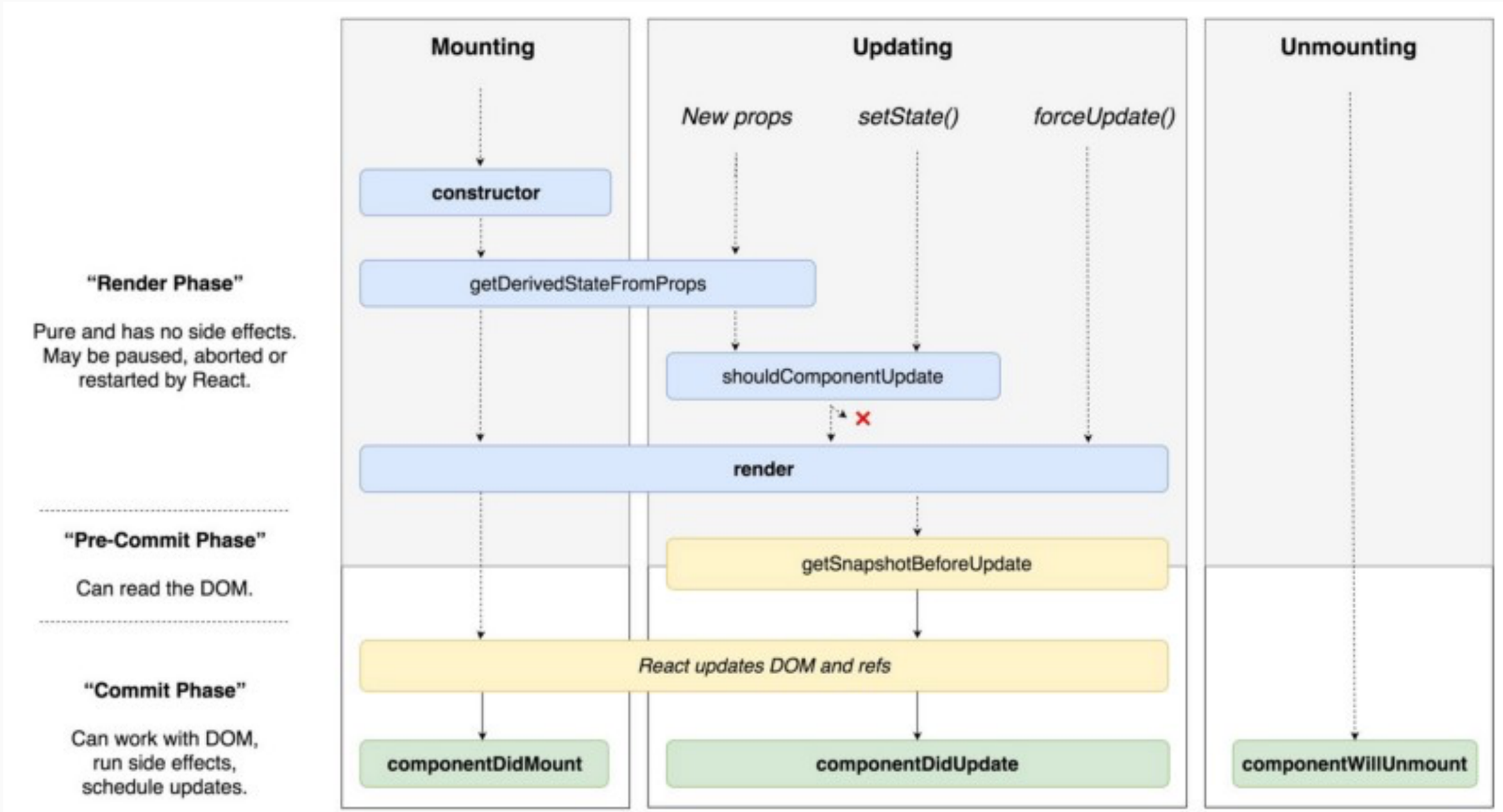
