REACT NATIVE SEOUL

리액트 라이프싸이클 개론

(V.16.4 기준)

목차 / 참조

- State & Props
- REACT COMPONENT LIFECYCLE
  A. 생성부터 initial render까지
  B. props나 state가 바뀌었을 때의 업데이트
  C. error-catch에 의한 업데이트

- 그림 1: http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/
- 컴포넌트 라이프싸이클: https://medium.com/@baphemot/understanding-reactjs-component-life-cycle-823a640b3e8d

STATE & PROPS

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

REACT COMPONENT LIFECYCLE

Figure 1

React v.16.4 기준

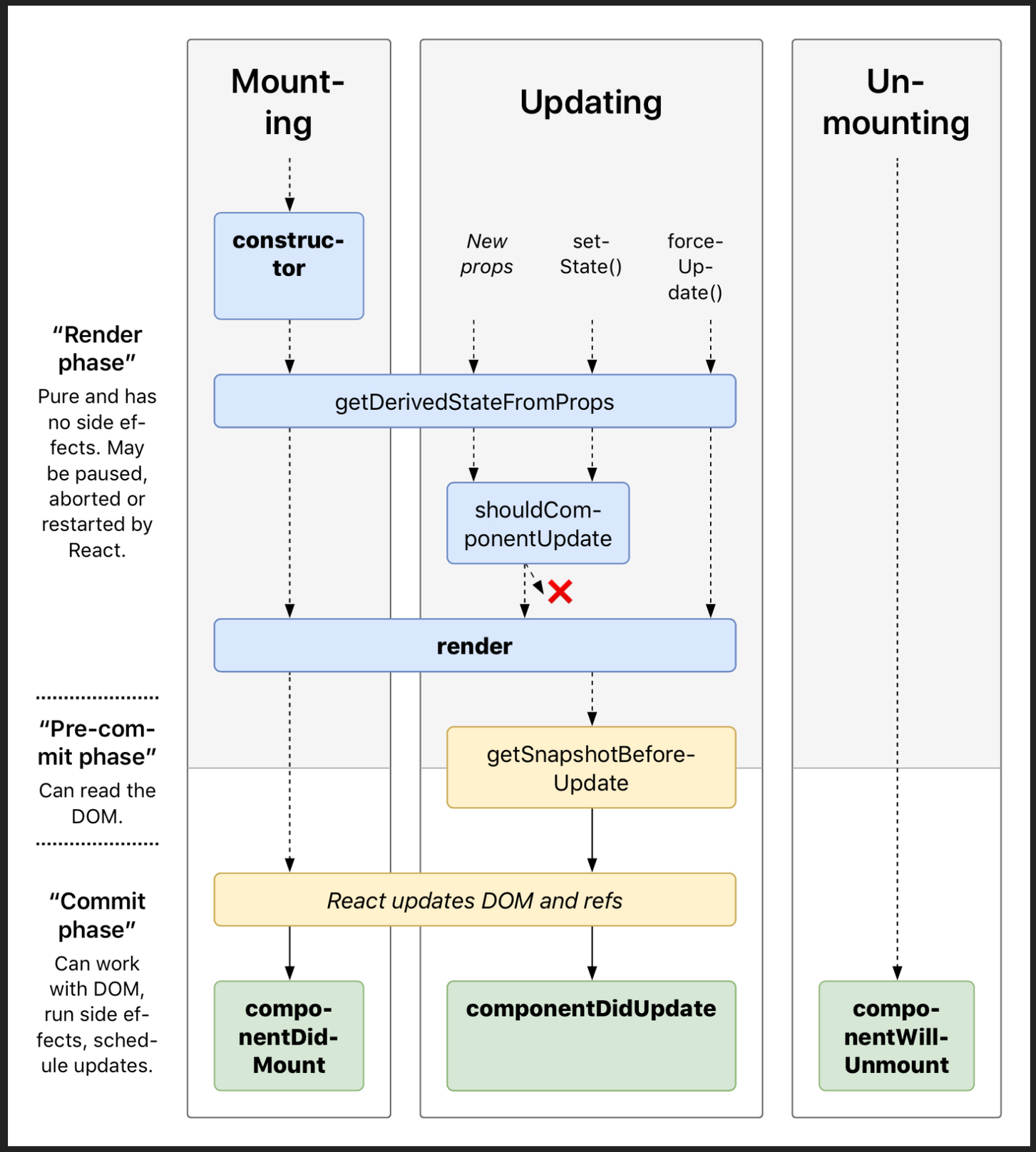
Virtual DOM -

DOM 에 직접 그리기 전에 접근 가능

Real DOM -

Virtual DOM과 비교(diff),

다른 부분만 업데이트



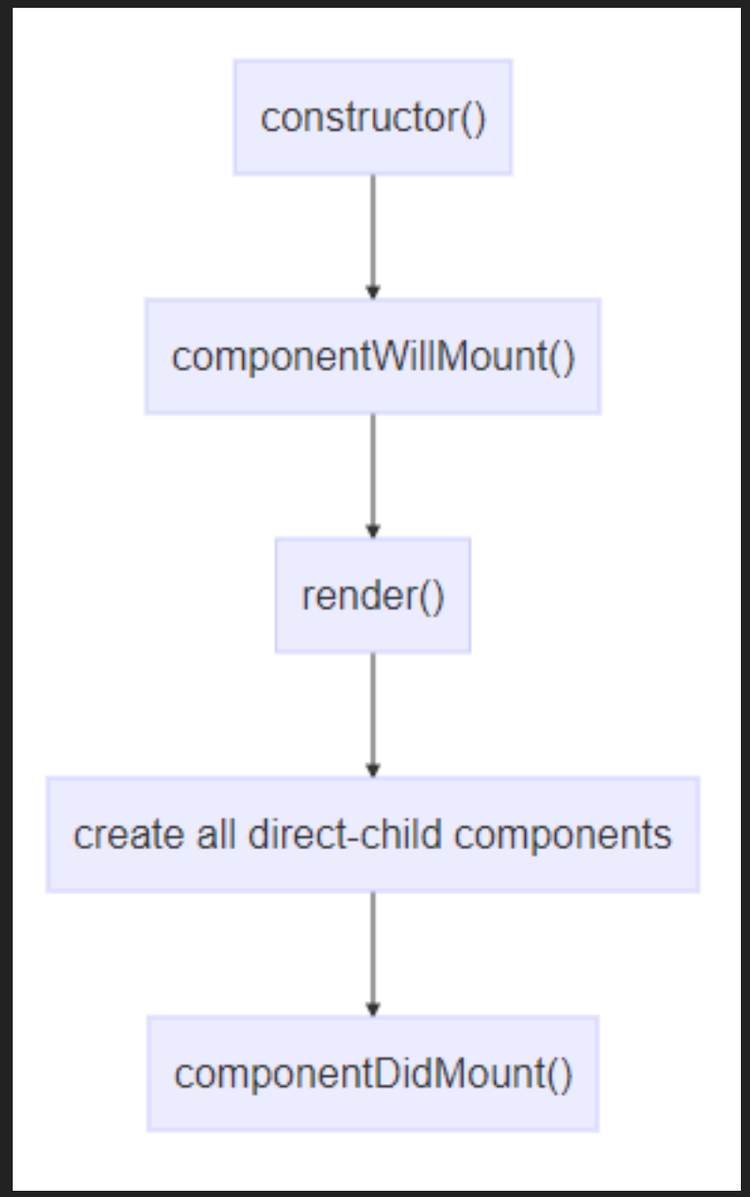
constructor()

- 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, inline functions

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()

