React

Кирилл Талецкий

TeachMeSkills
24 сентября 2023

Основы

Основы

- Компоненты и пропсы
- JS внутри JSX
  - Рендер данных
  - Условный рендер
  - Рендер списков
- Обработка событий

Состояние

Компоненты с состоянием

- Главное выгодное отличие компонент в React от тех, что мы писали в проекте по JS — возможность задать компонентам состояние

- Состояние компонент может изменяться со временем
  - Если рендер зависит от состояния, то компонента будет перерисована

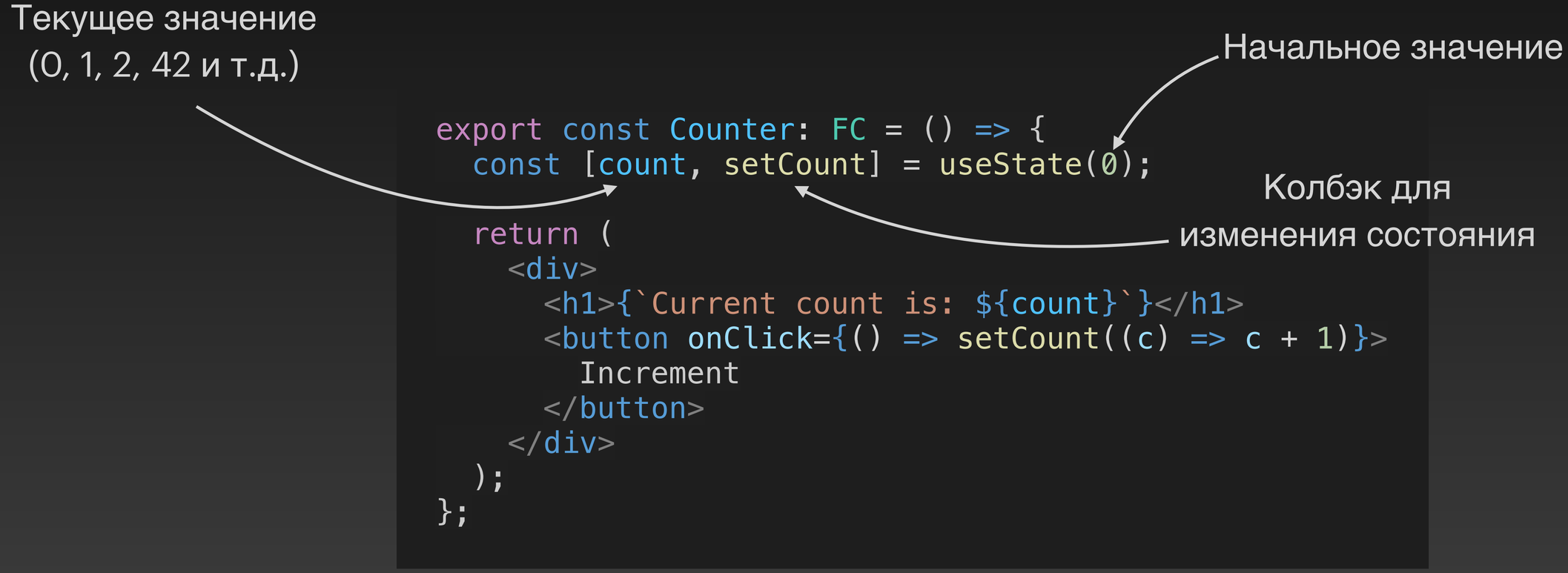
- Состояние нельзя мутировать напрямую! Только через специальную функцию

Хуки в React

- В функциональных компонентах за встроенный функционал отвечают специальные функции — хуки
- Основные хуки React мы рассмотрим на следующем занятии
- Всё, что нам сейчас нужно знать
  - Хуки — это обычные функции
  - Хуки предоставляемые реактом дают дополнительный функционал для компонент
  - Все хуки принято именовать с префиксом `use`, например
    - Хуки из React: useState(), useEffect(), useMemo(), useReducer() и т.д.
    - Примеры кастомных хуков: useViewportSize(), useAnimatedStyles() и т.д.

