

REACT NATIVE SEOUL

리액트 구조 개론

(V.16.4 기준)

목차 / 참조

▶ State & Props

▶ REACT COMPONENT LIFECYCLE

- A. 생성부터 initial render까지
- B. props나 state가 바뀌었을 때의 re-rendering
- C. error-catch에 의한 re-rendering

▶ Figure 1: https://twitter.com/dan_abramov/status/981712092611989509

▶ Component Life cycle: <https://medium.com/@baphemot/understanding-reactjs-component-life-cycle-823a640b3e8d>

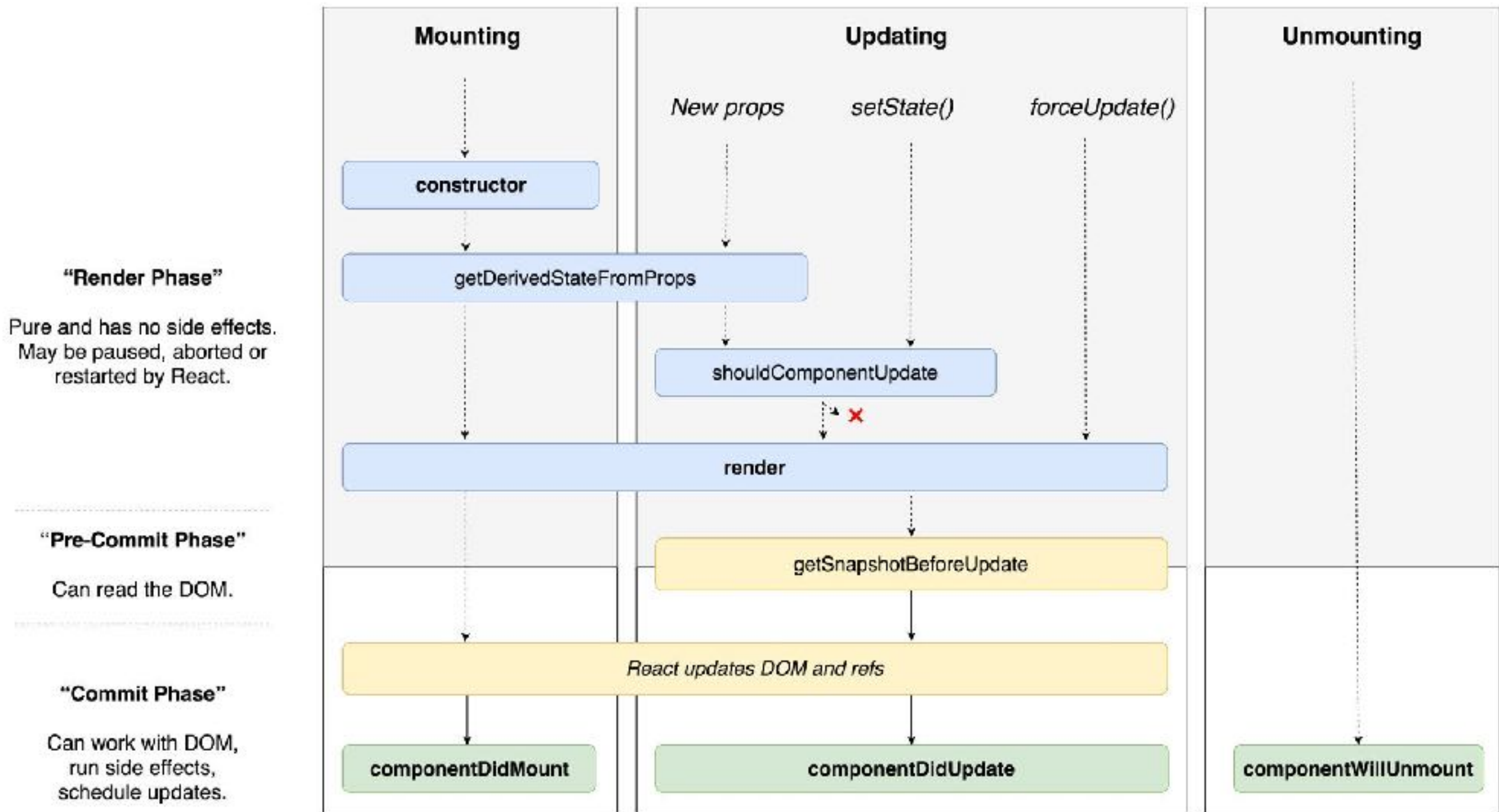


STATE & PROPS

- ▶ state: 컴포넌트 자기 자신이 가지고 있는 값. this.setState 명령으로 값을 변화시킴
- ▶ props: 부모 컴포넌트로부터 물려받는 값. read only
- ▶ props 나 state 값이 바뀌면 리렌더링 된다.
- ▶ state값을 let으로 주고 바꾸면 리렌더링 하지 않는다. 꼭 this.setState명령어로 업데이트 해야 라이프사이클 업데이트에 등록이 됨.

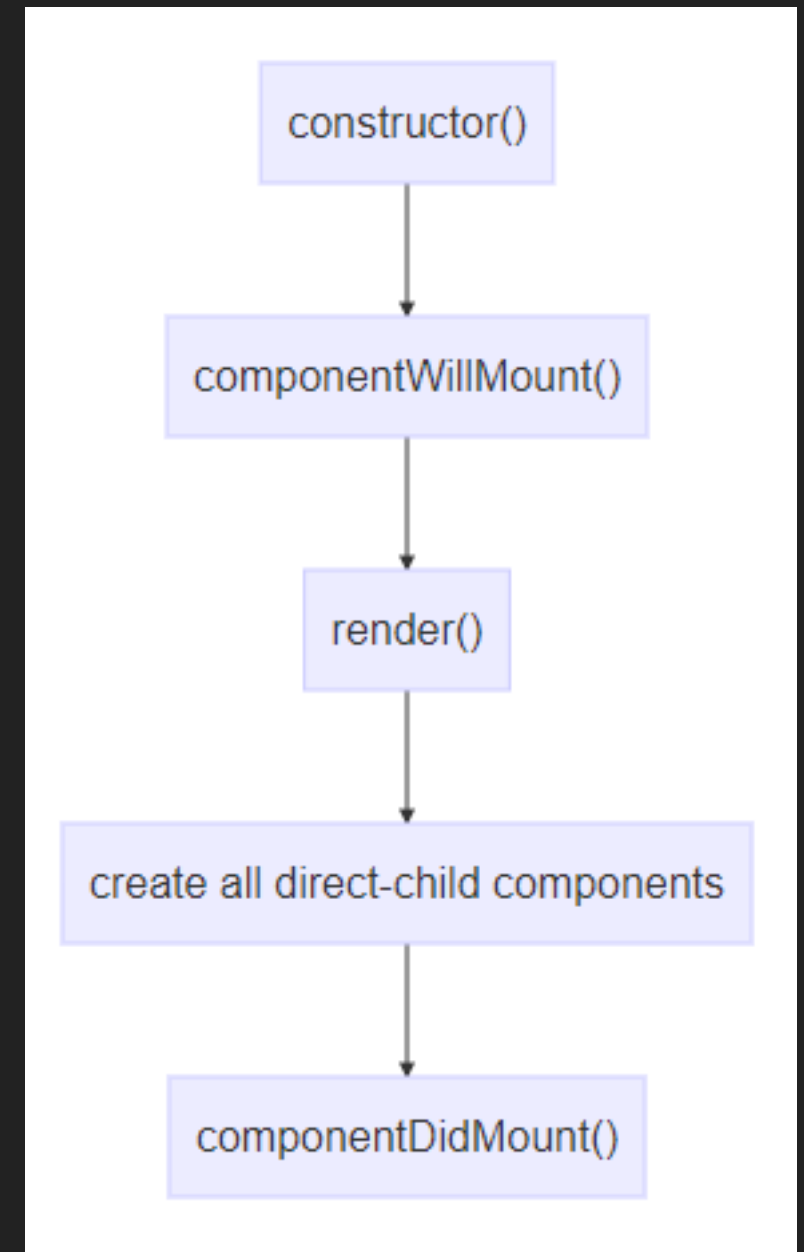
REACT COMPONENT LIFECYCLE

Figure 1



constructor()

