

# React, Angular, Vue를 한 번에 지원하기 위한 설계 (Cross Framework Component)

최연규  
FE-Platform

**NAVER**

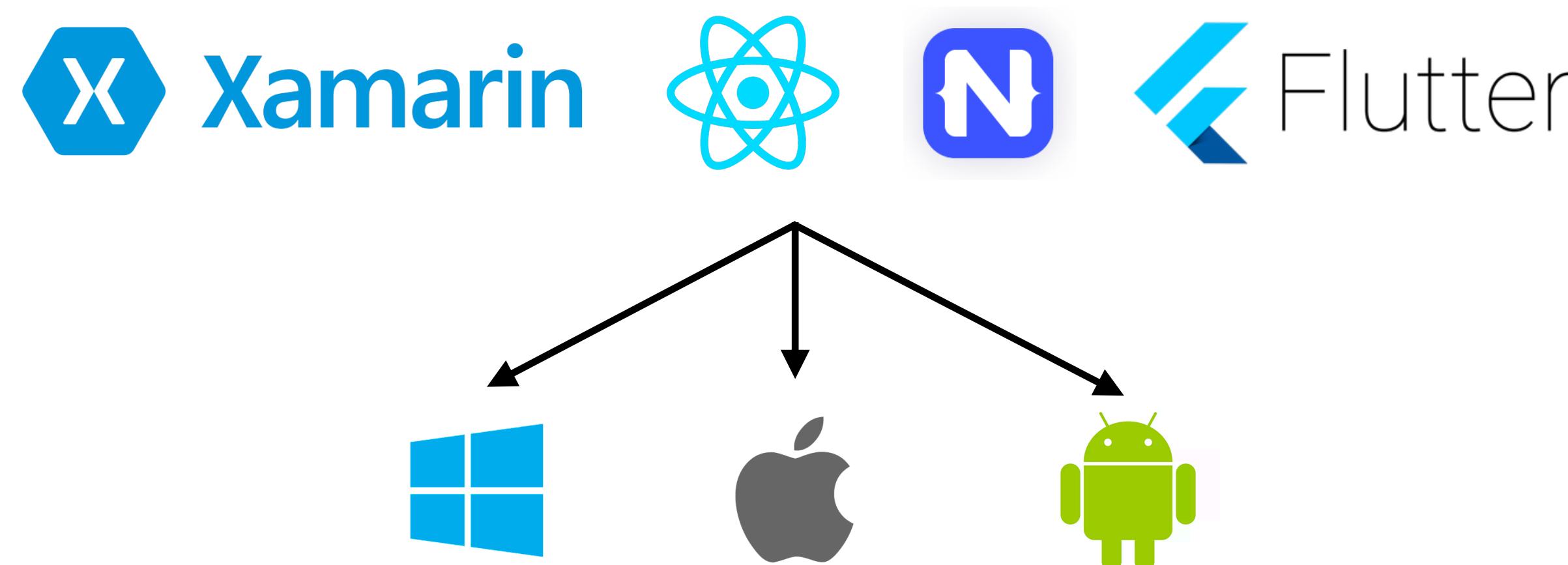
# CONTENTS

1. Cross Framework Component(CFC)?
2. 기존 바닐라 컴포넌트를 프레임워크에 적용한 사례와 문제점
3. 프레임워크 동 렌더링 원리(DOM Diff)
4. Cross Framework Component(CFC) 원리
5. Cross Framework Component(CFC) 적용 방법
6. 누구나 쉽게 만들 수 있는 Infinite Scroll에 CFC 적용해보기
7. egjs에서 CFC를 적용한 사례

# 1. Cross Framework Component

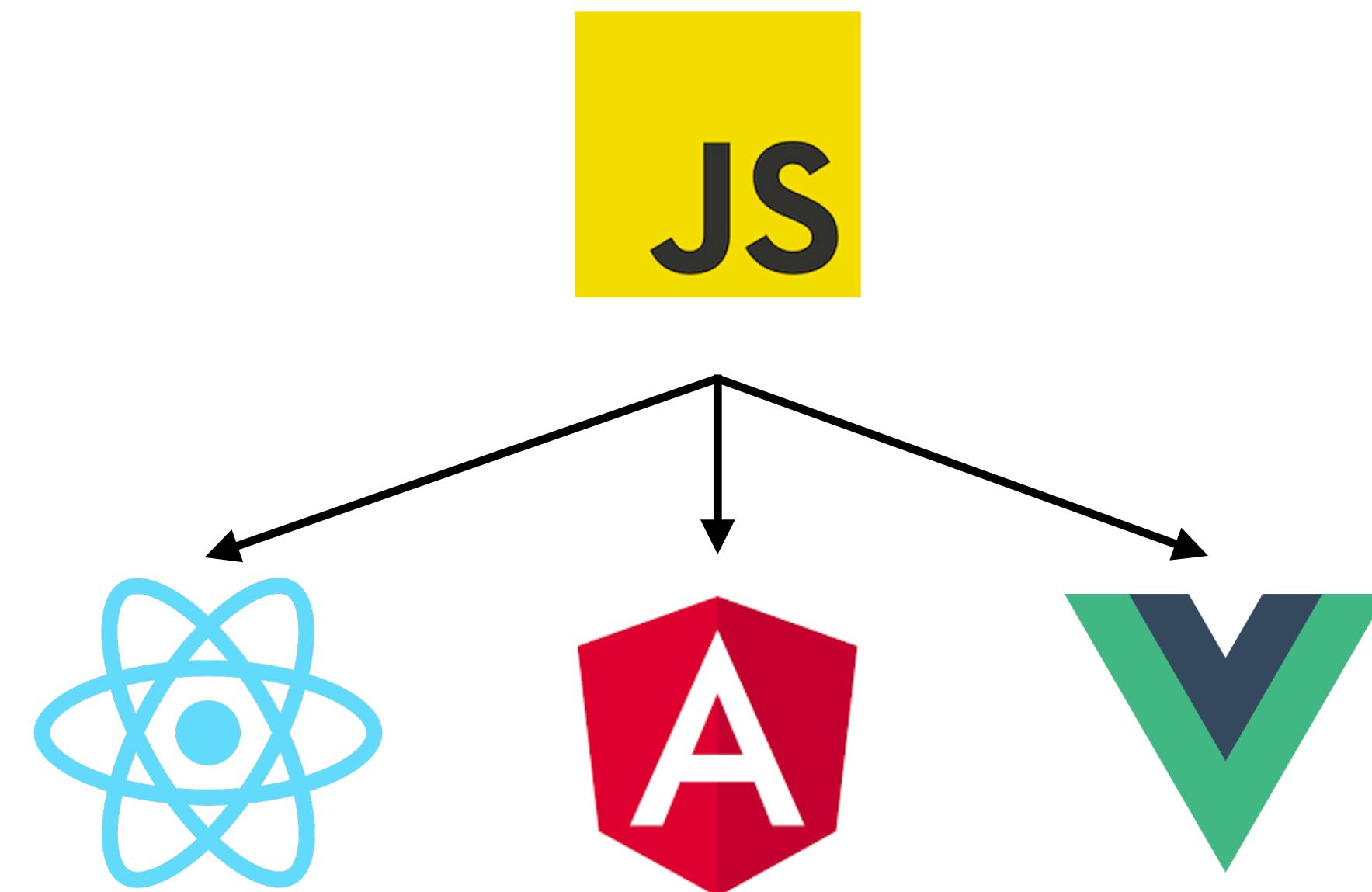
# Cross Platform?

크로스 플랫폼(Cross Platform)은 하나의 소스코드를 다양한 플랫폼(운영체제, 기기)에서 사용할 수 있는 뜻을 가진 용어



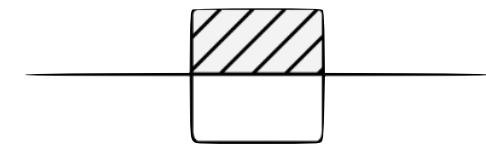
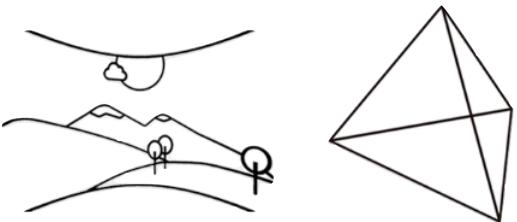
# Cross Framework Component

하나의 공통 모듈 기반으로 다양한 프레임워크를 지원하기 위해  
효과적인 구조를 가진 컴포넌트



# egjs의 바닐라 컴포넌트들

< Flicking >

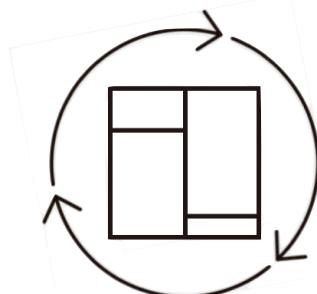


view360

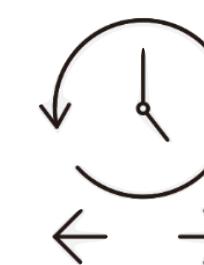


egjs

Visible



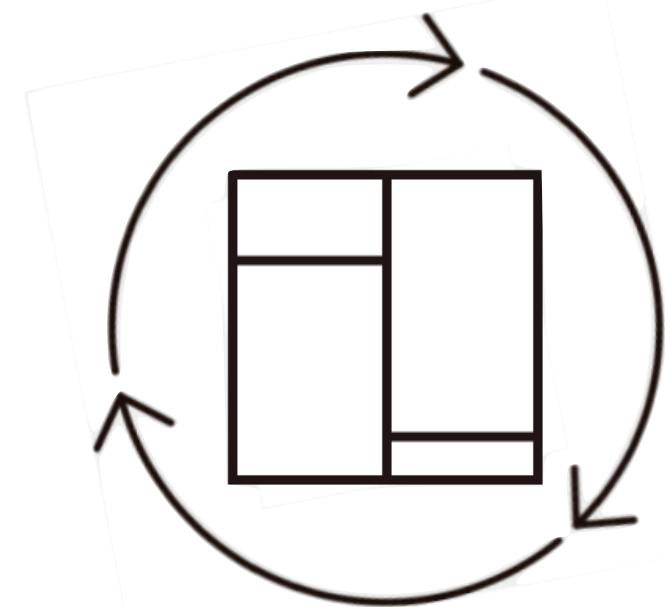
InfiniteGrid



Persist

## 2. 기존 바닐라 컴포넌트를 프레임워크에 적용한 사례와 문제점

## 2. 프레임워크에 적용한 사례와 문제점



< Flicking >  
...

**InfiniteGrid**

문항	O / X
1. DOM을 추가/삭제하는 메서드가 있습니까?	O
2. DOM의 순서가 바뀔 수 있습니까?	O

## 2. 프레임워크에 적용한 사례와 문제점

	단순하게 래핑하는 방법	별도로 만드는 방법
대표 모듈	react-flicking	react-infinitegrid
특징	단순하고 DOM의 변화가 없는 가장 많이 사용하는 방법	해당 프레임워크에 최적화된 컴포넌트를 만드는 방법
장점	쉽고 간단하다.	원하는 기능들이 동작을 할 수 있다.
단점	DOM의 추가/삭제/이동이 어렵다.	유지보수가 매우 어렵다. (처음 만들 때 많이 사용하는 방법)

# 2.1 단순하게 래핑하는 방법의 문제점

인스턴스만 초기화한 방법인 react-flicking 0.0.X

```
public componentDidMount() {
  this.flicking = new NativeFlicking(
    findDOMNode(this) as HTMLElement,
    this.options,
  );
}
```

# 2.1 단순하게 래핑하는 방법의 문제점

내부 변수 접근해서 동적 추가/삭제를 지원한 react-flicking 0.0.X

```
const flicking = (this.flicking as any);
const panel = flicking.conf.panel;
panel.$list = [].slice.call(flicking.$container.children);
flicking._arrangePanels();
flicking._movePanelPosition();
```

내부 변수 접근

# 2.2 별도로 만드는 방법의 문제점

React 전용 컴포넌트로 새로 만든 react-infinitegrid 1.x.x

Vanilla

```
this._items = new ItemManager();
this._renderer = new DOMRenderer(viewer, {
  container: isOverflowScroll ? this._container : null,
  isEqualSize,
  isConstantSize,
  horizontal,
});
this._watcher = new Watcher(
  this._renderer.view,
  {
    isOverflowScroll,
    horizontal,
    container: this._renderer.container,
    resize: () => setTimeout(() => { this.setState({layout: true}); },),
    check: param => this._onCheck(param),
  });
this._infinite = new Infinite(this._items, {
  horizontal,
  threshold,
  useRecycle: true,
  append: param => this._requestAppend(param),
  prepend: param => this._requestPrepend(param),
  recycle: param => this._recycle(param),
});
this._manager = new LayoutManager(this._items, this._renderer, {
  isEqualSize,
  isConstantSize,
});
this._manager.setLayout(this._layout);
this._updateLayout();
this._setSize(this._renderer.getViewportSize());
this._updateGroups();
this._watcher.setScrollPos();
```

React

```
this._items = new ItemManager();
this._renderer = new DOMRenderer(viewer, {
  container: isOverflowScroll ? this._container : null,
  isEqualSize,
  isConstantSize,
  horizontal,
});
this._watcher = new Watcher(
  this._renderer.view,
  {
    isOverflowScroll,
    horizontal,
    container: this._renderer.container,
    resize: () => setTimeout(() => { this.setState({layout: true}); },),
    check: param => this._onCheck(param),
  });
this._infinite = new Infinite(this._items, {
  horizontal,
  threshold,
  useRecycle: true,
  append: param => this._requestAppend(param),
  prepend: param => this._requestPrepend(param),
  recycle: param => this._recycle(param),
});
this._manager = new LayoutManager(this._items, this._renderer, {
  isEqualSize,
  isConstantSize,
});
this._manager.setLayout(this._layout);
this._updateLayout();
this._setSize(this._renderer.getViewportSize());
this._updateGroups();
this._watcher.setScrollPos();
```

비슷한 코드 X 2

# 2.2 별도로 만드는 방법의 문제점

다른 프레임워크에도 지원을 하게 된다면?

