하나의 코드로 React, Vue, Svelte 전부 지원하는 CFCs Reactive

최연규

NAVER Search

NAVER DEVIEW 2023

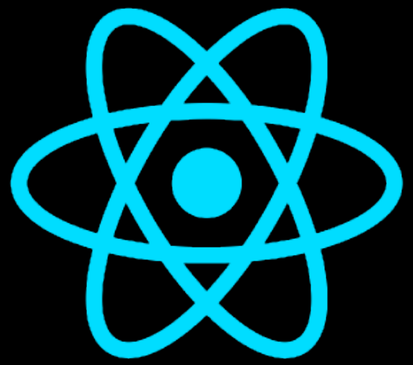
CONTENTS

1.Cross Framework Components (CFCs)

1.1 Frameworks

NAVER
DEVIEW
2023

우리들에게 알려진 프레임워크들



React

Vue

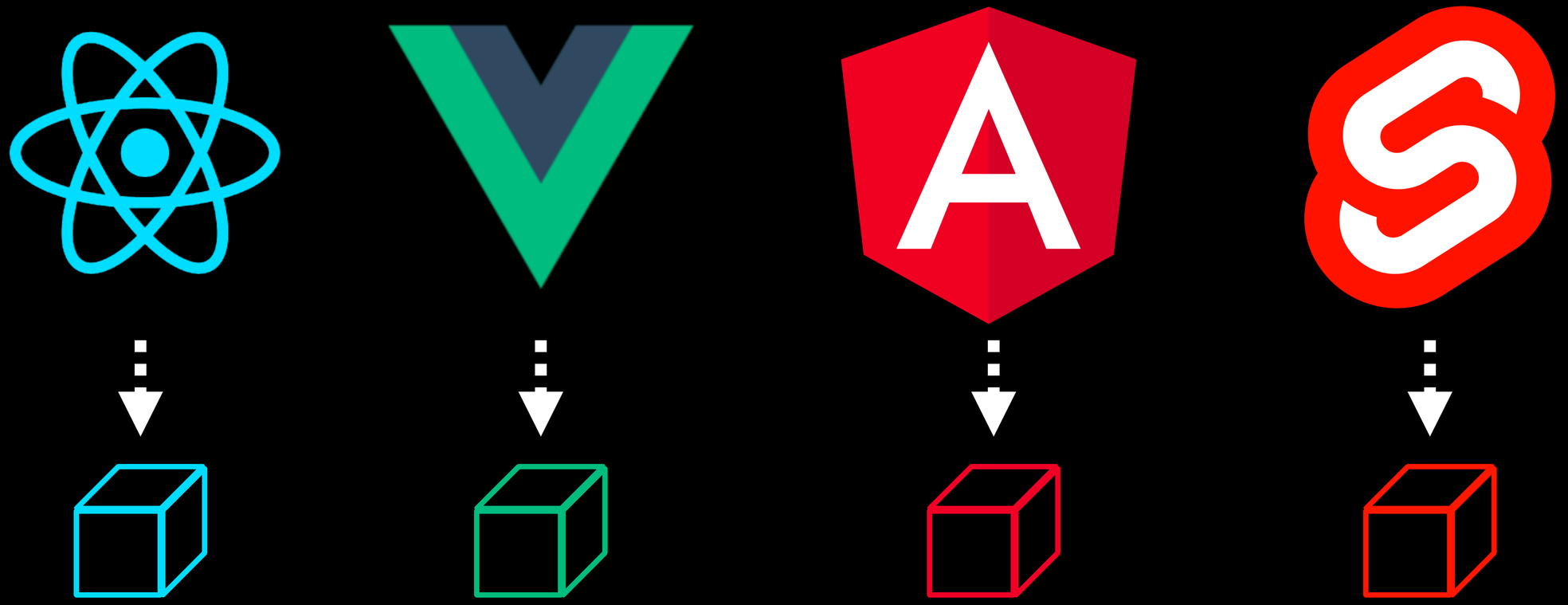
Angular

Svelte

1.1 Frameworks

NAVER
DEVIEW
2023

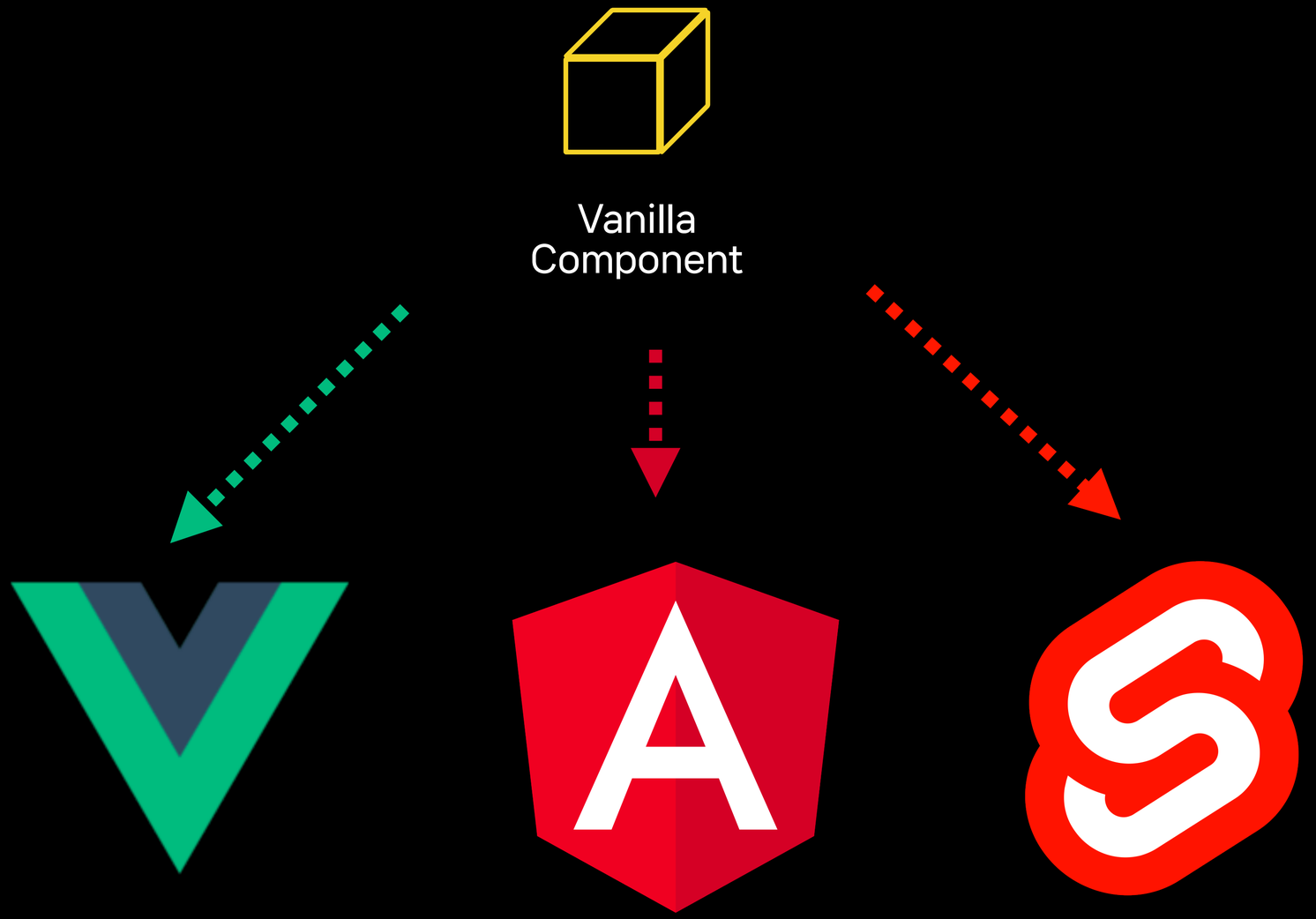
일반적으로는 프레임워크에 맞는 컴포넌트를 사용



1.1 Frameworks

NAVER
DEVIEW
2023

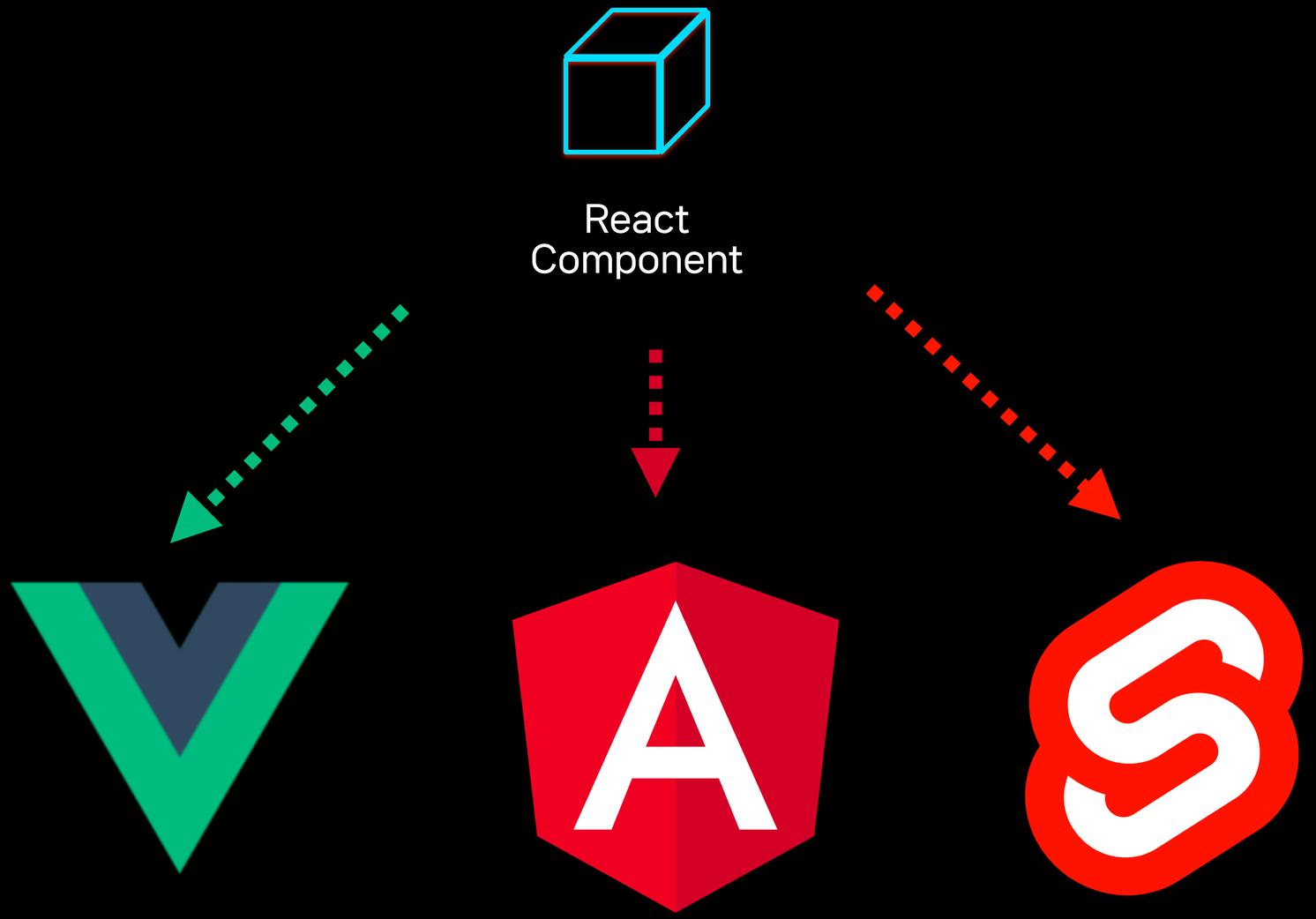
Vanilla 컴포넌트를 여러 프레임워크에서 사용이 가능할까?



1.1 Frameworks

NAVER
DEVIEW
2023

React 컴포넌트를 다른 프레임워크에서 사용이 가능할까?



