REACTJS UPDATE

STEPHEN WHITE

CADEC 2019.01.24 & 2019.01.30 | CALLISTAENTERPRISE.SE

CALLISTA

— ENTERPRISE —

REACTJS UPDATE - AGENDA

- *The Three Distractions of React*

- Redux to the rescue?
- Hooks
- Hooks - Refactor
- Summary

HOW DID I GET HERE

2015

2016

2017

2018

REACTEUROPE

Redux

styled components

SUSPENSE

HOOKS

FIBER

THE THREE MAIN DISTRACTIONS OF REACT

ABSTRACTIONS

SIDE EFFECTS

STATE - DATA FLOW

ABSTRACTIONS

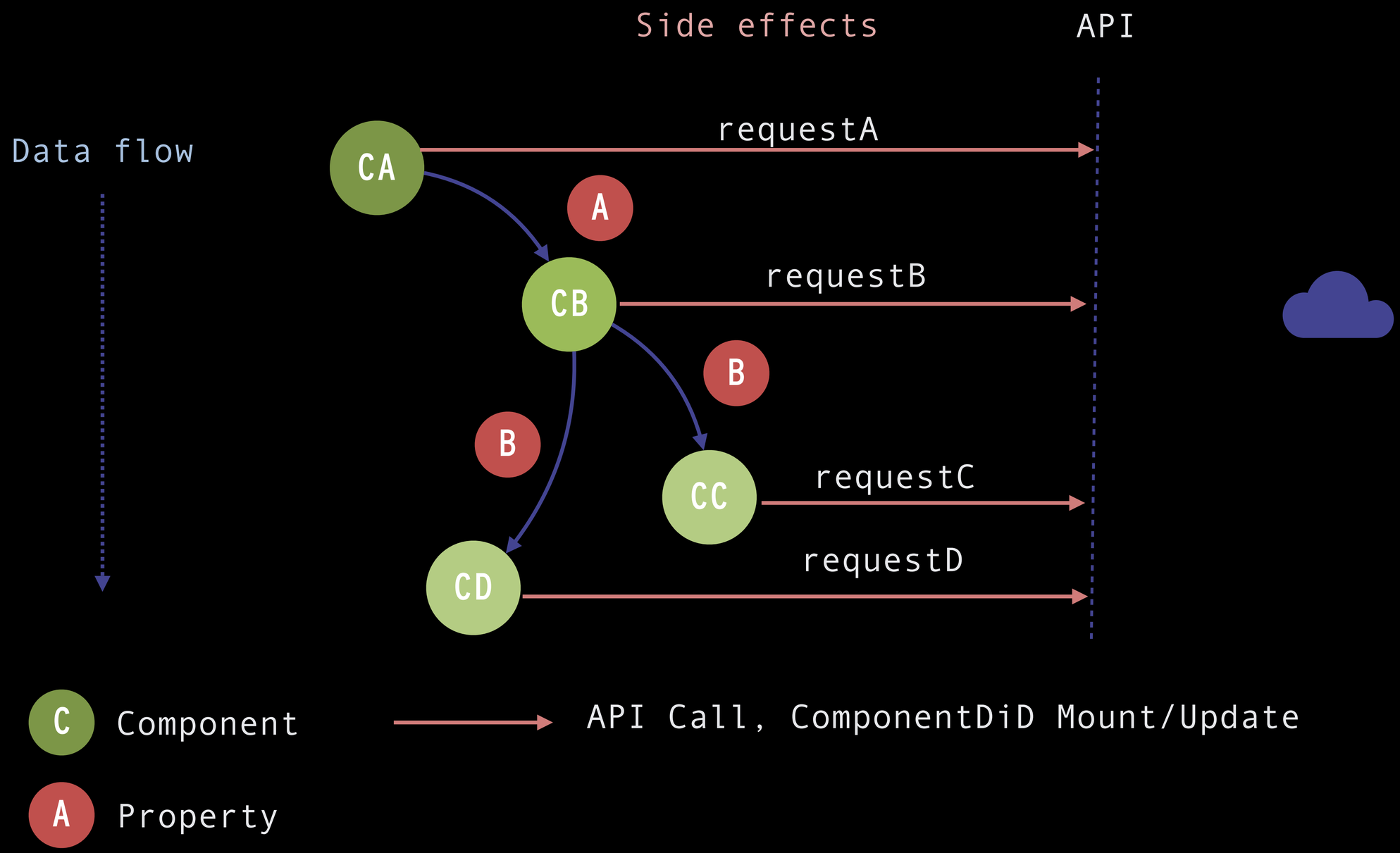
COMPONENTS WITHIN COMPONENTS

Component Wrapper for State/Behaviour

Component (Stateless)˝ That is wrapped in State/ Behaviour or connected to the context/Redux store