Vanilla

```
this._items = new ItemManager();
this._renderer = new DOMRenderer(viewer, {
  container: isOverflowScroll ? this._container : null,
  isEqualSize,
  isConstantSize,
  horizontal,
});
this._watcher = new Watcher(
  this._renderer.view,
  {
    isOverflowScroll,
    horizontal,
    container: this._renderer.container,
    resize: () => setTimeout(() => { this.setState({layout: true}); }, ),
    check: param => this._onCheck(param),
  });
this._infinite = new Infinite(this._items, {
  horizontal,
  threshold,
  useRecycle: true,
  append: param => this._requestAppend(param),
  prepend: param => this._requestPrepend(param),
  recycle: param => this._recycle(param),
});
this._manager = new LayoutManager(this._items, this._renderer, {
  isEqualSize,
  isConstantSize,
});

this._manager.setLayout(this._layout);
this._updateLayout();
this._setSize(this._renderer.getViewportSize());
this._updateGroups();
this._watcher.setScrollPos();
```

React

```
this._items = new ItemManager();
this._renderer = new DOMRenderer(viewer, {
  container: isOverflowScroll ? this._container : null,
  isEqualSize,
  isConstantSize,
  horizontal,
});
this._watcher = new Watcher(
  this._renderer.view,
  {
    isOverflowScroll,
    horizontal,
    container: this._renderer.container,
    resize: () => setTimeout(() => { this.setState({layout: true}); }, ),
    check: param => this._onCheck(param),
  });
this._infinite = new Infinite(this._items, {
  horizontal,
  threshold,
  useRecycle: true,
  append: param => this._requestAppend(param),
  prepend: param => this._requestPrepend(param),
  recycle: param => this._recycle(param),
});
this._manager = new LayoutManager(this._items, this._renderer, {
  isEqualSize,
  isConstantSize,
});

this._manager.setLayout(this._layout);
this._updateLayout();
this._setSize(this._renderer.getViewportSize());
this._updateGroups();
this._watcher.setScrollPos();
```

Angular

```
this._items = new ItemManager();
this._renderer = new DOMRenderer(viewer, {
  container: isOverflowScroll ? this._container : null,
  isEqualSize,
  isConstantSize,
  horizontal,
});
this._watcher = new Watcher(
  this._renderer.view,
  {
    isOverflowScroll,
    horizontal,
    container: this._renderer.container,
    resize: () => setTimeout(() => { this.setState({layout: true}); }, ),
    check: param => this._onCheck(param),
  });
this._infinite = new Infinite(this._items, {
  horizontal,
  threshold,
  useRecycle: true,
  append: param => this._requestAppend(param),
  prepend: param => this._requestPrepend(param),
  recycle: param => this._recycle(param),
});
this._manager = new LayoutManager(this._items, this._renderer, {
  isEqualSize,
  isConstantSize,
});

this._manager.setLayout(this._layout);
this._updateLayout();
this._setSize(this._renderer.getViewportSize());
this._updateGroups();
this._watcher.setScrollPos();
```

Vue

```
this._items = new ItemManager();
this._renderer = new DOMRenderer(viewer, {
  container: isOverflowScroll ? this._container : null,
  isEqualSize,
  isConstantSize,
  horizontal,
});
this._watcher = new Watcher(
  this._renderer.view,
  {
    isOverflowScroll,
    horizontal,
    container: this._renderer.container,
    resize: () => setTimeout(() => { this.setState({layout: true}); }, ),
    check: param => this._onCheck(param),
  });
this._infinite = new Infinite(this._items, {
  horizontal,
  threshold,
  useRecycle: true,
  append: param => this._requestAppend(param),
  prepend: param => this._requestPrepend(param),
  recycle: param => this._recycle(param),
});
this._manager = new LayoutManager(this._items, this._renderer, {
  isEqualSize,
  isConstantSize,
});

this._manager.setLayout(this._layout);
this._updateLayout();
this._setSize(this._renderer.getViewportSize());
this._updateGroups();
this._watcher.setScrollPos();
```

비슷한 코드 X 4

## 2.3 고민과 해결

DEVIEW  
2019

모든 문제가 다 DOM

# 3. 프레임워크 동 렌더링 원리 (key & DOM Diff)

# 3.1 프레임워크별 사용법

## Angular Template

```
<ngx-flicking [options]="{ gap: 10 }" (move)="...">
  <ng-template>
    <div class="panel"></div>
    <div class="panel"></div>
    <div class="panel"></div>
  </ng-template>
</ngx-flicking>
```

## React JSX

```
<Flicking gap={10} onMove={...}>
  <div className="panel"></div>
  <div className="panel"></div>
  <div className="panel"></div>
</Flicking>
```

## Vue Template

```
<flicking :options="{ gap: 10 }" @move="...">
  <div class="panel"></div>
  <div class="panel"></div>
  <div class="panel"></div>
</flicking>
```

## 3.2 프레임워크 렌더링 원리 - key

```
<Flicking>
  <div className="item" key={1}>1</div>
  <div className="item" key={2}>2</div>
  <div className="item" key={3}>3</div>
  <div className="item" key={4}>4</div>
  <div className="item" key={5}>5</div>
  <div className="item" key={6}>6</div>
</Flicking>
```

## 3.2 프레임워크 렌더링 원리 - key

React에서 key를 미설정시 뜨는 경고 메시지

- ✖ ►Warning: Each child in an array or iterator should have a unique "key" prop.

```
<Flicking>
  {[1, 2, 3, 4, 5].map( i => (
    <div className="item">{i}</div>
  )))
</Flicking>
```

# 3.3 프레임워크 렌더링 원리 - DOM Diff

1, 2, 3, 4, 5, 6

```
<Flicking>
  <div className="item" key={1}>1</div>
  <div className="item" key={2}>2</div>
  <div className="item" key={3}>3</div>
  <div className="item" key={4}>4</div>
  <div className="item" key={5}>5</div>
  <div className="item" key={6}>6</div>
</Flicking>
```

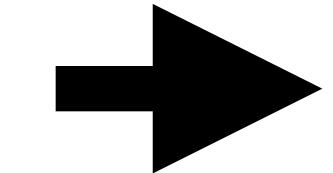
1, 2, 6, 5, 3

```
<Flicking>
  <div className="item" key={1}>1</div>
  <div className="item" key={2}>2</div>
  <div className="item" key={6}>6</div>
  <div className="item" key={5}>5</div>
  <div className="item" key={3}>3</div>
</Flicking>
```

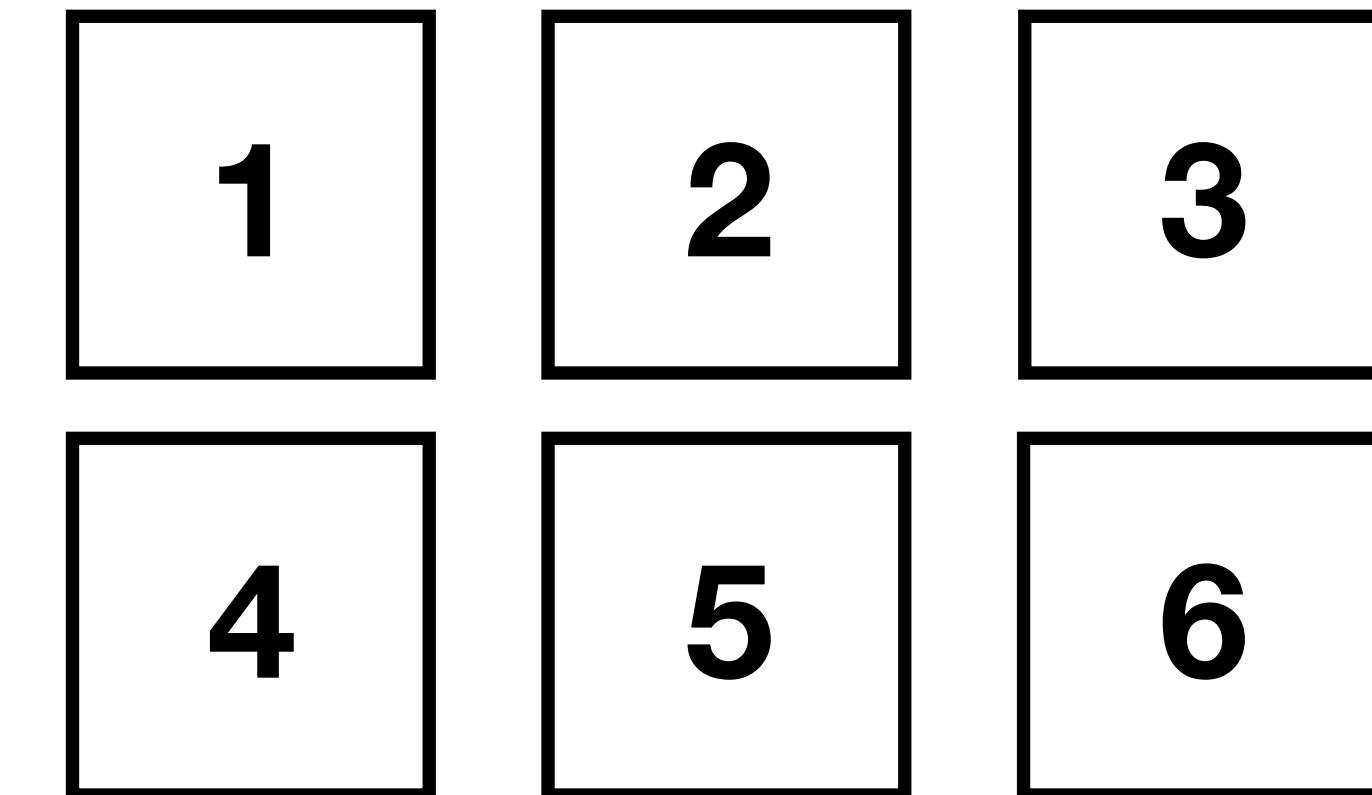
# 3.3 프레임워크 렌더링 원리 - DOM Diff

## Framework

```
<Flicking>
  <div className="item" key={1}>1</div>
  <div className="item" key={2}>2</div>
  <div className="item" key={3}>3</div>
  <div className="item" key={4}>4</div>
  <div className="item" key={5}>5</div>
  <div className="item" key={6}>6</div>
</Flicking>
```



## DOM

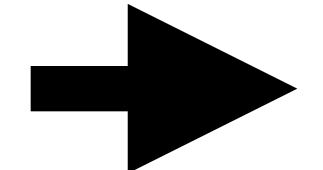


# 3.3 프레임워크 렌더링 원리 - DOM Diff

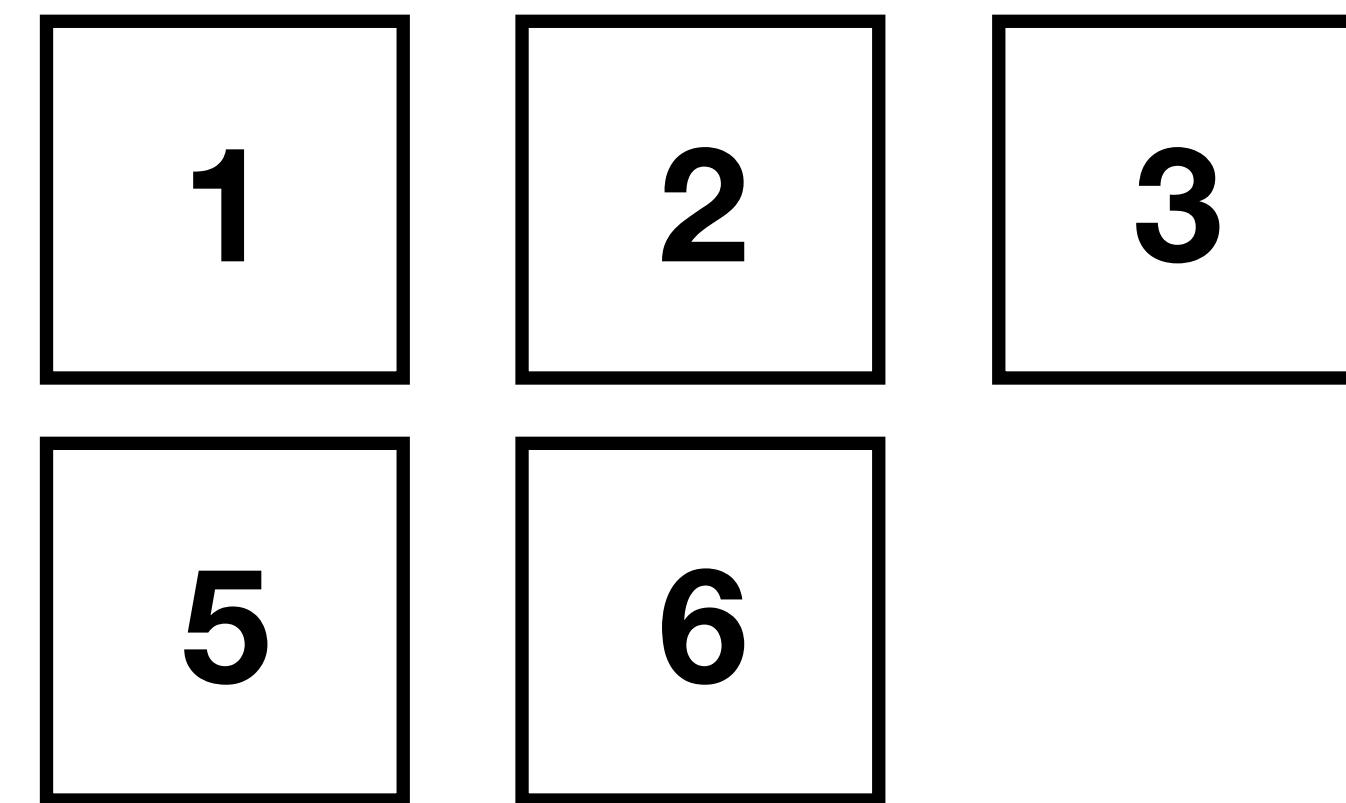
## 4번 데이터 삭제

### Framework

```
<Flicking>
  <div className="item" key={1}>1</div>
  <div className="item" key={2}>2</div>
  <div className="item" key={3}>3</div>
  <div className="item" key={5}>5</div>
  <div className="item" key={6}>6</div>
</Flicking>
```



### DOM

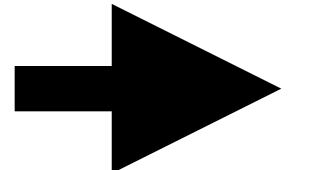


# 3.3 프레임워크 렌더링 원리 - DOM Diff

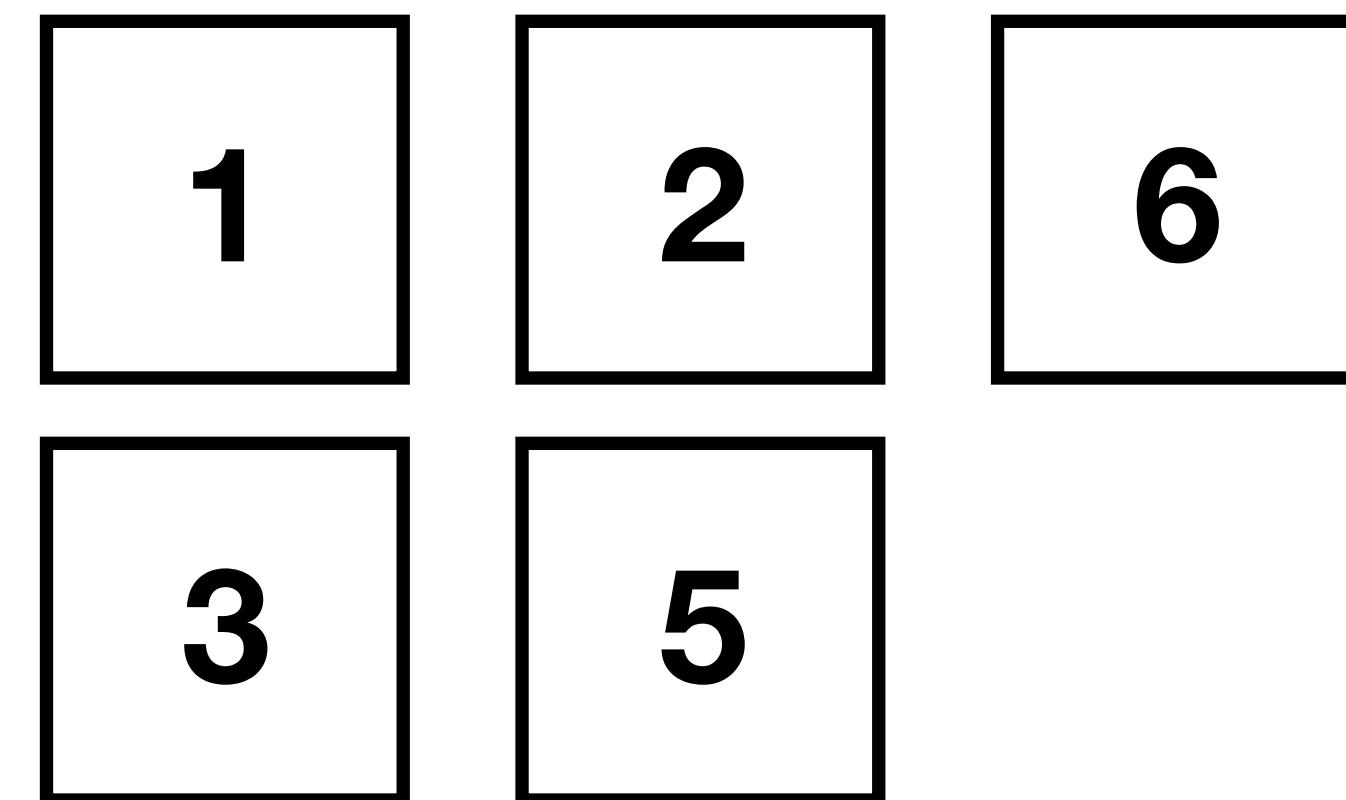
6번 데이터가 3번 데이터 앞으로 이동

Framework

```
<Flicking>
  <div className="item" key={1}>1</div>
  <div className="item" key={2}>2</div>
  <div className="item" key={6}>6</div>
  <div className="item" key={3}>3</div>
  <div className="item" key={5}>5</div>
</Flicking>
```



