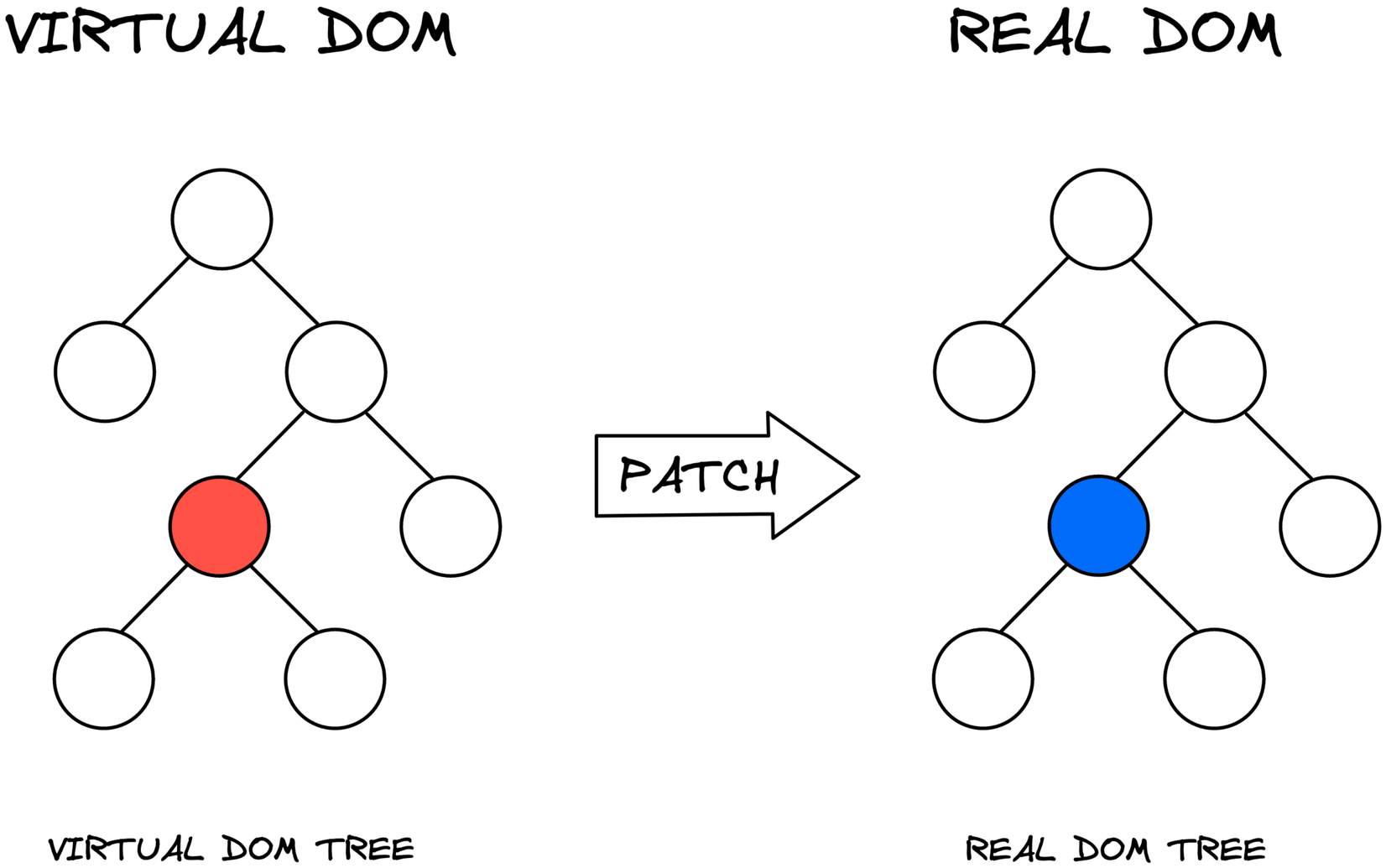
1. Монтирование (mount) вызывает рендеринг приложения.
2. Получившийся DOM вставляется в реальный DOM целиком, так как там еще ничего нет. А виртуальный DOM, в свою очередь, сохраняется внутри React для последующего обновления.
3. Изменение состояния приводит к вычислению нового виртуального DOM.
4. Вычисляется разница между старым виртуальным DOM и новым.
5. Разница применяется к реальному DOM.



