

M E T R O

## REACT AND REFLUX: *Components, Actions, Stores, and State*

28./29. September 2017,  
code.talks Hamburg



Philip Baues and Thomas Ströder

# Drilled down webshop example

METRO

3.88€



Aro H-Milch 3,5%

Unbelievable but true, this is really just milk. The exquisite milk comes in a handy 1l packaging.

0.99

Aro Blütenhonig flüssig

The real honey experience.

2.89

# Why React?

# Why React?

declarative view abstraction

# Why React?

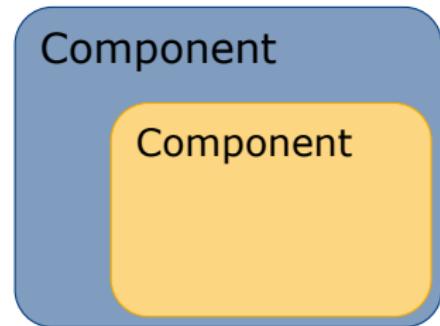
declarative view abstraction



Component

# Why React?

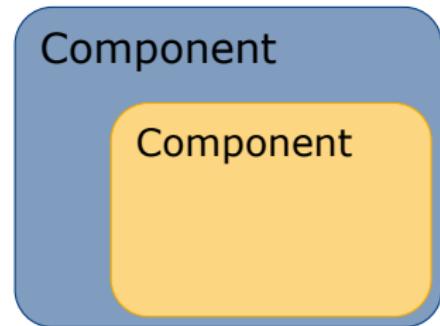
declarative view abstraction



# Why React?

declarative view abstraction

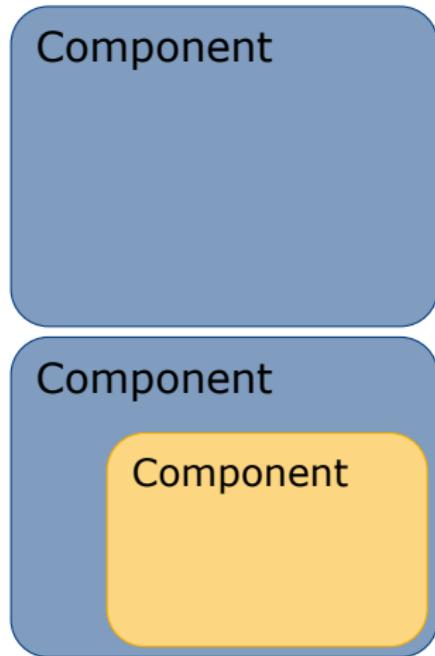
decoupling



# Why React?

declarative view abstraction

decoupling

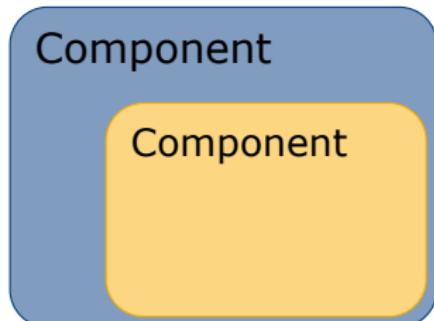


# Why React?

declarative view abstraction

decoupling

easy testing



# Why React?

declarative view abstraction

