

# Vue.jsの基本

# この講座で扱う事



とにかくわかりやすさを重視！



# Vue.jsの概要



# フロントエンドとバックエンド

## クライアントサイドとサーバーサイド




クライアント



サーバー



# JavaScriptでできる事は全て

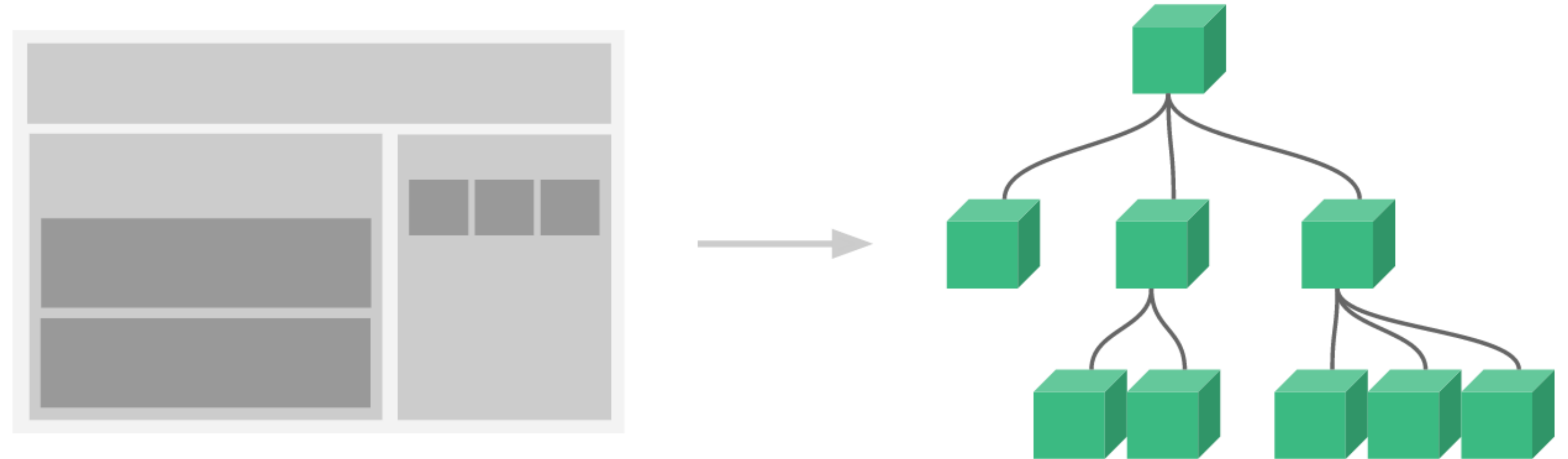


スライドショー、ポップアップウィンドウ、  
カウントダウン、ソート/検索、入力金額の  
自動計算、フォーム制御、スクロール、加速  
度センサ、GPS、ブラウザ制御、HTML/  
CSS操作、などなど。

