REACT AND REFLUX:

Components, Actions, Stores, and State

Philip Baues, Thomas Ströder
METRO

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

4.–6. October 2017,
TopConf Düsseldorf

Philip Baues and Thomas Ströder

Drilled Down Webshop Example

Why React?

Why React?
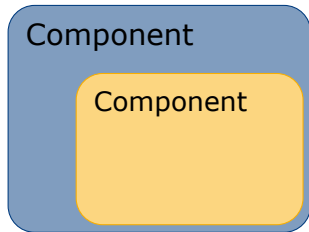
declarative view abstraction

METRO

Why React?

declarative view abstraction

Component

Why React?
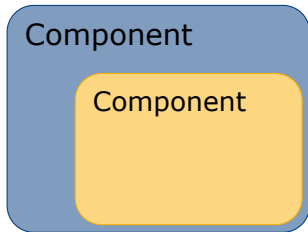
declarative view abstraction
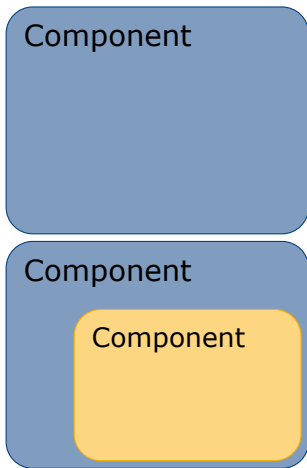
METRO

Why React?

declarative view abstraction

decoupling

METRO

Why React?
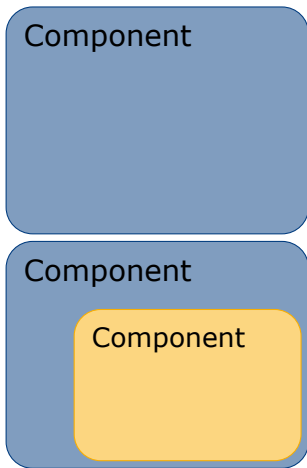
declarative view abstraction

decoupling

Why React?

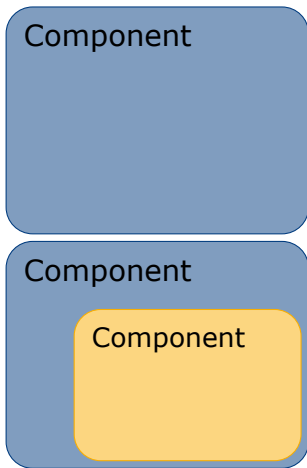declarative view abstraction

decoupling

easy testing

Component

Component

Component

METRO

Why React?

declarative view abstraction

decoupling

easy testing

fast rendering

Component

Component

Component

METRO

React Components

Component

React Components
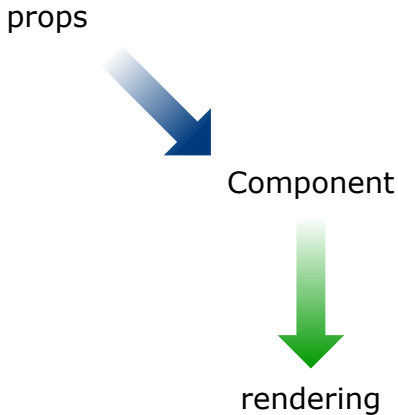
Component

↓

rendering

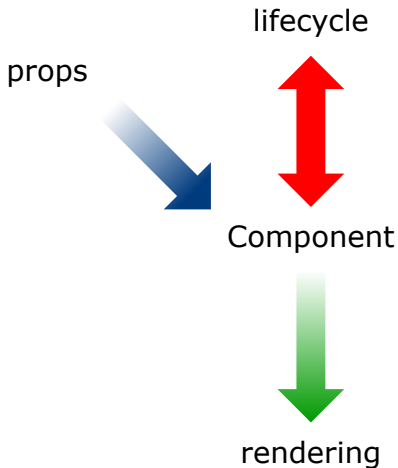# React Components
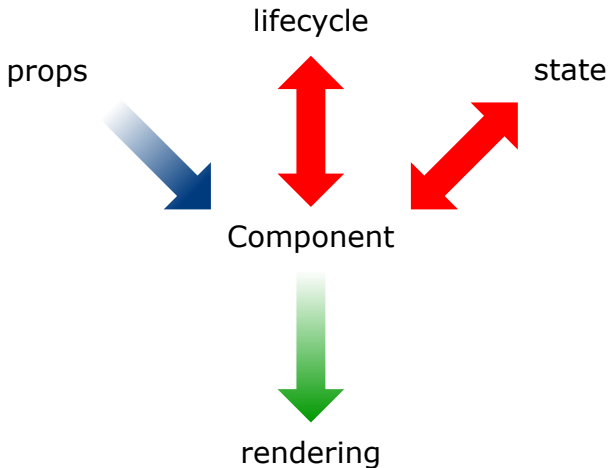
props