decoupling

easy testing

fast rendering

Component

Component

Component

# React Components

Component

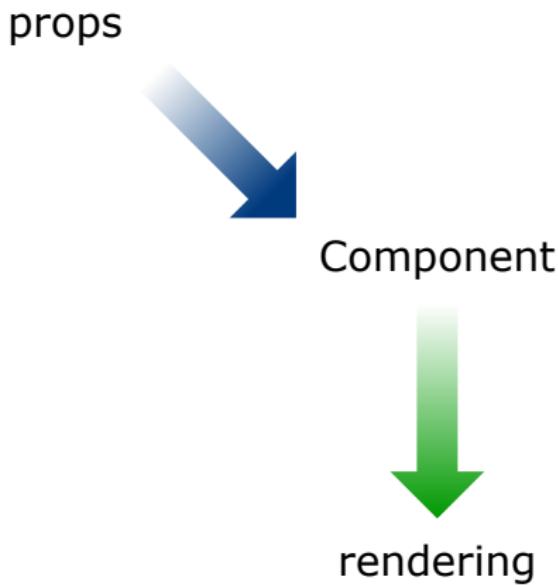
# React Components

Component

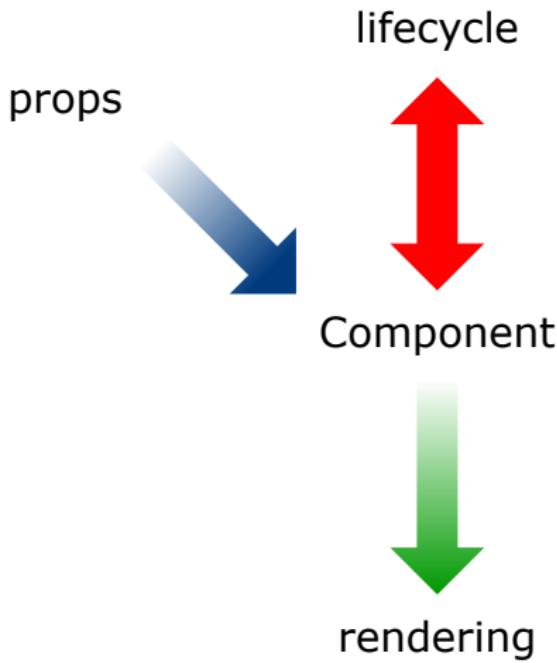


rendering

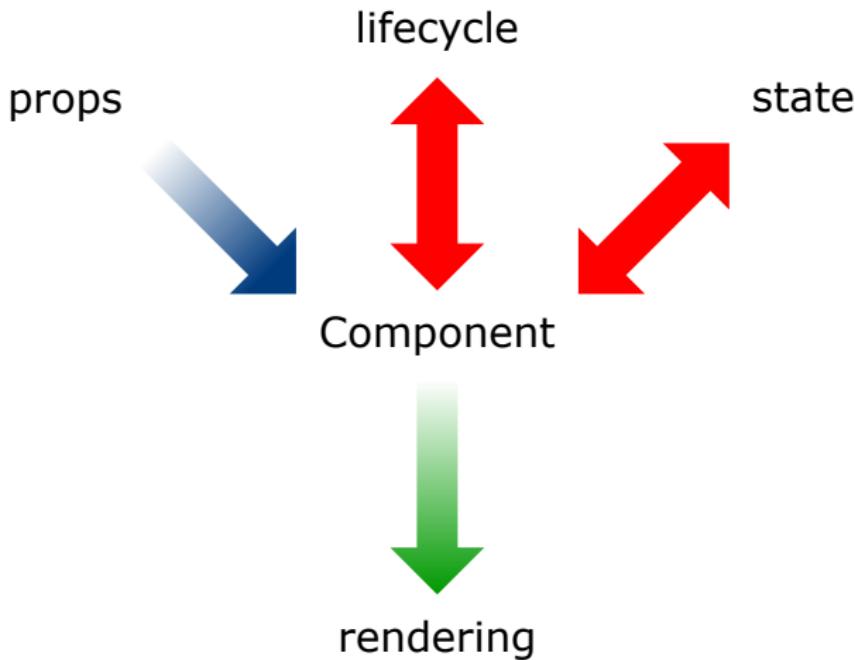
# React Components



# React Components



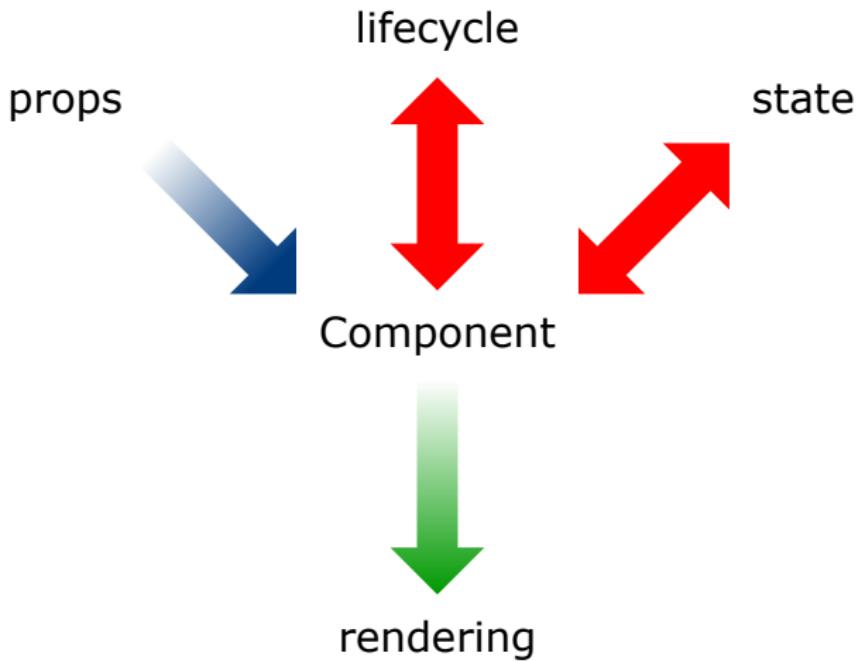
# React Components



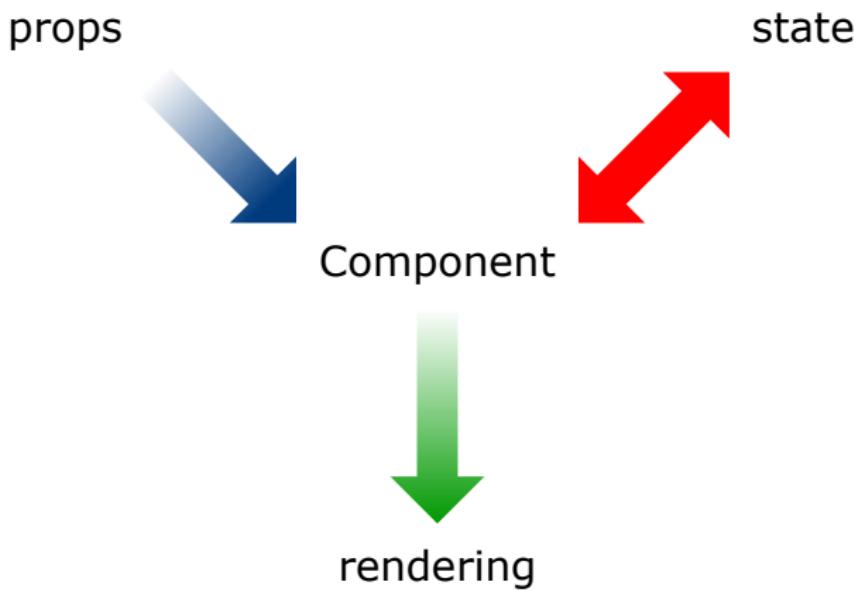
# React Components – Example

```
1 class Article extends React.Component {  
2  
3     render() {  
4         const article = this.props.article;  
5         return (  
6             <div key={article.id} className="col">  
7                 <div className="article">  
8                     <img src={article.image}/>  
9                     <span>{article.name}</span>  
10                    <span>{article.price}</span>  
11                </div>  
12            </div>  
13        );  
14    }  
15 }
```

