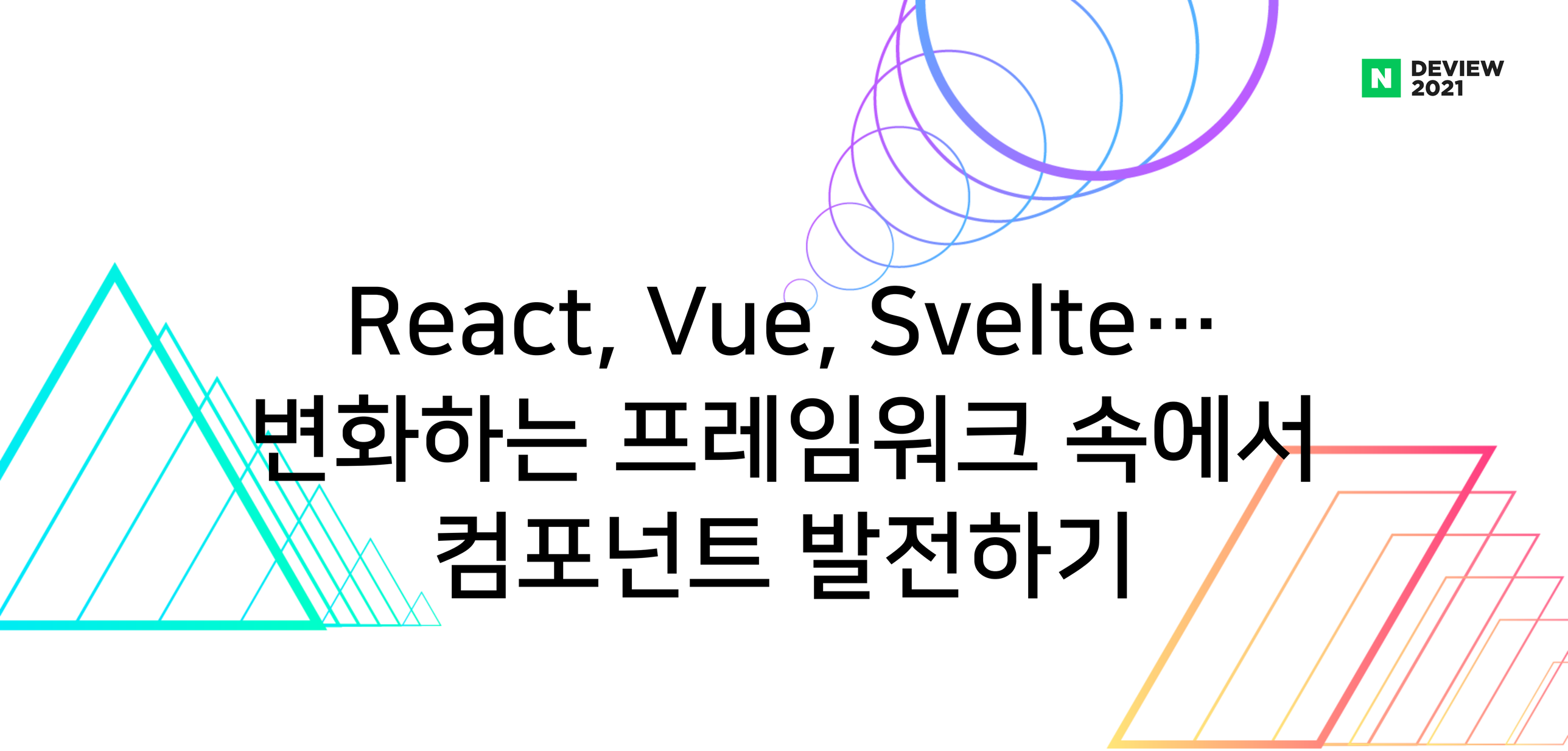


React, Vue, Svelte…
변화하는 프레임워크 속에서
컴포넌트 발전하기

전용우 NAVER

CONTENTS

1.우리가 해결하고 싶었던 문제

1.1 다양한 개발 환경

DEVIEW 2021

jQuery부터 React, Vue, Angular… 까지 다양한 개발 환경

- jQuery에서 프레임워크로 전환
- 기능은 그대로인 상태에서 개발 환경 전환
- 기존과 같은 기능을 하는 컴포넌트는 프레임워크에서 없음
- 프레임워크에서 지원하는 컴포넌트에 맞게 기능을 개발

1.2 같은 UI, 다른 프레임워크

프레임워크의 선정에 따라 지원하는 UI 및 기능이 변경되는 문제

- Vanilla JS로 작성한 컴포넌트를 각 프레임워크에서 사용할 수 있도록 Wrapping
- Vanilla JS의 개발 방법과 프레임워크 기반의 개발 방법은 매우 다름
- 프레임워크 기반의 개발은 DOM을 개발자가 핸들링하지 않음
- 직접 DOM을 핸들링하는 기존의 Component 개발은 적합하지 않음

1.3 DOM 동기화

DEVIEW 2021

프레임워크에서의 개발 방법을 컴포넌트에 적용

- DOM을 직접 핸들링하는 Component는 프레임워크 기반의 개발에 적합하지 않음
- 프레임워크에 DOM 핸들링을 위임하고 컴포넌트는 변경된 DOM을 동기화
- 컴포넌트는 DOM 조작을 추상화하여 Vanilla JS에서는 직접, 프레임워크는 위임

1.4 Cross Framework Component

DEVIEW 2021

유사한 프레임워크 패턴에서 쉽게 적용 가능

- 대부분 프레임워크는 유사한 Life Cycle을 가짐
- DOM에 접근 가능한 이벤트에 Vanilla JS을 초기화
- 변화는 감지할 때 DOM을 비교(diff)하여 컴포넌트에 반영
- 이로 인해 새로운 프레임워크에 대응이 빠름

2. 새로운 고민들

DOM Diff

2.1.1 가장 많이 호출 하는 기능

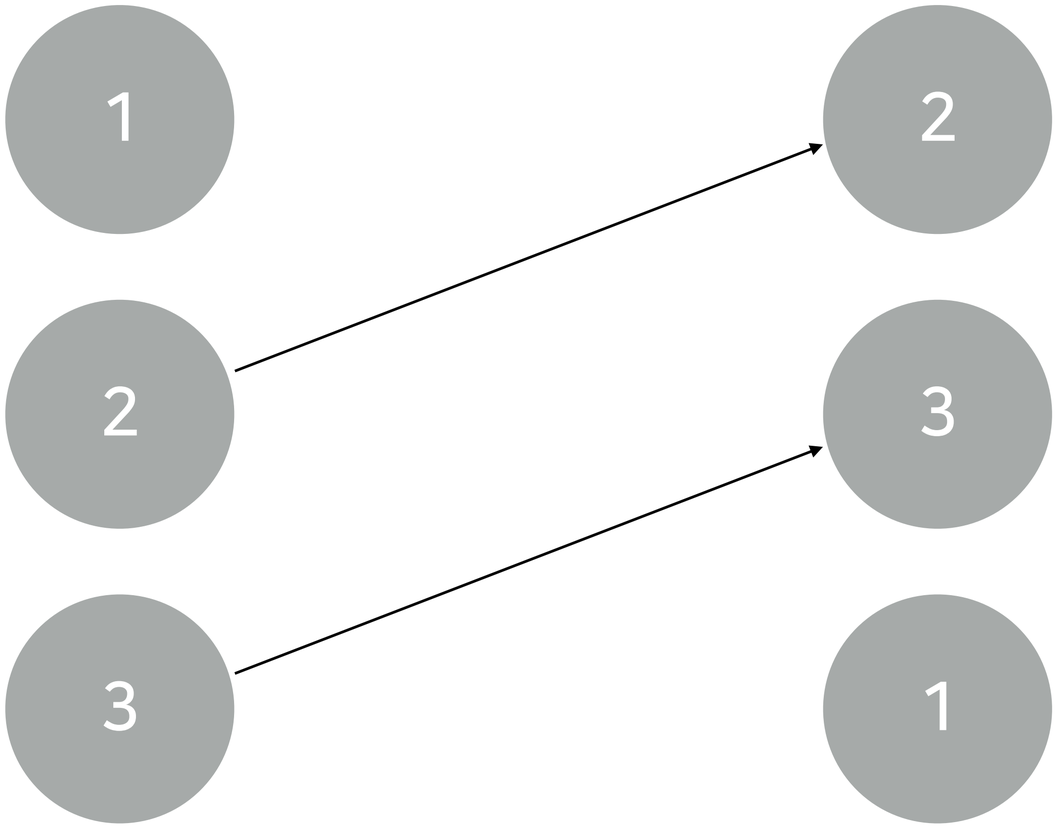
DEVIEW 2021

CFC에서는 DOM Diff 과정이 매우 많이 발생함

- 프레임워크에서 변화를 Vanilla JS에서 동기화 하는 과정이 필요
- 동기화 되는 과정은 매우 자주 발생하는 이벤트
- 이 부분을 개선하면 전반적인 개선이 가능