"Render Phase"

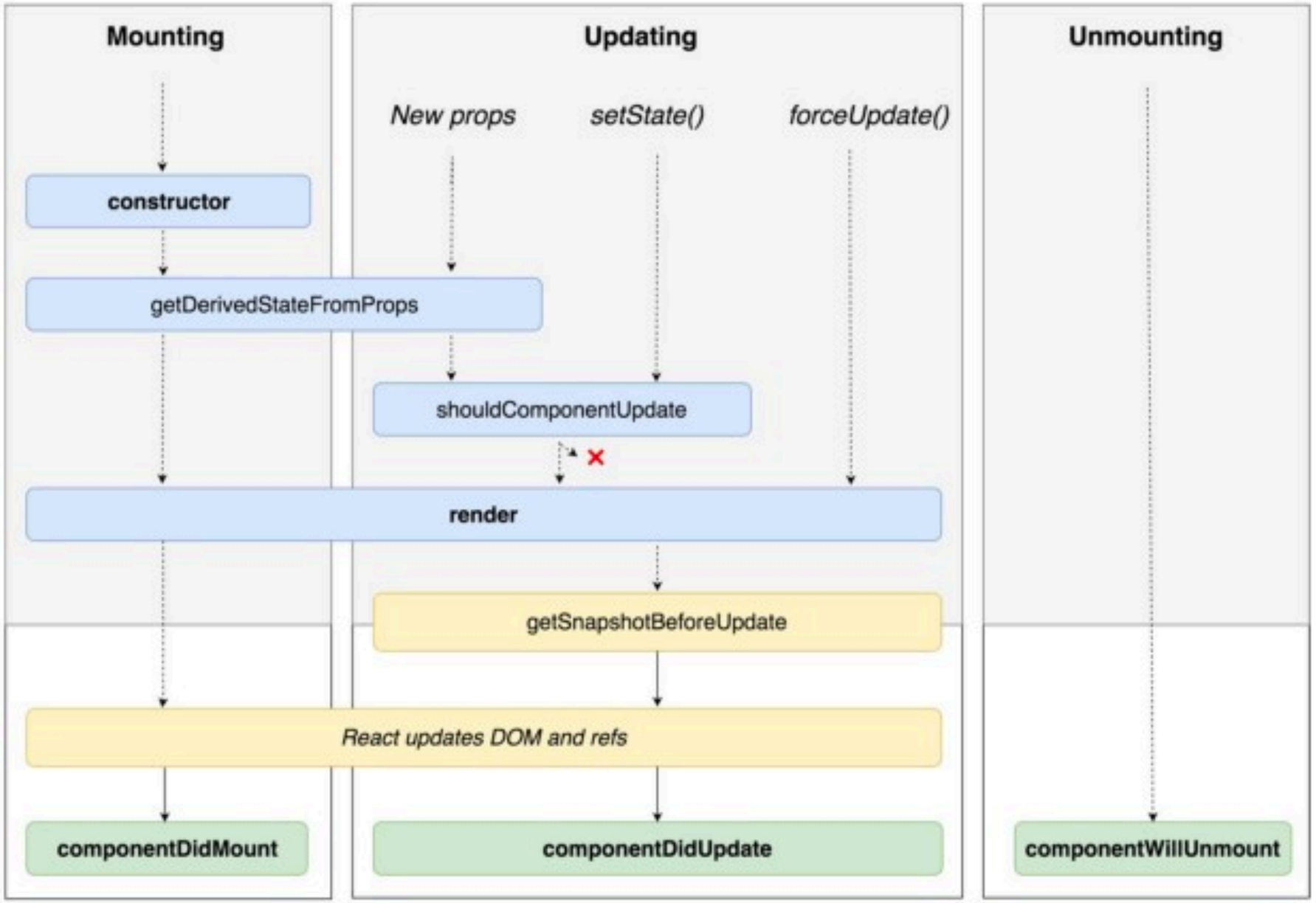
Pure and has no side effects. May be paused, aborted or restarted by React.

"Pre-Commit Phase"

Can read the DOM.