HOCS OR RENDER PROPS

DATAFLOW - SIDE EFFECTS - PROPS DOWN



REACTJS UPDATE - AGENDA

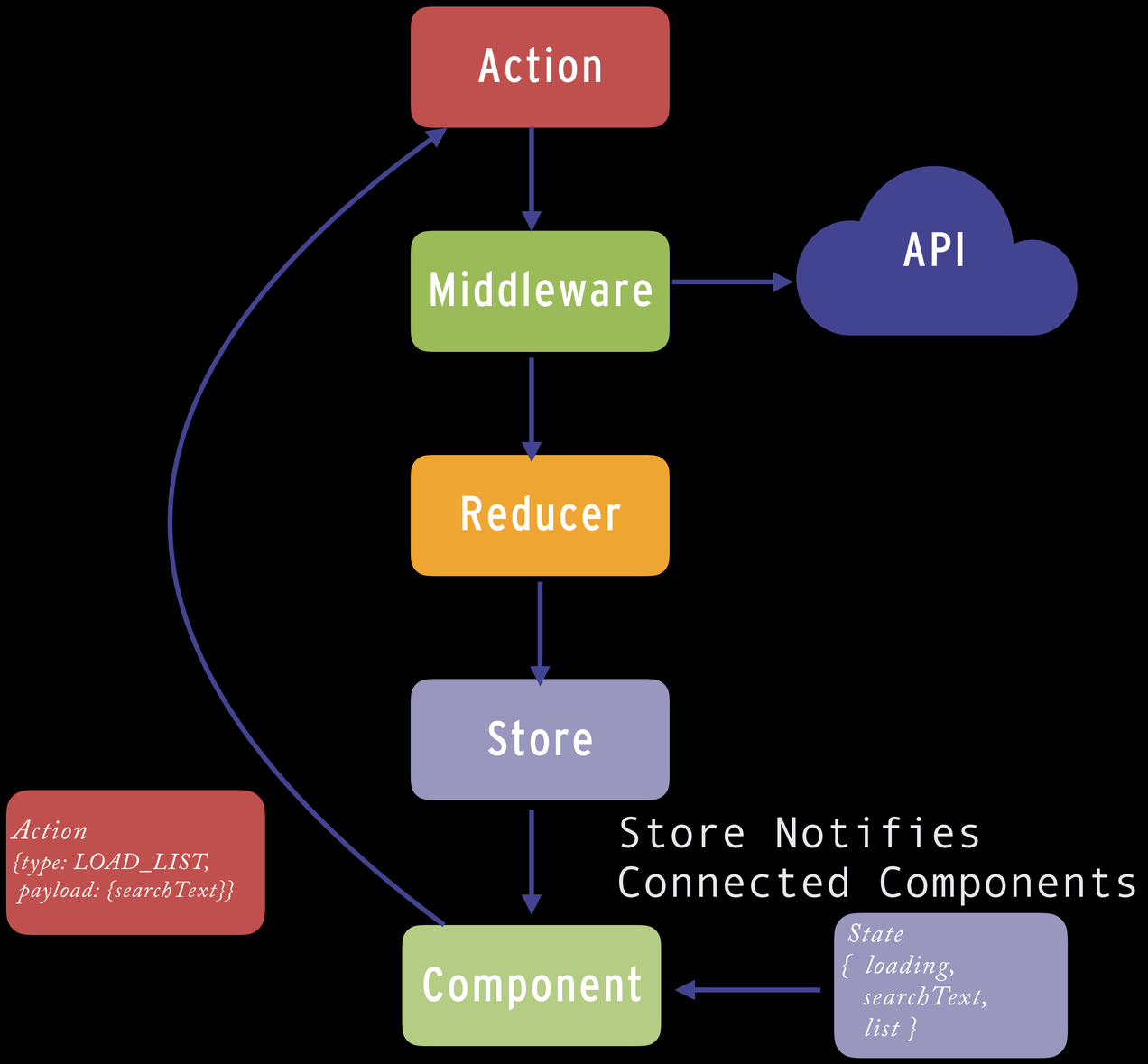
- The Three Distractions of React

