

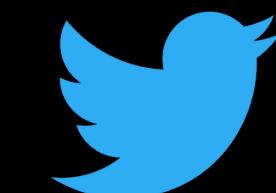


Understand Redux by Implementing It



John Crowson

Sr. Software Engineer, Capital One



@john_crowson

redux tutorial



All

Videos

Books

News

Images

More

Settings

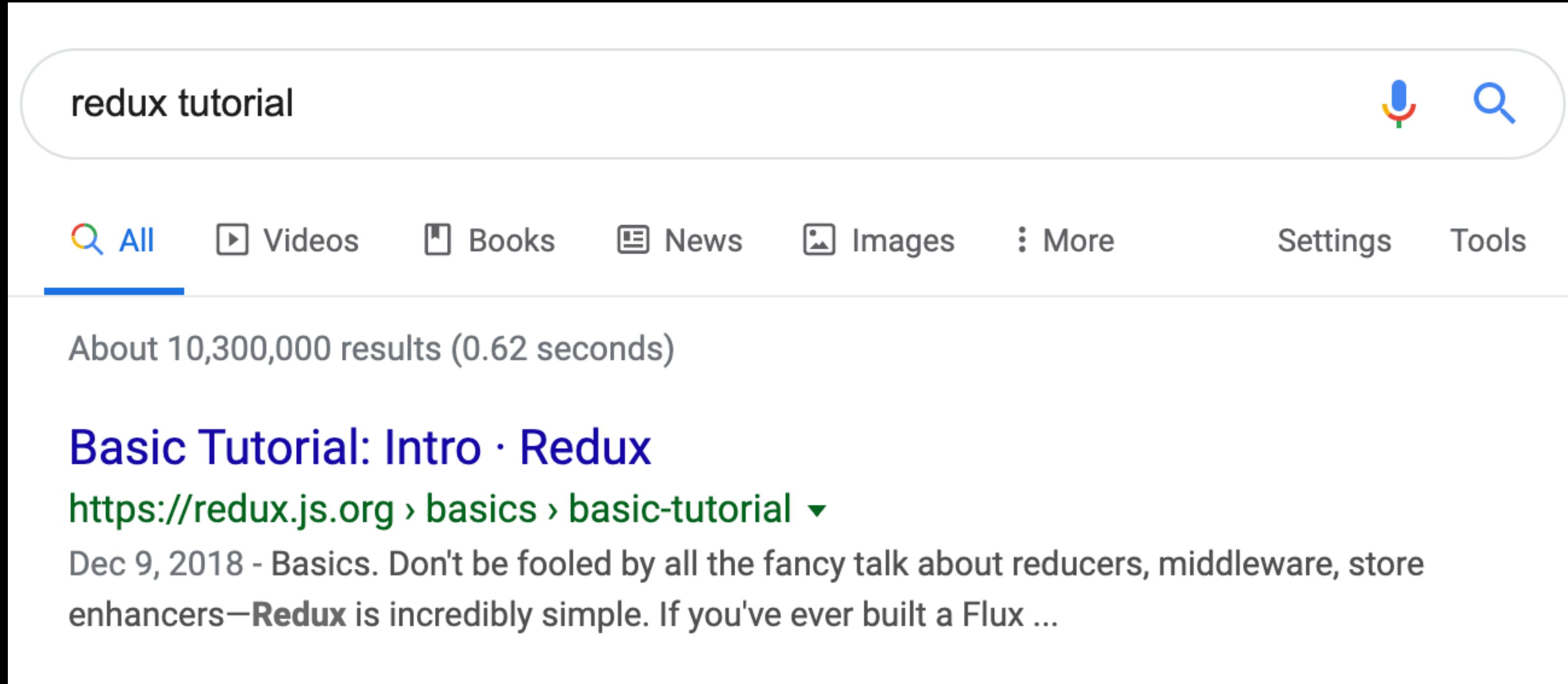
Tools

About 10,300,000 results (0.62 seconds)

Basic Tutorial: Intro · Redux

<https://redux.js.org> › basics › basic-tutorial ▾

Dec 9, 2018 - Basics. Don't be fooled by all the fancy talk about reducers, middleware, store enhancers—Redux is incredibly simple. If you've ever built a Flux ...



