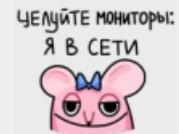


<Lib>React</Lib>

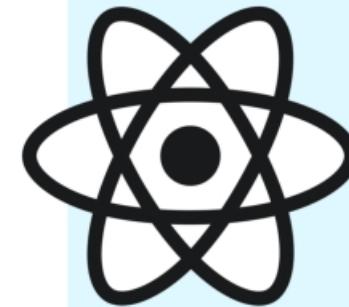


ЧЕЛУЙТЕ МОНИТОРЫ:  
Я В СЕТИ

Кулинич Ярослав

# ReactJs - библиотека для построения UI

- open-source (<https://github.com/facebook/react>)
- by Facebook
- Используют: facebook, instagram, periscope, imdb, twitch, uber, bbc и др.

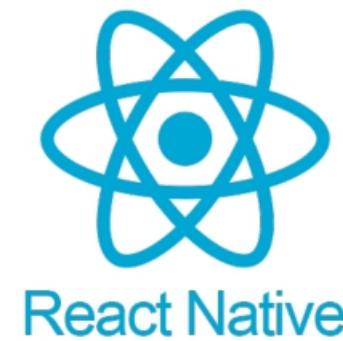


React

# Еще один способ верстать формы?

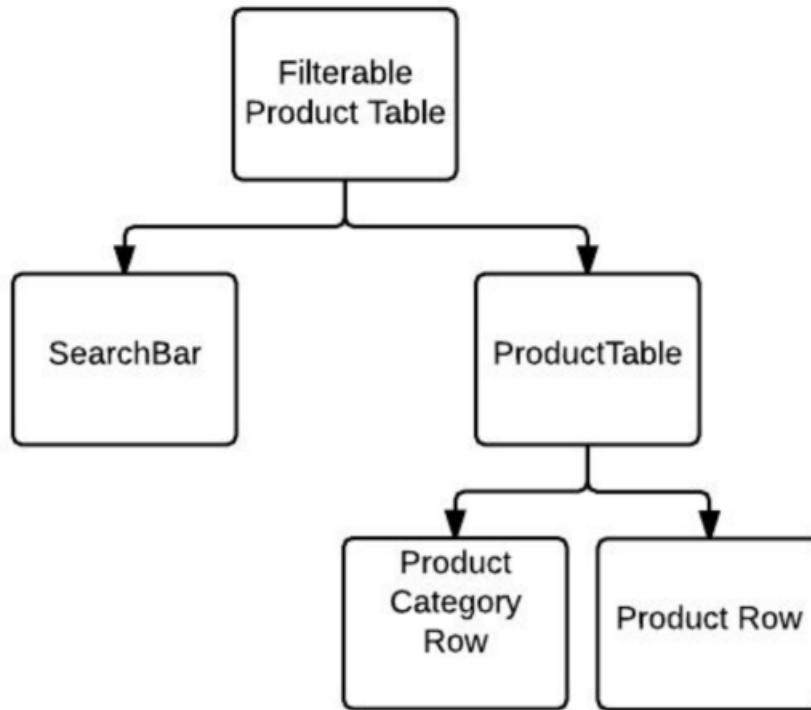
- Компоненты и их переиспользование
- Декларативность
- JSX разметка
- Легкая интеграция в существующие решения
- Можно писать нативные и фулстек приложения 😊

NEXT.js



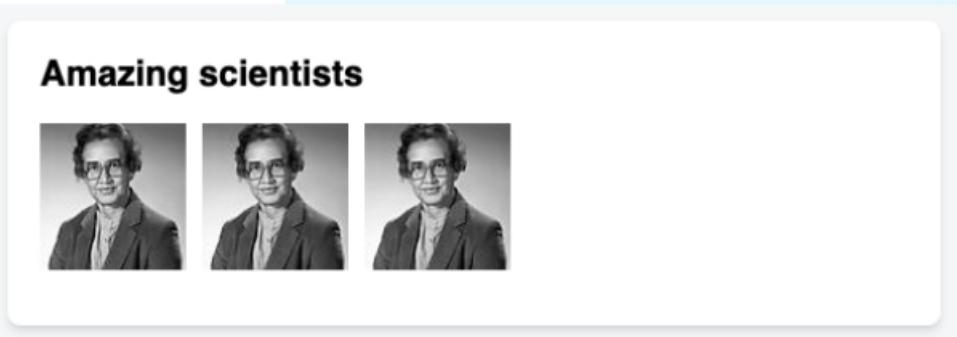
# React компоненты

Name	Price
<b>Sporting Goods</b>	
Football	\$49.99
Baseball	\$9.99
Basketball	\$29.99
<b>Electronics</b>	
iPod Touch	\$99.99
<b>iPhone 5</b>	\$399.99
Nexus 7	\$199.99



# Мой первый компонент

```
export default function Profile() {  
  return (  
      
  )  
}  
  
export default function Gallery() {  
  return (  
    <section>  
      <h1>Amazing scientists</h1>  
      <Profile />  
      <Profile />  
    </section>  
  );  
}
```



- Названия компонентов с большой буквы;
- Никаких объявлений компонентов друг в друге.



# Держим код чистым

- Стаемся не держать много сложных компонентов в одном файле;
- Пользуемся import/export для агрегации компонентов;
- Имхо - export default - зло.