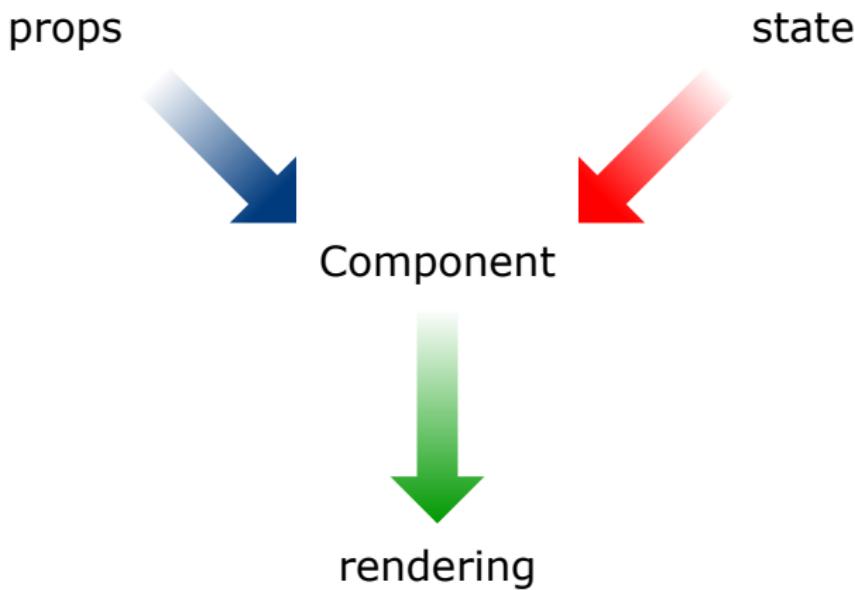
# Components



# Components



# Components



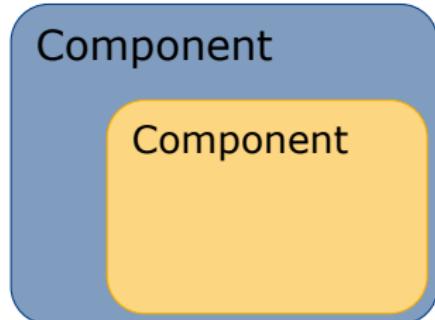
# Why Reflux?

declarative view abstraction

decoupling

easy testing

fast rendering



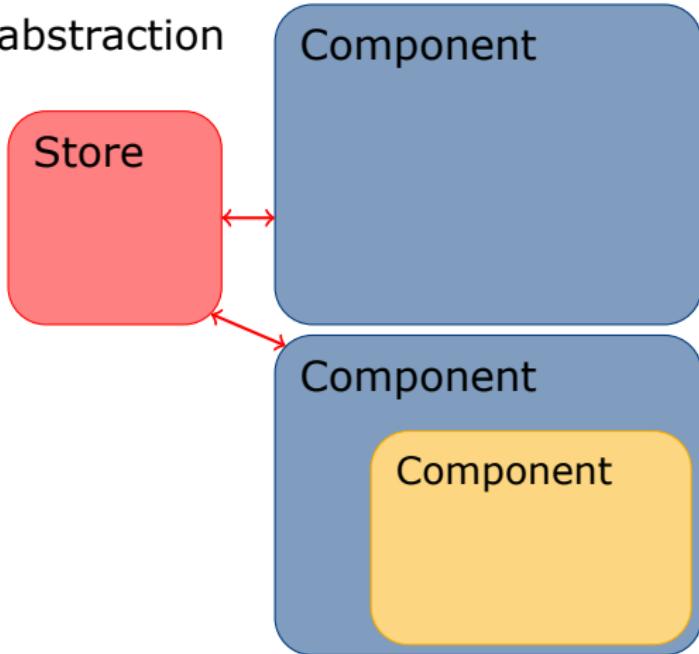
# Why Reflux?

declarative view abstraction

decoupling

easy testing

fast rendering



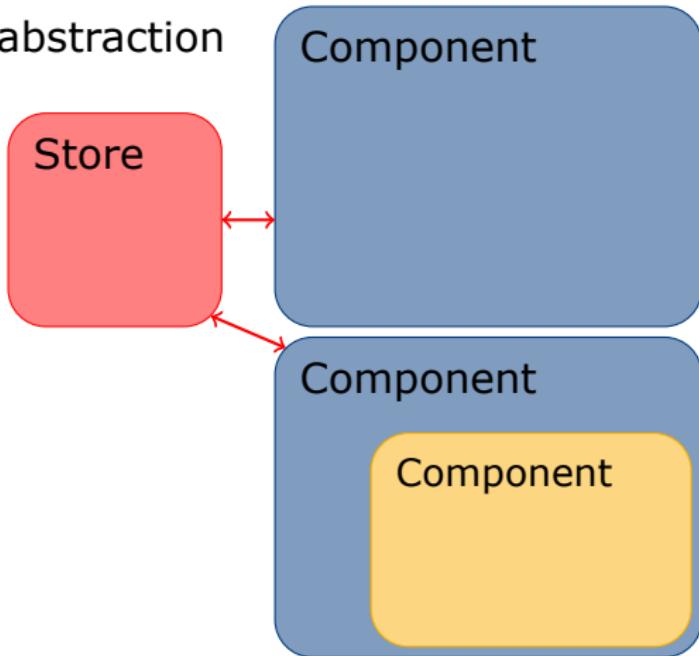
# Why Reflux?

declarative view abstraction

decoupling

easy testing

fast rendering



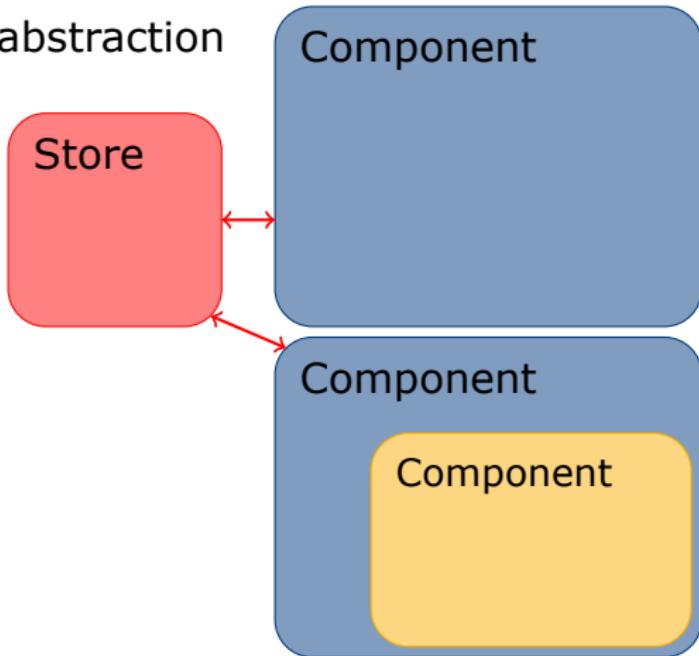
# Why Reflux?