1.2 egjs

NAVER
DEVIEW
2023

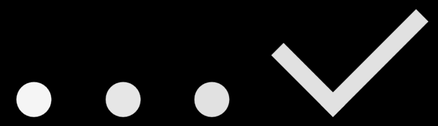
egjs에서 운영하는 컴포넌트들

View 360

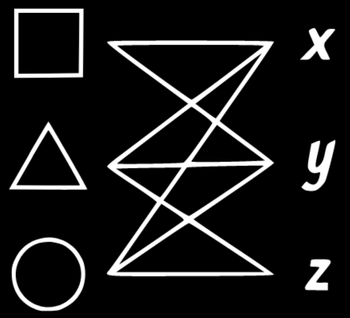
Flicking

View 3D

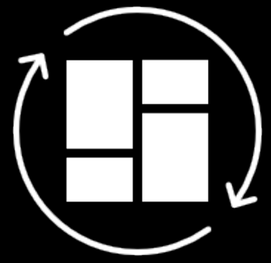
Conveyer



ImReady



Axes



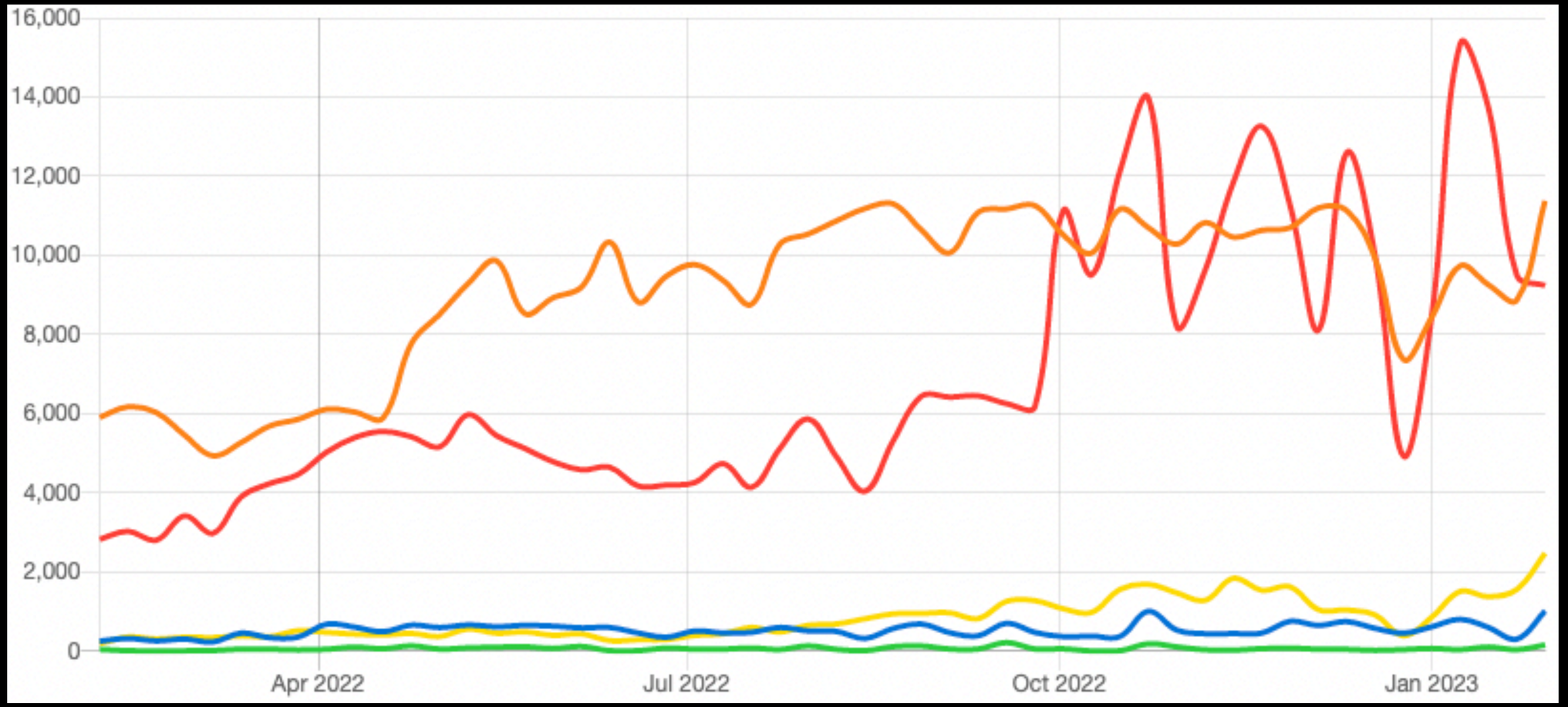
InfiniteGrid

1.2 egjs

NAVER
DEVIEW
2023

컴포넌트의 프레임워크별 다운로드 현황

- 조사 대상 프레임워크: React, Vue 2, Vue 3, Svelte, Angular



1.2 egjs

NAVER
DEVIEW
2023

egjs 컴포넌트의 프레임워크 지원 현황

- 세모: 문제는 없지만 내부 개선이 필요한 컴포넌트

	Vanilla	React	Vue 2 / 3	Angular	Svelte
Flicking	✓	△	△	△	△
InfiniteGrid	✓	△	△	△	△
View3D	✓	△	△	△	△
View360	✓	△	△	△	△
Conveyer	✓	✓	✓	△	✓
Axes	✓	✓	✓		✓
ImReady	✓	✓	✓	△	✓

1.3 CFCs

NAVER
DEVIEW
2023

CFCs (Cross Framework Components)

- CFCs는 하나의 컴포넌트로 프레임워크를 가로질러 사용할 수 있는 구조의 컴포넌트
- **CFCs 모듈은 하나의 컴포넌트를 모든 프레임워크에서 사용할 수 있는 컴포넌트로 변환을 해주는 모듈**

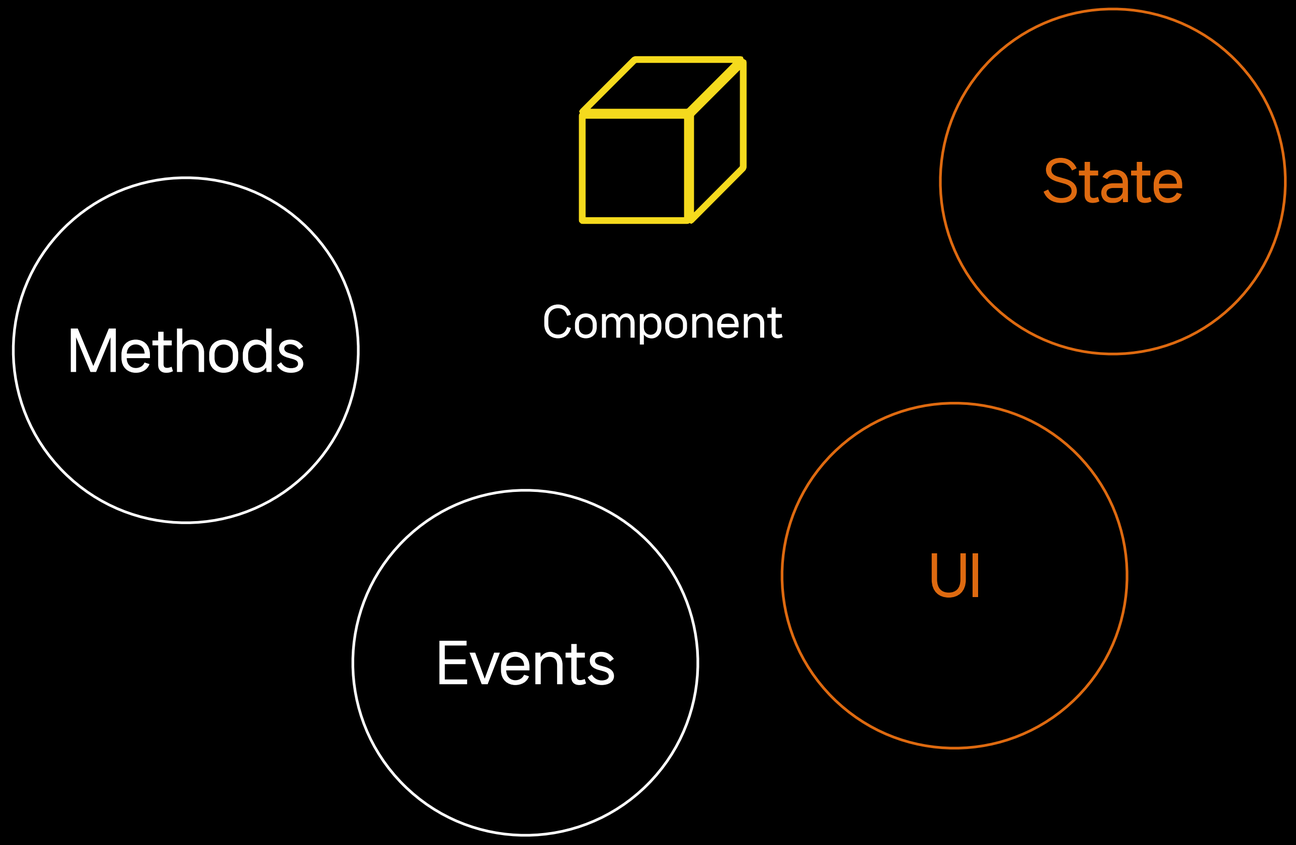
누구에게 도움이 될까?

- **개발중인 컴포넌트를 다양한 프레임워크에 지원하고 싶은 경우**
- **기존의 만들어진 컴포넌트를 다양한 프레임워크에 지원하고 싶은 경우**