DOM

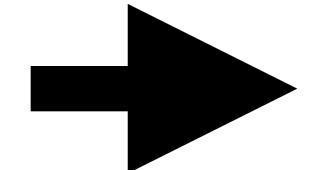


# 3.3 프레임워크 렌더링 원리 - DOM Diff

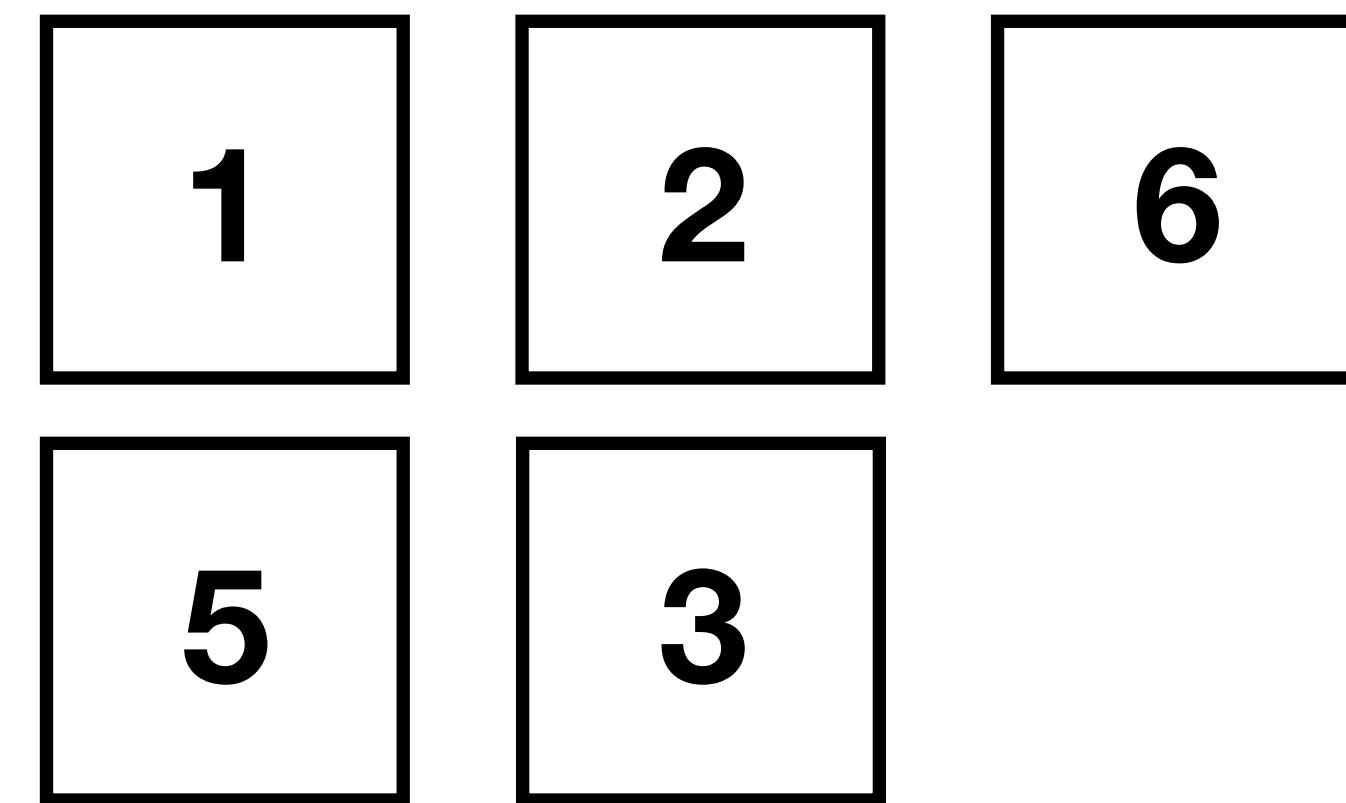
5번 데이터가 3번 데이터 앞으로 이동

Framework

```
<Flicking>
  <div className="item" key={1}>1</div>
  <div className="item" key={2}>2</div>
  <div className="item" key={6}>6</div>
  <div className="item" key={5}>5</div>
  <div className="item" key={3}>3</div>
</Flicking>
```



DOM



# 3.4 DOM을 조작하게 되면?

1, 2, 3, 4, 5, 6

```
<Flicking>
  <div className="item" key={1}>1</div>
  <div className="item" key={2}>2</div>
  <div className="item" key={3}>3</div>
  <div className="item" key={4}>4</div>
  <div className="item" key={5}>5</div>
  <div className="item" key={6}>6</div>
</Flicking>
```

1, 2, 6, 5, 3

```
<Flicking>
  <div className="item" key={1}>1</div>
  <div className="item" key={2}>2</div>
  <div className="item" key={6}>6</div>
  <div className="item" key={5}>5</div>
  <div className="item" key={3}>3</div>
</Flicking>
```

## 3.4 DOM을 조작하게 되면?

사용자가 `removeChild`를 호출해서 발생하는 에러

```
▶ Uncaught DOMException: Failed to execute 'insertBefore' on 'Node':  
inserted is not a child of this node.  
at insertBefore (http://localhost:3000/static/js/0.chunk.js:1467)  
at commitPlacement (http://localhost:3000/static/js/0.chunk.js:2)  
at commitAllHostEffects (http://localhost:3000/static/js/0.chunk.js:2)  
at HTMLUnknownElement.callCallback (http://localhost:3000/static/js/0.chunk.js:2)  
at Object.invokeGuardedCallbackDev (http://localhost:3000/static/js/0.chunk.js:2)  
at invokeGuardedCallback (http://localhost:3000/static/js/0.chunk.js:2)
```

# 4. Cross Framework Component 원리

# 4.1 CFC 원리 - DOM 조작 외부화

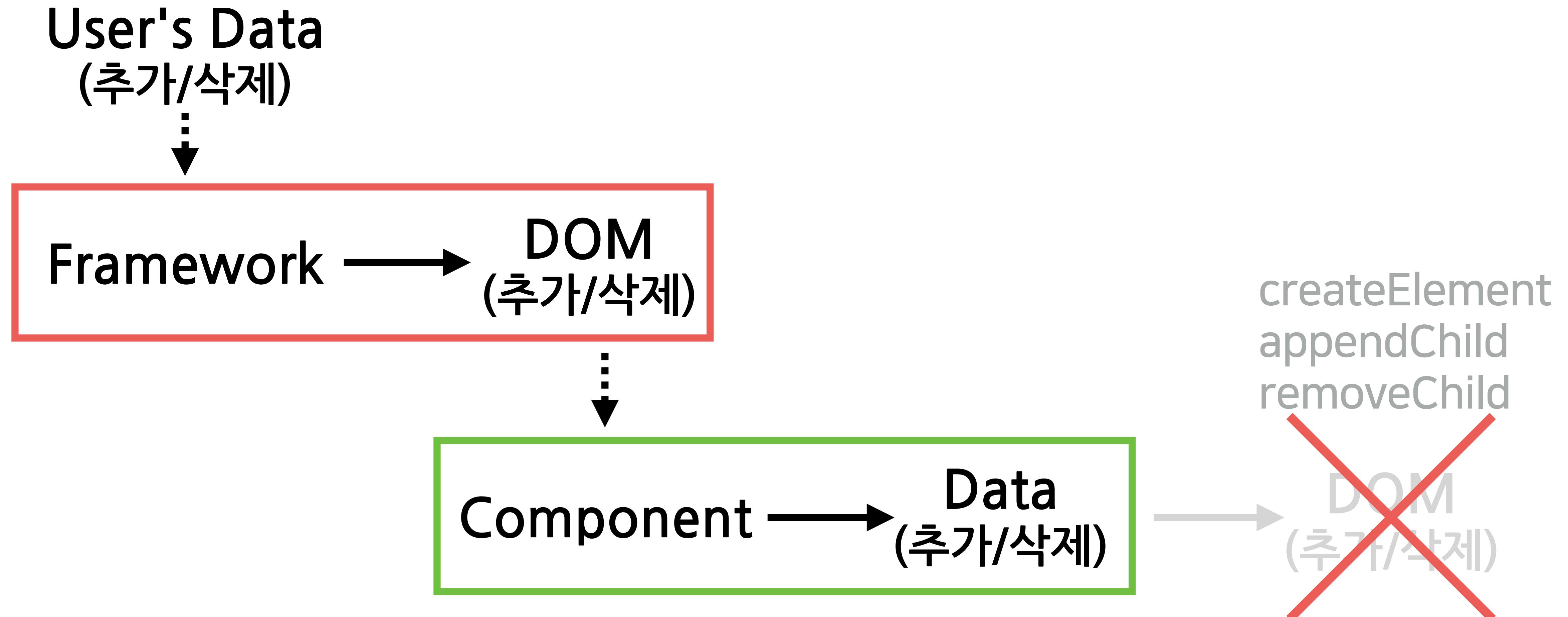
DOM을 추가/삭제하는 방법

DOM	Component	Framework
추가 (insertBefore)	추가 메서드 호출	배열 데이터 추가
삭제 (removeChild)	삭제 메서드 호출	배열 데이터 삭제

# 4.1 CFC 원리 - DOM 조작 외부화

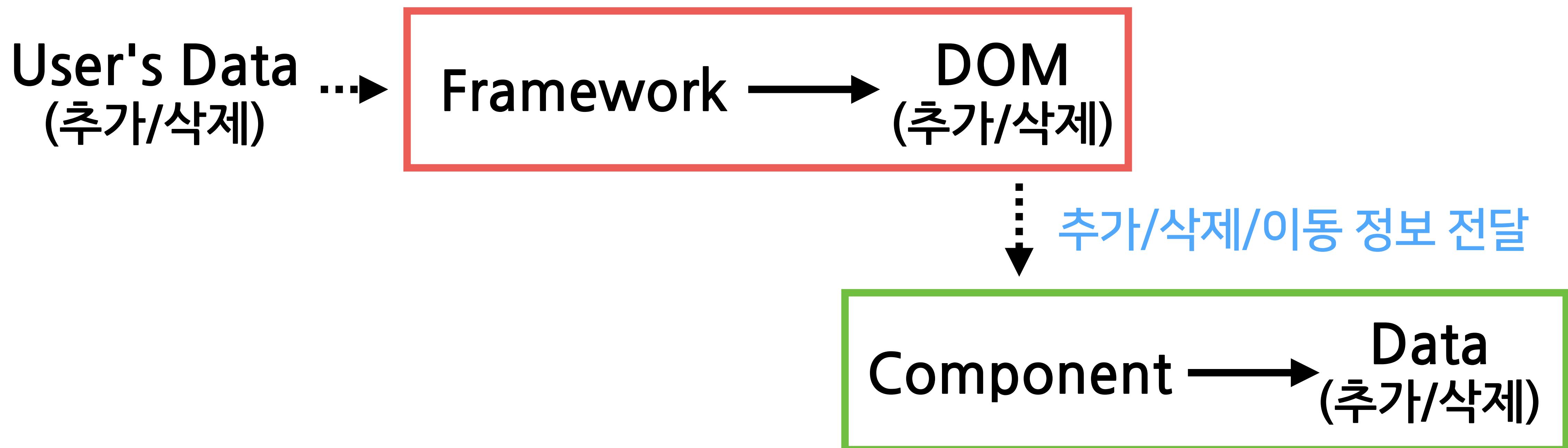


# 4.1 CFC 원리 - DOM 조작 외부화



## 4.2 CFC 원리 - 추가/삭제 메서드 호출 원리

추가/삭제 메서드를 통한 데이터 동기화



# 4.3 CFC 원리 - ListDiffer



# ListDiffer

# 4.3 CFC 원리 - ListDiffer

## 1. 추가(added):

추가된 요소의 인덱스 집합

## 2. 삭제(removed):

삭제된 요소의 인덱스 집합

## 3. 유지(maintained):

이전 배열에서 삭제되지 않은 아이템들의 [이전 인덱스, 이후 인덱스] 집합

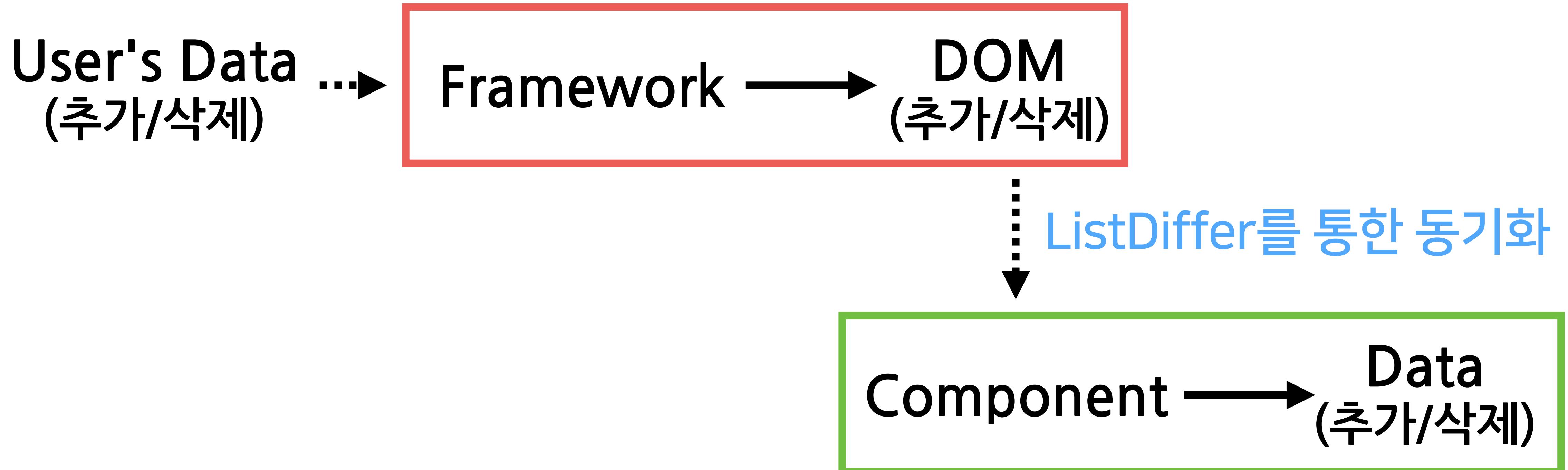
## 4.3 CFC 원리 - ListDiffer

삭제(removed) > 유지(maintaind) > 추가(added)