Syntax	Export statement	Import statement
Default	<code>export default function Button() {}</code>	<code>import Button from './Button.js';</code>
Named	<code>export function Button() {}</code>	<code>import { Button } from './Button.js';</code>

# JSX

- JSX != html
- Компоненты должны возвращать всегда 1 элемент
- Все теги должны быть закрытыми (<img> -> <img />)
- Почти все теперь в camelCase (className, onClick)
- Под капотом превращается в JS

```
Sidebar() {  
  if (isLoggedIn()) {  
  
    <p>Welcome</p>  
  
  } else {  
  
    <Form />  
  
  }  
}
```

Sidebar.js React component

```
Form() {  
  onClick() {...}  
  onSubmit() {...}  
  
  <form onSubmit>  
    <input onClick />  
    <input onClick />  
  </form>  
}
```

Form.js React component

Каждый компонент должен  
возвращать только 1 тег



# Магия JSX

```
function App() {  
  return <h1>Hello World</h1>;  
}  
  
function App() {  
  return React.createElement('h1', null, 'Hello world');  
}
```

# Встраивание JS в JSX

```
export default function Avatar() {
  const avatar = 'https://i.imgur.com/7vQD0fPs.jpg';
  const description = 'Gregorio Y. Zara';
  return (
    <img
      className="avatar"
      src={avatar}
      alt={description}
    />
  );
}
```

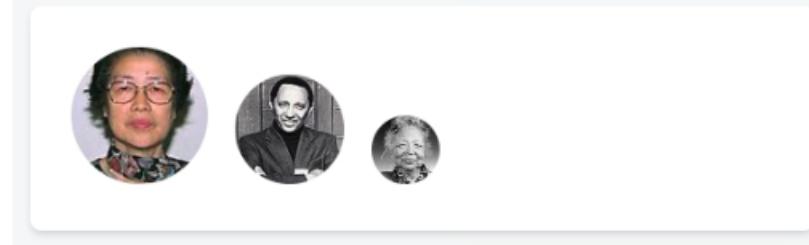
# Встраивание JS в JSX: объекты

```
export default function TodoList() {
  return (
    <ul style={{
      backgroundColor: 'black',
      color: 'pink'
    }>
      <li>Improve the videophone</li>
      <li>Prepare aeronautics lectures</li>
      <li>Work on the alcohol-fuelled engine</li>
    </ul>
  );
}
```

# Props

```
function Avatar({ person, size = 50 }) {  
  return (  
    <img  
      className="avatar"  
      src={getImageUrl(person)}  
      alt={person.name}  
      width={size}  
      height={size}  
    />  
  );  
}  
  
// In some component
```

```
<Avatar  
  size={100}  
  person={{  
    name: 'Katsuko Saruhashi',  
    imageId: 'YfeOqp2'  
  }}  
/>
```



# Props forwarding

```
function Profile({ person, size, isSepia, thickBorder }) {
  return (
    <div className="card">
      <Avatar
        person={person}
        size={size}
        isSepia={isSepia}
        thickBorder={thickBorder}>
      />
    </div>
  );
}
```

```
function Profile(props) {
  return (
    <div className="card">
      <Avatar {...props} />
    </div>
  );
}
```

# React компоненты

```
function Card({ children }) {  
  return (  
    <div className="card">  
      {children}  
    </div>  
  );  
}
```

```
export default function Profile() {  
  return (  
    <Card>  
      <Avatar  
        size={100}  
        person={{  
          name: 'Katsuko Saruhashi',  
          imageId: 'YfeOqp2'  
        }}  
      />  
    </Card>  
  );  
}
```

- Пропсы иммутабельные!
- Изменение пропсов приводит к перерисовке компонента.



# Conditional rendering

```
// Component 1
if (isPacked) {
  return <li className="item">{name} ✓</li>;
}
return <li className="item">{name}</li>;

// Component 2
if (isPacked) {
  return null;
}
return <li className="item">{name}</li>;

// Component 3
return (
  <li className="item">
    {isPacked ? name + ' ✓' : name}
  </li>
);
```

- 1 и 3 варианты эквивалентны?



# Conditional rendering

```
// Component 4
return (
  <li className="item">
    {name} {isPacked && '✓'}
  </li>
);

// Component 5
let itemContent = name;
if (isPacked) {
  itemContent = name + " ✓";
}
return (
  <li className="item">
    {itemContent}
  </li>
);
```

# List rendering

```
const listItems = chemists.map(person =>
  <li>
    <img
      src={getImageUrl(person)}
      alt={person.name}
    />
    <p>
      <b>{person.name}</b>
      {' ' + person.profession + ' '}
      known for {person.accomplishment}
    </p>
  </li>
);
return <ul>{listItems}</ul>;
```



**Creola Katherine Johnson** mathematician known for spaceflight calculations



**Mario José Molina-Pasquel Henríquez** chemist known for discovery of Arctic ozone hole



**Mohammad Abdus Salam** physicist known for electromagnetism theory

- Необходимость атрибута key



# «Pure» компоненты

- На одни и те же аргументы функция возвращает один и тот же результат;
- Никаких сайд-эффектов\*;
- <React.StrictMode />.

```
let guest = 0;

function Cup() {
  // Bad: changing a preexisting variable!
  guest = guest + 1;
  return <h2>Tea cup for guest #{guest}</h2>;
}
```