1.3 CFCs Concepts

NAVER
DEVIEW
2023

컴포넌트는 Methods, Events, UI, State 1개 이상으로 구성

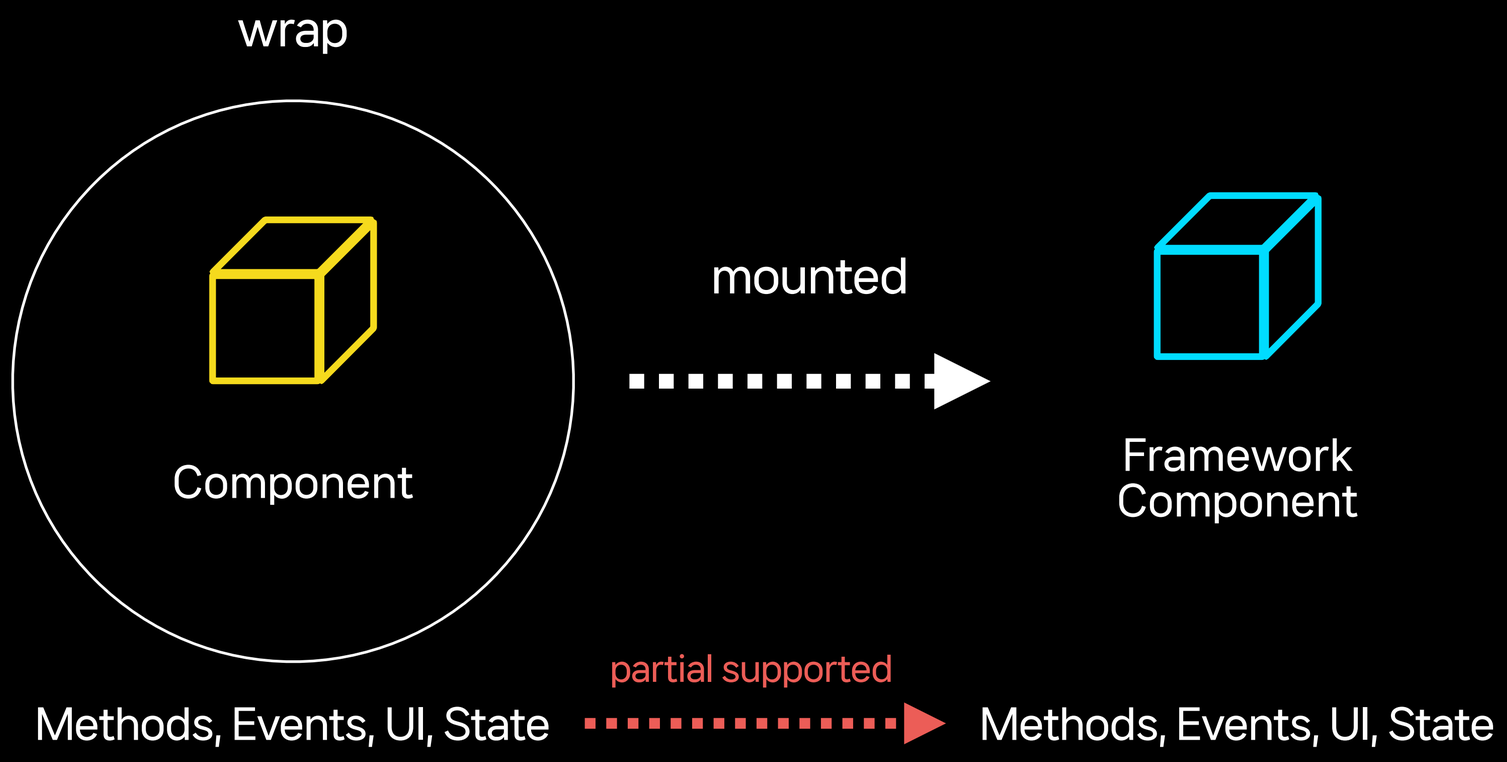


1.3 CFCs Concepts

NAVER
DEVIEW
2023

프레임워크 전용 컴포넌트가 아니라면 단순 Wrapping하여 사용하여 프레임워크를 지원

- ex) react-infinitegrid@1.x



1.3 CFCs Concepts

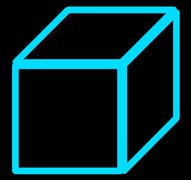
NAVER
DEVIEW
2023

여러 프레임워크를 지원

- 프레임워크마다 비슷한 코드가 존재하고 크기도 N배
- 기능 추가 / 버그 수정도 N배

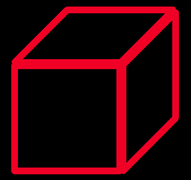
X 5

Vanilla
Component

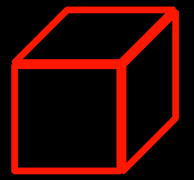


React
Component

Vue
Component



Angular
Component

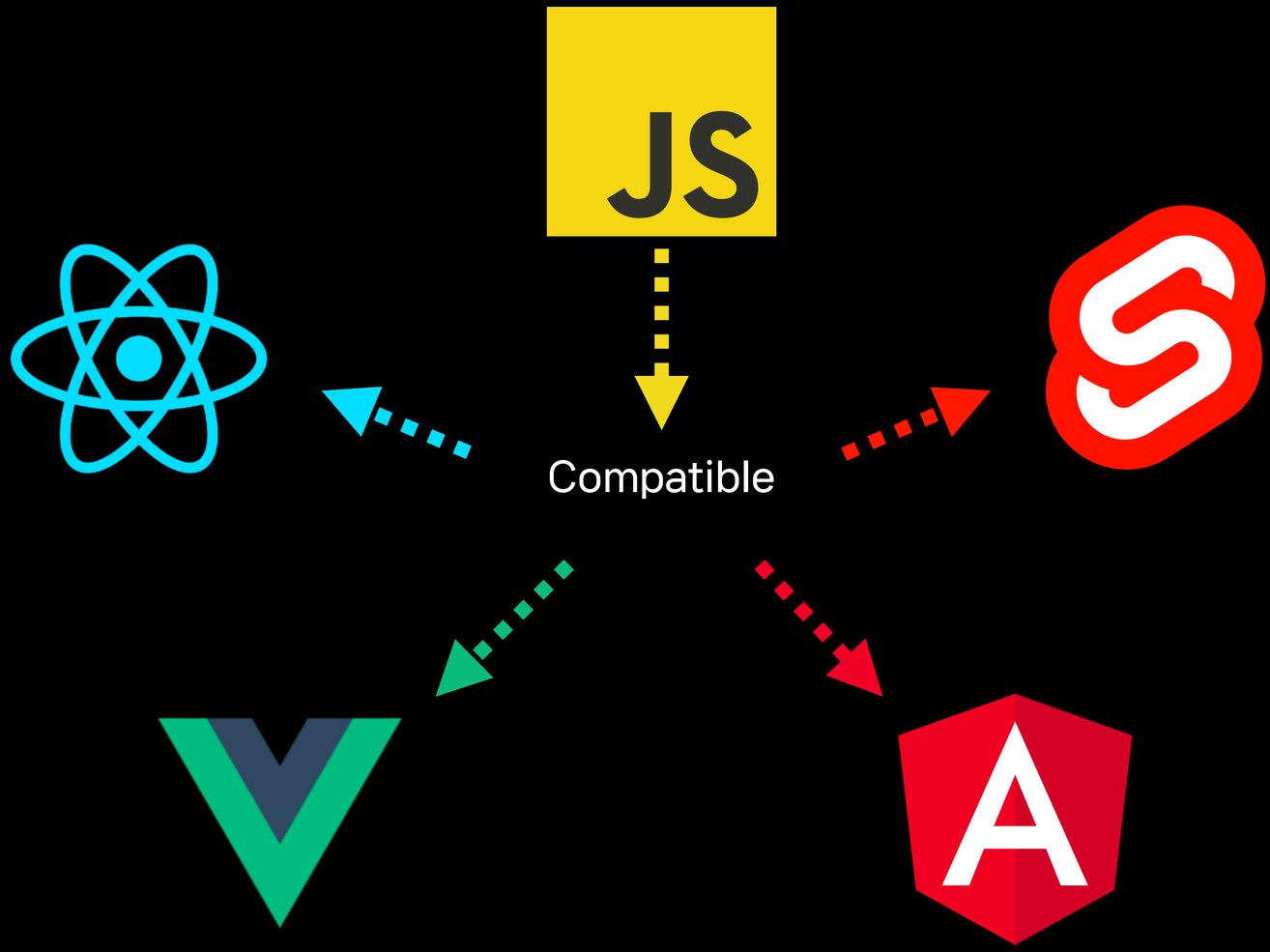


Svelte
Component

1.3 CFCs Concepts

NAVER
DEVIEW
2023

CFCs는 Compatible을 통해 하나의 공통 코드를 여러 프레임워크에 지원

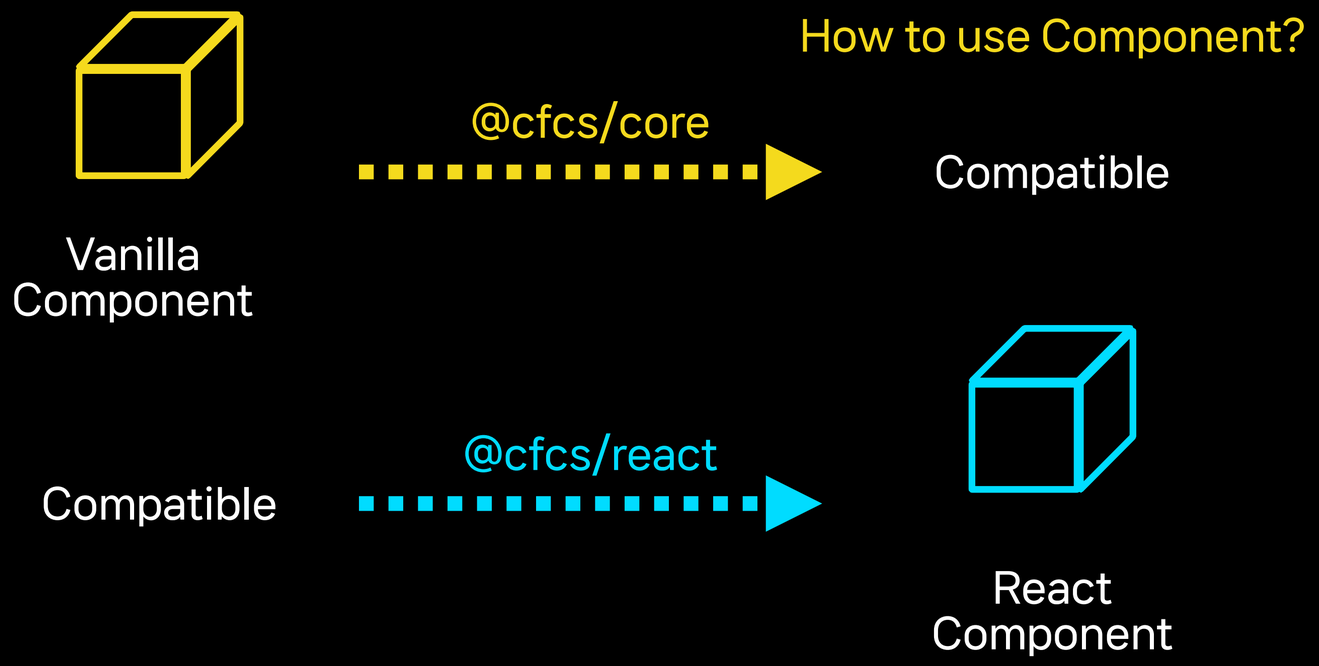


1.3 CFCs Concepts

NAVER
DEVIEW
2023

CFCs에서 제공하는 모듈을 통해 호환 코드를 작성

호환 코드를 CFCs에서 제공하는 모듈을 통해 프레임워크를 지원

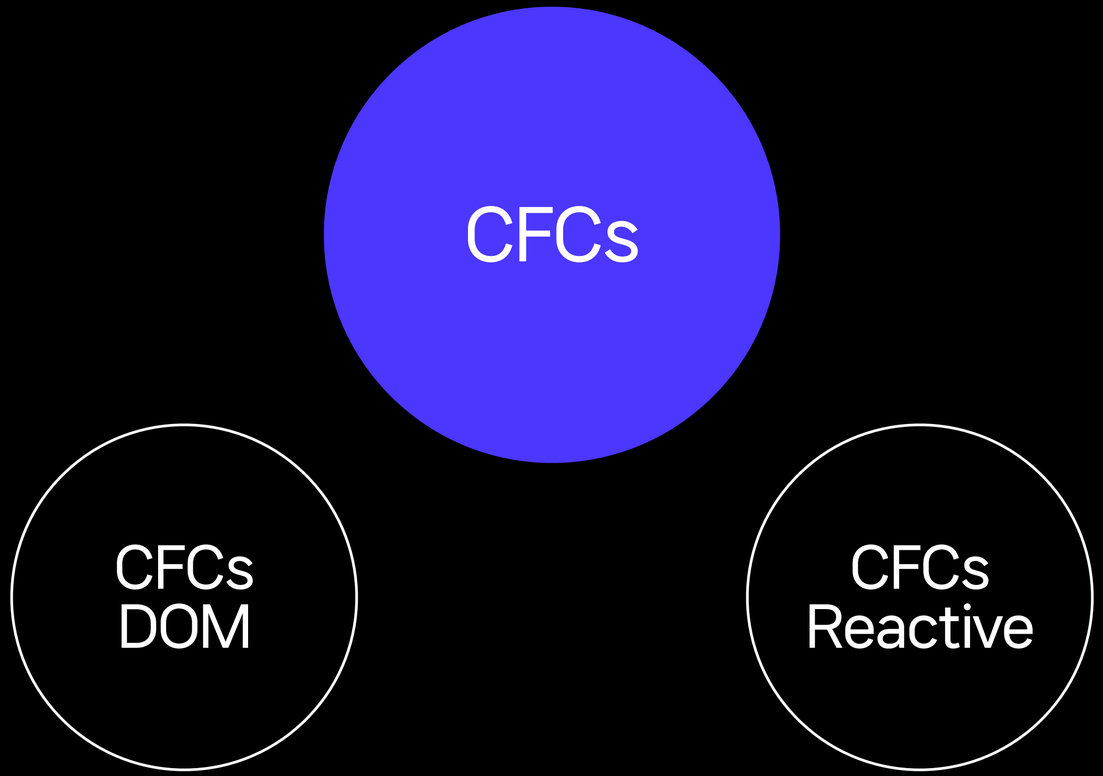


1.3 CFCs Concepts

NAVER
DEVIEW
2023

CFCs는 2가지 방법을 제공.

- DOM: DOM을 생성하거나 변경하는 UI 컴포넌트를 프레임워크에서 지원
- Reactive: DOM보다 유틸 컴포넌트에 상태를 부여하여 프레임워크에서 지원

