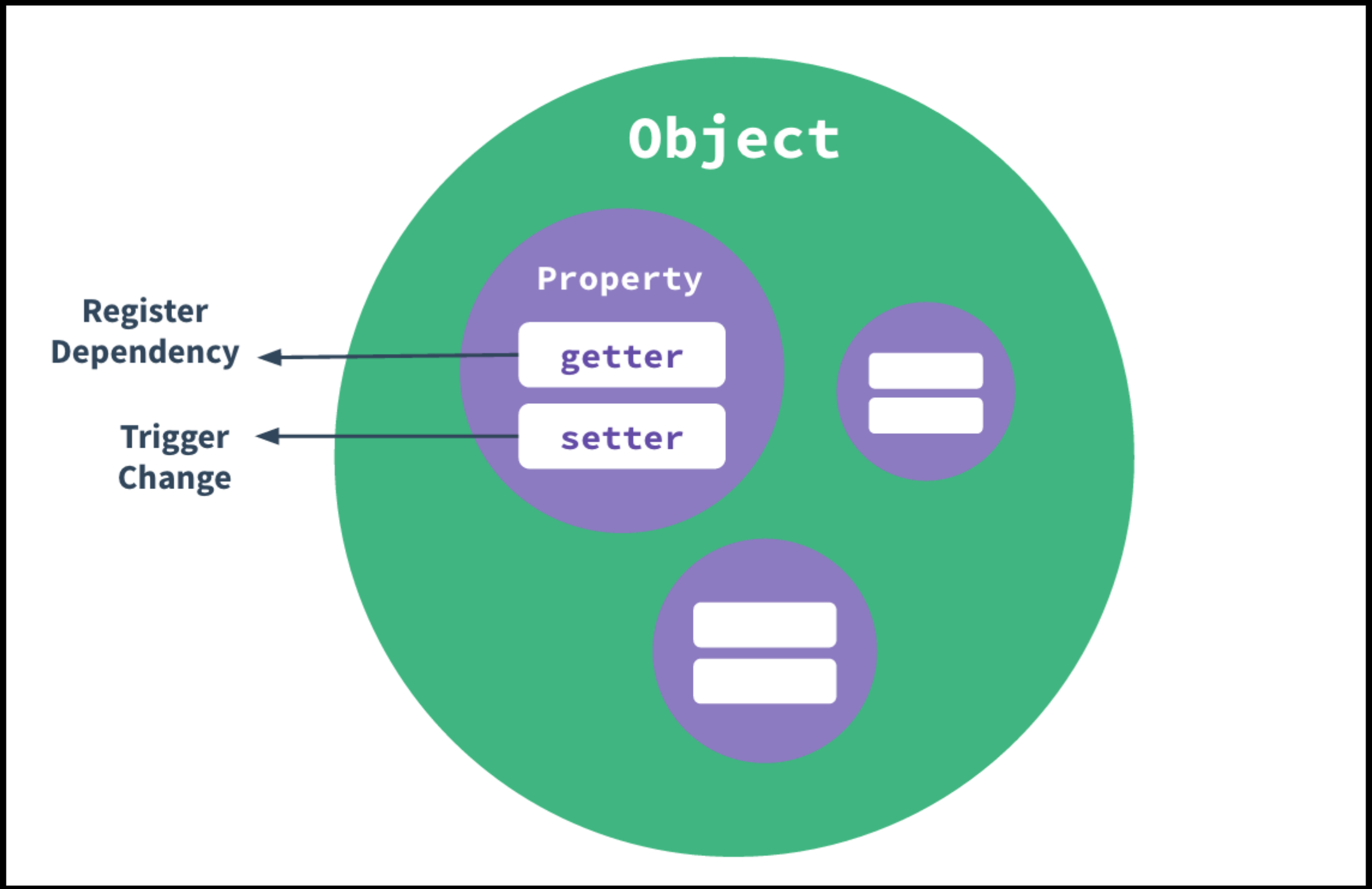

mounted
beforeUpdate
when data changes
Mounted
Virtual DOM re-render and patch
updated


when
vm.$destroy()
is called
beforeDestroy
Teardown
watchers, child
components and
event listeners
destroyed
Destroyed

Vue 2.0 hook	hook 說明
beforeCreate	元件實體剛被建立，屬性計算之前。
created	元件實體已建立，屬性已綁定，但 DOM 還沒生成。
beforeMount	模板（template）編譯或掛載至 HTML 之前
mounted	模板（template）編譯或掛載至 HTML 之後
beforeUpdate	元件被更新之前
updated	元件被更新之後
activated	keep-alive 用，元件被啟動時呼叫
deactivated	keep-alive 用，元件被移除時呼叫
beforeDestory	元件被銷毀前呼叫
destoryed	元件被銷毀後呼叫