- *Redux to the rescue?*

- Hooks
- Hooks - Refactor
- Summary

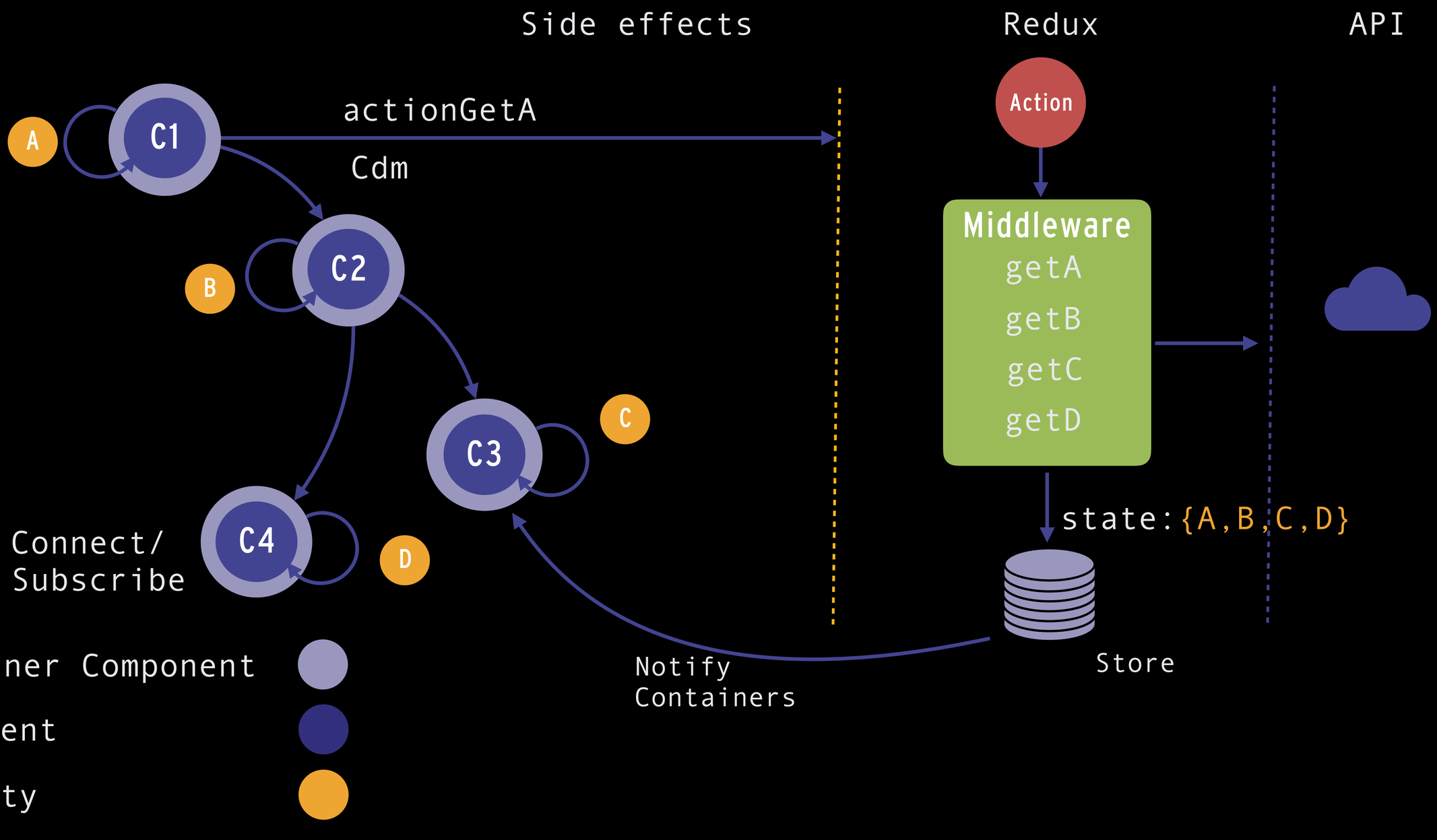
REACTS PROBLEM - SIDE EFFECTS - DATA FLOW - REDUX TO THE RESCUE!