2.1.2 각 요소에 따른 변화

CFC에서는 DOM의 변화를 감지하여 반영하는게 중요



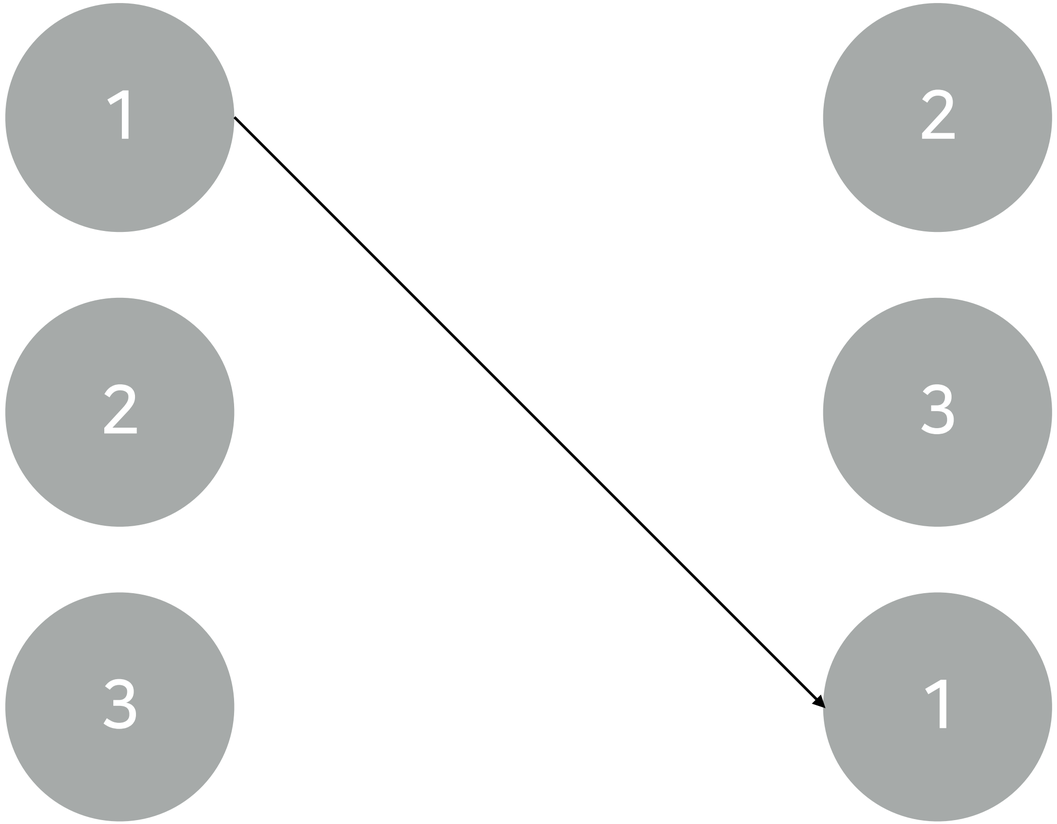
1번 인덱스를 0번으로
2번 인덱스를 1번으로

2번의 이동이 필요

2.1.2 최소의 변화로 동기화

DEVIEW 2021

하지만, 한번만 움직이면 해결이 가능



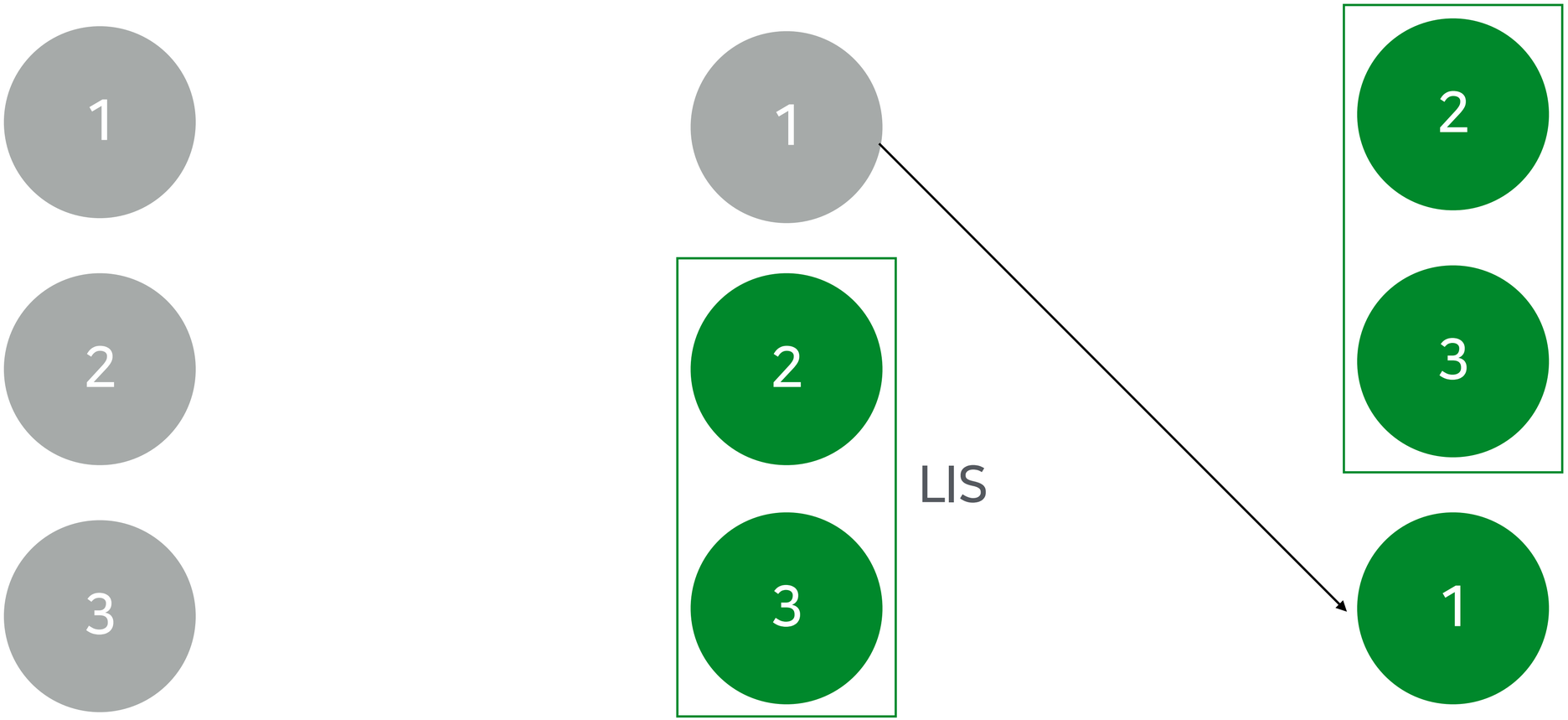
0번 인덱스를 2번으로

1번의 이동이 필요

2.1.3 Longest Increasing Subsequence

우린 가장 적게 움직이는 방법을 찾아야 함.

- 주어진 수열에서 오름차순으로 정렬된 가장 긴 부분 수열을 찾음 = LIS
- 정렬된 부분 수열을 고정 후 고정되지 않은 원소를 대상으로 정렬



2.1.4 개선 결과

기존 대비 성능 변화

- diff 시간, DOM 핸들링 시간. 10,000회 반복 평균

	100개 요소의 랜덤 변경	100개 요소 중 1개 변경
diff 시간	1% 지연 0.083ms -> 0.084ms	33% 개선 0.044ms -> 0.033ms

	16개 DOM의 랜덤 변경	16개 DOM 중 1개 변경
DOM 조작 시간	13% 개선 1.43ms -> 1.24ms	67% 개선 0.33ms -> 0.11ms

Server-Side Rendering

2.2.1 Server-Side Rendering

Vanilla JS에서는 Rendering을 고민하지 않았지만 프레임워크는 다름

- Vanilla JS는 서버에서 렌더링(Java, C, C++…)이 완료된 DOM을 가지고 핸들링
- 프레임워크에서는 Client와 Server(node.js)가 같은 코드를 렌더링
- SSR에서 DOM 접근 관련해서 이슈
- 이 부분이 프레임워크마다 다르기 때문에 개발이 필요

2.2.2 React / Vue

두 Framework는 Server와 Client의 Life Cycle이 다름

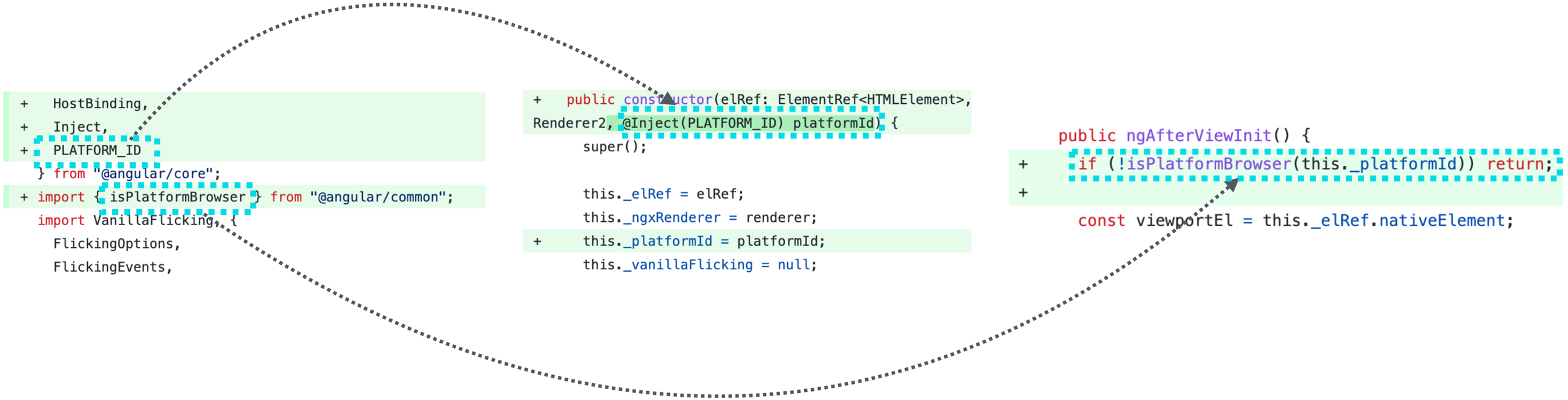
- DOM에 접근할 수 있는 라이프 사이클에서 컴포넌트가 초기화
  - React / Preact : “componentDidMount” -> DOM에 접근 가능
  - Vue 2/3 : “mounted” -> DOM에 접근 가능
- 두 프레임워크는 DOM에 접근하는 라이프 사이클이 서버에서 실행하지 않음
- 가장 편한 방법으로 쉽게 해결 가능