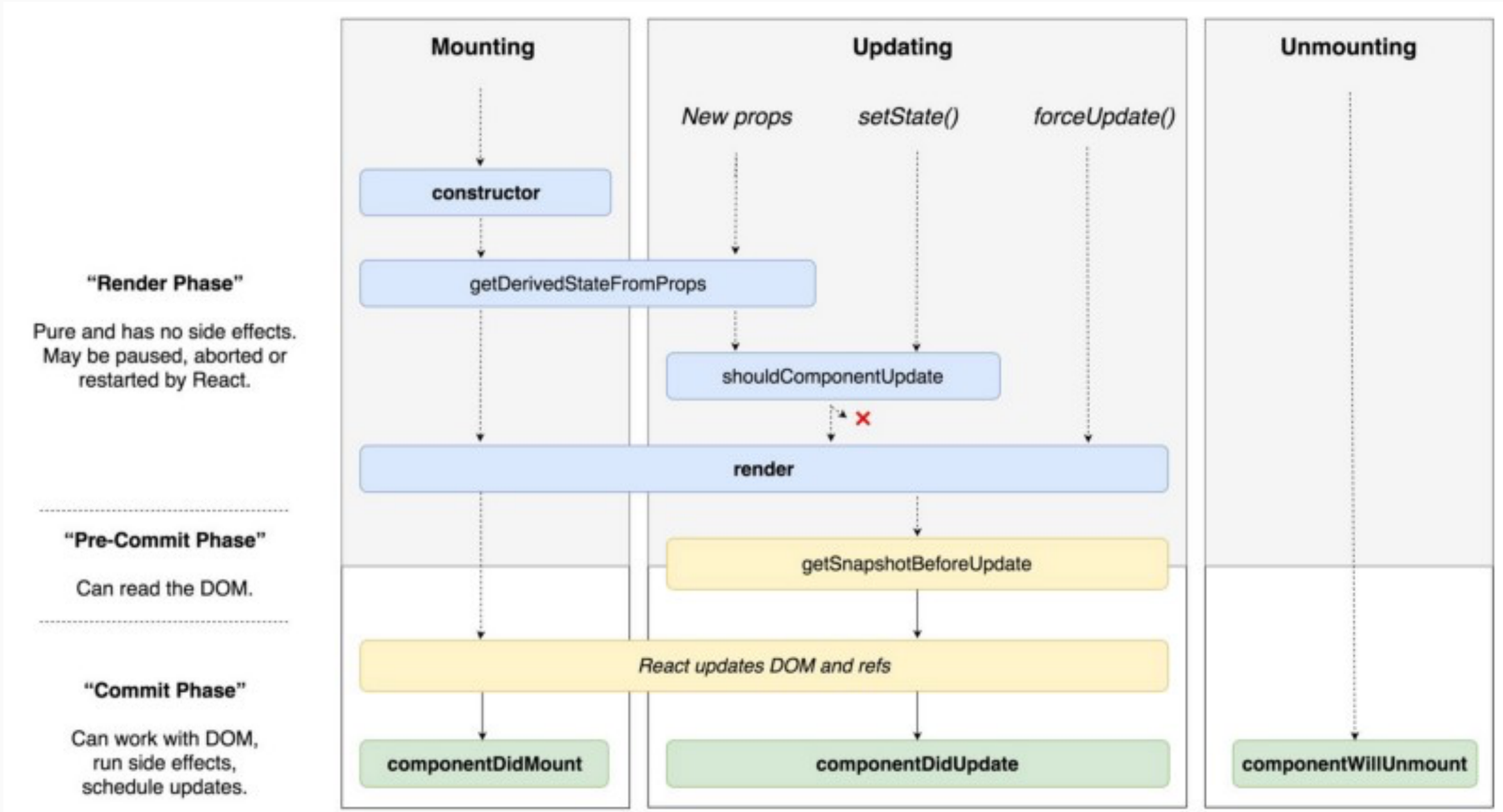
Методы жизненного цикла