Ajax(非同期通信)リアルタイムでどんどん動  
く

# Vue.jsの特徴

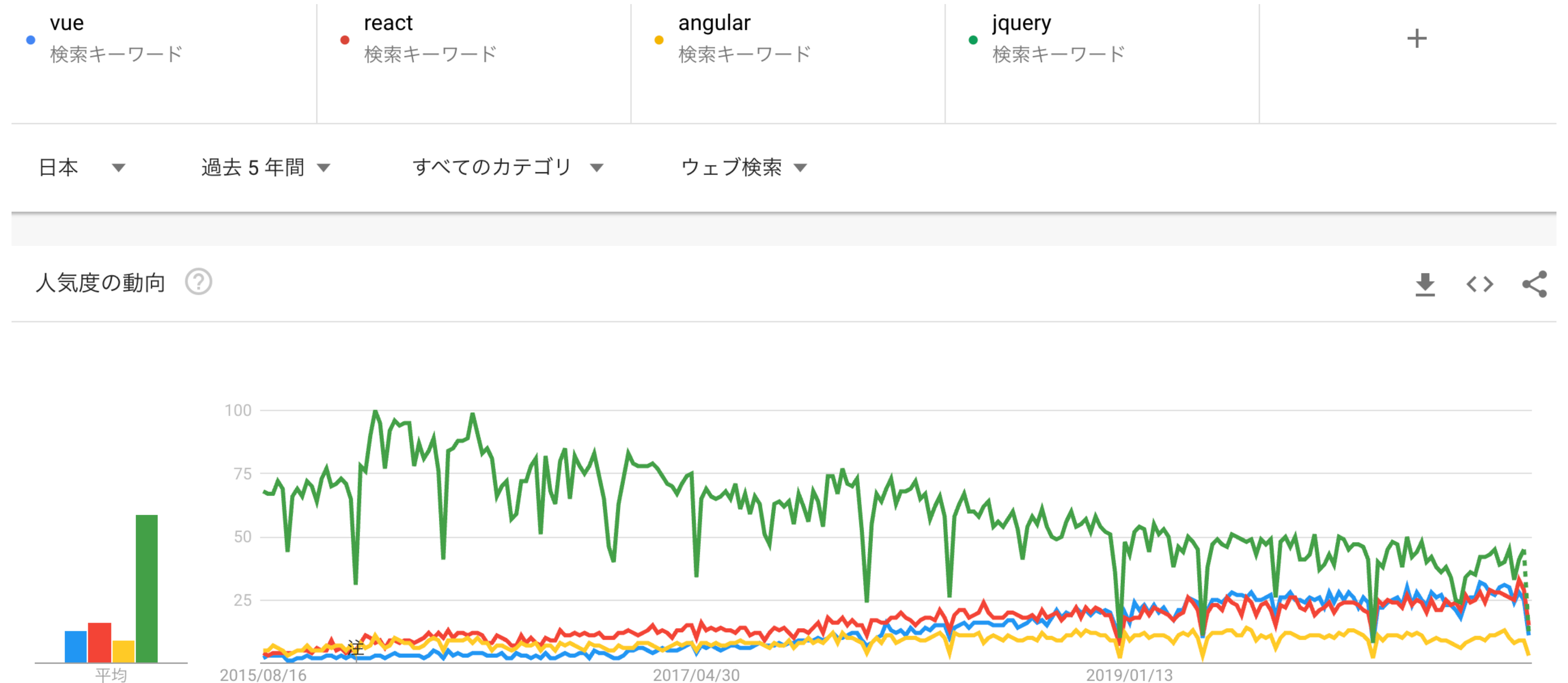
段階的に拡張できる・・・プログレッシブ  
メンテしやすい・・・コンポーネント分割  
ユーザビリティ向上・・・SPA



# プログレッシブ(段階的)

目的	jQueryの代わりに	メンテしやすく SFC (シングルファイル コンポーネント)	ユーザビリティ向上	ユーザビリティ向上 + 状態管理
ページ構成	MPA (マルチページ)	MPA	SPA (シングルページ)	SPA
開発環境	CDN	VueCLI	VueCLI + VueRouter	VueCLI + VueRouter + Vuex

# JSフレームワークのトレンド





# Vue.jsの年表



2014/2 v0.8 リリース

2015/4 Laravel(PHP)で採用

2016/10 Vue2 リリース

2020/9 Vue3 リリース

# Vue2? Vue3?



If you are just starting to learn the framework, you should start with **Vue 2 now**, since Vue 3 does not involve dramatic redesigns and the vast majority of your Vue 2 knowledge will still apply for Vue 3. There's no point in waiting if you are just learning.


<https://github.com/vuejs/vue/projects/6>



# Vue.js インストール

## CDN編

# Vue.js インストール CDN編



CDN

(コンテンツデリバリーネットワーク)

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.6.11/dist/vue.js"></script>
```

<https://jp.vuejs.org/index.html>

# Vue.js 最初の一歩

---