Component

rendering

METRO

React Components

METRO

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 }
```

Drilled Down Webshop Example

© METRO AG. P. Baues, T. Ströder: React and Reflux

Components

lifecycle

props

state

Component

rendering

METRO

Components

Components

props   state

Component

rendering

METRO

Why Reflux?

declarative view abstraction

decoupling

easy testing

fast rendering

Component

Component

Component

METRO

Why Reflux?

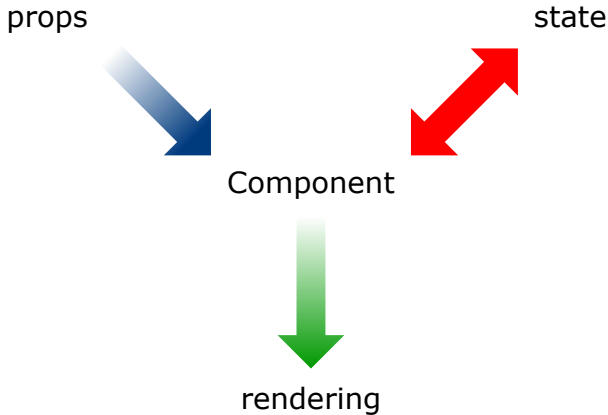declarative view abstraction

decoupling

easy testing

fast rendering
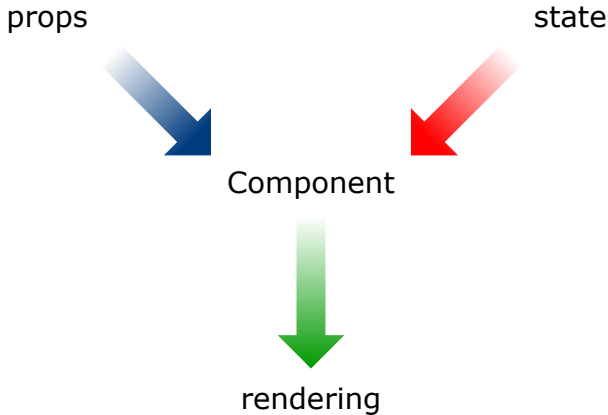
Component

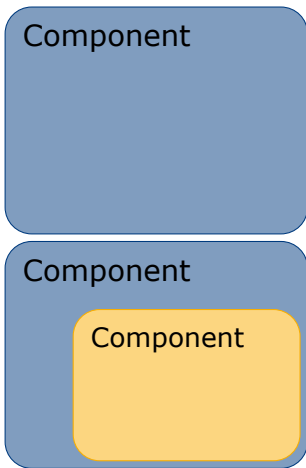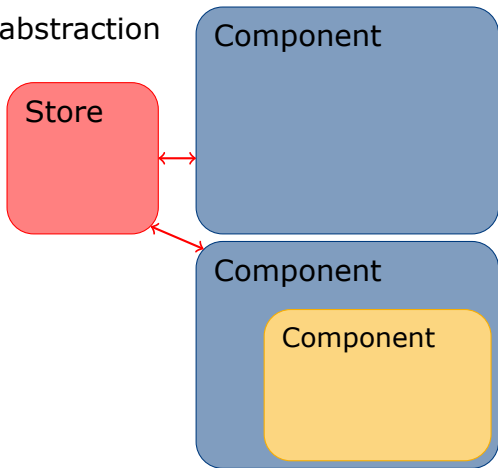Store

Component

Component

METRO

Why Reflux?