4 3 8 2 1 7

# 4.3 CFC 원리 - ListDiffer



# 5. Cross Framework Component

## 작용 방법

# 5.1 CFC 적용 방법 - 준비물

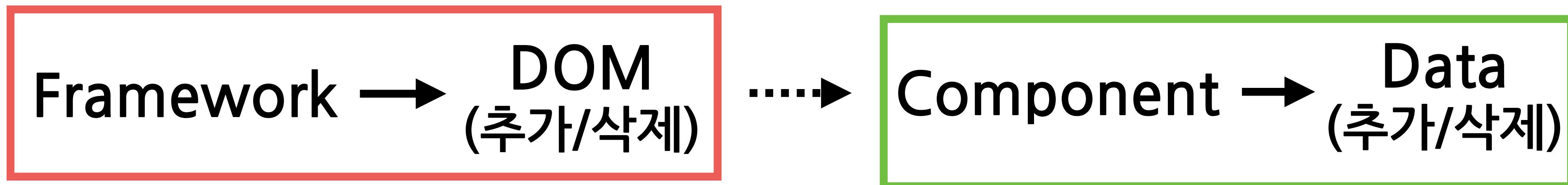
원리	준비물
기본	1. 추가 / 삭제 메서드
DOM 조작 외부화	2. 렌더링 외부화 옵션
추가/삭제 메서드 호출 원리	3. ListDiffer(diff함수)

## 5.2 CFC 적용 방법 - 렌더링 외부화 옵션 적용

렌더링 외부화 옵션: false



レン더링 외부화 옵션: true



## 5.2 CFC 적용 방법 - 렌더링 외부화 옵션 적용

### 렌더링 외부화 옵션 구현하기 전 코드

```
insert(data, index) {
  const el = makeDOM(data);

  this.items.splice(index, 0, el);
  document.body.insertBefore(el, this.items[index + 1]);
}
```

## 5.2 CFC 적용 방법 - 렌더링 외부화 옵션 적용

### 렌더링 외부화 옵션 구현한 코드

```
insert(data, index) {
  const el = this.renderExternal ? data : makeDOM(data);

  this.items.splice(index, 0, el);

  if (!this.renderExternal) {
    document.body.insertBefore(el, this.items[index + 1]);
  }
}
```

## 5.2 CFC 적용 방법 - 추가/삭제 메서드 호출

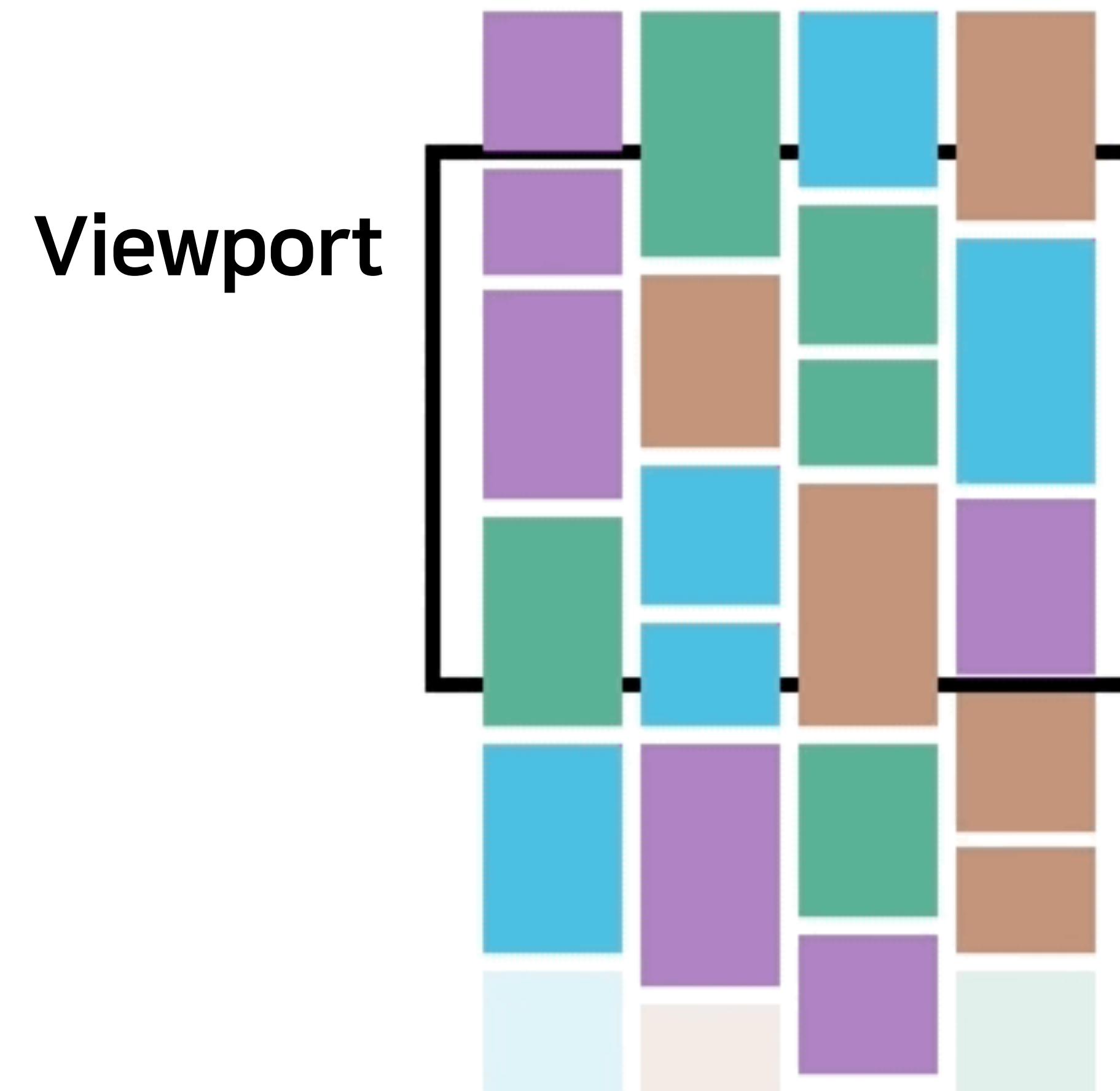
1. removed(삭제) > 2. maintained(유지) > 3. added(추가)

# 5.2 CFC 적용 방법 - 추가/삭제 메서드 호출

1. removed(삭제) > 2. maintained(유지) > 3. added(추가)

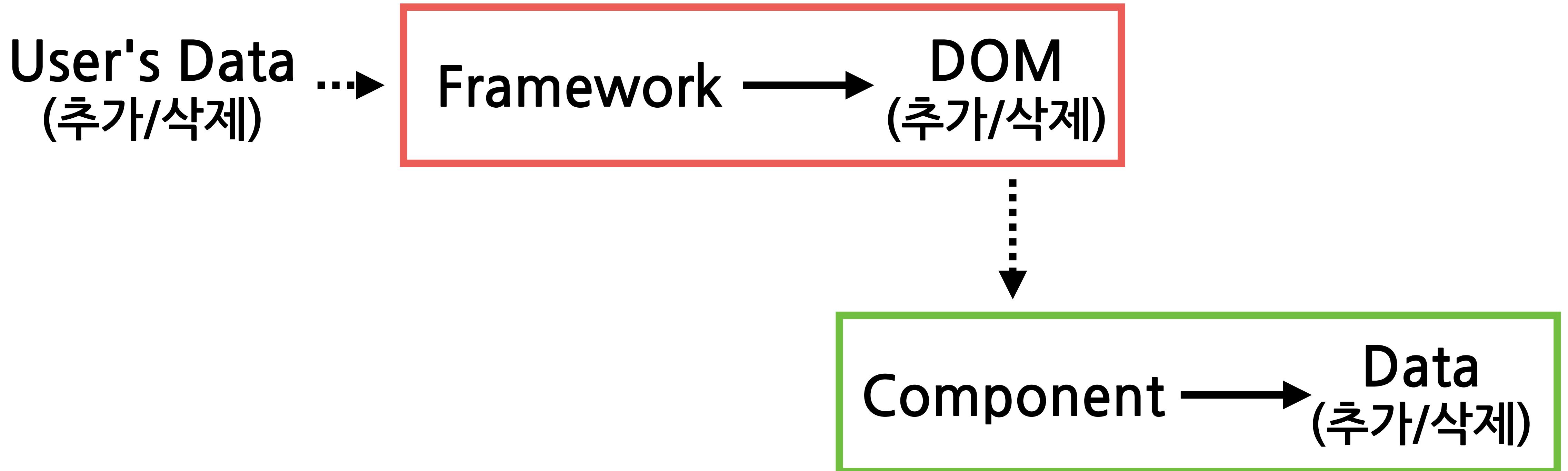
```
1 removed.forEach(index => {
    this.remove(index, list[index]);
});  
  
2 const nextList: Item[] = [];
maintained.forEach(([fromIndex]) => {
    nextList.push(items[fromIndex]);
});
this.items = nextList;  
  
3 added.forEach(index => {
    this.insert(index, list[index]);
});
```

# 5.3 CFC 적용 방법 - Recycle 구조



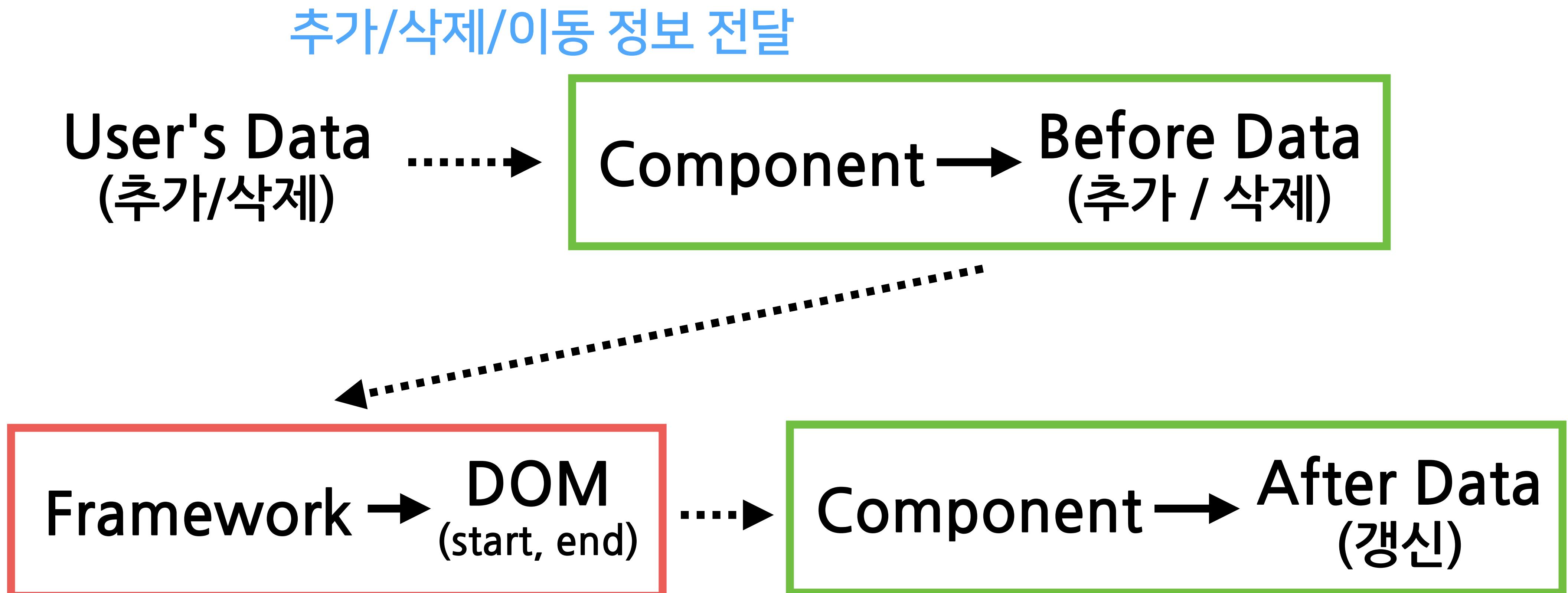
# 5.3 CFC 적용 방법 - Recycle 구조

No Recycle 구조



# 5.3 CFC 적용 방법 - Recycle 구조

## Recycle 구조



# 6. 누구나 쉽게 만들 수 있는 Infinite Scroll에 CFC 적용해보기

# 6 CFC 예제 - Infinite Scroll

## 1. CFC 구조 컴포넌트

1. Vanilla (@egjs/devview-infinite)
2. React (@egjs/react-devview-infinite)
3. Angular (@egjs/ngx-devview-infinite)
4. Vue (@egjs/vue-devview-infinite)

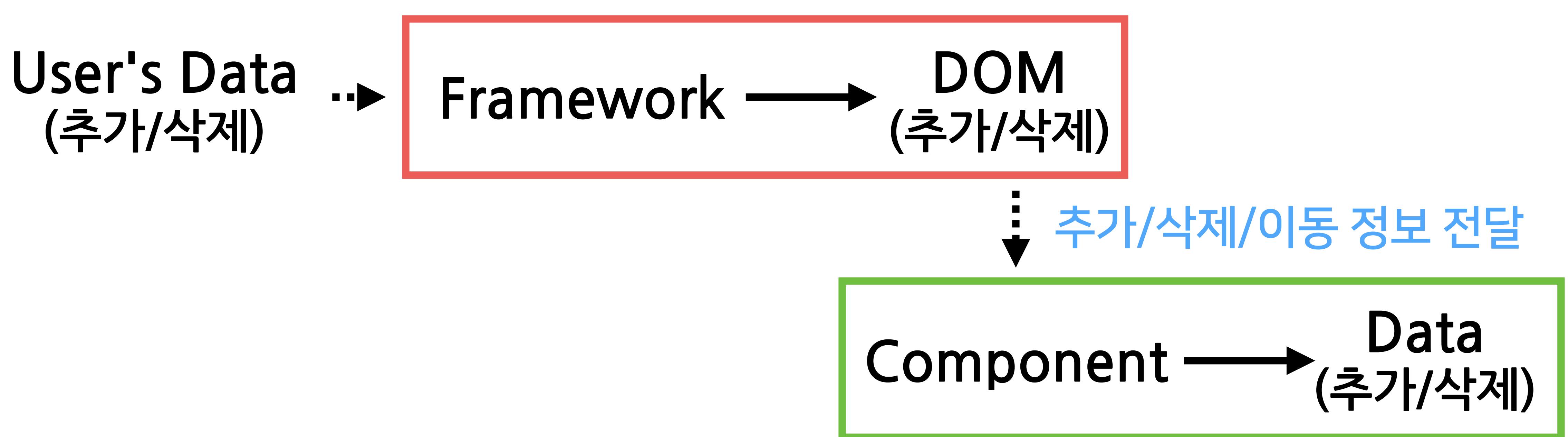
## 2. CFC Recycle 구조 컴포넌트

1. Vanilla (@egjs/devview-recycle)
2. React (@egjs/react-devview-recycle)
3. Angular (@egjs/ngx-devview-recycle)
4. Vue (@egjs/vue-devview-recycle)