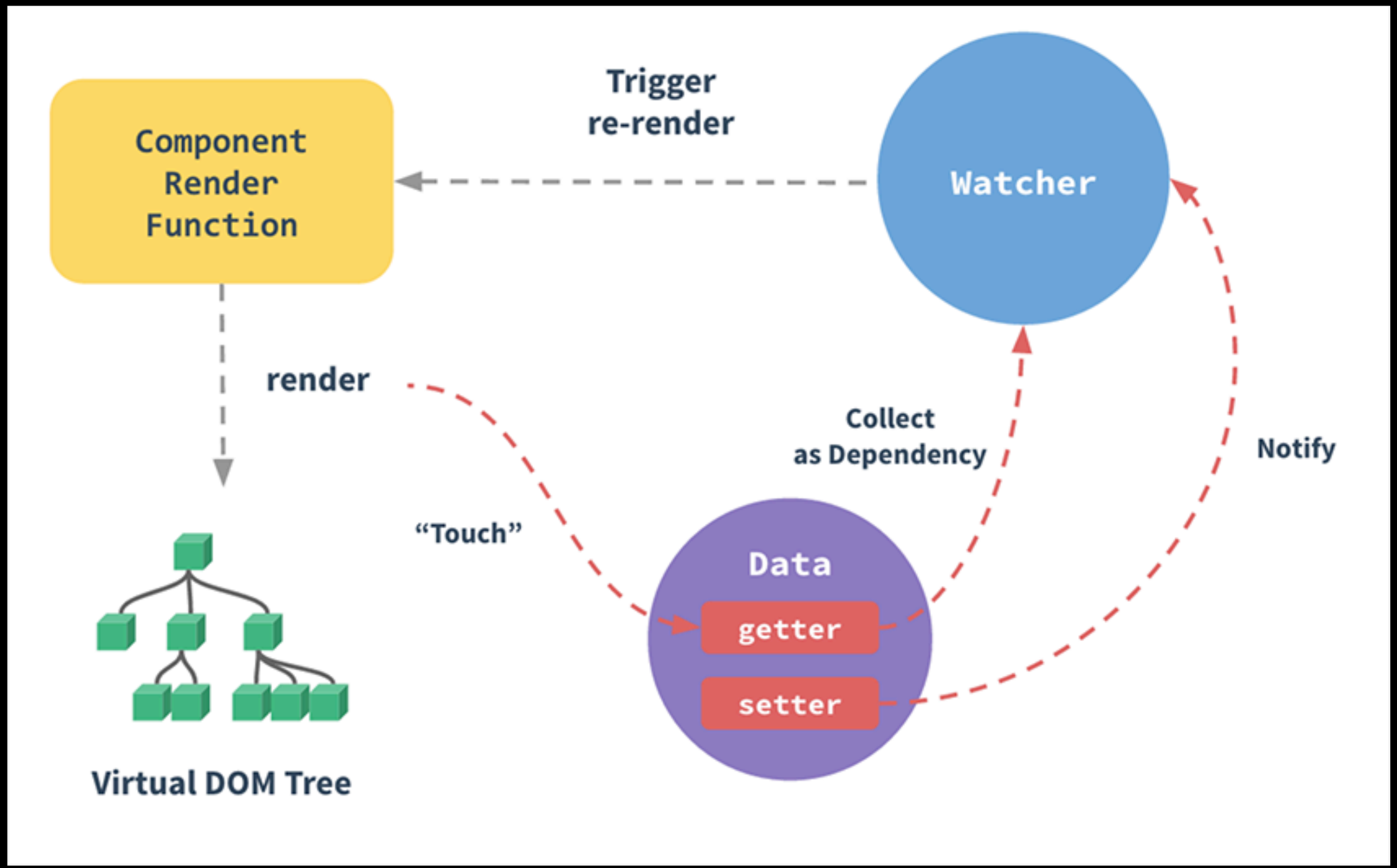
```
var _root = new Vue({
  el: '#app',
  data: {
    count: 0
  },
  beforeCreate: function(){
    // 元件實體剛被建立，屬性計算之前。
  },
  created: function(){
    // 元件實體已建立，屬性已綁定，但 DOM 還沒生成。
  },
  beforeMount: function(){
    // 模板 (template) 編譯或掛載至 HTML 之前
  },
  mounted: function(){
    // 模板 (template) 編譯或掛載至 HTML 之後
  },
  beforeUpdate: function(){
    // 元件被更新之前
  },
  updated: function(){
    // 元件被更新之後
  }
});
```