# Local mutations

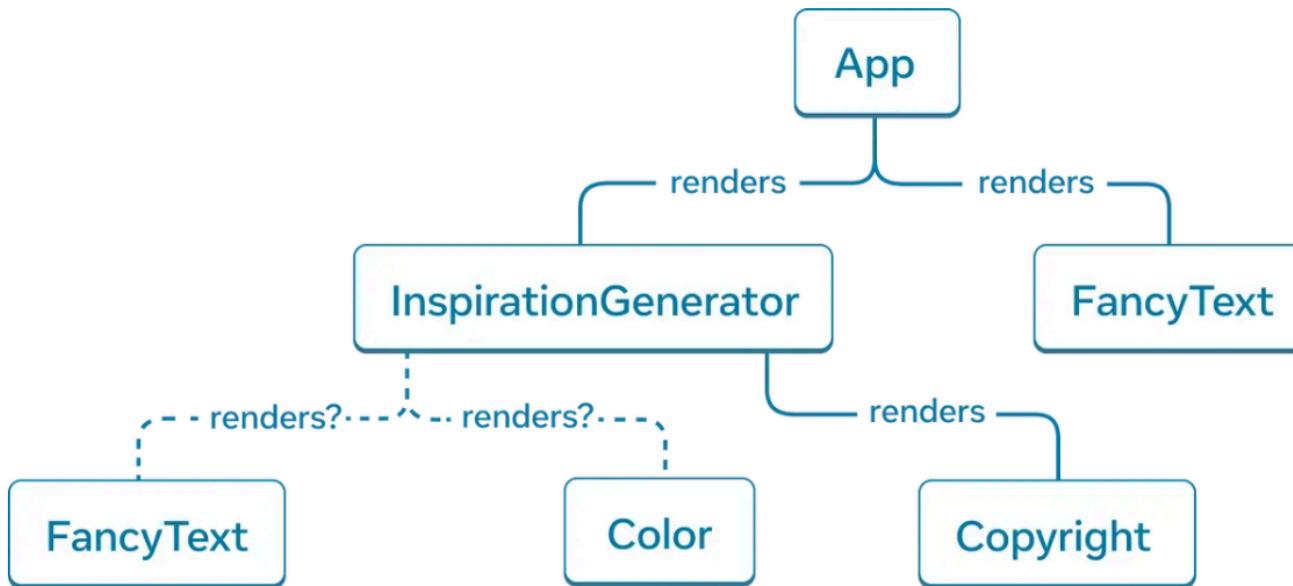
```
export default function TeaGathering() {  
  let cups = [];  
  for (let i = 1; i <= 12; i++) {  
    cups.push(<Cup key={i} guest={i} />);  
  }  
  return cups;  
}
```

- Event Handlers не обязательно Pure;
- Возможности северного рендеринга, скрипты фреймов и остановка отрисовки



# Render Tree

- Дерево из компонентов;
- Абстрагировано от платформы;
- Позволяет избегать лишних взаимодействий с платформой под которой работает.



# Event handling

```
export default function Button() {
  function handleClick() {
    alert('You clicked me!');
  }

  return (
    <button onClick={handleClick}>
      Click me
    </button>
  );
}
```

passing a function (correct)

```
<button onClick={handleClick}>
```

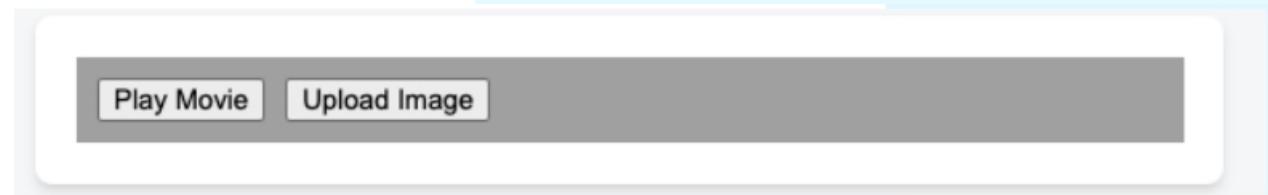
calling a function (incorrect)

```
<button onClick={handleClick()}>
```

# Event handling

```
export default function Toolbar() {
  return (
    <div className="Toolbar" onClick={() => {
      alert('You clicked on the toolbar!');
    }}>
      <button onClick={() => alert('Playing!')}>
        Play Movie
      </button>
      <button onClick={() => alert('Uploading!')}>
        Upload Image
      </button>
    </div>
  );
}
```

event.stopPropagation()  
event.preventDefault()



# State handle

```
export default function Gallery() {
  let index = 0;
  function handleClick() { index = index + 1; }

  let sculpture = sculptureList[index];
  return (
    <>
    <button onClick={handleClick}>
      Next
    </button>
    <h2>
      <i>{sculpture.name} </i>
      by {sculpture.artist}
    </h2>
    </>
  );
}
```

- Локальные переменные как Райан Гослинг;
- Также не инициирует перерисовку.



# State handle

```
export default function Gallery() {
  const [index, setIndex] = useState(0);
  function handleClick() { setIndex(index + 1); }

  let sculpture = sculptureList[index];
  return (
    <>
    <button onClick={handleClick}>
      Next
    </button>
    <h2>
      <i>{sculpture.name} </i>
      by {sculpture.artist}
    </h2>
    </>
  );
}
```

# Встречаем свой первый хук

- Именуются на use;
- Специальные функции, доступные только во время рендера компонента;
- Обязаны выполняться при любом рендре в одинаковом порядке.
- useState изолирован в компонентах;

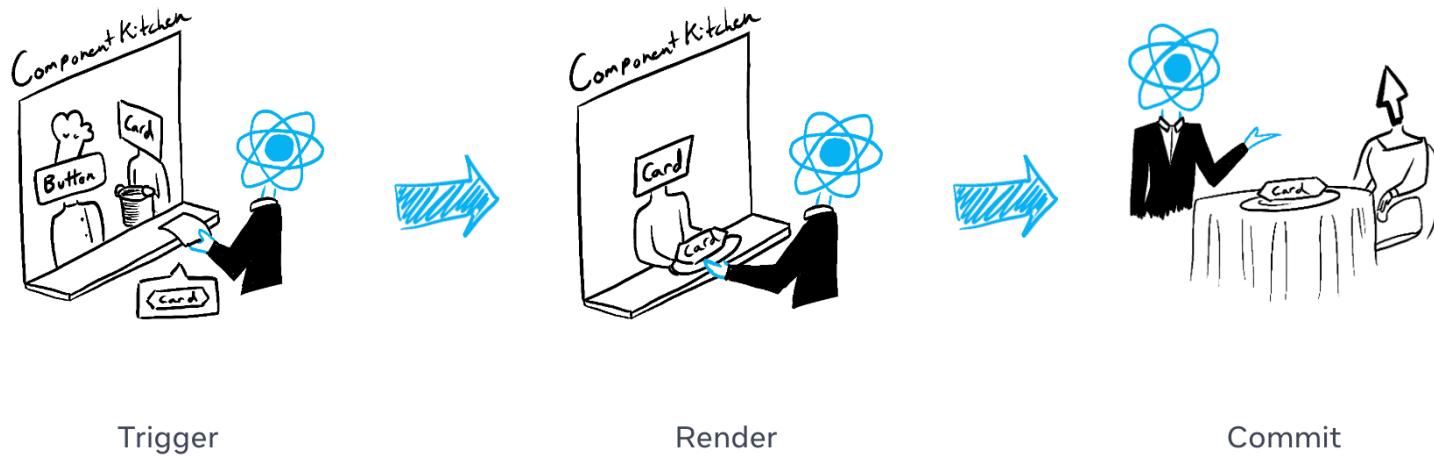


```
const [index, setIndex] = useState(0);
```