# 6.1 CFC 예제 - Infinite Scroll(No Recycle)

No Recycle 구조



# 6.1 CFC 예제 - Infinite Scroll(No Recycle)

## No Recycle 구조의 Lifecycle

인스턴스 초기화

mounted

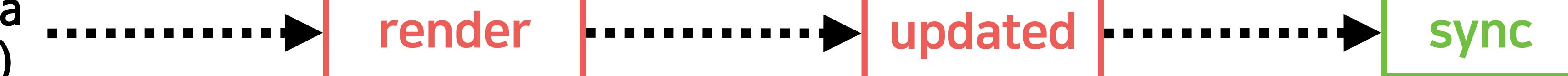
User's Data  
(추가/삭제)

render

추가/삭제/이동 정보 전달

updated

sync



# 6.1 CFC 예제 - Infinite Scroll(No Recycle)

## No Recycle 구조의 Lifecycle

인스턴스 초기화

mounted

User's Data  
(추가/삭제)

render

추가/삭제/이동 정보 전달

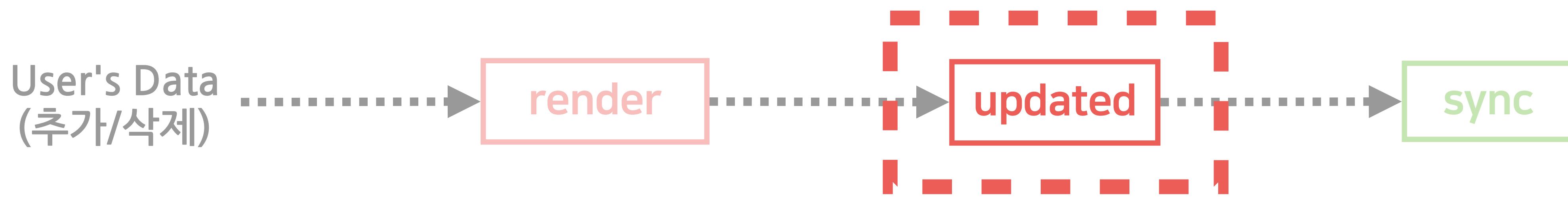
updated

sync



# 6.1 CFC 예제 - Infinite Scroll(No Recycle)

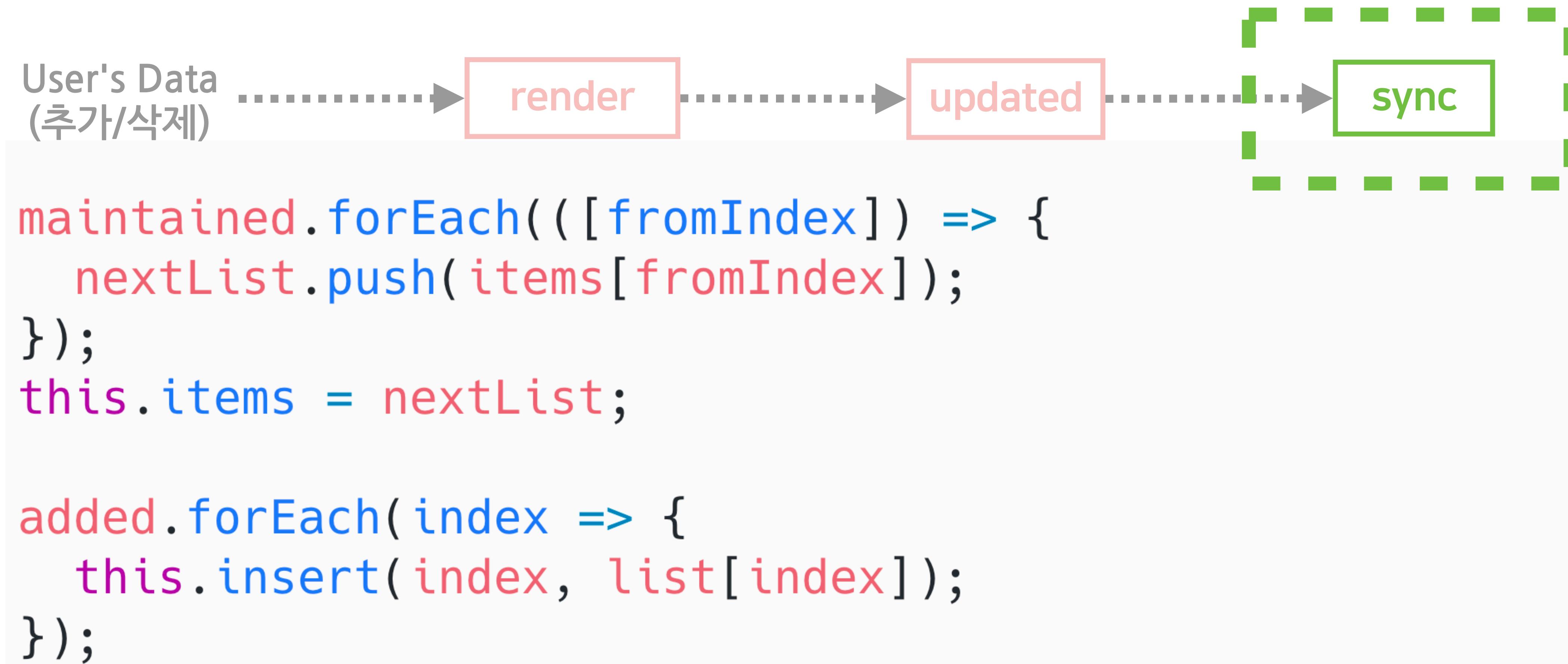
프레임워크에서 DOM을 추가/삭제한 후 sync 메서드 호출



```
public componentDidUpdate() {  
  this.infinite.sync(this.container.children);  
}
```

# 6.1 CFC 예제 - Infinite Scroll(No Recycle)

removed(삭제) > maintained(유지) > added(추가) 방법으로 동기화



# 6.1 CFC 예제 - Infinite Scroll(No Recycle)

## DOM 조작 외부화

```
public insert(index: number, data: HTMLElement | string) {
  items.splice(index, 0, item);

  if (!options.renderExternal) {
    const nextItem = items[index + 1];
    this.container.insertBefore(item.el, nextItem ? nextItem.el : null);
  }
}
```

# 6.1 CFC 예제 - Infinite Scroll(No Recycle)

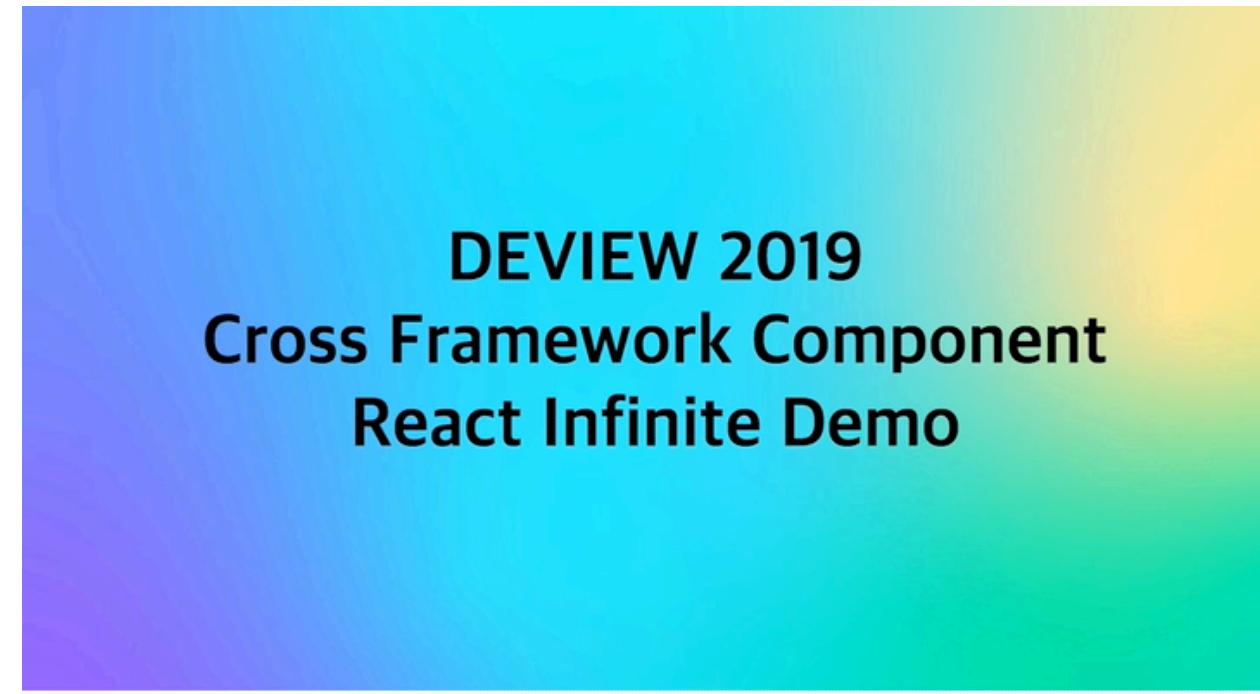
The screenshot shows a browser developer tools window with the 'Elements' tab selected. The DOM tree on the left displays a list of items from 'Item 1' to 'Item 13' within a container div. The right pane shows the corresponding JavaScript code. The code uses absolute positioning for each item within a container div, with top values increasing by 46px for each subsequent item. It includes imports for 'index.ts' and several script tags linking to external files like 'startup.2daca004.js' and 'common.sandbox.b14be9a3.chunk.js'. The bottom of the code pane shows the navigation path: html > body > div#app.container.

```
... <div id="app" class="container" style="height: 598px;"> == $0
    <div class="item" style="position: absolute; top: 0px;">Item 1</div>
    <div class="item" style="position: absolute; top: 46px;">Item 2</div>
    <div class="item" style="position: absolute; top: 92px;">Item 3</div>
    <div class="item" style="position: absolute; top: 138px;">Item 4</div>
    <div class="item" style="position: absolute; top: 184px;">Item 5</div>
    <div class="item" style="position: absolute; top: 230px;">Item 6</div>
    <div class="item" style="position: absolute; top: 276px;">Item 7</div>
    <div class="item" style="position: absolute; top: 322px;">Item 8</div>
    <div class="item" style="position: absolute; top: 368px;">Item 9</div>
    <div class="item" style="position: absolute; top: 414px;">Item 10</div>
    <div class="item" style="position: absolute; top: 460px;">Item 11</div>
    <div class="item" style="position: absolute; top: 506px;">Item 12</div>
    <div class="item" style="position: absolute; top: 552px;">Item 13</div>
</div>
<script src="src/index.ts"></script>
<script type="text/javascript" src="https://codesandbox.io/static/js/sandbox.startup.2daca004.js"></script>
<script type="text/javascript" src="https://codesandbox.io/static/js/common.sandbox.b14be9a3.chunk.js"></script>
<script type="text/javascript" src="https://codesandbox.io/static/js/vendors~sandbox.35cc9f7e.chunk.js"></script>
<script type="text/javascript" src="https://codesandbox.io/static/js/
```

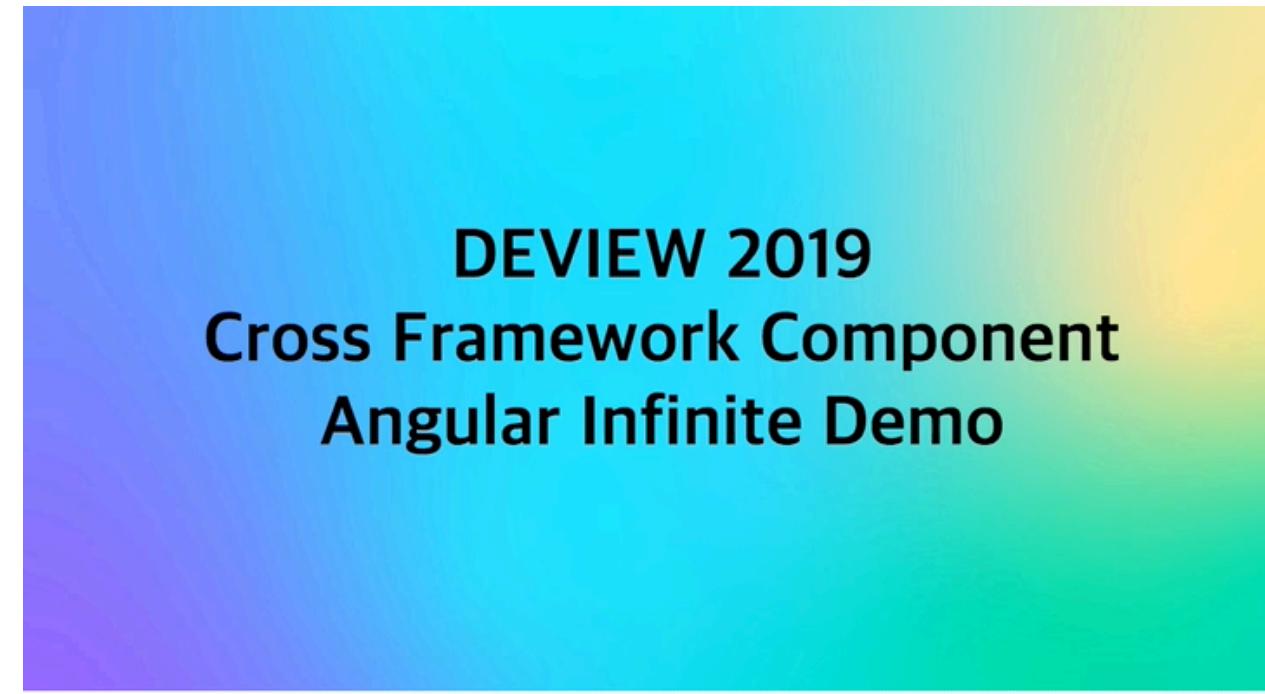
html body div#app.container

Vanilla Infinite

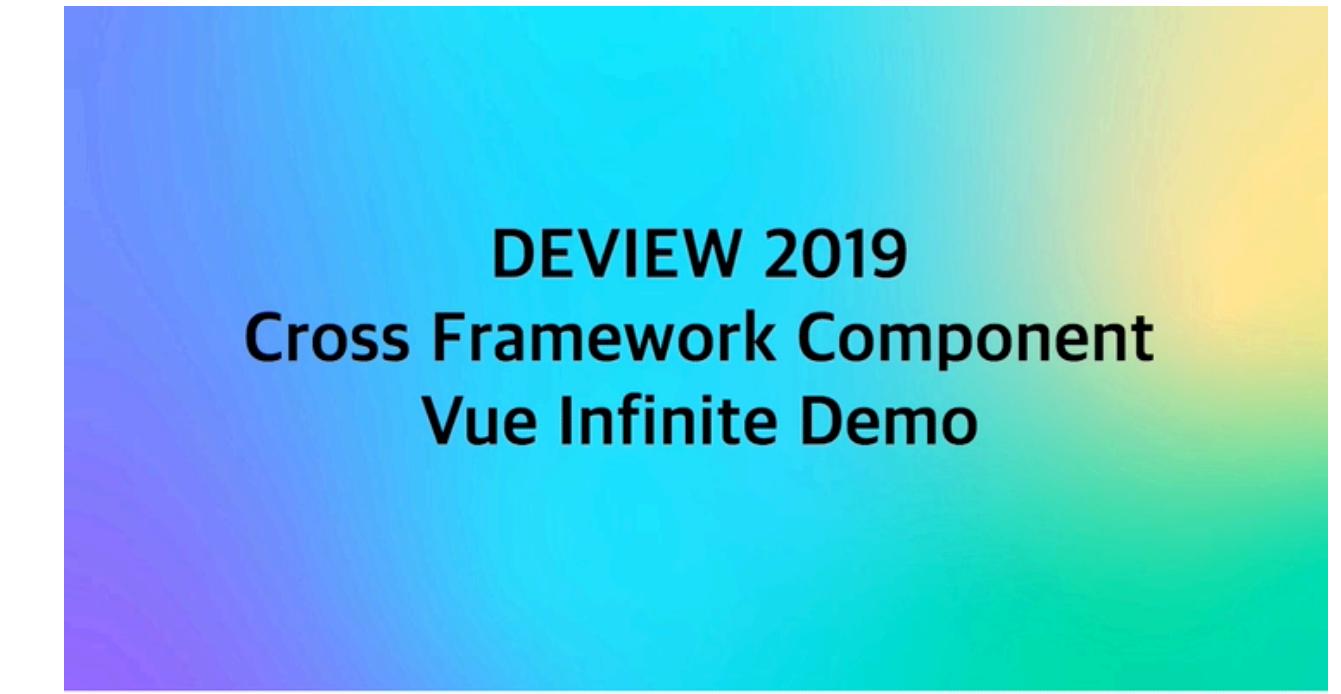
# 6.1 CFC 예제 - Infinite Scroll(No Recycle)



Items:

[Shuffle](#)

items:

[Shuffle](#)

items: 1

[Shuffle](#)

x7x43.csb.app의 응답을 기다리...