useState()

- Позволяет хранить и изменять состояние внутри компоненты



useState()

```tsx
export const Counter: FC = () => {
  const [count, setCount] = useState(0);

  return (
    <div>
      <h1>{`Current count is: ${count}`}</h1>
      <button onClick={() => setCount((c) => c + 1)}>
        Increment
      </button>
    </div>
  );
};
```

Показываем текущее значение пользователю

По нажатию на кнопку:

```tsx
setCount((c) => c + 1)}
```

Берём самое последнее значение

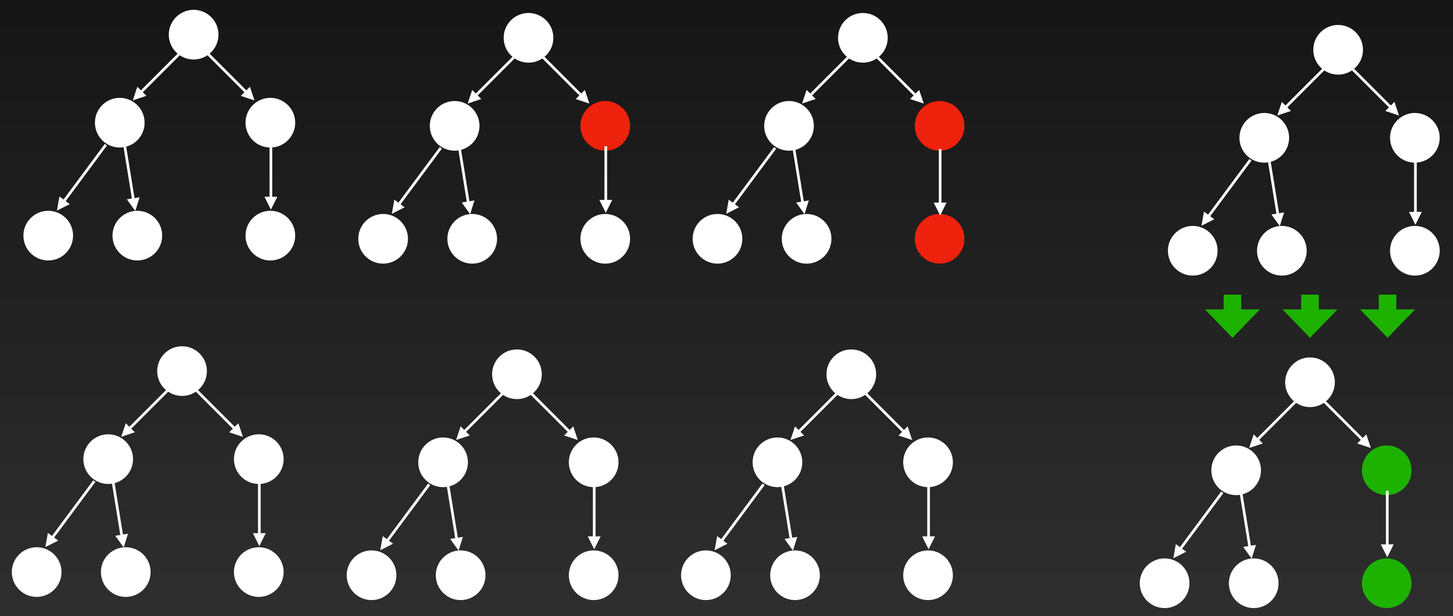
Прибавляем 1 и возвращаем результат, обновляя состояние