"Commit Phase"

Can work with DOM, run side effects, schedule updates.



useState

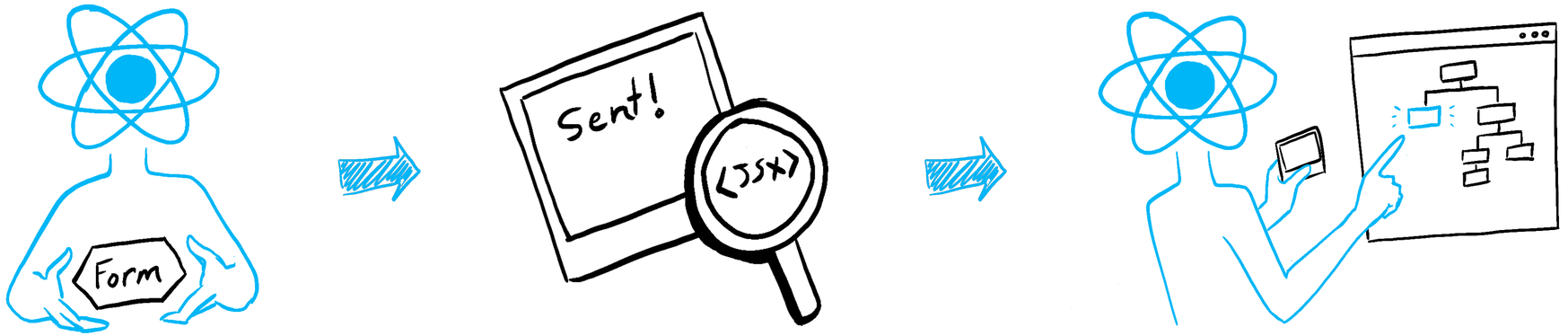
useState — это хук React, который позволяет вам добавить переменную состояния в ваш компонент.

```
const [state, setState] = useState(initialState);
```

Параметры

InitialState: значение, которое состояние будет иметь изначально. Это может быть значение любого типа, но для функций существует особое поведение. Этот аргумент игнорируется после начального рендеринга.

Rendering takes a snapshot in time



useContext

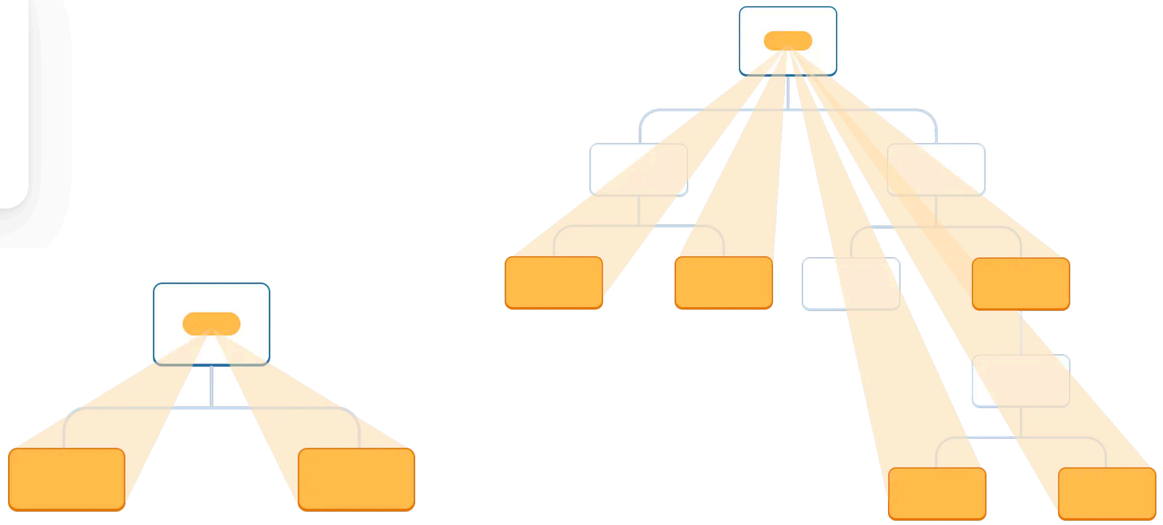
1. Контекст позволяет компоненту предоставлять некоторую информацию всему дереву под ним.
Чтобы передать контекст:
 - Создайте и экспортируйте его с помощью export const MyContext = createContext(defaultValue).
 - Передайте его хуку useContext(MyContext), чтобы прочитать его в любом дочернем компоненте, независимо от того, насколько он глубок.
 - Оберните дочерние элементы в <MyContext.Provider value={...}>, чтобы предоставить их от родителя.
2. Контекст проходит через любые компоненты в середине.
3. Контекст позволяет вам писать компоненты, которые «адаптируются к своему окружению».
4. Прежде чем использовать контекст, попробуйте передать реквизиты или передать JSX в качестве дочерних элементов.

