最近のモダンなJavaScriptによる画面開発

モダンなJavaScriptフレームワーク
って何がいいの？

本日のお題

最近、Web開発（フロントエンド系開発）でよく聞くようになった

「Vue.js」 「React」 「Angular」

これらJavaScriptフレームワークを使うと何がいいの？
について今日は説明します！

本日の登場人物

モダンなJavaScriptフレームワーク

- Vue.js
- React
- Angular

レガシーなJavaScriptフレームワーク

- jQuery

新旧フレームワークを比較して
新しいフレームワークの何がいいかを解説！

結論

いきなりだけど
ｵｵｰw(*ﾟoﾟ*)w

モダンなJavaScriptフレームワークはレガシーなフレームワークに比べて

ココがいい！

- 見た目をオブジェクトで操作できる
  - 複雑なUIになるほど効果絶大！
    DOMで直接操作するよりオブジェクトを操作した方が実装が楽！
- ビューとロジックを分離できる
  - レガシーだとロジックにビューが紛れ込む。
    モダンだと見た目と処理で責務分解ができてコードの可読性が上がる！

ちゃんと使いこなすと実装しやすく
なるんだヽ(=´▽`=)ﾉ

結論で言ってたことが本当かどうか

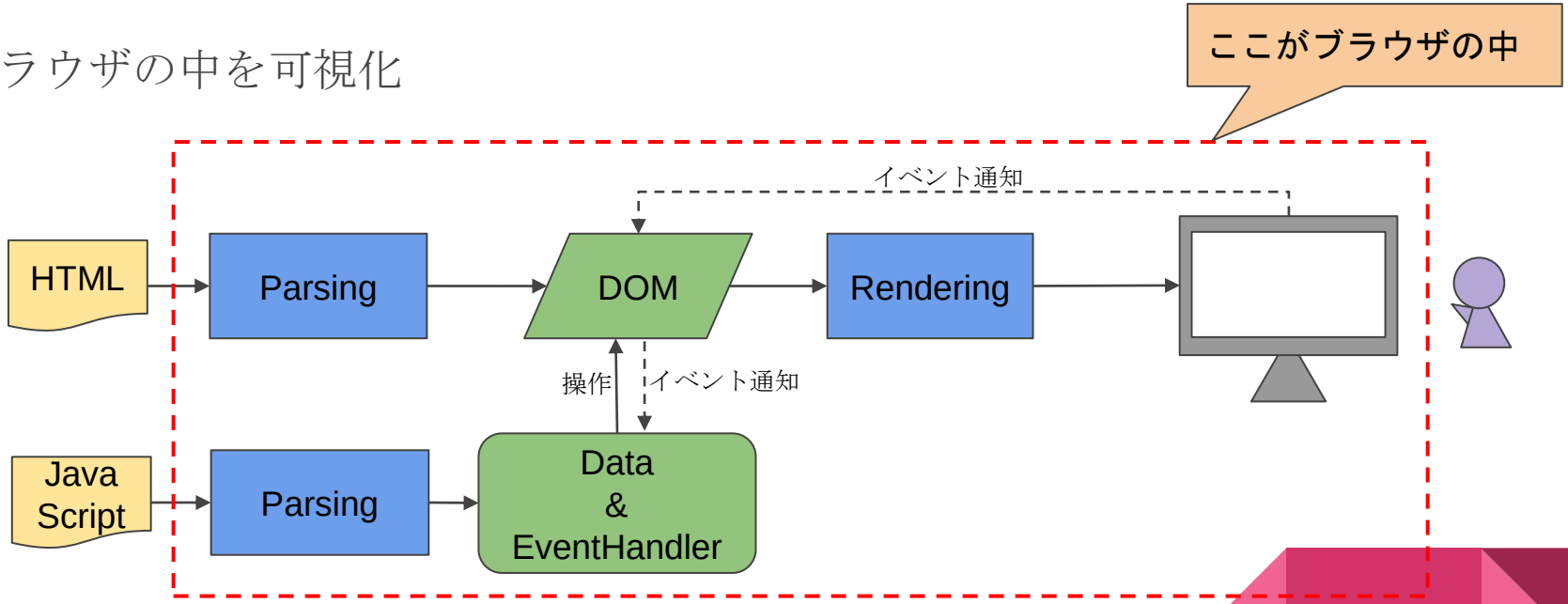
実際にモダン/レガシーを見比べてみよう！

- フレームワークのアーキテクチャーの違い
- 実際にコードを書いた際の違い

フレームワークの
アーキテクチャーの違い

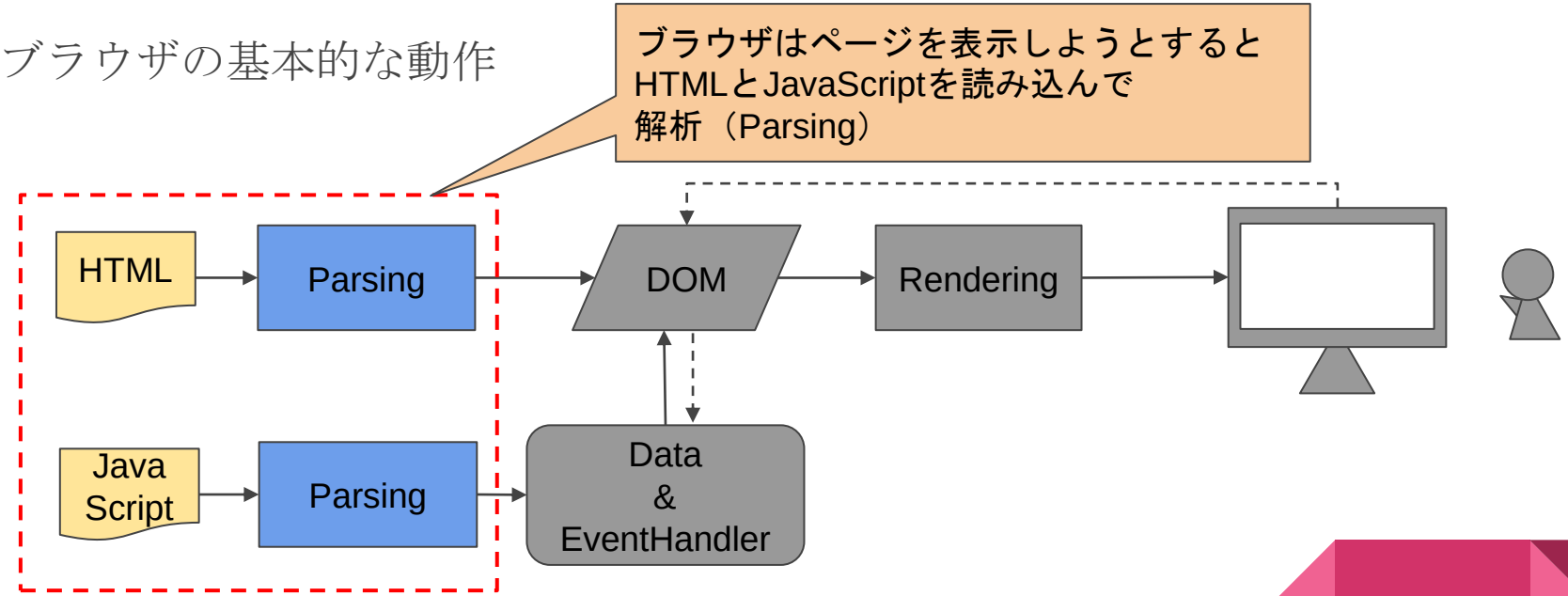
レガシーなフレームワークは？

ブラウザの中を可視化



レガシーなフレームワークは？

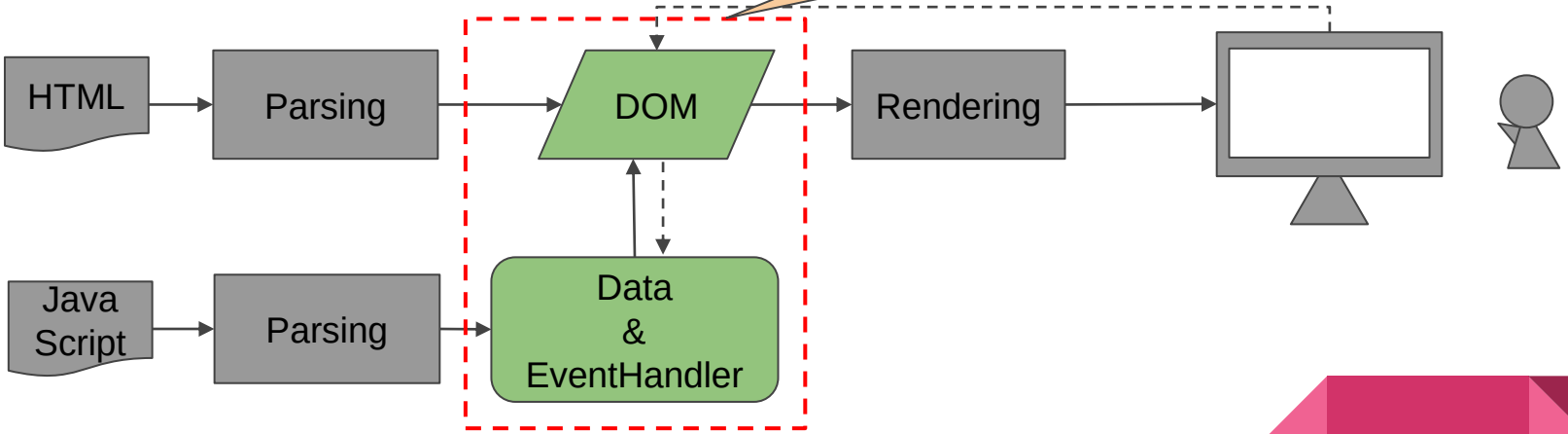
ブラウザの基本的な動作



レガシーなフレームワー

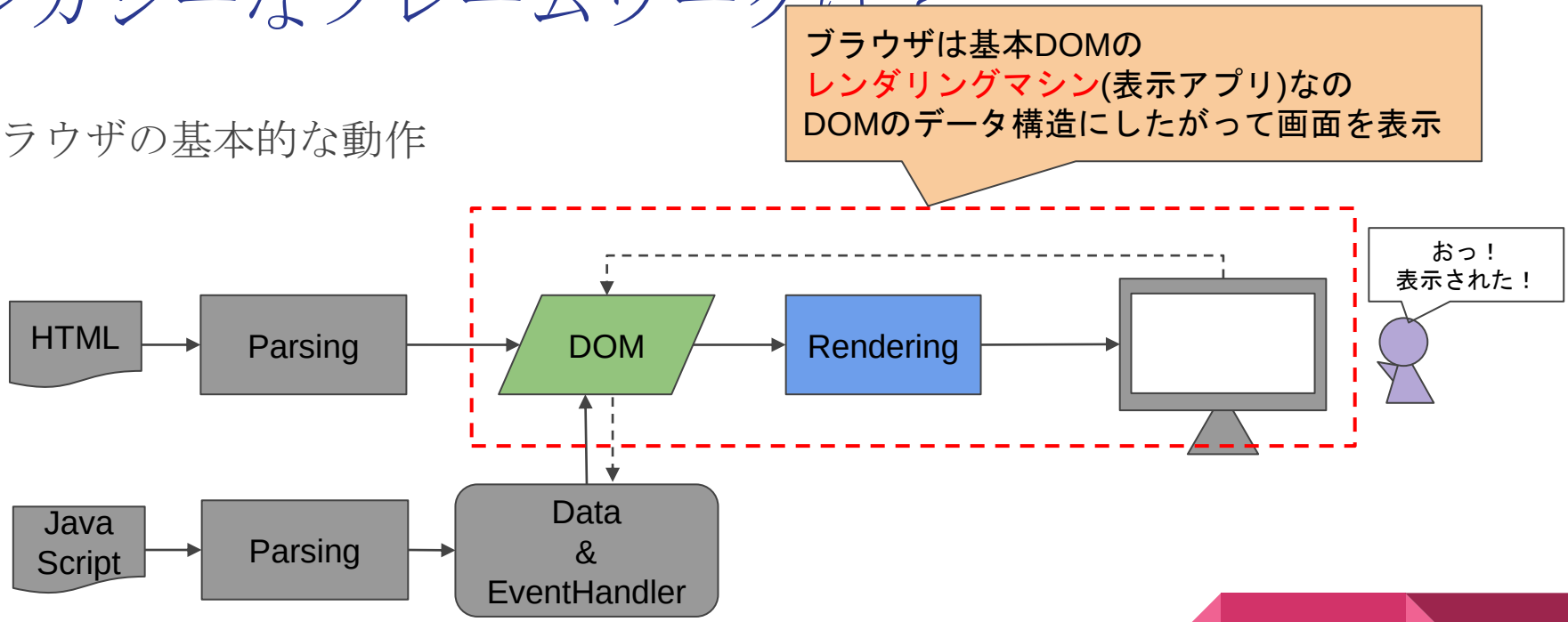
ブラウザの基本的な動作

解析するとHTMLのタグから
DOM(Document Object Model)というデータと
JavaScriptに書かれたコードからブラウザ内で発生したイベントを処理するための
データと処理をブラウザのメモリ上に格納