- ▶ JavaScript Class 의 기본 constructor
- ▶ Do: state의 default값들을 넣는다
e.g. `this.state = {
 filterBy: ''
}`
- ▶ Don't: side effects e.g. AJAX call 등



UNSAFE_componentWillmount()

- ▶ constructor() 바로 다음에 호출됨.
- ▶ 전에는 여러가지 용도로(SSR등) 쓰였으나, React v.16 부터 쓰이는 React Fiber Architecture 때문에 렌더 전까지 여러번 호출될 가능성이 있어 사용을 권장하지 않음. -> 버전17에서 deprecate될 예정.

render()

▶ JSX/html을 화면에 렌더한다 (그린다)

▶ render목록:

1. React elements
2. Arrays and fragments
3. Portals
4. Strings and numbers
5. Booleans or null

▶ Do: 최대한 위의 목록들을 return만 하자. 가벼운 삼항식 조건을 줘도 괜찮지만 너무 복잡해지면 별도의 메소드로 따로 빼자.

▶ Don't: 복잡한 조건식, lambda

e.g. TextInput 컴포넌트에서:

No-no: `onChangeText={text => onChangeFunc(text)}`

Yiaass: `onChangeText={onChangeFunc}` [왜 이렇게 해야되죠?](#)

componentDidMount()

- ▶ Component로직을 가지고 첫번째 렌더한 후에만 발동된다.
- ▶ 해당 component가 unmount될때까지 딱 한번만 발동하는 메소드.
- ▶ Do: side effects e.g. AJAX call 등
- ▶ Don't: this.setState() 이러면 렌더하자마자 한번 더 렌더하기 때문인데, 사실 렌더하자마자 텍스트가 들어찬 컨테이너의 길이 등을 잴때까지 하는 함수들을 위해 this.setState()를 호출하는 다른 함수를 만들어서 그걸 componentDidMount()에 넣곤 했는데, (lint를 피하는 방법) 그렇지만 이제 다음페이지에 소개할 static getDerivedStateFromProps가 initial render후에도 동작이 되기 때문에 이것을 대신 쓰면 된다.

static getDerivedStateFromProps(props, state)

- ▶ 특징: 드럽게 이름이 길다, 새로운 라이프싸이클 메소드다. (그래서 major IDE들에도 아직 이 메소드는 autocomplete지원이 되는것을 보지 못했다.)
- ▶ 특징2: 첫번째 렌더링 후부터 쪽 렌더링 할때마다 호출된다. (그래서 static)
- ▶ 특징3: return 값을 무조건 줘야 한다. 일단 return null 세팅해놓고 뭘 할지 고민해보자. return 값은 state object 와 동일한 키 값을 사용하면 된다.

▶ e.g.

```
export default class App extends React.Component {  
  state = {  
    sampleBooleanState: false,  
    componentLoaded: false  
  }  
  
  static getDerivedStateFromProps(nextProps, prevState) {  
    if (nextProps.sampleBooleanProp !== prevState.sampleBooleanState) {  
      return {  
        sampleBooleanState: nextProps.sampleBooleanProp,  
        componentLoaded: true,  
      }  
    }  
    return null  
  }  
}
```

PROPS나 STATE가 바뀌었을 때의 RE-RENDERING

컴포넌트 PROPS가 바뀌었을 때

- ▶ e.g. 부모 컴포넌트 App의 filterBy state가 업데이트 될 때마다:

```
// 부모 컴포넌트
export default class App extends React.Component {
  state = {
    filterBy: ''
  }

  updateFilter = text => {
    if (text.trim() === '') {
      this.setState({ filterBy: '' })
    } else {
      this.setState({ filterBy: text })
    }
  }

  render() {
    const { filterBy } = this.state;
    return (
      <SafeAreaView style={styles.container}>
        <Text>Awesome React Frameworks/Packages!</Text>
        <TextInput
          style={styles.filterStyle}
          placeholder="필터를 입력해주세요."
          onChangeText={this.updateFilter}
        />
        <List filterBy={filterBy} />
      </SafeAreaView>
    );
  }
}
```