- Actions - {type, payload}
- Middlewares
  - Thunks - Promises
  - Redux Observables - RXJS
  - Sagas - Generators
- Reducer - think list.reduce
- Store - Global State
- View - Subscribes to state changes

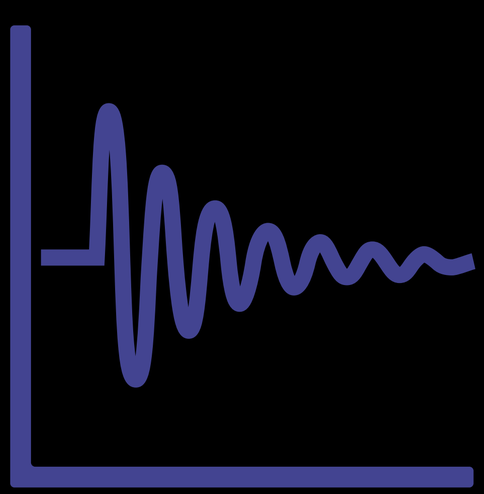


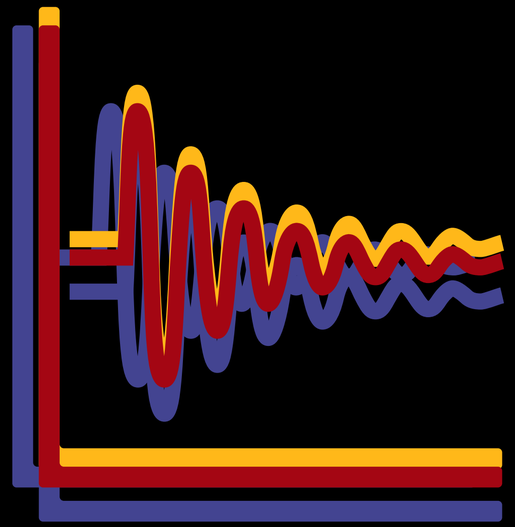

```
Action -> Middleware -> State mutation -> Notify Components
```