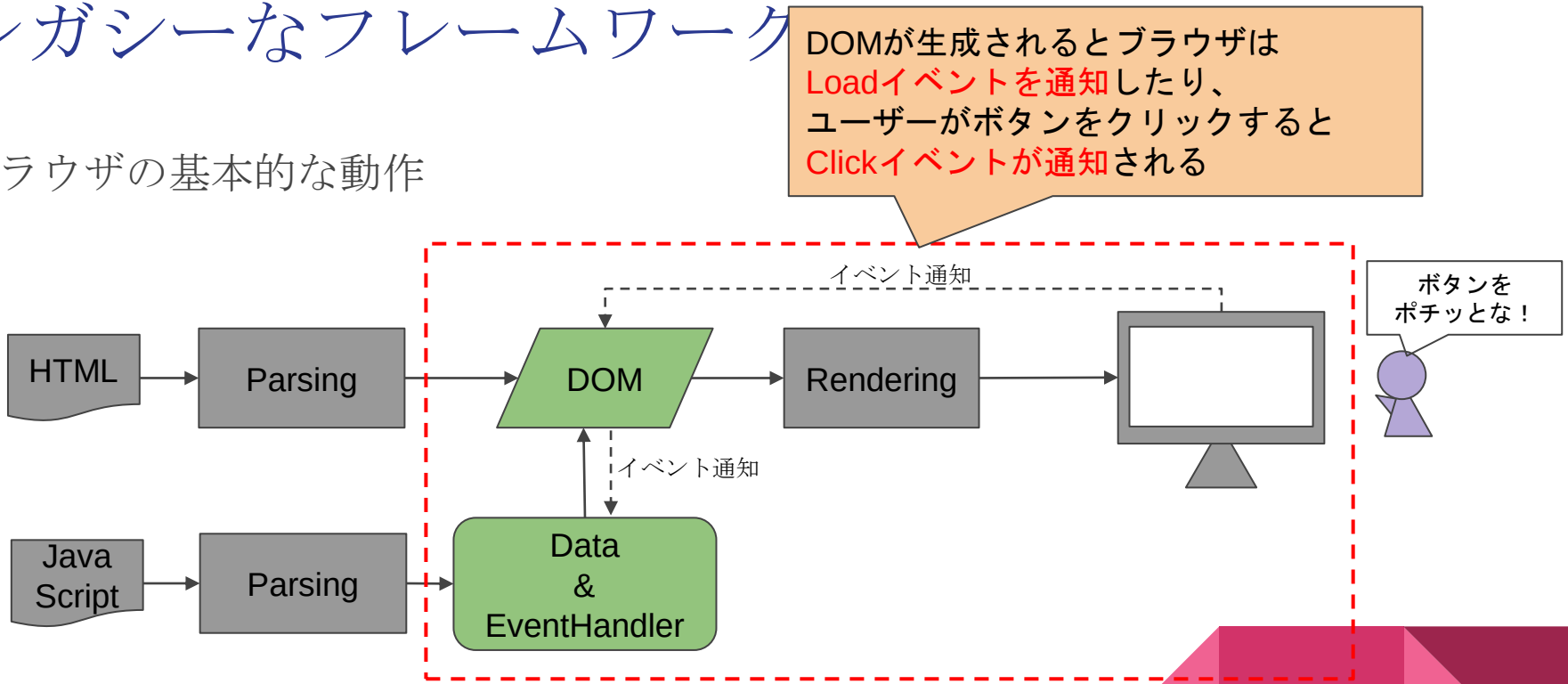
レガシーなフレームワークは？

ブラウザの基本的な動作



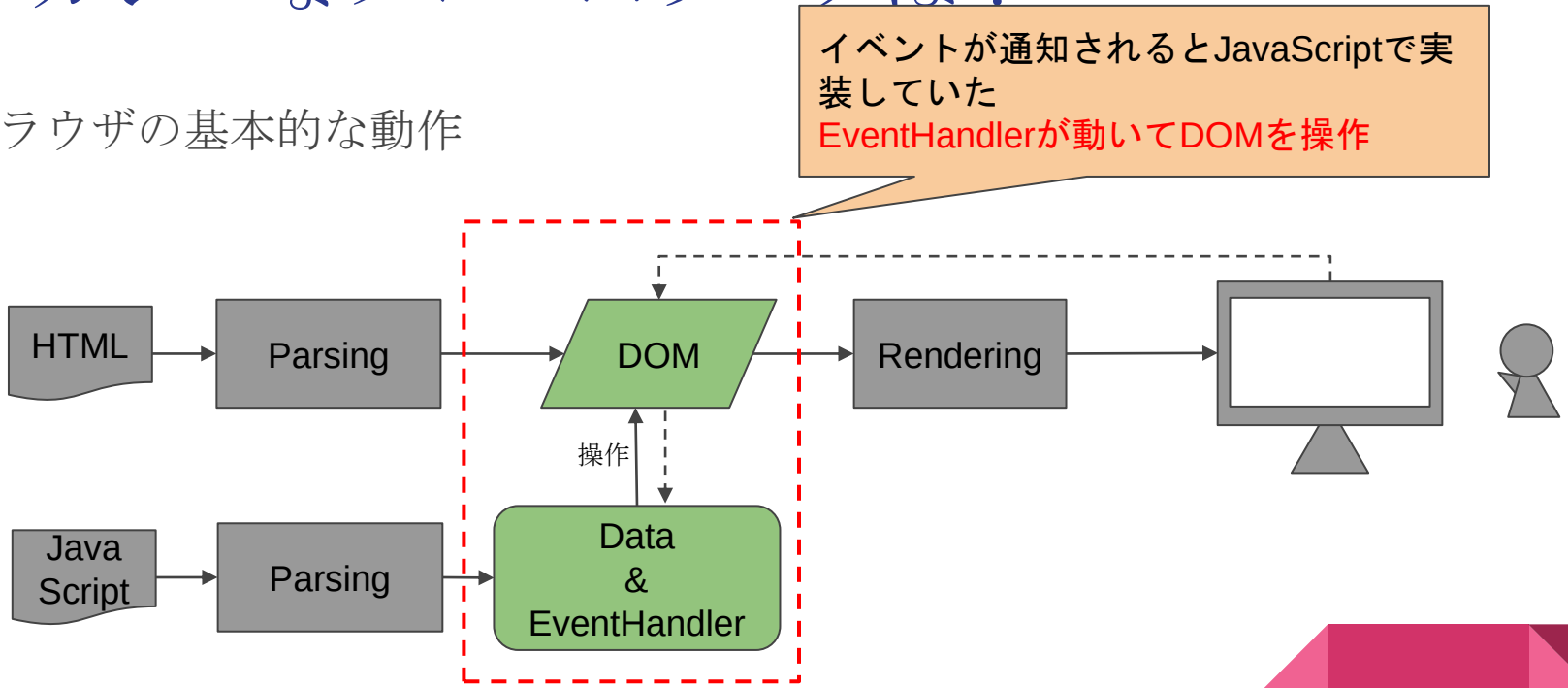
レガシーなフレームワーク

ブラウザの基本的な動作



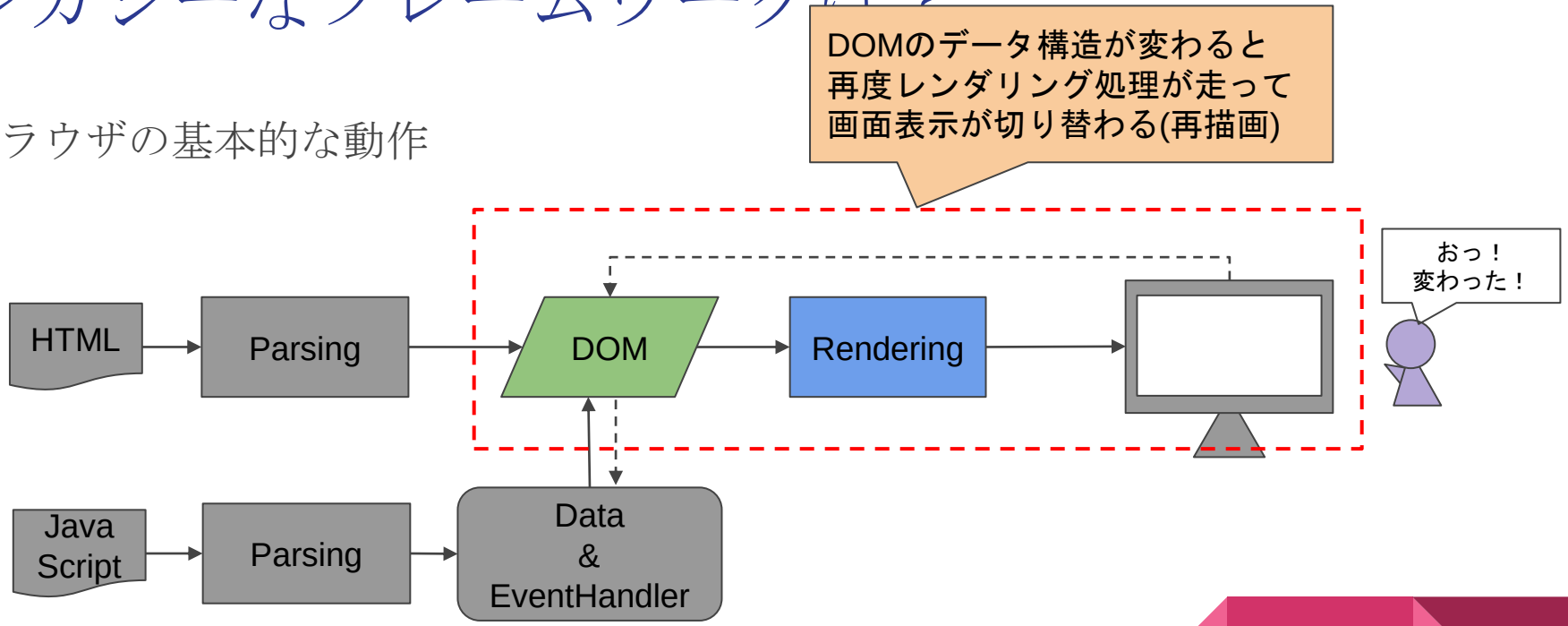
レガシーなフレームワークは？

ブラウザの基本的な動作



レガシーなフレームワークは？

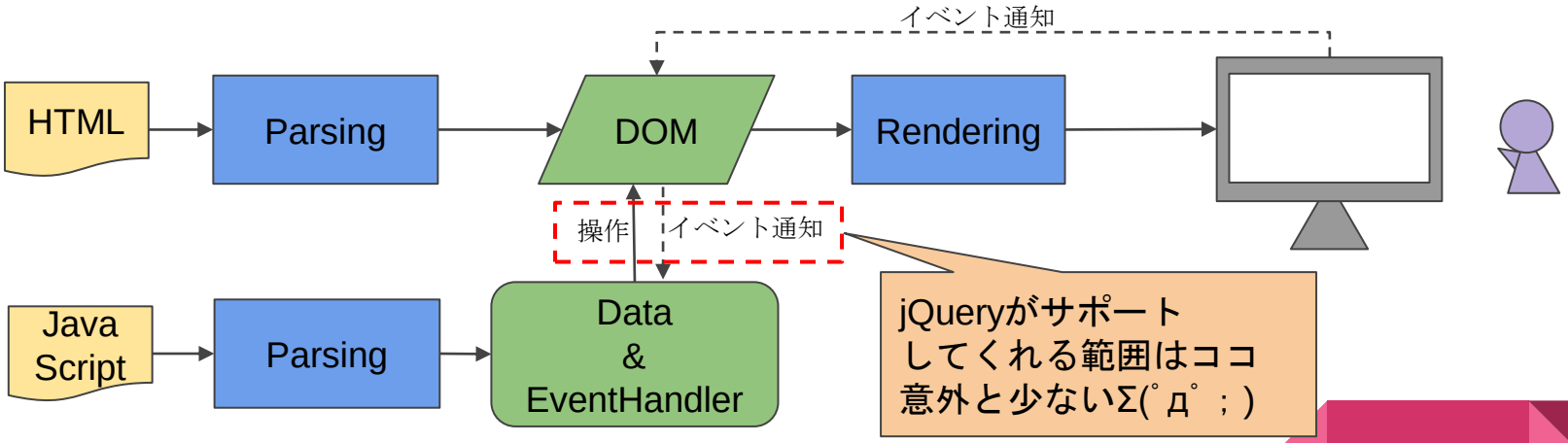
ブラウザの基本的な動作



レガシーなフレームワークは？

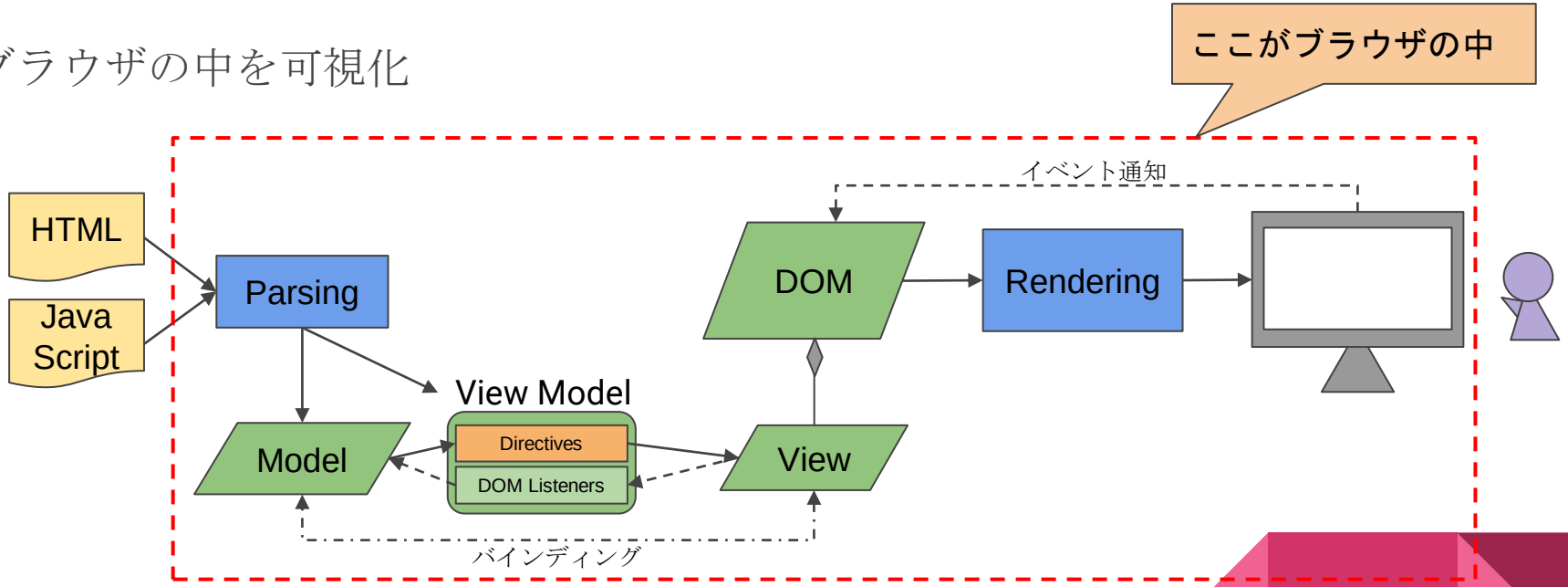
そういえばjQueryは？
Σ(･ロ･)