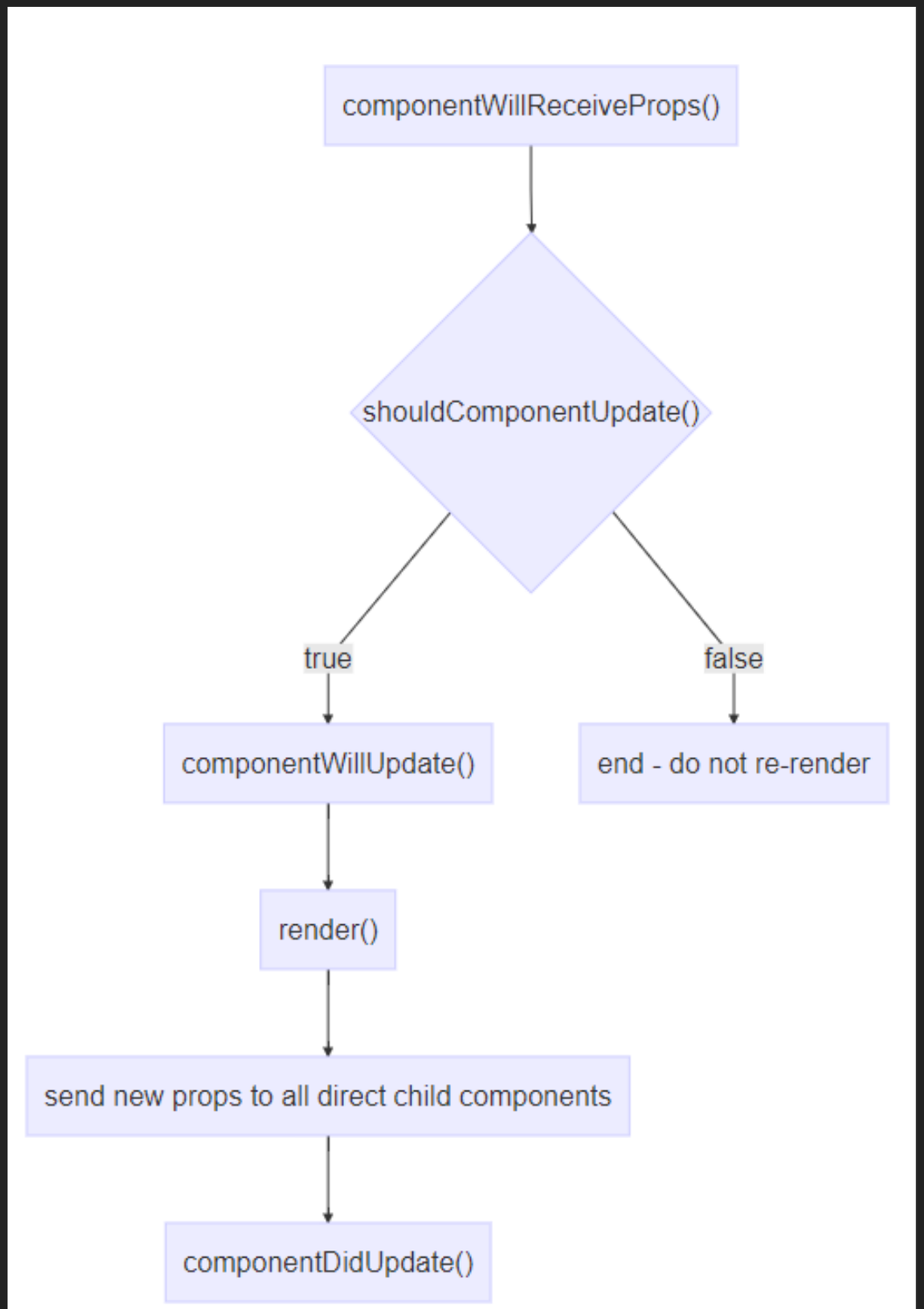
static getDerivedStateFromProps(props, state)

- 특징: 첫번째 렌더 후부터 쭉 렌더될때마다 호출된다. (static 메소드라서 this 사용불가)
- 특징2: return 값을 무조건 줘야 한다. 일단 return null 세팅해놓고 뭘 할지 고민해보자. return 값은 state object 와 동일한 키 값을 사용하면 된다.
- e.g.