"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.
Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount

constructor

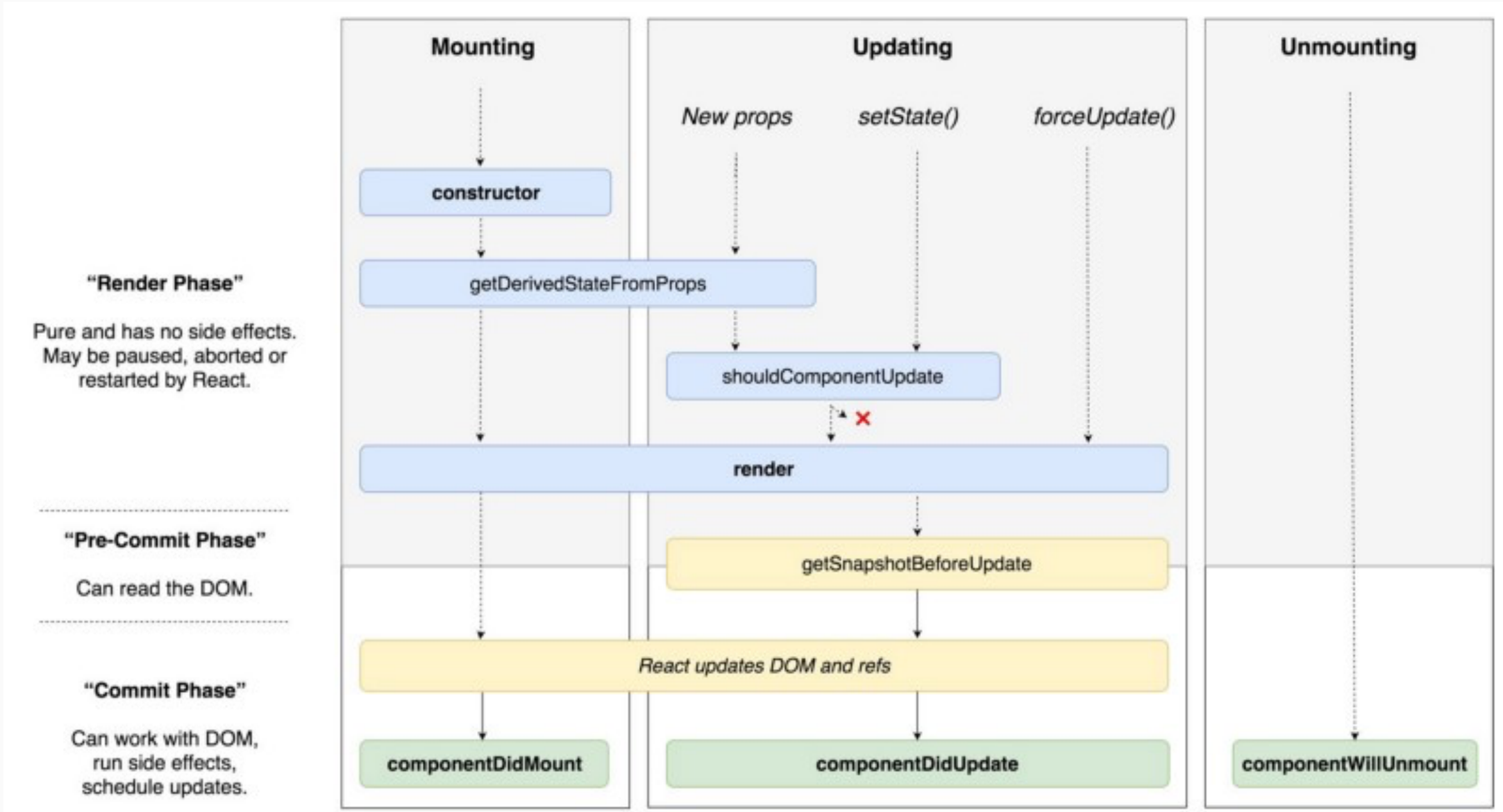
- Вызывается перед первым рендером

- Новые стандарты языка позволяют не использовать его при реализации React компонентов


"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.
Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount

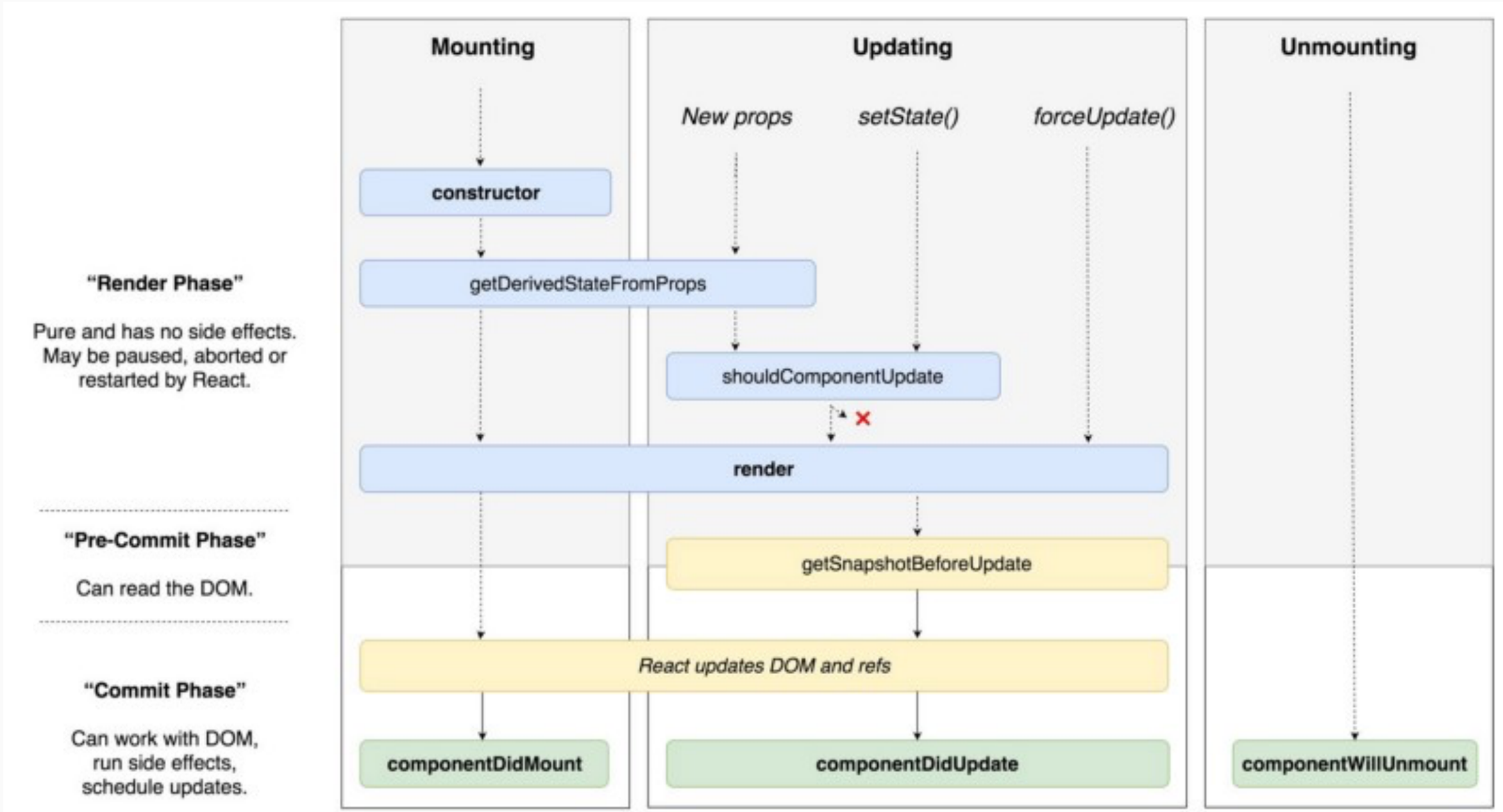
shouldComponentUpdate

- Вызывается перед каждым рендером, кроме первого
- Принимает два аргумента - пропсы с предыдущего шага и пропсы с текущего шага
- Должен возвращать true или false - нужно ли компоненту перерендерится


