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

Philip Baues, Thomas Ströder
METRO

TOPCONF
Duesseldorf

METRO

METRO

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

4.–6. October 2017,
TopConf Düsseldorf

Philip Baues and Thomas Ströder

Drilled down webshop example

METRO 3.88€

Aro H-Milch 3,5% 0.99

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

Aro Blütenhonig flüssig 2.89

The real honey experience.

METRO

Why React?

METRO

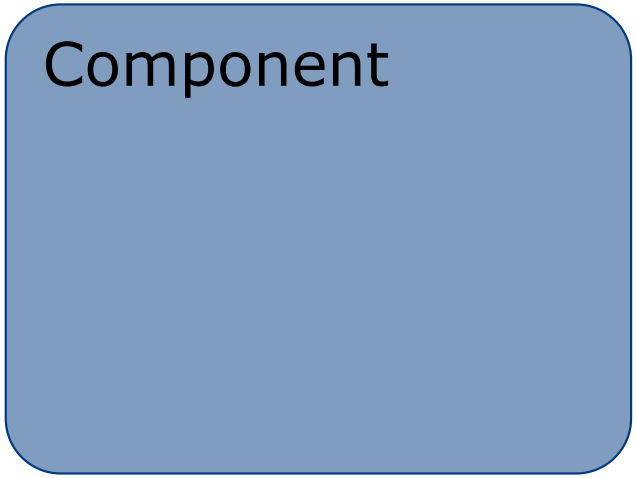
Why React?

declarative view abstraction

METRO

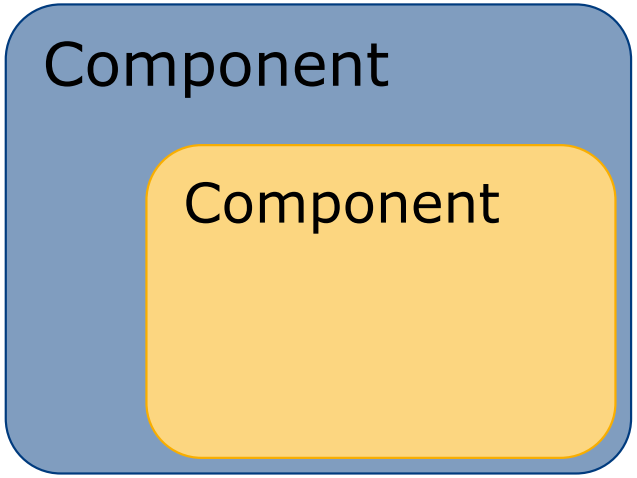
Why React?

declarative view abstraction



Why React?

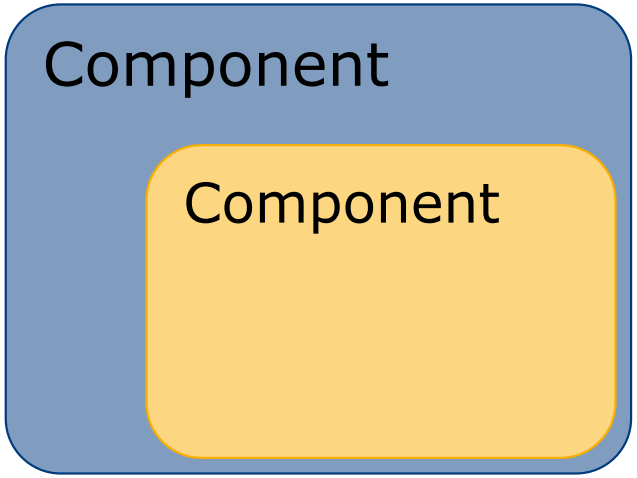
declarative view abstraction



Why React?

declarative view abstraction

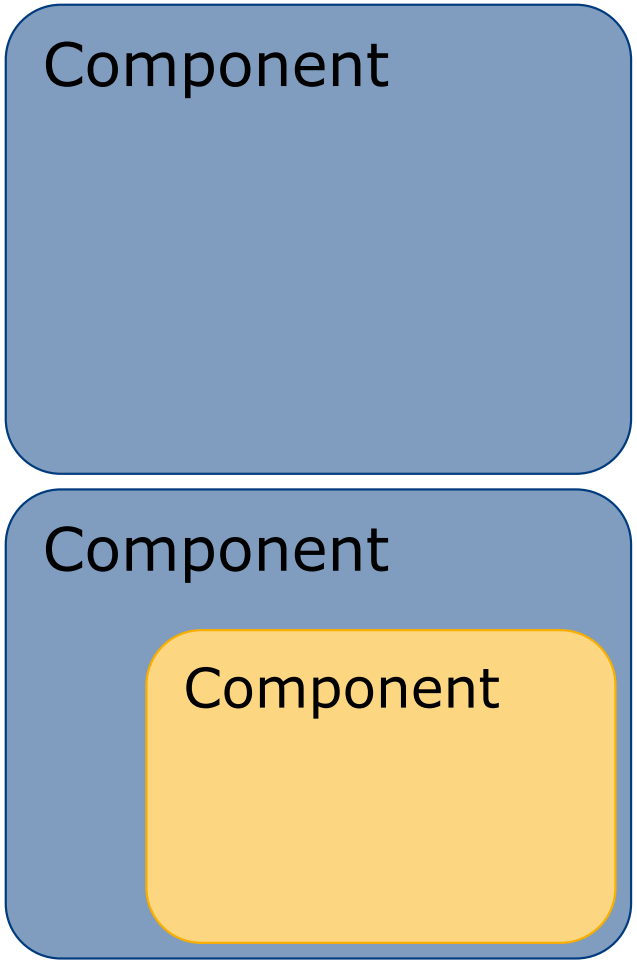
decoupling



Why React?

declarative view abstraction

decoupling

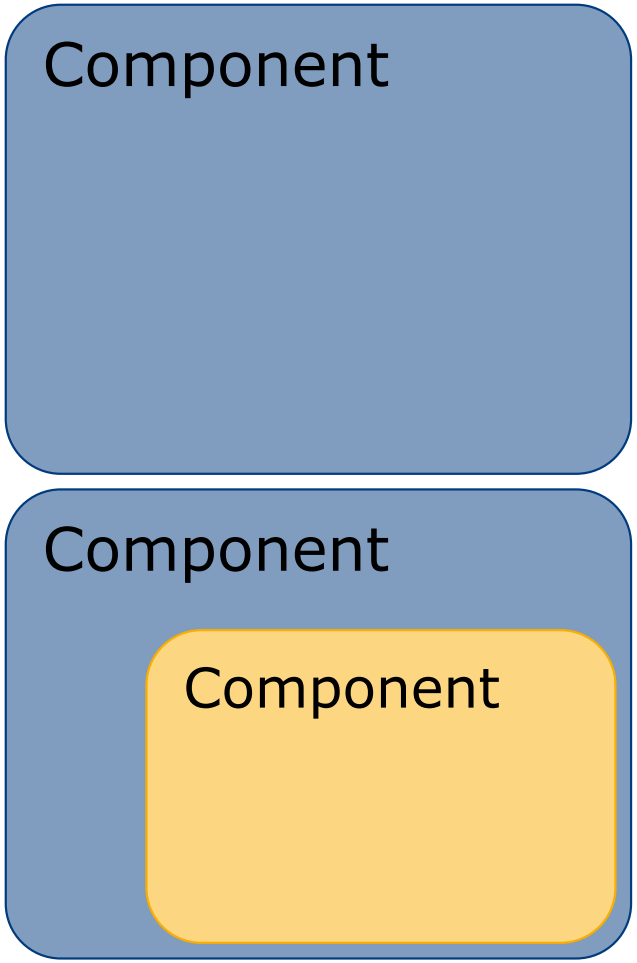


Why React?

declarative view abstraction

decoupling

easy testing



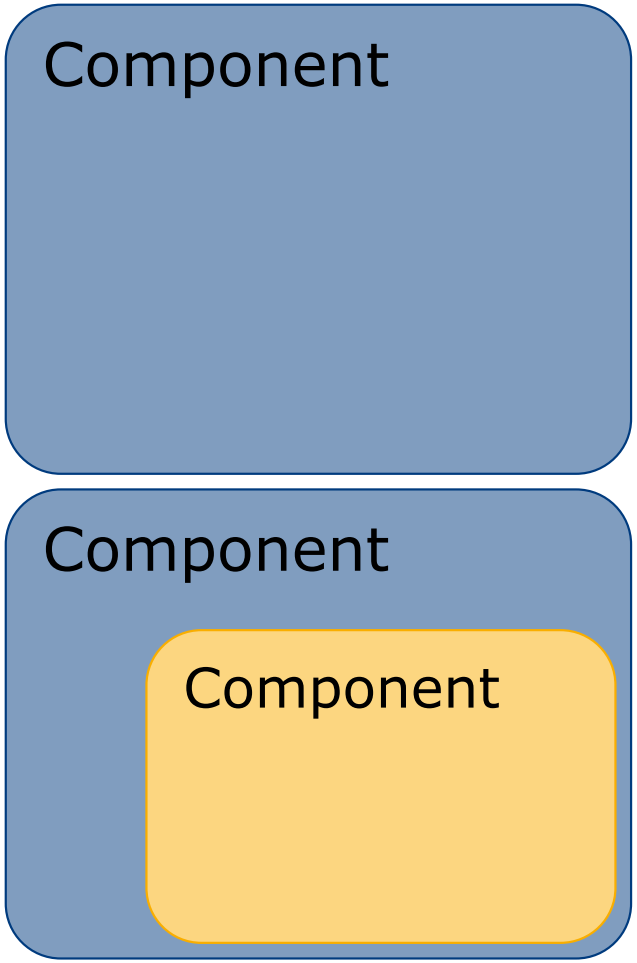
Why React?