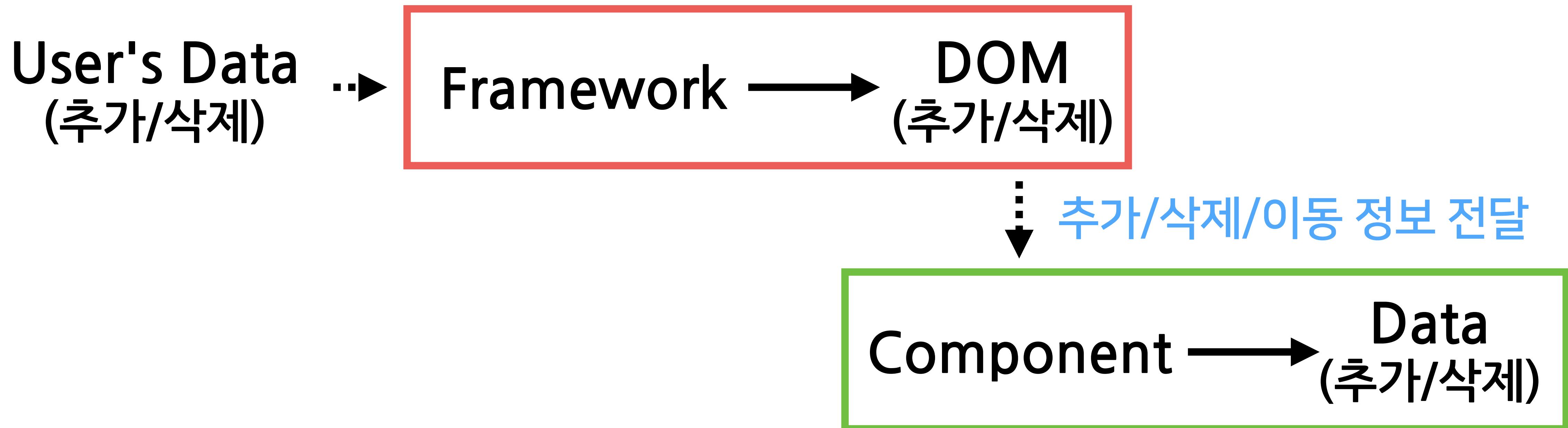
React Infinite

Angular Infinite

Vue Infinite

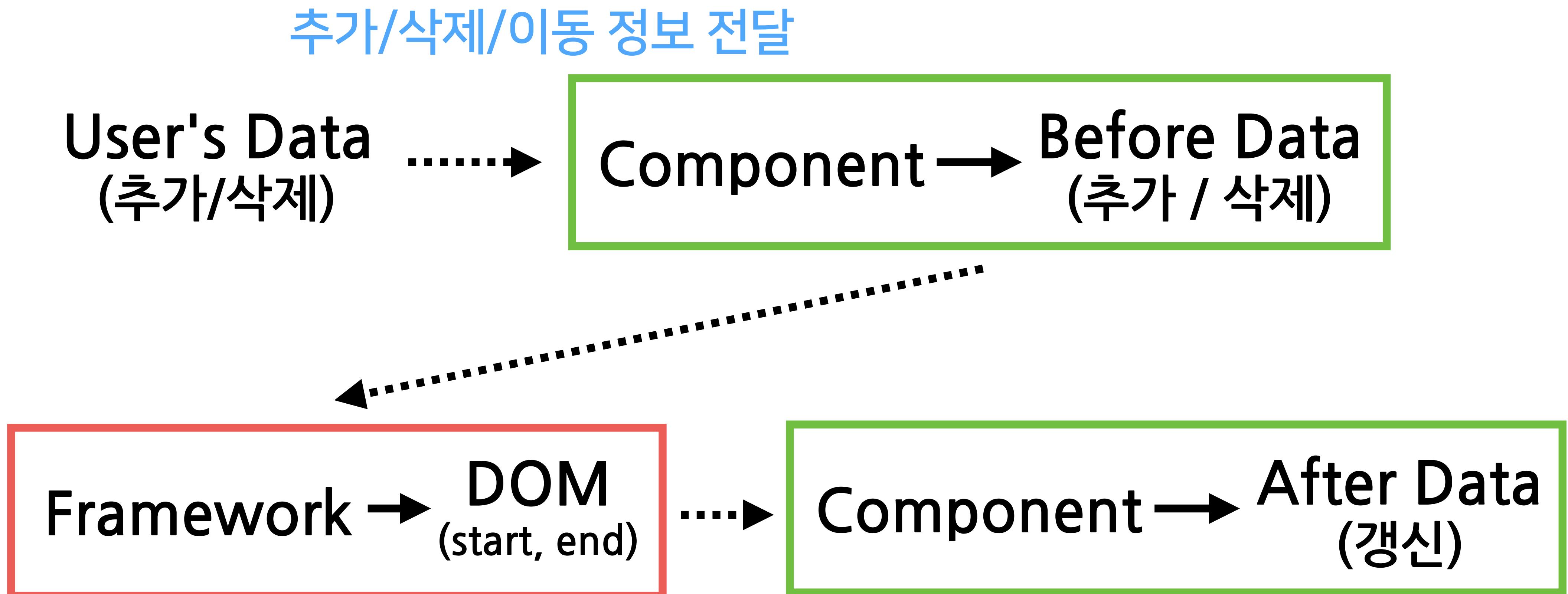
# 6.2 CFC 예제 - Infinite Scroll(Recycle)

No Recycle 구조



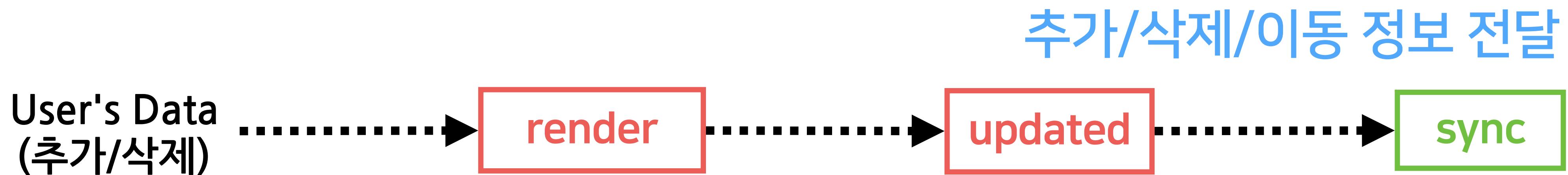
# 6.2 CFC 예제 - Infinite Scroll(Recycle)

## Recycle 구조

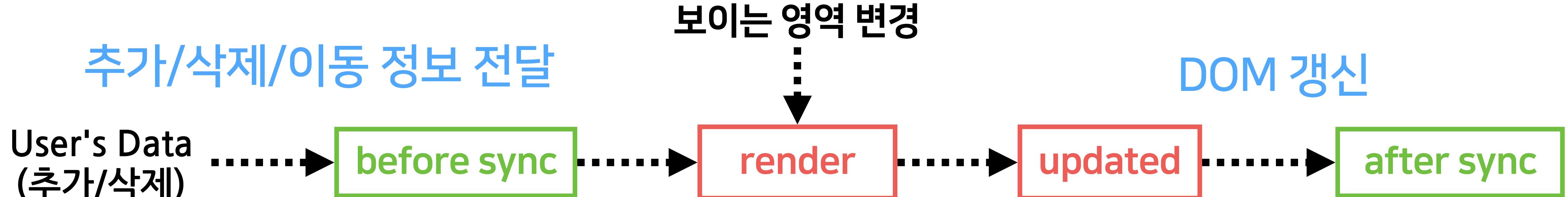


# 6.2 CFC 예제 - Infinite Scroll(Recycle)

## No Recycle 구조의 Lifecycle



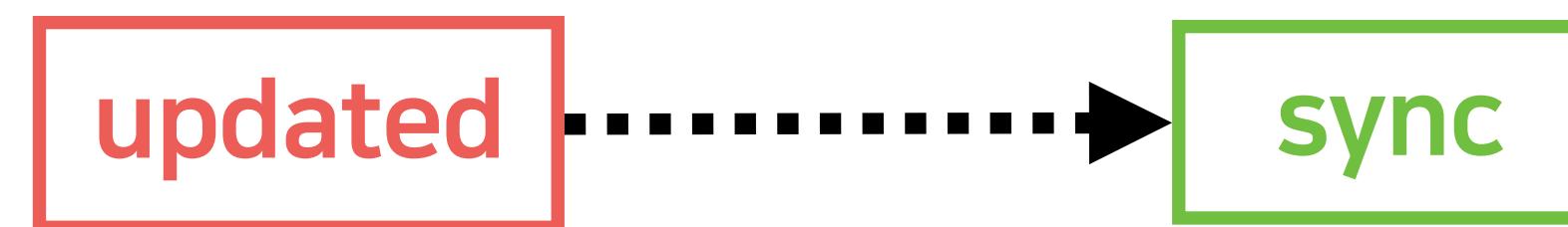
## Recycle 구조의 Lifecycle



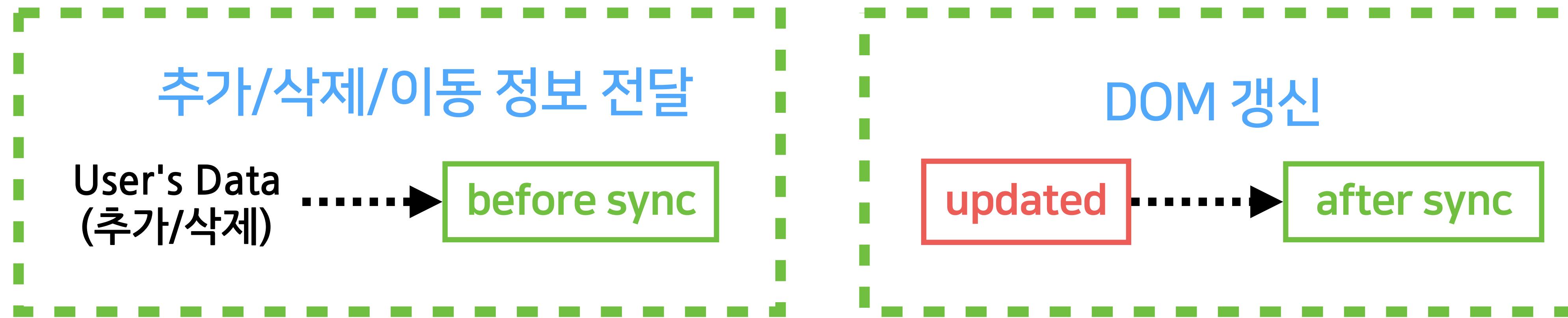
# 6.2 CFC 예제 - Infinite Scroll(Recycle)

No Recycle 구조의 sync

추가/삭제/이동 정보 전달

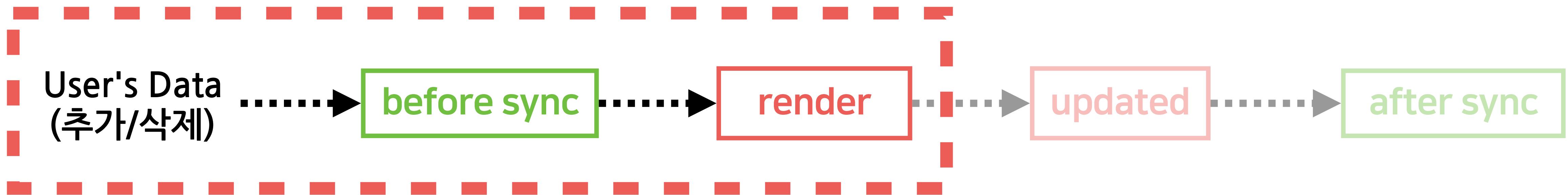


Recycle 구조의 sync



# 6.2 CFC 예제 - Infinite Scroll(Recycle)

DOM을 추가/삭제하기 전에 beforeSync 메서드 호출



```
public render() {
  const children = React.Children.toArray(this.props.children);

  this.recycle.beforeSync(children.map(child => child.key));
  const indexes = this.recycle.getRenderingIndexes();

  const visibleChildren = children.slice(indexes.start, indexes.end + 1);

  return (<Tag {...attributes}>{visibleChildren}</Tag>);
}
```

# 6.2 CFC 예제 - Infinite Scroll(Recycle)

DOM을 추가/삭제하기 전에 beforeSync 메서드 호출(Angular)

```
ngOnChanges() {
  const itemKeys = this.items.map((item, i) => this.trackBy(i, item));

  this.recycle.beforeSync(itemKeys);
  const indexes = this.recycle.getRenderingIndexes();

  this.visibleItems = items.slice(indexes.start, indexes.end + 1);
}
```

```
<div NgxDevviewRecycle #recycle [items]="items" [trackBy]="trackBy">
  <div class="item" *ngFor="let item of recycle.visibleItems; trackBy: trackBy;">
    Item {{item}}
  </div>
</div>
```

# 6.2 CFC 예제 - Infinite Scroll(Recycle)

removed(삭제) > maintained(유지) > added(추가) 방법으로 사전 동기화

```
public beforeSync(data: any[]) {
  const items = this.items;
  const itemKeys = this.items.map(item => item.key);
  const { maintained, added } = diff(itemKeys, data, key => key);

  const nextList: Item[] = [];

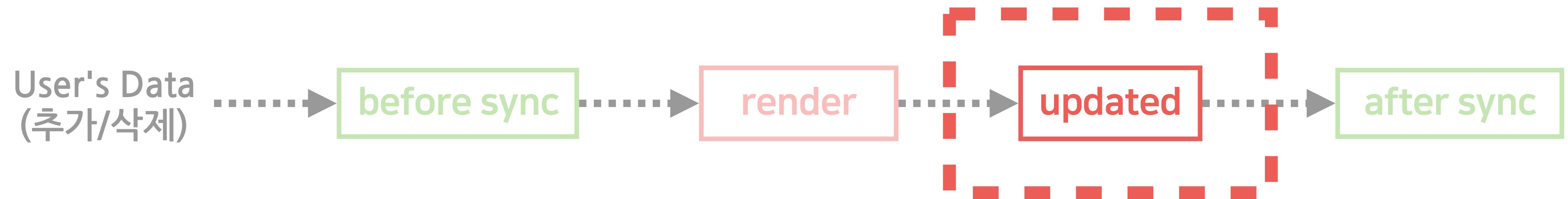
  maintained.forEach(([fromIndex]) => {
    nextList.push(items[fromIndex]);
  });
  this.items = nextList;

  added.forEach(index => {
    this.insert(index, "", data[index]);
  });

  this.refreshCursor();
}
```