```
<div id="app"> {{ message }}</div>
```

```
let vm = new Vue({ // インスタンス化
  el: '#app',
  data(){
    return {
      message: 'hello'
    }
  }
})
```





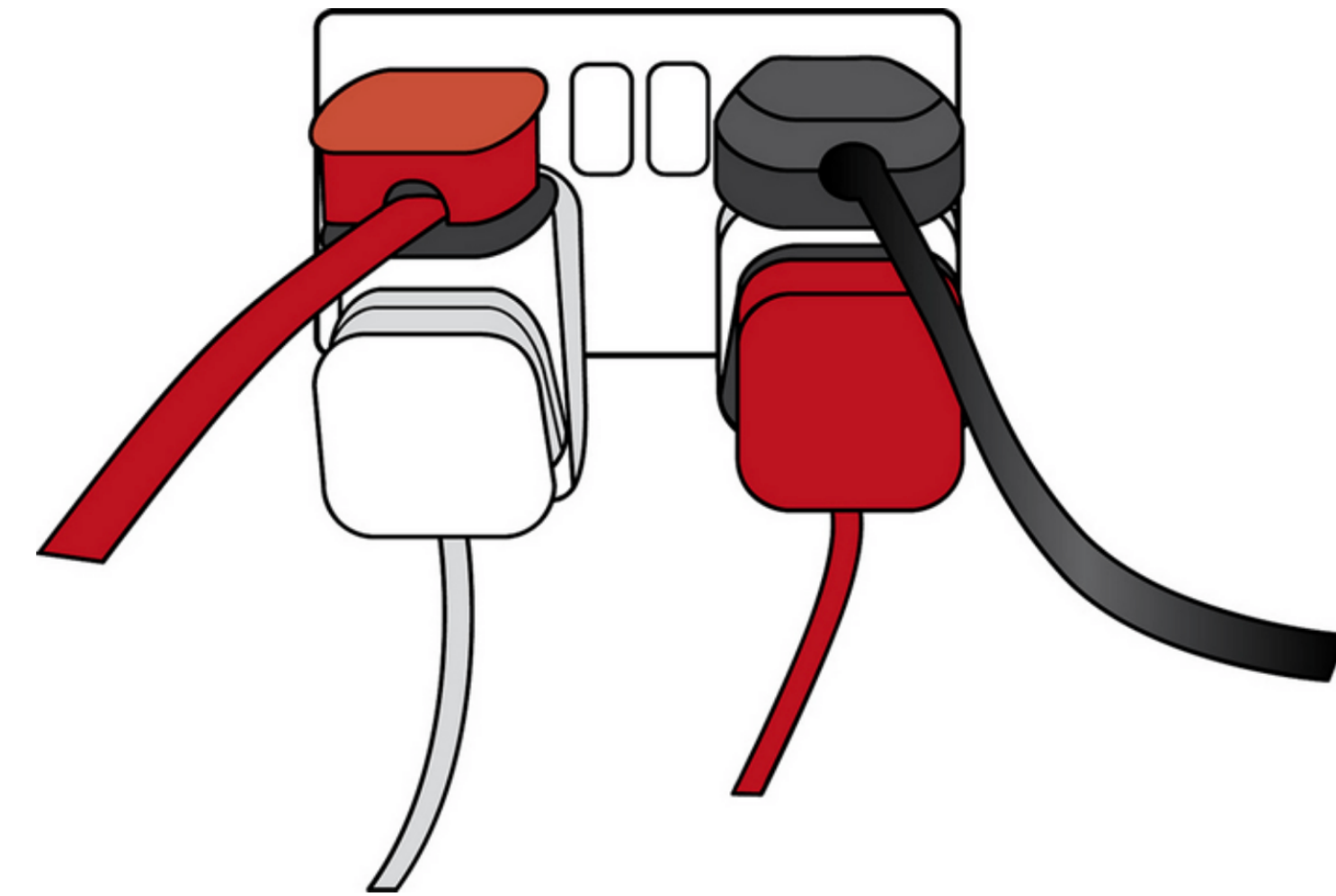
# Vue.jsのAPI

# Vue.jsのAPI

Application Programming Interface

コンセントの奥を知らなくても  
コンセントをさせば使える

<https://jp.vuejs.org/v2/api/>



# Vue.jsのAPI 基本



`el: '#app', // 仮想DOMの範囲`  
`data() {} // 仮想DOMで表示する値`

オブジェクトは `key:value` という書き方

`{{ }}` はマスタッシュ(口ひげ)構文 と呼ばれる



# DOMと仮想DOM

# DOM



Document Object Modelの略  
HTMLをプログラムから操作できる仕組み

DOMツリーなどで検索 (ツリー構造)  
JavaScriptではid,class,tagを指定して  