declarative view abstraction

decoupling

easy testing

fast rendering



React Components

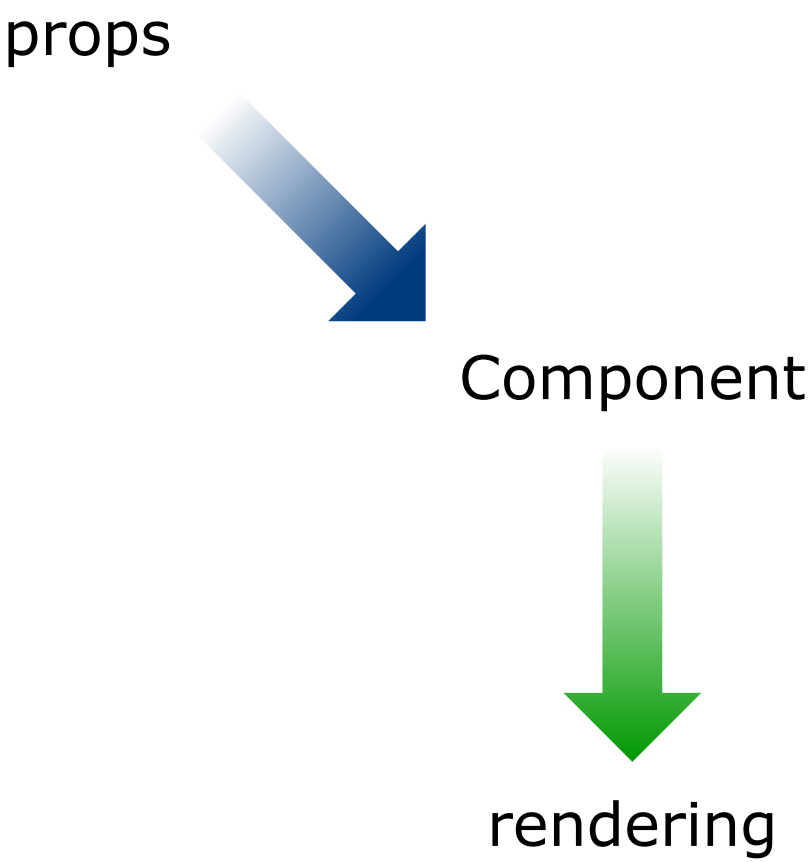
Component

React Components

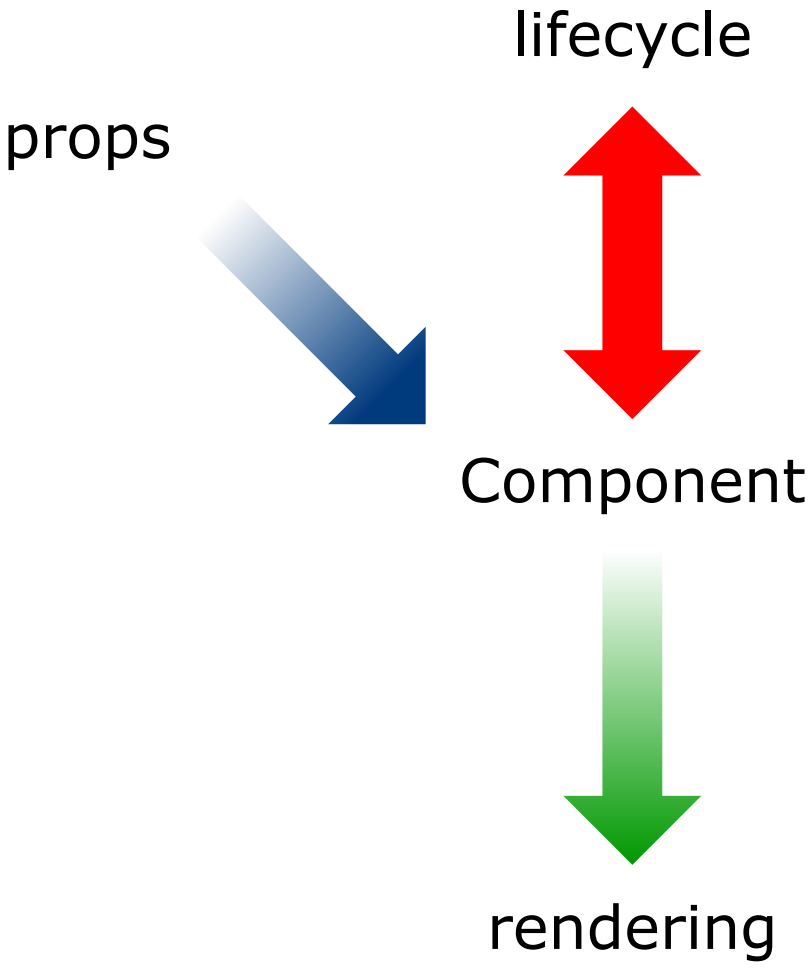
Component

rendering

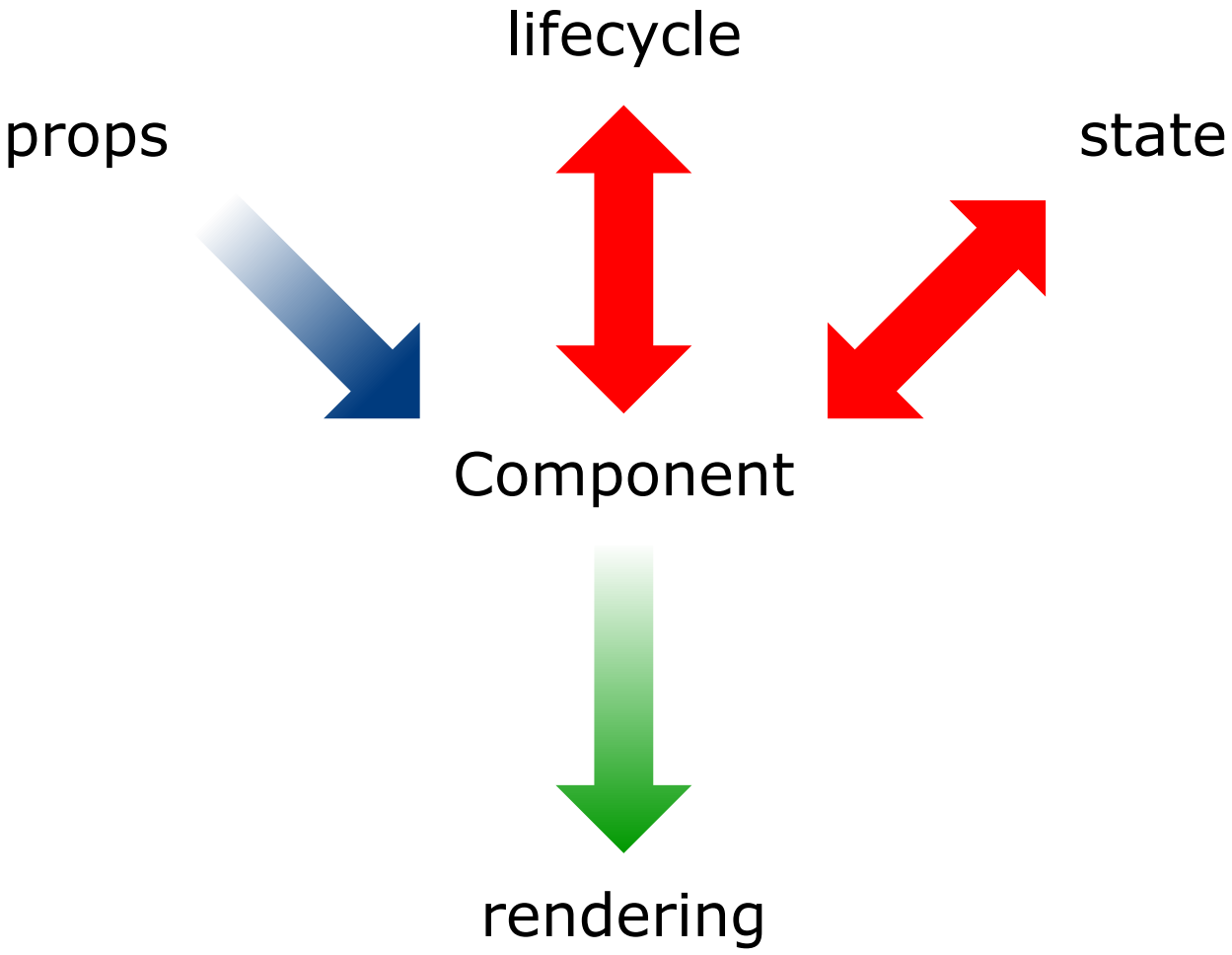
React Components



React Components



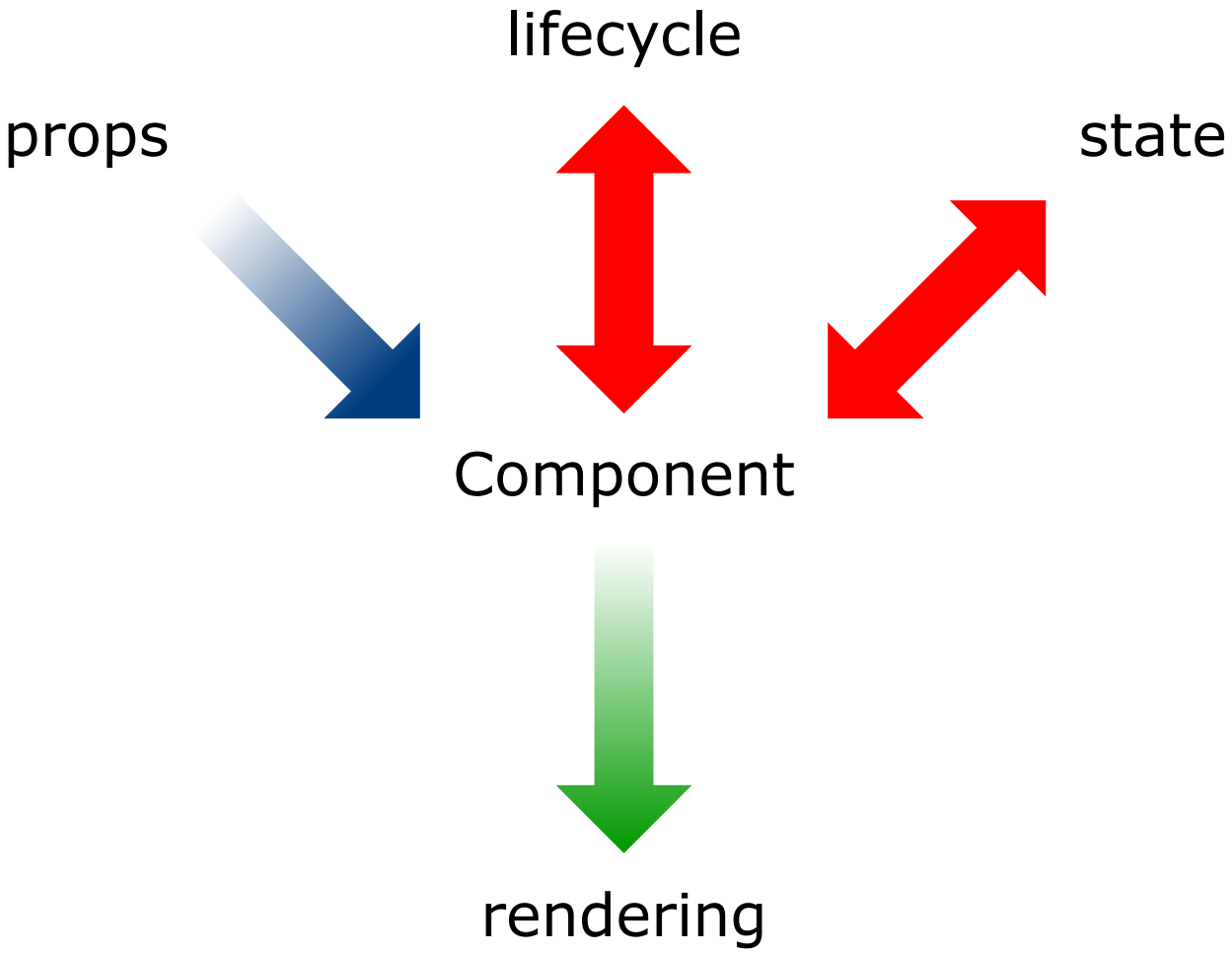
React Components



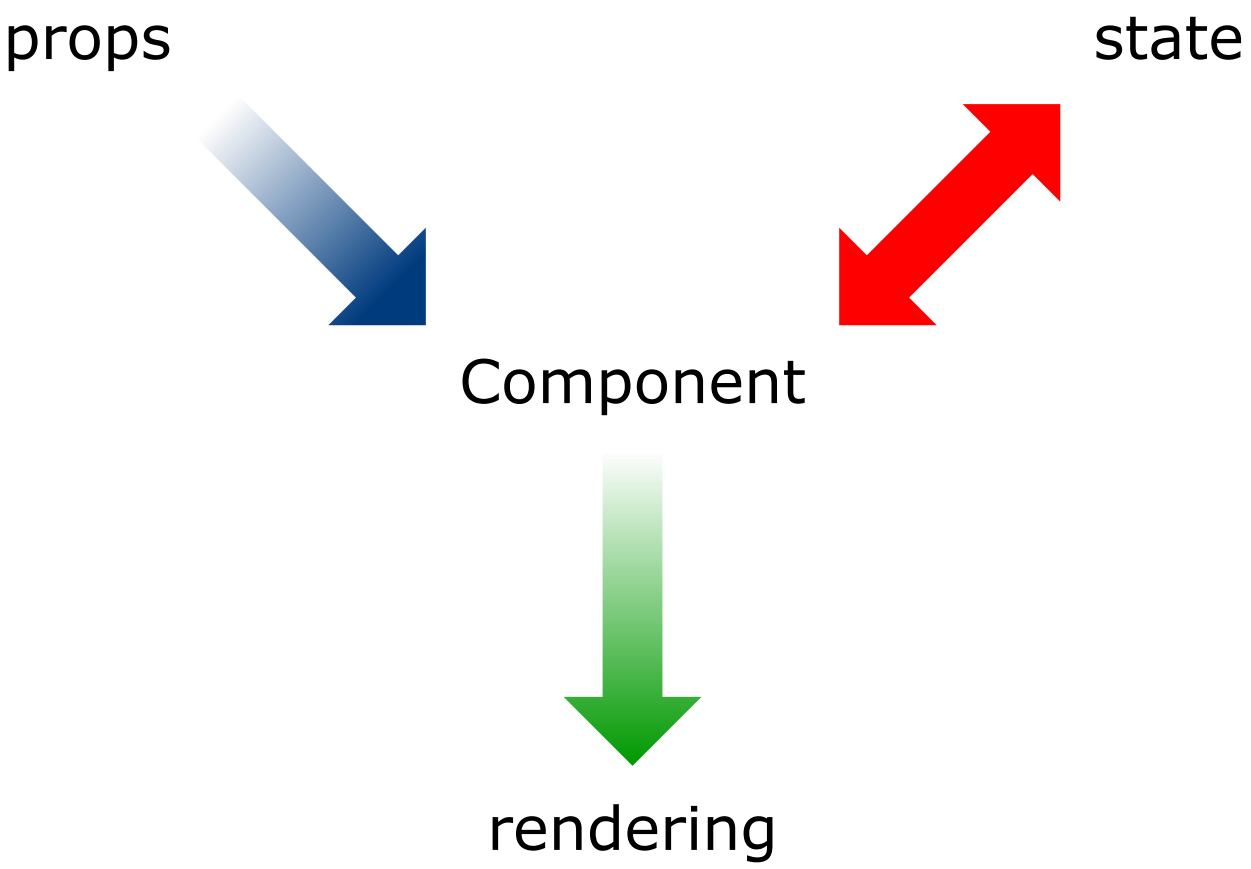
React Components – Example

```
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  }
```