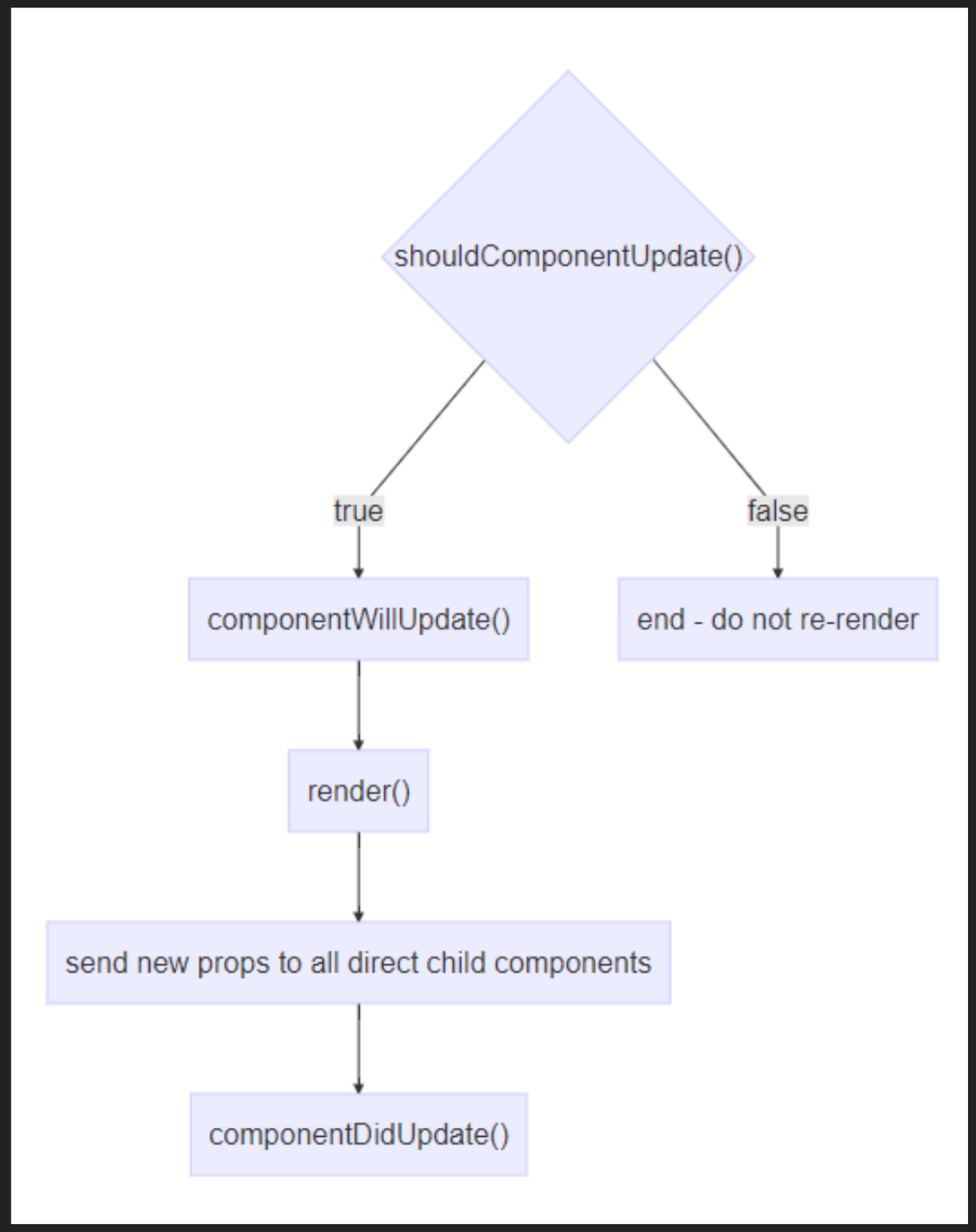
```
export default class App extends React.Component {
  state = {
    sampleBooleanState: false,
    componentLoaded: false
  }

  static getDerivedStateFromProps(nextProps, prevState) {
    if (nextProps.samepleBooleanProp !== prevState.sampleBooleanState) {
      return {
        sampleBooleanState: nextProps.samepleBooleanProp,
        componentLoaded: true,
      }
    }
    return null
  }
```