Hooks under the hood



# Render and commit



- Триггер случается при первой отрисовке или обновлении стейта;
- Во время первого рендера создаются DOM узлы;
- Во время ре-рендера узлы создаются при необходимости;
- Во время первого коммита узлы добавляются в DOM через `appendChild`;
- Во время ре-коммитов высчитывается разница и над DOM осуществляется только необходимая работа;
- Рендер рекурсивен - при rerендре компонента, rerендерятся все его потомки.

# State as a snapshot

```
export default function Counter() {  
  const [number, setNumber] = useState(0);  
  
  return (  
    <>  
      <h1>{number}</h1>  
      <button onClick={() => {  
        setNumber(number + 1);  
        setNumber(number + 1);  
        setNumber(number + 1);  
      }}>+3</button>  
    </>  
  )  
}
```

- useState возвращает слепки стейта, которые хранятся вне скоупа компонента;
- setState изменяет состояние относительно слепка на то, что будет при следующей отрисовке

# State as a snapshot

```
export default function Counter() {
  const [number, setNumber] = useState(0);

  return (
    <>
      <h1>{number}</h1>
      <button onClick={() => {
        setNumber(n => n + 1);
        setNumber(n => n + 1);
        setNumber(n => n + 1);
      }}>+3</button>
    </>
  )
}
```

queued update	n	returns
n => n + 1	0	0 + 1 = 1
n => n + 1	1	1 + 1 = 2
n => n + 1	2	2 + 1 = 3

# State as a snapshot

```
export default function Counter() {  
  const [number, setNumber] = useState(0);  
  
  return (  
    <>  
      <h1>{number}</h1>  
      <button onClick={() => {  
        setNumber(number + 5);  
        setNumber(n => n + 1);  
        setNumber(42);  
      }}>Increase the number</button>  
    </>  
  )  
}
```

queued update	n	returns
"replace with 5"	0 (unused)	5
n => n + 1	5	5 + 1 = 6
"replace with 42"	6 (unused)	42

# Objects as state

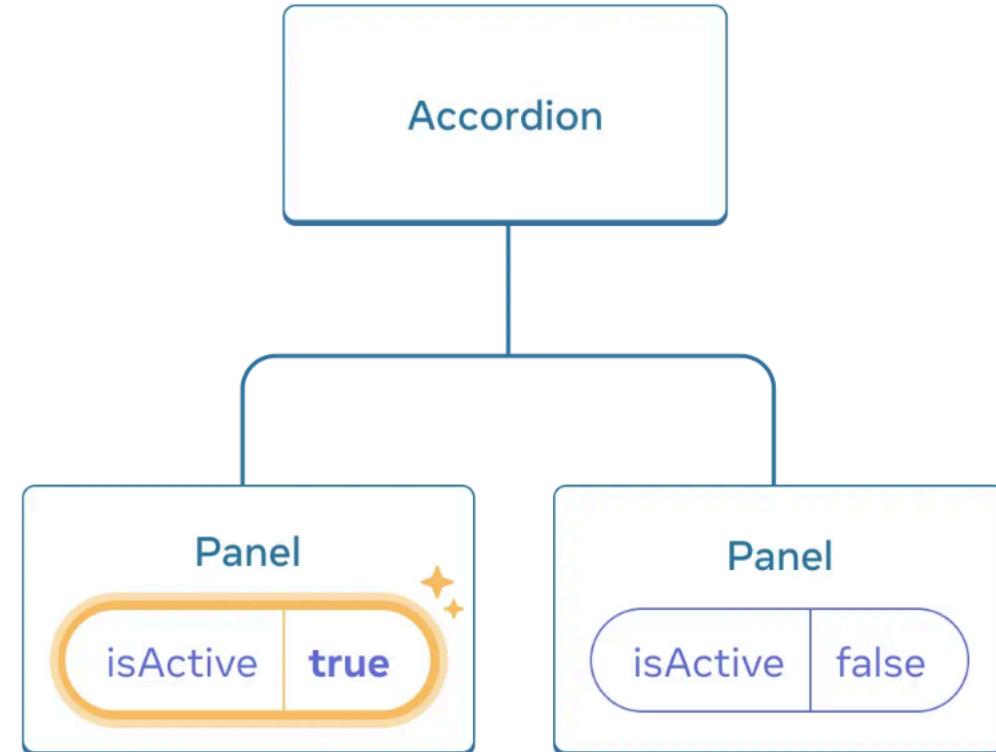
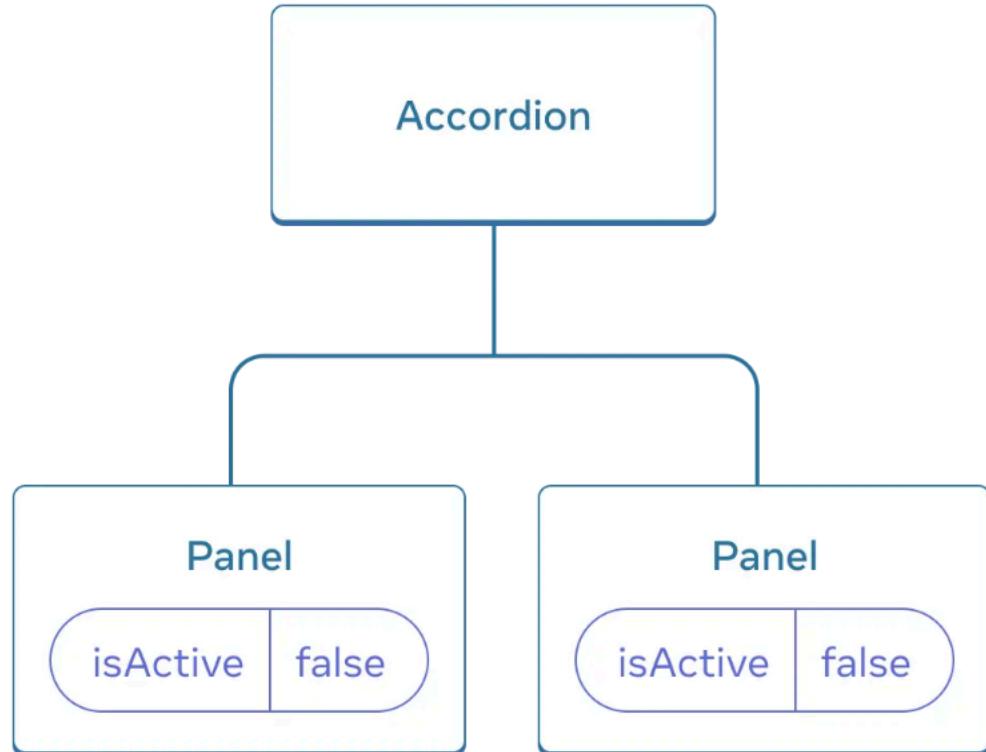
```
const [person, setPerson] = useState({  
  firstName: 'Barbara',  
  lastName: 'Hepworth',  
  email: 'bhepworth@sculpture.com'  
});  
  
function handleFirstNameChange(e) {  
  setPerson({  
    ...person,  
    firstName: e.target.value  
  });  
}
```