declarative view abstraction

decoupling

easy testing

fast rendering



# Why Reflux?

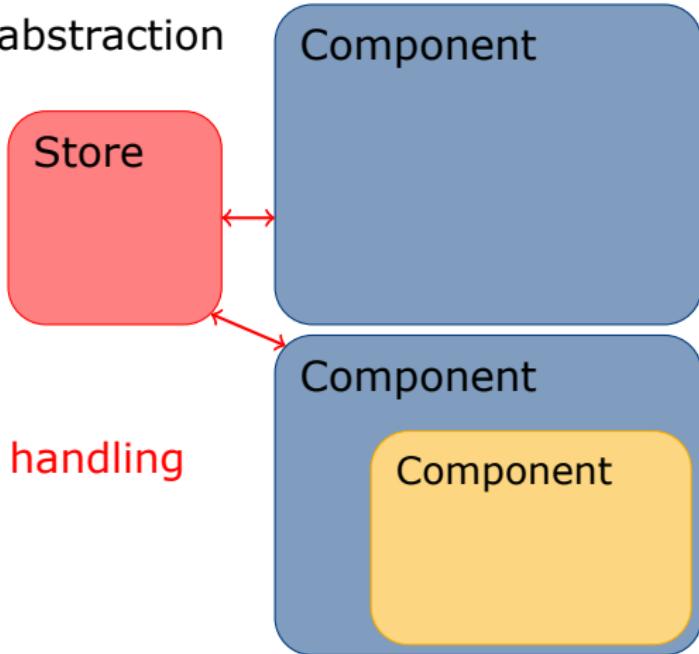
declarative view abstraction

decoupling

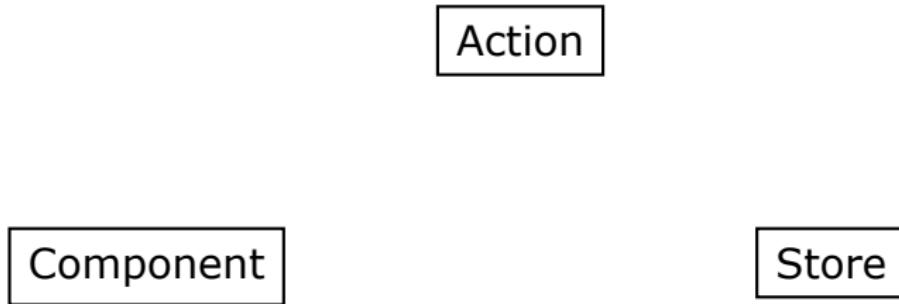
easy testing

fast rendering

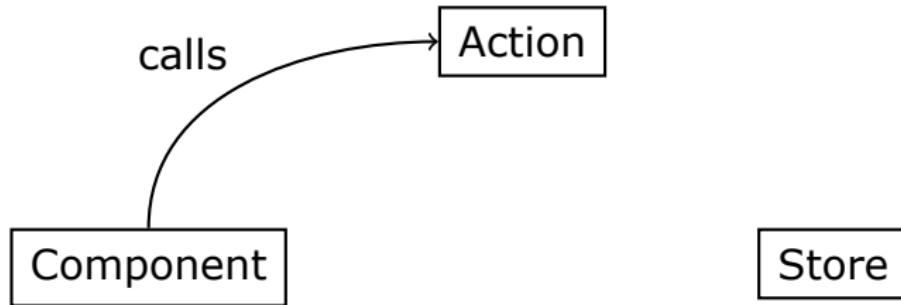
transversal state handling



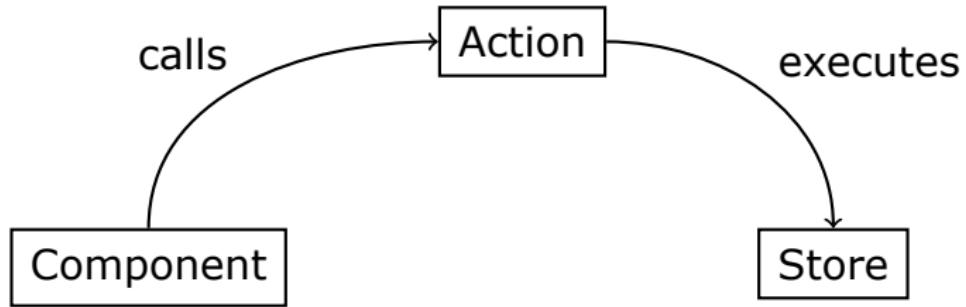
# Data Flow



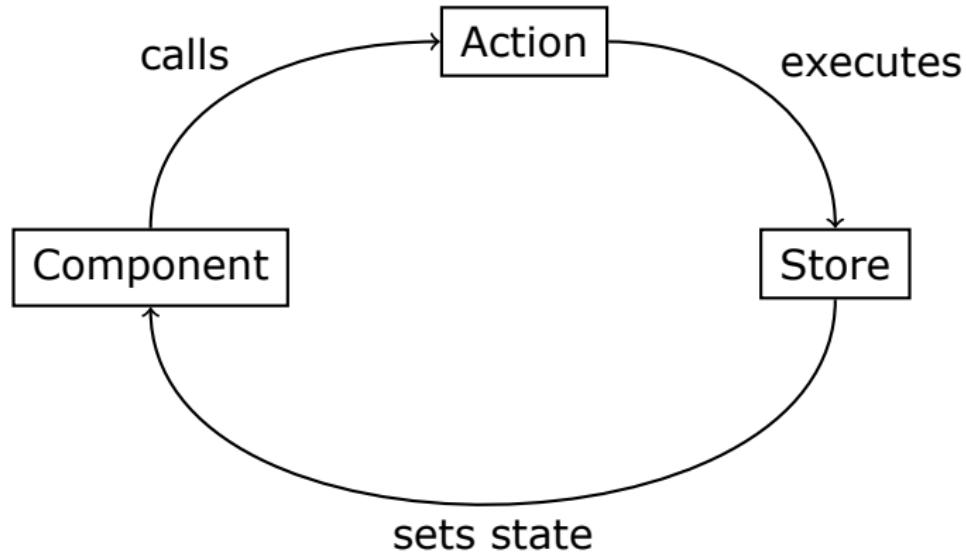
# Data Flow



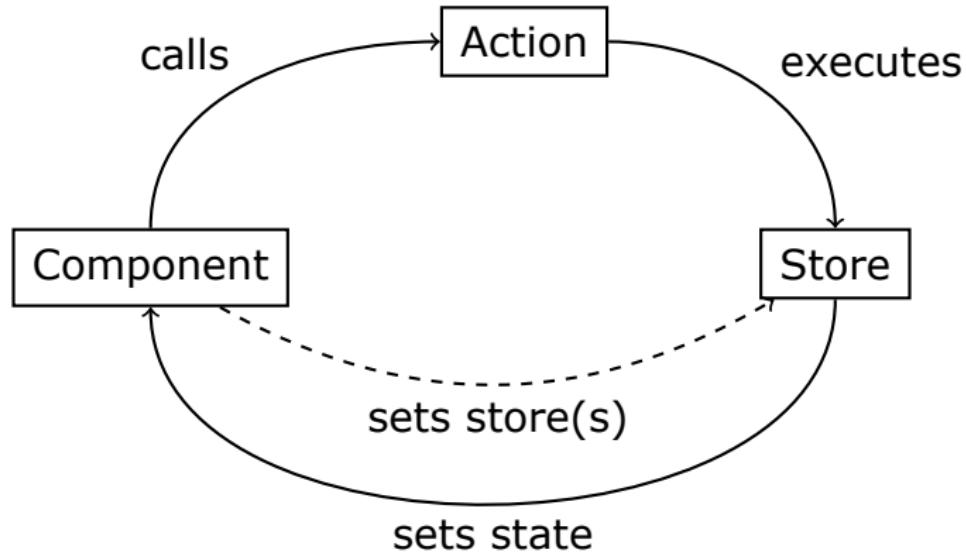
# Data Flow



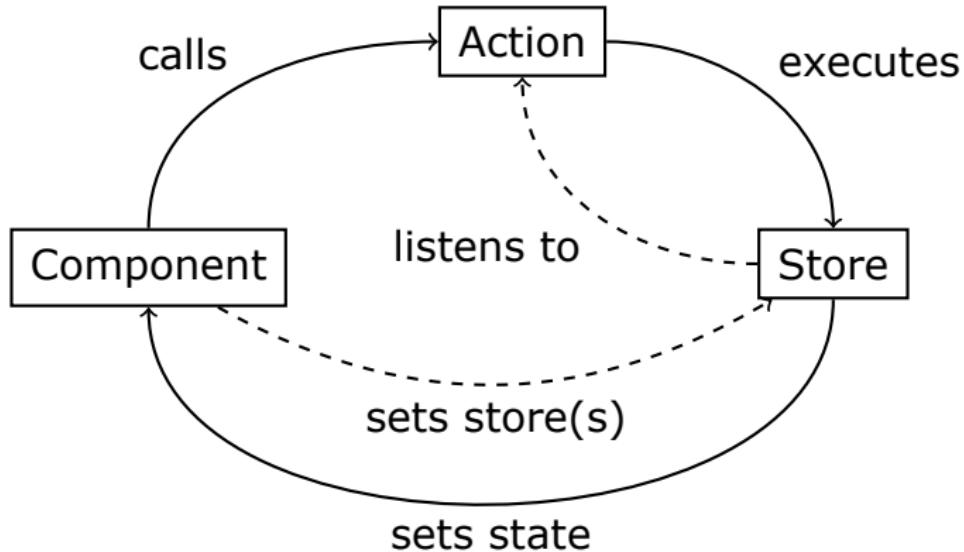
# Data Flow



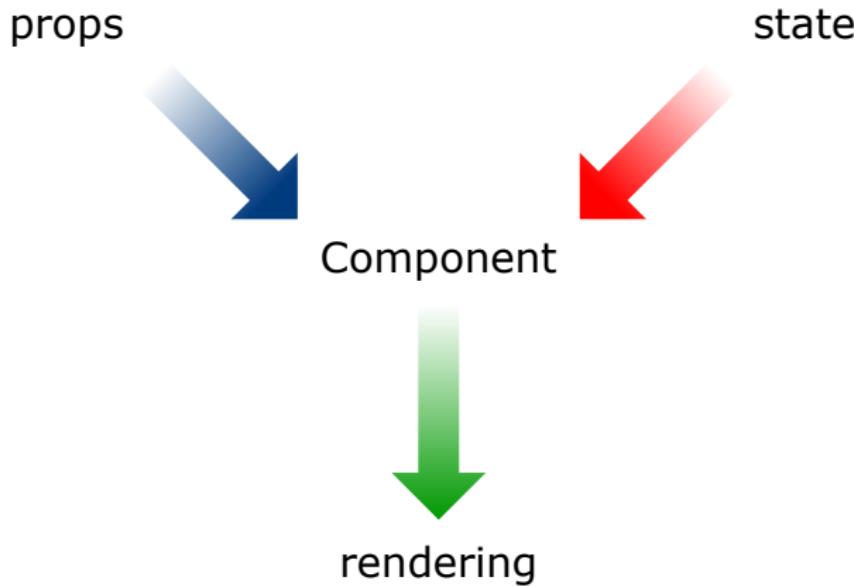