DATAFLOW - SIDE EFFECTS - PROPS POP FROM THE CONTEXT



REACTJS UPDATE - REDUX - SIGNAL TO NOISE





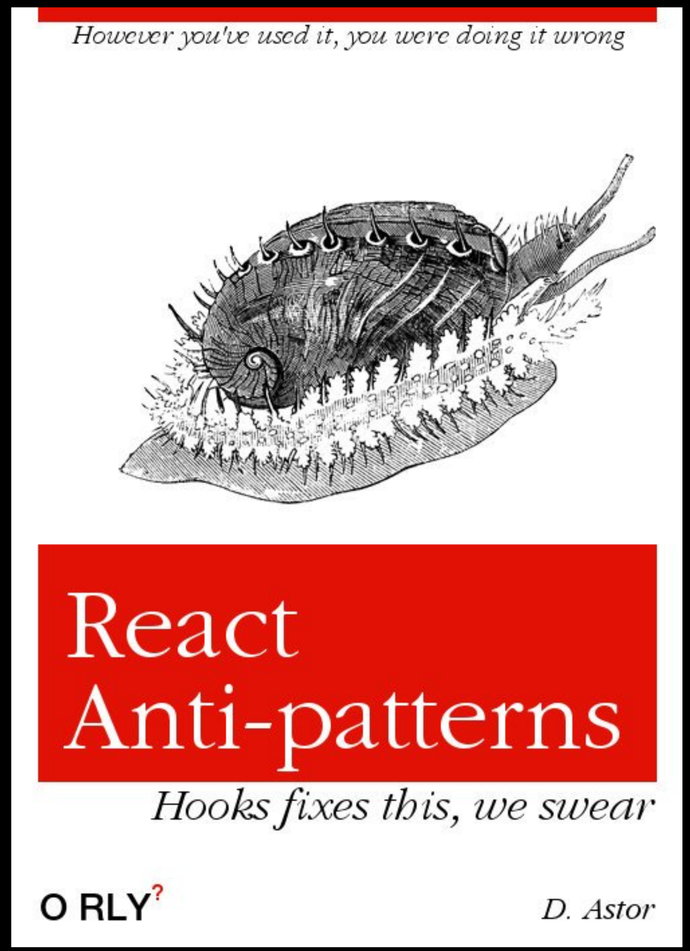
REACTJS UPDATE - AGENDA

- The Three Distractions of React
- Redux to the rescue?

- *Hooks*

- Hooks - Refactor
- Summary

REACTJS UPDATE - HOOKS TO THE RESCUE



REACTJS UPDATE - HOOKS TO THE RESCUE

UPDATE: IF YOU CAN USE REACT@16.8.0 THEN THE ANSWER IS TO NEVER USE RENDER PROPS AND ALWAYS USE A CUSTOM HOOK.

HOOKS ARE ALWAYS THE SUPERIOR PATTERN.

https://blog.kentcdodds.com/when-to-not-use-render-props-5397bbeff746

HOOKS - REACT 16.8.0-ALPHA.1

- useState
- useContext
- useEffect
- useRef
- useReducer
- useMemo Memoized functions
- *useObservable – (Custom hook)*

- Opt in, 100% backward combatible
- Don't replace classes
- Motivations
  - Hard to re-use state and behaviour
  - Can be tested independently
  - Complex components are hard to understand and test
  - Class optimisations

REACTJS UPDATE - AGENDA

- The Three Distractions of React
- Redux to the rescue?
- Hooks

- *Hooks - Refactor*

- Summary

REFACTOR - CLASS COMPONENT

```
class SearchComponentClass extends React.Component {
  // --- state
  constructor(props) {
    super(props);
    this.state = {
      searchText: "",
      list: [],
      loading: !!props.loading,
      error: undefined
    };
  }
  state = {searchText:"", loading: false}

  // --- behaviour
  handleUpdateSearchText = event => {
    this.setState({ searchText: event.target.value });
  };
  handleResetSearchText = () => {
    this.setState({ searchText: "" });
  };

  // --- lifecycle, side effects
  componentDidMount() {
    this.setState({ loading: true });
    api
      .search(this.state.searchText)
      .then(list => this.setState({ list, loading: false }));
  }
  componentDidUpdate(prevProps, prevState) {
    if (prevState.searchText !== this.state.searchText) {
      this.props.loading !== undefined && this.setState({ loading: true });
      api
        .search(this.state.searchText)
        .then(list => this.setState({ list, loading: false }));
    }
  }

  render() {
    const { searchText, list, loading } = this.state;
    return (
      <SearchComponent
        title={"Phil's-osophies Component"}
        loading={loading}
        searchText={searchText}
        list={list}
        handleUpdateSearchText={this.handleUpdateSearchText}
        handleResetSearchText={this.handleResetSearchText}
      />
    );
  }
}
```