- ▶ 자식 컴포넌트인 List 가 받은 filterBy prop이 업데이트 되며 List는 re-rendering된다

```
const frameworks = [
  'React',
  'React-Native',
  'Proton-Native',
  'Redux',
  'MobX',
  'React-Router'
]

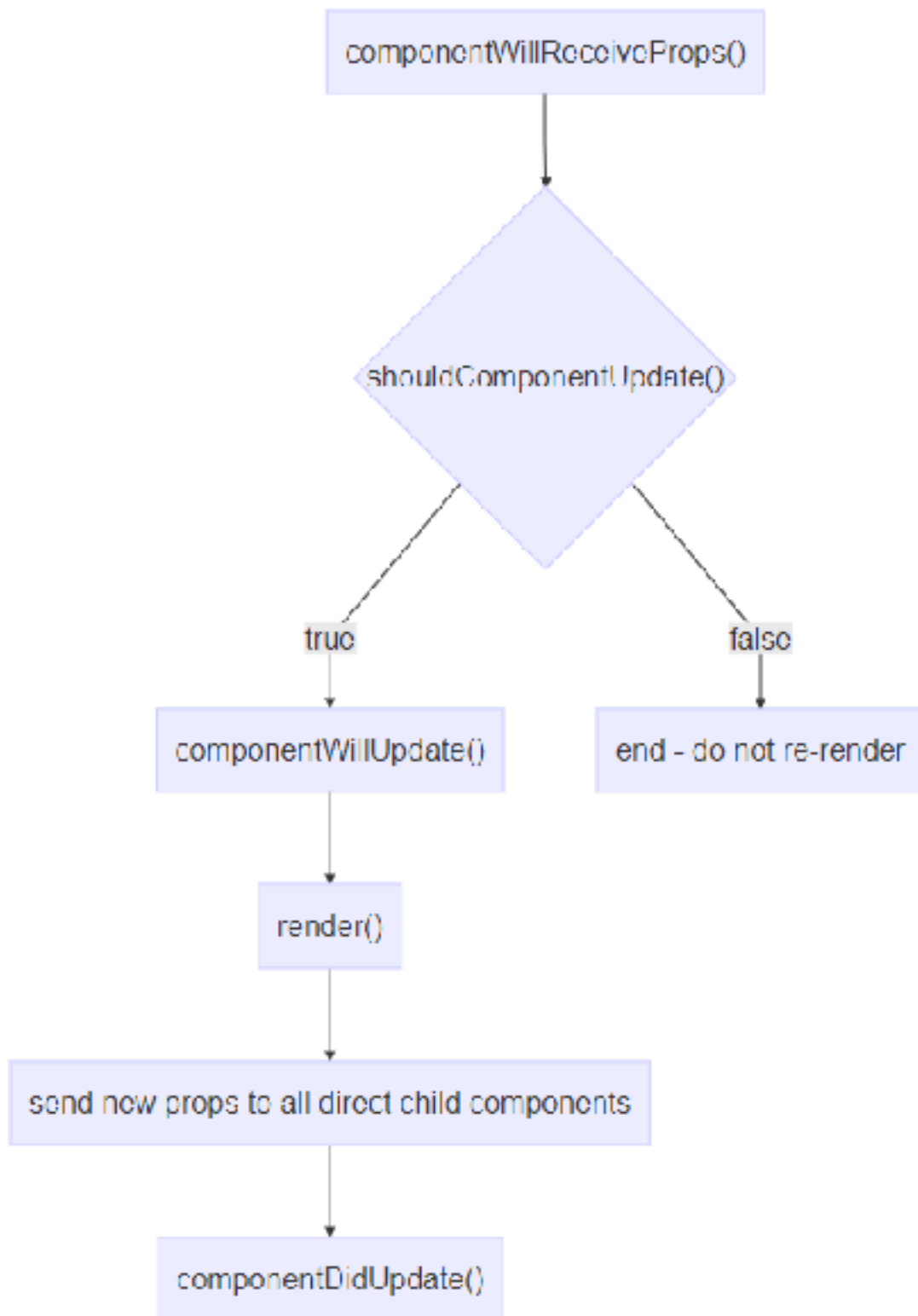
const _keyExtractor = item => item

const _renderItem = ({ item }) => (
  <Text style={styles.itemTextStyle}>
    {item}
  </Text>
)