# Arrays as state

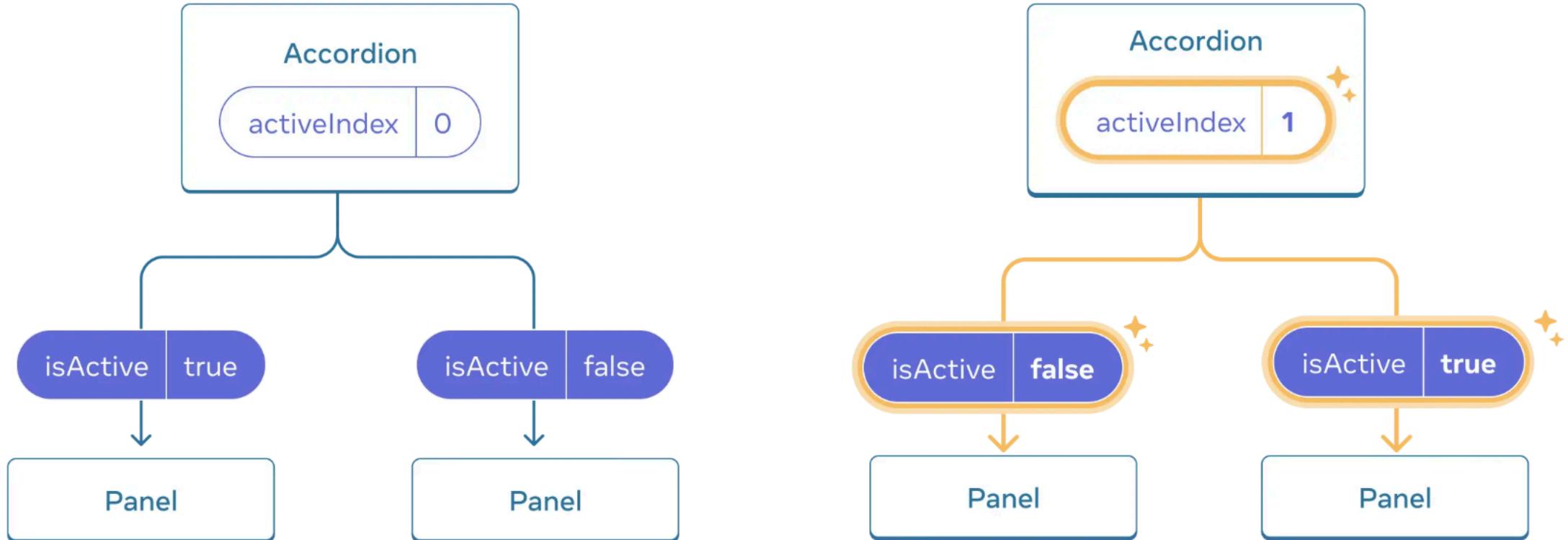
	avoid (mutates the array)	prefer (returns a new array)
adding	<code>push</code> , <code>unshift</code>	<code>concat</code> , <code>[...arr]</code> spread syntax ( <a href="#">example</a> )
removing	<code>pop</code> , <code>shift</code> , <code>splice</code>	<code>filter</code> , <code>slice</code> ( <a href="#">example</a> )
replacing	<code>splice</code> , <code>arr[i] = ...</code> assignment	<code>map</code> ( <a href="#">example</a> )
sorting	<code>reverse</code> , <code>sort</code>	copy the array first ( <a href="#">example</a> )

```
setArtists([
  { id: nextId++, name: name },
  ...artists // Put old items at the end
]);
```