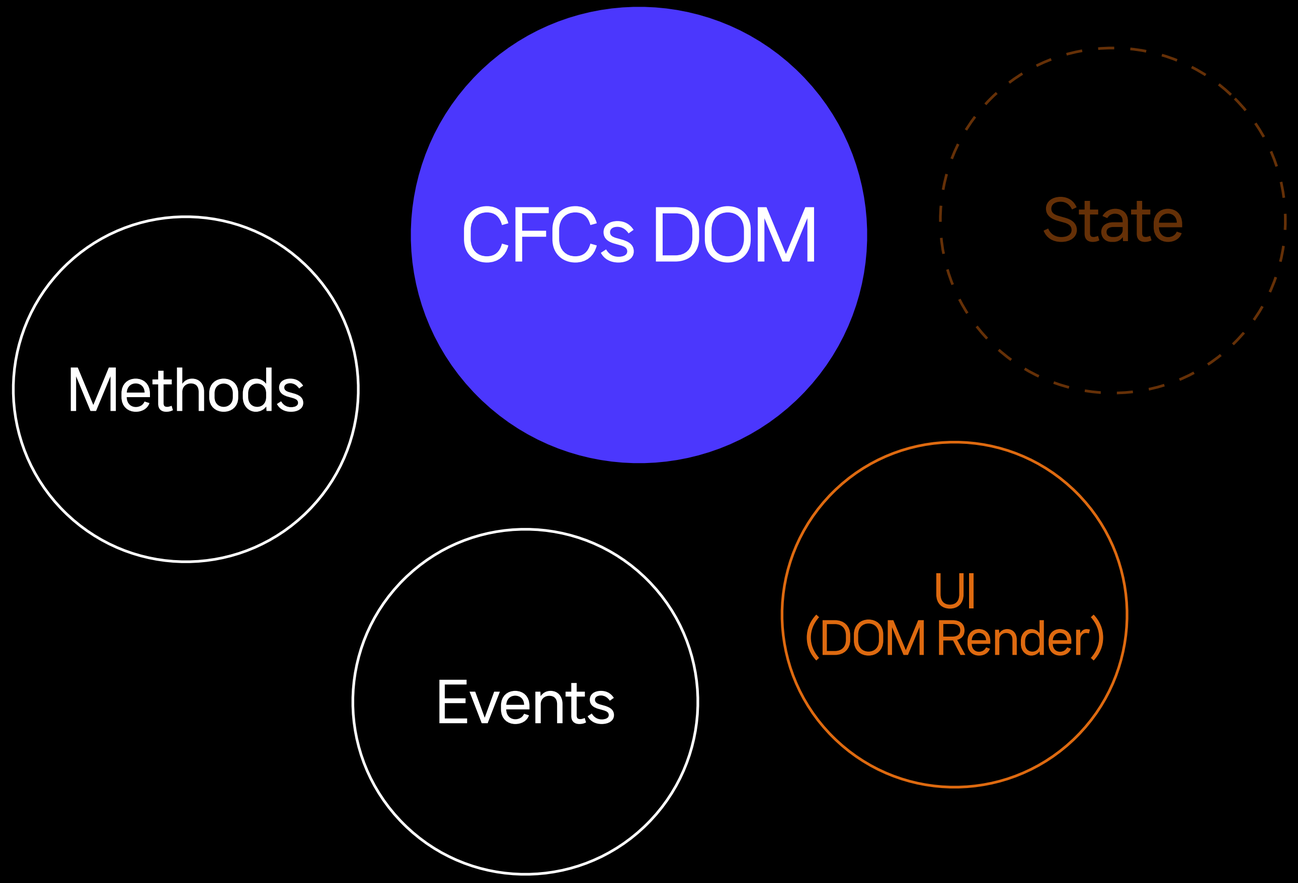


2. CFCs DOM
(DEVIEW 2019)

2.1 CFCs DOM (DEVIEW 2019)

NAVER
DEVIEW
2023

- 처음에는 CFCs DOM 방법만 존재
- CFCs DOM은 DOM render 기능을 프레임워크에 위임하여 프레임워크를 지원하는 방법

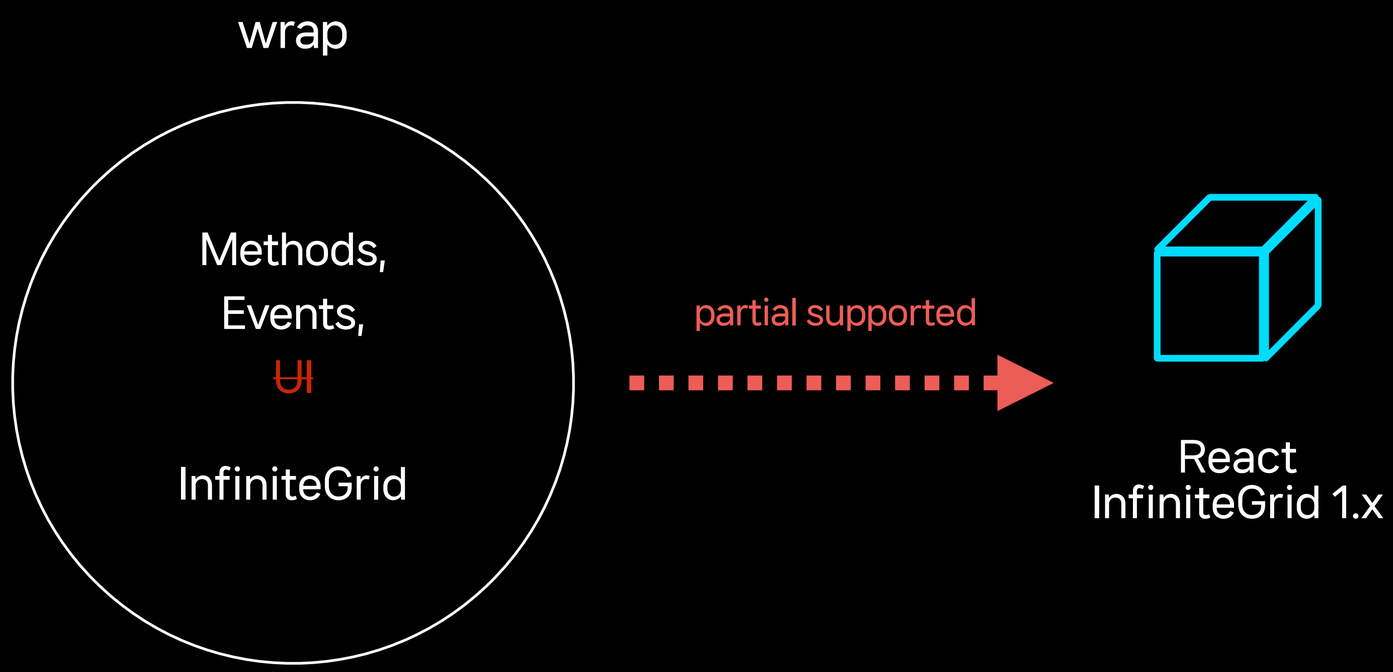


2.1 CFCs DOM (DEVIEW 2019)

NAVER
DEVIEW
2023

단순 Wrapping하면 Recycle, Infinite 등 Children, JSX과 관련된 UI 기능에 문제

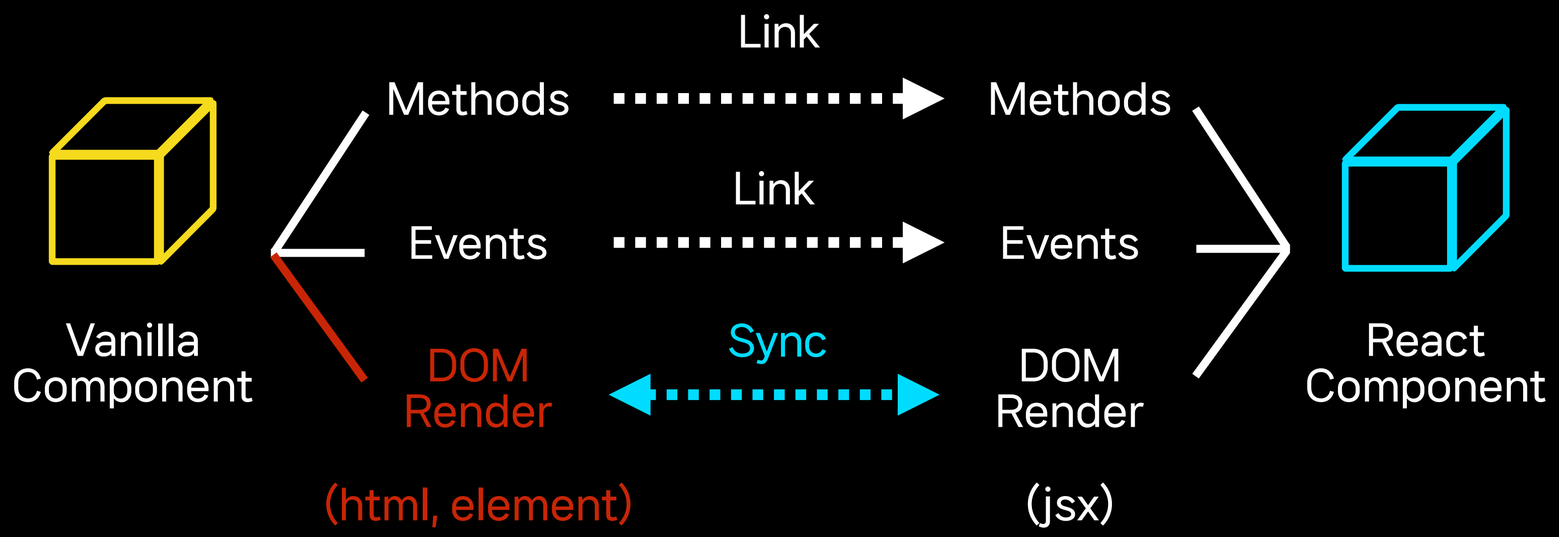
ex) React InfiniteGrid 1.x



2.1 CFCs DOM (DEVIEW 2019)

NAVER
DEVIEW
2023

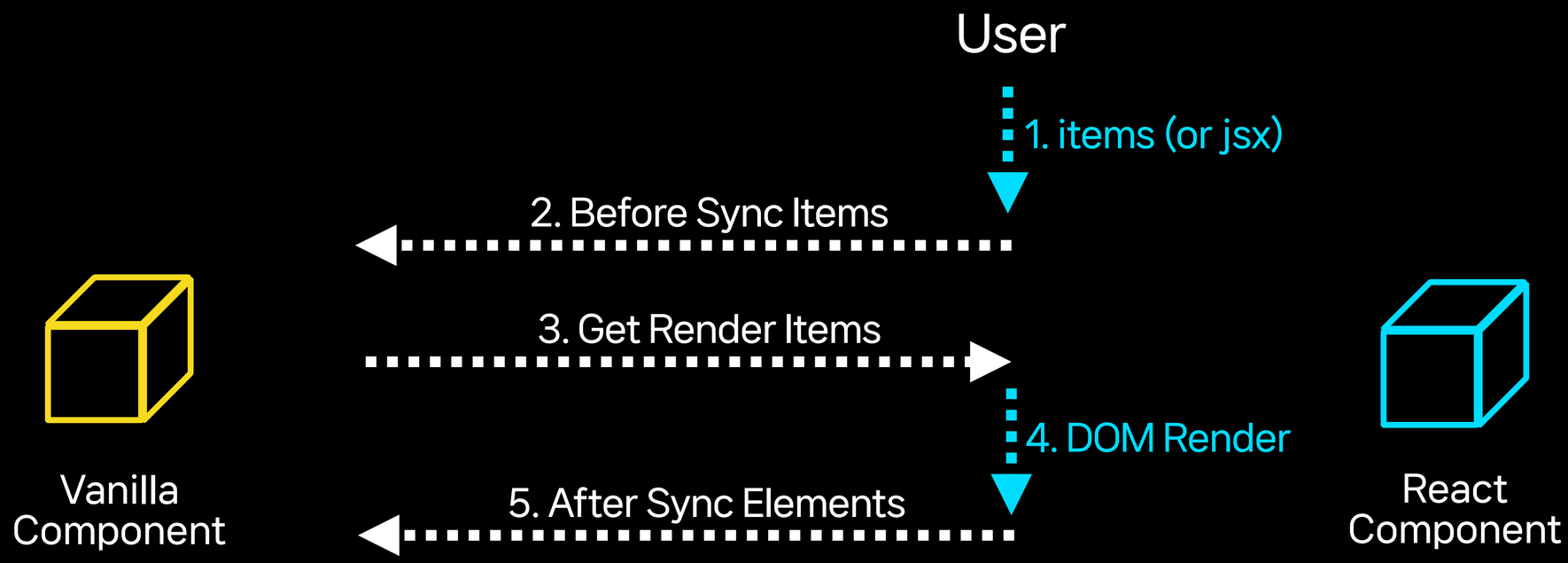
- DOM Render는 프레임워크의 동작을 반하면 안되기 때문에 Sync작업이 필요



2.1 CFCs DOM (DEVIEW 2019)

NAVER
DEVIEW
2023

- React에서 JSX와 Element의 Sync 과정



2.1 CFCs DOM (DEVIEW 2019)

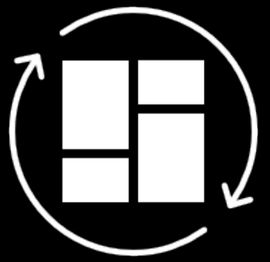
NAVER
DEVIEW
2023

- CFCs DOM 방법을 통해 만들어진 컴포넌트들

View 360

View 3D

Flicking



InfiniteGrid

2.1 CFCs DOM 지연 (DEVIEW 2019)

NAVER
DEVIEW
2023

CFCs DOM 방법으로 만들어진 컴포넌트들

- 하지만 DOM Rendering하는 방법이 다양하여 모듈로써 제작 지연

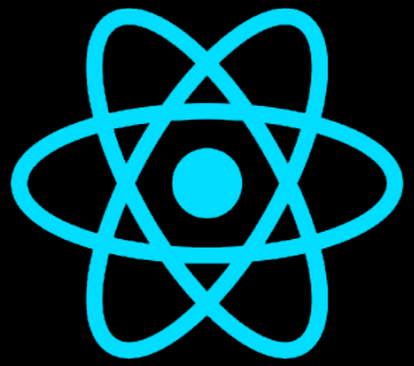
	Static	Infinite	Recycle	Clone
View3D	O			
View360	O			
Grid	O	O		
InfiniteGrid	O	O	O	
Flicking	O	O	O	O

3. Function Component And Reactive Component

3.1 Function Component

NAVER
DEVIEW
2023

- 클래스 컴포넌트: 클래스 형태의 컴포넌트와 인스턴스를 생성 및 메서드 호출
- 함수 컴포넌트: 함수 형태의 컴포넌트와 함수 호출



React

Vue

Svelte

함수 컴포넌트를 지원하는 대표적인 프레임워크들

3.2 Function Component의 특징

NAVER
DEVIEW
2023

- 함수를 호출하는 방식
  - React는 render 때마다 호출
  - Vue, Svelte은 생성될 때 최초 1번 호출
- 인스턴스가 존재하지 않기 때문에 내부에서는 this 키워드 접근이 불가
- 내부에서 this를 접근하지 못하나 외부에서는 방법에 따라 인스턴스(또는 오브젝트) 접근이 가능

3.2 Function Component의 상태

NAVER
DEVIEW
2023

- React는 순서에 기반하여 저장된 값을 읽고 쓰는 Hooks 방식의 상태 (value와 setValue 함수)

```
const [value, setValue] = useState();
```

- Vue는 Reference 방식의 상태 (Proxy, Getter, Setter)

```
const valueRef = ref();
valueRef.value;
```

- Svelte는 변수가 상태 (컴파일 과정에서 변수 값이 설정되는 문법에 업데이트를 하는 코드가 추가)

```
let value = 0;
```

3.2 Function Component의 Hooks

NAVER
DEVIEW
2023

- Hooks
  - 인스턴스가 없기 때문에 내부 컴포넌트 정보가 존재하고 Hooks를 순서대로 추가
  - root부터 시작하여 하위 컴포넌트까지 초기화 및 렌더링하면서 현재 컴포넌트 정보가 변경
  - 비동기로 호출 불가
- Lifecycle Hooks
  - 메서드 형태가 아닌 함수 호출 방식을 통해 현재 컴포넌트 정보에 Hooks 정보를 저장
- React Hooks
  - 매번 함수 호출을 하기 때문에 내부 컴포넌트 정보와 순서에 의존
  - 함수 컴포넌트에서 사용하는 기능은 대부분 Hooks 함수
  - Hooks 함수 호출의 개수와 종류가 변경되는 경우 에러가 발생 (조건문, 반복문, forEach 사용에 주의)

3.3 Reactive Component

NAVER
DEVIEW
2023

Reactive Component는 특정 조건에 따라 상태 값이 변하거나 이벤트가 발생하는 컴포넌트

- 직접 상태 변경 호출하는게 아닌 이벤트나 내부적으로 사용하는 컴포넌트에 의하여 자동으로 변경
- 변경되는 상태를 이용하여 다채로운 컴포넌트를 만들기 때문에 함수컴포넌트에 적합
- State와 Lifecycle Hooks를 적절히 이용
- 상태 기반의 UI 구성

3.4 Before React Hooks

NAVER
DEVIEW
2023

- 이벤트 또는 Children의 함수를 통해 Hooks와 비슷한 기능을 구현
  - ex) react-axes 1.x
- 그렇기 때문에 prop 또는 내부에만 영향끼치고 부모 컴포넌트에는 영향을 끼치지 않음

```
<Axes>
  {(frame) => <div style={{
        left: `${frame.left}px`,
  }}></div>}
</Axes>
```

3.5 Window Size Reactive Component

NAVER
DEVIEW
2023

화면의 사이즈의 변화에 따라 업데이트 하는 Reactive 컴포넌트

- 변화에 따라 Component 또는 App이 업데이트
- window의 resize 이벤트를 통해 확인
- 상태: width, height

```
window.addEventlistener("resize", () => {
  console.log(window.innerWidth);
  console.log(window.innerHeight);
});
```

https://github.com/egjs/deview2023

3.6 React Component

NAVER
DEVIEW
2023

화면의 사이즈 (width, height) 에 따라 재렌더링하는 Reactive Component

- 함수에서 useState를 사용하여 state(상태) 선언

```
const { width, height } = useWindowSize();

<div style={{
  width: `${width / 10}px`,
  height: `${height / 10}px`,
}}></div>
```

https://github.com/egjs/deview2023

3.6 React Component

NAVER
DEVIEW
2023

Reactive Component는 크게 state, mounted, unmounted, result로 구성

- 함수에서 useState를 사용하여 state(상태) 선언

state

mounted

unmounted

result

```
export function useWindowSize() {
    const [width, setWidth] = useState(0);
    const [height, setHeight] = useState(0);

    useEffect(() => {
        const callback = () => {
            setWidth(window.innerWidth);
            setHeight(window.innerHeight);
        };
        window.addEventListener("resize", callback);

        return () => window.removeEventListener("resize", callback);
    }, []);

    return { width, height };
}
```

3.6 Vue 2 / Vue 3 Component

NAVER
DEVIEW
2023

화면의 사이즈 (width, height) 에 따라 재렌더링하는 Reactive Component

- setup에서 ref(또는 reactive)를 사용하여 상태를 선언
- template에서 사용한 상태의 변화만 업데이트가 발생

```
<script setup>
  const { width, height } = useWindowSize();
</script>
<template>
  <div :style="`width: ${width / 10}px;`"></div>
</template>
```

https://github.com/egjs/deview2023

3.6 Vue 2 / Vue 3 Component

NAVER
DEVIEW
2023

Reactive Component는 크게 state, mounted, unmounted, result로 구성

state

mounted

unmounted

result

```
export function useWindowSize() {
  const width = ref(0);
  const height = ref(0);
  const callback = () => {
    width.value = window.innerWidth;
    height.value = window.innerHeight;
  };

  onMounted(() => {
    window.addEventListener("resize", callback);
  });
  onUnmounted(() => {
    window.removeEventListener("resize", callback);
  });

  return { width, height };
}
```

3.6 Svelte Component

NAVER
DEVIEW
2023

화면의 사이즈 (width, height) 에 따라 재렌더링하는 Reactive Component
- 외부 컴포넌트(또는 함수)를 사용하는 경우 store를 사용하여 상태를 선언
- store를 접근하기 위해서는 $ 접두사를 사용

```
<script>
  const { width, height } = useWindowSize();
</script>
<div style={`width: ${$width / 10}px;`}></div>
```

https://github.com/egjs/deview2023

3.6 Svelte Component

NAVER
DEVIEW
2023

Reactive Component는 크게 state, mounted, unmounted, result로 구성

state

mounted

unmounted

result

```
export function useWindowSize() {
  const width = writable(0);
  const height = writable(0);
  const callback = () => {
    width.set(window.innerWidth);
    height.set(window.innerHeight);
  };
  onMount(() => {
    window.addEventListener("resize", callback);
  });
  onDestroy(() => {
    if (typeof window !== "undefined") {
      window.removeEventListener("resize", callback);
    }
  });
  return { width, height };
}
```

3.6 CFCs를 위한 공통점 분석

NAVER
DEVIEW
2023

React, Vue 2 / 3, Svelte의 Reactive Component는 state, mounted, unmounted, result로 구성