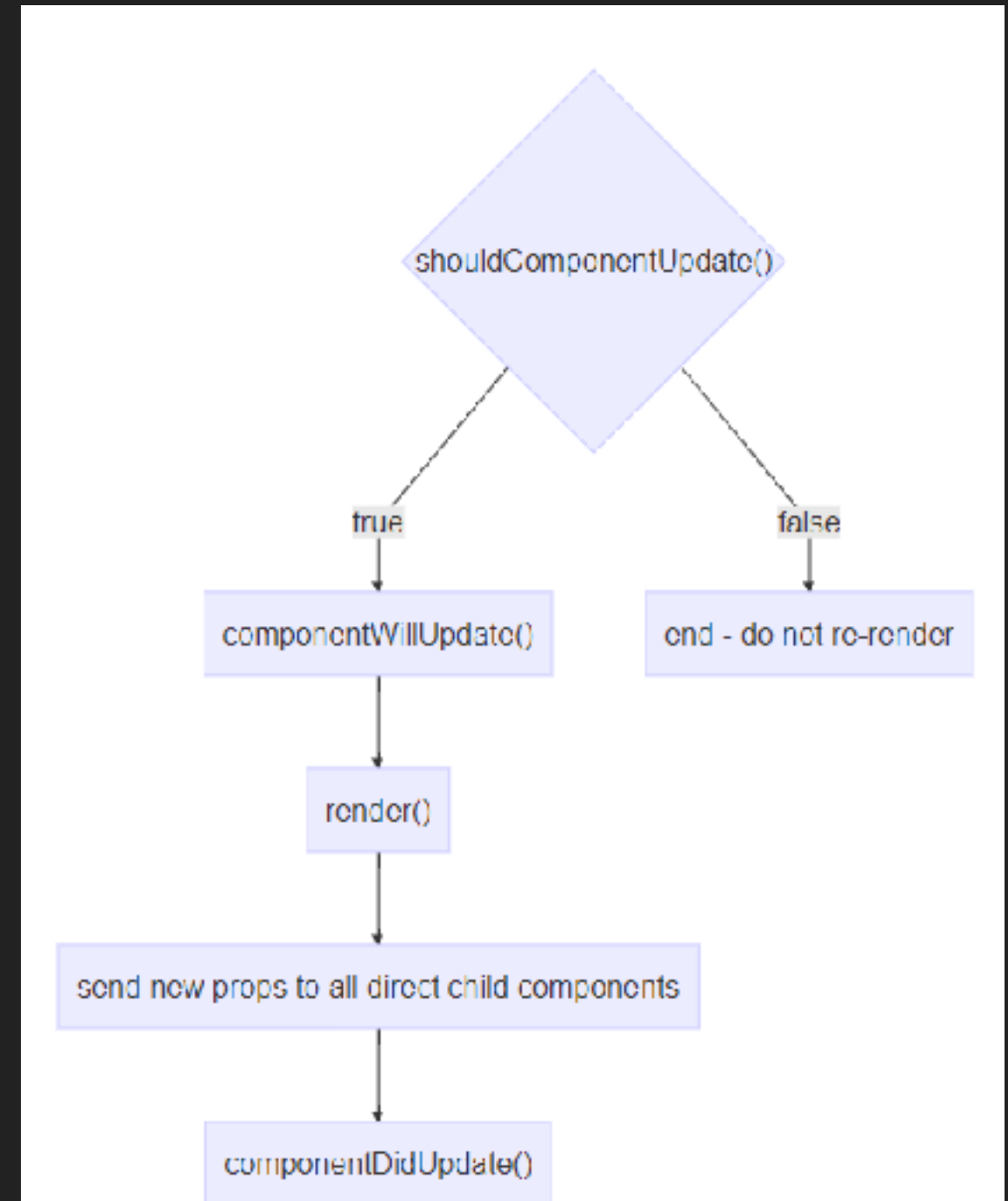
// 자식 컴포넌트
const List = ({ filterBy }) => {
  const filteredData = filterBy.length > 0
    ? frameworks.filter(item => item.indexOf(filterBy) > -1)
    : frameworks
  return (
    <FlatList
      contentContainerStyle={styles.filterContainer}
      data={filteredData}
      keyExtractor={_keyExtractor}
      renderItem={_renderItem}
    />
  )
}
```