2.2.3 Angular

DEVIEW 2021

Angular에서는 Angular Universal을 이용하여 SSR 구현

- Angular는 DOM에 접근 가능한 라이프 사이클이 SSR에서 실행
- DOM에 접근 가능한 라이프 사이클인 “ngAfterViewInit” 에 예외 처리가 필요
- 필요한 기능은 Angular에서 제공



2.2.4 Svelte

DEVIEW 2021

Svelte는 Life Cycle도 다르고, SSR과 CSR는 원본 파일이 필요

- 유사하게 onMount은 server에서 호출되지 않지만, onDestroy가 호출
- Client와 Server가 각각 빌드해야 하기 때문에 원본 파일이 필요
- JS만 가능하기 때문에 TypeScript을 사용하지 못함

```
128       onDestroy(() => {
129   -     grid.destroy();
130       });
```

```
125       onDestroy(() => {
126   +     vanillaGrid && vanillaGrid.destroy();
127       });
```

```
    "sideEffects": false,
+   "svelte": "src/index.js",
    "types": "src/index.d.ts",
```

package.json

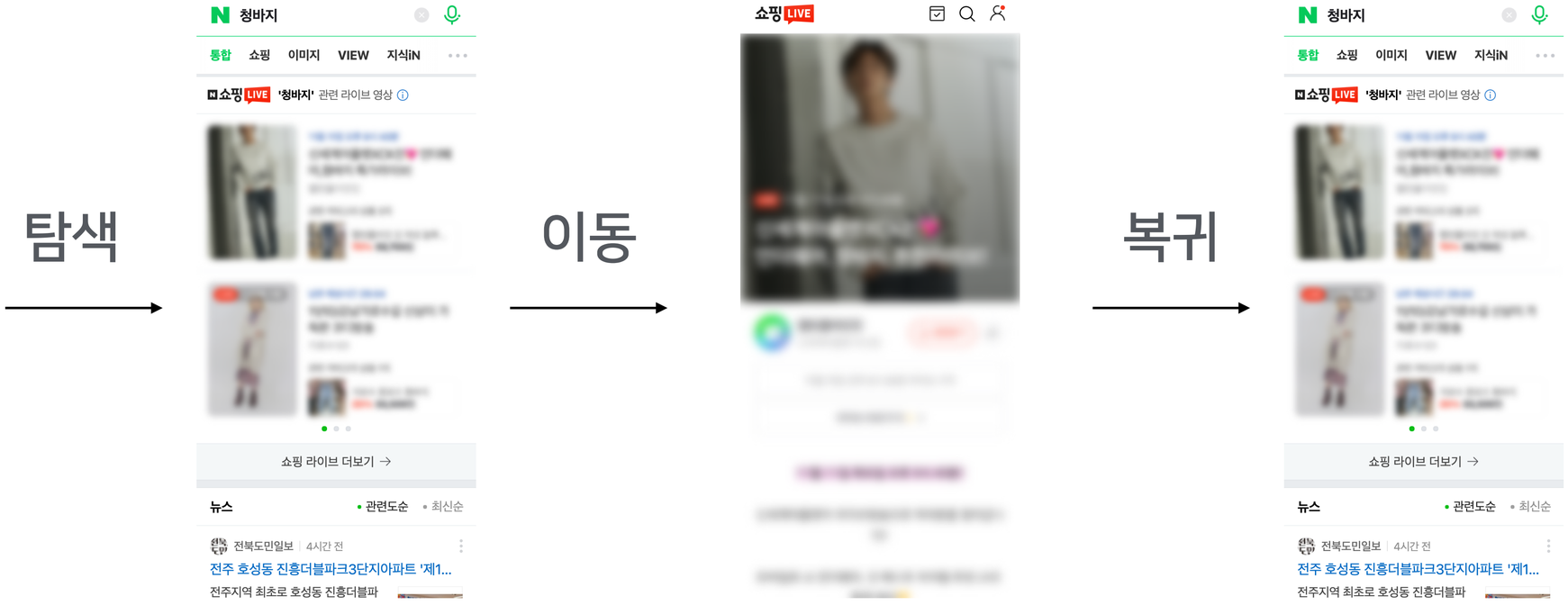
State Restore

2.3.1 State Restore

DEVIEW 2021

상태를 복구하는 기능은 UX 적으로 중요한 기능

- 현재 대표적인 UI 구성은 요약된 여러 개의 결과를 탐색한 후 상세보기로 이동
- 뒤로 돌아왔을 때 다시 사용자는 전에 위치를 복구하기를 기대함
- Vanilla JS에서는 BFCache와 History.scrollRestoration이 합쳐지면 해결



2.3.2 Vanilla JS State Restore

DEVIEW 2021

상태를 복구하기 필요한 정보는 Component의 상태 정보 + HTML

- Component가 현재 상태를 알기 위해 필요한 각종 index 등 정보가 필요
- 복구할 시점의 HTML
- 페이지를 떠날 때 getStatus로 restore 정보를 얻어 Persist(로컬 스토리지) 저장
- 페이지에 돌아와서 Persist에 정보를 찾아 setStatus로 복구

```
const persist = new Persist("search-persist");
const vanillaMasonryInfiniteGrid = new MasonryInfiniteGrid({...});
persist.set("infinitegrid",vanillaMasonryInfiniteGrid.getStatus());
```

→ 각종 Status와 HTML 저장

```
const persist = new Persist("search-persist");
const vanillaMasonryInfiniteGrid = new MasonryInfiniteGrid({...});
vanillaMasonryInfiniteGrid.setStatus(persist.get("infinitegrid"));
```

→ 저장된 정보(+HTML)를 복구

2.3.3 Framework State Restore

Component의 상태 정보는 문제없지만 DOM은 문제가 생김

- CFC에서 발생했던 컴포넌트와 프레임워크 간의 DOM 동기화 문제 발생
- Framework에서는 HTML이 아닌 구성하기 위한 데이터를 Persist에 저장
- HTML = 구성하기 위한 정보(key등 각종 정보) + 데이터
- Persist에서 DOM을 구성하기 위한 데이터를 기반으로 Framework에서 렌더링

```
const persist = new Persist("search-persist");
const infiniteRef = React.useRef();
persist.set("infinitegrid",{
 "status":infiniteRef.current.getStatus(),
 "data":store.getState()
});
<MasonryInfiniteGrid ref={infiniteRef}>...</MasonryInfiniteGrid>
```

→ 각종 Status와 Data를 저장

```
const persist = new Persist("search-persist");
const {status, data} = persist.get("infinitegrid");
const items = status.groupManager.groups[0].items;

<MasonryInfiniteGrid status={status}>
  {items.map((item,i) => {
    return  <Item data-grid-groupkey={item.groupKey} key={item.key}>
               <img src={data[i].src}>
            </Item>
  }}
</MasonryInfiniteGrid>
```

→ 복구는 직접

3. 다른 타입의 CFC

3.1 확장된 Cross Framework Component

DEVIEW 2021

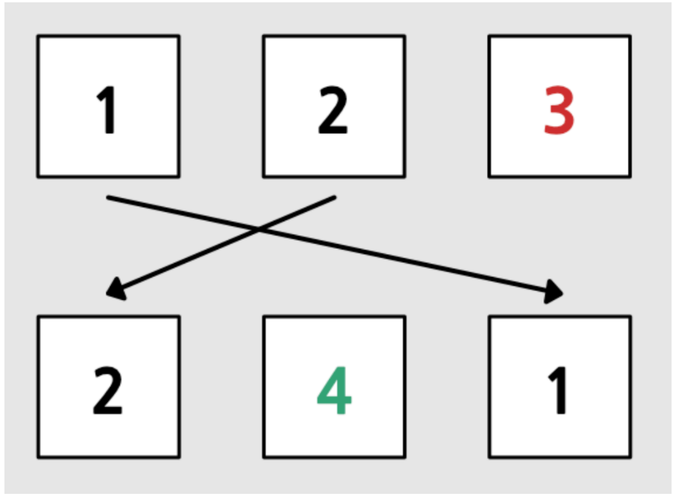
DOM 동기화가 필요한 컴포넌트 이외에 2가지 추가 타입

- Dynamic DOM : 컴포넌트 DOM정보와 프레임워크의 DOM정보를 동기화 해야 경우
- Reactive : 상태의 변화만 감지하는 경우
- Static DOM : 컴포넌트에서 초기 이외에 DOM정보를 동기화 하지 않아도 되는 경우