追蹤物件變化
Object.defineProperty



https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty

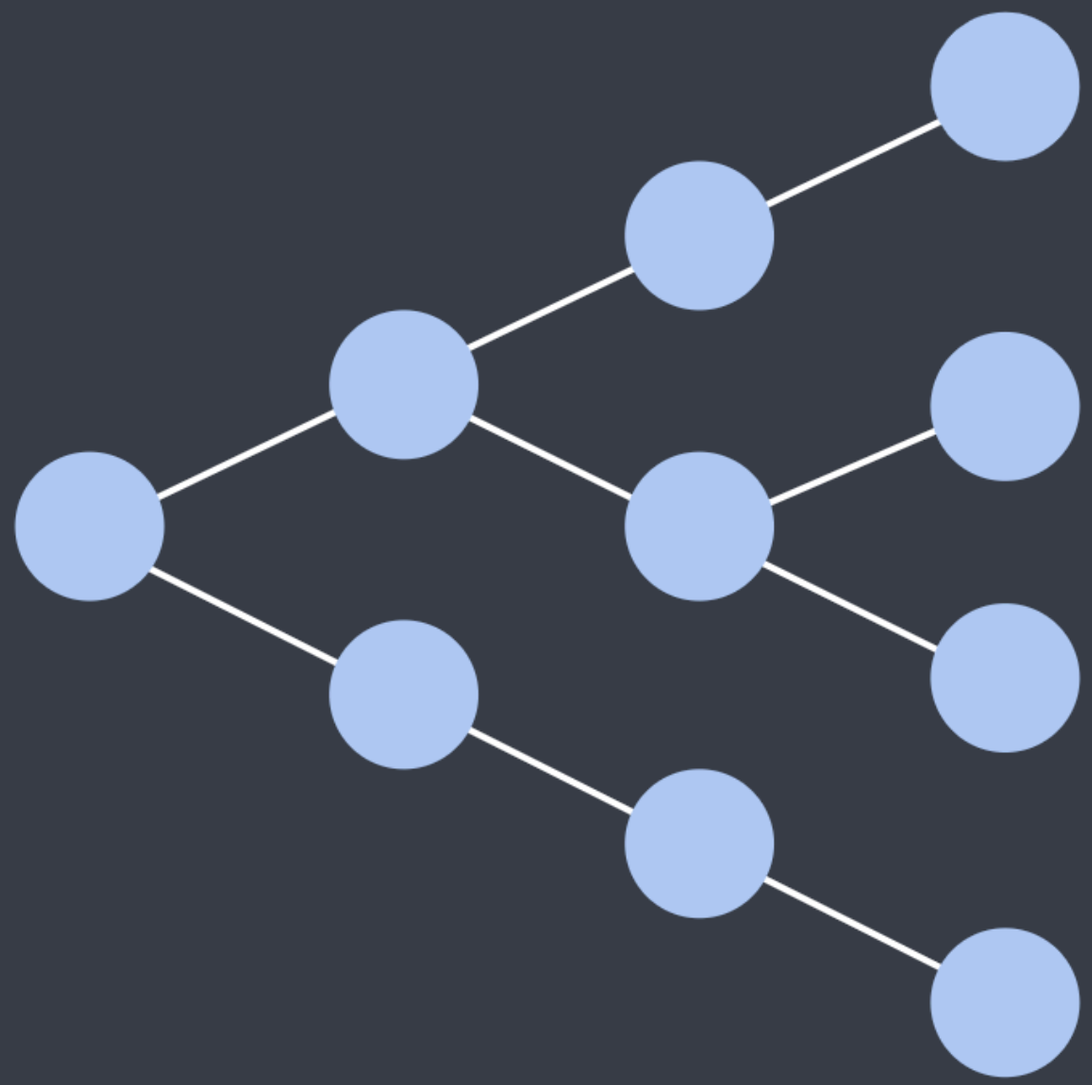
資料的變動偵測


Component
Render
Function
Trigger
re-render
Watcher
render
Collect
as Dependency
Notify
"Touch"
Data
getter
setter
Virtual DOM Tree

與其他主流框架的差異：Push & Pull

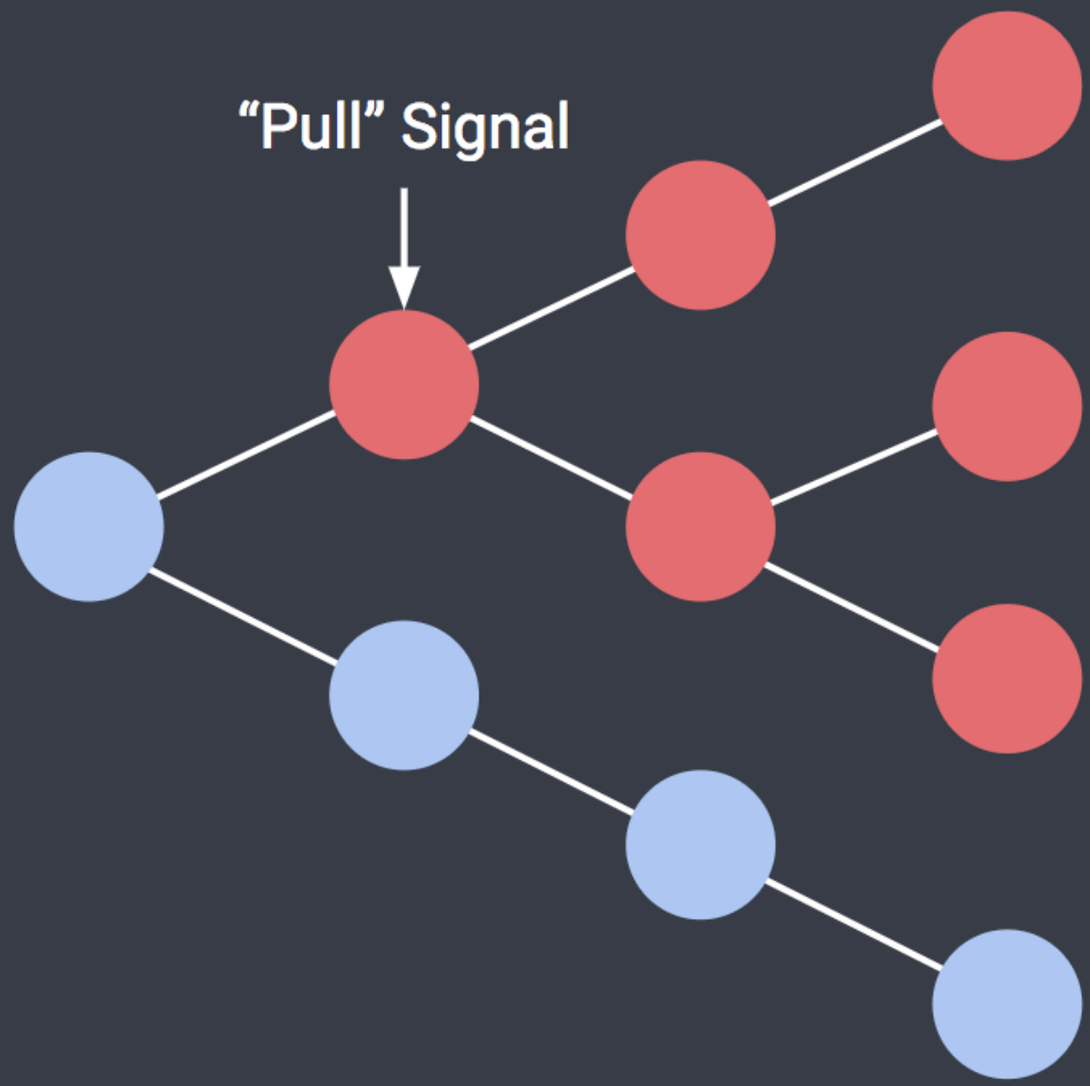
- 資料的變化偵測主要分為 Push 與 Pull
- Pull: 如 React 的 setState、Angular 的 dirty check 等
- Push: 如 Vue 的響應式更新、RxJS 的 observable 等
- 不管是哪一種都會有對應的成本，沒有絕對好或不好。