要素を追加/変更/削除



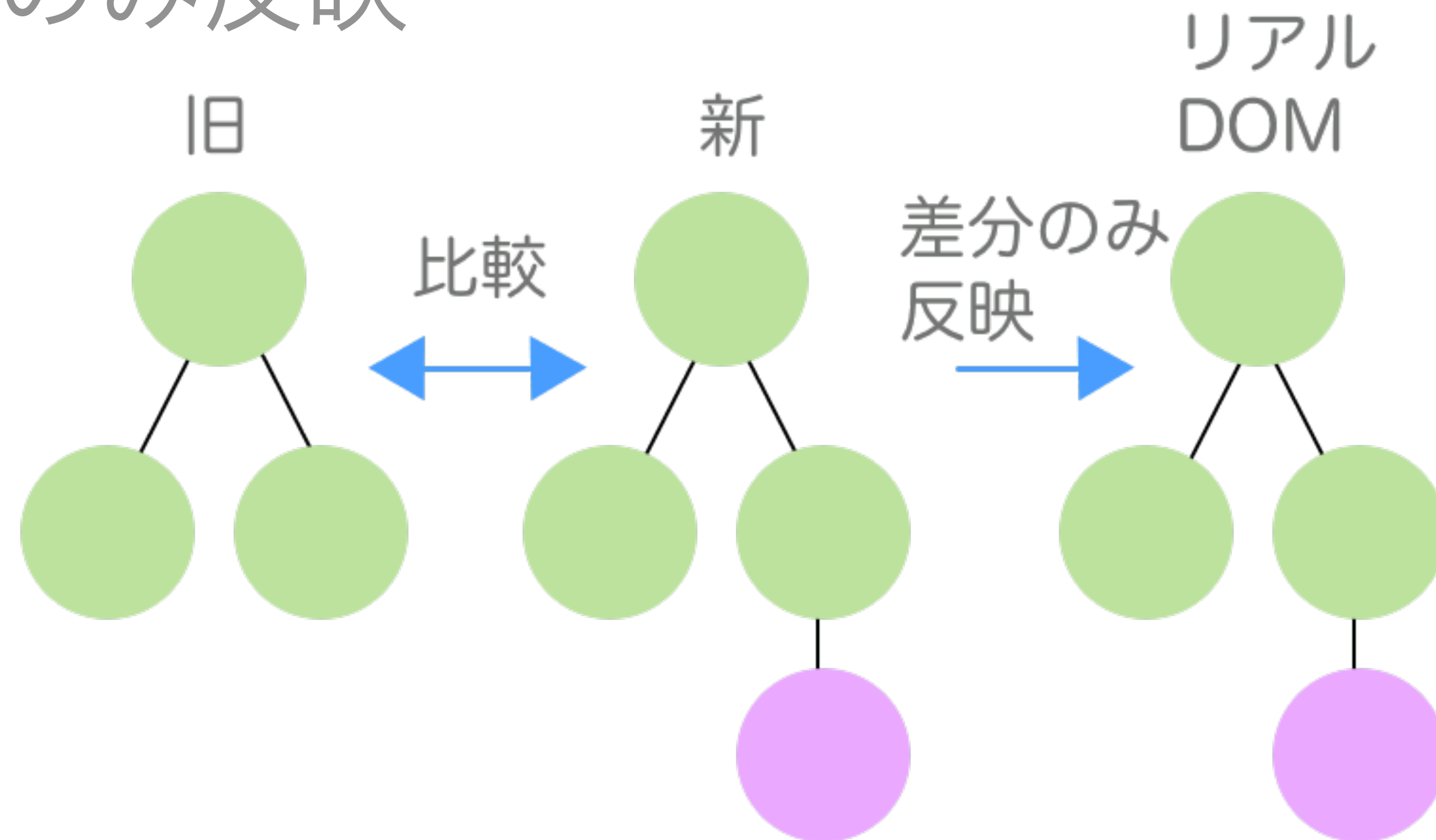
# 仮想DOM (Virtual DOM)のメリット



1. 処理が遅くなりづらい・メンテしやすい  
(コードが複雑になるほど  
直接のDOM操作はカオスに..)
2. 見た目(DOM)とデータ(JSONなど)の分離  
View - **ViewModel** - Model (MVVMパターン)

# 仮想DOM (Virtual DOM)

オブジェクトを2つ比較  
差分のみ反映





# タグに属性をつける 場合

# リアルDOMの場合



属性・・・Attribute

「mdn html タグ」 で検索

HTML・・・JSが紐付いているかわからない

JS・・・HTML側(idなど)が変わればJSも変更必要

# Vue.jsの場合 v-bind 書き方1



タグに属性をつけるには、`{}` ではなく  
`v-bind` (紐付けるという意味)を使う

`<a v-bind:href="google">google</a>`  
省略形 `<a :href="google">google</a>`



# v-bind その2 オブジェクト



オブジェクト名.キー