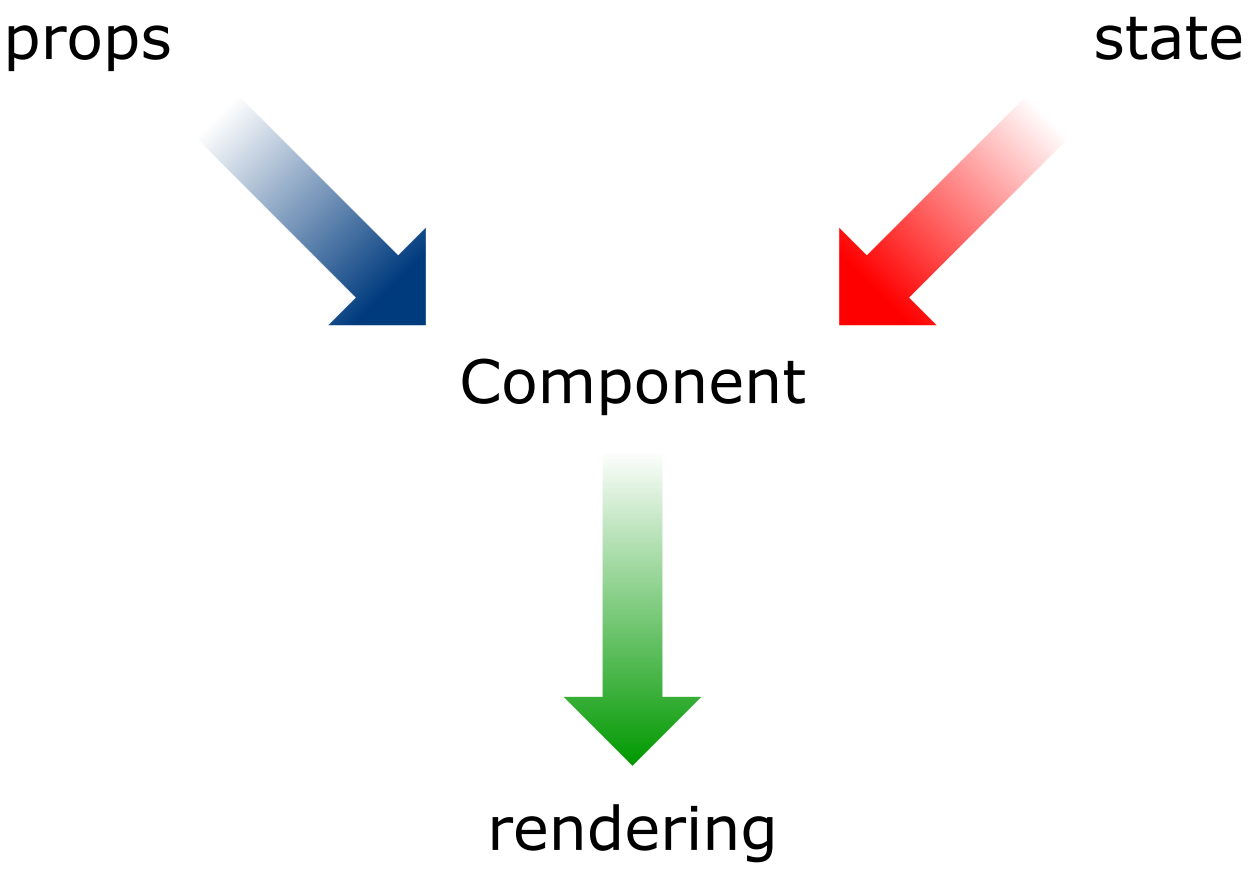
Components



Components



Components



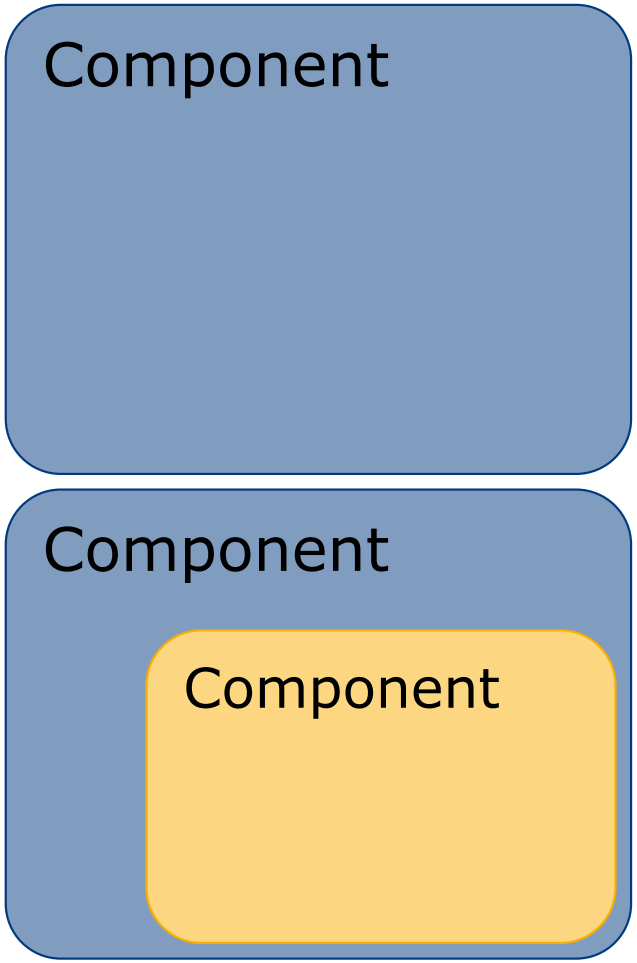
Why Reflux?

declarative view abstraction

decoupling

easy testing

fast rendering



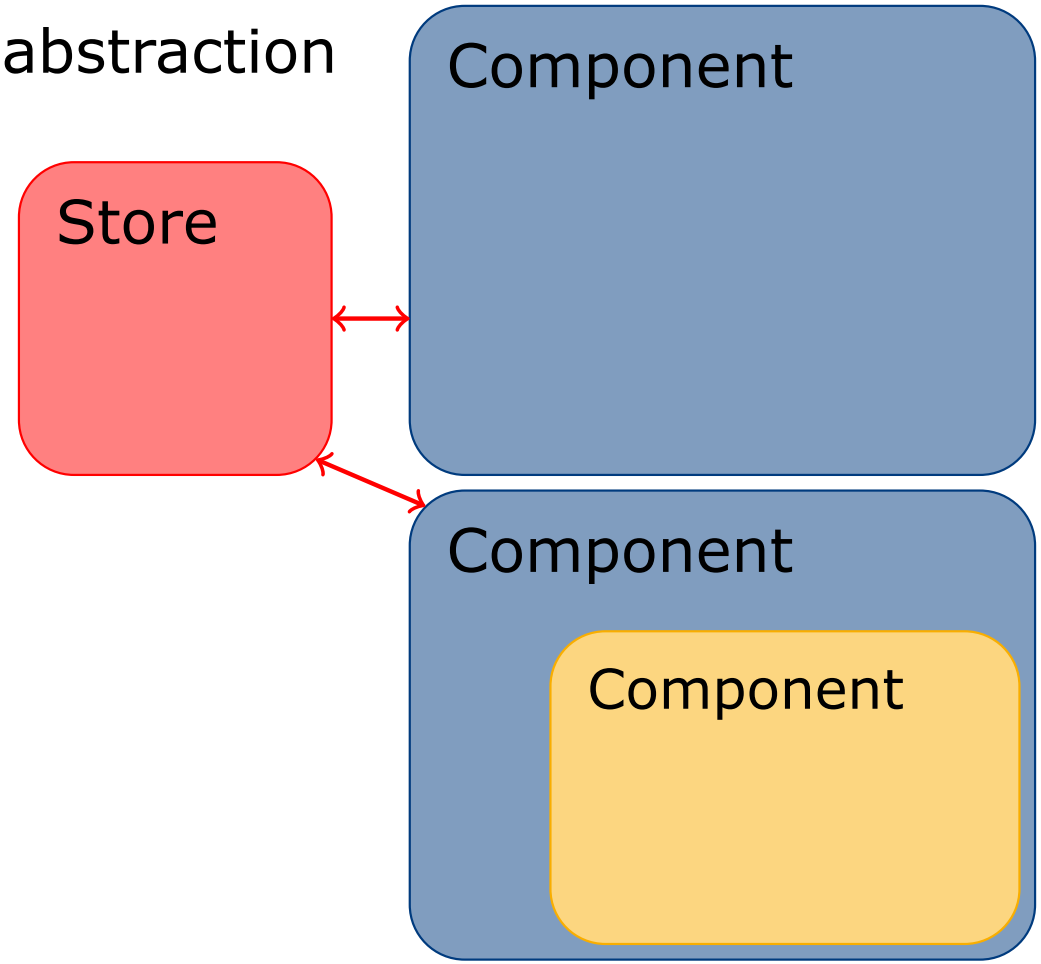
Why Reflux?

declarative view abstraction

decoupling

easy testing

fast rendering



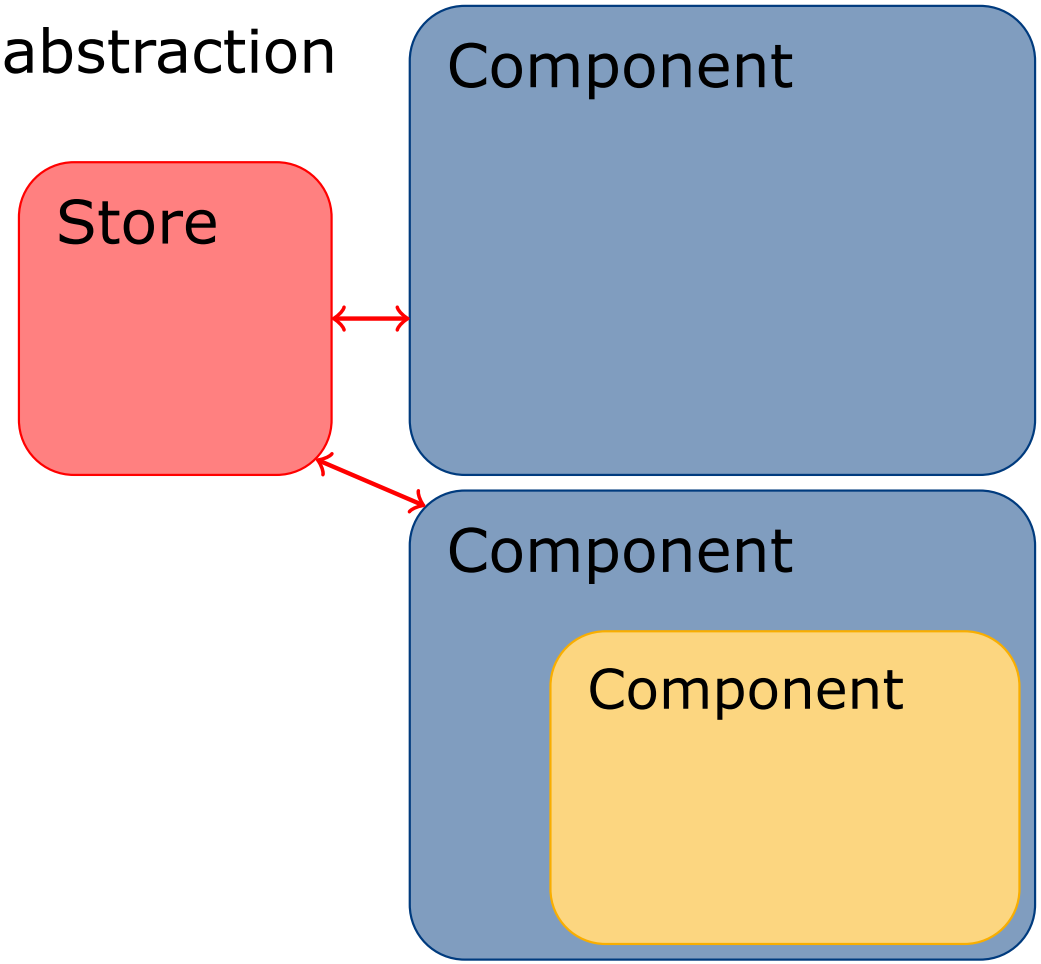
Why Reflux?