1. state 선언
2. mounted에서 등록
3. unmounted에서 해제
4. result에서 state를 반환

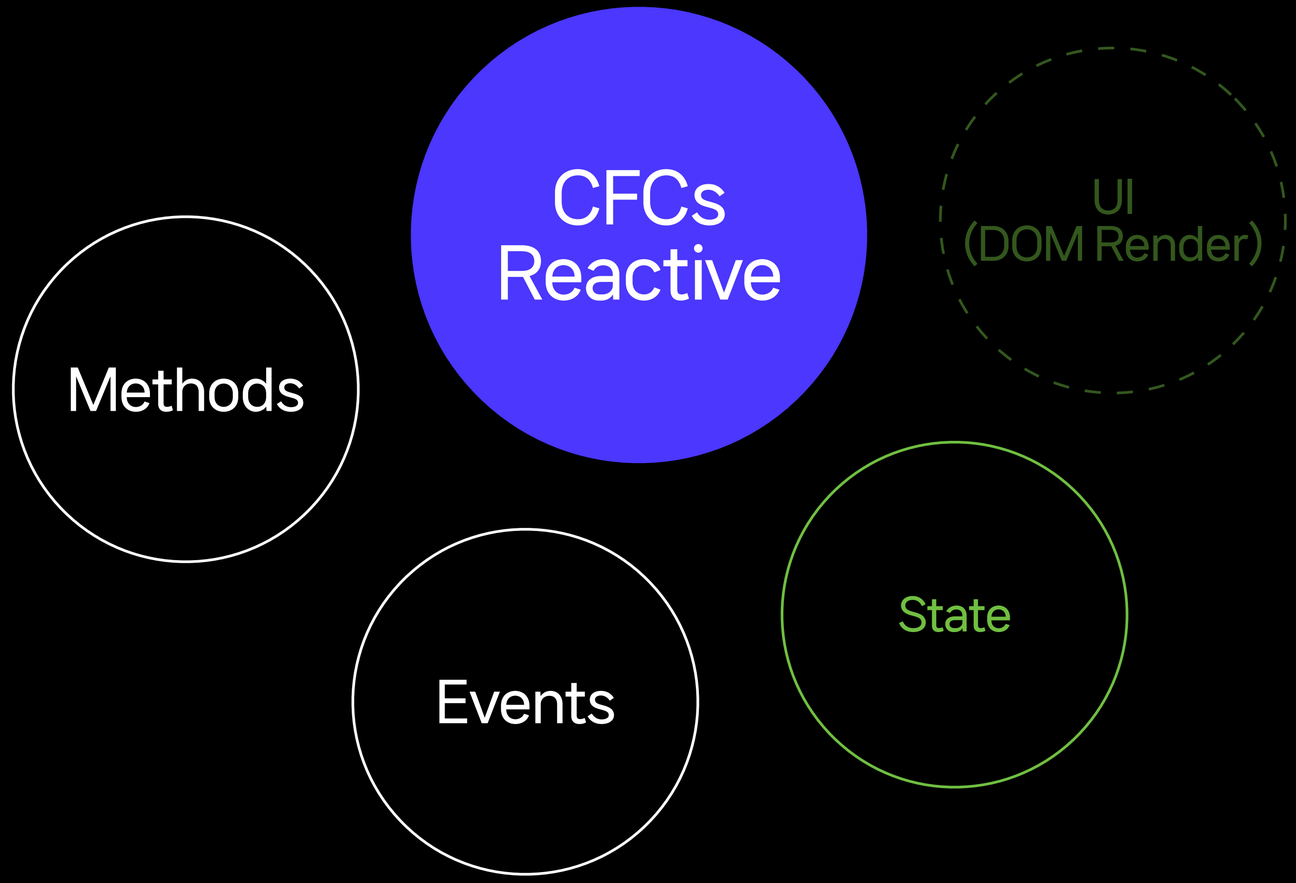
모든 프레임워크 컴포넌트의 mounted, unmounted의 코드는 동일

4. CFCs Reactive

4.1 CFCs Reactive

NAVER
DEVIEW
2023

- CFCs DOM은 돔을 렌더링하는 과정이 있어 UI 컴포넌트에 적합
- CFCs Reactive는 돔 렌더링 대신 상태가 있으며 상태에 따라 반응을 할 수 있는 유틸 컴포넌트에 적합

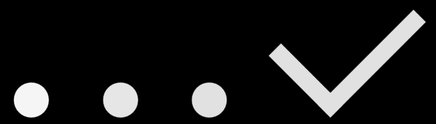


4.2 CFCs Reactive 개발 이유

NAVER
DEVIEW
2023

ImReady: 이미지, 비디오들이 로딩이 됐는지 체크하는 유틸 컴포넌트

- 2019년부터 개발
- InfiniteGrid, Flicking, View360 등에서 사용



ImReady

4.2 CFCs Reactive 개발 이유

NAVER
DEVIEW
2023

CFCs DOM 이후 처음으로 유틸 컴포넌트를 함수 컴포넌트에 지원

- 상태만 반환하는 컴포넌트
- 당시에는 CFCs Reactive에 대한 생각을 하지 못함

```
const {
  isReady,    이미지가 전부 로드가 되었는지 여부
  readyCount, 이미지가 로드된 개수
} = useImReady();
```

ImReady의 Reactive Component 초창기 코드

4.2 CFCs Reactive 개발 이유

NAVER
DEVIEW
2023

CFCs DOM 이후 처음으로 유틸 컴포넌트를 함수 컴포넌트에 지원

- 당시에는 CFCs Reactive에 대한 생각을 하지 못함

```
import { useEffect, useState } from "react";
import ImReady from "@egjs/imready";

export function useImReady() {
    const [isReady, setIsReady] = useState(false);

    useEffect(() => {
        const im = new ImReady(options);
        im.check(checkedElements).on("ready", () => {
            setIsReady(true);
        });
        ...
    }, []);

    return {
        isReady,
    };
}
```

100줄이 넘는 React ImReady 코드

4.2 CFCs Reactive 개발 이유

NAVER
DEVIEW
2023

상태 vs 이벤트

- 상황에 따라 상태가 필요하기도 하고 이벤트 방식도 필요

상태

```
const {
  width,
  height,
} = useWindowSize();
```

VS

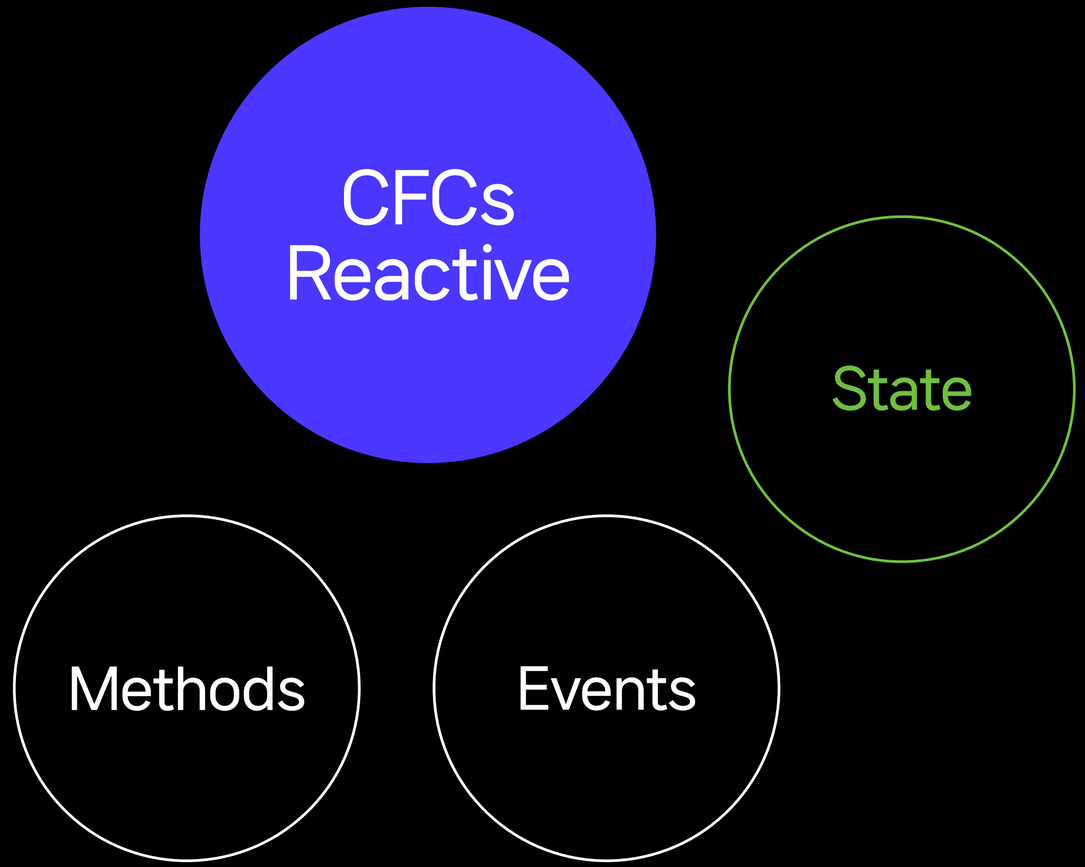
이벤트

```
onResize(({ width }) => {
  console.log(width);
});
```

4.2 CFCs Reactive 개발 이유

NAVER
DEVIEW
2023

- Vanilla에서 사용하던 메서드까지 프레임워크에서 필요
- 상태, 이벤트, 메서드 3가지 요소가 전부 필요



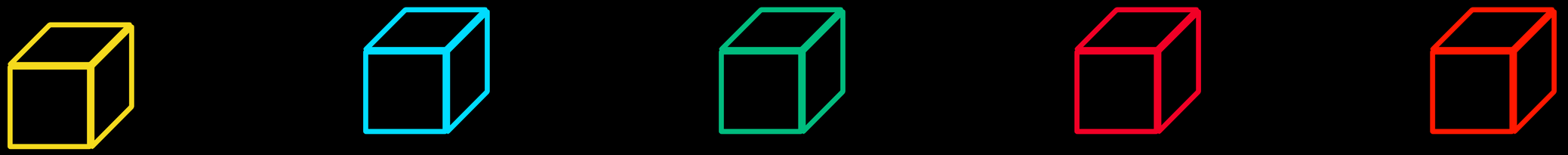
4.2 CFCs Reactive 개발 이유

NAVER
DEVIEW
2023

여러 프레임워크를 지원

- 프레임워크마다 비슷한 코드가 존재하고 크기도 N배
- 기능 추가 / 버그 수정도 N배

X 5



Vanilla Component

React Component

Vue Component

Angular Component

Svelte Component

4.2 CFCs Reactive 개발 이유

NAVER
DEVIEW
2023

프레임워크의 공통점을 확인

- state, mounted, unmounted, result

state

mounted

unmounted

result

```
export function useWindowSize() {
    const [width, setWidth] = useState(0);
    const [height, setHeight] = useState(0);

    useEffect(() => {
        const callback = () => {
            setWidth(window.innerWidth);
            setHeight(window.innerHeight);
        };
        window.addEventListener("resize", callback);

        return () => window.removeEventListener("resize", callback);
    }, []);

    return { width, height };
}
```

4.3 CFCs Reactive Concepts

NAVER
DEVIEW
2023

- 이벤트보다 상태가 더 직관적인 이름
- 2개 이상의 이벤트에서 속성의 변화 가능성
- 이벤트가 아닌 속성값의 변화만 체크

```
window.addEventListener("resize", () => {
  console.log(window.innerWidth);
  console.log(window.innerHeight);
});
```

@cfcs/core

```
subscribe("width", width => {
  console.log(width);
});
subscribe("height", height => {
  console.log(height);
});
```

4.3 CFCs Reactive Concepts

NAVER
DEVIEW
2023

- 상태값은 프레임워크의 상태와 1:1 연결 가능
- 상태가 변하면 업데이트 (재렌더링이 발생)

```
subscribe("width", width => {
  console.log(width);
});
subscribe("height", height => {
  console.log(height);
});
```

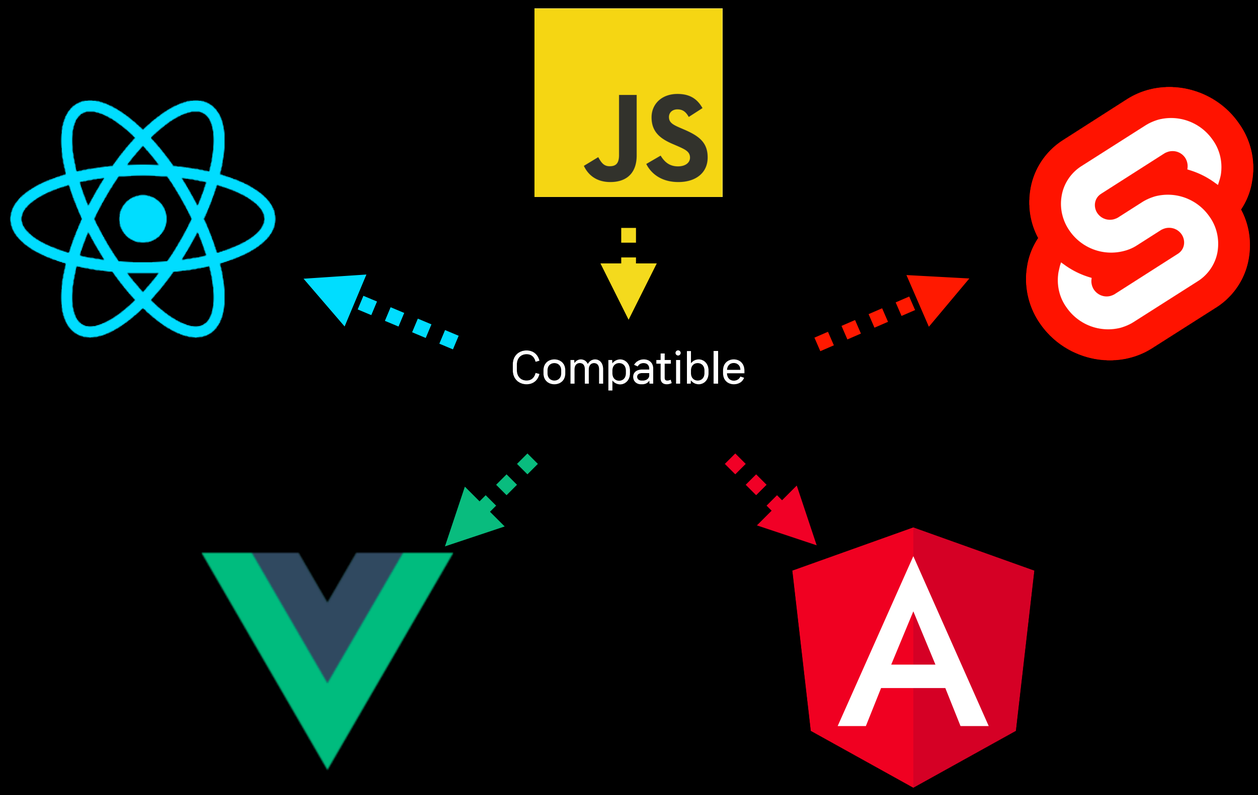
@cfcs/react

```
const {
  width,
  height,
} = useWindowSize();
```