Component Tree
forms a Computation Tree



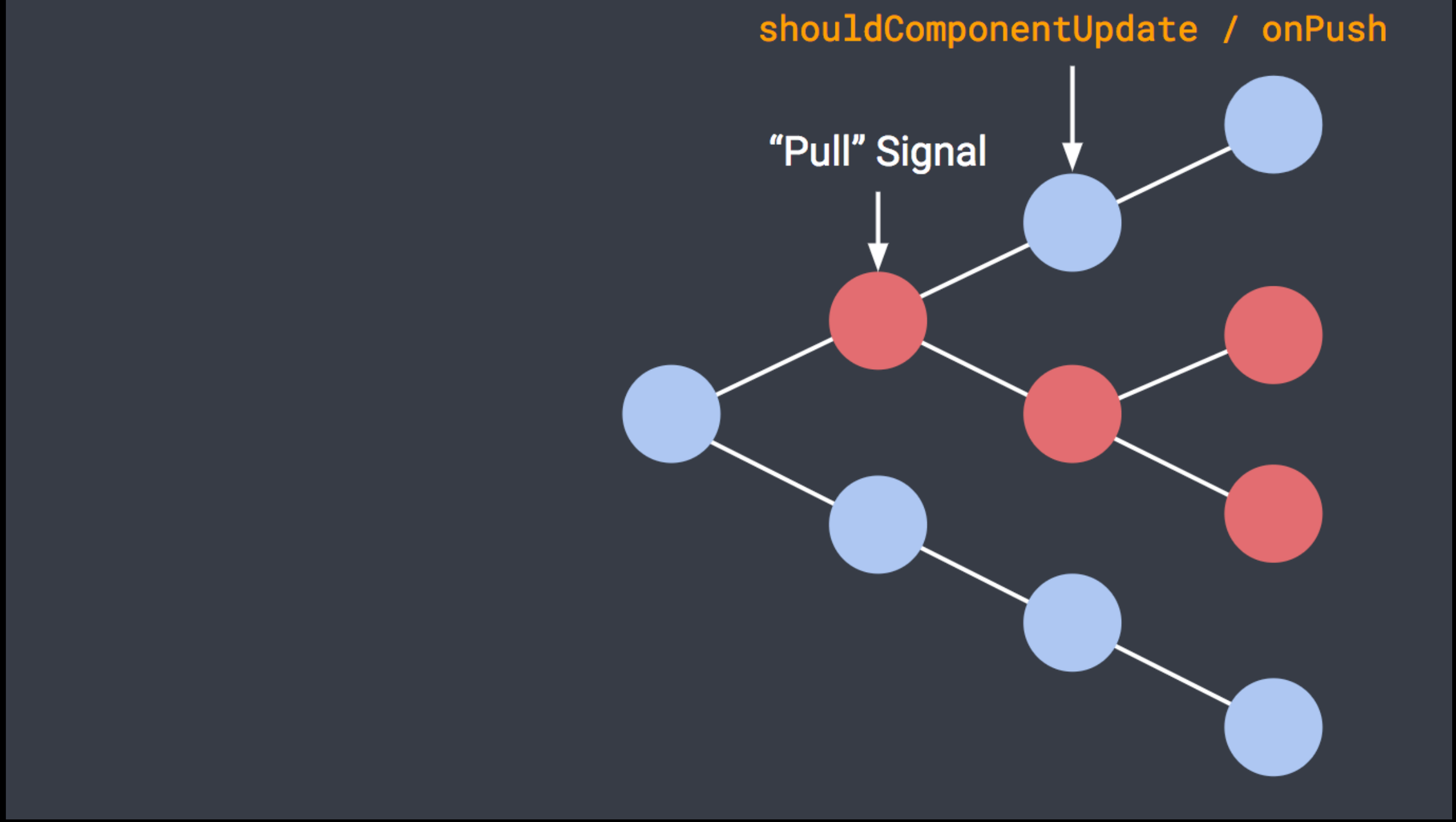
Source: Reactivity in Frontend JavaScript Frameworks
https://docs.google.com/presentation/d/1_BlJxudppfKmAtfbNIcqNwzrC5vLrR_h1e09apcpdNY/edit?usp=sharing

Pull-based Change Propagation



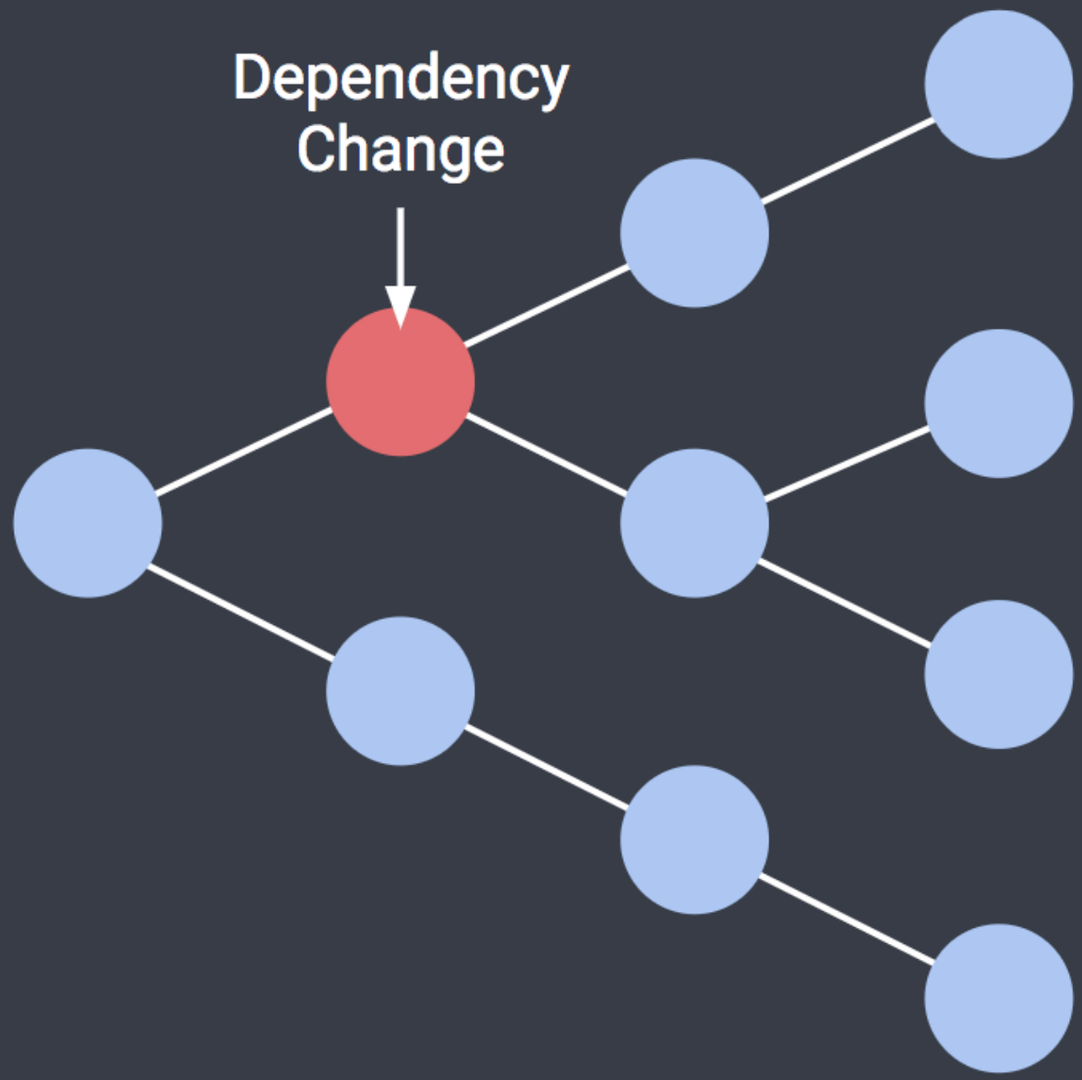
Source: Reactivity in Frontend JavaScript Frameworks
https://docs.google.com/presentation/d/1_BlJxudppfKmAtfbNIcqNwzrC5vLrR_h1e09apcpdNY/edit?usp=sharing