declarative view abstraction

decoupling

easy testing

fast rendering

METRO

Why Reflux?

declarative view abstraction

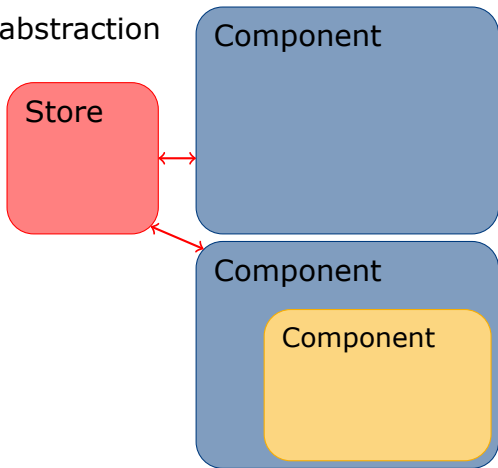decoupling

easy testing

fast rendering

Component
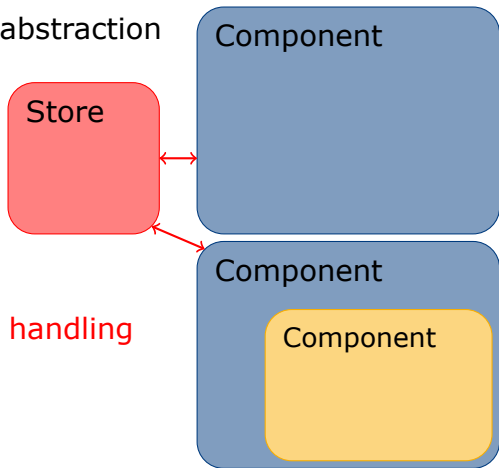
Store

Component

Component

METRO
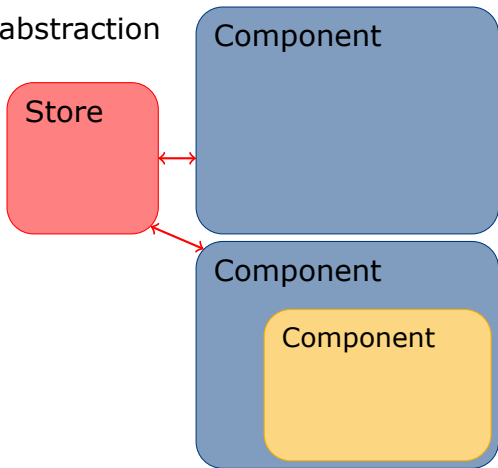
Why Reflux?

declarative view abstraction

decoupling

easy testing

fast rendering

transversal state handling

Data Flow

Action

Component

Store

METRO

Data Flow

calls

Action

Component

Store

METRO
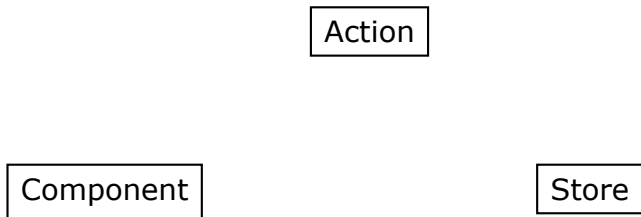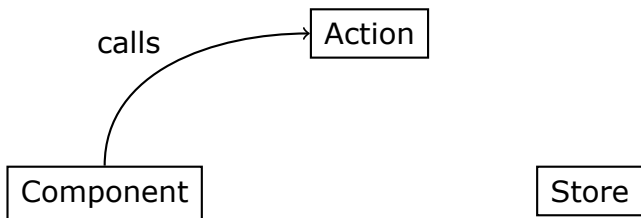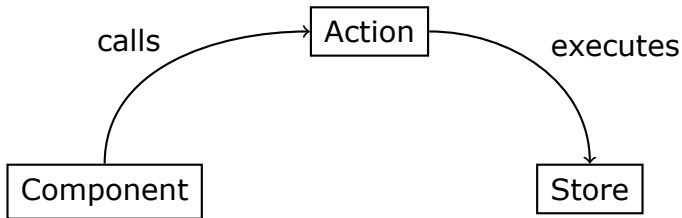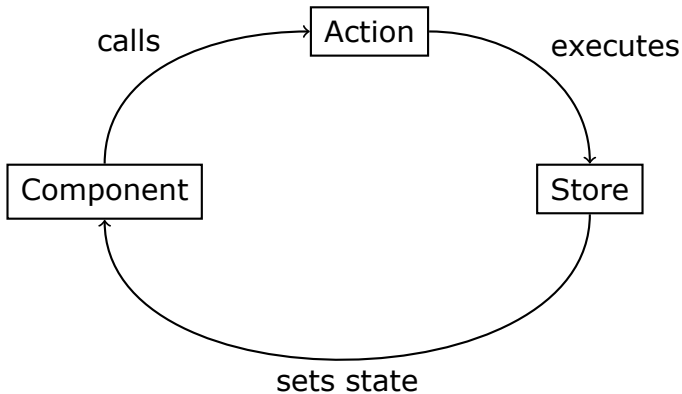
Data Flow
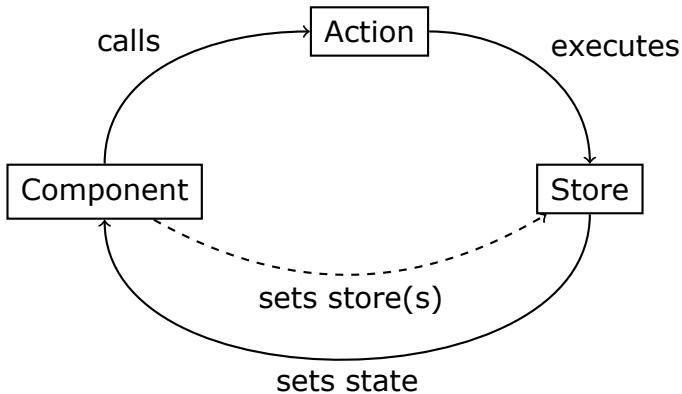
Data Flow

Data Flow