declarative view abstraction

decoupling

easy testing

fast rendering



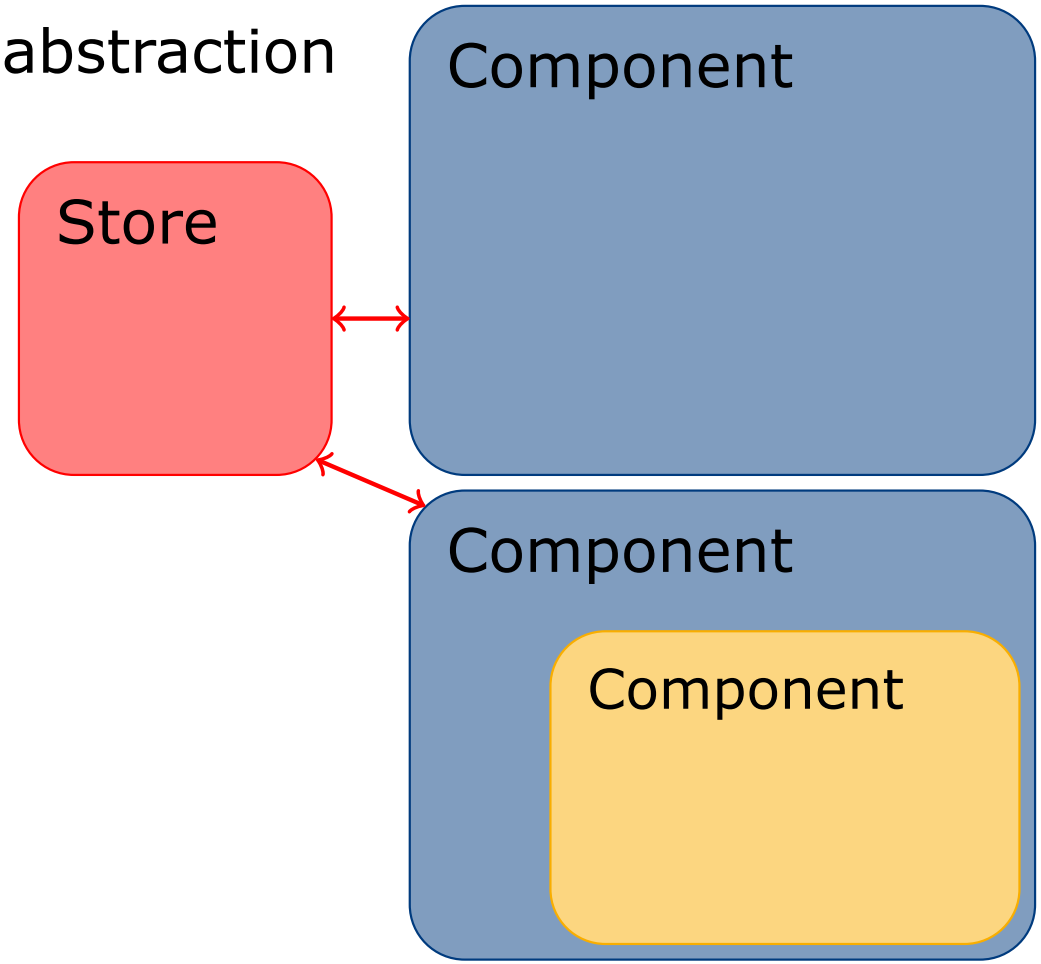
Why Reflux?

declarative view abstraction

decoupling

easy testing

fast rendering



Why Reflux?