Вспоминаем vDOM

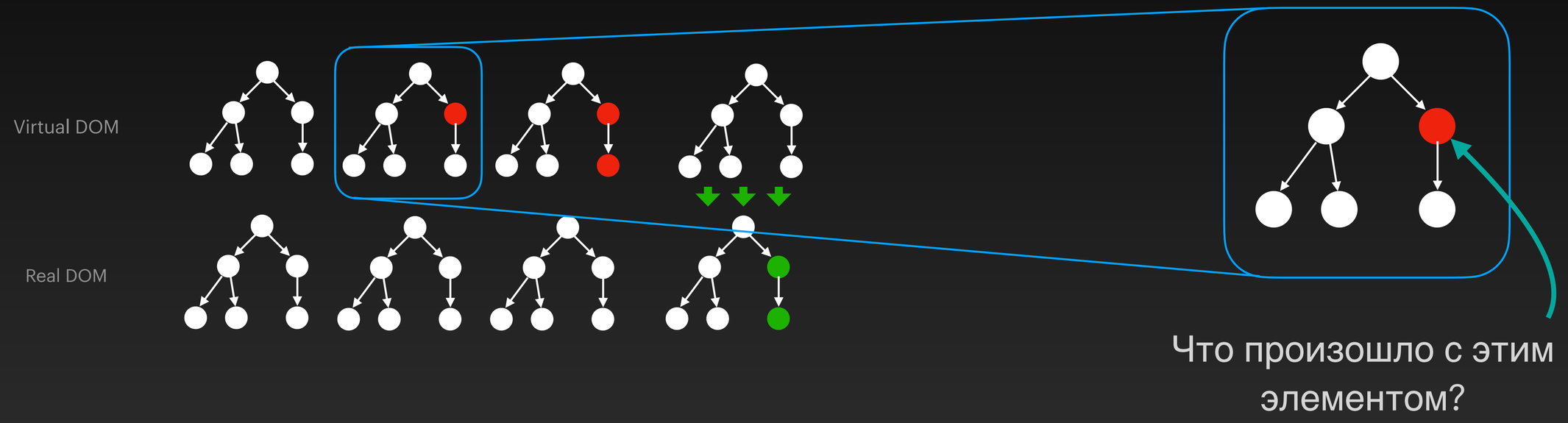
Virtual DOM vs Real DOM

Virtual DOM

Real DOM



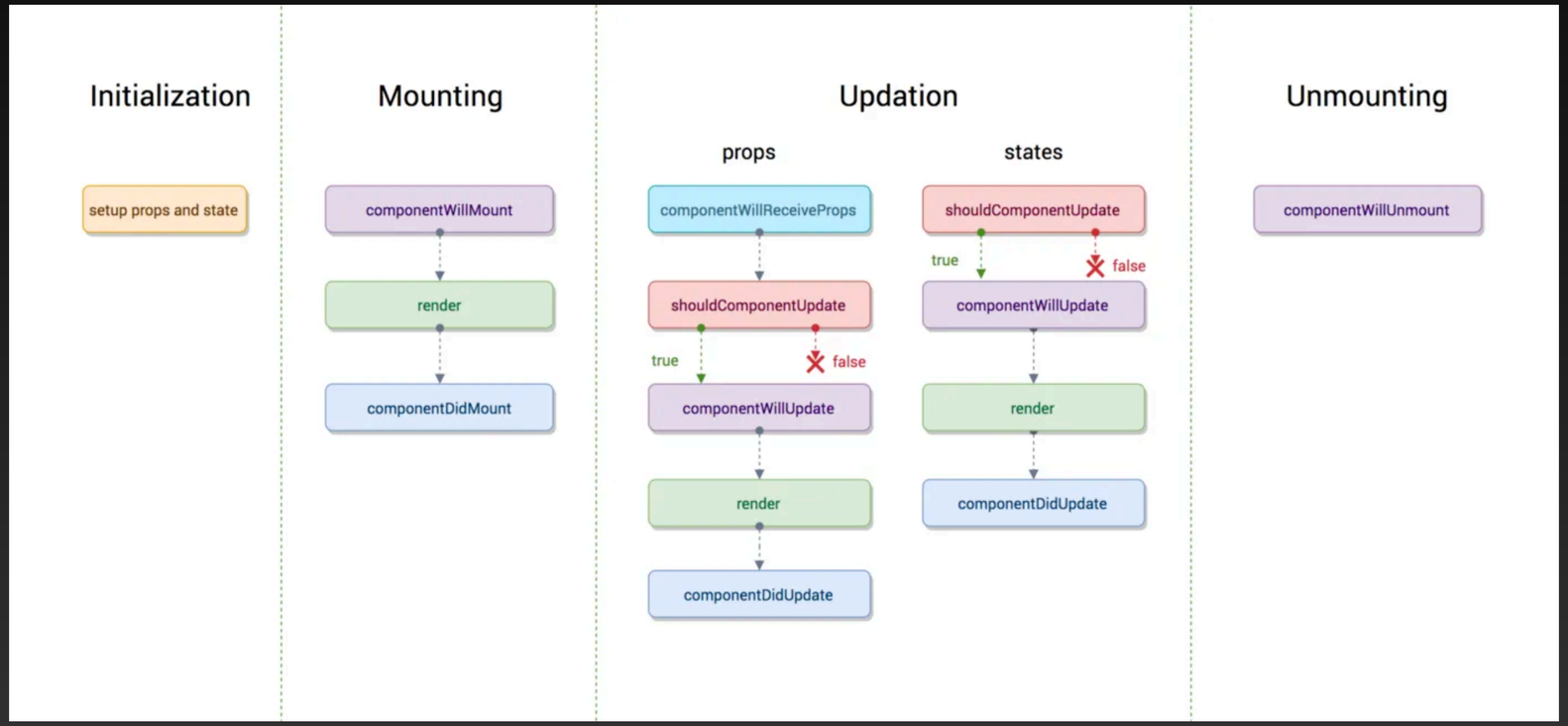
Virtual DOM vs Real DOM



Что вообще может происходить с компонентом в React?
Что, как и когда вызывает обновление vDOM?