4.3 CFCs Reactive Concepts

NAVER
DEVIEW
2023

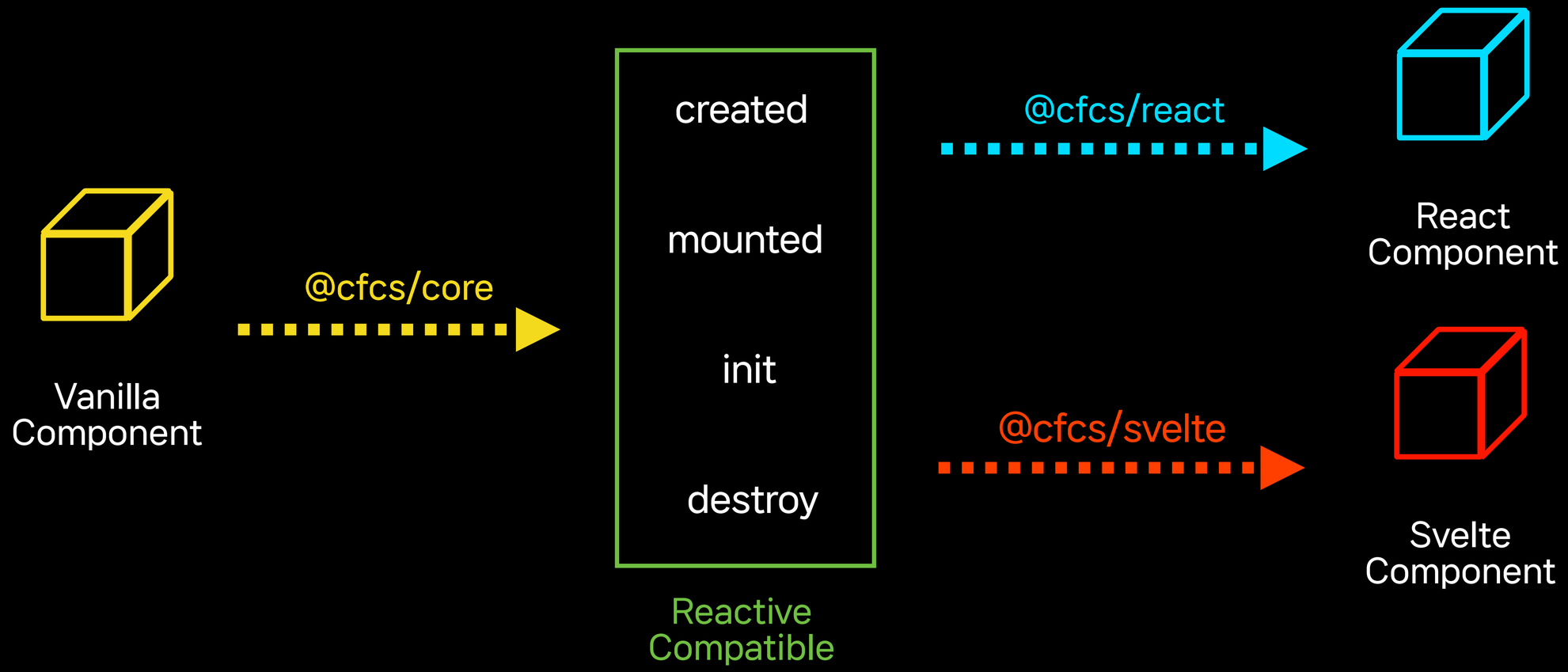
- CFCs의 목표인 공통화처럼 CFCs Reactive도 공통화된 영역 Compatible을 작성
  - 이벤트 등록, 해제 방법
  - 인스턴스 생성, 초기화, 제거 방법



4.3 CFCs Reactive Concepts

NAVER
DEVIEW
2023

1. Vanilla Component를 CFCS Lifecycle에 작성
2. CFCs Lifecycle을 Framework Lifecycle에 연결



5. 모듈로써 공개하는 CFCs Reactive

5.1 모듈로써 공개하는 CFCs Reactive

NAVER
DEVIEW
2023

- 프레임워크와 유사한 CFCs Reactive만의 Lifecycle을 제공
- 공통 코드를 작성함으로써 프레임워크에서 작성하는 코드를 최소화

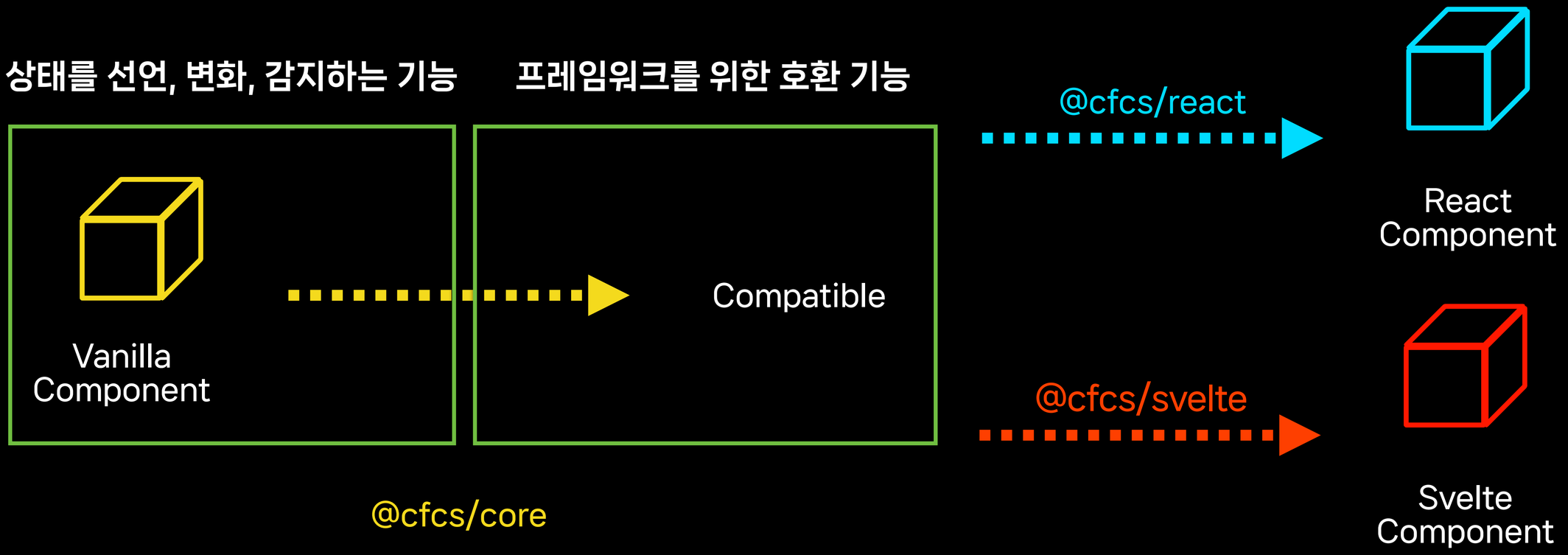
Cross Framework Components

https://github.com/naver/cfcs

5.2 CFCs Reactive의 기능

NAVER
DEVIEW
2023

- 상태를 선언, 변화, 감지하는 기능
- 프레임워크를 위한 호환 기능



5.3 Vanilla에서의 상태

NAVER
DEVIEW
2023

- 상태는 프레임워크의 상태와 매칭시키기 위해 선언(declaration), 감지(subscribe), 변화(change)로 구성
- reactive 함수로 상태를 선언하고 변화된 상태를 감지

```
import { reactive } from "@cfcs/core";
```

declaration

```
const obj = reactive({
  width: 0,
  height: 0,
});
```

subscribe

```
obj.subscribe("width", width => {
   console.log(width);
});
```

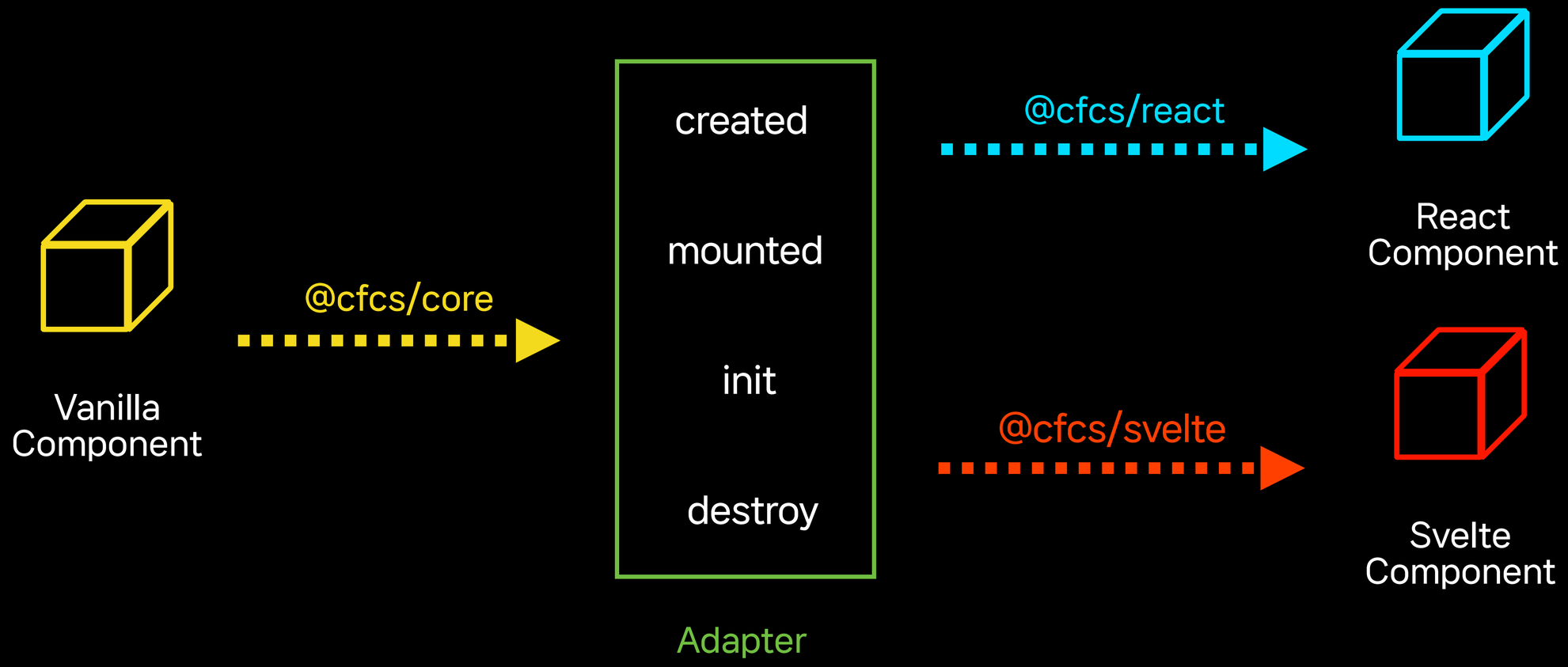
change

```
obj.width = 100;
```

5.4 CFCs Reactive Adapter

NAVER
DEVIEW
2023

1. CFCs의 Vanilla 컴포넌트를 프레임워크에 지원하기 위한 중간 단계의 호환 코드
2. CFCs 라이프사이클과 프레임워크의 라이프사이클 매칭하여 코드가 실행



5.5 CFCs Reactive의 Lifecycle

NAVER
DEVIEW
2023

- CFCs Reactive는 크게 4개의 Lifecycle으로 구성

5.5 CFCs Reactive의 Lifecycle

NAVER
DEVIEW
2023

- CFCs의 Lifecycle의 매칭된 프레임워크들의 상태와 Lifecycle

Core (@cfcs/core)			React (@cfcs/react)	Vue 2 / 3 (@cfcs/vue2,3)	Svelte (@cfcs/svelte)
reactive	>	State >	useState	ref	store (writable, readable)
setup Function	>	created >	useMemo	setup Function	Function
onMounted onInit	>	mounted >	useEffect	onMounted	onMount
onDestroy	>	unmounted >	useEffect return	onUnmounted	onDestroy

5.6 Compatible Adapter 작성

NAVER
DEVIEW
2023

- 함수 형태로 Adapter를 작성
- Framework 처럼 state, mounted, unmounted, result 동일한 구성

state

mounted

unmounted

result

```
export const REACTIVE = ({ onInit, onDestroy }) => {
  const obj = reactive({ width: 0, height: 0 });
  const callback = () => {
    obj.width = window.innerWidth;
    obj.height = window.innerHeight;
  };
  onInit(() => {
    window.addEventListener("resize", callback);
    callback();
  });
  onDestroy(() => {
    window.removeEventListener("resize", callback);
  });
  return obj;
};
```

https://github.com/egjs/deview2023

5.6 Compatible Adapter 작성

NAVER
DEVIEW
2023

- 이벤트, 메서드를 지원하고 싶다면 제공하는 함수를 통해 추가적인 정보를 입력

```
const REACTIVE_ADAPTER = ({ setMethods, setEvents, on }) => {
  ...
  setMethods(["setWidth"]);
  setEvents(["resize"]);

  on((name, callback) => {
    window.addEventListener(name, callback);
    return () => window.removeEventListener(name, callback);
  });
  ...
}
```

https://github.com/egjs/deview2023

5.7 CFCs Reactive React 결과

NAVER
DEVIEW
2023

- React Resize Watcher 코드
- useReactive 함수를 사용하면 프레임워크에서 사용가능한 Reactive Component를 사용

```
import { useReactive } from "@cfcs/react";

export function useResizeWatcher() {
  return useReactive(REACTIVE);
}
```

https://github.com/egjs/deview2023

5.7 CFCs Reactive React 결과

NAVER
DEVIEW
2023

- React Resize Watcher 코드
- React 같은 경우 상태를 사용하지 않는다면 값이 변경되더라도 재렌더링이 발생되지 않게 조치

```
const { width, height } = useResizeWatcher();

<div>{width} x {height}</div>;
```

https://github.com/egjs/deview2023

5.7 CFCs Reactive Vue 결과

NAVER
DEVIEW
2023

- 모듈을 @cfcs/vue3로 변경하면 Vue3 Resize Watcher 코드
- Vue 2의 경우 @cfcs/vue2
- useReactive 함수를 사용하면 프레임워크에서 사용가능한 Reactive Component를 사용

```
import { useReactive } from "@cfcs/vue3";

export function useResizeWatcher() {
  return useReactive(REACTIVE);
}
```

https://github.com/egjs/deview2023

5.7 CFCs Reactive Vue 결과

NAVER
DEVIEW
2023

- 모듈을 @cfcs/vue3로 변경하면 Vue3 Resize Watcher 코드
- Vue 2의 경우 @cfcs/vue2

```
<script setup>
  const { width, height } = useResizeWathcer();
</script>
<template>
  <div>{{width}} x {{height}}</div>
</template>
```

https://github.com/egjs/deview2023

5.7 CFCs Reactive Svelte 결과

NAVER
DEVIEW
2023

- 모듈을 @cfcs/svelte로 변경하면 Svelte Resize Watcher 코드
- useReactive 함수를 사용하면 프레임워크에서 사용가능한 Reactive Component를 사용

```
import { useReactive } from "@cfcs/svelte";

export function useResizeWatcher() {
  return useReactive(REACTIVE);
}
```

https://github.com/egjs/deview2023

5.7 CFCs Reactive Svelte 결과

NAVER
DEVIEW
2023

- 모듈을 @cfcs/svelte로 변경하면 Svelte Resize Watcher 코드

```
<script>
  const { width, height } = useResizeWatcher();
</script>
<div>${$width} x ${$height}</div>
```

https://github.com/egjs/deview2023

5.7 TypeScript 지원

NAVER
DEVIEW
2023

- Adapter에 타입을 부여하면 모든 프레임워크에서 타입 지원
  - Instance, State, Methods, Props, Events 타입을 입력