# Data Flow



# Data Flow



# Reflux Components



# Reflux Components

extend React components

props



state



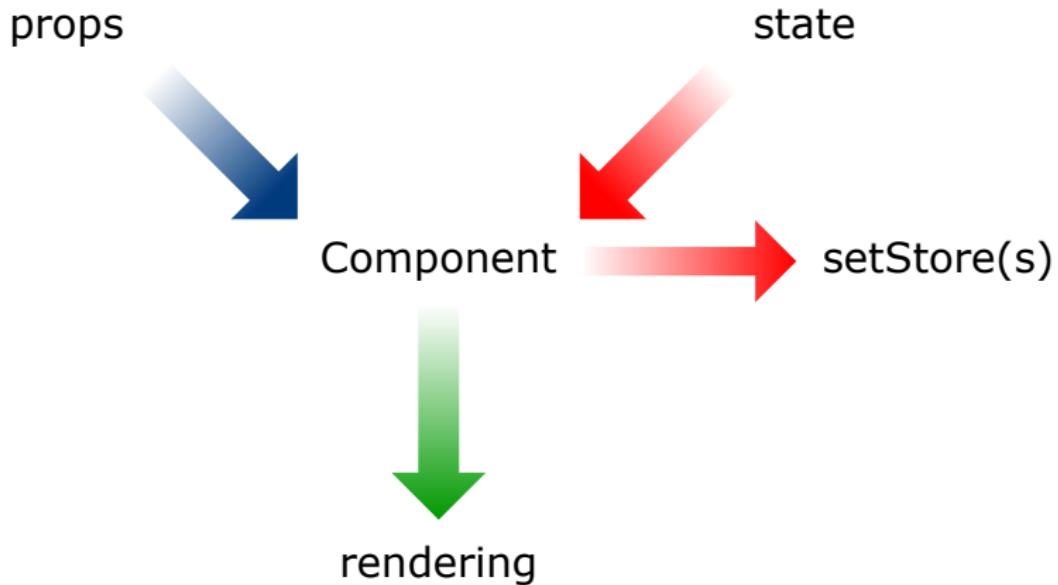
Component



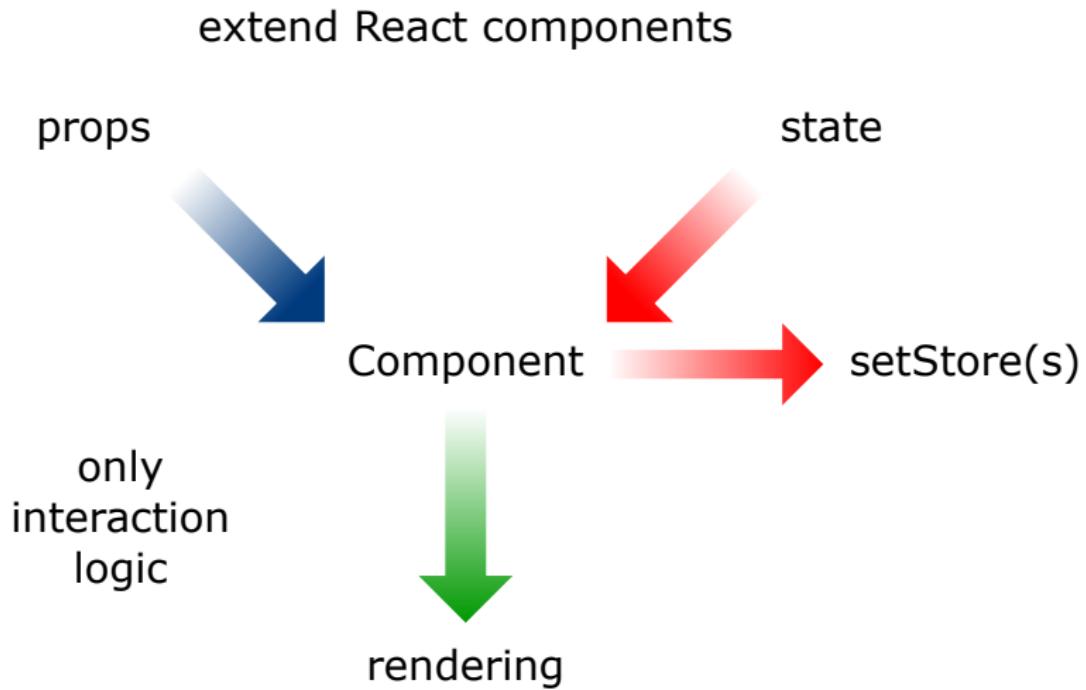
rendering

# Reflux Components

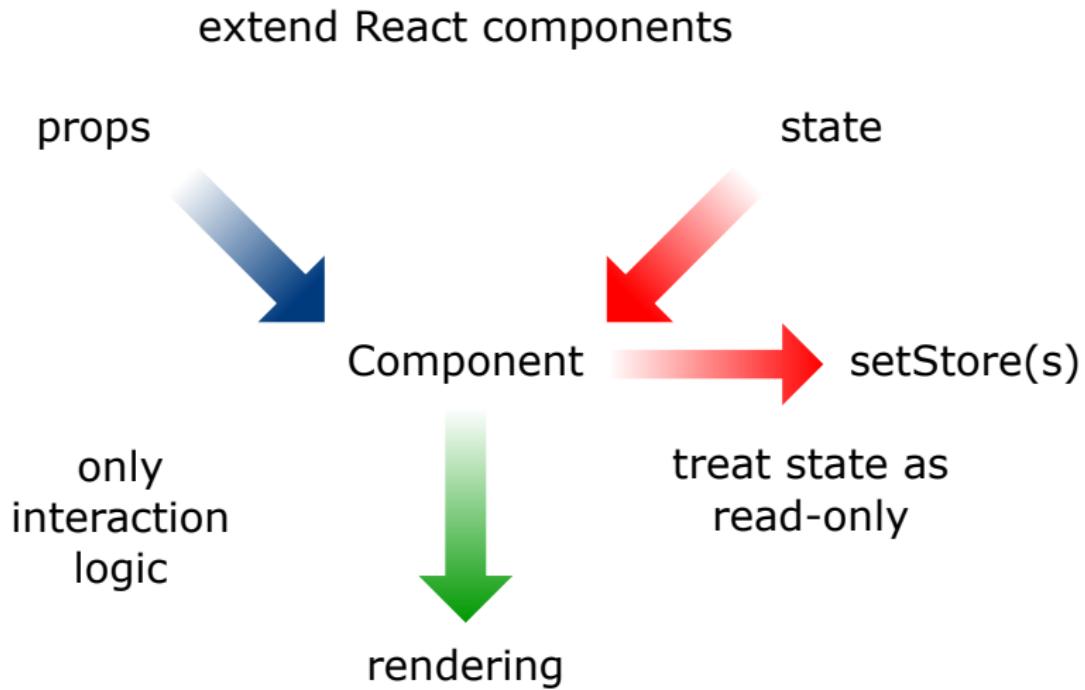
extend React components



# Reflux Components



# Reflux Components



# Actions

Action

# Actions

just a function

Action

# Actions

just a function

Action



called by  
component

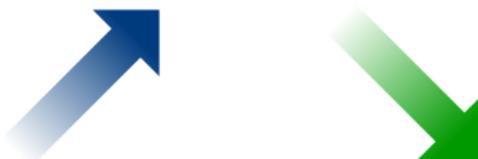
# Actions

just a function

Action

called by  
component

transports  