jQueryはDOM操作やイベント通知の受け取りを
簡単に行えるようにするためのフレームワーク



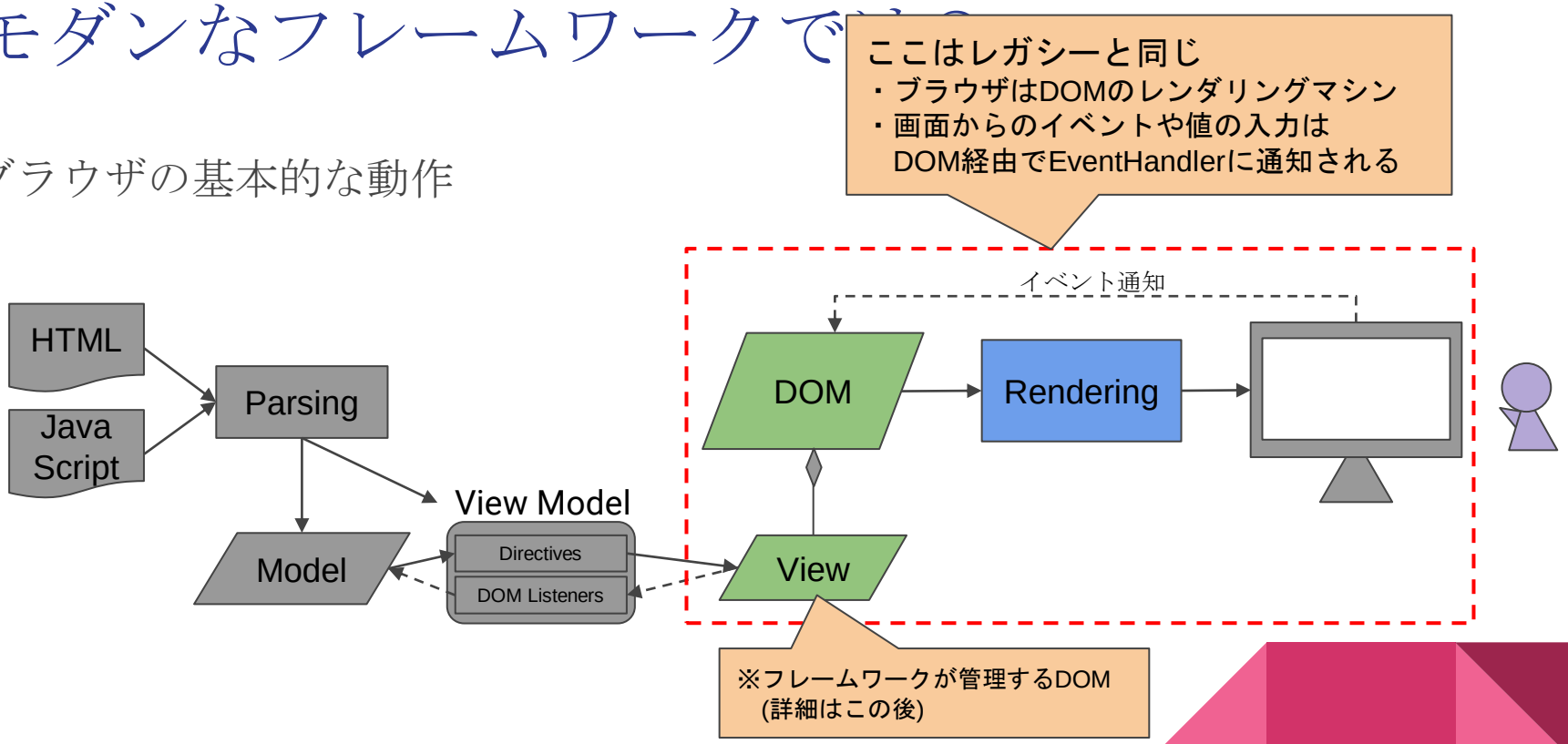
モダンなフレームワークでは？

ブラウザの中を可視化



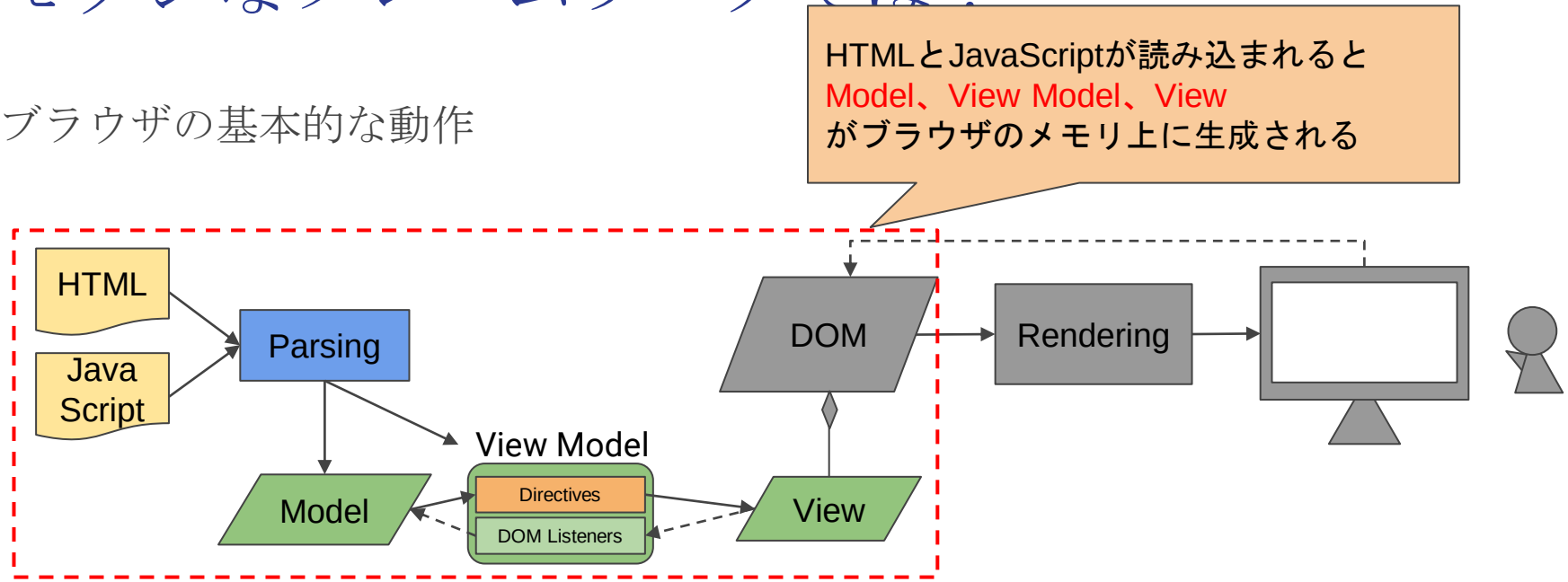
モダンなフレームワークでは？

ブラウザの基本的な動作



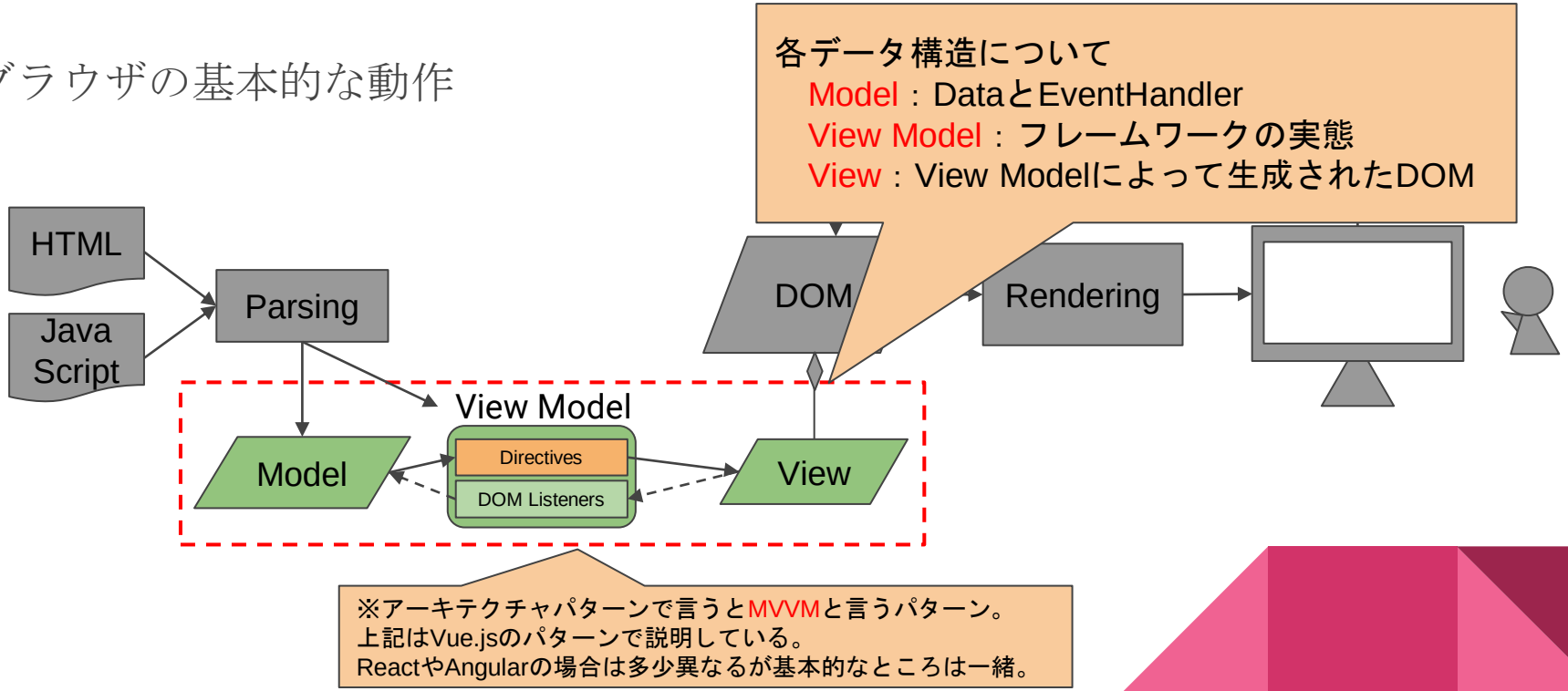
モダンなフレームワークでは？