3.2 Dynamic DOM

컴포넌트에서 DOM을 삭제/수정/추가 등 변경이 있다면 DOM diff

- Flicking/InfiniteGrid와 같이 아이템을 추가하고 삭제하는 컴포넌트
- DOM을 핸들링 하는 컴포넌트는 프레임워크용 컴포넌트를 만들 때 컴포넌트에서 DOM을 핸들링하지 않고, 프레임워크로 위임 후 라이프 사이클에서 동기화



Reactive[new]

3.3.1 상태의 변화만 감지

DEVIEW 2021

컴포넌트가 DOM을 핸들링 하지 않고 상태의 변화만 있는 경우

- ImReady와 같이 DOM을 직접 핸들링하지 않고
- 이미지 등 리소스가 로딩이 됐는지 상태의 변화만 알림
- 이런 경우라면 일반 컴포넌트 방식 보다 프레임워크에서 제공하는 적절한 기능을 이용

개발 흐름

- mount 시점에 초기화, 이벤트에 state 연동 -> 이벤트에서 state 변경 -> 변경 전파
- 프레임워크마다 mount 시점 찾기, state 사용 방법, 변경 전파 적용

3.3.2 React Hooks

DEVIEW 2021

React에서는 상태의 변화를 처리하기 위한 Hooks가 있음

- mount 시점 : useEffect
- state / 전파 : useState

```
const children: HTMLElement[] = [];
const [preReadyCount, setPreReadyCount] = useState(0);
const [readyCount, setReadyCount] = useState(0);
const [errorCount, setErrorCount] = useState(0);
const [totalErrorCount, setTotalErrorCount] = useState(0);
const [totalCount, setTotalCount] = useState(0);
const [isPreReady, setIsPreReady] = useState(false);
const [isReady, setIsReady] = useState(false);
const [hasError, setHasError] = useState(false);
const [isPreReadyOver, setIsPreReadyOver] = useState(false);
```

```
useEffect(() => {
    const im = new ImReady(options);
    im.check(checkedElements).on("error", e => {
        if (useError) {
            setHasError(true);
            setErrorCount(e.errorCount);
            setTotalErrorCount(e.totalErrorCount);
        }
    }).on("preReadyElement", e => {
        if (usePreReadyElement) {
            setPreReadyCount(e.preReadyCount);
        }
    }).on("readyElement", e => {
        if (useReadyElement) {
            setReadyCount(e.readyCount);
            setIsPreReadyOver(e.isPreReadyOver);
        }
    }).on("preReady", () => {
        if (usePreReady) {
            setIsPreReady(true);
        }
    }).on("ready", () => {
        if (useReady) {
            setIsReady(true);
        }
    });
    return () => {
        im.destroy();
    };
}, []);
```

```
function App() {
  const im = useImReady({
    selector: "img",
  });
  const { register, readyCount, totalCount, errorCount, isReady } = im;
  return (
    <div className="App">
      <header className="App-header">
        <p>isReady: <span id="isReady">{isReady ? "I'm Ready" : "Loading.
        <p>progress: <span id="readyCount">{readyCount}</span> / <span id
        <p>errors: <span id="readyCount">{errorCount}</span></p>
      </header>
    </div>
  );
}
```

3.3.3 Vue2 Composition API

DEVIEW 2021

Vue2에서는 상태의 변화를 처리하기 위한 Composition API가 있음

- mount 시점 : onMounted
- state / 전파 : reactive

```
import { onBeforeUnmount, onMounted, reactive, Ref, ref } fro
const state = reactive({
    preReadyCount: 0,
    readyCount: 0,
    errorCount: 0,
    totalErrorCount: 0,
    totalCount: 0,
    isPreReady: false,
    isReady: false,
    isPreReadyOver: false,
    hasError: false
});
```

```
onMounted(() => {
  const im = new ImReady(options);
  im.check(checkedElements)
    .on("error", e => {
      if (useError) {
        state.hasError = true;
        state.errorCount = e.errorCount;
        state.totalErrorCount = e.totalErrorCount;
      }
    })
    .on("preReadyElement", e => {
      if (usePreReadyElement) {
        state.preReadyCount = e.preReadyCount;
      }
    })
    .on("readyElement", e => {
      if (useReadyElement) {
        state.readyCount = e.readyCount;
        state.isPreReadyOver = e.isPreReadyOver;
      }
    })
    .on("ready", () => {
      if (useReady) {
        state.isReady = true;
      }
    });
```

```
<template>
<div class="page" id="home" v-bind:ref="im.re
   <h2>{{im.isReady ? "I`m Ready" : "Loading"
</div>
<script>
export default {
  setup() {
    const im = useImReady({
      selector: "img"
    });
    return {
      im,
    };
  },
};
</script>
```

3.3.4 Vue3 Reactive

Vue3에서는 상태의 변화를 처리하기 위한 Reactive가 있음

- mount 시점 : onMounted
- state / 전파 : reactive

```
const state = reactive({
  preReadyCount: 0,
  readyCount: 0,
  errorCount: 0,
  totalErrorCount: 0,
  totalCount: 0,
  isPreReady: false,
  isReady: false,
  isPreReadyOver: false,
  hasError: false
});
```

```
onMounted(() => {
  const im = new ImReady(options);
  im.check(checkedElements)
    .on("error", e => {
      if (useError) {
        state.hasError = true;
        state.errorCount = e.errorCount;
        state.totalErrorCount = e.totalErrorCount;
      }
    })
    .on("preReadyElement", e => {
      if (usePreReadyElement) {
        state.preReadyCount = e.preReadyCount;
      }
    })
    .on("readyElement", e => {
      if (useReadyElement) {
        state.readyCount = e.readyCount;
        state.isPreReadyOver = e.isPreReadyOver;
      }
    })
    .on("ready", () => {
      if (useReady) {
        state.isReady = true;
      }
    });
```

```
<template>
    <div v-bind:ref="im.register()">
        <h2>{{im.isReady ? "I'm Ready" : "Loadin
        <h2>{{im.readyCount}}/{{im.totalCount}}<
        <h2>{{im.isPreReady ? "I'm PreReady" : "
        <h2>{{im.preReadyCount}}/{{im.totalCount
        <img src="..." />
        <img src="..." />
        <img src="..." />
    </div>
</template>
<script>
import { useImReady } from "@egjs/vue-imready";

export default {
    setup() {
        const im = useImReady({
            selector: "img",
        });
        return {
            im,
        }
    }
}
</script>
```

3.3.5 Angular Directive

Angular는 유사한 기능은 없지만, directive를 이용하여 구현

- mount 시점 : ngAfterViewInit
- state / 전파 : EventEmitter

```
@Directive({
  selector: '[NgxImReady]',
})
```

```
ngAfterViewInit(): void {

  const im = new ImReady(options);

  EVENTS.forEach((name) => {
    im.on(name, e => {
      this[name].emit(e as any);
    });
  });

  im.check(checkedElements);
}
```

```
export class AppComponent {
  title = 'ngx-imready';
  public isReady = false;
  onReady(e) {
    this.isReady = e.isReady;
  }
}
```

```
<div class="App" NgxImReady selector="img" (ready)="onReady($e
  <header class="App-header">
    <p>isReady: <span id="isReady">{{isReady ? "I'm Ready" : "
  </header>
</div>
```

3.3.6 Svelte Store

DEVIEW 2021

Svelte의 Store을 이용하여 React의 Hooks와 유사하게 구현

- mount 시점 : onMount
- state / 전파 : Store.writeable

```
const preReadyCount = writable(0);
const readyCount = writable(0);
const errorCount = writable(0);
const totalErrorCount = writable(0);
const totalCount = writable(0);
const isPreReady = writable(false);
const isReady = writable(false);
const hasError = writable(false);
const isPreReadyOver = writable(false);
```

```
im.check(checkedElements).on("error", e => {
  if (useError) {
    hasError.set(true);
    errorCount.set(e.errorCount);
    totalErrorCount.set(e.totalErrorCount);
  }
}).on("preReadyElement", e => {
  if (usePreReadyElement) {
    preReadyCount.set(e.preReadyCount);
  }
}).on("readyElement", e => {
  if (useReadyElement) {
    readyCount.set(e.readyCount);
    isPreReady.set(e.isPreReadyOver);
  }
```

```
  const {
    register,
    readyCount,
    totalCount,
    errorCount,
    isReady,
  } = useImReady({
    selector: "img",
  });
</script>

<style>

</style>
    <p>isReady: <span id="isReady">{$isReady ? "I'm Ready
    <p>progress: <span id="readyCount">{$readyCount}</spa
    <p>errors: <span id="readyCount">{$errorCount}</span>
  <div class="images" use:register>
    <img src="https://naver.github.io/egjs-infinitegrid/a
  </div>
```

Static DOM[new]

3.4.1 초기 로딩 후 DOM의 동기화 없음

DEVIEW 2021

한번 로딩되면 컴포넌트 내부에서 모두 처리되는 컴포넌트

- View360, View3D와 같이 로딩 이후 DOM의 동기화가 필요 없는 컴포넌트
- 이런 컴포넌트는 각 프레임워크 쉽게 사용할 수 있도록 살짝 감싸는 게 좋음
- 옵션은 Prop으로 처리

개발 흐름

- Dynamic DOM - DOM Sync = Static DOM
- 프레임워크마다 mount 시점 찾기, 이벤트 연결, instance 메서드 연결

4. 새로 생기고 발전하는 프레임워크

4.1 일타 쌍피 Preact

Preact는 사실 React와 다르지 않음

- Preact 컴포넌트는 React 컴포넌트를 사용
- preact-compat을 사용 가능
- rollup-plugin-preact으로 변환 가능

```
const preact = require("rollup-plugin-preact");

const defaultOptions = {
    sourcemap: false,
    tsconfig: "tsconfig.build.json",
    external: {
      "preact": "preact",
      "preact/compat": "preact/compat"
    },
    plugins: [
      preact({
        extensions: ["js", "jsx", "ts", "tsx"],
        noPropTypes: false,
        noEnv: false,
        noReactIs: false,
        resolvePreactCompat: true,
        usePreactX: true,
        aliasModules: {
          'react-dom': 'preact/compat',
          'react-is': 'preact/compat'
        }
      })
    ]
};
```

Vue2와 Vue3를 같이 적용할 수 있나?