payload



# Stores

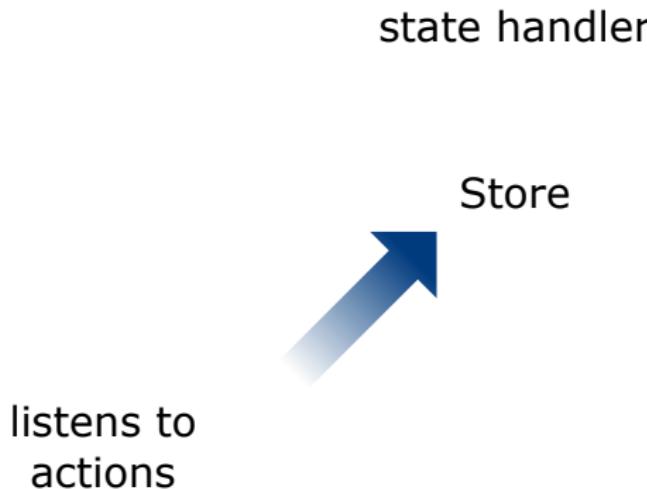
Store

# Stores

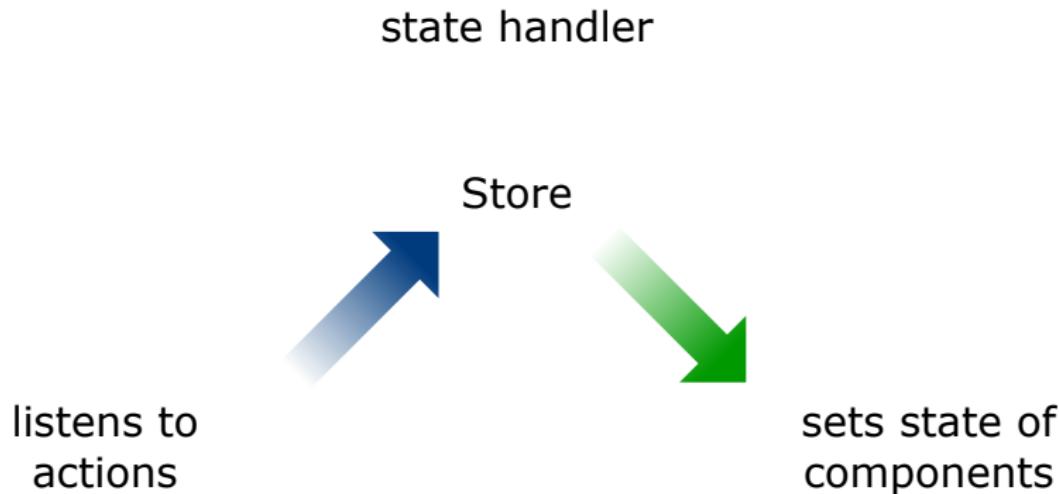
state handler

Store

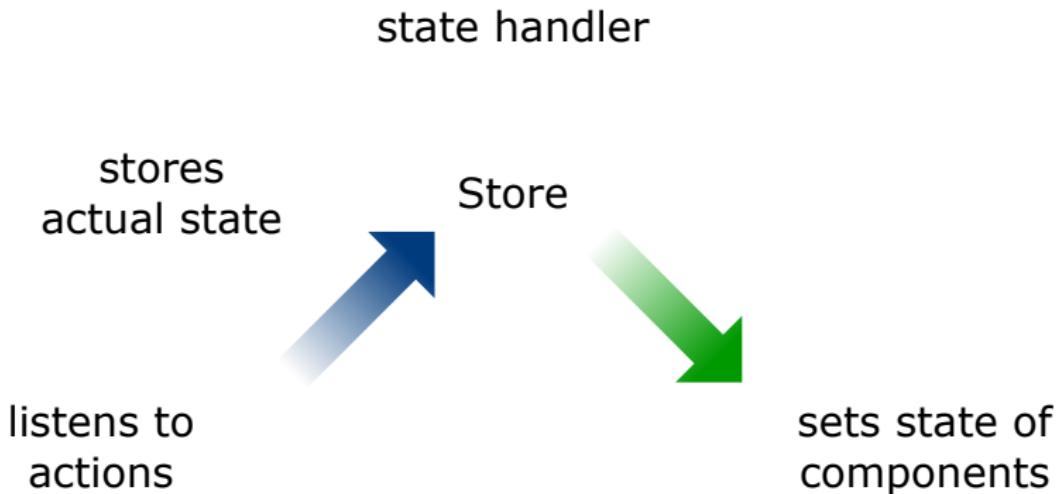
# Stores



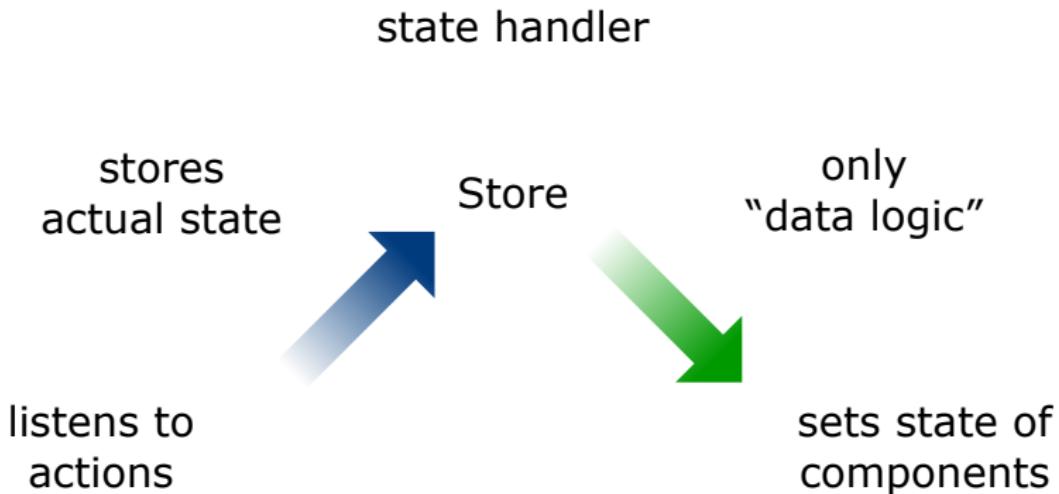
# Stores



# Stores



# Stores



# Reflux Components, Actions, and Stores – Example

```
1 const Actions = Reflux.createActions([
2     "loadArticles",
3     "addToBasket",
4     "removeFromBasket",
5     "clearBasket"
6 ]);
```

# Reflux Components, Actions, and Stores – Example

```
1 class ArticleStore extends Reflux.Store {  
2     constructor() {  
3         super();  
4         this.state = {articles: []};  
5         this.listenTo(Actions.loadArticles,  
6                         this.loadArticles);  
7     }  
8     loadArticles() {  
9         ArticleClient.loadArticles()  
10            .then(this.loadCompleted.bind(this))  
11            .catch(this.loadFailed.bind(this));  
12     }  
13     loadCompleted(newArticles) {  
14         this.setState({articles: newArticles});  
15     }  
16     loadFailed(response) {  
17         console.warn("Loading articles failed:", response);  
18     }  
19 }
```