```
<a :href="book.url">{{ book.title }}</a>
```

複数の属性をオブジェクトで一括設定

```
<input v-bind="formInput">
```

# v-bind その3 style / class

---

HTML側はケバブケース font-size

JS側はキャメルケース fontSize

ケバブケースで書くなら”で囲む

```
<div :style="{fontSize:fontSize}"></div>
```

```
<div :style="{ 'font-size':fontSize}"></div>
```

classを表示したり消したり

```
<div :class="{active:isActive}"></div>
```

JS側 isActive: true (またはfalse)



ディレクティブ

# ディレクティブ (directive 指示)

---

v-show, v-if/v-else/v-else-if, v-for,  
v-on (@), v-bind (:), v-model  
v-slot, v-cloak など

<https://jp.vuejs.org/v2/api/>

# v-show



表示非表示の切り替え (true/false)

!で否定

```
<div v-show="isDisplay">表示</div>
```

```
<div v-show="!isDisplay">false</div>
```

CSS で display:noneがっくだけなので  
処理が早い



# v-if / v-else-if / v-else

---

<div v-if="isDisplay">ifで表示</div>

v-showと同じ使い方

<div v-if="signal === 'red'">赤</div>

<div v-else-if="signal === 'yellow'">黄</div>

<div v-else-if="signal === 'blue'">青</div>

<div v-else>赤青黄ではありません</div>

# v-for その1



## 配列

<li v-for="member in members">

<li v-for="(member, index) in members">

## オブジェクト

<li v-for="(value, key, index) in book">

# v-for その2



```
<li v-for="book in books" :key="book.id">  
  {{ book.id }}
```

表示後、追加/削除/並び替えなどするなら  
ユニークキー(他と重ならないidなど)を  
:key="" で設定する  
(indexは指定しない)


# v-text v-html

---

<div v-text="test"></div> //テキストで表示  
<div v-html="test"></div> //htmlとして表示

```
data(){  
  return {  
    test : 'あああ <br> いいい'  
  }  
}
```

# v-clock {{ }} を非表示に



CSS

```
[v-clock]{  
display:none;}
```

```
<div id="#app" v-cloak>  
  <div v-cloak> {{ test }} </div>  
</div>
```



# v-on その1

---

```
<button v-on:click="btnClicked">
```

```
<button @click="btnClicked">
```

設定する関数(メソッド)を methods内に書く

```
methods: {  
  btnClicked(){  
    console.log("")  
  }  
}
```

# v-on その2

---

<button @click="btnClickedEvent">

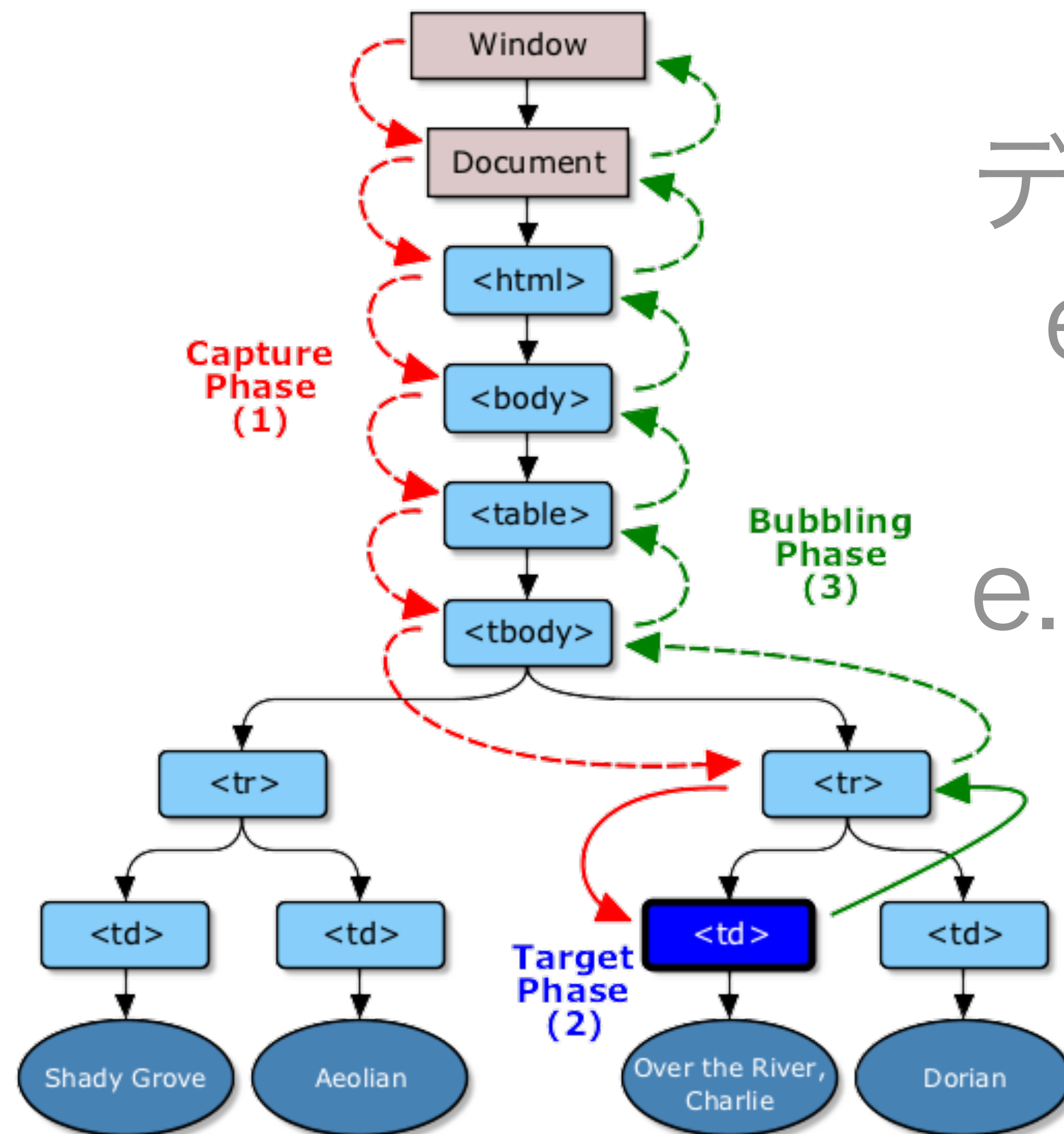
eでイベントオブジェクト取得