컴포넌트 개발 관점

4.2.1 같이 적용하기 힘든 이유

DEVIEW 2021

같이 사용할 수 있을 거라 생각하지만 많은 부분이 다름

- Global API의 이름이 변경됨
- Template 문법을 사용하면 호환되지 않음

```
// Vue 2
import Vue from "vue";

Vue.aaa

// Vue 3
import { bbb } from "vue";

bbb
```

```
<template>
<div></div>
</template>
```

```
// Vue 2
var __vue_render__ = function () {
  var _vm = this;

  var _h = _vm.$createElement;

  var _c = _vm._self._c || _h;
  ...
}
// Vue 3
import { openBlock, createBlock, createVNode, renderSlot } from 'vue';
```

4.2.2 같이 적용할 수 있는 방법

DEVIEW 2021

제약하여 사용하면 일부 가능

- Global API를 사용하지 않고 template 문법을 사용하지 않음
- Vue 2와 Vue 3을 동시에 사용할 수 있는 vNode을 사용
- Grid는 동시에 대응

```
function hForVue3(instance: any) {
 return function h(type: string, props: any, children: any[]) {
   let ref = null;

   if (props.ref) {
     ref = {
       i: instance.$,
       r: props.ref,
     };
   }
   return {
     // https://github.com/vuejs/vue-next/blob/master/packages/shared/src/shapeFlags.ts
     // ELEMENT 1, ARRAY_CHILDREN 16
     shapeFlag: 17,
     type,
     props,
     ref,
     children,
   }
 };
}
```

```
export default {
 render(this: any, h1: any) {
   const h = typeof h1 === "function" ? h1 : hForVue3(this);

   return h("div");
 },
};
```

4.2.3 어렵다.. @vue/compat

Vue 3에서 Vue 2 컴포넌트를 제약적 사용 가능

- vue-class-component는 안됨.
- Vue 2를 사용하는 서비스에서 Vue 3 컴포넌트는 불가능

```
"dependencies": {
-  "vue": "^2.6.12",
+  "vue": "^3.1.0-0",
+  "@vue/compat": "^3.1.0-0"
  ...
},
"devDependencies": {
-  "vue-template-compiler": "^2.6.12"
+  "@vue/compiler-sfc": "^3.1.0-0"
}
```

```
// webpack.config.js
module.exports = {
  resolve: {
    alias: {
      vue: '@vue/compat'
    }
  },
  module: {
    rules: [
      {
        test: /\.vue$/,
        loader: 'vue-loader',
        options: {
          compilerOptions: {
            compatConfig: {
              MODE: 2
            }
          }
        }
      }
    ]
  }
}
```

4.2.4 우리는 두 버전으로 지원

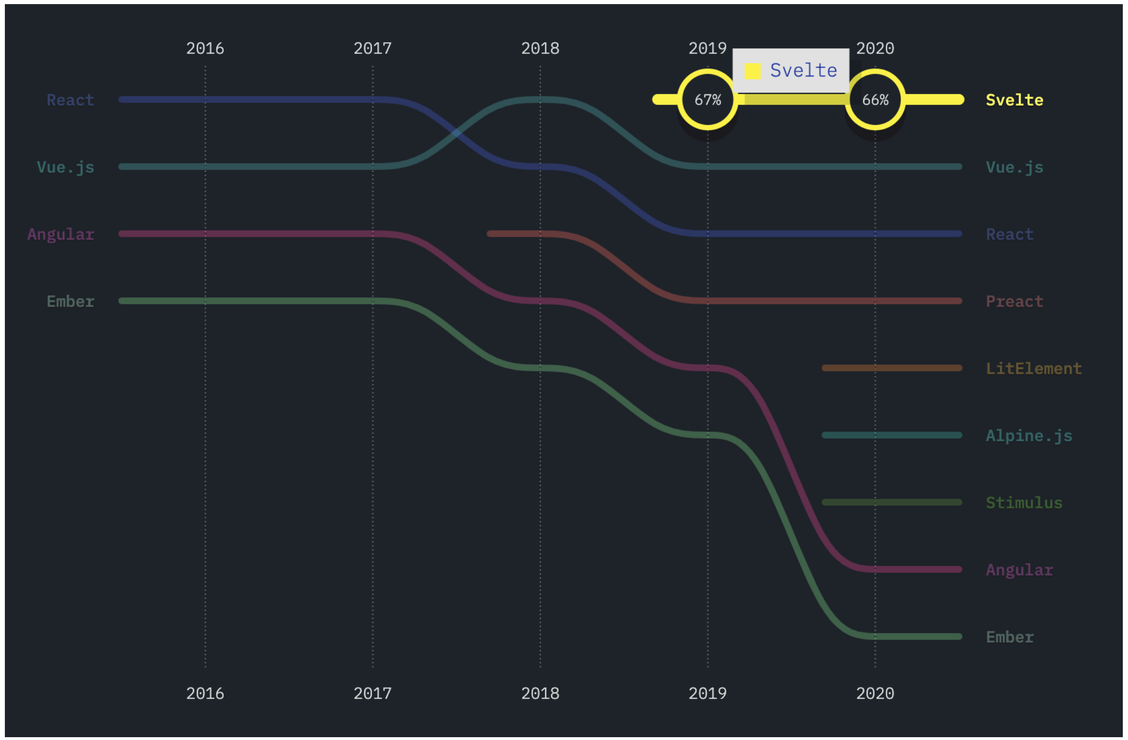
DEVIEW 2021

아키택쳐도 다르고 기능도 지속적으로 분리되어 제약이 생김

- 처음엔 비용 문제로 가능하면 동시 지원을 하려고 했음
- 시간이 지날수록 서로 다르게 될 텐데 Vue2와 Vue3 사이의 공통된 부분만 사용
- 추후 유지 보수 비용이 더 많이 발생할 것 같아 방향을 각각 개발하기로 전환
- Grid는 Vue2 / Vue3 동시 대응, Flicking/InfiniteGrid는 Vue2 / Vue3 각자 개발

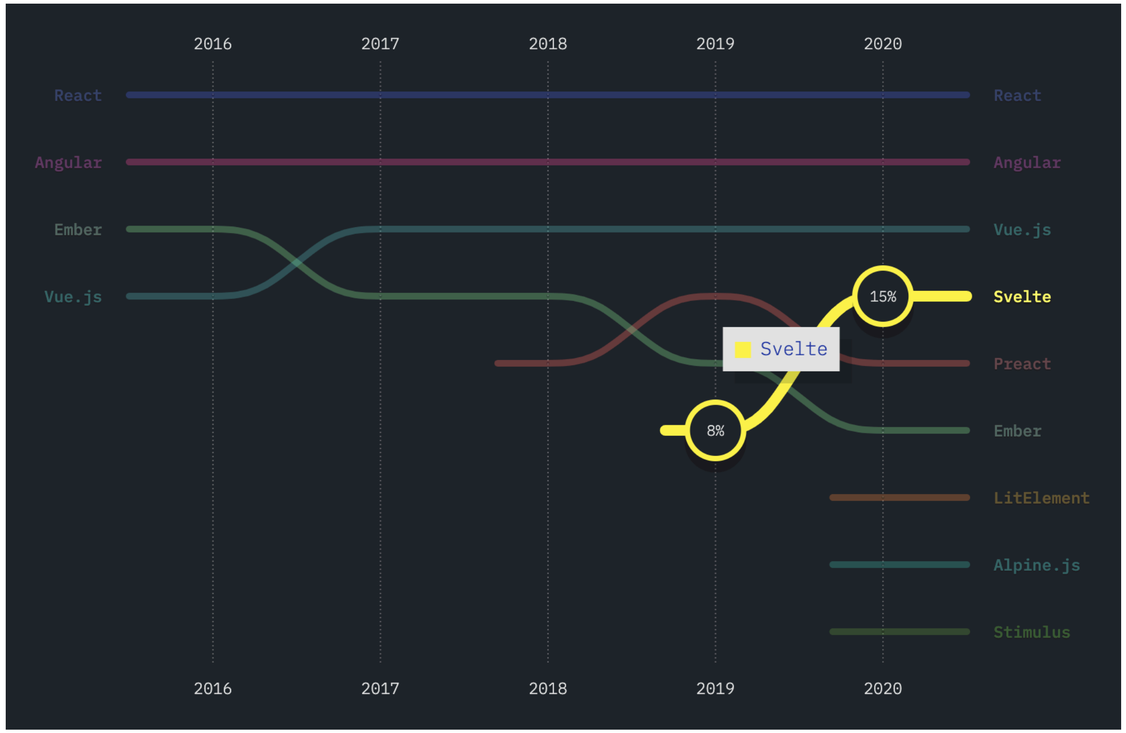
신흥 강자 Svelte

4.3.1 신흥 강자 Svelte



관심도 19년, 20년 1위

Interest: want to learn / (want to learn + not interested)