Data Flow

Reflux Components
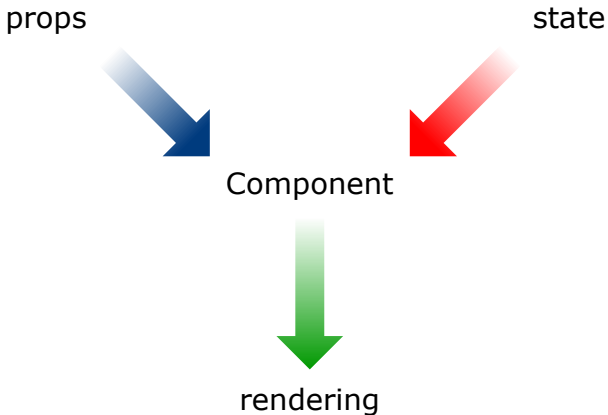
props   state

Component

rendering

Reflux Components

extend React components

props   state

Component

rendering

METRO

Reflux Components

extend React components

props   state

Component   setStore(s)

rendering

METRO

Reflux Components

extend React components

props   state

Component   setStore(s)

only
interaction
logic

rendering

METRO

Reflux Components

extend React components

props   state

Component   →   setStore(s)

only
interaction
logic

treat state as
read-only

rendering
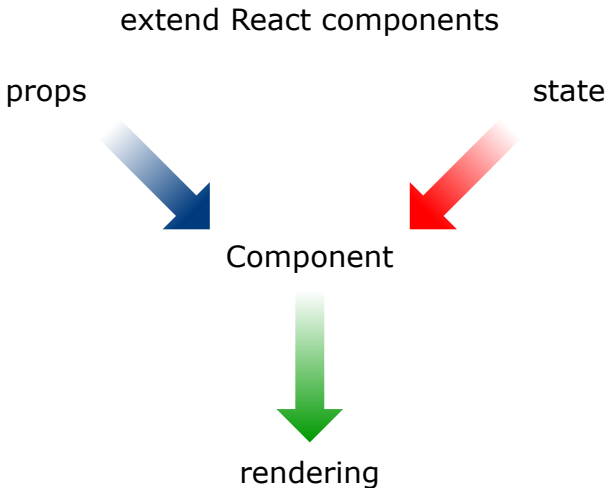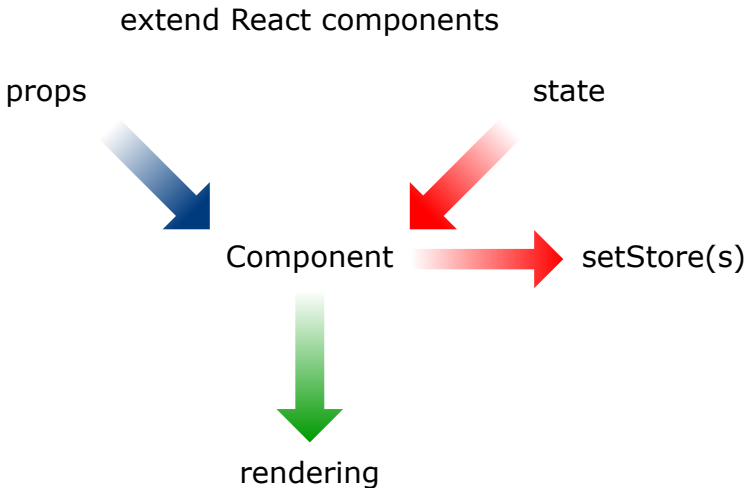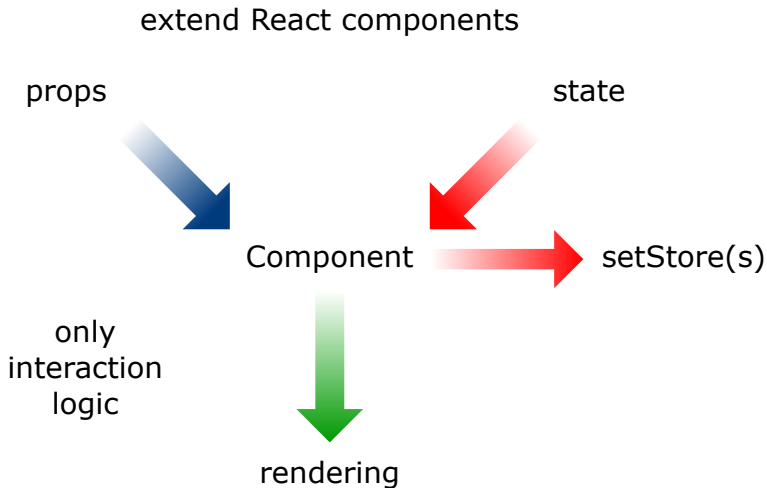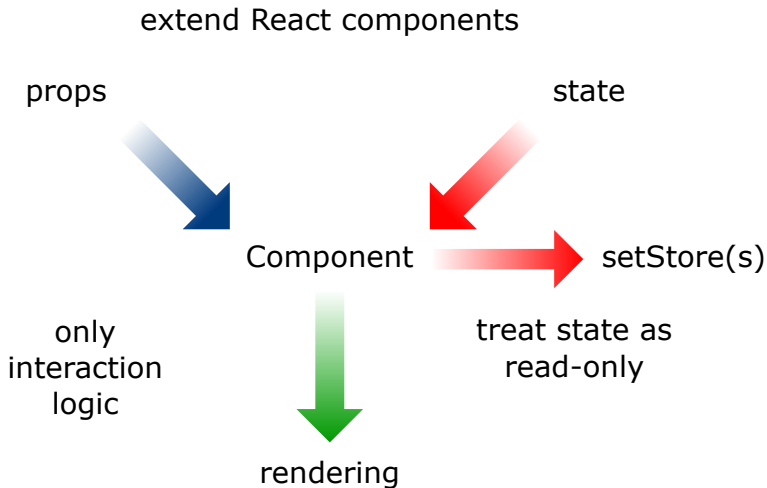
Actions

Action

Actions