PROPS나 STATE가 바뀌었을 때의 RE-RENDERING

- ▶ 부모 component가 준 props가 바뀌었을 때



- ▶ 컴포넌트 자체 내의 state가 바뀌었을 때



UNSAFE_componentWillReceiveProps(nextProps)

- ▶ prop이 바뀌었을 때 발동된다.
- ▶ 처음 렌더 되었을때는 발동되지 않는다 (componentDidMount(), static getDerivedStateFromProps(nextProps, prevState) 만 처음 렌더시에 발동됨)
- ▶ 아마도 리액트 개발자들이 가장 많이 사용하는 메소드 중 하나라고 생각된다. 리액트 측은 이 메소드에서 많은 버그가 발생하기 때문에 (버전17에서 deprecate 예정) 용도에 따라 static getDerivedStateFromProps(props, state) 나 componentDidUpdate(prevProps, prevState, snapshot) 사용을 권장하고 있다.

shouldComponentUpdate(nextProps, nextState)

- ▶ Only for performance optimization
- ▶ 만약 너무 많은 렌더링을 막으려면, 차라리 PureComponent를 사용하자
- ▶ return값은 boolean인데, return true를 기본으로 해 놓고 조건부로 false를 설정해 놓자. return false일때는 리렌더링이 되지 않는다.
- ▶ Do: 웬만하면 쓰지 말거나 PureComponent를 사용하자
- ▶ Don't: return값을 주지 않는것. return true/false는 꼭 줘야한다

UNSAFE_componentWillUpdate(nextProps, nextState)

- ▶ componentDidUpdate(prevprops, prevState, snapshot) 사용을 대신 권장하고 있다 (버전17에서 deprecate예정)
- ▶ 렌더 전의 어떤 값을 저장해서 렌더 후의 componentDidUpdate메소드에 넘겨줄 경우, 새로운 메소드 중 하나인 getSnapshotBeforeUpdate(prevProps, prevState)의 사용을 소개하고 있다.

componentDidUpdate(prevProps, prevState, snapShot)