사용률 19년 6위 -> 20년 4위

Usage: (would use again + would not use again) / total

https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/

4.3.2 DOM 이슈

DOM 상태 정보의 접근 문제

- DOM Diff가 가능해야 CFC(Dynamic DOM)을 만들 수 있음
  - 예를 들어, Flicking 안쪽의 Panel 엘리먼트를 알아야 변경 사항을 반영
  - DOM의 상태를 접근할 수 없어, 별도의 FlickingPanel 같은 컴포넌트로 해결

DOM의 변경 문제

- 컴포넌트의 렌더링 되는 엘리먼트를 지정 없이, div로 렌더링
- DOM의 순서를 변경할 수 없어 별도의 방법을 찾아야 함
  - CSS의 order로 변경 (absolute 이면 이슈 없음)
  - 이로 인해 브라우저 커버리지가 IE10+으로 낮아짐

4.3.3 Vanilla JS의 메서드 확장 이슈

DEVIEW 2021

Svelte안에서는 Prototype으로 접근이 안됨

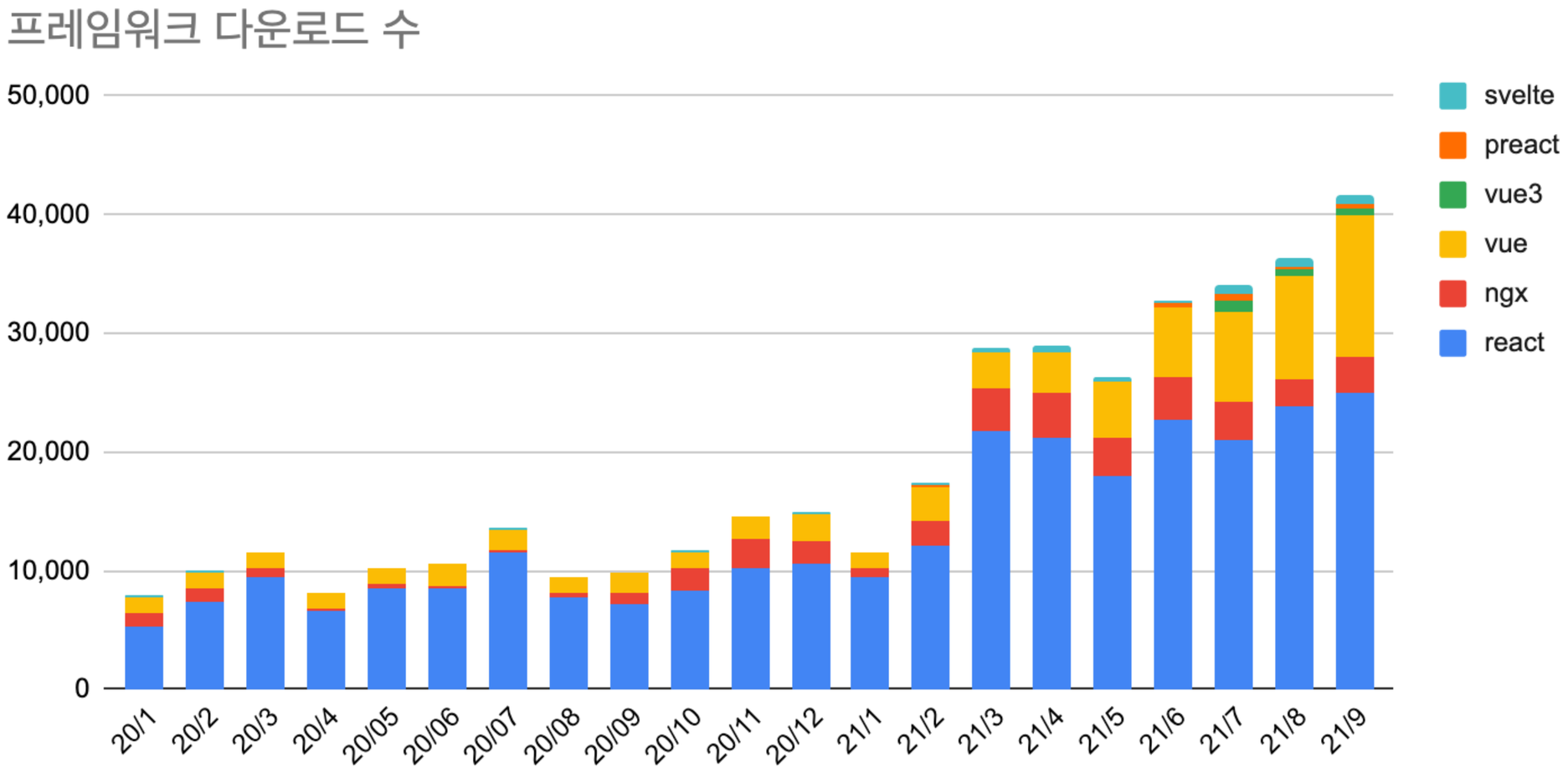
- svelte에서는 instance을 반환
- 외부에서 Prototype으로 확장함

```
// * InfiniteGrid.svelte
export function getInstance() {
    return vanillaGrid;
}

// * InfiniteGrid.js

export default /*#__PURE__*/ (() => {
  const prototype = InfiniteGrid.prototype;

  if (prototype) {
    INFINITEGRID_METHODS.forEach(name => {
      if (name in prototype) {
        return;
      }
      prototype[name] = function (...args) {
        const self = this.getInstance();
        const result = self[name](...args);

        if (result === self) {
          return this;
        } else {
          return result;
        }
      };
    });
  }
  return InfiniteGrid;
})();
```

5. 그리고 이후

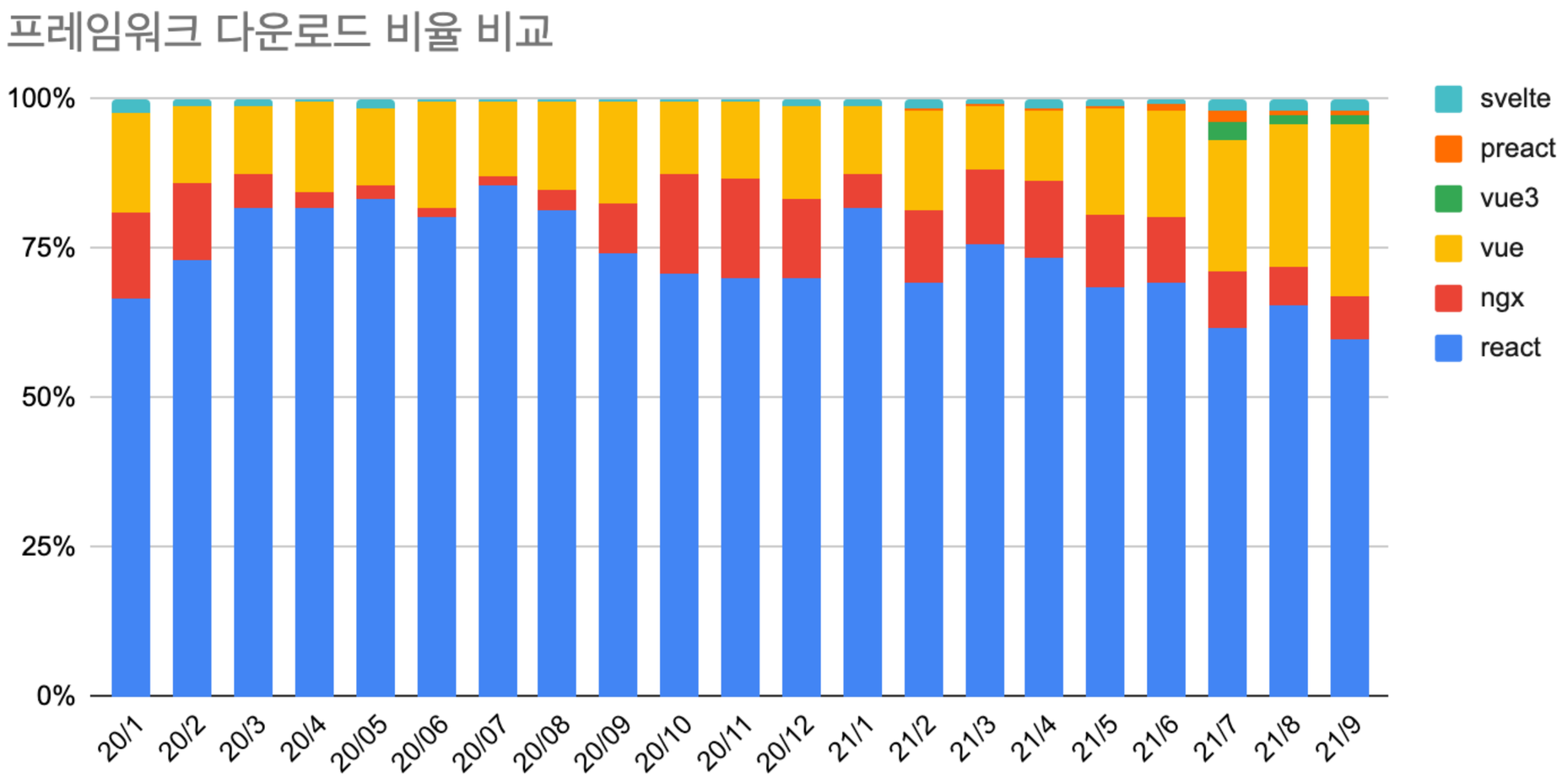
Scale UP

5.1.1 프레임워크 전체 다운로드



전체 다운로드 수 대비 프레임워크를 지원하는 컴포넌트의 프레임워크 다운로드 수는 약 40~50%

5.1.2 프레임워크 별 다운로드



React는 여전히 높지만 줄어드는 추세, Vue는 다음 강자, Svelte가 늘어나고 있음

5.1.3 Scale UP

DEVIEW 2021

egjs의 컴포넌트들은 모두 CFC 전환

- 현재 40~50%의 다운로드는 프레임워크에서 발생
- 다양한 타입의 CFC로 전환 예정하고, 적합하지 않는 컴포넌트들은 deprecated
- 기존의 보유한 컴포넌트 이외에 좀 더 기본적인 컴포넌트를 추가 개발