```
methods: {  
  btnClickedEvent(e){  
    console.log(e)  } }
```

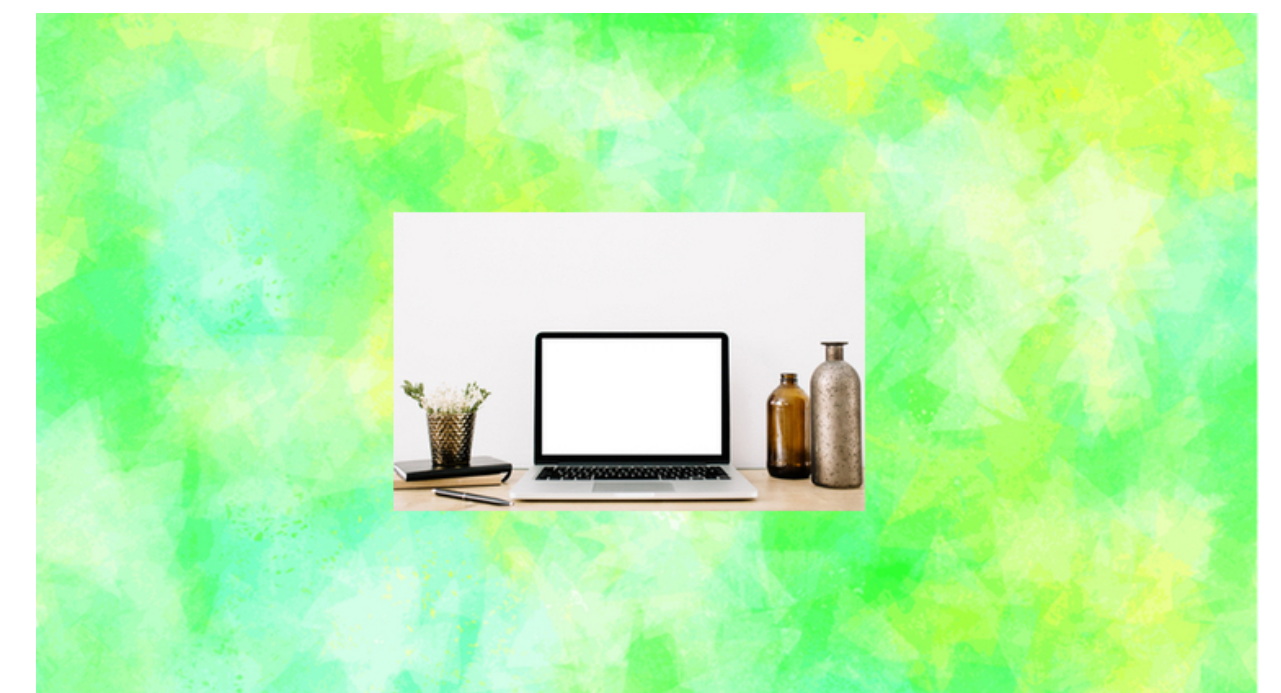
引数とイベントオブジェクト取得なら \$event

<button @click="btnClicked(1, \$event)"

# バブリングとキャプチャリング



デフォルト動作を防ぐ  
`e.preventDefault();`  
バブリングを止める  
`e.stopPropagation();`



# v-on その3




<button @click.prevent="">

<button @click.stop.prevent="">

よく使うイベント @click @change @submit  
@input など

[https://developer.mozilla.org/ja/docs/  
Web/Events](https://developer.mozilla.org/ja/docs/Web/Events)





オプション/データ

# 3種類の比較表

オプション名	methods	computed	watch
	メソッド	算出プロパティ	監視プロパティ
使い方	一般的な関数と同様	return内に 特定したいdataを含める this.xxx	特定したいdata名で作成 コールバック関数含める
キャッシュ	キャッシュされない	キャッシュする	キャッシュする
実行 タイミング	再描画の度に実行	特定 data 変更時 特定dataを元に 派生したデータを使う時	特定 data変更時 特定のコールバック関数を実行 非同期処理・Ajax など



# computed

dataの値が変わる時だけ実行  
dataの中の値はもれなく  
get(取得時に実行) と  
set(変更時に実行) が設定される