REFACTOR - RENDER -> FUNCTIONAL COMPONENT

```
const SearchContainerHooks = () => {
  return (
    <SearchComponent
      title={"Search Class"}
      loading={false}
      searchText={""}
      list={[]}
      handleUpdateSearchText={() => {}}
      handleResetSearchText={() => {}}
    />
  );
};
```

REFACTOR - STATE -> USE STATE

```
const SearchContainerHooks = props => {
  const [list, setList] = useState([]);
  const [searchText, setSearchText] = useState("");
  const [loading, setLoading] = useState(false);

  return (
    <SearchComponent
      title={"Use state"}
      loading={loading}
      searchText={searchText}
      list={list}
      handleUpdateSearchText={({ target: { value } }) => setSearchText(value)}
      handleResetSearchText={() => setSearchText("")}
    />
  );
};
```

REFACTOR - STATE -> HANDLERS

```
const mapStateToHandlers = (searchState, setSearchState) => ({
  handleUpdateSearchText: ({ target: { value: searchText } }) =>
    searchText !== searchState.searchText && setSearchState({ searchText }),
  handleResetSearchText: () => setSearchState({ searchText: "" })
});

const SearchContainerHooks = props => {

  const [searchState, setSearchState] = useReducer(
    (state, newState) => ({ ...state, ...newState }),
    initialState
  );

  const handlers = mapStateToHandlers(searchState, setSearchState);

  return (
    <SearchComponent
      title={"Search useReducer as State"}
      loading={searchState.loading}
      searchText={searchState.searchText}
      list={searchState.list}
      handleUpdateSearchText={handlers.handleUpdateSearchText}
      handleResetSearchText={handlers.handleResetSearchText}
    />
  );
};
```

Handlers wrap the state updater with semantically named handlers

REFACTOR - LIFECYCLE -> USE EFFECT

```
const SearchContainerHooks = props => {
  useEffect(
    () => {
      handlers.handleRequestSearchList(searchState.searchText);
      api
        .search(searchState.searchText)
        .then(list => handlers.handleSuccessSearchList(list))
        .catch(err => handlers.handleFailureSearchList(err));
    },
    [searchState.searchText]
  );
  return (
    <SearchComponent
      title={"Search useReducer as State"}
      loading={searchState.loading}
      searchText={searchState.searchText}
      list={searchState.list}
      handleUpdateSearchText={handlers.handleUpdateSearchText}
      handleResetSearchText={handlers.handleResetSearchText}
    />
  );
};
```

Effect is called:
after mounting
Then on changes of searchText

REFACTOR - USE CUSTOM HOOKS

```
export const useSearchRequest = (state, requestHandlers) =>
  useEffect(
    () => {
      requestHandlers.handleRequestSearchList(state.searchText);
      api
        .search(state.searchText)
        .then(list => requestHandlers.handleSuccessSearchList(list))
        .catch(err => requestHandlers.handleFailureSearchList(err));
    },
    [state.searchText]
  );

export const userSearchContext = () => useContext(RootStateContext);

export const useSearchReducer = props => {
  const [state, dispatch] = useReducer(searchReducer, initialState(props));
  return { state, actions: mapDispatchToHandlers(dispatch) };
};
```

REFACTOR - USE STATE -> USE REDUCER

```
export const useSearchReducer = props => {
  const [state, dispatch] = useReducer(searchReducer, initialState(props));
  return { state, actions: mapDispatchToHandlers(dispatch) };
};
```

```
export const searchReducer = (state, action) => {
  switch (action.type) {
    case ESearchActions.UPDATE_SEARCH_TEXT:
      return { ...state, ...action.payload };
    case ESearchActions.REQUEST_SEARCH_LIST:
      return { ...state, ...action.payload, loading: true };
    case ESearchActions.SUCCESS_SEARCH_LIST:
      return { ...state, ...action.payload, loading: false };
    case ESearchActions.FAILURE_SEARCH_LIST:
      return { ...state, ...action.payload, loading: false };
    default:
      return state;
  }
};
```