Quality UP

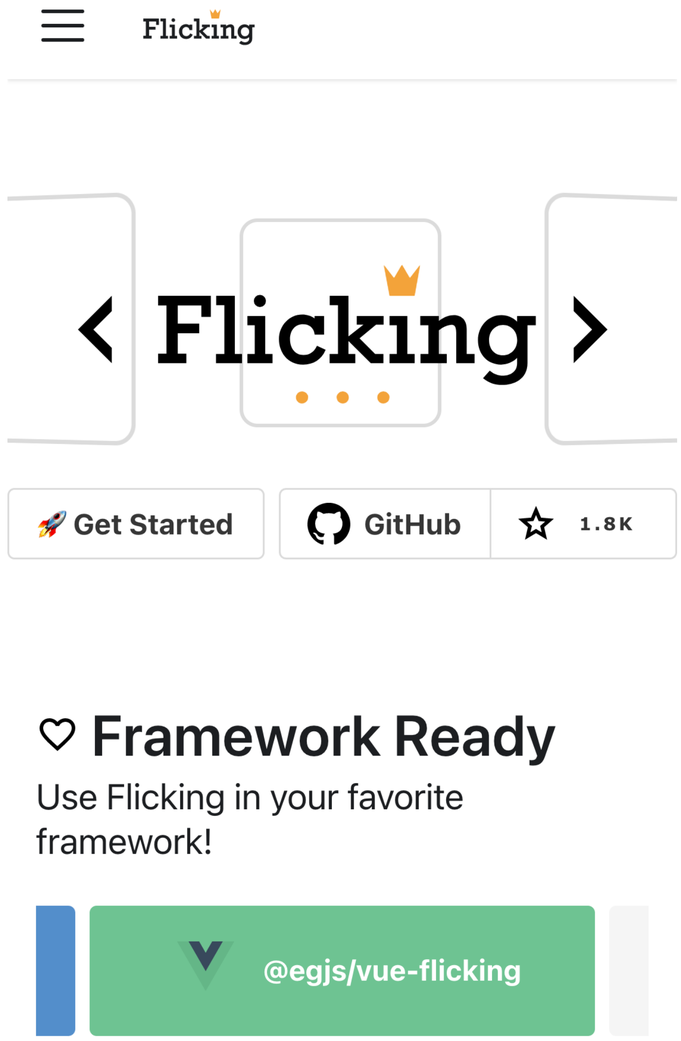
5.2.1 Documentation

기능도 많아져 JSDoc에서 CFC를 표현하기 아쉬움

- 프레임워크 별 예제 코드를 보여주기 어려움
- docusaurus가 최근에 많이 사용되는 문서화 도구

Docusaurus로 전환

- 다양한 템플릿
- 기존의 JSDoc의 내용을 다시 작성하기란 어려움
- 유사한 도구가 있었지만, 확장하기가 어려워
- 전환 도구 개발 후 mdx로 전환 (jsdoc-to-mdx)

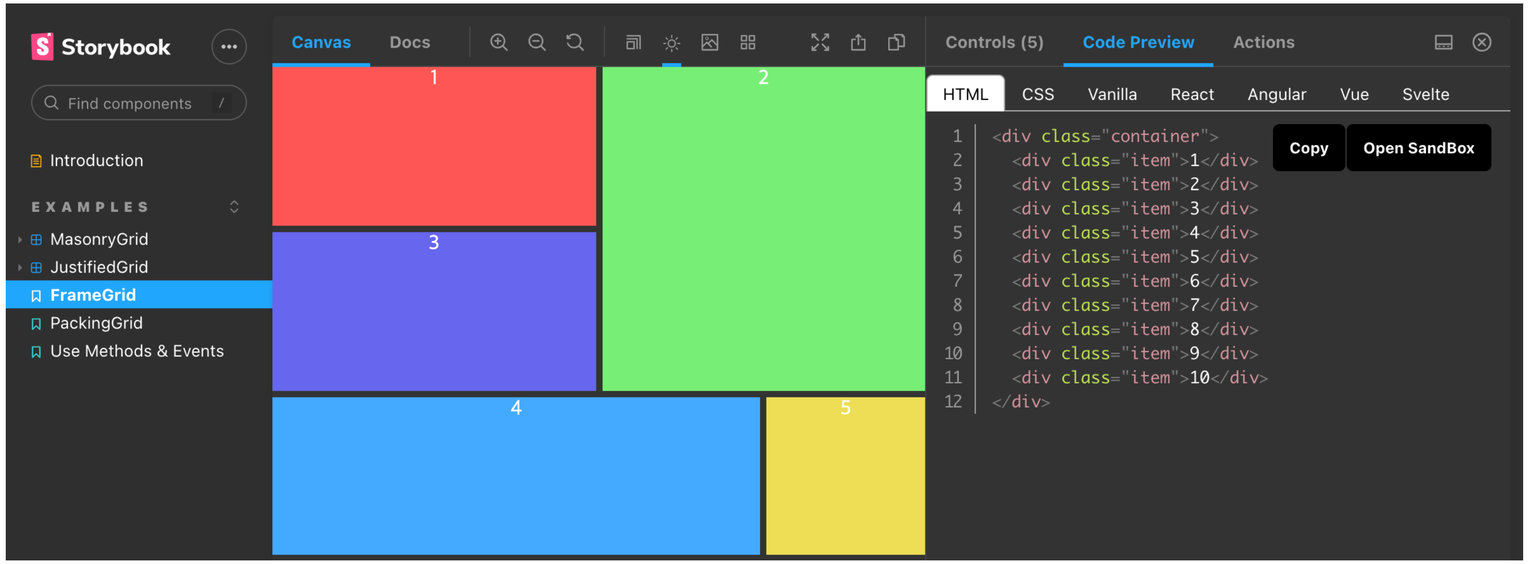


5.2.2 Demo

DEVIEW 2021

인터랙티브하게 값을 변경하면서 확인할 수 있는 데모가 필요

- StoryBook이 가장 이상적인 도구
- 다양한 예제를 Framework 별로 예제 코드를 보여주는게 필요
- 옵션이 변경되면 Framework 별로 예제 코드가 즉시 반영이 필요



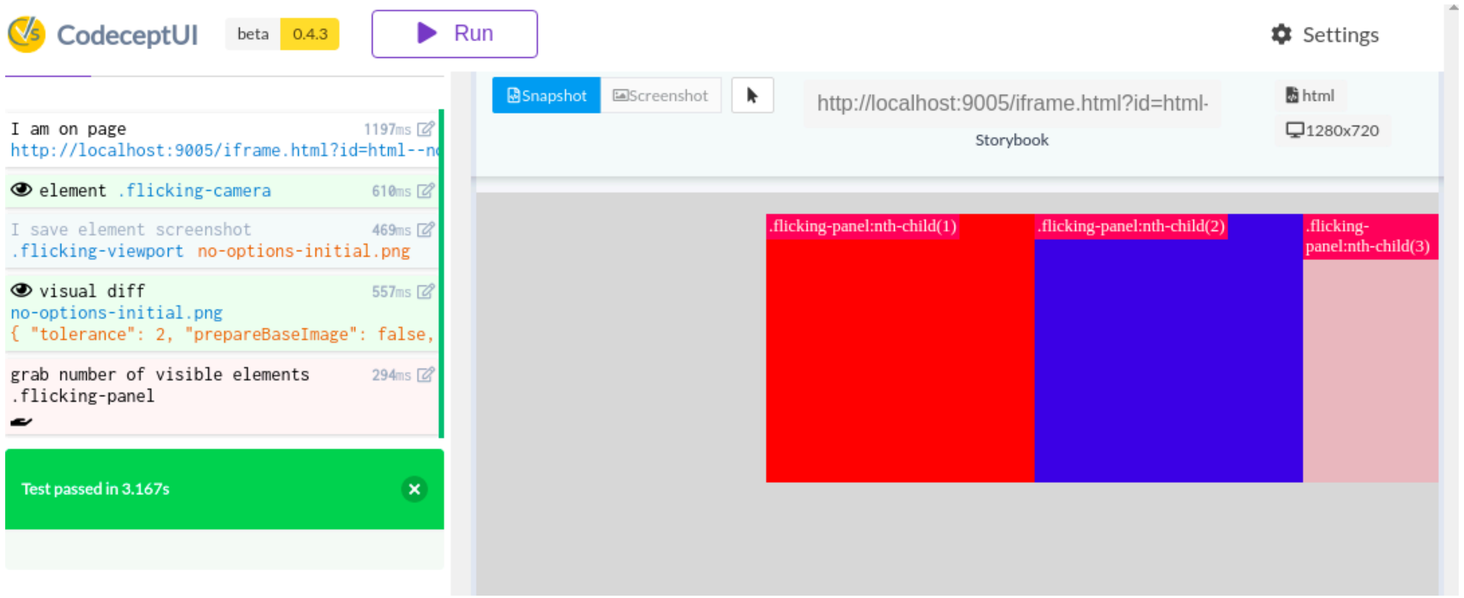
storybook-addon-preview

5.2.3 TestCase

DEVIEW 2021

Vanilla JS가 정상적으로 동작한다고 모든 Framework가 동작?