just a function
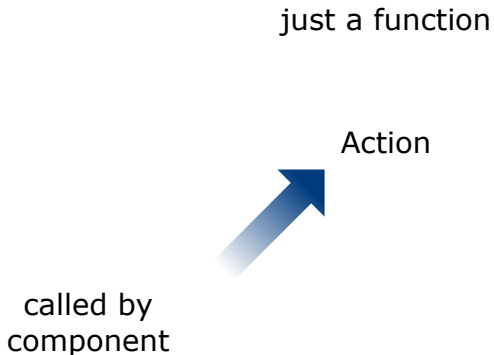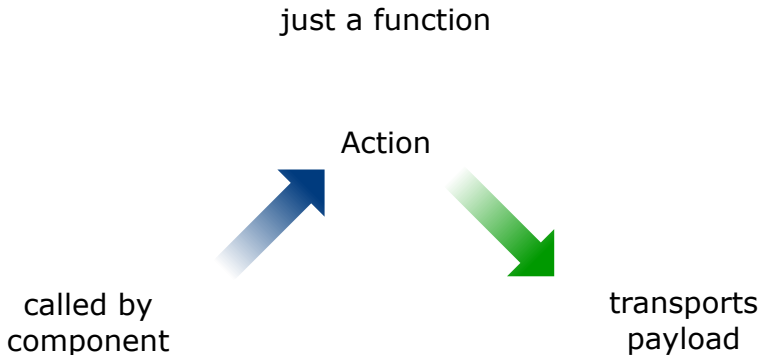
Action

METRO

Actions

just a function

Action

called by
component

METRO

Actions

just a function

Action

called by
component

transports
payload

METRO

Stores

Store

METRO

Stores

state handler

Store

METRO

Stores

state handler

Store

listens to
actions

METRO

Stores

state handler

Store

listens to
actions

sets state of
components

METRO

Stores

state handler

stores
actual state

Store

listens to
actions

sets state of
components

METRO

Stores

state handler

stores
actual state

Store

only
"data logic"

listens to
actions

sets state of
components

METRO
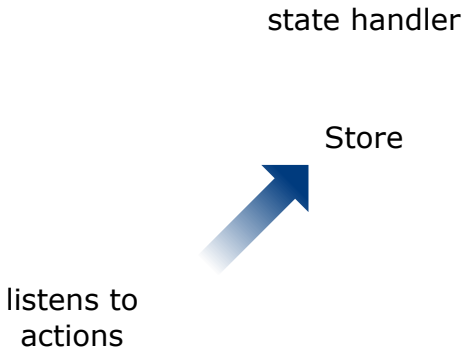
Reflux Components, Actions, and Stores – Example