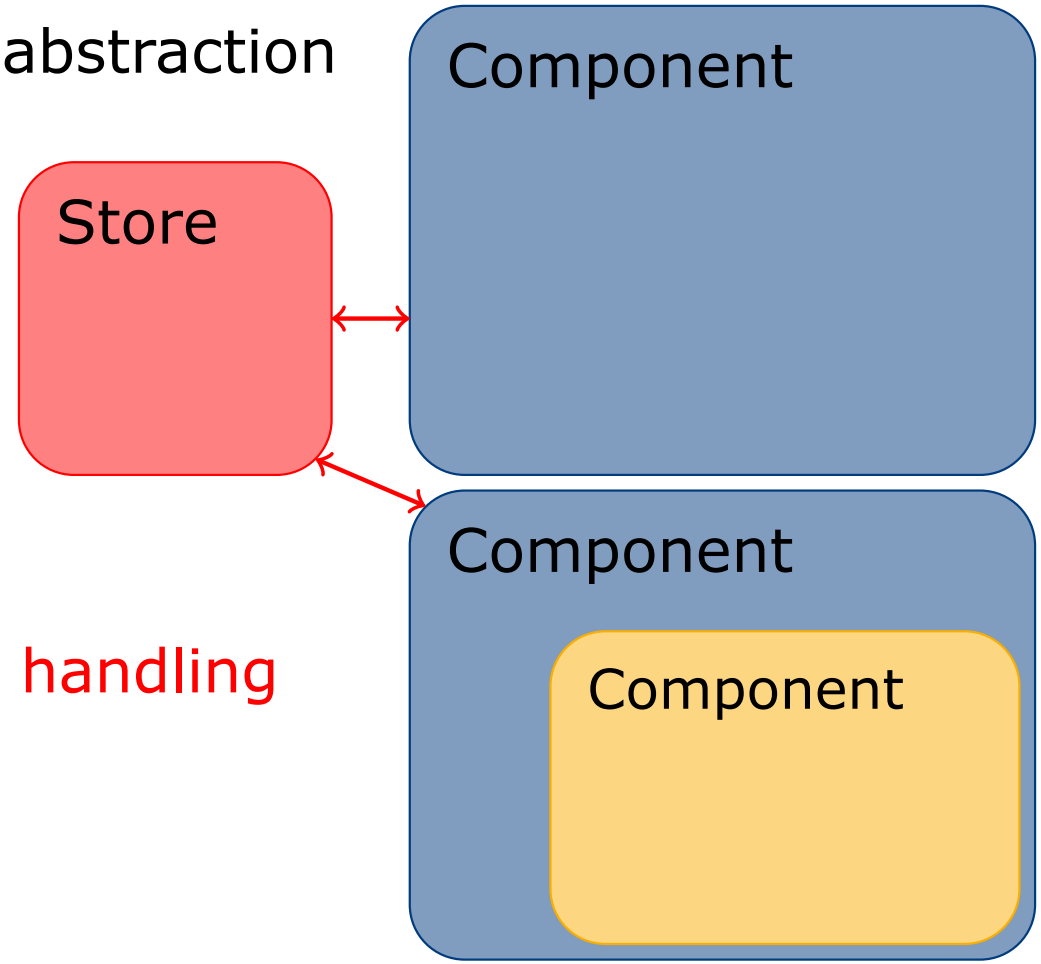
declarative view abstraction

decoupling

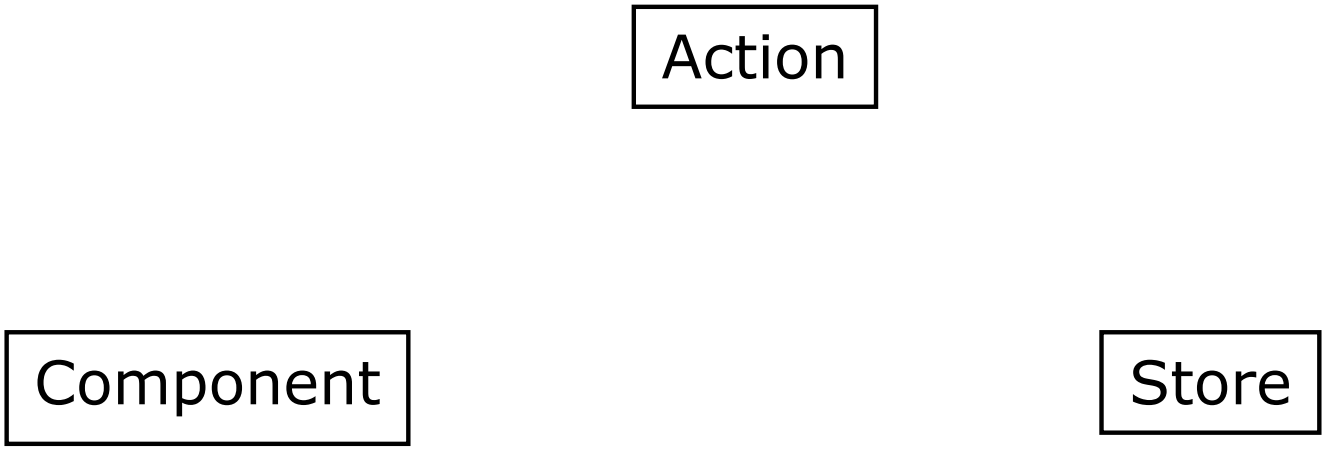
easy testing

fast rendering

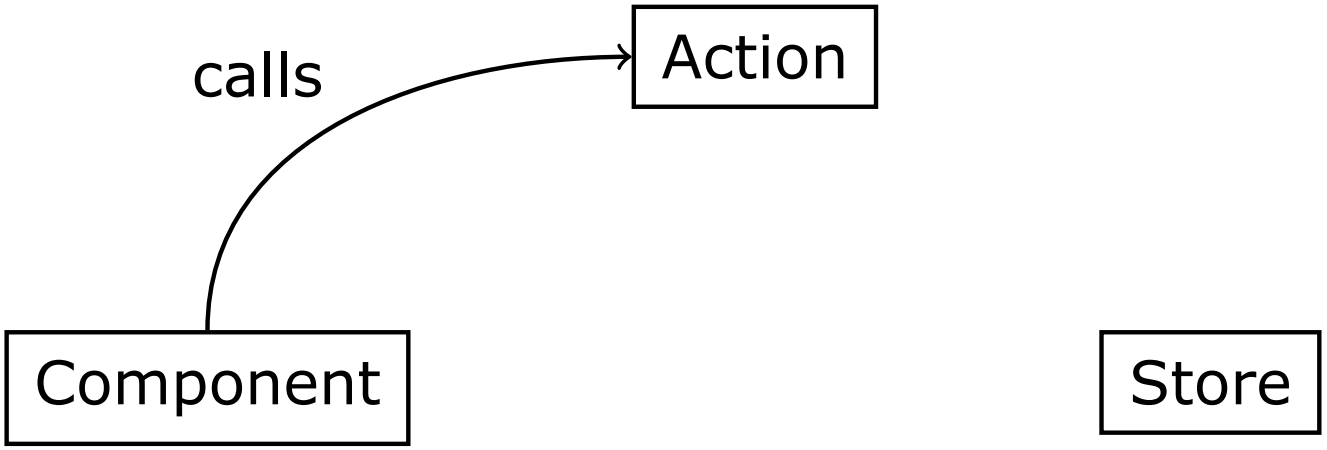
transversal state handling



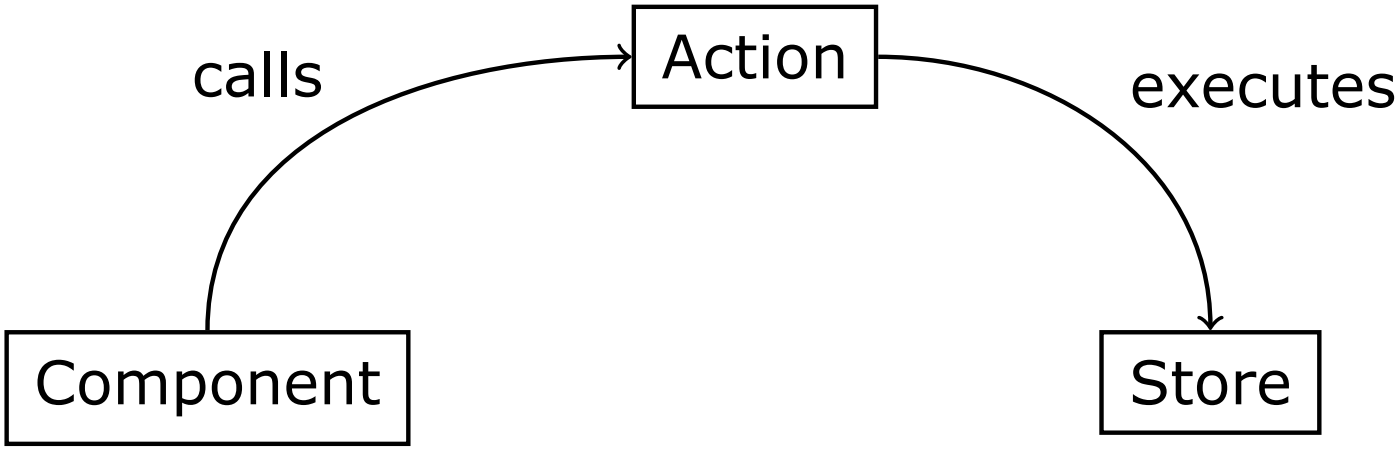
Data Flow



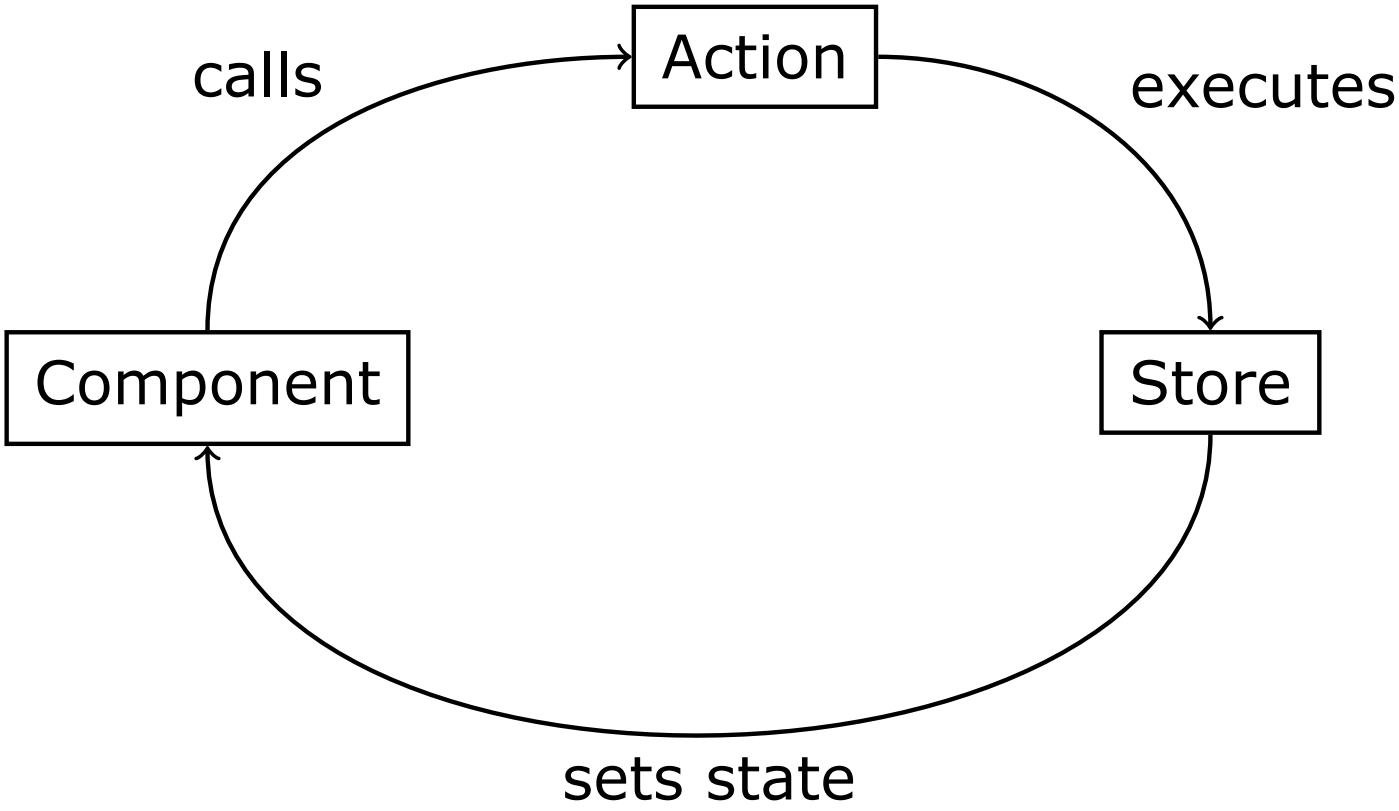
Data Flow



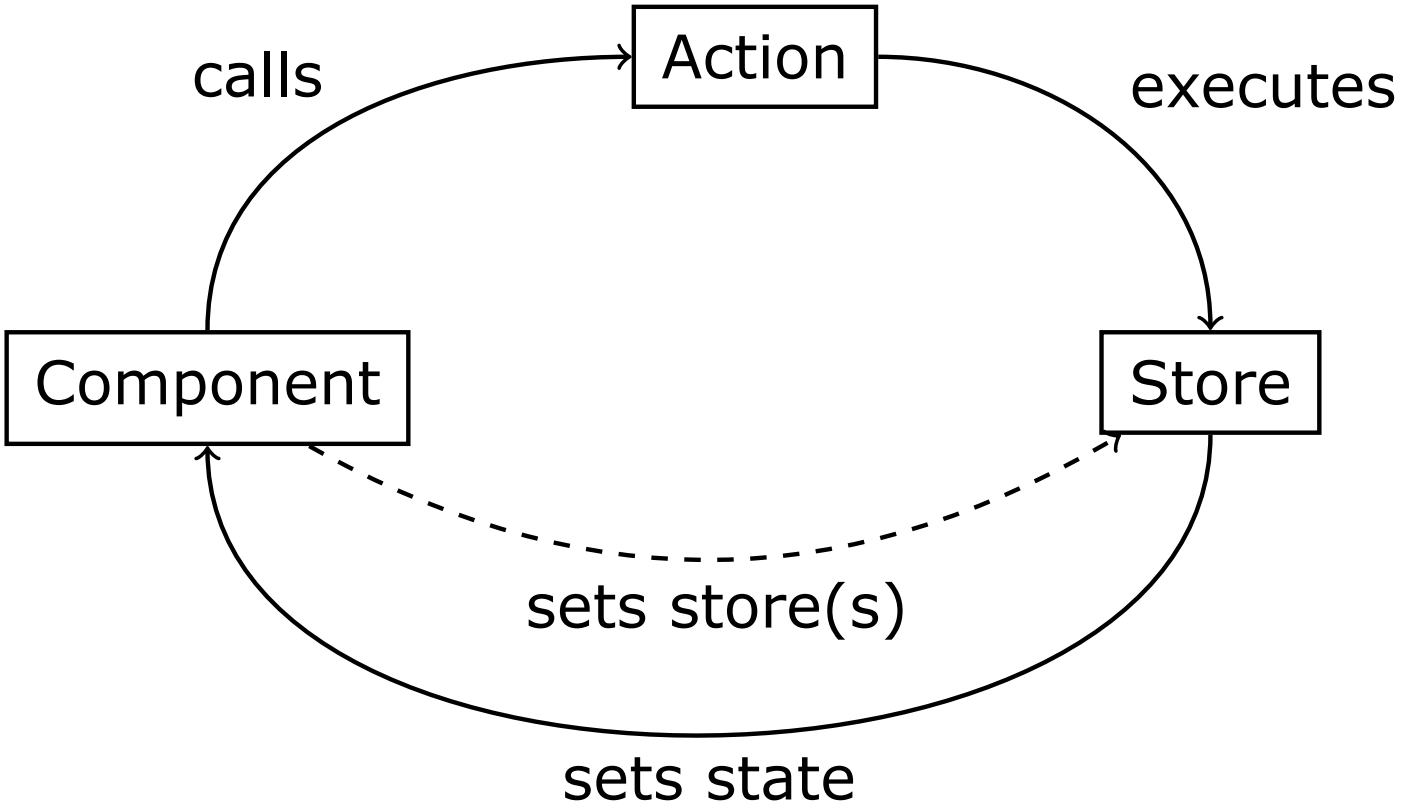
Data Flow



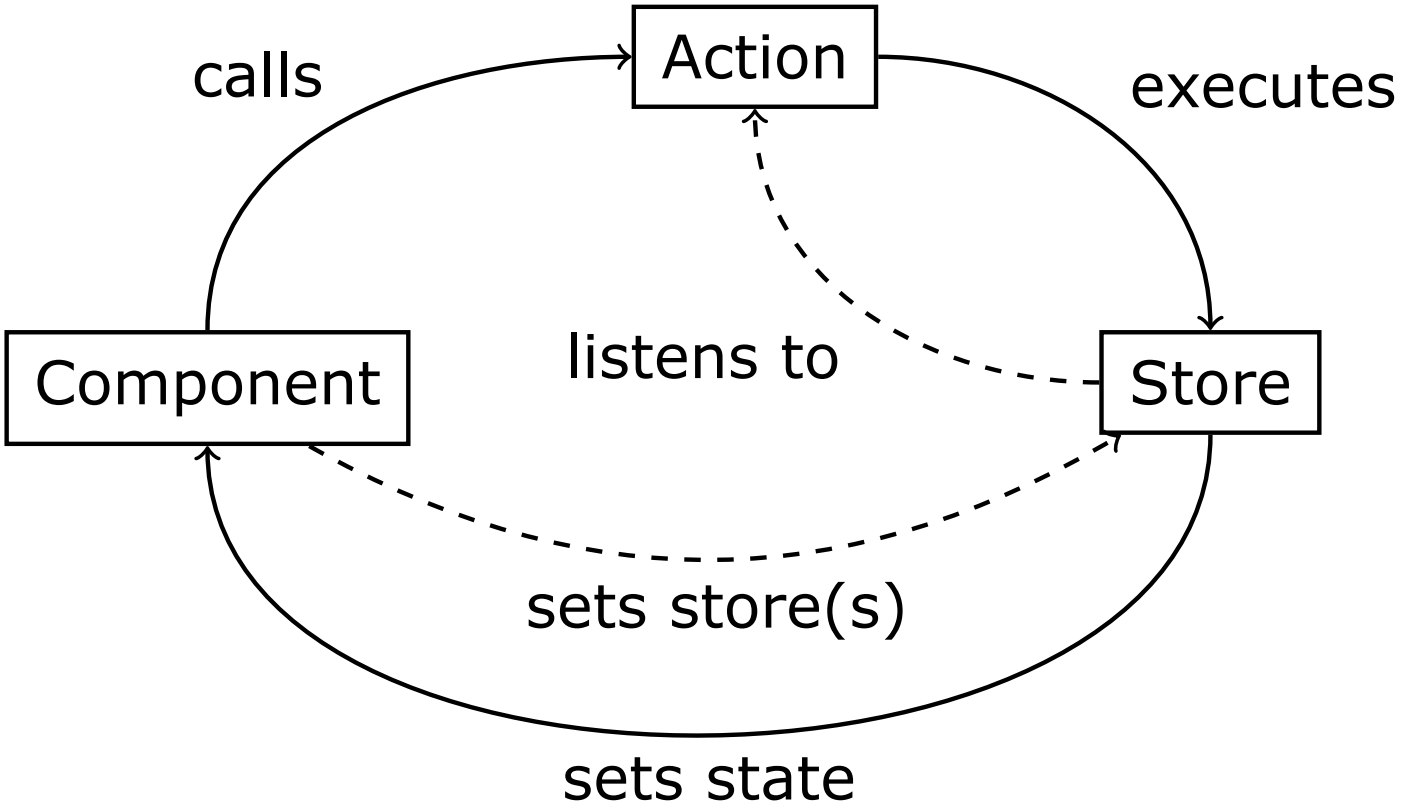
Data Flow



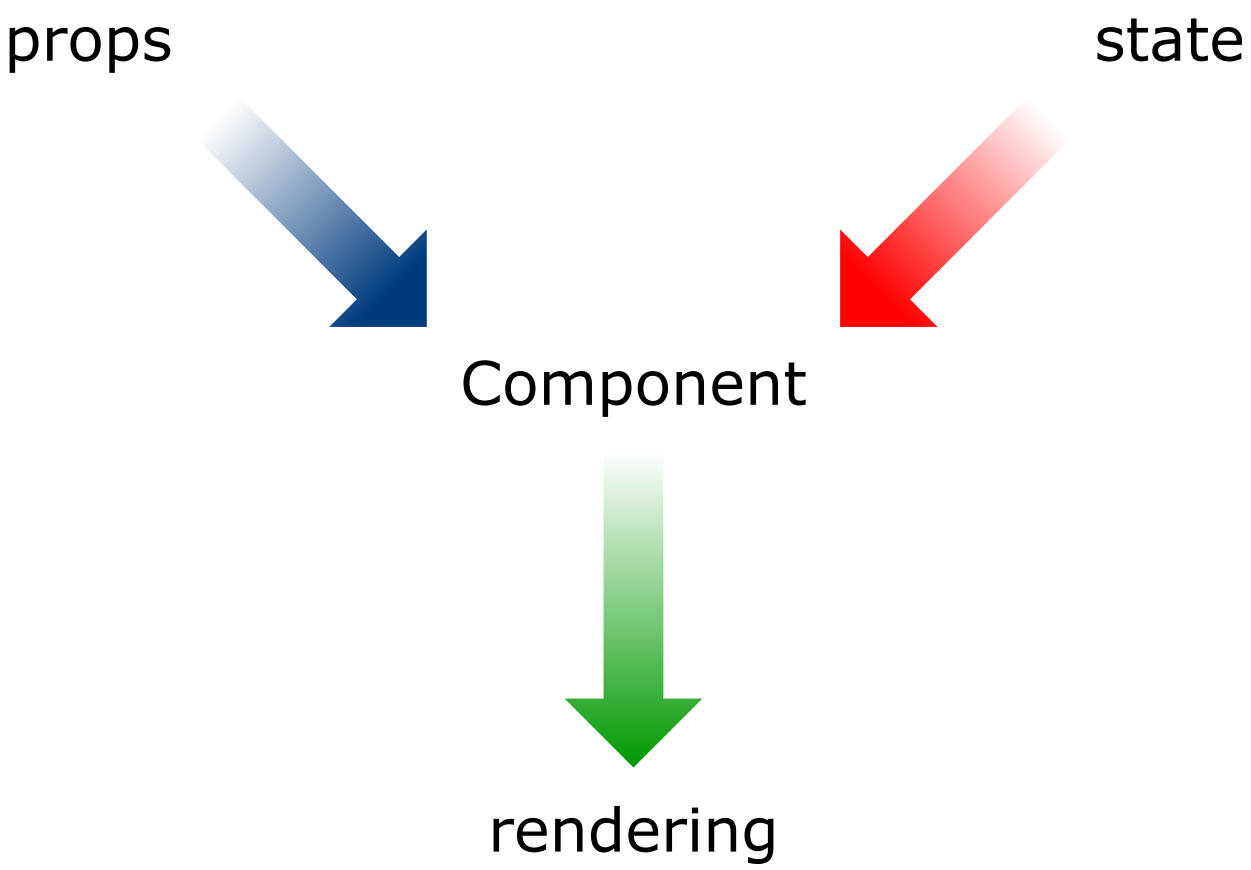
Data Flow


Action
calls
executes
Component
Store
sets store(s)
sets state

Data Flow

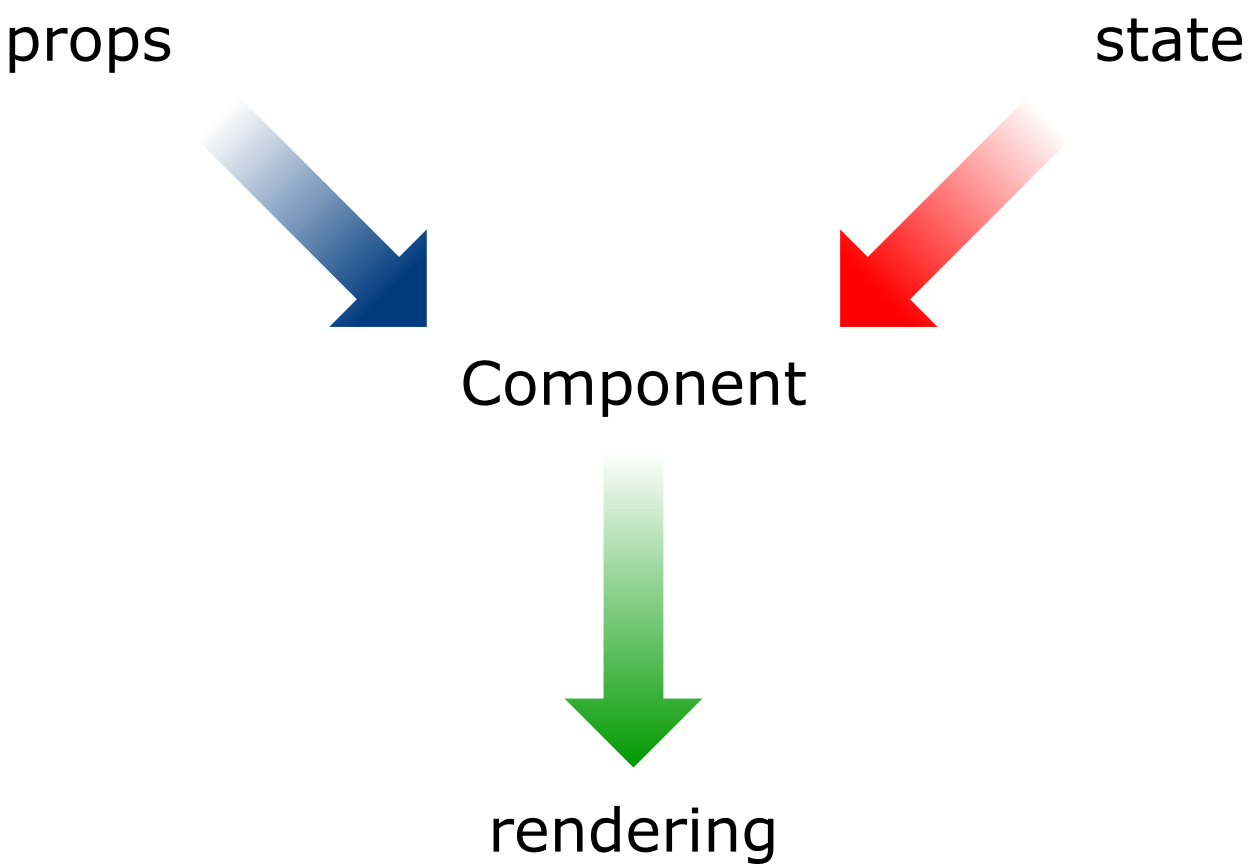

Action
calls
executes
Component
listens to
Store
sets store(s)
sets state

Reflux Components

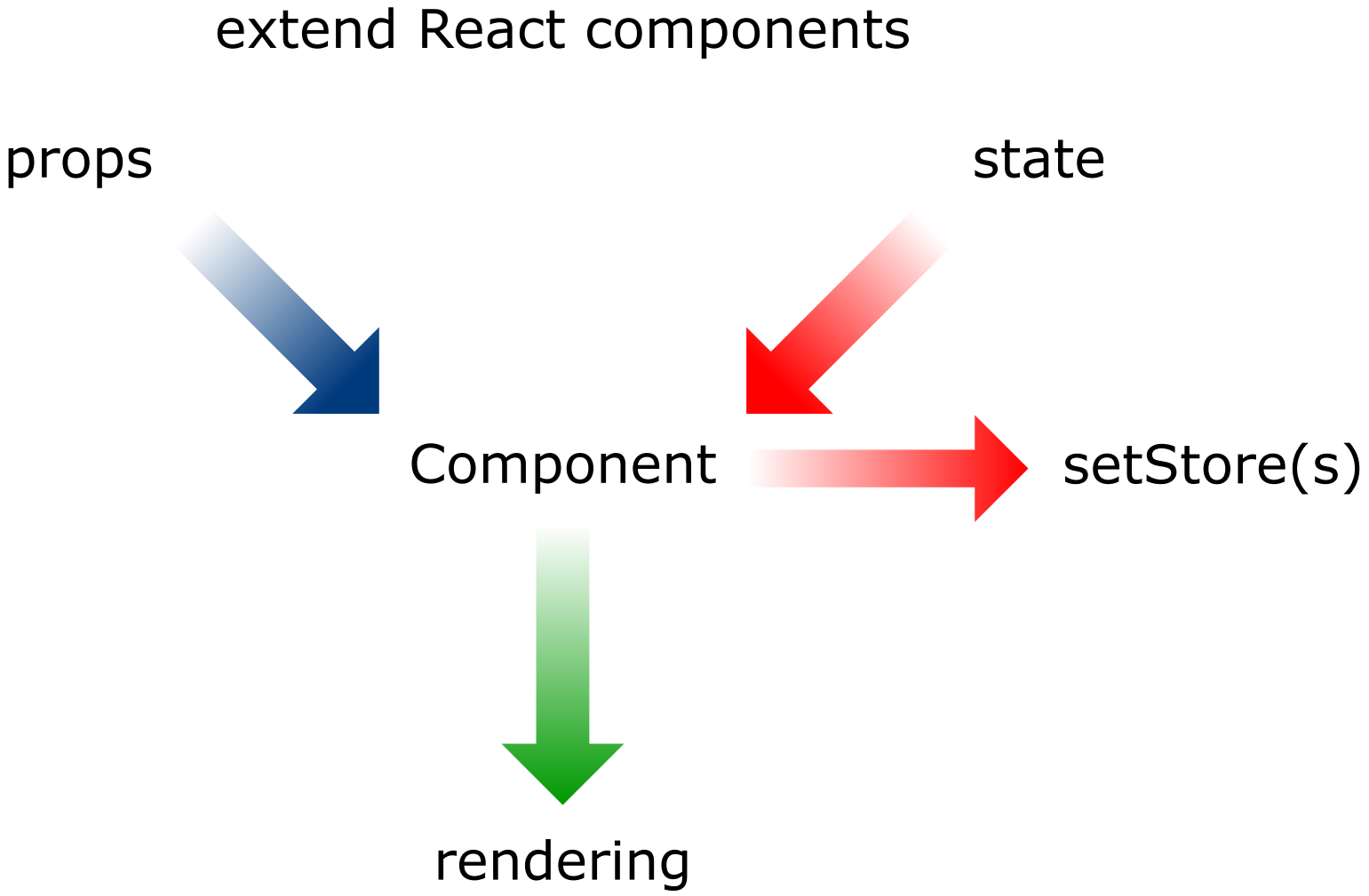


Reflux Components

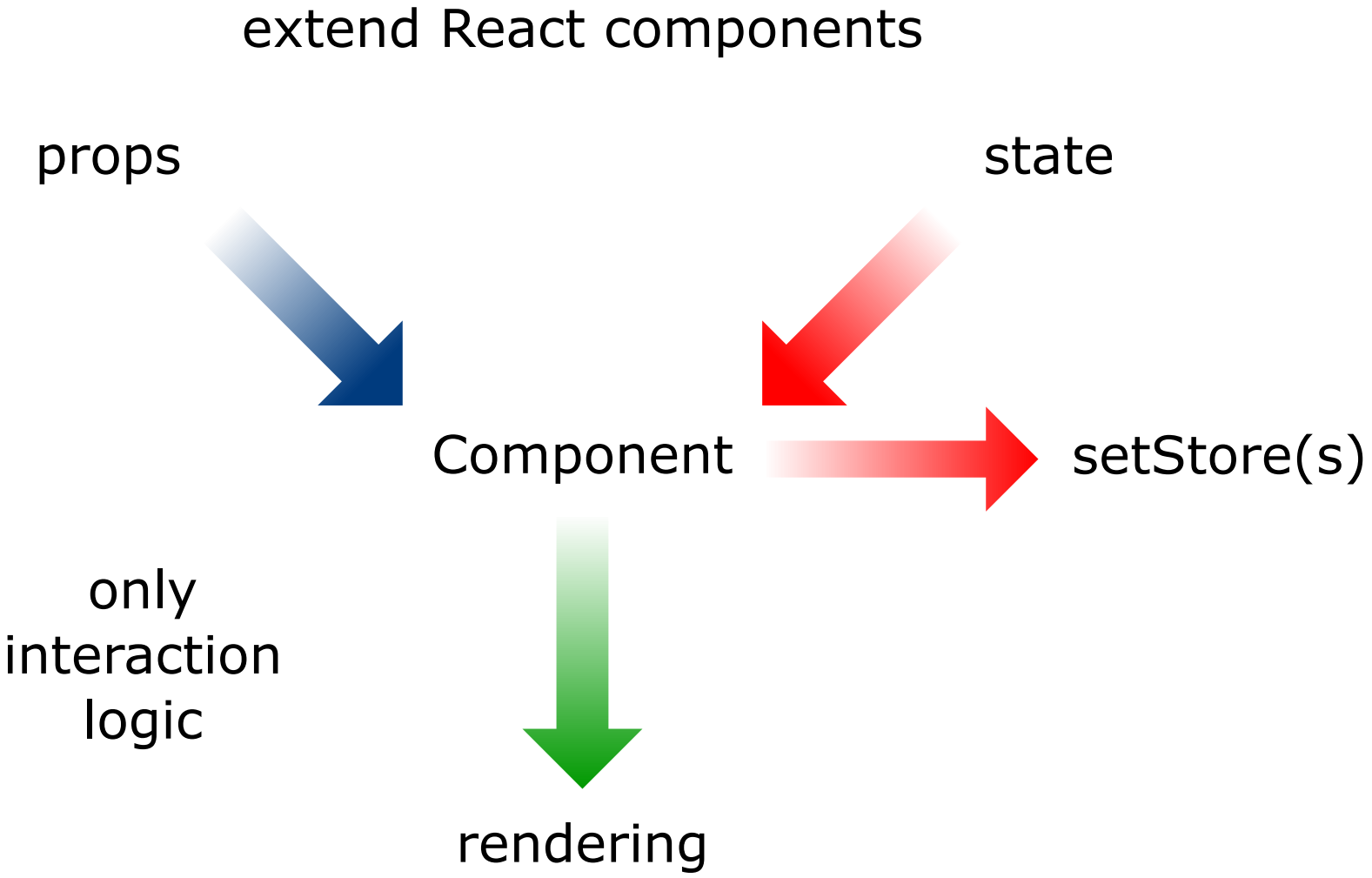
extend React components



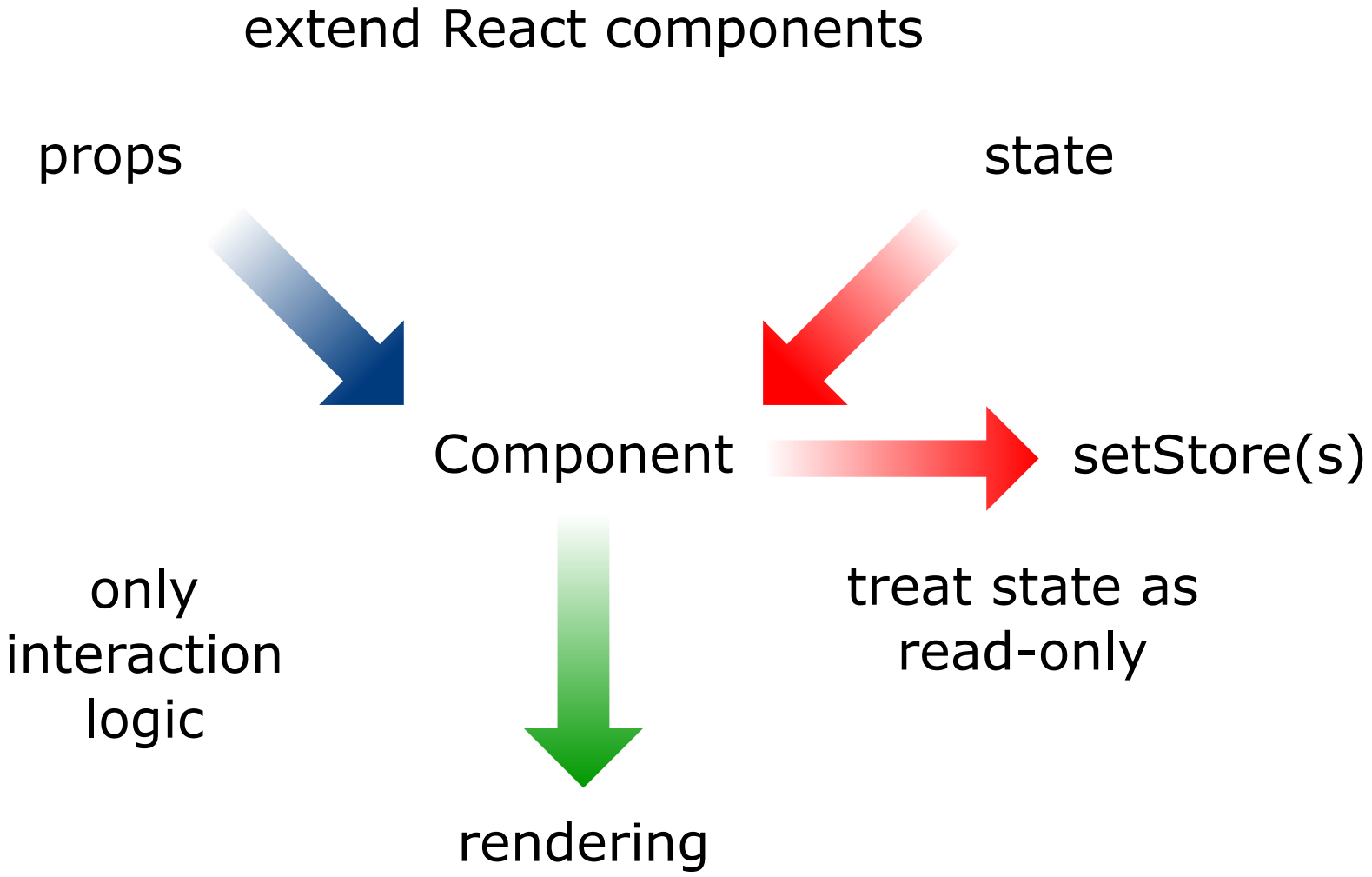
Reflux Components



Reflux Components



Reflux Components



Actions