Жизненный цикл компонент

Жизненный цикл компоненты

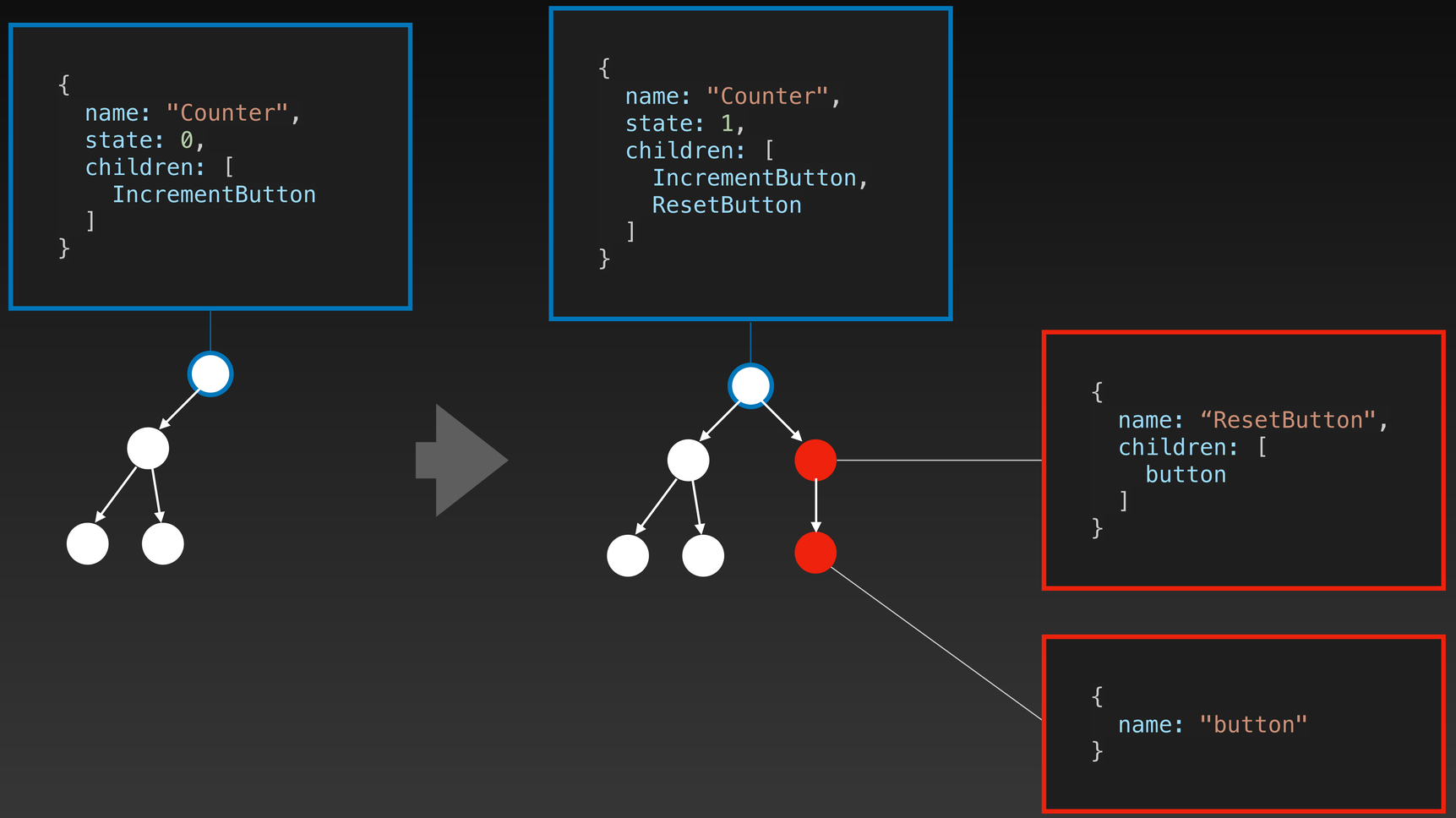


Монтирование

```tsx
interface IncrementButtonProps {
  count: number;
  increment: () => void;
}

const IncrementButton: FC<IncrementButtonProps> = ({ count, increment }) => {
  return (
    <button onClick={increment}>
      <span>{`Current count is: ${count}`}</span>
      <span>{`Click to increment`}</span>
    </button>
  );
};

interface ResetButtonProps {
  reset: () => void;