# Sharing state - lifting up

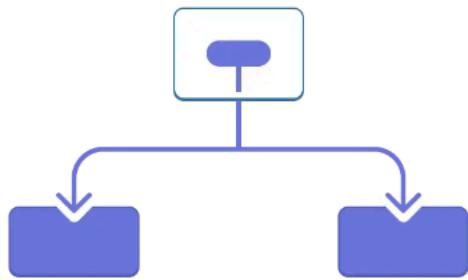


# Sharing state - lifting up

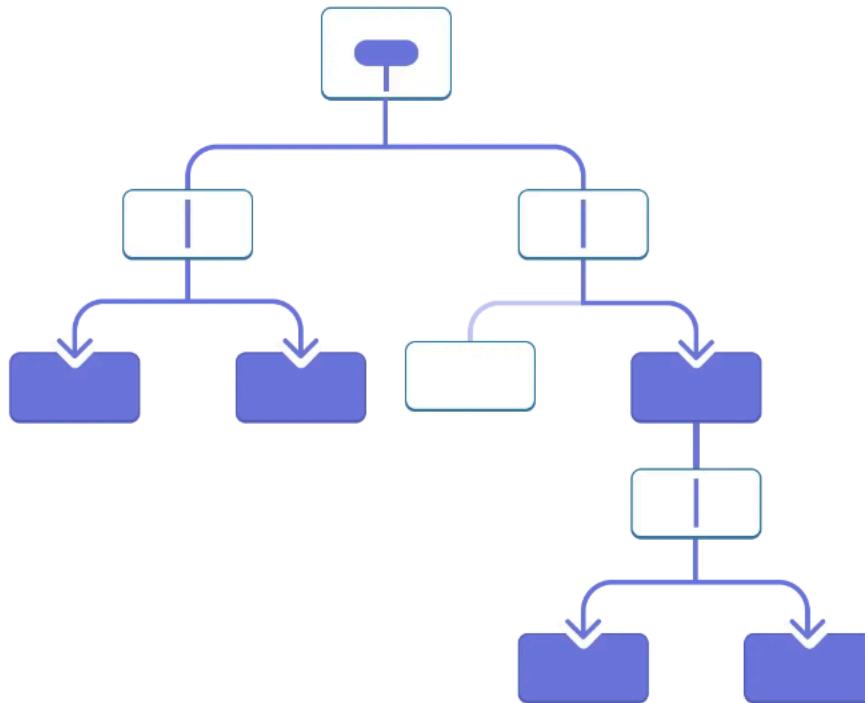


# Sharing state - context

Lifting state up



Prop drilling



# Sharing state - context

```
export const LevelContext = createContext(1);

export default function Heading({ children }) {
  const level = useContext(LevelContext);
  // ...
}

<section className="section">
  <LevelContext.Provider value={level}>
    {children}
  </LevelContext.Provider>
</section>
```

Контекст - не золотой молоток;  
Пробуем сначала пропсы (значения или  
jsx);  
Идеальные кандидаты - account, theme.

# Effects

- Компонент обычно состоит из логики рендера и обработки ивентов;
- Обработчики являются идеальным местом для side-effects;
- Порой их может не хватать и side-effect необходим во время рендера - тут помогают effects;
- Эффекты начинают работать после коммит фазы;
- Эффект - не ультимативное решение, не надо его везде использовать.

```
function VideoPlayer({ src, isPlaying }) {
  const ref = useRef(null);

  if (isPlaying) {
    ref.current.play(); // Calling these while rendering isn't allowed.
  } else {
    ref.current.pause(); // Also, this crashes.
  }

  return <video ref={ref} src={src} loop playsInline />;
}
```

# Effects

```
function VideoPlayer({ src, isPlaying }) {
  const ref = useRef(null);

  useEffect(() => {
    if (isPlaying) {
      console.log('Calling video.play()');
      ref.current.play();
    } else {
      console.log('Calling video.pause()');
      ref.current.pause();
    }
  }, [isPlaying]);

  return <video ref={ref} src={src} loop playsInline />;
}
```

# Effects

```
useEffect(() => {  
  // This runs after every render  
});  
  
useEffect(() => {  
  // This runs only on mount (when the component appears)  
, []);  
  
useEffect(() => {  
  // This runs on mount *and also* if either a or b have changed since  
  // the last render  
, [a, b]);|
```

# Effects

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

# React Router

```
' const router = createBrowserRouter([
' {
|   path: "/",
|   element: <div>Hello world!</div>,
| },
|]);
'
ReactDOM.createRoot(document.getElementById("root")).render(
<React.StrictMode>
| <RouterProvider router={router} />
| </React.StrictMode>
|);
```

# React Router

```
const router = createBrowserRouter([
  {
    path: "contacts/:contactId",
    element: <ProfilePage />,
  },
]);
function ProfilePage() {
  // Get the userId param from the URL.
  let { contactId } = useParams();
  // ...
}
```

# React Router

```
const navigate = useNavigate();

useEffect(() => {
  if (userIsInactive) {
    fake.logout();
    navigate("/session-timed-out");
  }
}, [userIsInactive]);
```

# Redux

Заявленные преимущества:

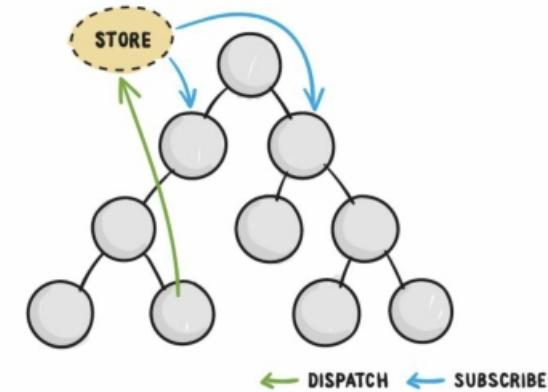
- Возможность сохранения истории всех изменений, а как следствие...
- простота отладки приложения
- Единственный "источник правды" — все компоненты отображают одно и то же состояние
- Компоненты не имеют состояния — их легче тестировать

Недостатки (незаявленные):

- Простейшие вещи требуют написания большого количества кода
- To be continued...

# Store

Redux предоставляет единое хранилище состояния. Ключевой особенностью является неизменяемость (привет, функциональщина!) — все изменения в него вносятся посредством создания нового объекта состояния.