```
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,
                this.loadArticles);
6       }
7       loadArticles() {
8           ArticleClient.loadArticles()
9             .then(this.loadCompleted.bind(this))
10            .catch(this.loadFailed.bind(this));
11      }
12      loadCompleted(newArticles) {
13          this.setState({articles: newArticles});
14      }
15      loadFailed(response) {
16          console.warn("Loading articles failed:", response);
17      }
```
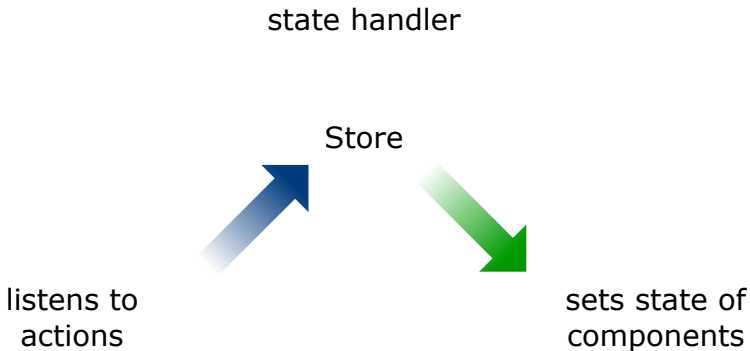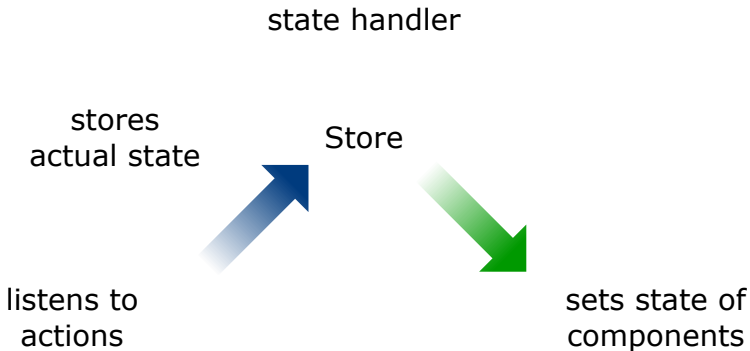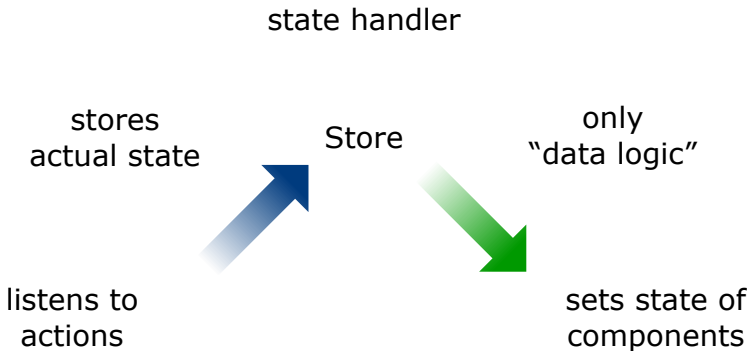
METRO

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 }
```