- 초기엔 Vanilla JS만 테스트 코드 작성
- Framework에서만 오동작 하는 사례들이 발생
- Framework는 데모를 만들고 이를 e2e로 테스트하여 대응



Storybook + CodeceptJS + Playwright

5.2.4 Quality UP

DEVIEW 2021

코드 작성 비용 보다 문서/데모 비용이 너무 큼

- 프레임워크 별 문서나 데모를 만드는 작업은 고비용 작업
- 시간의 문제도 있지만, 재미없는 작업이라 퀄리티가 낮음

테스트 케이스 실행 비용 증가

- 코어 기능은 Vanilla JS에서 단위 테스트를 작성함
- 프레임워크 e2e 테스트는 다양한 라이브러리의 설치부터 실행까지 시간이 오래 걸림
- 오래 걸려 횟수를 줄이고 실패 했을 때 맨붕

6. 정리

6.1 새로운 고민들

DEVIEW 2021

Server-Side Rendering

- React / Vue은 SSR일 경우 별도 라이프 사이클로 어렵지 않음
- Angular / Svelte는 별도의 작업을 통해서 해결해야 함

State Restore

- Vanilla : 컴포넌트에 필요한 데이터와 HTML로 복구 가능
- Framework : 상태 복구에 필요한 HTML은 데이터화 하여 처리

6.2 다른 타입의 CFC

DEVIEW 2021

Reactive

- React : Hooks
- Vue2 : Composition
- Vue3 : Reactive
- Angular : Directive
- Svelte : Store.writeable

Static DOM

- Dynamic DOM - DOM Sync

6.3 새로 생기고 발전하는 프레임워크

DEVIEW 2021

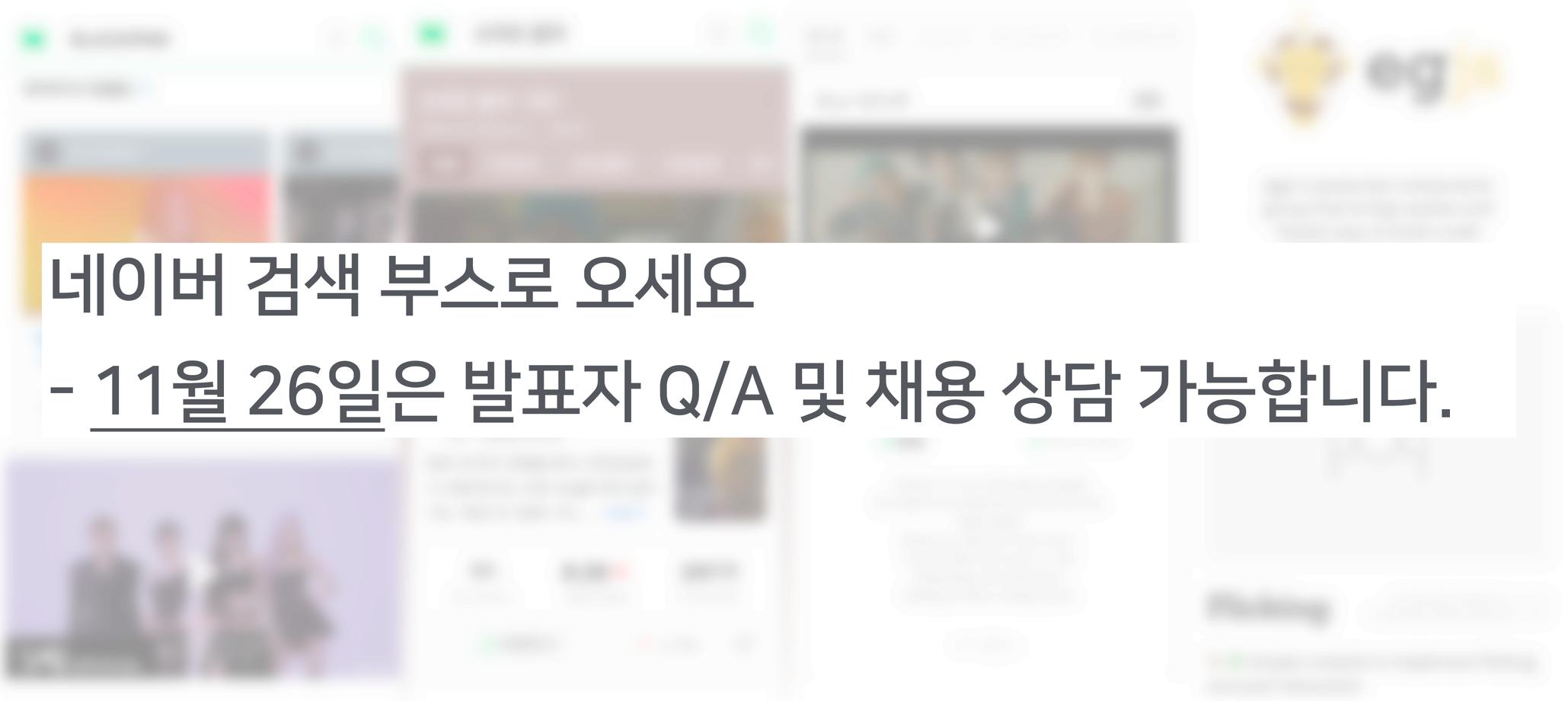
Vue2와 Vue3를 같이 적용할 수 있나?

- 제약을 가지고 컴포넌트를 만들 수 있음
- @vue/compat을 이용

신흥 강자 Svelte

- DOM 접근에 제약
- Prototype을 직접 접근할 수 없음

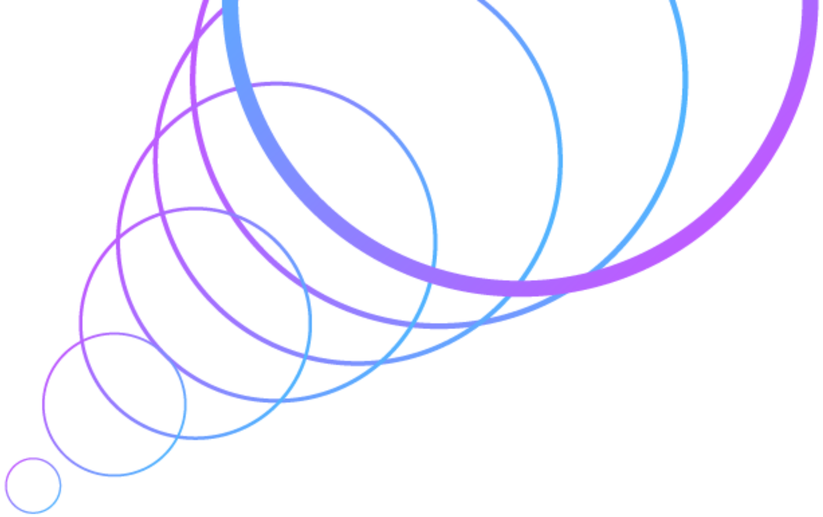
6.4 채용

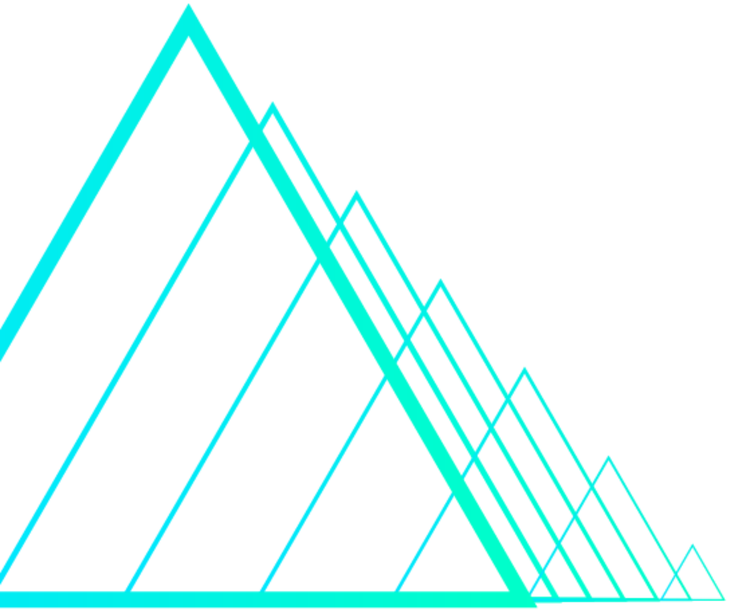


네이버 검색 부스로 오세요

- 11월 26일은 발표자 Q/A 및 채용 상담 가능합니다.

https://naver-career.gitbook.io/kr/service/search/reliability-platform/search-fe





Q & A

Thank You