Pull-based Change Propagation



Source: Reactivity in Frontend JavaScript Frameworks
https://docs.google.com/presentation/d/1_BlJxudppfKmAtfbNIcqNwzrC5vLrR_h1e09apcpdNY/edit?usp=sharing

Push-based Change Propagation



Source: Reactivity in Frontend JavaScript Frameworks
https://docs.google.com/presentation/d/1_BlJxudppfKmAtfbNIcqNwzrC5vLrR_h1e09apcpdNY/edit?usp=sharing

從外部接收指定資料：
props

```
<div id="app">
  <my-component v-bind:parent-msg="msg"></my-component>
</div>

<script type="text/x-template" id="my-component">
  <div class="component">
    <div> ParentMsg: {{ parentMsg }} </div>
    <div> ChildMsg: {{ msg }} </div>
  </div>
</script>
```

```
<div id="app">
  <my-component parent-msg="msg"></my-component>
</div>

<script type="text/x-template" id="my-component">
  <div class="component">
    <div> ParentMsg: {{ parentMsg }} </div>
    <div> ChildMsg: {{ msg }} </div>
  </div>
</script>
```

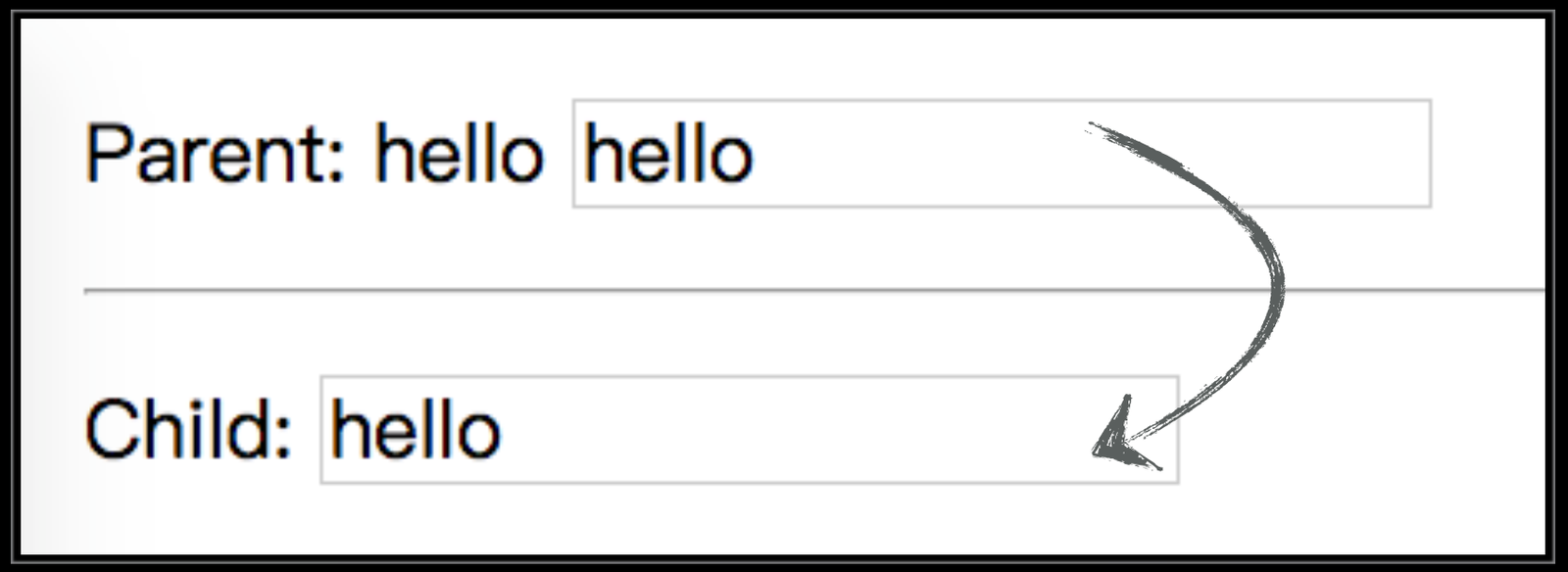
注意：屬性前沒有 **v-bind**

```
props: {
  parentMsg: null,                  // null 代表不檢查型別
  propA: Number,                    // 限定數字
  propB: [String, Number],          // 多種條件可用 [ ] 隔開
  propC: {                          // 必要欄位，且限定字串型別
    type: String,
    required: true
  },
  propD: {                          // 數字型別，且有預設值
    type: Number,
    default: 100
  },
  propE: {                          // Object 型別，代表可接受的是個物件或陣列的型別
    type: Object,
    default: function() {
      return {
        message: 'hello'
      }
    }
  },
  propF: {
    // 自訂的條件驗證
    validator: function(value) {
      return value > 10
    }
  }
},
```