A screenshot of a Google search results page. The search query "redux tutorial" is entered in the search bar. Below the search bar are navigation links for "All", "Videos", "Books", "News", "Images", "More", "Settings", and "Tools". A message indicates "About 10,300,000 results (0.62 seconds)". The top result is a link titled "Basic Tutorial: Intro · Redux" from <https://redux.js.org>, dated Dec 9, 2018. The snippet describes Redux as simple and similar to Flux.

redux tutorial

All Videos Books News Images More Settings Tools

About 10,300,000 results (0.62 seconds)

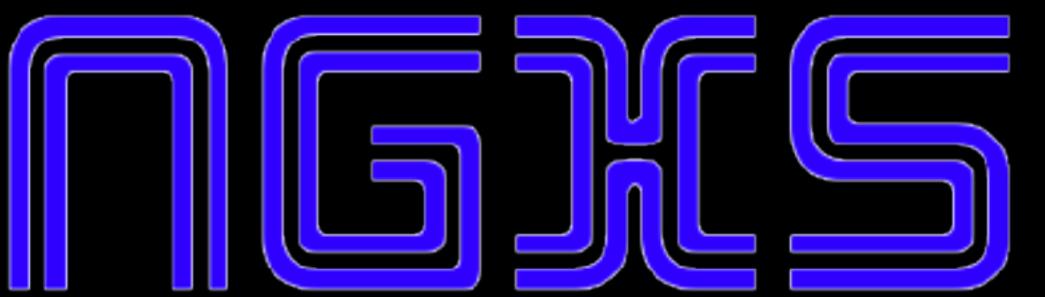
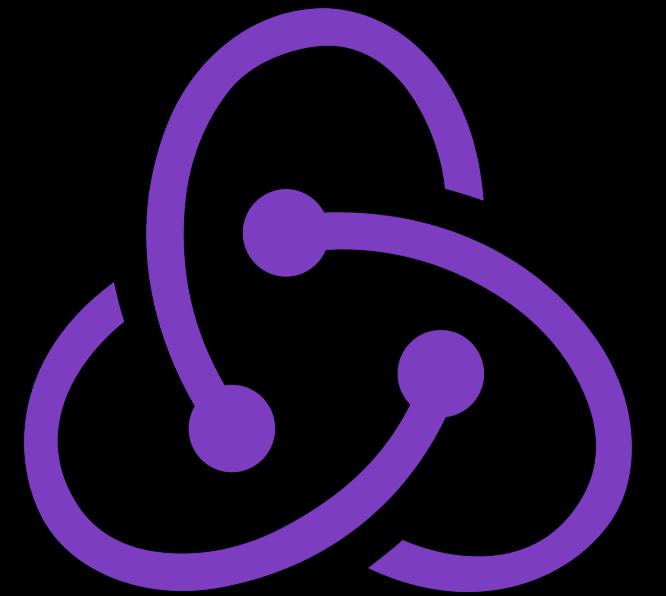
Basic Tutorial: Intro · Redux

<https://redux.js.org> › basics › basic-tutorial ▾

Dec 9, 2018 - Basics. Don't be fooled by all the fancy talk about reducers, middleware, store enhancers—Redux is incredibly simple. If you've ever built a Flux ...

How I initially envisioned
NgRx in my codebase





@ngrx example application

Example application utilizing @ngrx libraries, showcasing common patterns and best practices. Try it on [StackBlitz](#).

This app is a book collection manager. The user can authenticate, use the Google Books API to search for books and add them to their collection. This application utilizes [@ngrx/store](#) to manage the state of the app and to cache requests made to the Google Books API; [@ngrx/effects](#) to isolate side effects; [@angular/router](#) to manage navigation between routes; [@angular/material](#) to provide design and styling.



Examples

Redux is distributed with a few examples in its [source code](#). Most of these examples are also on [CodeSandbox](#), an online editor that lets you play with the examples online.

Angular State Management with NGXS

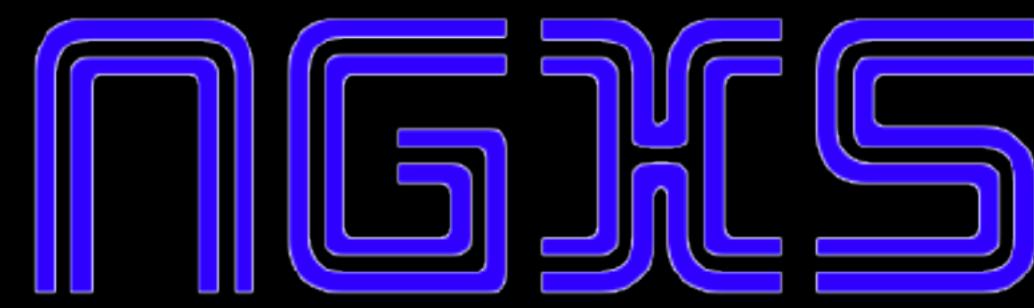
Start by installing the latest [@ngxs/store](#) package from npm.

```
$ npm i @ngxs/store --save
```