Параметры

SomeContext: контекст, который вы ранее создали с помощью createContext. Сам контекст не содержит информацию, он представляет только ту информацию, которую вы можете предоставить или прочитать из компонентов.

```
import { useContext } from 'react';

function MyComponent() {
  const theme = useContext(ThemeContext);
  // ...
```



useCallback

useCallback — это React Hook, который позволяет кэшировать определение функции между повторными рендерингами.

```
const cachedFn = useCallback(fn, dependencies)
```

Параметры

fn: значение функции, которое вы хотите кэшировать. Он может принимать любые аргументы и возвращать любые значения. React вернет (не вызовет!) вашу функцию обратно вам во время первоначального рендеринга. При следующем рендеринге React снова предоставит вам ту же функцию, если зависимости не изменились с момента последнего рендеринга. В противном случае он даст вам функцию, которую вы передали во время текущего рендеринга, и сохранит ее на случай, если ее можно будет повторно использовать позже. React не будет вызывать вашу функцию. Функция возвращается вам, чтобы вы могли решить, когда и нужно ли ее вызывать.

dependencies: список всех реактивных значений, на которые ссылается код fn.

```
import { useCallback } from 'react';

export default function ProductPage({ productId, referrer, theme }) {
  const handleSubmit = useCallback((orderDetails) => {
    post('/product/' + productId + '/buy', {
      referrer,
      orderDetails,
    });
  }, [productId, referrer]);
```

useEffect

useEffect — это React Hook, который позволяет синхронизировать компонент с внешней системой.

```
useEffect(setup, dependencies?)
```

Параметры

setup: Функция с логикой вашего Эффекта. Ваша функция настройки также может дополнительно возвращать cleanup функцию. Когда ваш компонент впервые добавляется в DOM, React запустит вашу функцию настройки. После каждого повторного рендеринга с измененными зависимостями React сначала запускает функцию очистки (если вы ее предоставили) со старыми значениями, а затем запускает функцию настройки с новыми значениями. После того, как ваш компонент будет удален из DOM, React в последний раз запустит вашу функцию очистки.

необязательные зависимости: список всех реактивных значений, на которые ссылается код установки. Реактивные значения включают реквизиты, состояние и все переменные и функции, объявленные непосредственно внутри тела вашего компонента.

```
import { useEffect } from 'react';
import { createConnection } from './chat.js';

function ChatRoom({ roomId }) {
  const [serverUrl, setServerUrl] = useState('https://localhost:1234');

  useEffect(() => {
    const connection = createConnection(serverUrl, roomId);
    connection.connect();
    return () => {
      connection.disconnect();
    };
  }, [serverUrl, roomId]);
  // ...
}
```

useMemo

useMemo — это React Hook, который позволяет кэшировать результат вычислений между повторными рендерингами.

```
const cachedValue = useMemo(calculateValue, dependencies)
```

Параметры

calculateValue: функция, вычисляющая значение, которое вы хотите кэшировать. Она должна быть чистой, не должна принимать аргументов и должна возвращать значение любого типа. React вызовет вашу функцию во время начального рендеринга. При следующем рендеринге React снова вернет то же значение, если зависимости не изменились с момента последнего рендеринга. В противном случае он вызовет calculateValue, вернет результат и сохранит его, чтобы его можно было повторно использовать позже.

зависимости: список всех реактивных значений, на которые есть ссылки внутри кода calculateValue. Реактивные значения включают реквизиты, состояние и все переменные и функции, объявленные непосредственно внутри тела вашего компонента.

```
import { useMemo } from 'react';

function TodoList({ todos, tab }) {
  const visibleTodos = useMemo(
    () => filterTodos(todos, tab),
    [todos, tab]
  );
  // ...
}
```