REFACTOR - USE CONTEXT

```
const App = props => (
  <RootStateProvider>
    <div className="App">
      <SearchContainerUseContextSuspense />
    </div>
  </RootStateProvider>
);
```

RootStateProvider – provides state and actions in the global context

```
export const userSearchContext = () => useContext(RootStateContext);

const SearchBarContainer = () => {
  const { actions, state } = userSearchContext();
  return (
    <SearchBar
      searchText={state.searchText}
      handleUpdateSearchText={actions.handleUpdateSearchText}
    >
      <LoadingContainer />
    </SearchBar>
  );
};
```

Any components that useContext will have access to its state and actions…

or anything in the context!

REACTJS UPDATE - AGENDA

- The Three Distractions of React
- Redux to the rescue?
- Hooks
- Hooks - Refactor

- *Summary*

SUMMARY - MOTIVATIONS

- Issues with the Component
  - OOP versus FP
  - *this*
  - Abstractions - too much time dithering over an FP or OOP approach.
  - Components have a tendency to grow and become unwieldily…
  - Hard to optimise when compiling (apparently ..)
  - Need for 3rd party libs to follow Functional Programming (recompose)
- Although
  - They've made React hugely successful so why change?

HOOKS - ADDRESSES

- Hooks
  - Functional Programming, first class citizen!
  - No more THIS, that, self or the other!
  - Abstractions - Just functional
    - » A common way for Side effects, dataflow, state, behaviour and abstractions
  - Encouraged to write small chunks of code and share!
  - Great optimisation for the compiler
  - No need for libraries like recompose
- Avoid
  - The three distractions and try hooks!

REACTJS UPDATE - STEPHEN WHITE

- IOT - Telenor Connexion
- React Native - Recorded Future
- stephen.white@callistaenterprise.se
- @maitriyogin
- https://github.com/callistaenterprise/hooks.git
- https://www.meetup.com/ReactJS-Goteborg/

HOOKS - REACT IS EASY

```
const Hooks = ({ functional }) => <h1>React gets {functional}</h1>;
```

Referential Transparancy

HOOKS - REACT IS HARD!

```
class SearchComponentClass extends React.Component {
  // --- state
  state = {searchText:"", loading: false, list:[]}

  // --- behaviour
  handleUpdateSearchText = event => {};

  // --- lifecycle, side effects
  componentDidMount() {}

  render() {
    const { searchText, list, loading } = this.state;
    return (
      <SearchComponent
        title={"Phil's-osophies Component"}
        loading={loading}
        searchText={searchText}
        list={list}
        handleUpdateSearchText={this.handleUpdateSearchText}
        handleResetSearchText={this.handleResetSearchText}
      />
    );
  }
}
```

- State
- Behaviour
- Lifecycle / side effects
- Render
  - Style
  - Structure
  - Etc ..

ASYNC - FALLBACK - AWAIT

REACTJS 16.7.0 ALPHA - SUSPENSE

- Allows you to defer rendering part of your application tree until some condition is met offering an intermediate state.
- React Suspense is a generic way for components to suspend rendering while they load data from a cache
  - Assets
  - Code
  - Async calls

SUSPENSIFY THE SEARCH COMPONENT

```
const SearchBar = React.lazy(() => import("./SearchBarContainer"));
const SearchList = React.lazy(() => import("./SearchListContainer"));
const SearchComponent = props => (
  <div className={"Search"} data-testid="search-container">
    <h2>{props.title}</h2>
    <Suspense
        fallback={<LoadingMessage message={"Search Bar"} />}
        >
      <SearchBar />
    </Suspense>
    <Suspense fallback={<LoadingMessage message={"Search List"} />}>
      <SearchList />
    </Suspense>
  </div>
);
```

Lazily load Components

Wrap them in React.Suspense

Provide a fallback component