}

const ResetButton: FC<ResetButtonProps> = ({ reset }) => {
  return <button onClick={reset}>Reset</button>;
};
```

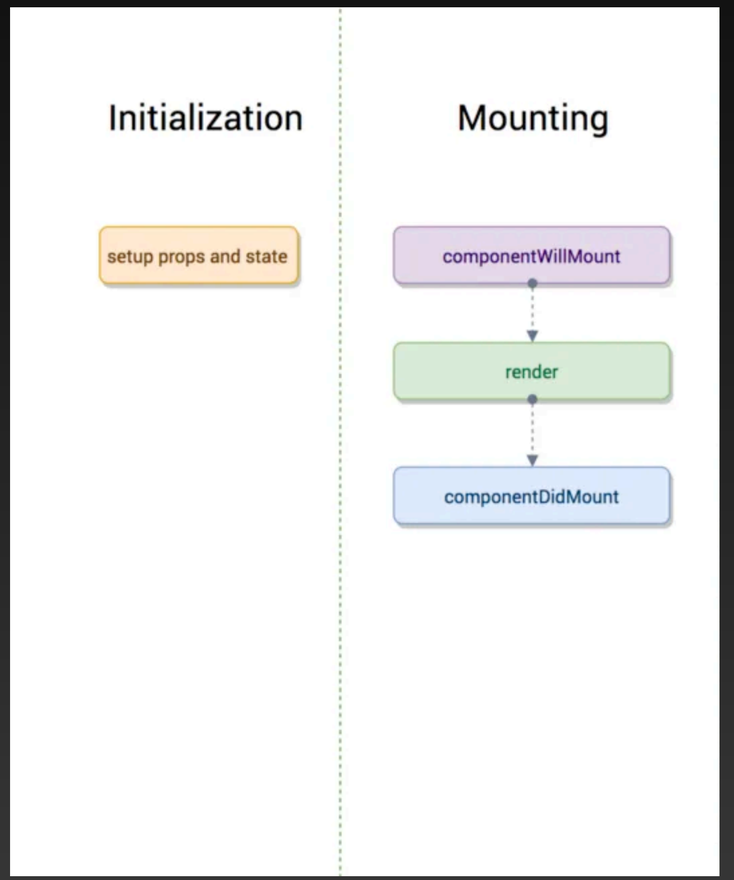
```tsx
export const Counter: FC = () => {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount((c) => c + 1);
  };

  const reset = () => {
    setCount(0);
  };

  return (
    <div>
      <IncrementButton count={count} increment={increment} />
      {count > 0 && <ResetButton reset={reset} />}
    </div>
  );
};
```