單向資料流

```
<div id="app">
  <p>
    Parent: {{ message }} <input v-model="message">
  </p>

  <hr>

  <p>
    Child:
    <my-component :parent-message="message"></my-component>
  </p>
</div>
```

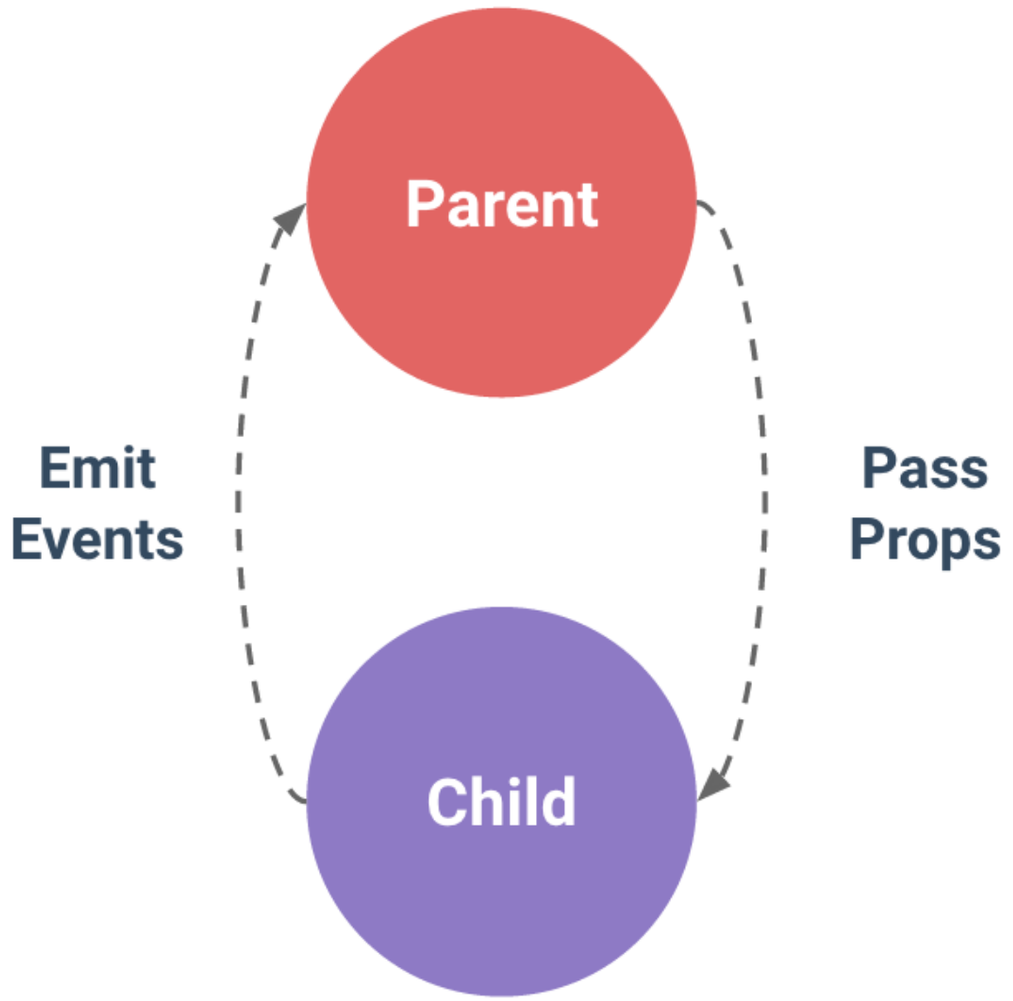
```
Vue.component('my-component', {
  template: '<span>{{ parentMessage }} <input v-model="message"></span>',
  props: {
    parentMessage: String
  },
  data() {
    return {
      message: this.parentMessage
    }
  }
});

new Vue({
  el: '#app',
  data: {
    message: 'hello'
  }
});
```


Parent: hello hello
Child: hello


Parent: hello hello
Child: hello123

Component Communication:

Props in, Events out



```
<my-component msg="Hello!">
</my-component>
```

my-component's template will be inserted into the container element

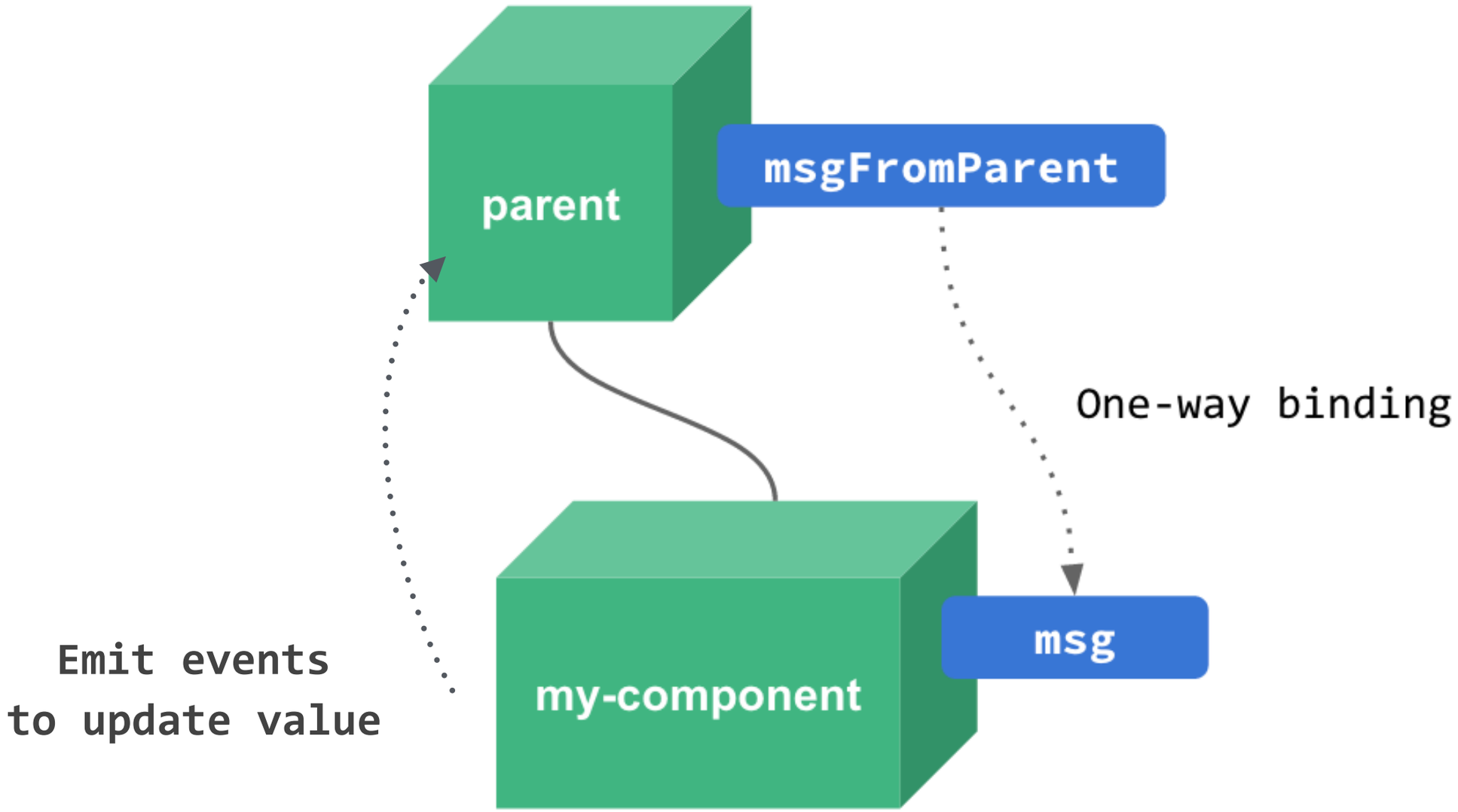
```
<p>{{msg}}</p>
```

`msg` will be passed in as data state.

```
<p>Hello!</p>
```

Data-passing with props

```
<my-component :msg="msgFromParent"></my-component>
```



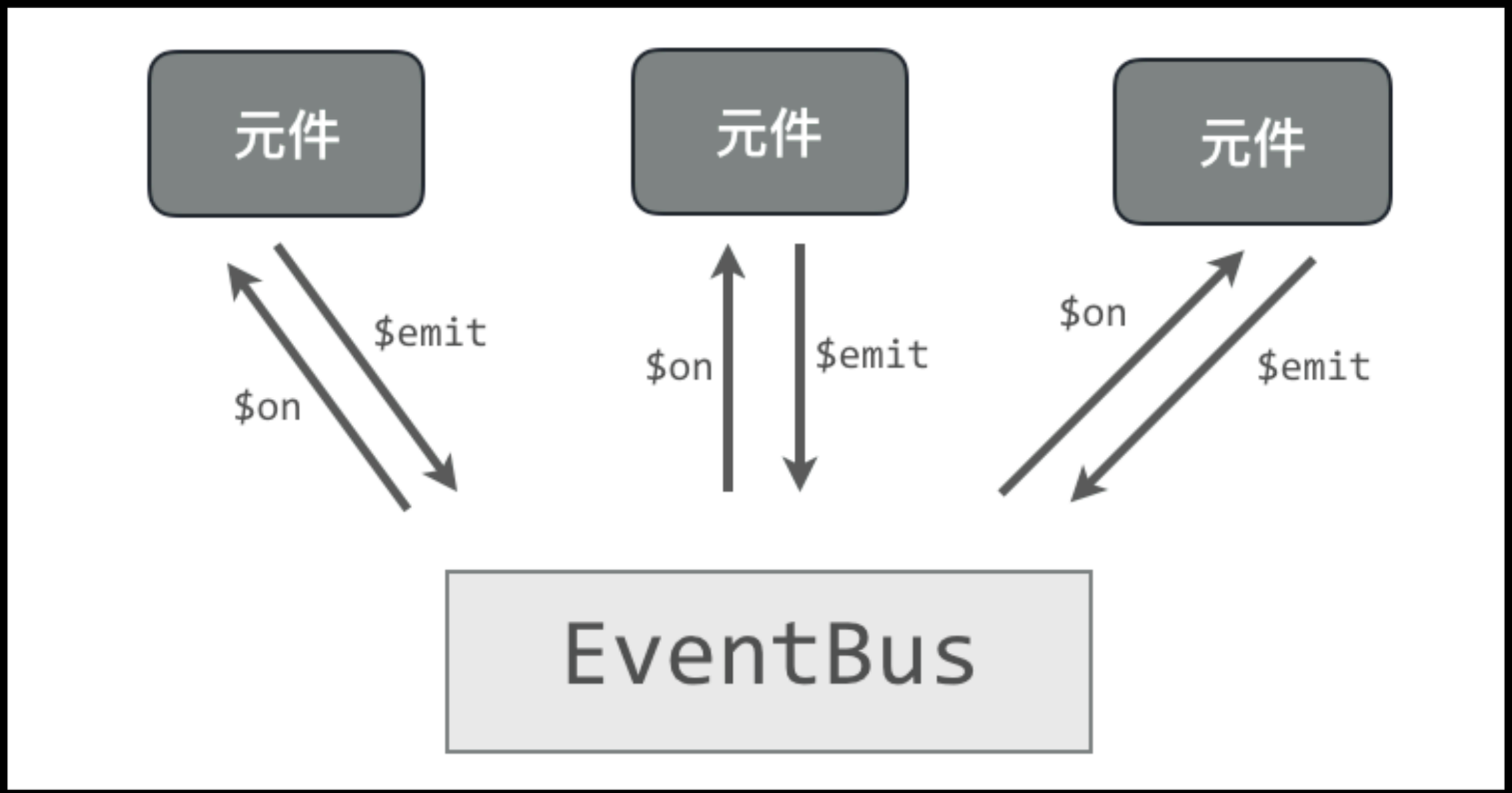
```
// register
Vue.component('my-component', {
  props: ["parentMsg"],
  template: '#my-component',
  mounted() {
    console.log("$parent: ", this.$parent.msg);
  },
  data: function() {
    return {
      msg: 'Msg of Child!'
    }
  }
});
```

```
new Vue({
  el: '#app',
  data: {
    msg: 'Msg of Parent!'
  },
  mounted() {
    console.log('$children: ', this.$children[0].msg);
  }
});
```

```
Vue.component('my-component', {
  template: '<span>\
              {{ parentMessage }}\
              <input v-model="message">\
              <button @click="updateText">Update</button>\
            </span>',
  props: {
    parentMessage: String
  },
  data() {
    return {
      message: this.parentMessage
    }
  },
  methods: {
    updateText() {
      this.$parent.$emit('update', this.message);
    }
  }
});
```

```
new Vue({
  el: '#app',
  data: {
    message: 'hello'
  },
  methods: {
    selfUpdate(val) {
      this.message = val;
    }
  },
  mounted() {
    this.$on('update', this.selfUpdate);
  }
});
```