PROPS나 STATE가 바뀌었을 때의 업데이트

- 부모 component가 준 props가 바뀌었을 때



- 컴포넌트 자체 내의 state가 바뀌었을 때



컴포넌트 PROPS가 바뀌었을 때

- e.g. 부모 컴포넌트 App의 filterBy state가 업데이트 될 때마다:
- 자식 컴포넌트인 List 가 받은 filterBy prop이 업데이트 되며 List는 re-rendering된다

```
//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>
    );
  }
}
```

```
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 업데이트

컴포넌트 PROPS가 바뀌었을 때 ▸ 예제의 Initial mount 상황

FilterContainer(부모) 컴포넌트 시작 filterContainer.tsx:53

filterContainer.tsx:54

(index) ▲	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
FilterContainer	"constructor"	undefined	"{"filterBy":"","hasError":fal…	undefined	"{}"	undefined

▶ Object

FilterContainer(부모) 컴포넌트 업데이트 필요유무 체크 filterContainer.tsx:30

업데이트 필요없음 filterContainer.tsx:31

filterContainer.tsx:32

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
FilterContainer	"static getDerivedStateFromPro…	"{"filterBy":"","hasError":fa…	null	undefined	"{}"	undefined

▶ Object

FilterContainer(부모) 컴포넌트 렌더 filterContainer.tsx:185

filterContainer.tsx:186

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
FilterContainer	"render"	undefined	"{"filterBy":"","hasError":fal…	undefined	"{}"	undefined

▶ Object

List 자식 컴포넌트

List(자식) 컴포넌트 시작 list.tsx:88

list.tsx:89

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
List	"constructor"	undefined	"{"filterBy":"","filteredData"…	undefined	"{"filterBy":""}"	undefined

▶ Object

List(자식) 컴포넌트 업데이트 필요유무 체크 list.tsx:60

업데이트 필요없음 list.tsx:68

list.tsx:69

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
List	"static getDerivedStateFromPro…	"{"filterBy":"","filteredData…	"null"	undefined	"{"filterBy":""}"	undefined

▶ Object

List(자식) 컴포넌트 렌더 list.tsx:180

list.tsx:181

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
List	"render"	undefined	"{"filterBy":"","filteredData"…	undefined	"{"filterBy":""}"	undefined

▶ Object

List(자식) 컴포넌트 마운트됨 list.tsx:123

list.tsx:124

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
List	"componentDidMount"	undefined	"{"filterBy":"","filteredData"…	undefined	"{"filterBy":""}"	undefined

▶ Object

FilterContainer(부모) 컴포넌트 마운트됨 filterContainer.tsx:109

filterContainer.tsx:110

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