Монтирование



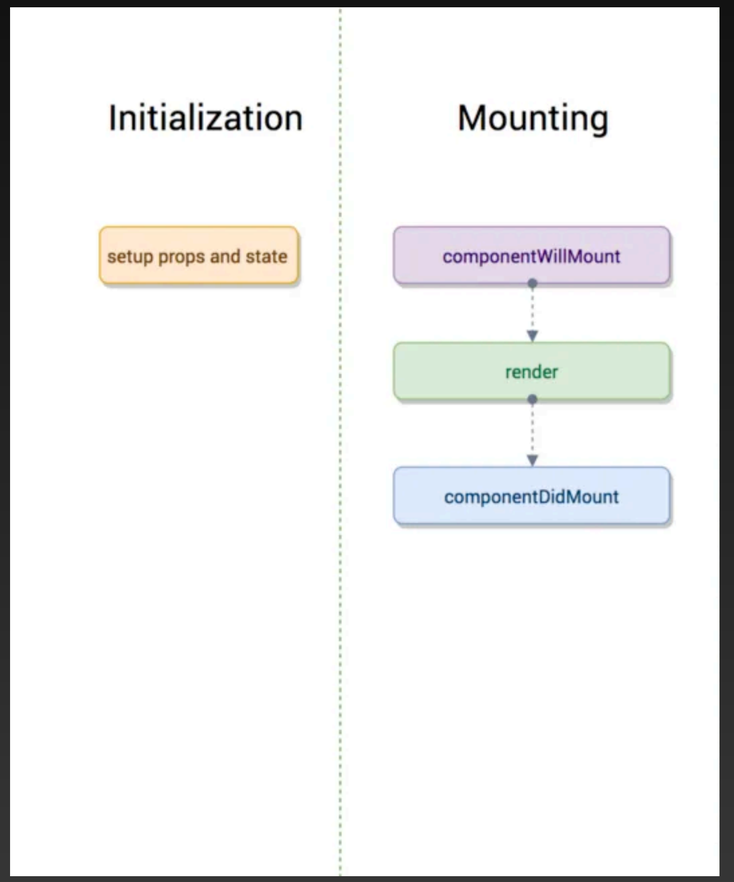
Монтирование

- Mount — монтирование, первый рендер
  - Создание состояния
  - Добавление элемента в vDOM
  - Монтирование детских элементов



Монтирование

- Под капотом:
  - Реакт в первый раз вызывает функцию-компонент с пропсами
  - Запоминает название функции и её родителя
  - Если внутри есть вызов useState, реакт создаст состояние и начнёт отслеживать его изменения
  - Реакт также монтирует “детские” компоненты из `return`