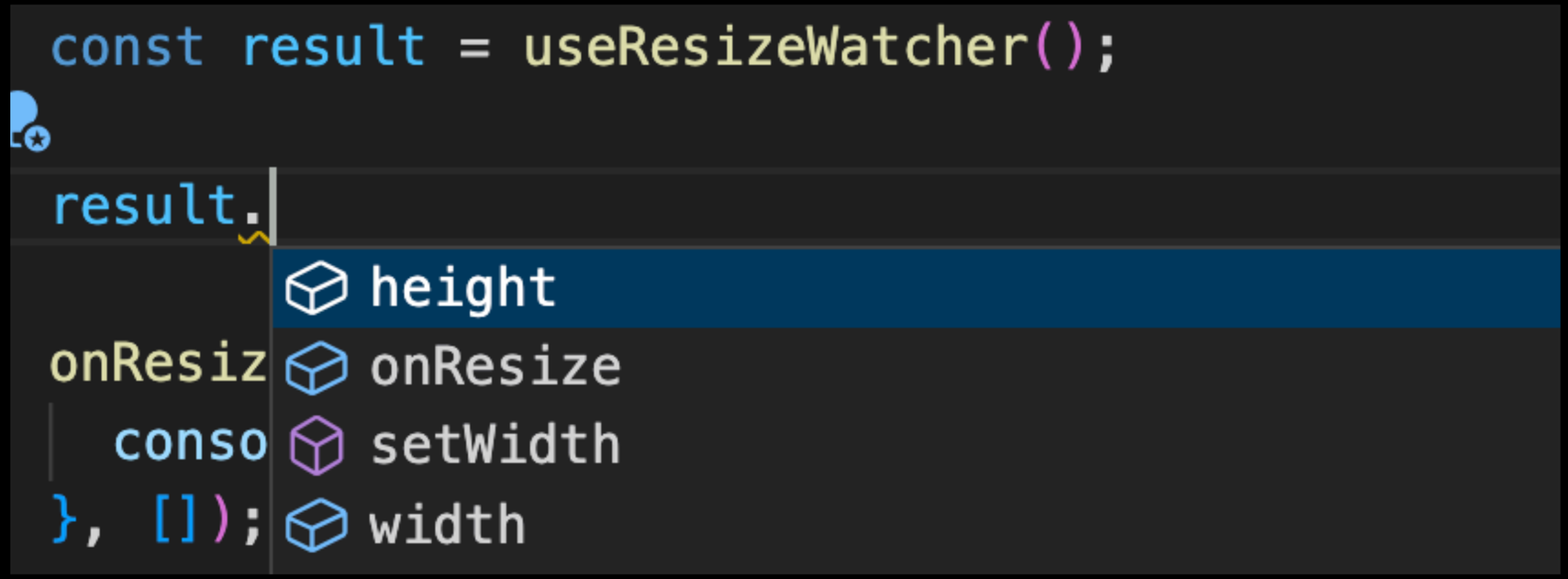
```
import { ReactiveSetupAdapter } from "@cfcs/core";

export const REACTIVE: ReactiveSetupAdapter<
  InstanceType,
  StateType,
  MethodsType,
  PropsType,
  EventsType,
> = () => { .... };
```

https://github.com/egjs/deview2023

5.7 TypeScript 지원

NAVER
DEVIEW
2023

- Adapter에 타입을 부여하면 모든 프레임워크에서 타입 지원



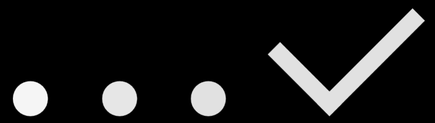
https://github.com/egjs/deview2023

5.8 CFCs Reactive가 적용된 컴포넌트

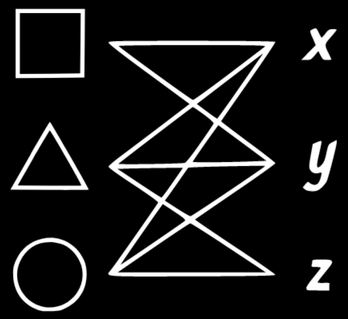
NAVER
DEVIEW
2023

- ImReady: 이미지/비디오 모두 로드됐는지 체크하는 유틸 컴포넌트
  - 상태: 이미지 로드 여부 관련
- Conveyer: Native Scroll에 Drag가 가능하게 해주는 유틸 컴포넌트
  - 상태: scroll 위치 관련
- Axes: Drag, Wheel, Key, Gesture에 Animation을 추가한 유틸 컴포넌트
  - 상태: 축 또는 위치 관련

Conveyer



ImReady



Axes

5.9 Conveyer의 CFCs Adapter

NAVER
DEVIEW
2023

- Conveyer의 CFCs Adapter 코드

state (instance)

mounted

unmounted

result

```
({ getProps, onInit, on, onDestroy }) => {
  const { container, props } = getProps();
  const conveyer = new Conveyer(container, { ...props, autoInit: false });

  on((name, callback) => {
    conveyer.on(name, callback);
    return () => conveyer.off(name, callback);
  });
  onInit(() => conveyer.init());

  onDestroy(() => conveyer.destroy());

  return conveyer;
}
```

https://github.com/naver/egjs-conveyer

5.9 Conveyer의 Vanilla 코드

NAVER
DEVIEW
2023

- subscribe을 통해 상태 감지
- on을 통해 이벤트 확인

```
conveyer.subscribe("isReachStart", isReachStart => {
  console.log(isReachStart);
});

conveyer.on("reachStart", () => {
    console.log("reachStart");
});
```

https://github.com/naver/egjs-conveyer

5.9 Conveyer의 React 코드

NAVER
DEVIEW
2023

- isReachStart 상태가 변화하면 재렌더링이 발생
- onReachStart를 사용하여 이벤트 확인

```
const {
  isReachStart,
  onReachStart,
} = useConveyer(...);

onReachStart(() => {
  console.log("reach start");
}, []);

console.log(isReachStart);
```

https://github.com/naver/egjs-conveyer

5.9 Conveyer의 Vue 코드

NAVER
DEVIEW
2023

- isReachStart 상태가 변화하면 재렌더링이 발생
- onReachStart를 사용하여 이벤트 확인

```
const {
  isReachStart,
  onReachStart,
} = useConveyer(...);

onReachStart(() => {
  console.log("reach start");
});

watchEffect(() => console.log(isReachStart.value));
```

https://github.com/naver/egjs-conveyer

5.9 Conveyer의 Svelte 코드

NAVER
DEVIEW
2023

- isReachStart 상태가 변화하면 재렌더링이 발생
- onReachStart를 사용하여 이벤트 확인

```
const {
  isReachStart,
  onReachStart,
} = useConveyer(...);

onReachStart(() => {
  console.log("reach start");
});

$: console.log($isReachStart);
```

https://github.com/naver/egjs-conveyer

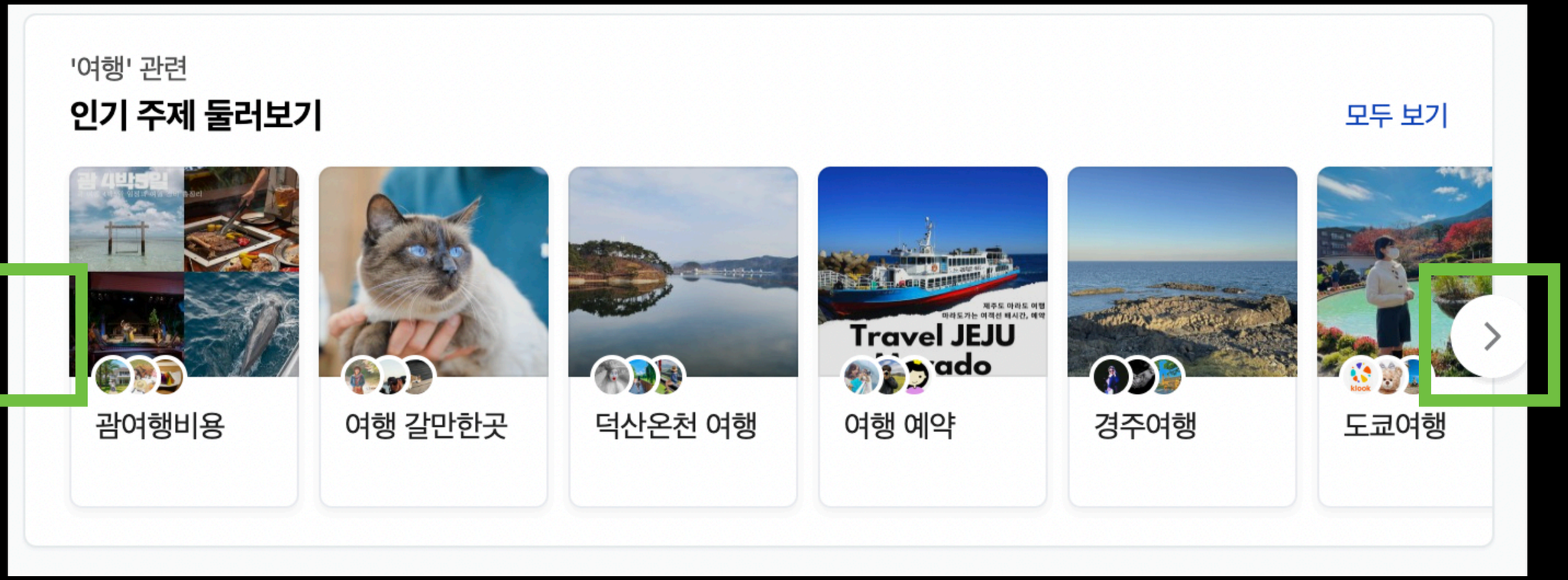
5.9 Conveyer의 예시

NAVER
DEVIEW
2023

- 상태에 따른 UI 변화
- 스크롤이 왼쪽에 닿으면 isReachStart = true

isReachStart = true

isReachEnd = false



PC 네이버 여행 검색

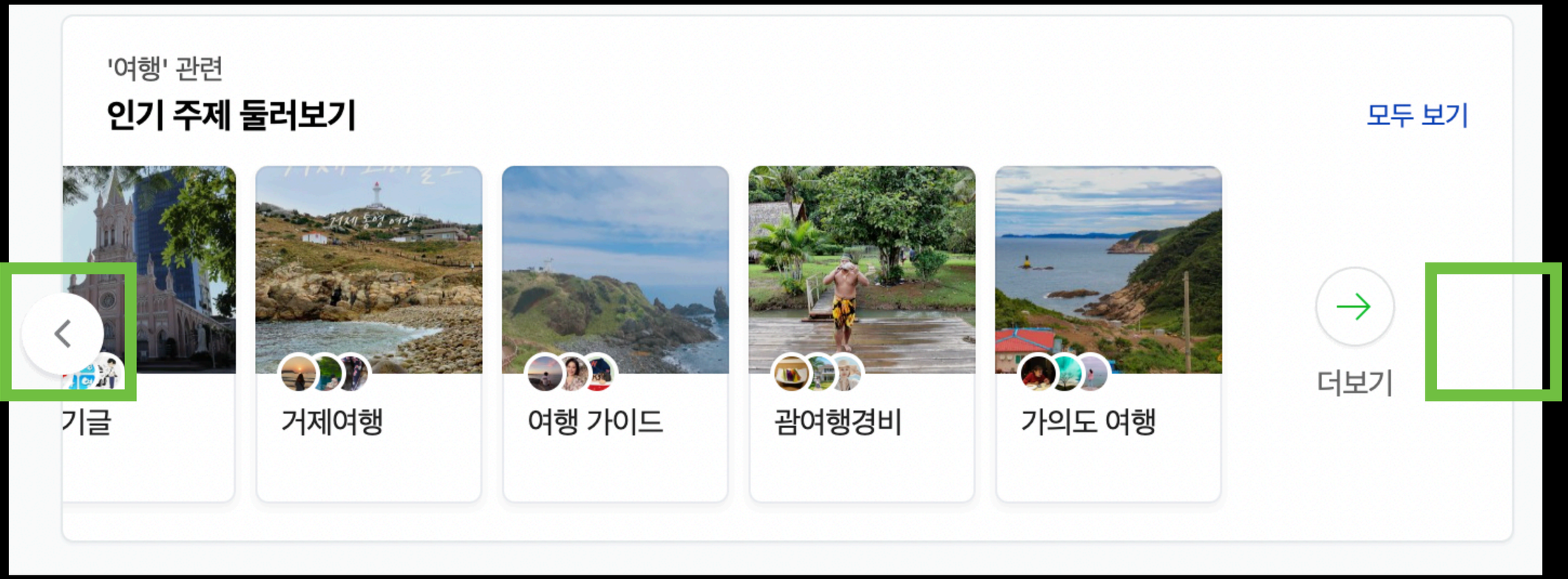
5.9 Conveyer의 예시

NAVER
DEVIEW
2023

- 상태에 따른 UI 변화
- 스크롤이 오른쪽에 닿으면 isReachEnd = true

isReachStart = false

isReachEnd = true



네이버 여행 검색

5.10 CFCs 장단점

NAVER
DEVIEW
2023

- 장점

- Core(Vanilla)를 제외한 프레임워크 코드가 매우 적고 간결
- 대부분 문제는 공통 코드 또는 Core 컴포넌트의 문제
  - Core만 기능 추가 / 수정하면 모든 프레임워크 컴포넌트에 기능 추가 / 수정 반영
- 일부 프레임워크에서의 문제라면 대부분 CFCs 모듈의 문제
  - CFCs 사용한 다른 컴포넌트도 잠재적 오류 수정 반영