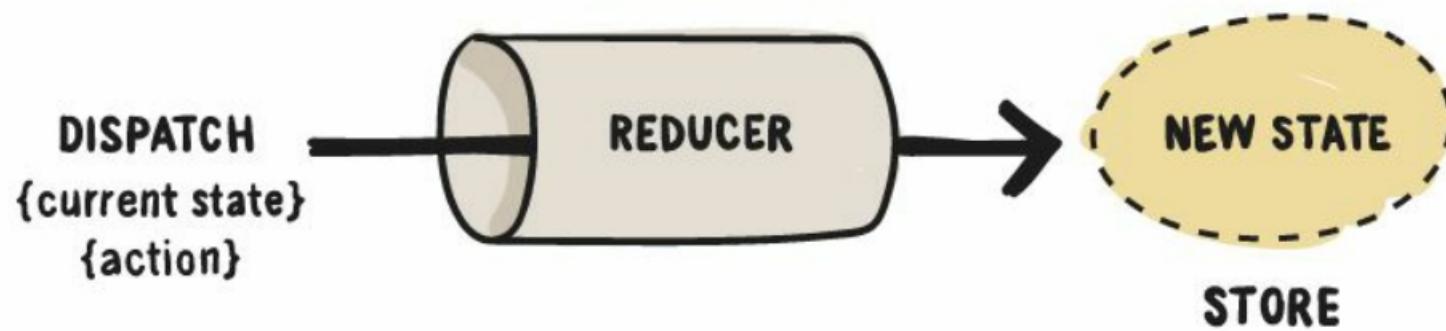
В “правильном” React-приложении с использованием Redux у компонентов нет state’а

## Action

Некоторое событие, которое должно привести к изменению состояния (созданию нового объекта состояния), будь то ответ от сервера или нажатие на кнопку в интерфейсе.

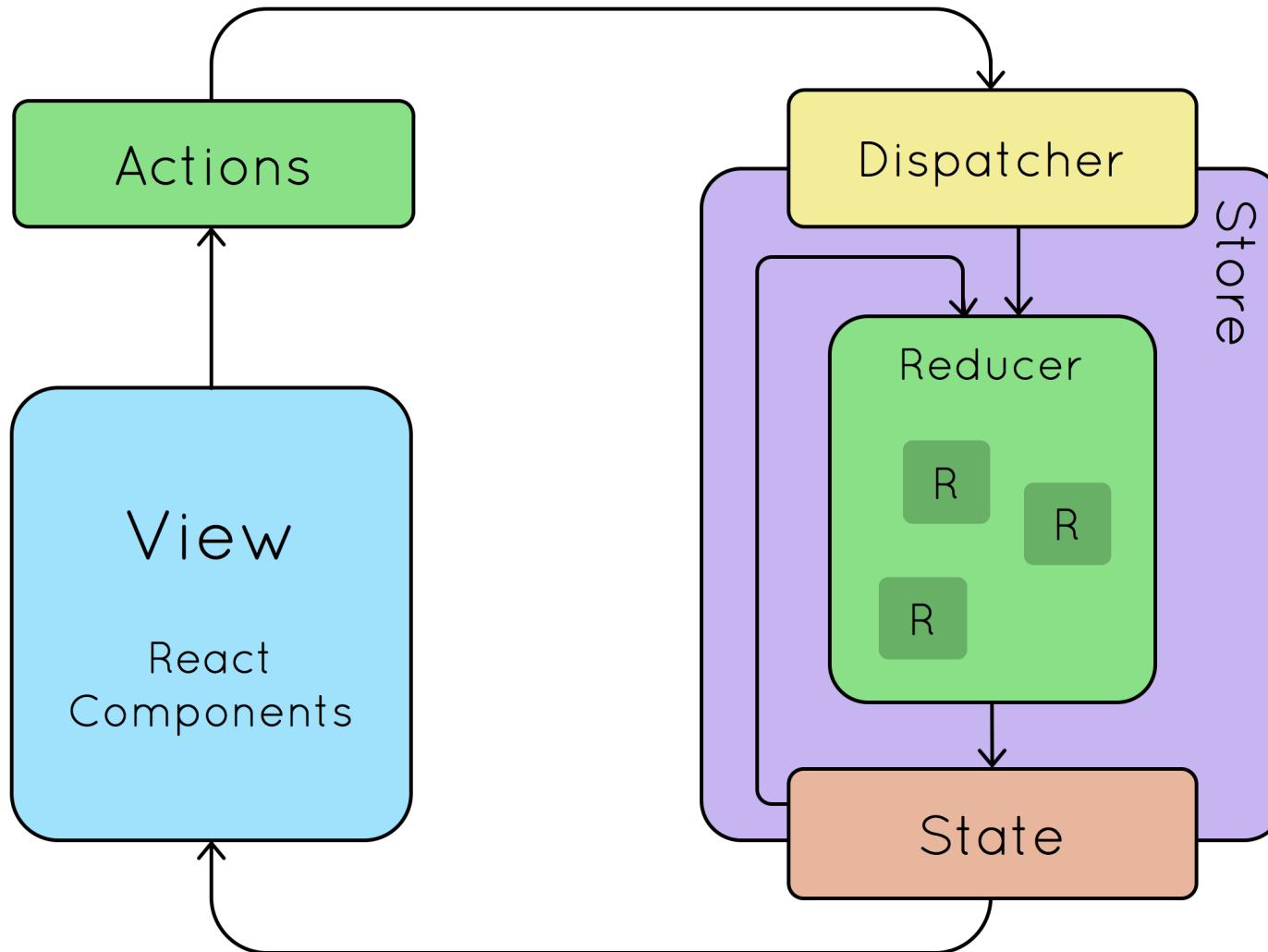
С точки зрения библиотеки — любой JS объект, у которого есть идентификатор типа события (обычно строковое свойство `type`) и некоторая полезная нагрузка.