Action

METRO

Actions

just a function

Action

METRO

Actions

just a function

Action

called by
component

Actions

just a function

Action

called by component

transports payload

Stores

Store

Stores

state handler

Store

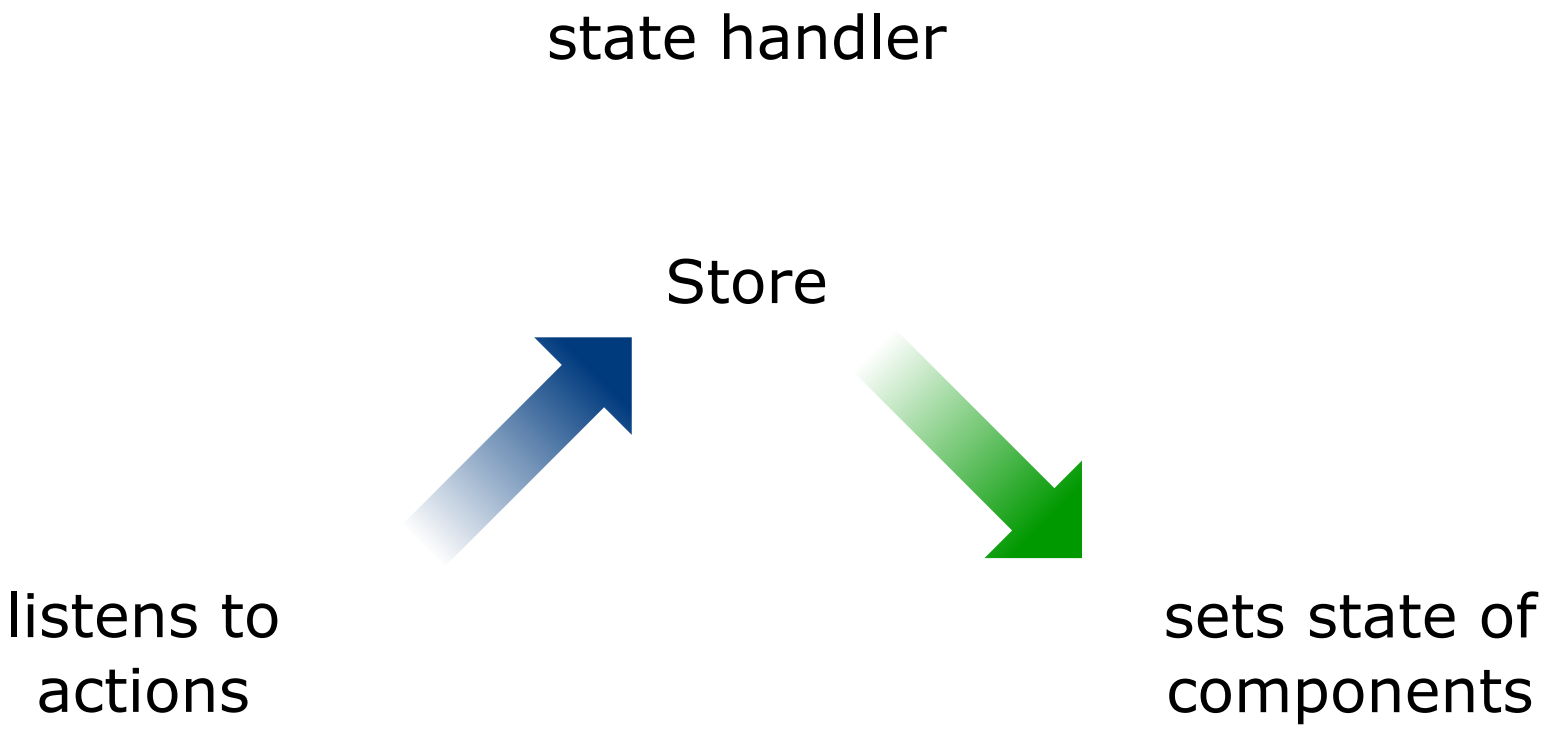
Stores

state handler

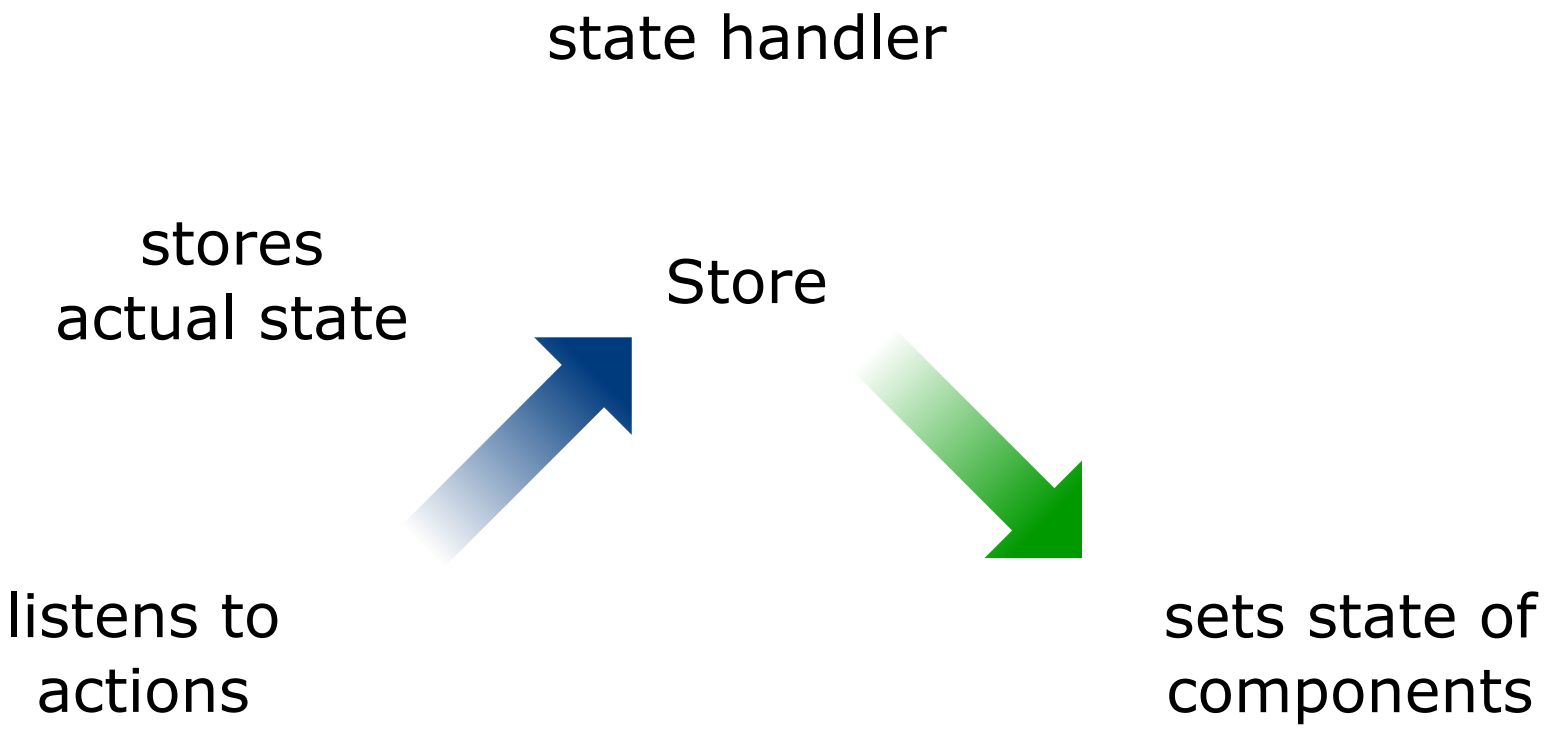
Store

listens to
actions

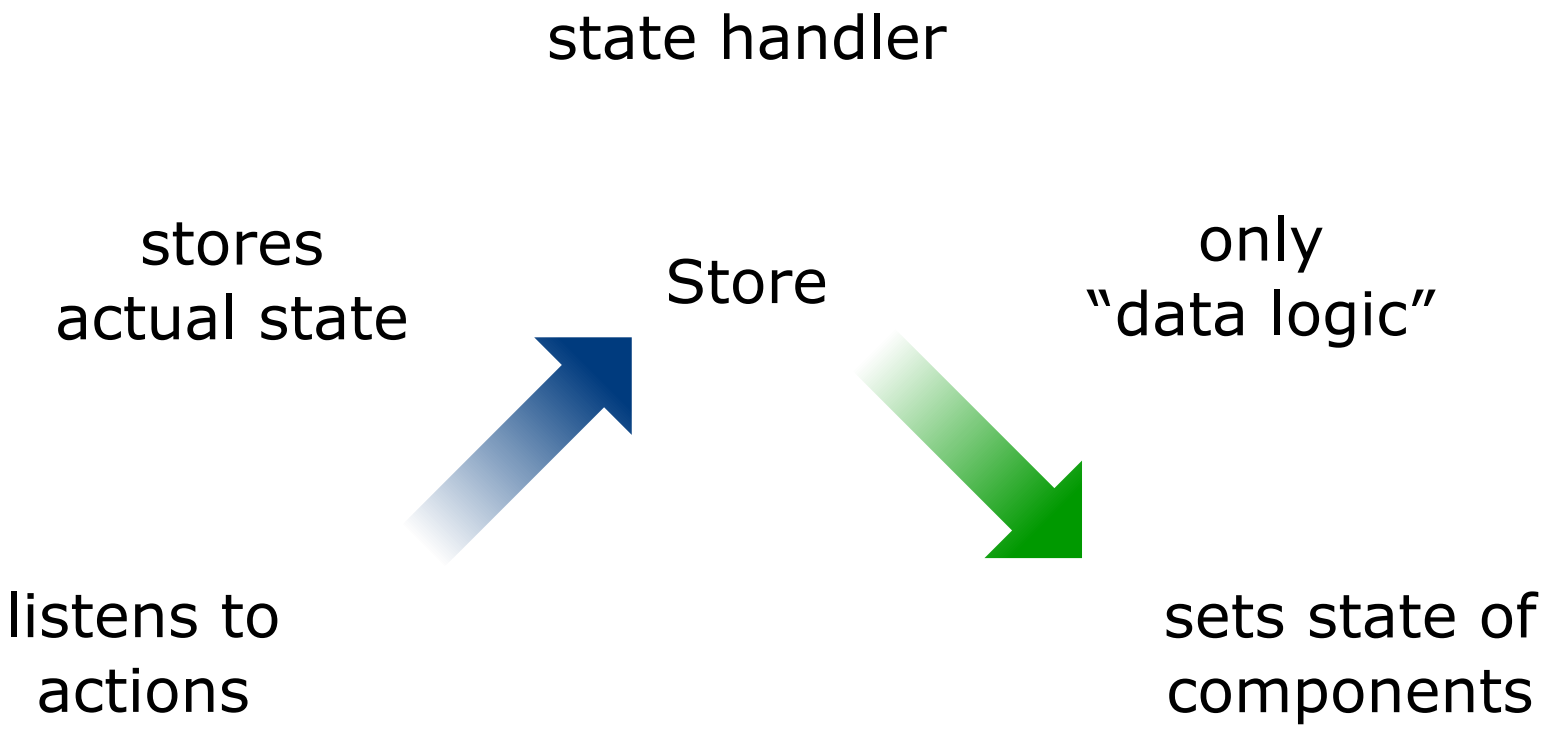
Stores


state handler
Store
listens to actions
sets state of components

Stores


state handler
stores
actual state
Store
listens to
actions
sets state of
components

Stores


state handler
stores
actual state
Store
only
“data logic”
listens to
actions
sets state of
components

Reflux Components, Actions, and Stores – Example

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


METRO

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      }
```

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

Problematic state handling