```
▼ $data: Object
  number: (...)
  price: (...)
  ▶ __ob__: Observer {value: {...}, dep: Dep, vmCou...
  ▶ get number: f reactiveGetter()
  ▶ set number: f reactiveSetter(newVal)
  ▶ get price: f reactiveGetter()
  ▶ set price: f reactiveSetter(newVal)
  ▶ __proto__: Object
  $isServer: (...)
  $props: (...)
  $ssrContext: (...)
```

# watch



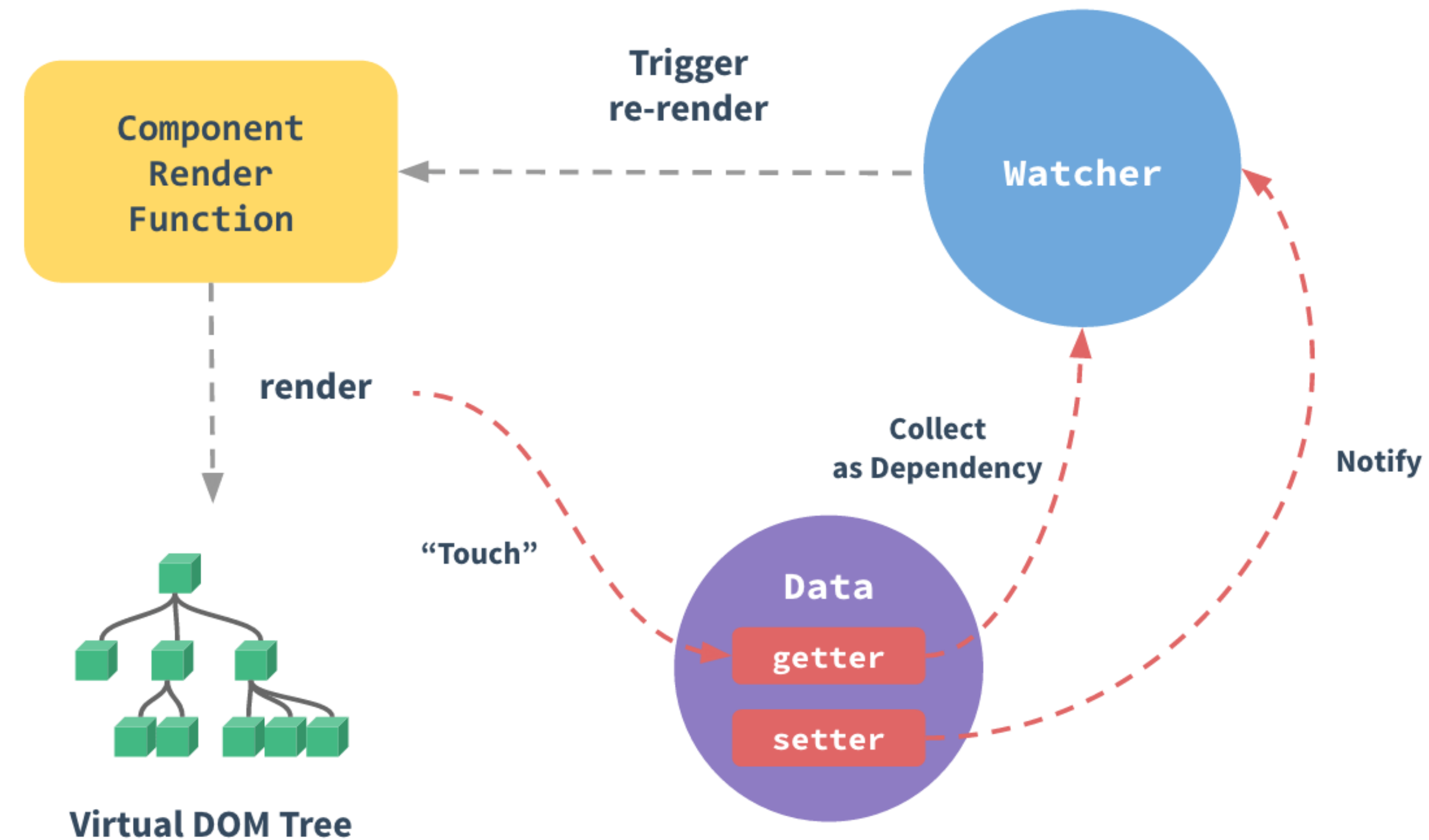
computedより複雑な処理をしたい場合  
主に非同期関係やオブジェクトの監視

オブジェクトの監視は  
handler(){} と  
deep: true を設定する  
immediate (即座)というオプションもある




# リアクティブ システム

# 変更を検知して再描画



<https://jp.vuejs.org/v2/guide/reactivity.html>

# オブジェクトや配列の追加



事前に空データをつくっておく