- 단점

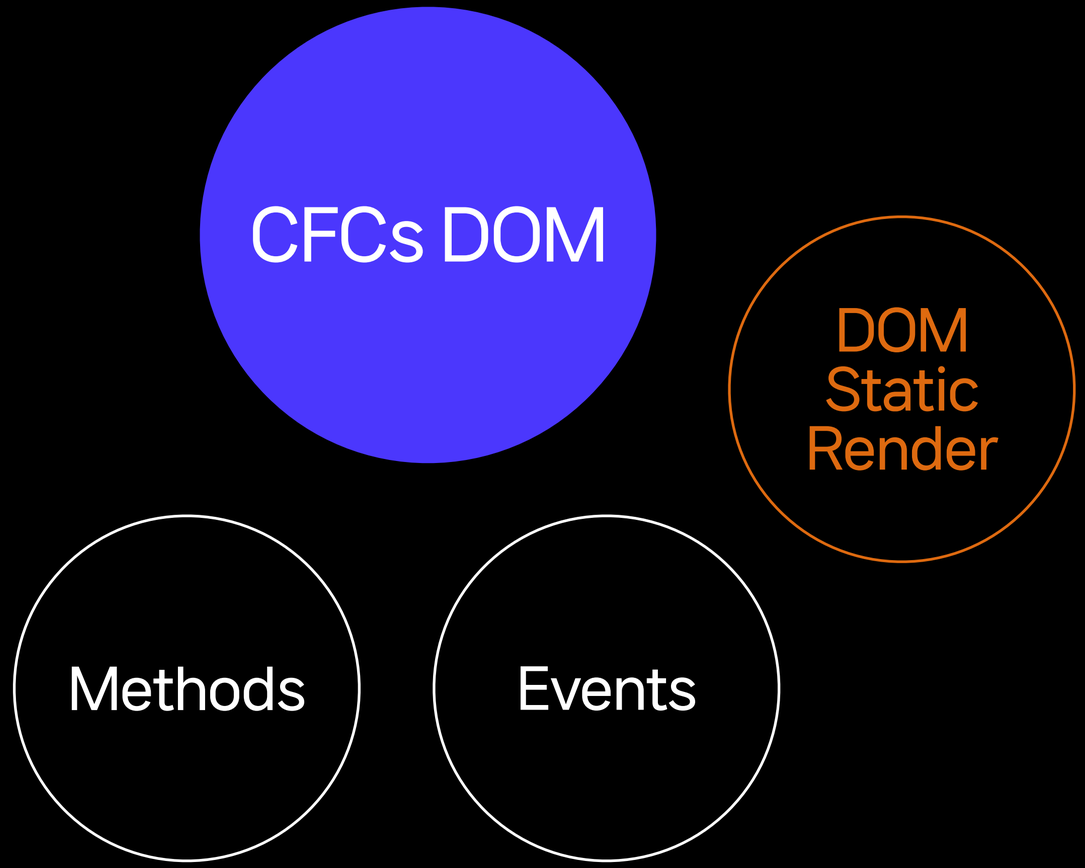
- CFCs 학습 비용이 발생
- 단일 프레임워크 지원이라면 비효율
- CFCs 모듈의 코드가 추가됨으로써 용량이 어느 정도 증가

6. Next CFCs

6.1 CFCs Static DOM

NAVER
DEVIEW
2023

CFCs Static DOM은 DOM 렌더링을 1번만 하는 형태의 컴포넌트에 적합

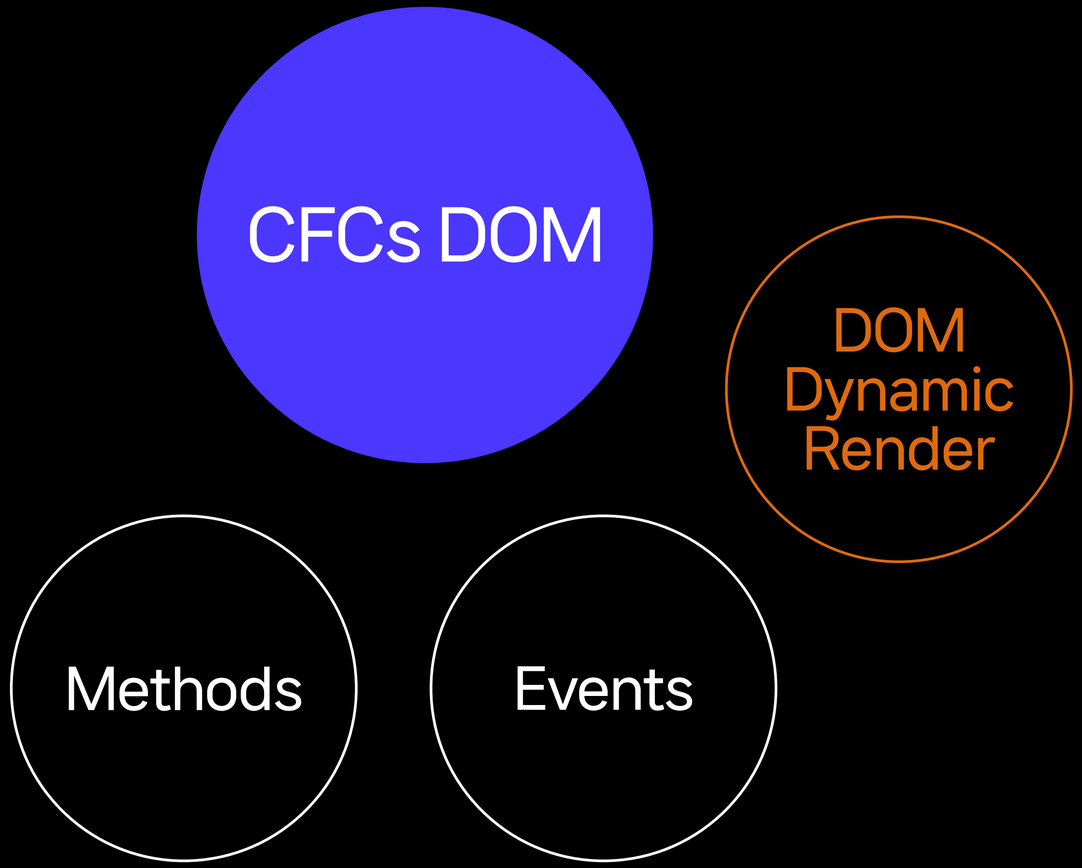


6.2 CFCs Dynamic DOM

NAVER
DEVIEW
2023

CFCs Dynamic DOM은 특정 조건에 의하여 DOM 렌더링을 1번 이상하는 형태의 컴포넌트에 적합

- Infinite, Recycle, Clone 등

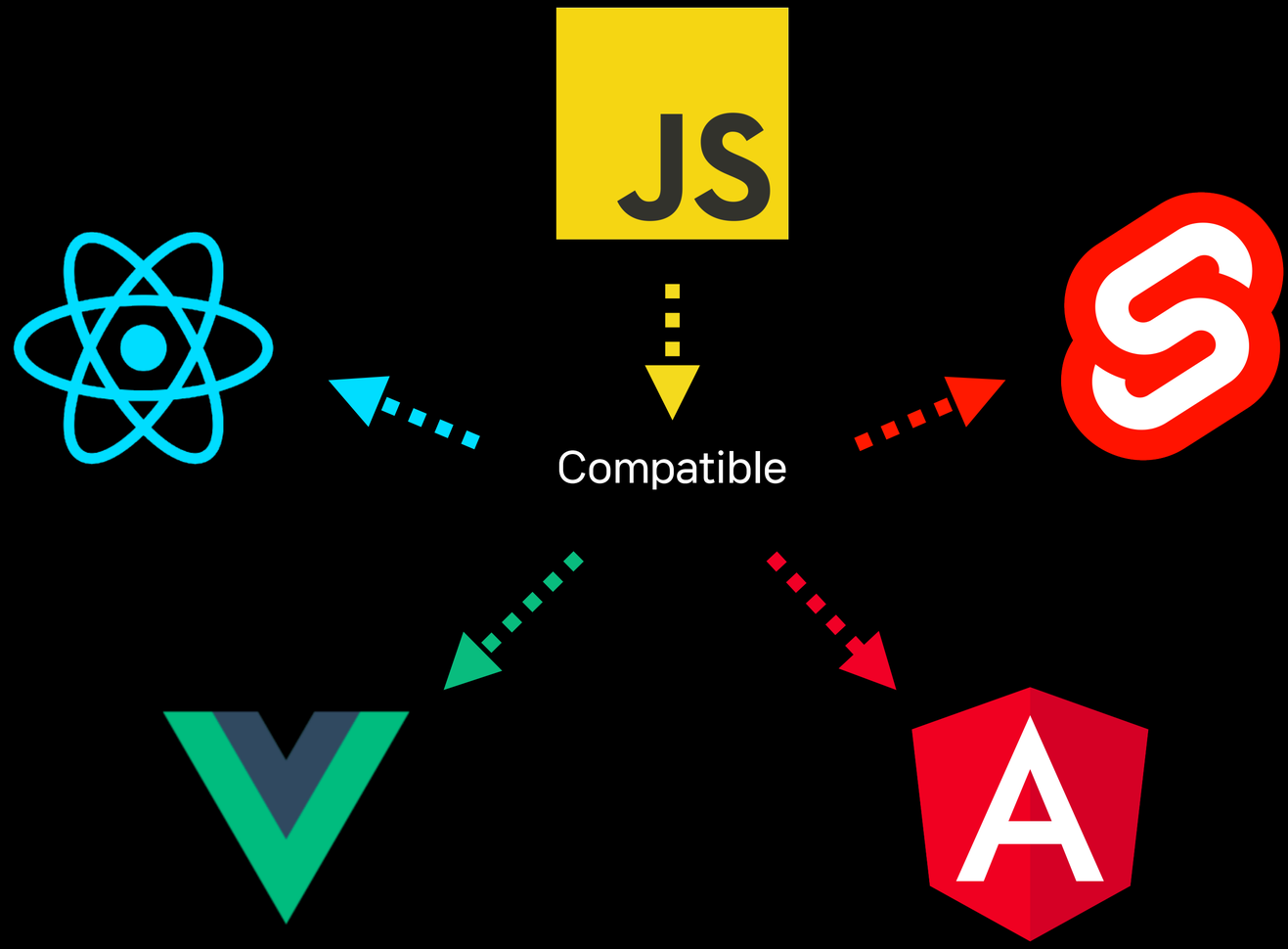


6.3 CFCs Warp Concept

NAVER
DEVIEW
2023

CFCs는 Compatible을 통해 하나의 공통 코드로 변환하여 여러 프레임워크를 지원

CFCs Warp는 프레임워크 컴포넌트를 다른 프레임워크에 지원



6.3 CFCs Warp Concept

NAVER
DEVIEW
2023

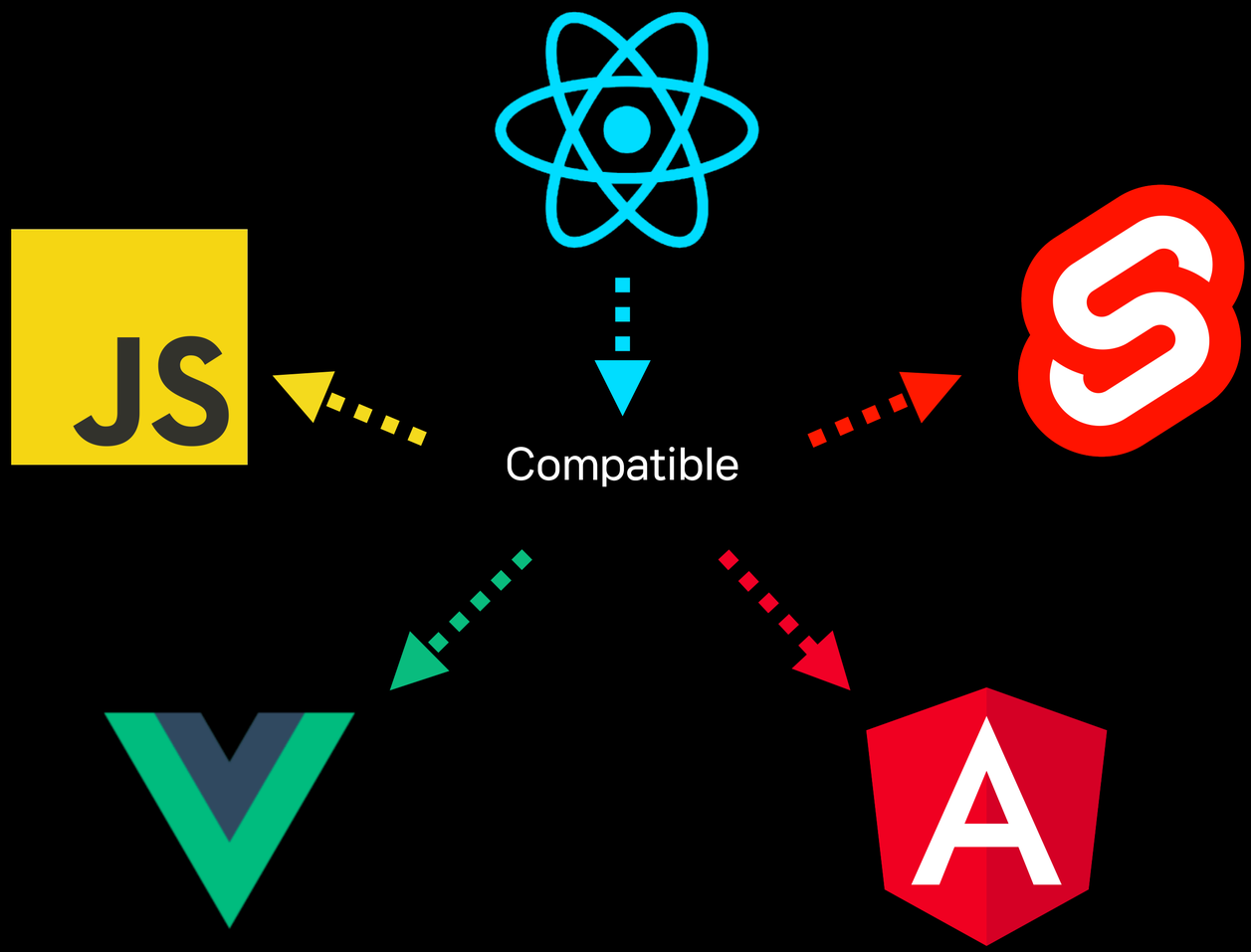
Preact는 가벼운 React

- React의 컴포넌트를 Preact에서 사용하기 위해서는 compatible 모듈이 필요
- preact/compat은 React 모듈을 1:1 매칭이 된 Preact 모듈로 전환

6.3 CFCs Warp Concept

NAVER
DEVIEW
2023

- 프레임워크를 위한 Compatible에는 기능이 1:1 매칭 필요
- React Component를 다른 프레임워크에서 사용



CFCs: https://github.com/naver/cfcs

CFCs 사이트: https://naver.github.io/cfcs

egjs 컴포넌트 사이트: https://naver.github.io/egjs

DEVIEW 2023 예제: https://github.com/egjs/deview2023

Q & A

Thank You