ブラウザの基本的な動作



モダンなフレームワークでは？

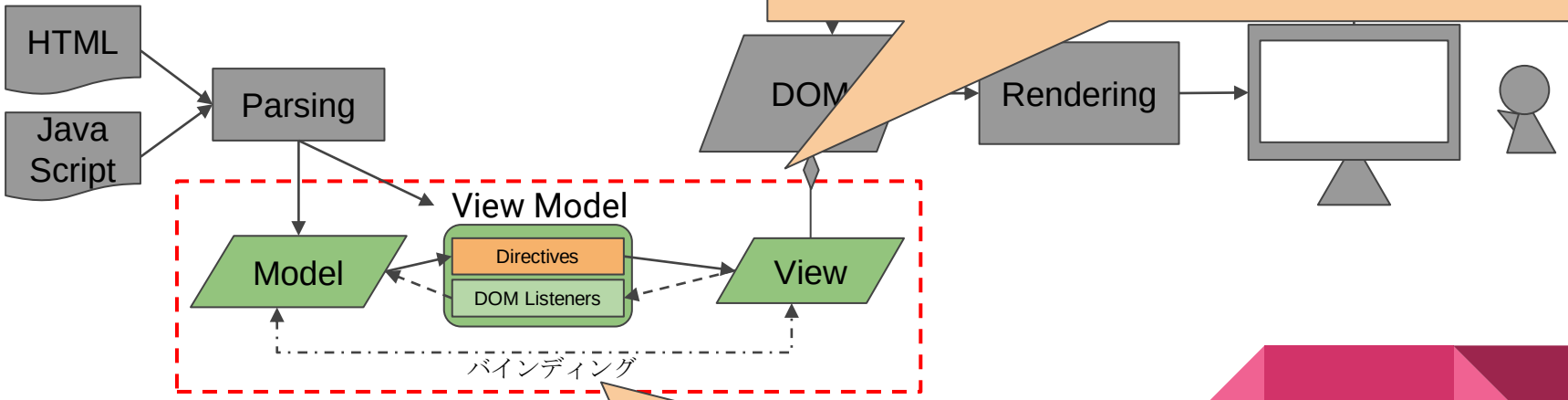
ブラウザの基本的な動作



モダンなフレームワークでは？

ブラウザの基本的な動作

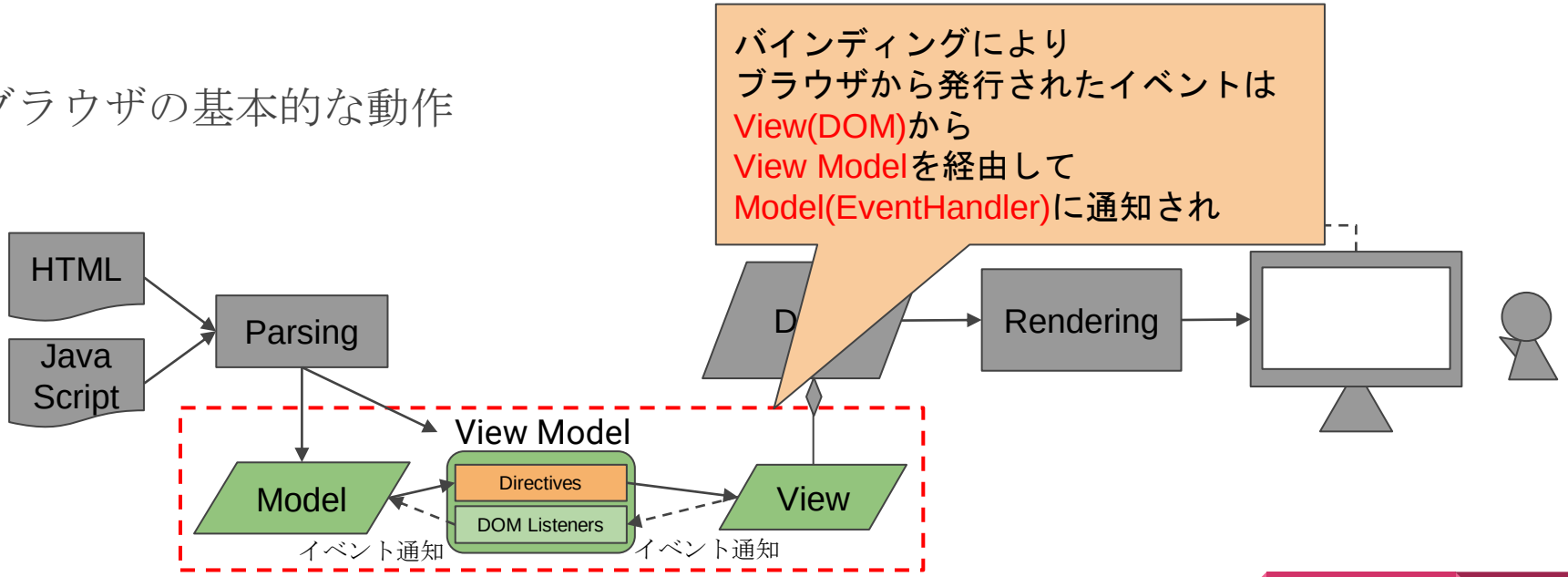
[ポイント]
ModelとViewはバインディングしておく！
バインディングすることで
Modelがイベント通知を受けれるようになったり
Modelで抱えているデータを変更するとViewに反映されるようになる！