# Reducer



- Чистая функция, её выходное значение зависит только от входных
- Необходимо возвращать новое состояние, состояние должно быть неизменяемо

# Redux



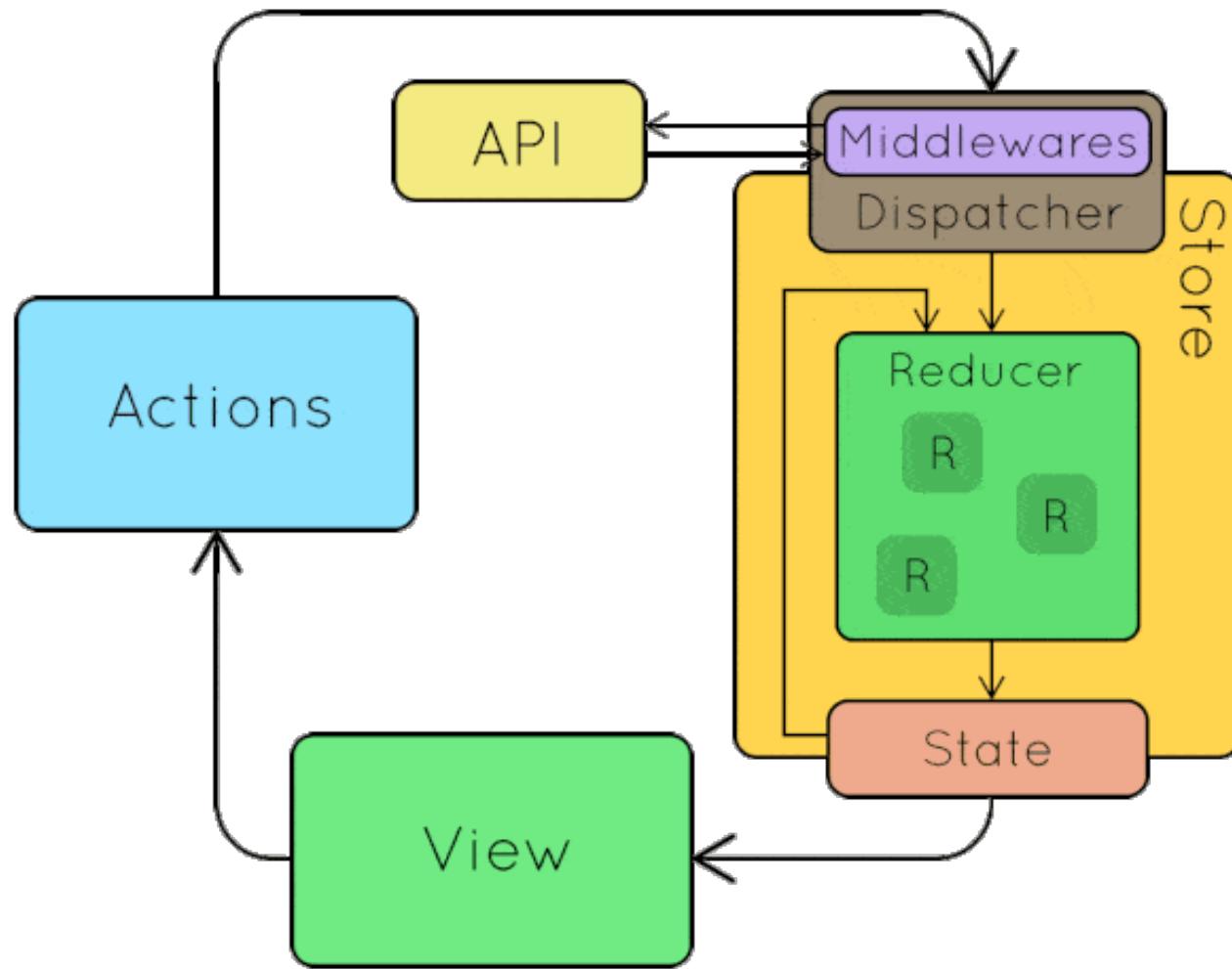
# Middleware

**Каррирование** или **карринг** ([англ. currying](#)) в [информатике](#) — преобразование функции от многих аргументов в набор функций, каждая из которых с одним аргументом. Возможность такого преобразования впервые отмечена в трудах [Готтлоба Фреге](#), систематически изучена [Моисеем Шейнфинкелем](#) в 1920-е годы, а наименование получило по имени [Хаскелла Карри](#) — разработчика [комбинаторной логики](#), в которой сведение к функциям одного аргумента носит основополагающий характер.

# Redux



# Redux



# Redux Toolkit

```
const initialState = {
  value: 0,
}

export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => {
      state.value += 1
    },
    decrement: (state) => {
      state.value -= 1
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload
    },
  },
})
export const { increment, decrement, incrementByAmount } =
  counterSlice.actions
export default counterSlice.reducer
```

# Redux Toolkit

```
export const store = configureStore({  
  reducer: {  
    counter: counterReducer,  
  },  
})
```

```
ReactDOM.render(  
  <Provider store={store}>  
    <App />  
  </Provider>,  
  document.getElementById('root')  
)
```

# Redux Toolkit

```
export function Counter() {
  const count = useSelector((state) => state.counter.value)
  const dispatch = useDispatch()

  return (
    <div>
      <div>
        <button
          onClick={() => dispatch(increment())}>
          Increment
        </button>
        <span>{count}</span>
        <button
          onClick={() => dispatch(decrement())}>
          Decrement
        </button>
      </div>
    </div>
  )
}
```

# Redux Toolkit

```
export const pokemonApi = createApi({  
  reducerPath: 'pokemonApi',  
  baseQuery: fetchBaseQuery({ baseUrl: 'https://pokeapi.co/api/v2/' }),  
  endpoints: (builder) => ({  
    getPokemonByName: builder.query<Pokemon, string>({  
      query: (name) => `pokemon/${name}`,  
    }),  
  }),  
})
```

```
export default function App() {  
  const { data, error, isLoading } =  
    useGetPokemonByNameQuery('bulbasaur')  
  
  // render UI based on data and loading state  
}
```

# Спасибо за внимание



Кулинич Ярослав