FilterContainer	"componentDidMount"	undefined	"{"filterBy":"","hasError":fal…	undefined	"{}"	undefined

▶ Object

PROPS나 STATE가 바뀌었을 때의 업데이트

컴포넌트 PROPS가 바뀌었을 때 ▸ filter에 'x'라고 타이핑한 상황

FilterContainer(부모) 컴포넌트 업데이트 시작 — filterContainer.tsx:72, filterContainer.tsx:74

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
FilterContainer	"shouldComponentUpdate"	"{"filterBy":"","hasError":fa…	"{"filterBy":"x","hasError":fa…	"{}"	"{}"	undefined

▶ Object

FilterContainer(부모) 컴포넌트 렌더 — filterContainer.tsx:185, filterContainer.tsx:186

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
FilterContainer	"render"	undefined	"{"filterBy":"x","hasError":fa…	undefined	"{}"	undefined

▶ Object

List(자식) 컴포넌트 업데이트 필요유무 체크 — list.tsx:60

List 자식 컴포넌트

업데이트 필요 — list.tsx:68, list.tsx:69

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
List	"static getDerivedStateFromPro…	"{"filterBy":"","filteredData…	"{"filterBy":"x","filteredData…	undefined	"{"filterBy":"x"}"	undefined

▶ Object

List(자식) 컴포넌트 업데이트 시작 — list.tsx:107, list.tsx:109

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
List	"shouldComponentUpdate"	"{"filterBy":"","filteredData…	"{"filterBy":"x","filteredData…	"{"filterBy":""}"	"{"filterBy":"x"}"	undefined

▶ Object

List(자식) 컴포넌트 렌더 — list.tsx:180, list.tsx:181

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
List	"render"	undefined	"{"filterBy":"x","filteredData…	undefined	"{"filterBy":"x"}"	undefined

▶ Object

list.tsx:137

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
List	"getSnapshotBeforeUpdate"	"{"filterBy":"","filteredData…	"{"filterBy":"x","filteredData…	"{"filterBy":""}"	"{"filterBy":"x"}"	null

▶ Object

FilterContainer 부모 컴포넌트의 getSnapshotBeforeUpdate

filterContainer.tsx:123

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
FilterContainer	"getSnapshotBeforeUpdate"	"{"filterBy":"","hasError":fa…	"{"filterBy":"x","hasError":fa…	"{}"	"{}"	null

▶ Object

List(자식) 컴포넌트 렌더됨 — list.tsx:151, list.tsx:152

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
List	"componentDidUpdate"	"{"filterBy":"","filteredData…	"{"filterBy":"x","filteredData…	"{"filterBy":""}"	"{"filterBy":"x"}"	"null"

▶ Object

FilterContainer(부모) 컴포넌트 렌더됨 — filterContainer.tsx:137, filterContainer.tsx:138