※HTML(テンプレート)側で指示することでバインディングできる(後ほど説明)

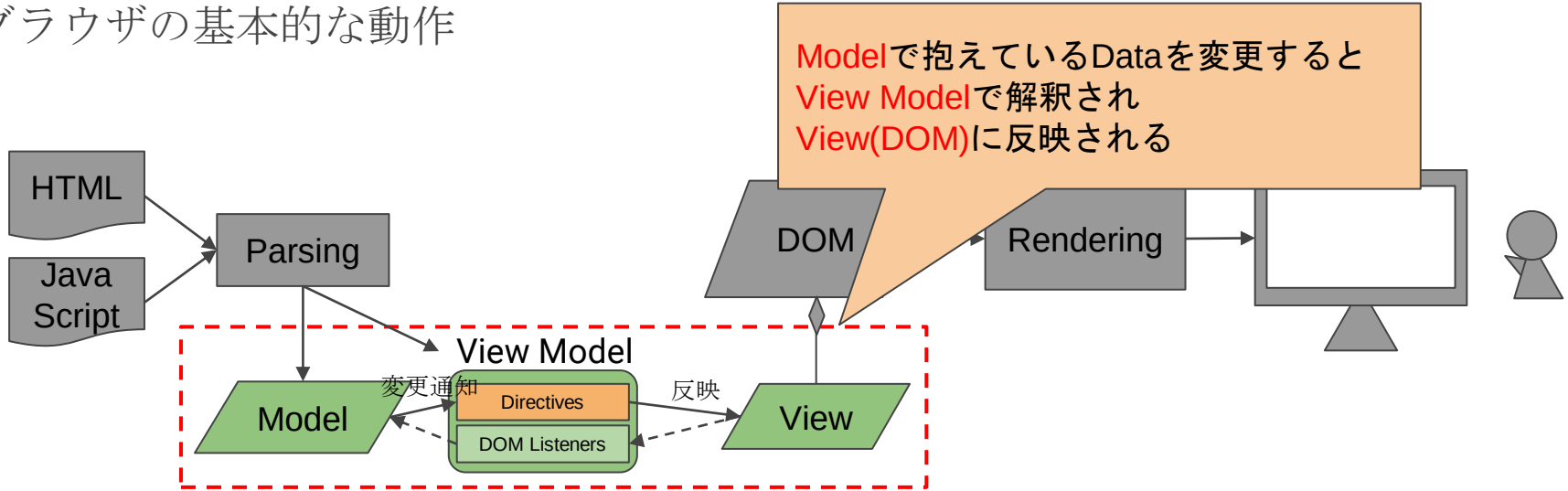
モダンなフレームワークでは？

ブラウザの基本的な動作



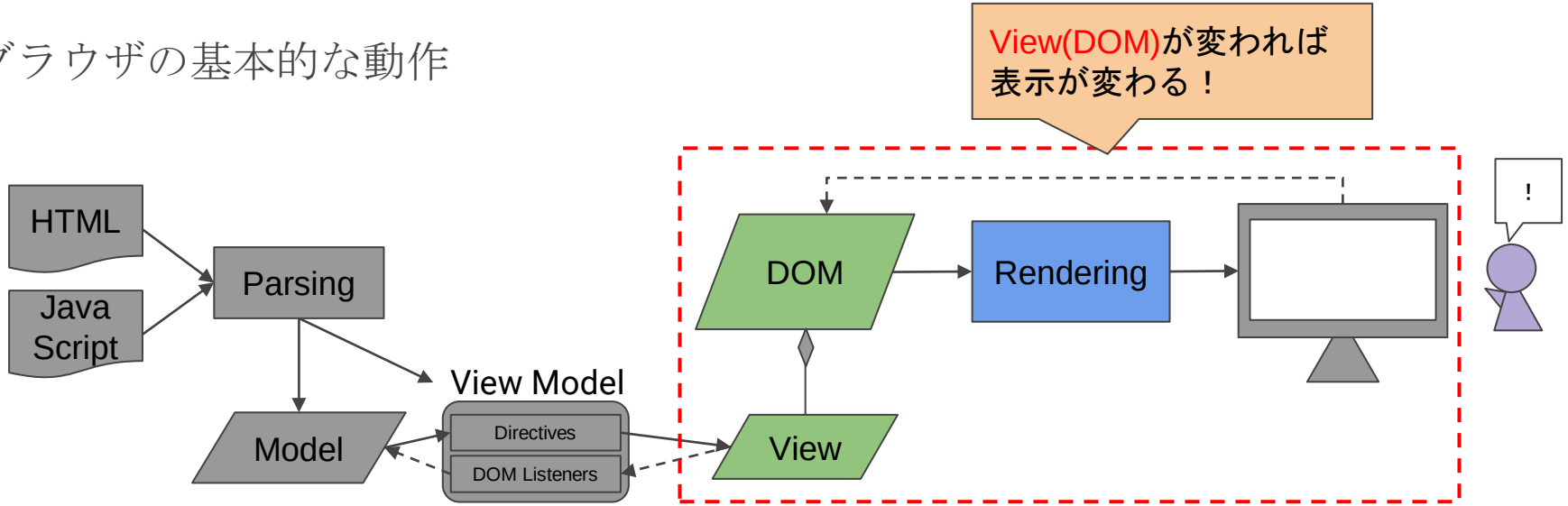
モダンなフレームワークでは？

ブラウザの基本的な動作



モダンなフレームワークでは？

ブラウザの基本的な動作



実際にコードを書いた際の違い

実際にコードを見てみよう！

レガシーなフレームワーク(jQuery)とモダンなフレームワーク(今回はVue.js)で実際のコードを比較してみよう！

文字を入れ替える

実際に動かしてみるなら
https://codepen.io/maru510/pen/PoGrqde

コードはこちらの記事から引用させていただきました
(注:一部説明の為に改変してます)
https://qiita.com/mio3io/items/e7b2596d06b8005e8e6f

まずは簡単な例から。ボタンを押したら文字を入れ替えてみる。

jQuery版コード

HTML

```html
<div>
  Hello <span id="message">World</span> !
  <button id="update">change</button>

</div>
```

JavaScript

```javascript
$('#update').on('click', function() {
  $('#message').text('jQuery')

});
```

jQuery版コード

HTML

```
<div>
  Hello <span id="message">World</span> !
  <button id="update">change</button>
</div>
```

ボタンとボタンを押された時の
処理(EventHandler)を紐づけておく

JavaScript

```
$('#update').on('click', function() {
   $('#message').text('jQuery')

});
```

jQuery版コード

HTML

```
<div>
  Hello <span id="message">World</span>!
  <button id="update">change</button>

</div>
```

ボタンが押されたらspanタグの中身を
「jQuery」に書き換える

JavaScript

```
$('#update').on('click', function() {
  $('#message').text('jQuery')
});
```

Vue.js版コード

HTML

```
<div id="vue">
  Hello {{ message }} !
  <button @click="update">change</button>

</div>
```

JavaScript

```
Vue.createApp({
  data() {
    return {
      message: 'World'
    };
  },
  methods: {
    update() {
      this.message = 'Vue.js'
    }
  }
}).mount('#vue');
```

Vue.js版コード

HTML

```
<div id="vue">
  Hello {{ message }} !
  <button @click="update">change</button>

</div>
```

「{{ message }}」に「message」の値がリアルタイムに反映されるようバインディング

```
Vue.createApp({
  data() {
    return {
      message: 'World'
    };
  },
  methods: {
    update() {
      this.message = 'Vue.js'
    }
  }
}).mount('#vue');
```

Vue.js版コード

HTML

```
<div id="vue">
  Hello {{ message }} !
  <button @click="update">change</button>
</div>
```

ボタンをクリックした時にメソッドが
呼ばれるようにバインディング

```
Vue.createApp({
  data() {
    return {
      message: 'World'
    };
  },
  methods: {
    update() {
      this.message = 'Vue.js'
    }
  }
}).mount('#vue');
```

Vue.js版コード

HTML

```
<div id="vue">
  Hello {{ message }} !
  <button @click="update
</div>
```

③バインディングされているので
表示が「World」から「Vue.js」に
切り替わる

JavaScript

```
Vue.createApp({
  data() {
    return {
      message: 'World'
    };
  },
  methods: {
    update() {
      this.message = 'Vue.js'
    }
  }
}).mount('#vue');
```

②値が「World」から「Vue.js」に
変更される

①ボタンがクリックされるとmessage
の値を変更する

簡単な例だとレガシー(jQuery)の方が楽そう…

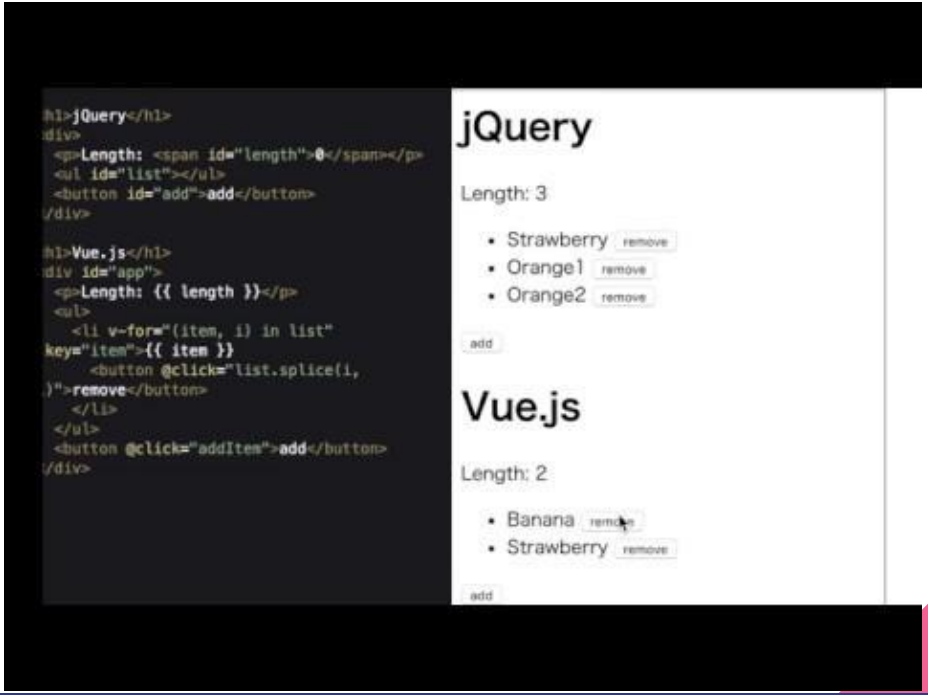
- コード量はそんなに変わらない（むしろVue.jsの方が少し多い）
- モダンなJavaScriptフレームワークのメリットは少ない