METRO

Drilled Down Webshop Example

Problematic State Handling

- Getter in stores

METRO

Problematic State Handling

- Getter in stores
- Unnecessary state in components

METRO

Problematic State Handling

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

METRO

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
11     addToBasket(article) {
12         if (!this.isArticleInBasket(article)) {
13             this.basket = this.basket.concat([article])
14         }
15         this.trigger(this.basket());
16     }
17 };
```

METRO

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(
                            article.id) < 0
11                      ? "Add to basket"
12                      : "Remove from basket"}
13                  </div>
14              </div>
15          );
16      }
17  }
```

METRO

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(
                        this.props.article) < 0
10                 ? "Add to basket"
11                 : "Remove from basket"}
12             </div>
13         );
14     }
15 }
```

METRO

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  }
```

METRO

How to Handle State

How to Handle State

DON'T!

How to Handle State
(if you have to)

- Access state from as few components as possible

METRO

How to Handle State
(if you have to)

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

METRO

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

METRO

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

METRO

Stateless Component – Example

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

METRO

React and Reflux @METRO

Tech Stack @METRO

 P. Baues, T. Ströder: React and Reflux
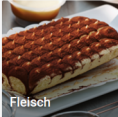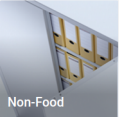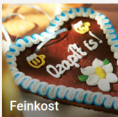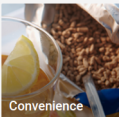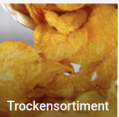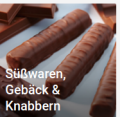
Tech Stack @METRO

Docker

Tech Stack @METRO

REST API

Docker

METRO

Tech Stack @METRO

`http://domain/service/endpoint`

REST API

Docker

METRO

Tech Stack @METRO

`http://domain/service/endpoint`

REST API

Docker

kubernetes

METRO

Tech Stack @METRO

`http://domain/service/endpoint`

cassandra

REST API

Docker

kubernetes

METRO

Tech Stack @METRO

`http://domain/service/endpoint` React

cassandra

REST API

Docker

kubernetes

METRO

Tech Stack @METRO

`http://domain/service/endpoint` React

Java 8

cassandra

REST API

Docker

kubernetes

METRO

Tech Stack @METRO

`http://domain/service/endpoint` React

Java 8

cassandra

REST API

Docker

Scala

kubernetes

METRO

Tech Stack @METRO

`http://domain/service/endpoint` React

cassandra

Java 8   golang

REST API

Docker

Scala

kubernetes

METRO

Tech Stack @METRO

Tech Stack @METRO

`http://domain/service/endpoint` React

cassandra

Java 8 golang

REST API Haskell

Docker Clojure

Scala

kubernetes

METRO

Tech Stack @METRO

Tech Stack @METRO

`http://domain/service/endpoint`   React

cassandra

Java 8

golang

REST API   Haskell

Docker   Cloiure

github.com/techmetro
metro-cc.com/transformers

Scala

kubernetes   LaTeX

Thank you!

METRO

Philip Baues   philip.baues@metrosystems.net
Thomas Ströder   thomas.stroeder@metrosystems.net

METRO   metro-cc.com/transformers

TOPCONF
Duesseldorf