- Getter in stores

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 }
```

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  }
```

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

How to handle state

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

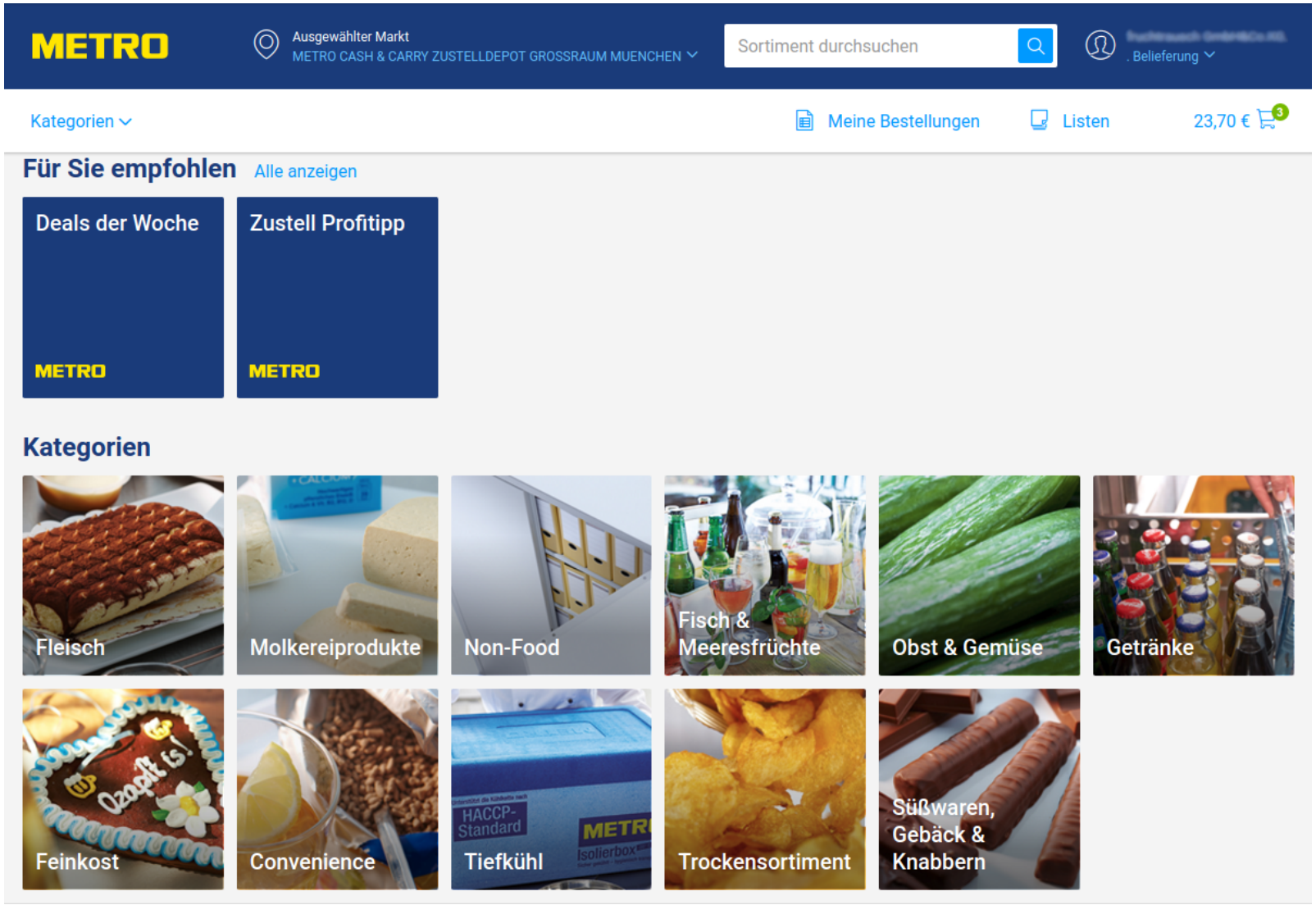
METRO

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

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

METRO

Tech Stack @METRO

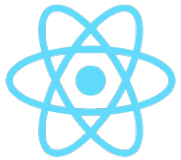
`http://domain/service/endpoint`

REST API

Docker

Tech Stack @METRO

`http://domain/service/endpoint`



React

REST API

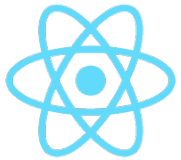
Docker

kubernetes

METRO

Tech Stack @METRO

`http://domain/service/endpoint`



React

Java 8

REST API

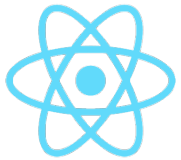
Docker

kubernetes

METRO

Tech Stack @METRO

`http://domain/service/endpoint`



React

Java 8

REST API

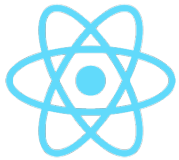
Docker

kubernetes

METRO

Tech Stack @METRO

`http://domain/service/endpoint`



React

Java 8

REST API

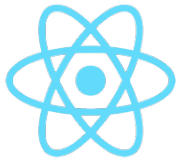
Docker

kubernetes

METRO

Tech Stack @METRO

`http://domain/service/endpoint`



React

Java 8

REST API

Docker

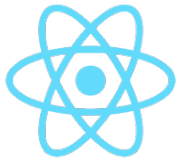
Clojure

kubernetes

METRO

Tech Stack @METRO

`http://domain/service/endpoint`



React

Java 8

REST API

Docker

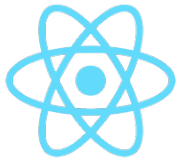
Clojure

kubernetes

METRO

Tech Stack @METRO

`http://domain/service/endpoint`



React

Java 8

REST API

Docker

Clojure

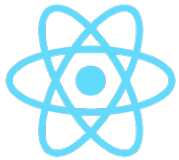
kubernetes

LaTeX

METRO

Tech Stack @METRO

http://domain/service/endpoint



React

Java 8

REST API

Docker

Clojure

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

kubernetes

LaTeX

Thank you!

METRO

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

METRO metro-cc.com/transformers