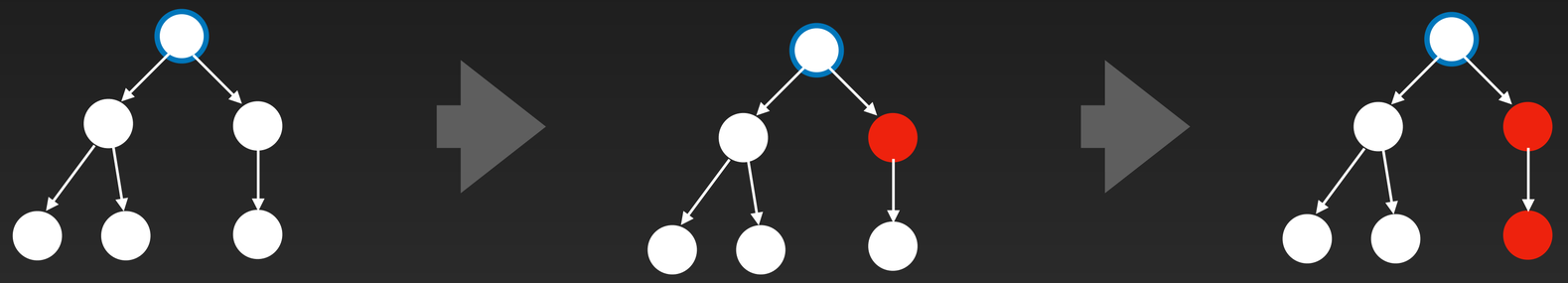


Обновление — re-render

Кто-то что-то кликнул

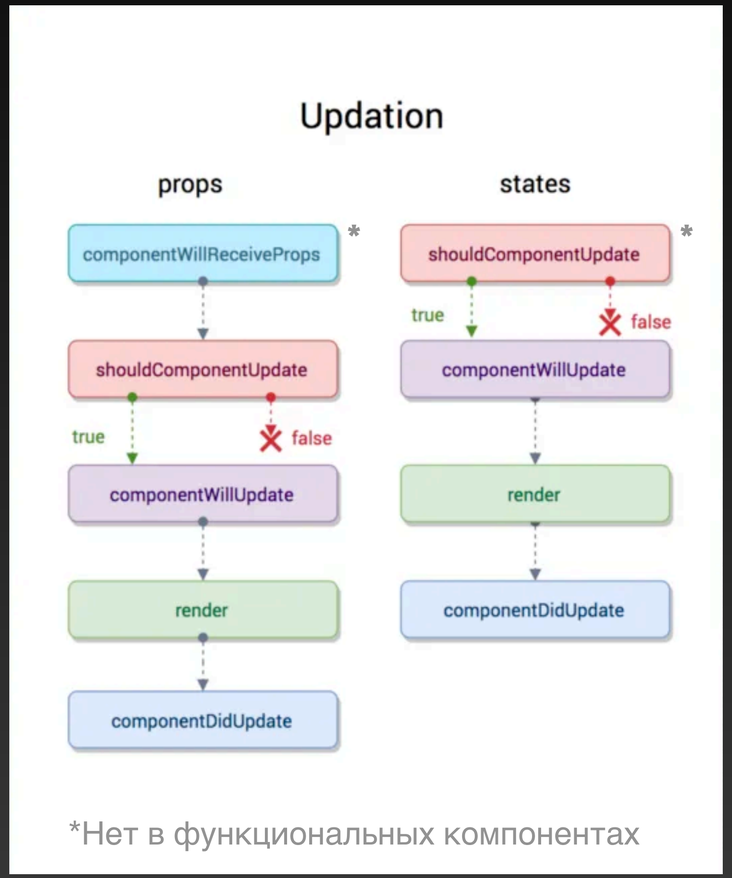
Сработал колбэк из очереди задач

Завершился промис



Обновление — re-render

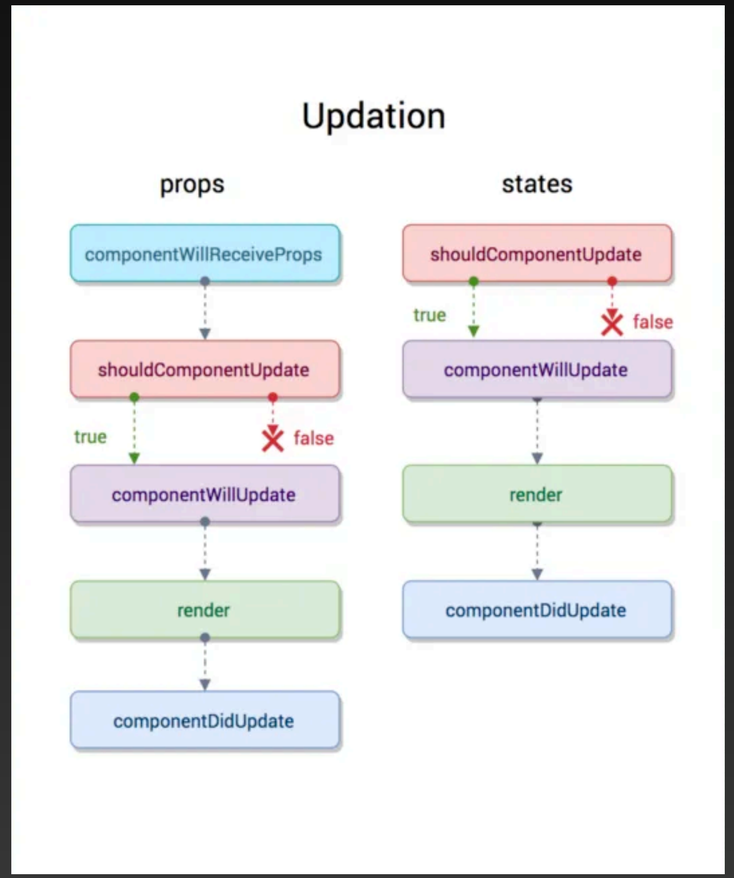
- Может быть вызвано *только*:
  - Изменением пропсов
  - Изменением состояния
  - Обновлением родителя
- Результат — изменение DOM



*Нет в функциональных компонентах

Обновление — re-render

- Под капотом:
  - Сравнение пропсов с предыдущими, принятие решения о необходимости обновления
  - Обновление — запуск функции-компонента с новыми пропсами
  - Обновление детских компонент



Размонтирование — unmounting

- Удаление состояния
- Опционально — очистка добавленных event listener’ов и подобных им
- Удаление элемента из vDOM

Размонтирование — unmounting

```tsx
interface IncrementButtonProps {
  count: number;
  increment: () => void;
}

const IncrementButton: FC<IncrementButtonProps> = ({ count, increment }) => {
  return (
    <button onClick={increment}>
      <span>{`Current count is: ${count}`}</span>
      <br />
      <span>{`Click to increment`}</span>
    </button>
  );
};

interface ResetButtonProps {
  title: string;
  reset: () => void;
}

const ResetButton: FC<ResetButtonProps> = ({ title, reset }) => {
  return <button onClick={reset}>{title}</button>;
};
```

```tsx
const TITLE_BY_STATE = {
  ready: "Reset",
  cooldown: "Please, wait to reset",
};

export const Counter: FC = () => {