リスト要素の追加と削除

実際に動かしてみるなら
https://codepen.io/maru510/pen/mdrZyjP

コードはこちらの記事から引用させていただきました
(注:一部説明の為に改変してます)
https://qiita.com/mio3io/items/e7b2596d06b8005e8e6f

addボタンで要素の追加、removeボタンで要素を削除してみる。



jQuery版コード

HTML

```
<div id="jq">
  <p>Length: <span id="length">0</span></p>
  <ul id="list"></ul>
  <button id="add">add</button>
</div>
```

JavaScript

```
$(function() {
  let counter = 0
  let list = ['Apple', 'Banana', 'Strawberry']

  $('#add').on('click', function() {
    addItem('Orange' + (++counter).toString());
  });
  $('#jq').on('click', '.remove',
     function(event) {
    $(event.target).parent().remove();
    updateLength();
  });
  function init() {
    for (const i of list) {
      addItem(i);
    }
  }
  function addItem(name) {
    $('#list').append('<li>' + name
  + ' <button class="remove">remove</button></li>');
    updateLength();
  }
  function updateLength() {
    $('#length').text($('#list li').length);
  }
  init();
});
```

jQuery版コード

HTML

```html
<div id="jq">
  <p>Length: <span id="length">0</span></p>
  <ul id="list"></ul>
  <button id="add">add</button>
</div>
```

JavaScript

```javascript
$(function() {
  let counter = 0
  let list = ['Apple', 'Banana', 'Strawberry']

  $('#add').on('click', function() {
    addItem('Orange' + (++counter).toString());
  });
  $('#jq').on('click', '.remove',
     function(event) {
    $(event.target).parent().remove();
    updateLength();
  });
  function init() {
    for (const i of list) {
      addItem(i);
    }
  }
  function addItem(name) {
    $('#list').append('<li>' + name
  + ' <button class="remove">remove</button></li>');
    updateLength();
  }
  function updateLength() {
    $('#length').text($('#list li').length);
  }
  init();
});
```

コード量が多いので詳細は割愛するが
このコードの問題点は
ロジック側にビューが紛れ込んで
しまっている

Vue.js版コード

HTML

```
<div id="vue">
  <p>Length: {{ length }}</p>
  <ul>
    <li v-for="(item, index) in items">
      {{ item }}
      <button @click="removeItem(index)"
          >remove</button>
    </li>
  </ul>
  <button @click="addItem">add</button>

</div>
```

JavaScript

```
Vue.createApp({
  data() {
    return {
      counter: 0,
      items: ['Apple', 'Banana', 'Strawberry']
    };
  },
  computed: {
    length() {
      return this.items.length;
    }
  },
  methods: {
    addItem() {
      this.items.push('Orange' +
          (++this.counter).toString());
    },
    removeItem(index) {
      this.items.splice(index, 1);
    }
  }
}).mount('#vue');
```

Vue.js版コード

リストと配列を
バインディング

```
<div id="vue">
  <p>Length: {{ length }}</p>
  <ul>
    <li v-for="(item, index) in items">
      {{ item }}
      <button @click="removeItem(index)"
          >remove</button>
    </li>
  </ul>
  <button @click="addItem">add</button>

</div>
```

JavaScript

```
Vue.createApp({
  data() {
    return {
      counter: 0,
      items: ['Apple', 'Banana', 'Strawberry']
    };
  },
  computed: {
    length() {
      return this.items.length;
    }
  },
  methods: {
    addItem() {
      this.items.push('Orange' +
          (++this.counter).toString());
    },
    removeItem(index) {
      this.items.splice(index, 1);
    }
  }
}).mount('#vue');
```

Vue.js版コード

HTML

```
<div id="vue">
  <p>Length: {{ length }}</p>
  <ul>
    <li v-for="(item, index) in items">
      {{ item }}
      <button @click="removeItem(index)"
        >remove</button>
    </li>
  </ul>
  <button @click="addItem">add</button>
</div>
```

追加/削除ボタンと
各メソッドを
バインディング

JavaScript

```
Vue.createApp({
  data() {
    return {
      counter: 0,
      items: ['Apple', 'Banana', 'Strawberry']
    };
  },
  computed: {
    length() {
      return this.items.length;
    }
  },
  methods: {
    addItem() {
      this.items.push('Orange' +
          (++this.counter).toString());
    },
    removeItem(index) {
      this.items.splice(index, 1);
    }
  }
}).mount('#vue');
```

Vue.js版コード

```
<div id="vue">
  <p>Length: {{ length }}</p>
  <ul>
    <li v-for="(item, index) in items">
      {{ item }}
      <button @click="removeItem(index)"
          >remove</button>
    </li>
  </ul>
  <button @click="addItem">add</button>

</div>
```

③リストが配列の値に従って表示される
（つまり「Orange1」が追加される）

JavaScript

```
    data() {
      return {
        counter: 0,
        items: ['Apple', 'Banana', 'Strawberry']
      };
    },
    computed: {
      length() {
        return this.items.le
      }
    },
    methods: {
      addItem() {
        this.items.push('Orange' +
            (++this.counter).toString());
      },
      removeItem(index) {
        this.items.splice(index, 1);
      }
    }
}).mount('#vue');
```

②配列の最後に「Orange1」
が追加される

①追加ボタンが押されたら
配列の要素を追加

Vue.js版コード

③リストが配列の値に従って表示される
（つまり自分の要素が削除される）

```
<div id="vue">
  <p>Length: {{ length }}</p>
  <ul>
    <li v-for="(item, index) in items">
      {{ item }}
      <button @click="removeItem(index)"
          >remove</button>
    </li>
  </ul>
  <button @click="addItem">add</button>

</div>
```

JavaScript

```
    return {
      counter: 0,
      items: ['Apple', 'Banana', 'Strawberry']
    };
  },
  computed: {
    length() {
      return this.items.le
    }
  },
  methods: {
    addItem() {
      this.items.push('Orange' +
          (++this.counter).toString());
    },
    removeItem(index) {
      this.items.splice(index, 1);
    }
  }
}).mount('#vue');
```

②配列から自分の要素
が消える

①削除ボタンが押されたら
配列の自分の要素を削除

複雑になってくるとモダン(Vue.js)の方が楽！

- コード量もVue.jsの方が気持ち少なくてすむ
- DOMで直接操作するよりオブジェクトで操作するので楽！
- レガシー(jQuery)だとロジックにビュー要素が混入しがちだが
  モダン(Vue.js)だとロジックとビューは綺麗に分離できる

結論

まとめでーす
ヾ(=^▽^=)ノ

モダンなJavaScriptフレームワークはレガシーなフレームワークに比べて

ココがいい！

- 見た目をオブジェクトで操作できる
  - 複雑なUIになるほど効果絶大！
    DOMで直接操作するよりオブジェクトを操作した方が実装が楽！
- ビューとロジックを分離できる
  - レガシーだとロジックにビューが紛れ込む。
    モダンだと見た目と処理で責務分解ができてコードの可読性が上がる！

ぜひ(^_^)/

モダンなJavaScriptフレームワークを使いましょう！

終

制作・著作

SWB

（おまけ）SPAについて

SPA(Single Page Application)とは

- Webアプリケーションの形式の1つで、「1つのWebページによって、1つのアプリを構築する」というもの

- 従来型のアプリケーションは、ユーザのアクションに応じてサーバーからHTMLを受け取りそのたびにヘッダやフッダを含めて全て更新する方式をとっていた
- SPAでは、最初にHTMLやCSS、JavaScriptを受け取った後は必要なデータだけをサーバに要求し差分を表示する方式で構築される

SPAのメリデメ

- メリット
  - ユーザにリッチなUIを提供できる
  - 高速な表示更新が可能になる
  - ネイティブアプリのようなアプリを作れる
- デメリット
  - 最初のローディングに時間がかかる
  - 開発にコストがかかる
  - 開発者が少ない

代表的なSPAフレームワーク

- Angular（2016/09〜）
  - Google中心のコミュニティで開発
  - AngularJS（2009〜）の後継
  - TypeScriptで記述
- React（2013/07〜）
  - Facebook中心のコミュニティで開発
  - JSXで記述
- Vue.js （2013/12〜）
  - JavaScript/TypeScriptで記述