```
message: ""  
books: []
```

専用メソッドで追加

Vue.set(オブジェクト名, key, value)

配列

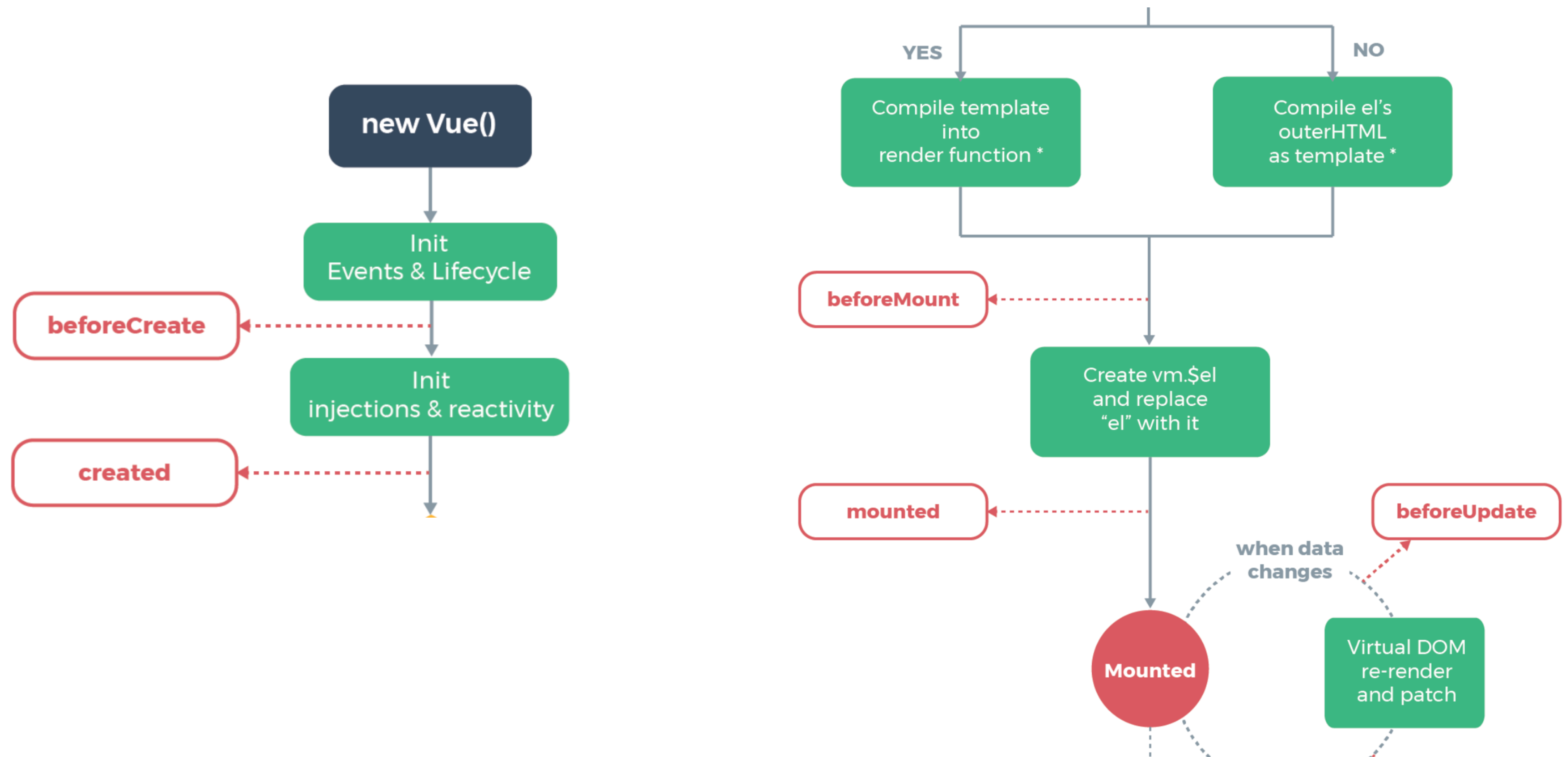
push/pop/shift/unshift/splice/sort/reverse



# ライフサイクル フック



# Vuejs生成時のタイミング



# よく使うのは2つ

---

created ・ ・ data生成のタイミング  
(非同期通信でデータ取得したい場合)

mounted ・ ・ DOM生成のタイミング  
(まずはmounted後で)

computedはmountedより前に生成  
vm.\$nextTick でDOM生成後に実行