(index)	LifeCycle Hook Method	prevState	nextState	prevProps	nextProps	snapshot
FilterContainer	"componentDidUpdate"	"{"filterBy":"","hasError":fa…	"{"filterBy":"x","hasError":fa…	"{}"	"{}"	"null"

▶ Object

UNSAFE_componentWillReceiveProps(nextProps)

- prop이 바뀌었을 때 발동된다.
- 처음 렌더 되었을때는 발동되지 않는다 (componentDidMount() 와 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 용도에 따라 static getDerivedStateFromProps(props, state) 나 componentDidUpdate(prevProps, prevState, snapshot) 사용을 권장하고 있다. (버전17에서 deprecate 예정)

shouldComponentUpdate(nextProps, nextState)

- Only for performance optimization
- 첫번째 렌더에서는 작동하지 않음
- 만약 너무 많은 렌더링을 막으려면, 차라리 PureComponent를 사용하자
- return값은 boolean인데, return true를 기본으로 해 놓고 조건부로 false를 설정해 놓자. return false일때는 해당 컴포넌트가 리렌더링이 되지 않는다.
- Do: 왠만하면 쓰지 말거나 PureComponent를 사용하자
- Don't: return값을 설정하지 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 componentDidUpdate에 소드에 넘겨줄 경우, 새로운 메소드 중 하나인 getSnapshotBeforeUpdate(prevProps, prevState)의 사용을 소개하고 있다.

componentDidUpdate(prevProps, prevState, snapShot)

- 컴포넌트가 state 나 props 의 변화로 업데이트 된 후에 발동되는 메소드
- 처음 렌더시에는 발동되지 않는다
- 세번째 인자인 snapshot은 getSnapshotBeforeUpdate(prevProps, prevState) 메소드의 리턴 값을 가져온다. 렌더 순서는 getSnapshotBeforeUpdate -> render -> componentDidUpdate이다
- Do: side effects e.g. AJAX calls
- Don't: this.setState(). 렌더직후의 렌더를 바로 또 하게 되는 현상 방지

getSnapshotBeforeUpdate(prevProps, prevState)

- 업데이트 바로 전에 발동된다
- 리턴 값을 설정해야한다. return null 줘놓고 시작하자. return type에 제약은 없다. 리턴된 값은 componentDidUpdate(prevProps, prevState, snapshot)의 세번째 인자에 들어가서 해당 메소드 사용 시 쓸 수 있다
- 예시는 데이터 피드가 계속 들어오는 상태에서(트위터 피드나 페이스북 월) 자신이 보고 있는 부분의 scrollHeight위치를 계속 유지시키게 하는 로직이다. (그렇지 않으면 새로 들어오는 데이터가 만들어내는 뷰의 높이만큼 내가 보고있던 스크롤 높이가 밀리게 될 것이므로)

componentWillUnmount()

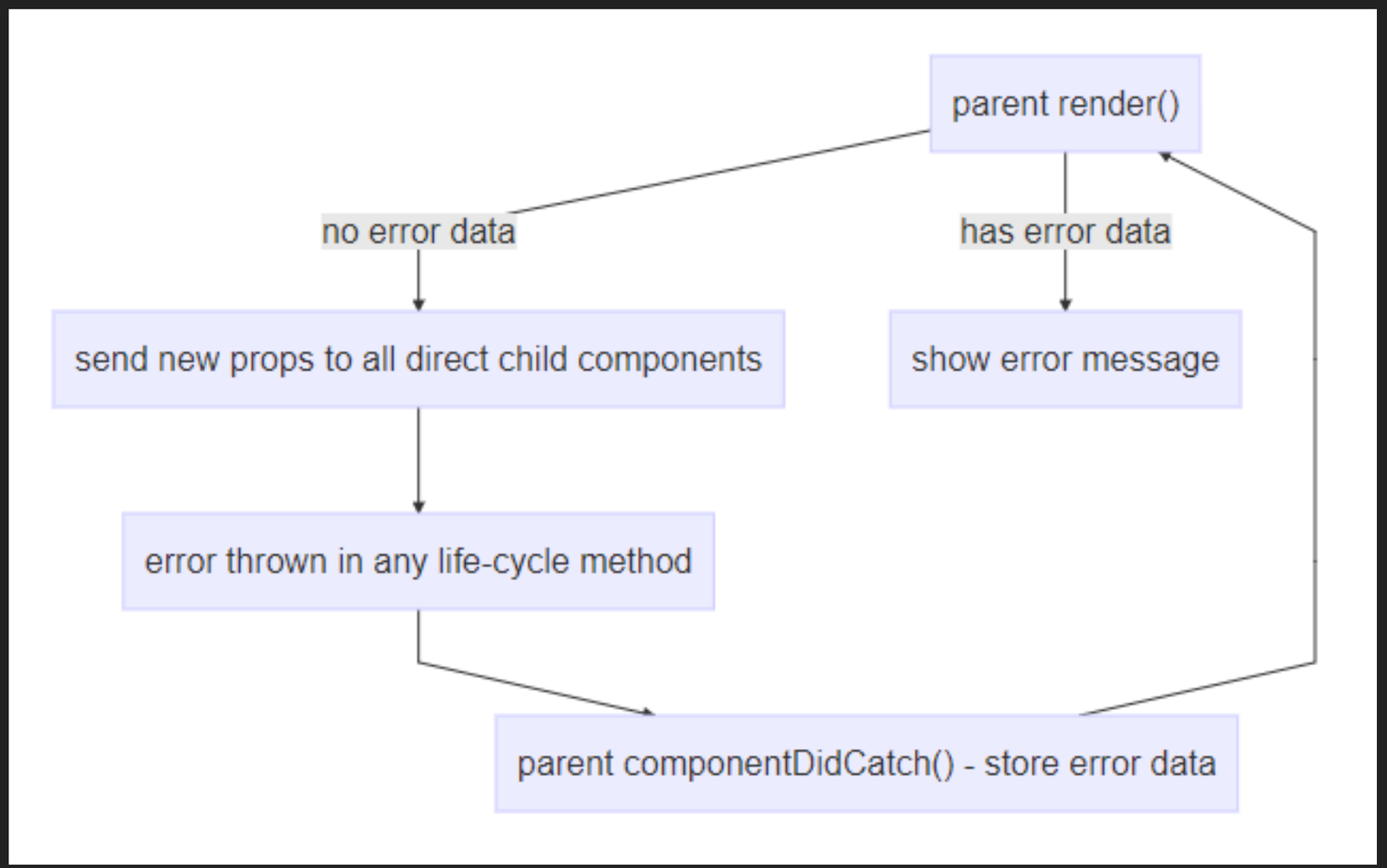
- 컴포넌트가 DOM에서 사라지기 직전에 실행되는 메소드
- do: unsubscribe, removeEventListener, clearInterval, etc.
- don't: this.setState 등 컴포넌트가 계속 마운트된 상태로 유지되어 있을꺼라고 생각하고 하는 작업들

componentDidCatch(error, info)

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

- 자기 자신의 에러는 감지하지 못하고, 클래스여야만 동작하는 메소드다.

(그래서 꼭 에러 바운더리 부모 클래스를 만들어 자식 컴포넌트의 에러를 감지하게끔 예시에서 스타일가이드를 주고 있다)



감사합니다

- 옛 컴포넌트 라이프싸이클에 대해서 좀 더 알고싶어요
- this.forceUpdate() 예제
- 10번 슬라이드의 filter 예제
- 이 슬라이드의 강의 Youtube link