Vue.jsについてもう少し

The Progressive
JavaScript Framework

公式サイトの引用

親しみやすい

すでに HTML、CSS そして JavaScript を知っていますか？
ガイドを読んで、すぐにモノ作りを開始しましょう！

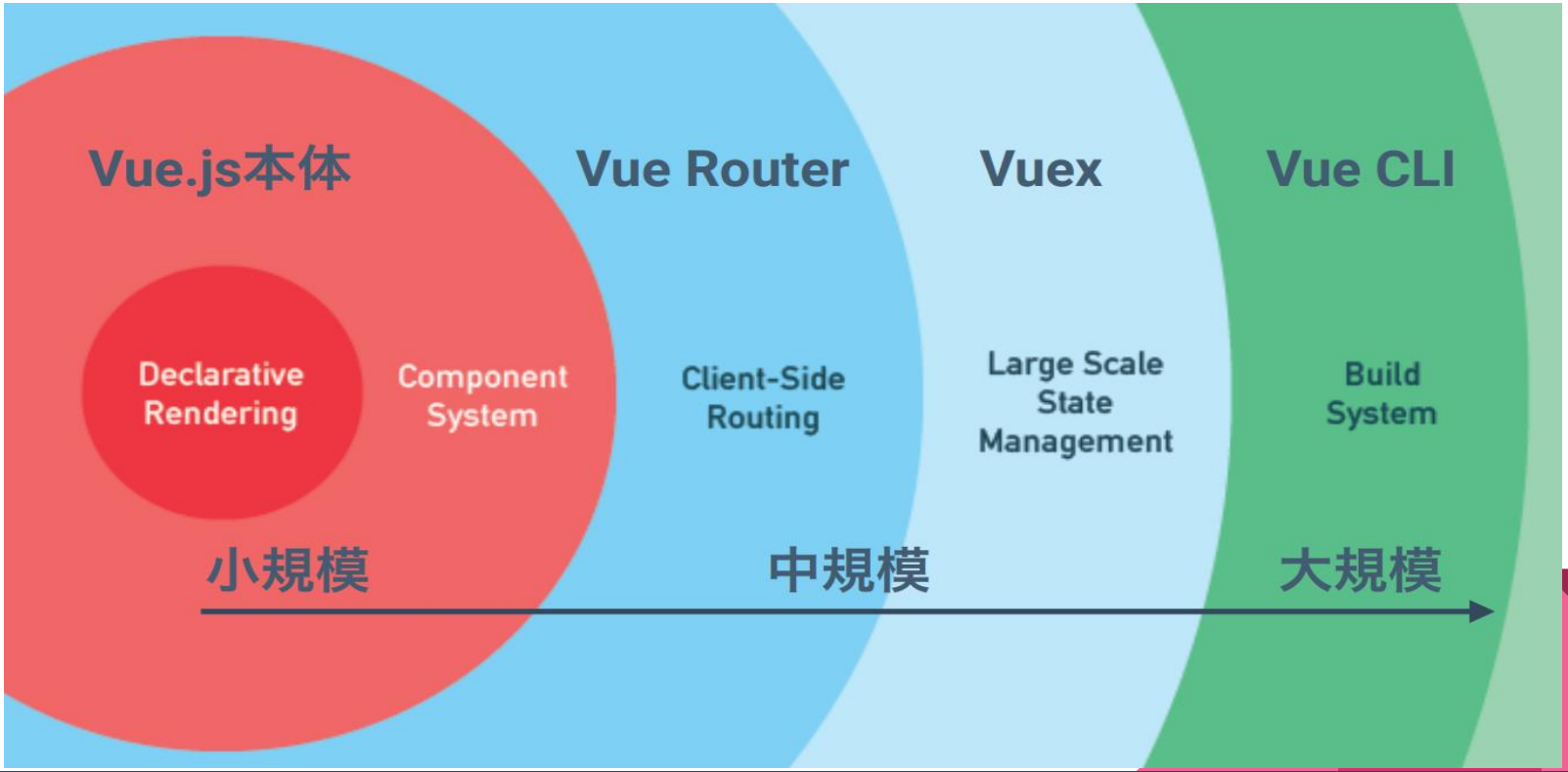
融通が効く

ライブラリと完全な機能を備えたフレームワークの間で
拡張できる徐々に採用可能なエコシステム

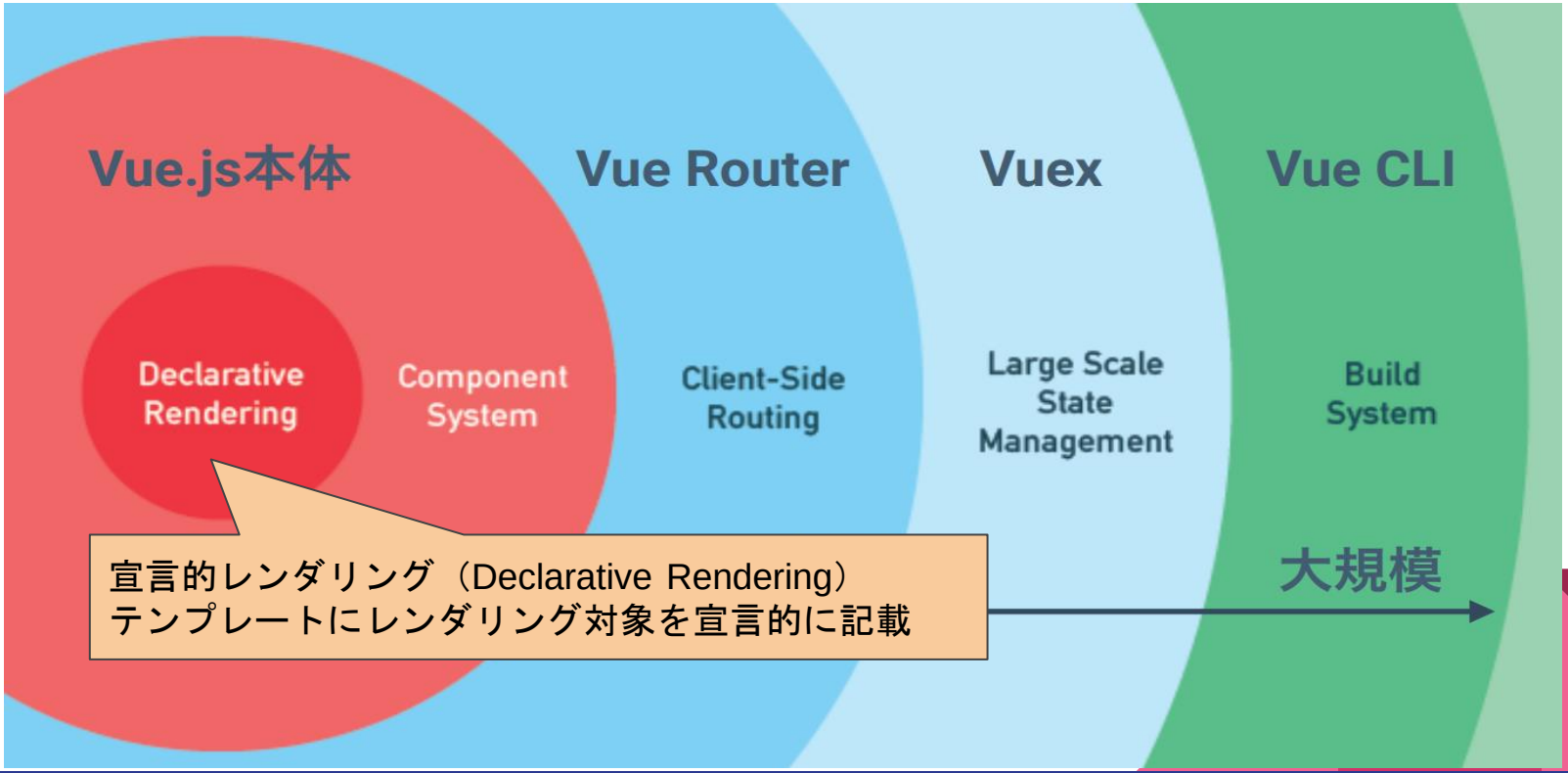
高性能

20KB min+gzip ランタイム
猛烈に速い Virtual DOM
最小限の努力で最適化が可能

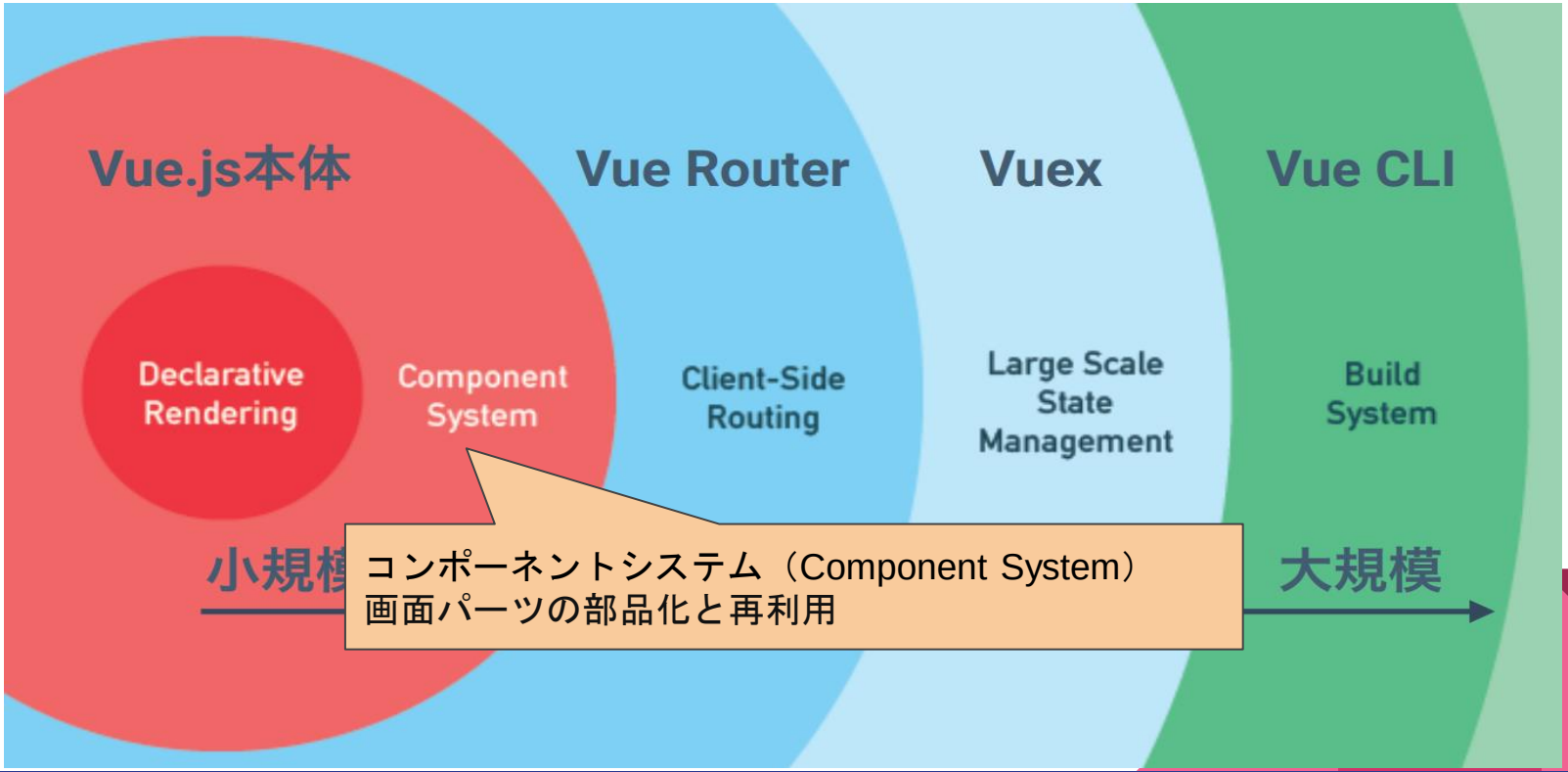
Progressive Frameworkとは？



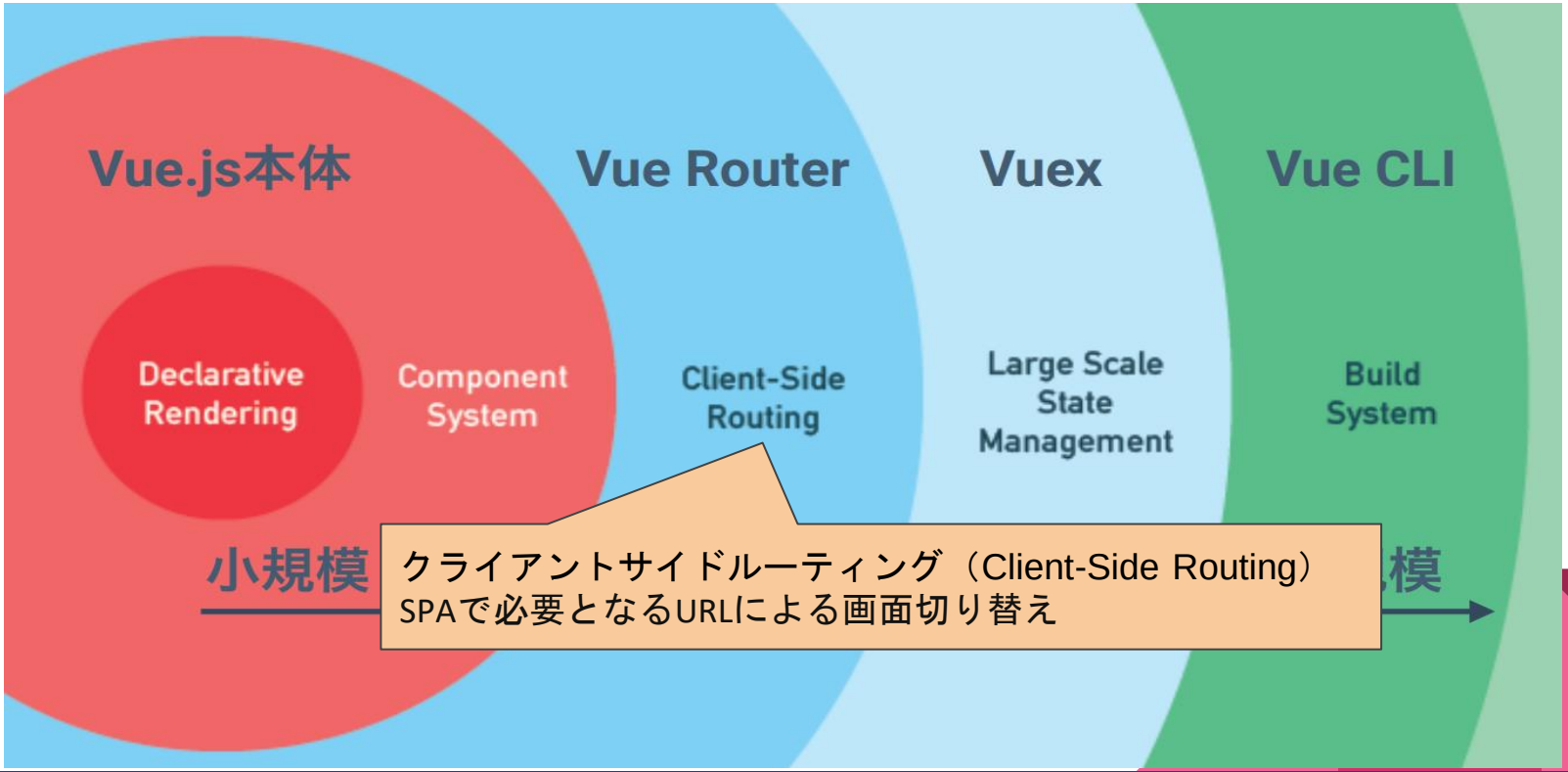
Vue.jsの適用段階


Vue.js本体
Vue Router
Vuex
Vue CLI
Declarative Rendering
Component System
Client-Side Routing
Large Scale State Management
Build System
宣言的レンダリング（Declarative Rendering）
テンプレートにレンダリング対象を宣言的に記載
大規模

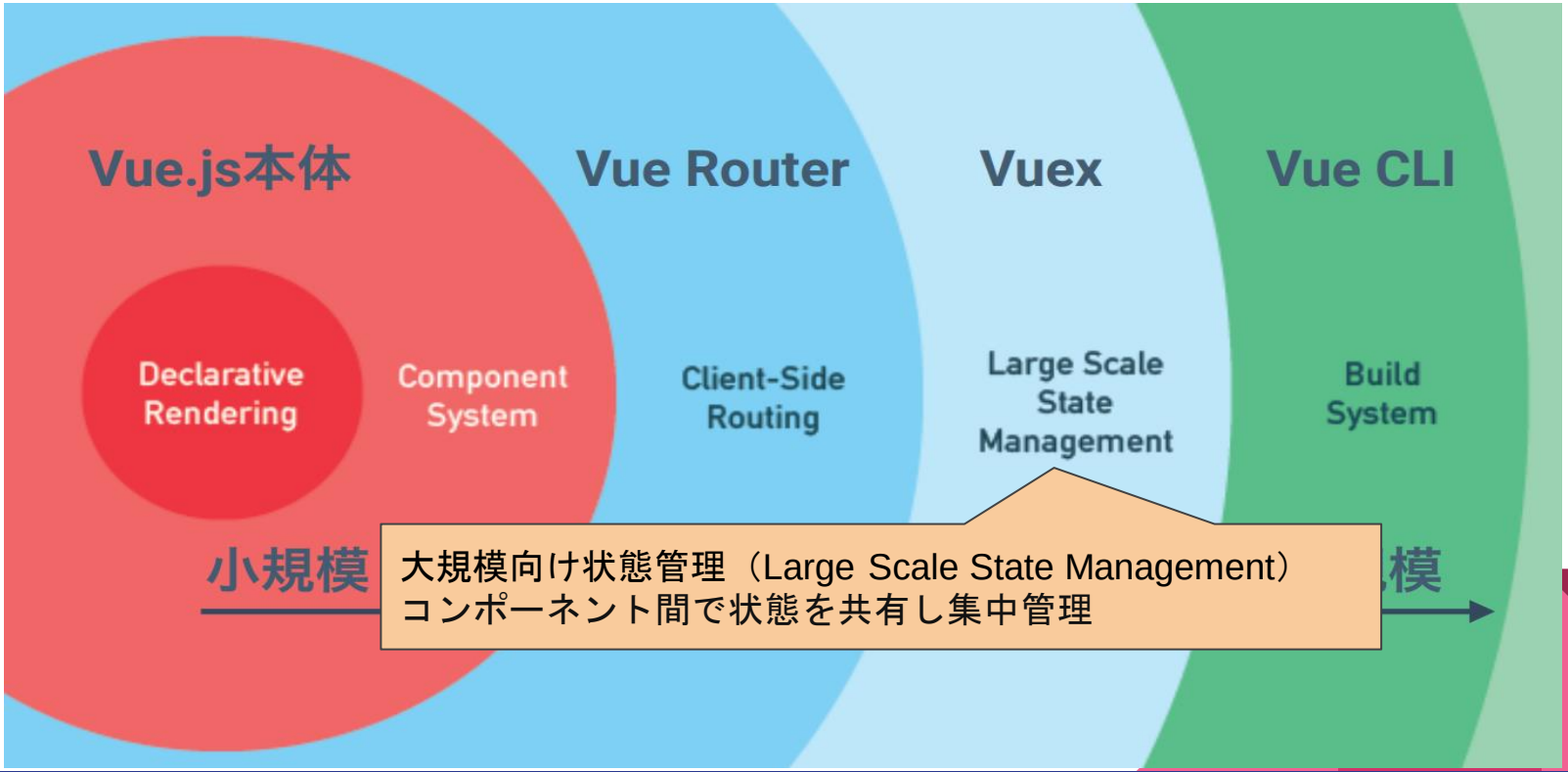
Vue.jsの適用段階


Vue.js本体
Vue Router
Vuex
Vue CLI
Declarative Rendering
Component System
Client-Side Routing