  const [count, setCount] = useState(0);
  const [resetState, setResetState] = useState<"ready" | "cooldown">("ready");

  const increment = () => {
    setCount((c) => c + 1);
  };

  const reset = () => {
    if (resetState !== "ready") {
      return;
    }

    setCount(0);
    setResetState("cooldown");

    setTimeout(() => {
      setResetState("ready");
    }, 2000);
  };

  return (
    <div>
      <IncrementButton count={count} increment={increment} />
      <br />
      <br />
      {count > 0 && (
        <ResetButton title={TITLE_BY_STATE[resetState]} reset={reset} />
      )}
    </div>
  );
};
```

React Dev Tools

Домашнее задание

Event handling

- Напишите компонент-кнопку, которая по клику меняет цвет фона страницы с белого на чёрный
- Цвет фона задавайте через document.body.style.backgroundColor = ‘black’

Рендер списков и useState()

- Напишите компонент, который принимает в себя список элементов через пропс, и выводит на экран эти элементы в виде прямоугольников с текстом

```
enum ItemStatus {
  NEW = "NEW",
  COMPLETED = "COMPLETED",
}

const mockTasks = [
  { id: "h18ebac", title: "Вынести мусор", status: ItemStatus.NEW },
  { id: "lDAc9If", title: "Приготовить ужин", status: ItemStatus.NEW },
  { id: "I1aba0c", title: "Выспаться", status: ItemStatus.COMPLETED },
]
```

- Текст должен быть зачёркнут, если у элемента статус COMPLETED
- Каждому элементу добавьте кнопку, по которой элемент будет удалён из списка