"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.
Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount

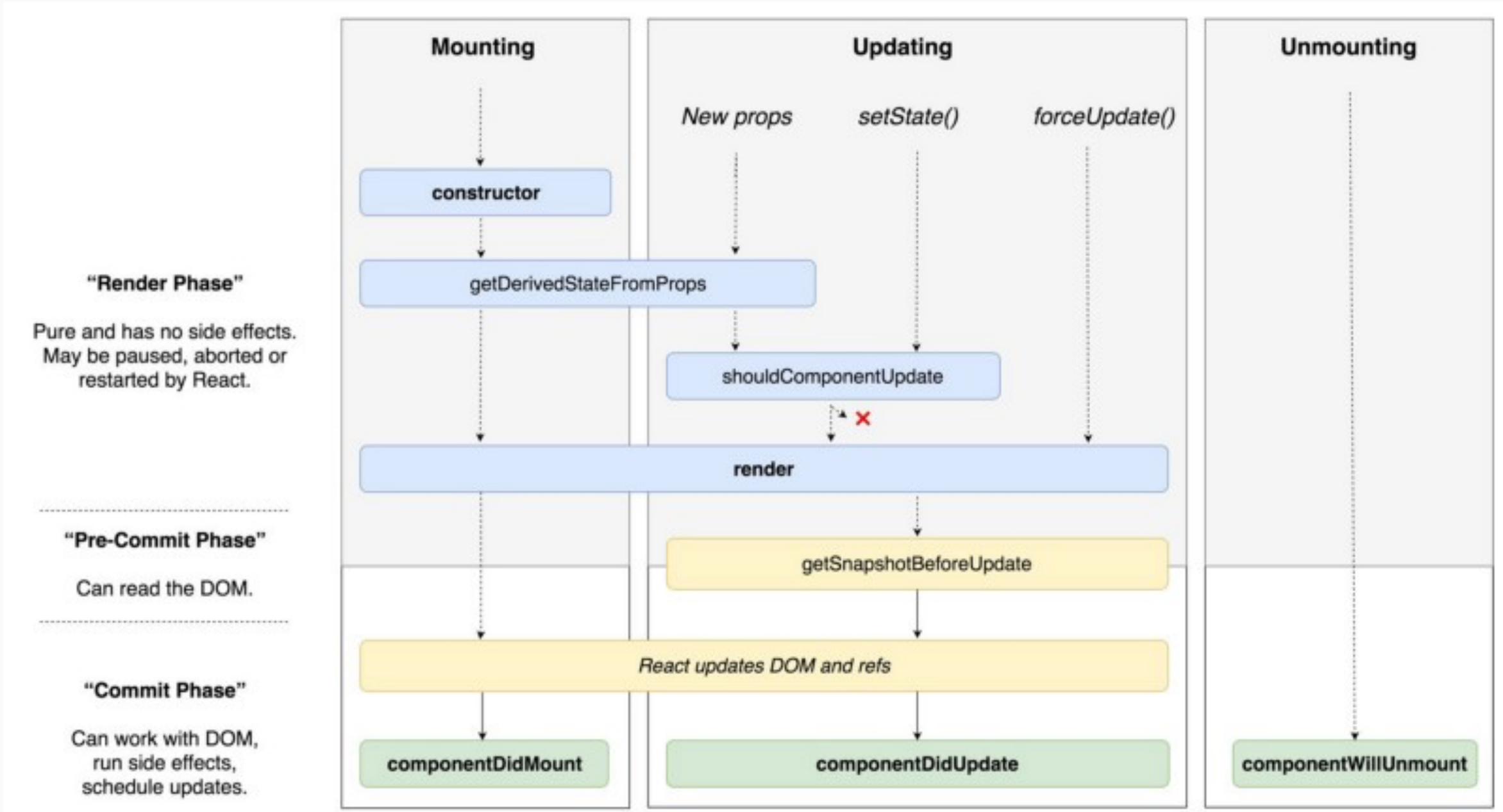
componentDidMount

- Нет аргументов
- Вызывается только после первого рендера
- Самый часто используемый метод жизненного цикла
- В нем инициируются запросы за данным на backend, создаются различные eventListener’ы и тд.