# Reflux Components, Actions, and Stores – Example

```
1 class ExampleWebshop extends Reflux.Component {  
2     constructor(props) {  
3         super(props);  
4         this.store = ArticleStore;  
5     }  
6     componentDidMount() {  
7         Actions.loadArticles();  
8     }  
9     render() {  
10         return (  
11             <div>  
12                 <div className="row">  
13                     {this.state.articles.map(article =>  
14                         <Article article={article} />)}  
15                 </div>  
16             </div>);  
17     }  
18 }
```

# Problematic state handling

- Getter in stores

# Problematic state handling

- Getter in stores
- Unnecessary state in components

# Problematic state handling

- Getter in stores
- Unnecessary state in components
- State vs. props

# Getter in stores – Example

```
1 const BasketStore = Reflux.createStore({
2     constructor() {
3         this.basket = [];
4         this.listenTo(Actions.addToBasket, this.addToBasket
5             );
6     },
7     getBasket() {
8         return this.basket;
9     },
10    addToBasket(article) {
11        if (!this.isArticleInBasket(article)) {
12            this.basket = this.basket.concat([article])
13        }
14        this.trigger(this.basket());
15    }
16 }
17 };
```

# Getter in stores – Example

```
1 class Article extends Reflux.Component {  
2     render() {  
3         const article = this.props.article;  
4         return (  
5             <div className="article" >  
6                 <img src={article.image}/>  
7                 <span>{article.name}</span>  
8                 <span>{article.name}</span>  
9                 <div onClick={() => {  
10                     BasketStore.getBasket().indexOf(  
11                         article.id) < 0  
12                         ? "Add to basket"  
13                         : "Remove from basket"}  
14                 </div>  
15             </div>  
16         );  
17     }  
18 }
```

# Unnecessary state in components – Example

```
1 class Article extends Reflux.Component {  
2     constructor(props) {  
3         super(props);  
4         this.store = BasketStore;  
5     }  
6     render() {  
7         return (  
8             <div className="article" onClick={() => {  
9                 this.state.basket.indexOf(  
10                    this.props.article) < 0  
11                    ? "Add to basket"  
11                    : "Remove from basket"}  
12             </div>  
13         );  
14     }  
15 }
```

# State vs. props – Example

```
1 class Article extends Reflux.Component {  
2     constructor(props) {  
3         super(props);  
4     }  
5     render() {  
6         return (  
7             <div className="article" onClick={() => {  
8                 this.props.articleIsAlreadyInBasket  
9                     ? "Add to basket"  
10                     : "Remove from basket"}  
11             </div>  
12         );  
13     }  
14 }
```

# How to handle state

DON'T!

# How to handle state (if you have to)

- Access state from as few components as possible

# How to handle state (if you have to)

- Access state from as few components as possible
- Do not alter state in a component directly

# How to handle state (if you have to)

- Access state from as few components as possible
- Do not alter state in a component directly
- Extract state on a high level and hand it to child components via props

# How to handle state (if you have to)

- Access state from as few components as possible
- Do not alter state in a component directly
- Extract state on a high level and hand it to child components via props
- Stateless components

# Stateless component – Example

```
1 const ExampleWebshopHeader = ({sumPrice}) => {
2     return (
3         <div className="row webshop-header">
4             
6             <div className="basket"
7                 onClick={() => Actions.clearBasket()}>
8                 <span>{sumPrice + "€"}</span>
9             </div>
10        </div>
11    );
12
13 ExampleWebshopHeader.propTypes = {
14     sumPrice: PropTypes.number.isRequired
15 };
16
17 export default ExampleWebshopHeader;
```

# React and Reflux @METRO

**METRO** Ausgewählter Markt METRO CASH & CARRY ZUSTELLEDEPOT GROSSRAUM MUENCHEN Sortiment durchsuchen [Bestellstatus ändern](#) Belieferung

Kategorien

Für Sie empfohlen [Alle anzeigen](#)

Deals der Woche METRO

Zustell Profitipp METRO

Kategorien

Fleisch

Molkereiprodukte

Non-Food

Fisch & Meeresfrüchte

Obst & Gemüse

Getränke

Feinkost

Convenience

Tiefkühl

Trockensortiment

Süßwaren, Gebäck & Knabbern

# Tech Stack @METRO

# Tech Stack @METRO

Docker



# Tech Stack @METRO

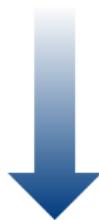
## REST API

Docker



# Tech Stack @METRO

`http://domain/service/endpoint`



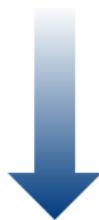
**REST API**

Docker



# Tech Stack @METRO

`http://domain/service/endpoint`



**REST API**

Docker



**kubernetes**

# Tech Stack @METRO

`http://domain/service/endpoint`



*cassandra*



**REST API**

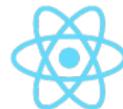
Docker



**kubernetes**

# Tech Stack @METRO

http://domain/service/endpoint



React



cassandra



**REST API**

Docker



kubernetes

# Tech Stack @METRO



Docker



`http://domain/service/endpoint`



**REST API**



React

Java 8



kubernetes

# Tech Stack @METRO

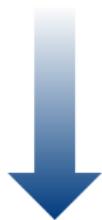


*cassandra*

`http://domain/service/endpoint`



React



Java 8

**REST API**

Docker



kubernetes

# Tech Stack @METRO



Docker



kubernetes

http://domain/service/endpoint



**REST API**



Java 8

React



Scala

# Tech Stack @METRO



Docker



kubernetes

http://domain/service/endpoint



**REST API**



Java 8

React



Clojure

Scala

The Scala logo, which consists of three red horizontal bars of increasing height followed by the word "Scala" in a bold, black, sans-serif font.

# Tech Stack @METRO



Docker



kubernetes

http://domain/service/endpoint



**REST API**



React



golang

Java 8



Clojure



# Tech Stack @METRO



Docker

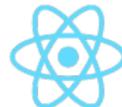


kubernetes

http://domain/service/endpoint



REST API



Java 8



React



Clojure



LATEX