# 6.2 CFC 예제 - Infinite Scroll(Recycle)

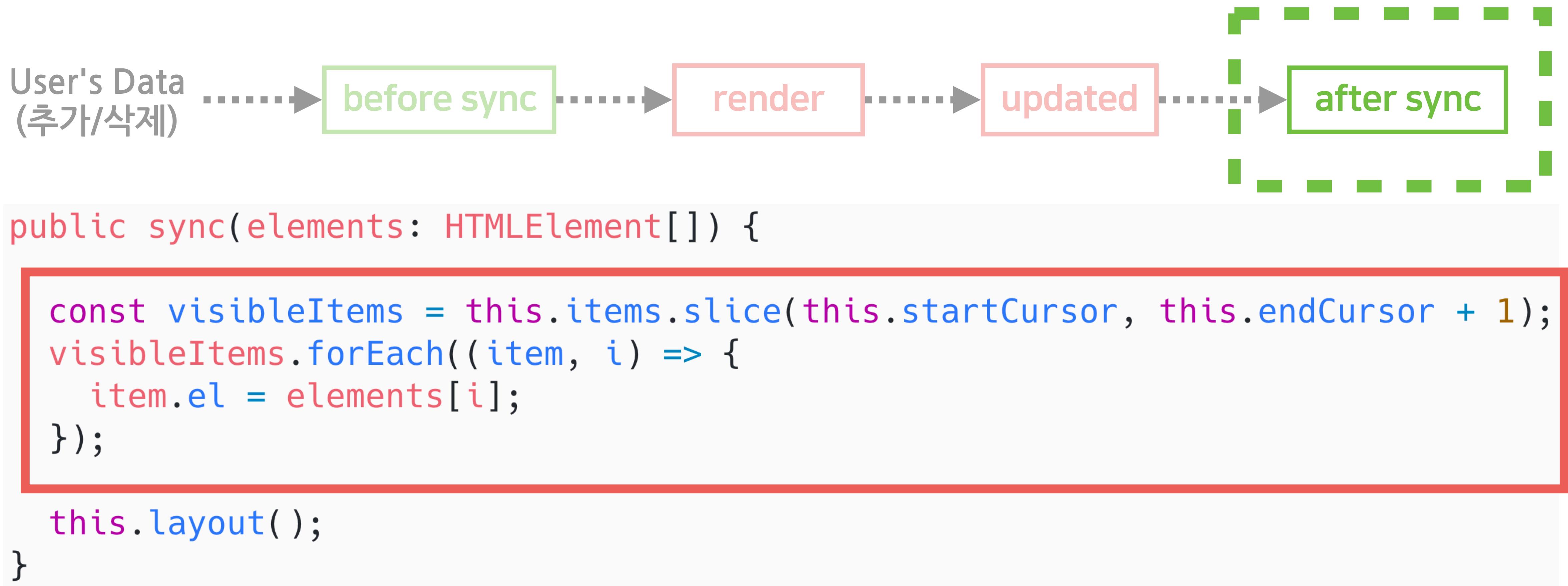
프레임워크에서 DOM을 추가/삭제한 후 sync 메서드 호출



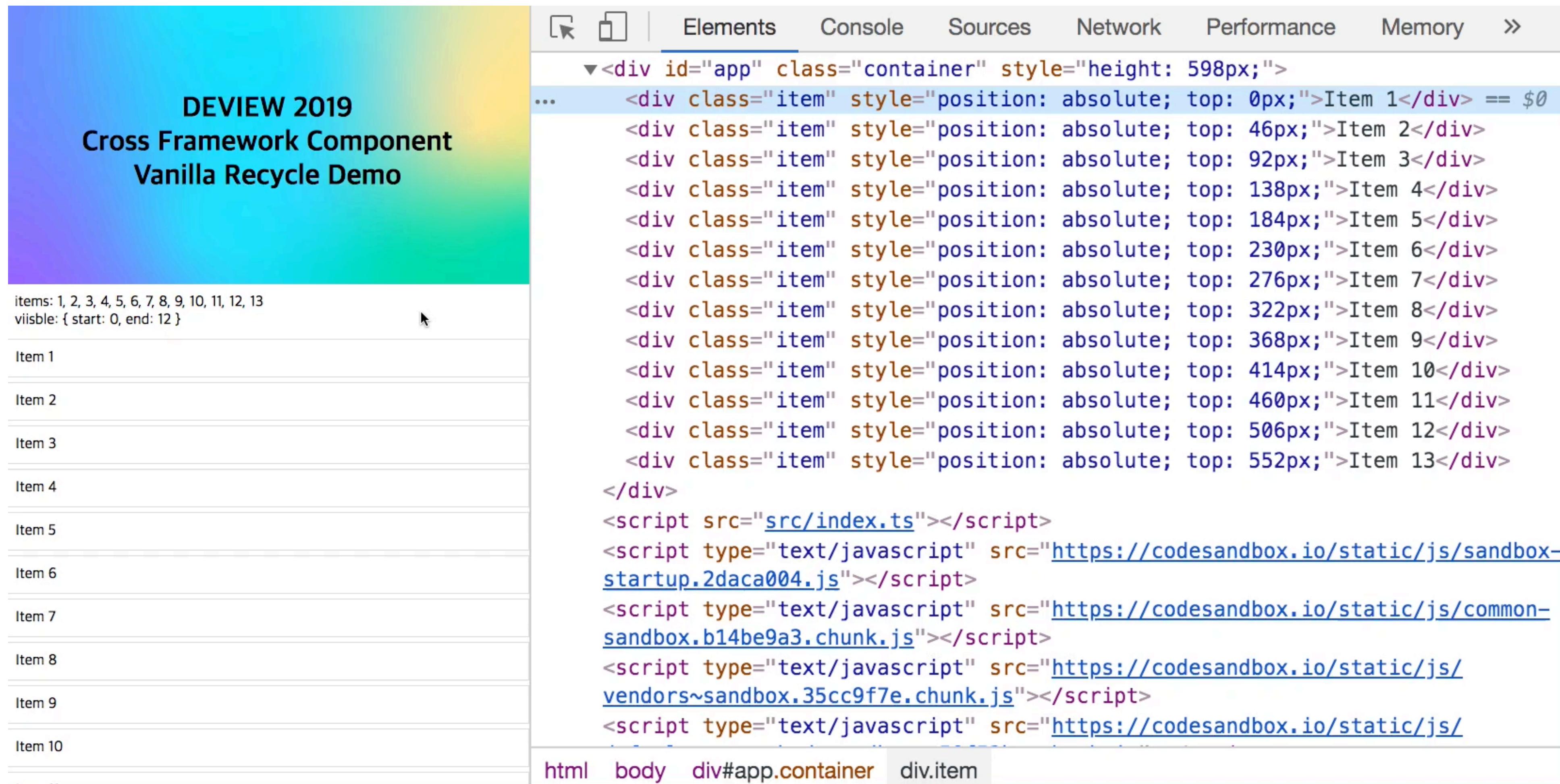
```
public componentDidUpdate() {  
  this.infinite.sync(this.container.children);  
}
```

# 6.2 CFC 예제 - Infinite Scroll(Recycle)

## DOM 갱신



# 6.2 CFC 예제 - Infinite Scroll(Recycle)



## Vanilla Recycle

# 6.2 CFC 예제 - Infinite Scroll(Recycle)



Shuffle



{ start: 0, end: 0 }

Shuffle

Item 1



{ start: 0, end: 0 }

Shuffle

spiq9.csb.app의 응답을 기다리...

React Recycle

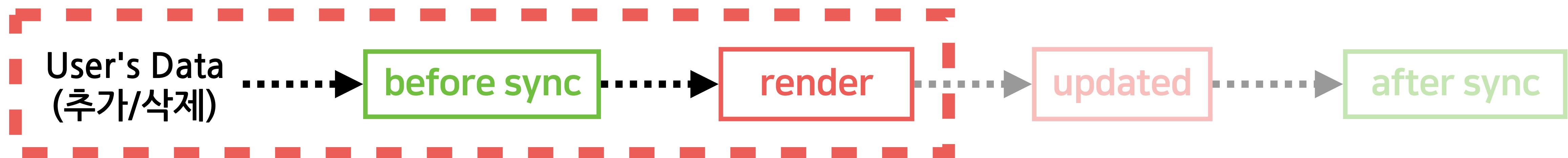
Angular Recycle

Vue Recycle

# 7. egjs에서 CFC를 적용한 사례

# 7.1 CFC 사례 - InfiniteGrid

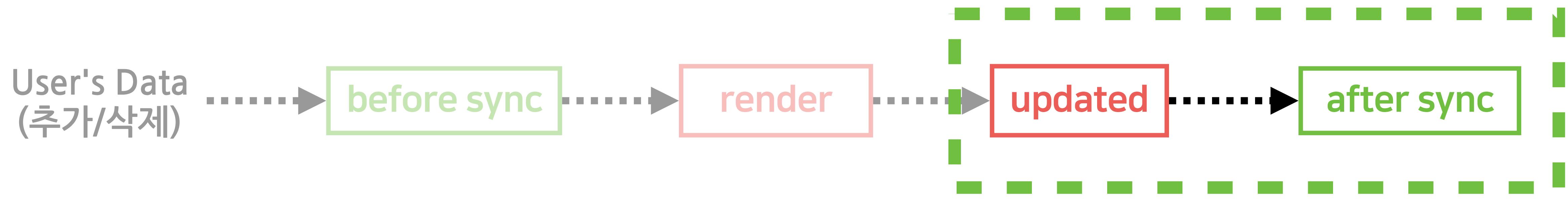
removed(삭제) > maintained(유지) > added(추가) 방법으로 사전 동기화



```
const newGroups = categorize(newItems);
...
removed.forEach(removedIndex => {
  delete groupKeys[groups[removedIndex].groupKey];
});
maintained.forEach(([fromIndex]) => {
  nextGroups.push(groups[fromIndex]);
});
added.forEach(addedIndex => {
  this.insertGroup(newGroups[addedIndex], addedIndex);
});
...
```

# 7.1 CFC 사례 - InfiniteGrid

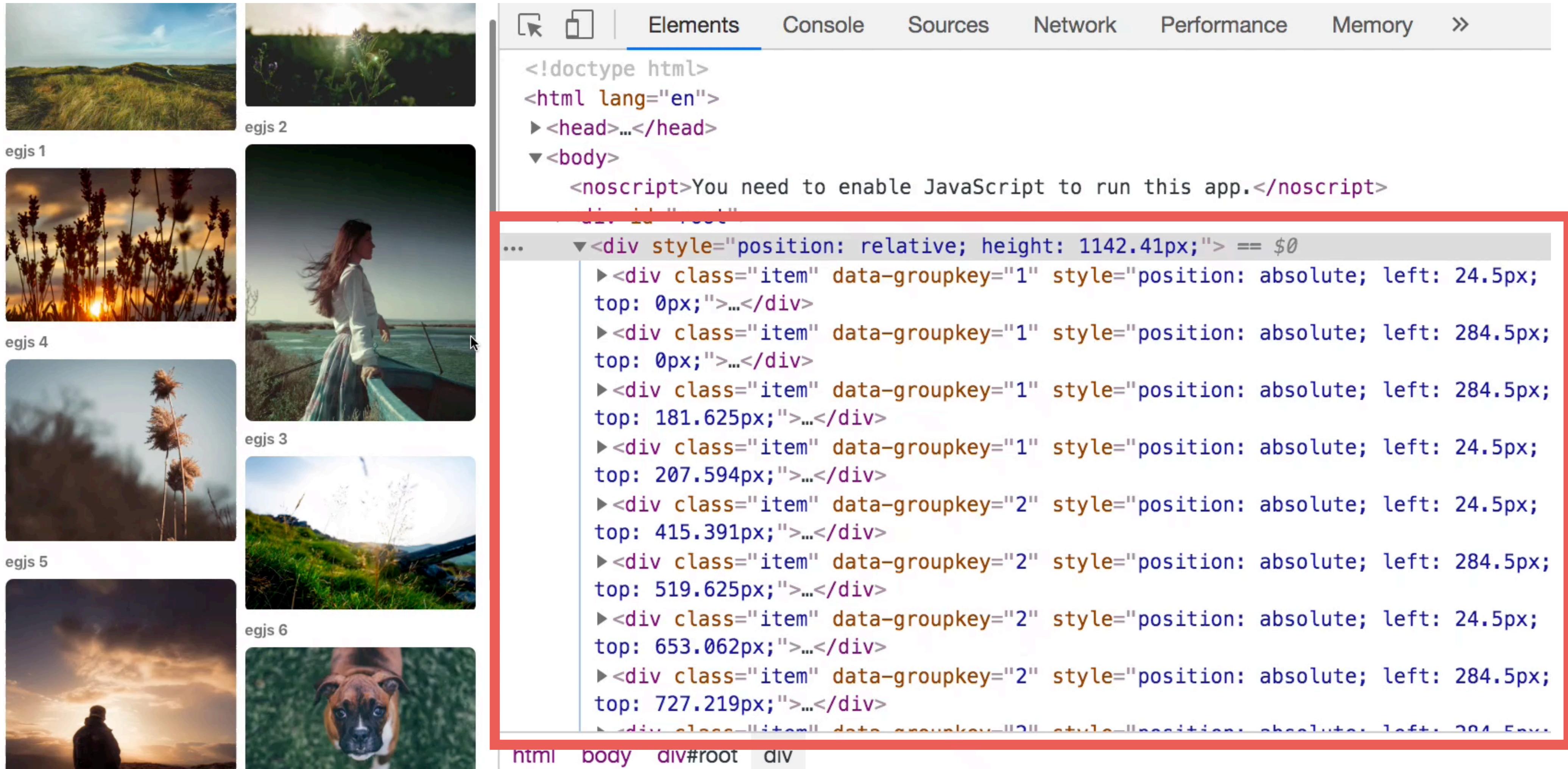
## DOM 갱신



```
public sync(elements: HTMLElement[]) {
  const items = this.getRenderingItems().forEach((item, i) => {
    item.el = elements[i];

    DOMRenderer.renderItem(item, item.rect);
  });
}
```