"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.
Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount

componentDidUpdate

- Вызывается после каждого ренедера кроме первого
- Принимает два аргумента - пропсы с предыдущего шага и пропсы с текущего шага
- Используется для применения сайд-эффектов после обновления компонента (новый запрос за данными, применение анимаций и тд)


"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.
Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount

componentWillUnmount

- Не принимает аргументов

- Используется для удаления таймеров, интервалов, закрытия веб-сокетов, removeEventListener и тд

```
const Example extends React.Component {
  state = {
    message: ''
  }

  updateMessage = () => {
    this.setState({ message: 'Updated message' })
  }

  componentDidMount() {
    setTimeout(updateMessage, 5000)
  }

  render() {
    return (
      <div>Your message: {this.state.message}</div>
    )
  }
}
```

setTimeout(someFunction, waitTime)

Функция someFunction будет вызвана спустя waitTime миллисекунд

```
const someFunction = () => {
  // do smth
}

setTimeout(someFunction, 5000) // waitTime === 5 секунд

setTimeout(() => {
  // do smth
}, 3000)
```

setInterval(someFunction, waitTime)

Функция someFunction будет вызвана через каждые waitTime миллисекунды

```
const someFunction = () => {
  // do smth
}

setInterval(someFunction, 5000) // waitTime === 5 секунд

setInterval(() => {
  // do smth
}, 3000)
```

Реализация электронных часов

```
const date = new Date();

setInterval(foo, timeInMilliseconds);

setInterval(() => { /* do smth */ }, 1000);

// life cycle method
componentDidMount() { /* do smth */ }
```

Inputs

```
class InputExample extends React.Component {
  state = {
    inputValue: ''
  }

  handleInputChange = event => {
    const { value } = event.target

    this.setState({ inputValue: value })
  }

  render() {
    return (
      <input
        value={this.state.inputValue}
        onChange={this.handleInputChange}
      />
    )
  }
}
```

Передача ф-ии в кач-ве пропса

```jsx
const MyBeautifulInput = ({
  value,
  onChange,
  placeholder = 'Type in me!'
}) => (
  <input
    value={value}
    onChange={onChange}
    placeholder={placeholder}
  />
)

class InputExample extends React.Component {
  // same as in previous example
  render() {
    return (
      <MyBeautifulInput
        value={this.state.inputValue}
        onChange={this.handleInputChange}
      />
    )
  }
}
```

Обмен данными между компонентами

- «Сверху-вниз» - через пропсы
- «Снизу-вверх» - родитель должен передать ребенку функцию, в теле которой меняется стейт. Ребенок вызвав эту функцию меняет стейт у родителя. Таким образом родитель может узнать о каких-либо эвентах/изменениях в ребенке

Spread operator

Создает новый массив или объект, совершая **неглубокое** копирование элементов

```
const originalArray = [1234, 4321, 5678]
const copyArray = [...original]

const originalObject = {
  key: 'value'
}

const copyObject = {
  ...originalObject
}
```

Spread operator

Создает новый массив или объект, совершая **неглубокое** копирование элементов

```
const original = {
  key1: 'value',
  nested: {
    key2: 1234
  }
}

const copy = {
  ...original,
  nested: {
    ...original.nested
  }
}
```

Хранение в state массива и его изменение

```
class App extends React.Component {
  state = {
    collection: [1, 2, 3, 4]
  }

  addFive = () => {
    // this.state.collectioin.push(5) НЕЛЬЗЯ!!!
    this.setState(currentState => {
      const newCollection = [...currentState.collection, 5]

      return {
        collection: newCollection
      }
    })
  }

  render() {
    return (
      <div>
        <ul>
          {this.state.collection.map(item => <li>{item}</li>)}
        </ul>
        <button onClick={this.addFive}>ADD FIVE</button>
      </div>
    )
  }
}
```