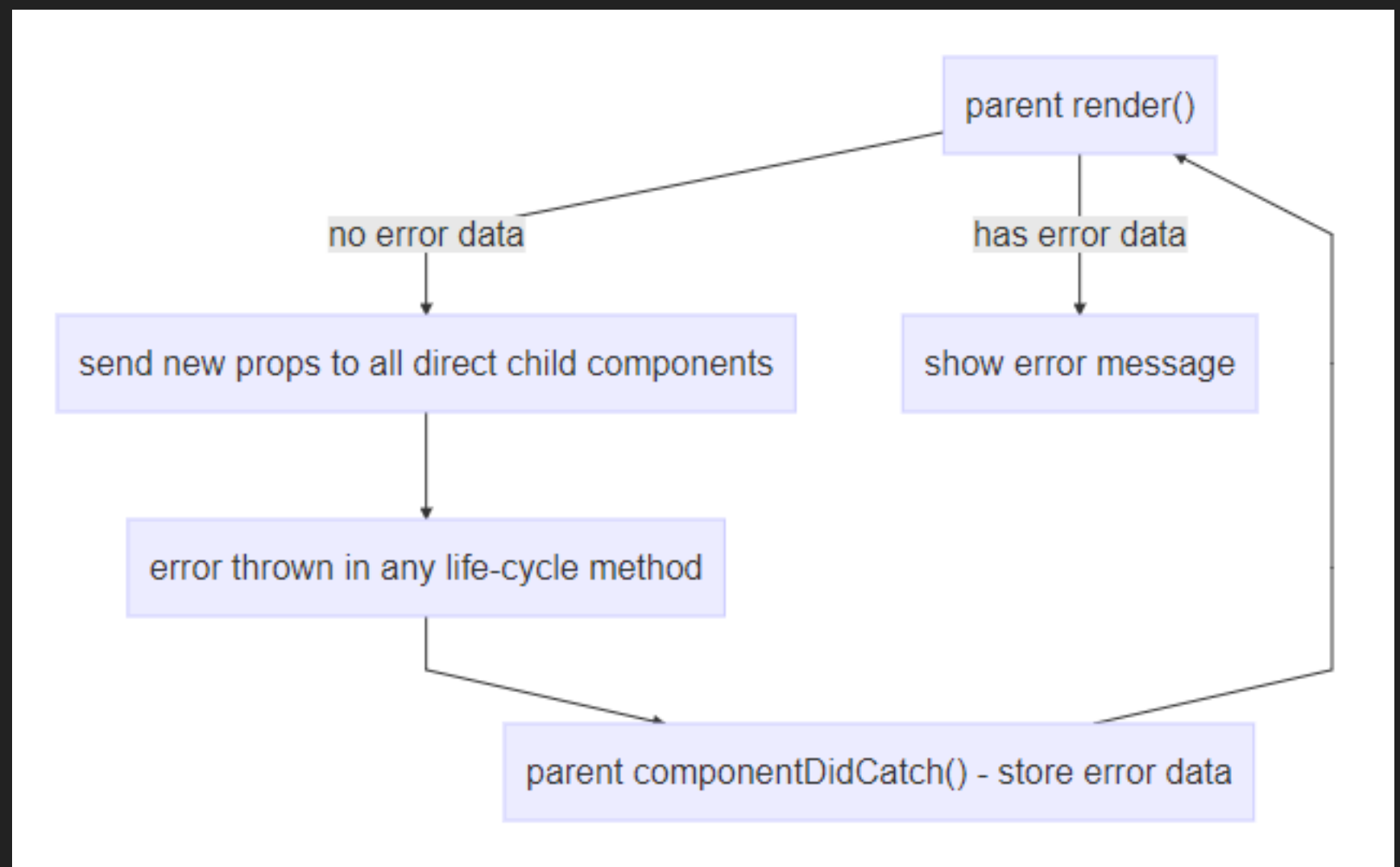
- ▶ 컴포넌트가 state 나 props 의 변화로 리렌더링 된 후에 발동되는 메소드
- ▶ 처음 렌더시에는 발동되지 않는다
- ▶ 세번째 인자인 snapshot은
getSnapshotBeforeUpdate(prevProps, prevState) 메소드의 리턴 값을 쓴다. 렌더 순서는 getSnapshotBeforeUpdate -> render -> componentDidUpdate이다
- ▶ Do: side effects e.g. AJAX calls
- ▶ Don't: this.setState() 왜냐면 렌더했는데 또 하게 되니까

getSnapshotBeforeUpdate(prevProps, prevState)

- ▶ 렌더 바로 전에 발동된다
- ▶ 리턴 값을 무조건 줘야한다. return null 줘놓고 시작하자. return type 은 아무거나 상관없다. 리턴된 값은 componentDidUpdate(prevProps, prevState, snapshot)의 세번째 인자에 들어가서 해당 메소드 사용시 쓸 수 있다
- ▶ 예시에는 데이터가 계속 들어오는 상태에서(트위터 피드나 페이스북 월) 자신이 보고 있는 부분의 scrollHeight 위치를 계속 유지시키게 하는 로직이다. (그렇지 않으면 새로 들어오는 데이터가 만들어내는 뷰만큼 내가 보고있던 스크롤 높이가 밀리게 될 것이므로)

componentDidCatch(error, info)

- ▶ 예시: error boundary라는 컴포넌트에 넣어서 자식 컴포넌트에서 발생하는 thrown error를 감지하여 this.setState로 에러가 발생한 자식 컴포넌트를 더이상 렌더하지 않게 해서 (대신에 에러 메시지를 띄우던가) 앱이 브레이크다운되는것을 막는다



감사합니다

- ▶ 옛 컴포넌트들의 라이프사이클에 대해서 좀 더 알고싶어요
- ▶ this.forceUpdate() 예제
- ▶ 10번 슬라이드의 filter 예제