Large Scale State Management
Build System
小規模
大規模
コンポーネントシステム（Component System）
画面パーツの部品化と再利用

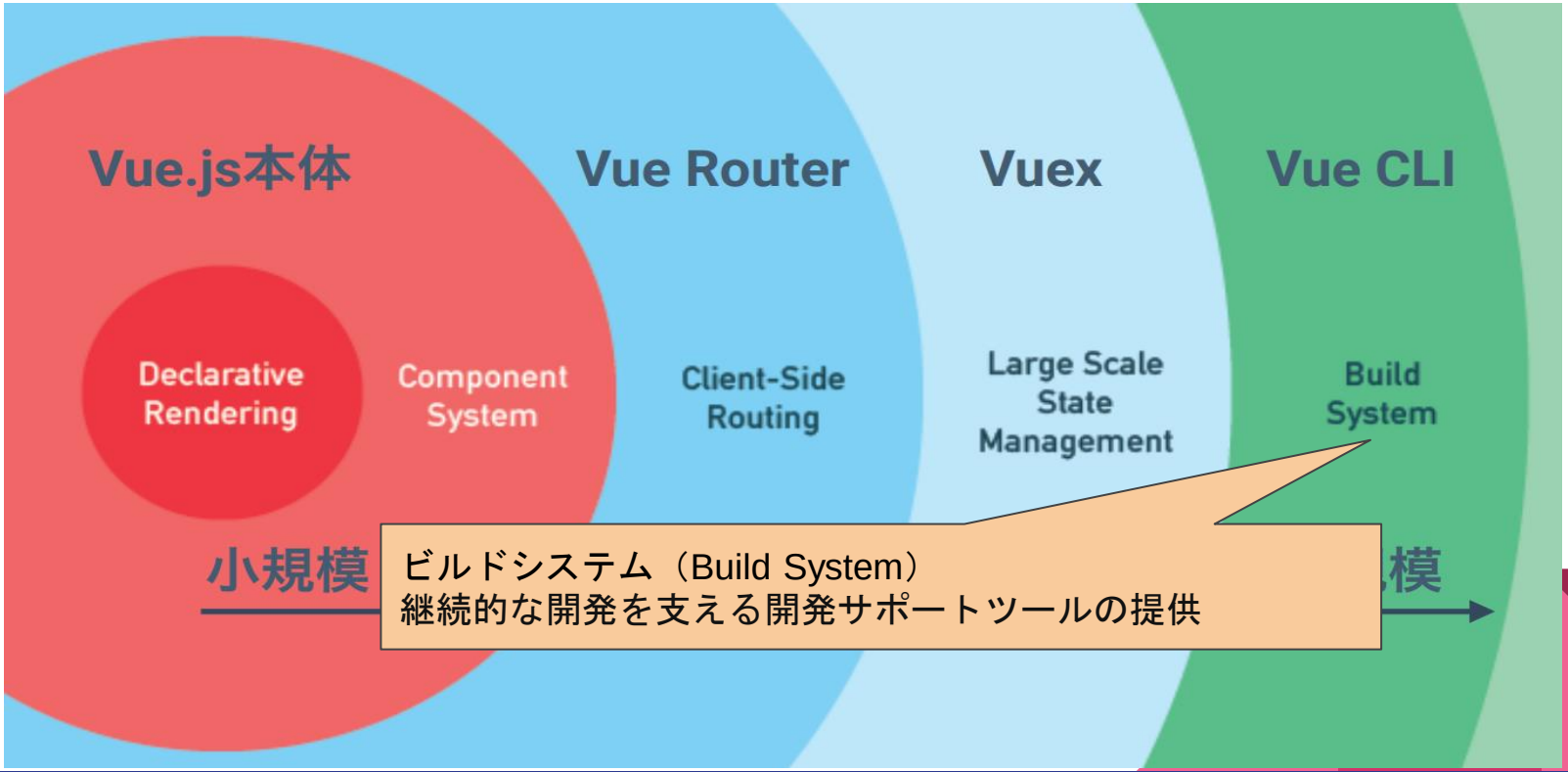
Vue.jsの適用段階


Vue.js本体
Vue Router
Vuex
Vue CLI
Declarative Rendering
Component System
Client-Side Routing
Large Scale State Management
Build System
小規模
クライアントサイドルーティング（Client-Side Routing）
SPAで必要となるURLによる画面切り替え

Vue.jsの適用段階


Vue.js本体
Vue Router
Vuex
Vue CLI
Declarative Rendering
Component System
Client-Side Routing
Large Scale State Management
Build System
小規模
大規模向け状態管理（Large Scale State Management）
コンポーネント間で状態を共有し集中管理

Vue.jsの適用段階


Vue.js本体
Vue Router
Vuex
Vue CLI
Declarative Rendering
Component System
Client-Side Routing
Large Scale State Management
Build System
小規模
ビルドシステム（Build System）
継続的な開発を支える開発サポートツールの提供

ぜひ(^_^)/

次の案件はSPAで開発しましょう！

終