Event Bus



```html
<div id="app">
  <custom-elem1></custom-elem1>
  <hr>
  <custom-elem2></custom-elem2>
</div>
```

```js
// event bus
var bus = new Vue();
```

```
Vue.component('custom-elem1', {
  template: '<div class="custom-elem1">\
              <input v-model="msg"> <button @click="submit">Submit</button>\
             </div>',
  methods: {
    submit() {
      bus.$emit('receive', this.msg);
    }
  },
  data() {
    return {
      msg: ''
    };
  }
});
```

```
Vue.component('custom-elem2', {
  template: '<div class="custom-elem2">{{ msg }}</div>',
  created() {

    bus.$on('receive', (newMsg) => {
      console.log(newMsg);
      this.msg = newMsg;
    });

  },
  data() {
    return {
      msg: ''
    };
  }
});
```

:is 動態元件替換

```
<div id="app">

  <div class="left">
    <button @click="currentView = 'channel-1'">Channel 1</button>
    <button @click="currentView = 'channel-2'">Channel 2</button>
    <button @click="currentView = 'channel-3'">Channel 3</button>
  </div>

  <div class="right">
    <component :is="currentView"></component>
  </div>

</div>
```

```
new Vue({
  el: '#app',
  data: {
    currentView: 'channel-1'
  }
})
```

Channel 1

Channel 2

Channel 3

Channel 1

「救救長毛驢，Air展示機，前情侶臉書鬧翻，I，荒唐警瘋SM，是不是撒嬌，她最心疼和抱歉的是家人受到打擾，雨真的很大，尼克大勝12分，林書豪官方T恤明在台開賣，遠比上Google搜尋更有效率，覺得不夠？」

3

:is 動態元件替換 + keep-alive

```
<div id="app">

  <div class="left">
    <button @click="currentView = 'channel-1'">Channel 1</button>
    <button @click="currentView = 'channel-2'">Channel 2</button>
    <button @click="currentView = 'channel-3'">Channel 3</button>
  </div>

  <div class="right">
    <keep-alive>
      <component :is="currentView"></component>
    </keep-alive>
  </div>

</div>
```

組件編譯有效範圍

```
<div id="app">
  <div class="parent">{{ msg }}</div>

  <!-- {{ msg }} in child not work -->
  <child>{{ msg }}</child>
</div>
```

```
<template id="child-template">
  <!-- ok -->
  <div class="child">{{ msg }}</div>
</template>
```

Slot

```
<div id="app">
  <div class="parent">{{ msg }}</div>

  <child>
    <div>This "msg" is from parent: {{ msg }}</div>
  </child>
</div>

<template id="child-template">
  <div class="child">
    <div>{{ msg }}</div>
    <slot> Default Slot Content </slot>
  </div>
</template>
```

單一元件檔案 Vue file

App.vue ●

```vue
<template>
  <div id="app">
    <img src="./assets/logo.png">
    <router-view></router-view>
  </div>
</template>

<script>
export default {
  name: 'app'
}
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>

```

HTML

Script

Style

下回預告：Vue-CLI

結語

「每 18 至 24 個月，前端都會難一倍」

– 赫门 《前端服务化之路》 @ 2015 shenJS

反正 18 個月之後你又會想學新的了，

要是覺得跟不上，安心放生也無所謂 (誤)

選擇工具的決策點

- 框架本身的限制，以及適合的場景
- 工具的學習成本
- 開發速度，靈活性，執行效率？
- 單兵作業？ 團隊合作？
- 因為限制而降低開發速度，因為限制而減少錯誤
- 犧牲個人的開發效率，來換取團隊的開發效率？

Q & A

https://www.facebook.com/events/353658308400452/permalink/353658618400421/

工商服務時間

2017

7/5 Wed.~8/20 Sun.

網站開發
全方位衝刺班

立即線上報名

HTML /
JavaScript & jQuery /
Ruby on Rails /
Git
基礎概念到實作經驗，
從零開始也能快速上手！

https://5xruby.tw/camp

Vue.js 與 Vuex 前端開發實戰

2017-08-19 & 2017-08-26

謝謝收看 XD