# 7.1 CFC 사례 - InfiniteGrid



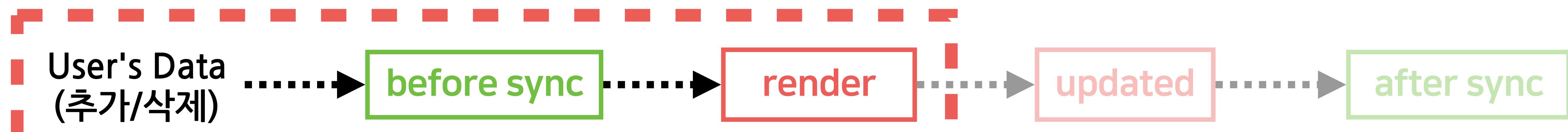
The screenshot shows a web page with a grid of images. The grid is 6 rows by 2 columns. The images are labeled egjs 1 through egjs 6. The first row contains a landscape and a close-up of plants. The second row contains a sunset over a field and a woman in a boat. The third row contains tall grass. The fourth row contains a person at sunset. The fifth row contains a dog's face. The sixth row contains a landscape. To the right of the grid is a developer tools Elements tab showing the DOM structure. A red box highlights the list of 'item' divs under the root div, which have absolute positions and specific left and top coordinates.

```
<!doctype html>
<html lang="en">
  <head>...</head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root">
      ...
      <div style="position: relative; height: 1142.41px;"> == $0
        <div class="item" data-groupkey="1" style="position: absolute; left: 24.5px; top: 0px;">...</div>
        <div class="item" data-groupkey="1" style="position: absolute; left: 284.5px; top: 0px;">...</div>
        <div class="item" data-groupkey="1" style="position: absolute; left: 284.5px; top: 181.625px;">...</div>
        <div class="item" data-groupkey="1" style="position: absolute; left: 24.5px; top: 207.594px;">...</div>
        <div class="item" data-groupkey="2" style="position: absolute; left: 24.5px; top: 415.391px;">...</div>
        <div class="item" data-groupkey="2" style="position: absolute; left: 284.5px; top: 519.625px;">...</div>
        <div class="item" data-groupkey="2" style="position: absolute; left: 24.5px; top: 653.062px;">...</div>
        <div class="item" data-groupkey="2" style="position: absolute; left: 284.5px; top: 727.219px;">...</div>
      ...
    </div>
  </body>
</html>
```

<https://naver.github.io/egjs-infinitegrid/assets/html/grid.html>

# 7.2 CFC 사례 - Flicking

removed(삭제) > maintained(유지) > added(추가) 방법으로 사전 동기화



```
render() {  
  const children = React.Children.toArray(this.props.children);  
  const diffResult = this.jsxDiffer.update(children.map(child => child.key));
```

```
flicking.beforeSync(diffResult);  
const visibleChildren = flicking.getRenderingIndexes(diffResult).map(index => {  
  return children[index];  
});
```

```
return <div>{visibleChildren}</div>;  
}
```

# 7.2 CFC 사례 - Flicking

## DOM 갱신

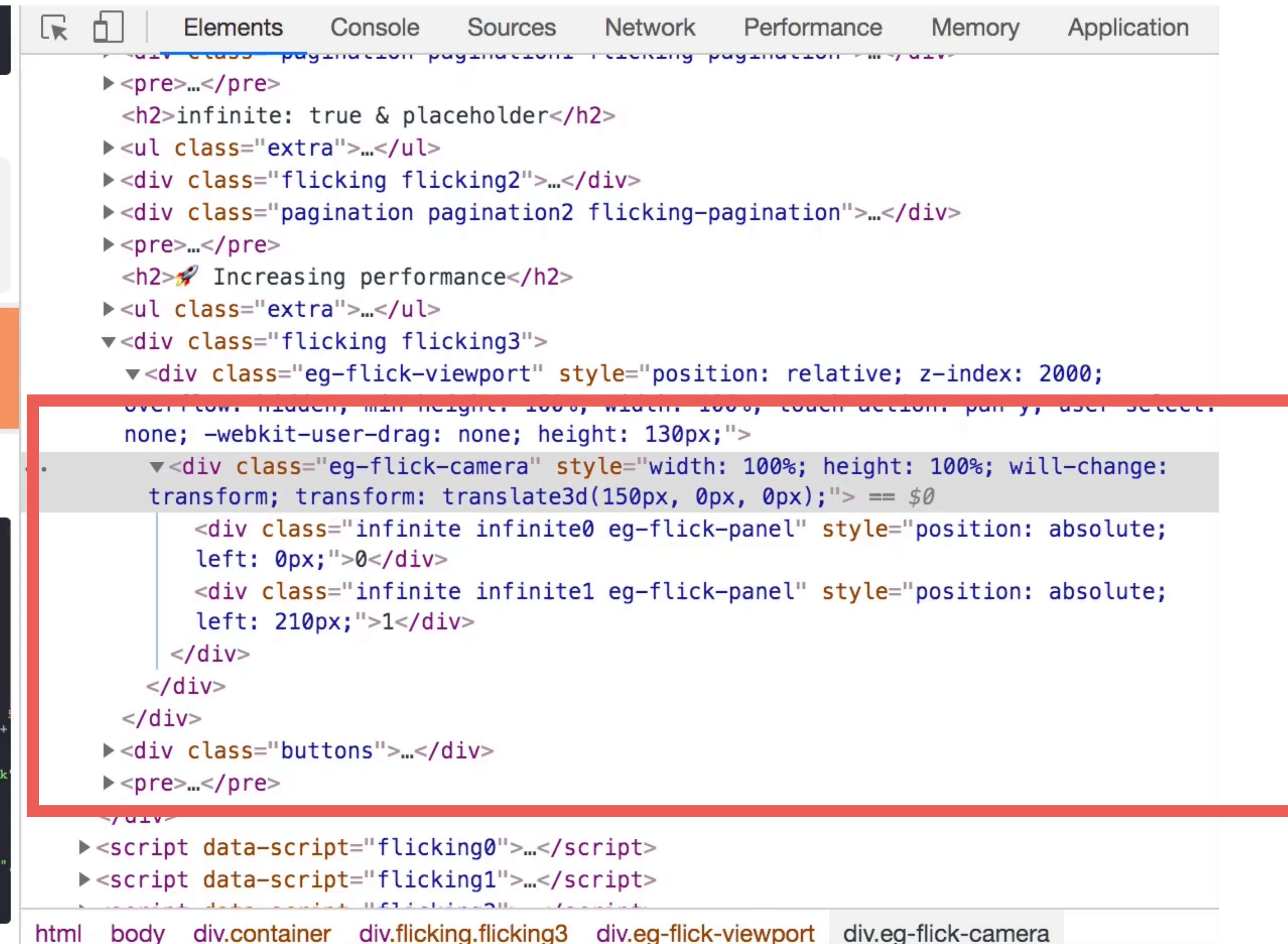


```
public sync(diffInfo: SyncResult): this {
  const { list, added } = diffInfo;
  const visiblePanels = this.viewport.getVisiblePanels();

  added.forEach(addedIndex => {
    const addedElement = list[addedIndex];
    const beforePanel = visiblePanels[addedIndex];

    beforePanel.setElement(addedElement);
  });
}
```

# 7.2 CFC 사례 - Flicking



The screenshot shows a browser's developer tools with the 'Elements' tab selected. The DOM tree on the right is highlighted with a red box around the `div.eg-flick-viewport` and `div.eg-flick-camera` elements. The code block on the left contains the JavaScript and CSS for the Flicking component.

```

    });
  }, 1500);
};

// Increasing performance


- You can increase performance by enabling renderOnlyVisible, isEqualSize, and isConstantSize of Flicking
- See its effect: https://www.youtube.com/watch?v=R-DKq-fgeSI
- Check the details of the options: Link



0
1



Start
Stop



```

var flicking3 = new eg.Flicking(".flicking3", {
  renderOnlyVisible: true,
  isEqualSize: true,
  isConstantSize: true,
  infinite: true,
  gap: 10,
  duration: 5,
  moveType: "freeScroll"
});

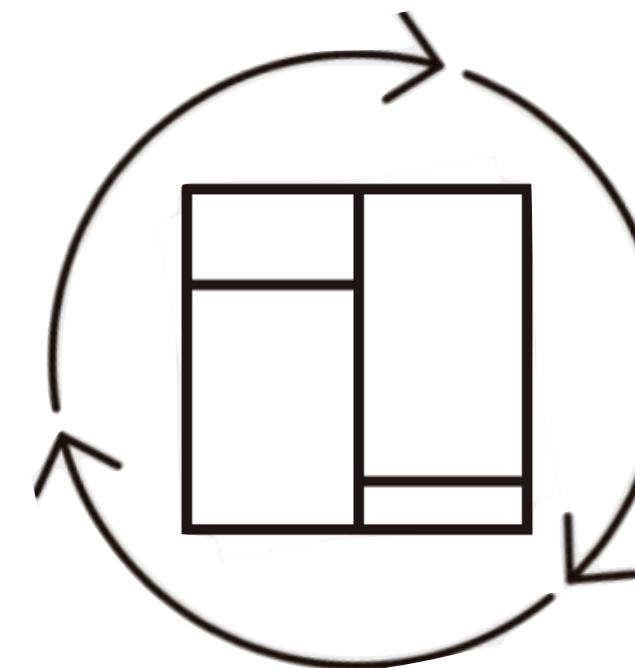
flicking3.on("needPanel", function (e) {
  var panels = e.fill([
    '<div class="infinite infinite0 eg-flick-panel" style="position: absolute; left: 0px;">0</div>',
    '<div class="infinite infinite1 eg-flick-panel" style="position: absolute; left: 210px;">1</div>'
  ]);
});
document.querySelector("#start").addEventListener("click", function() {
  flicking3.on("moveEnd", function(e) {
    flicking3.next();
  });
  flicking3.next();
});
document.querySelector("#stop").addEventListener("click", function() {
  flicking3.off("moveEnd");
});

```

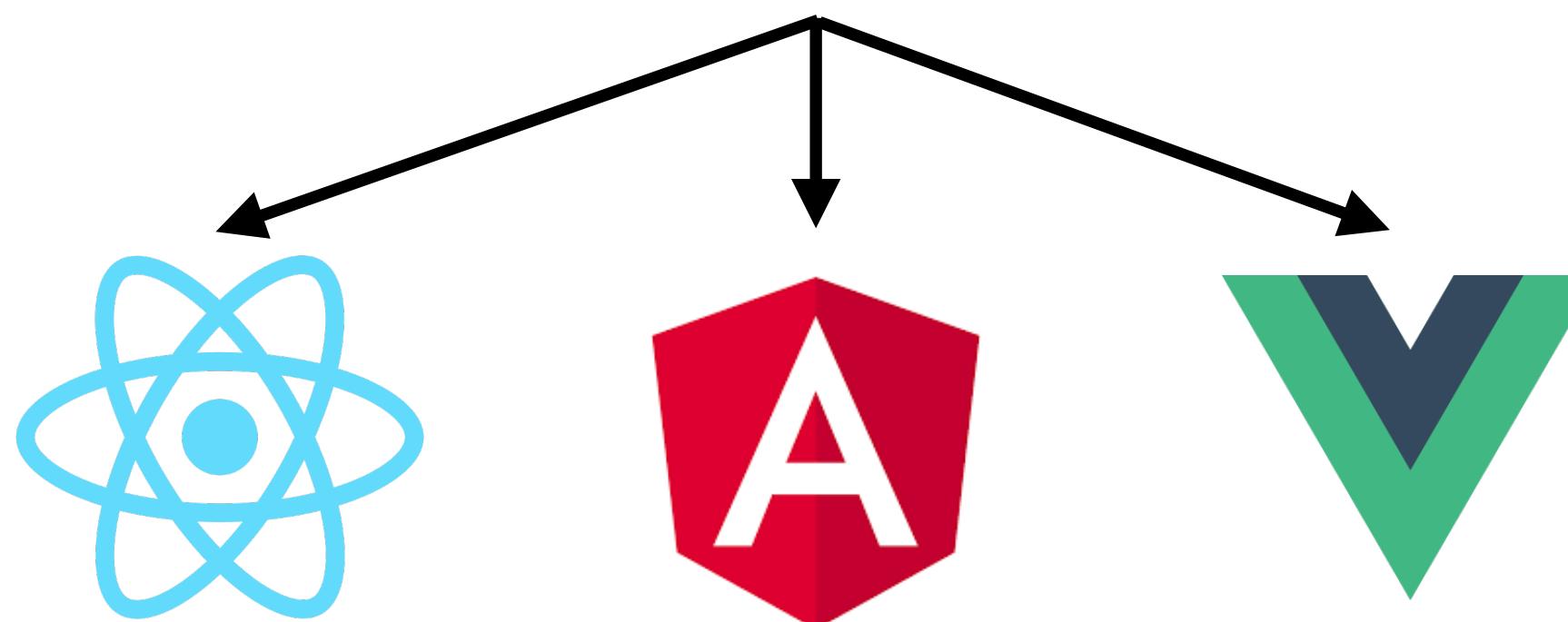

```

<https://naver.github.io/egjs-flicking/features/infiniteflicking.html>

## 7.3 CFC 사례

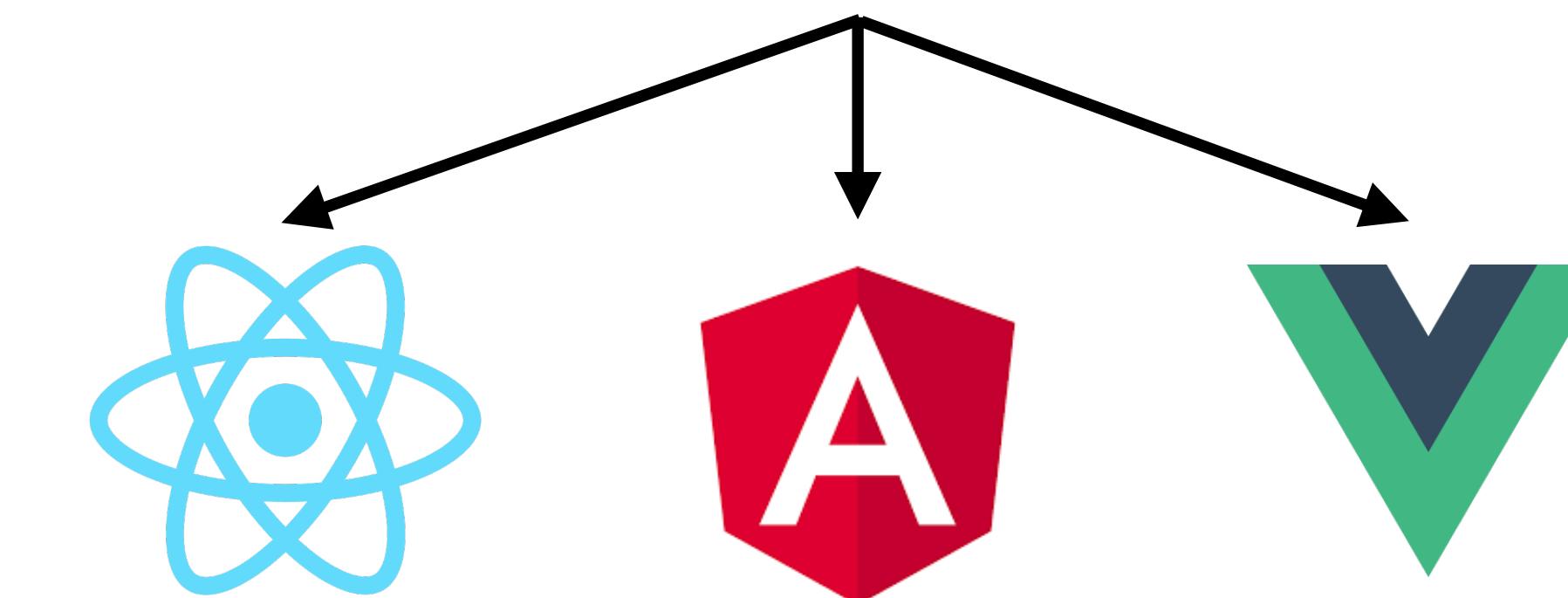


InfiniteGrid



<https://github.com/naver/egjs-infinitegrid>

< Flicking >  
...  
...



<https://github.com/naver/egjs-flicking>

**DEVIEW**  
**2019**



**egjs**

# Q & A

# Thank You