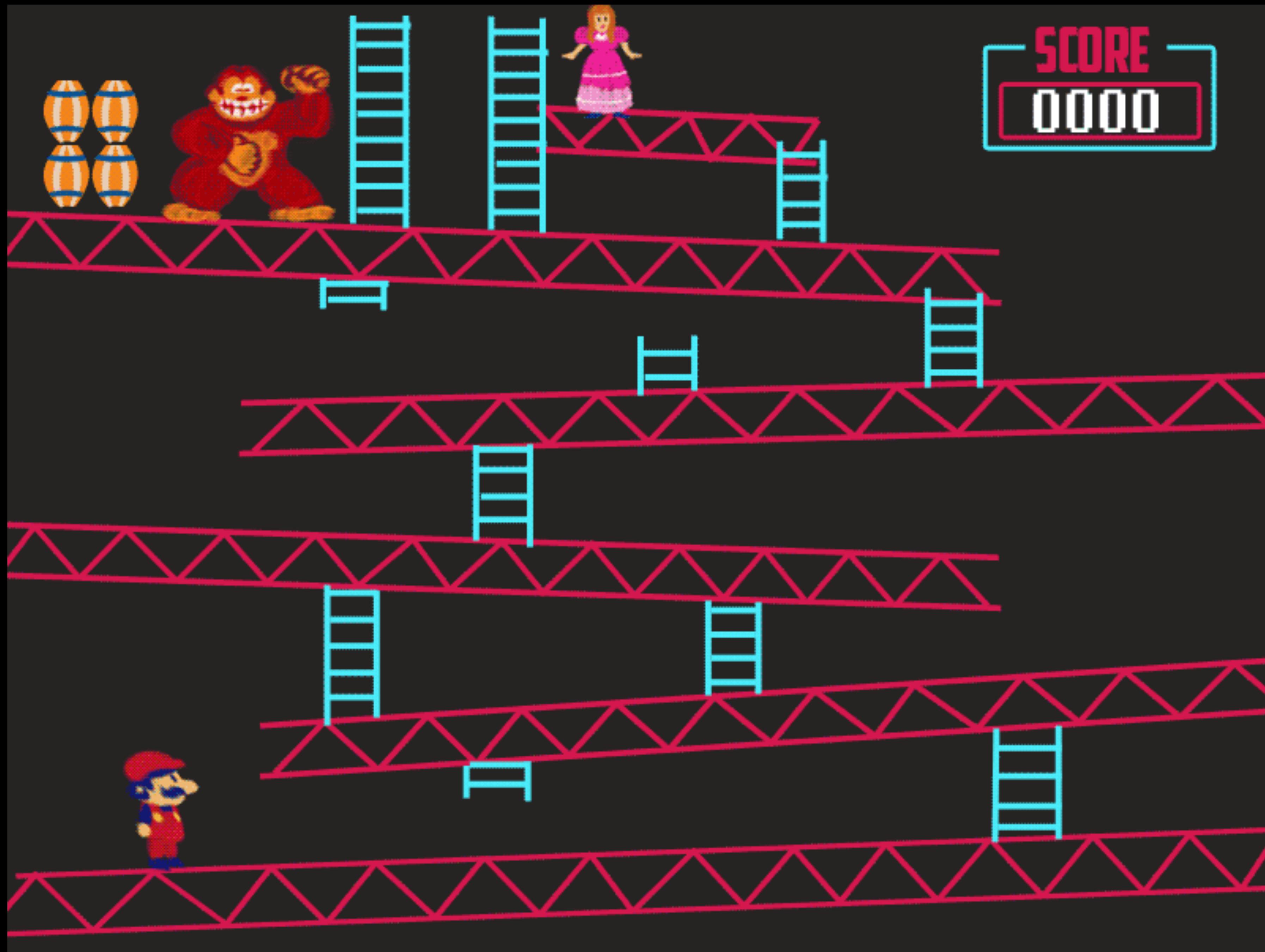


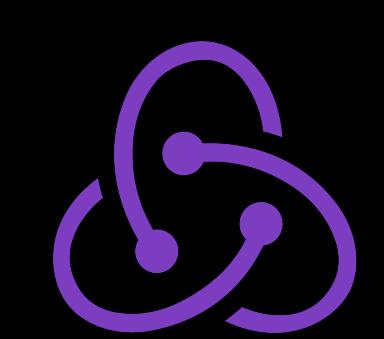
Independent



Many more!

Fearless

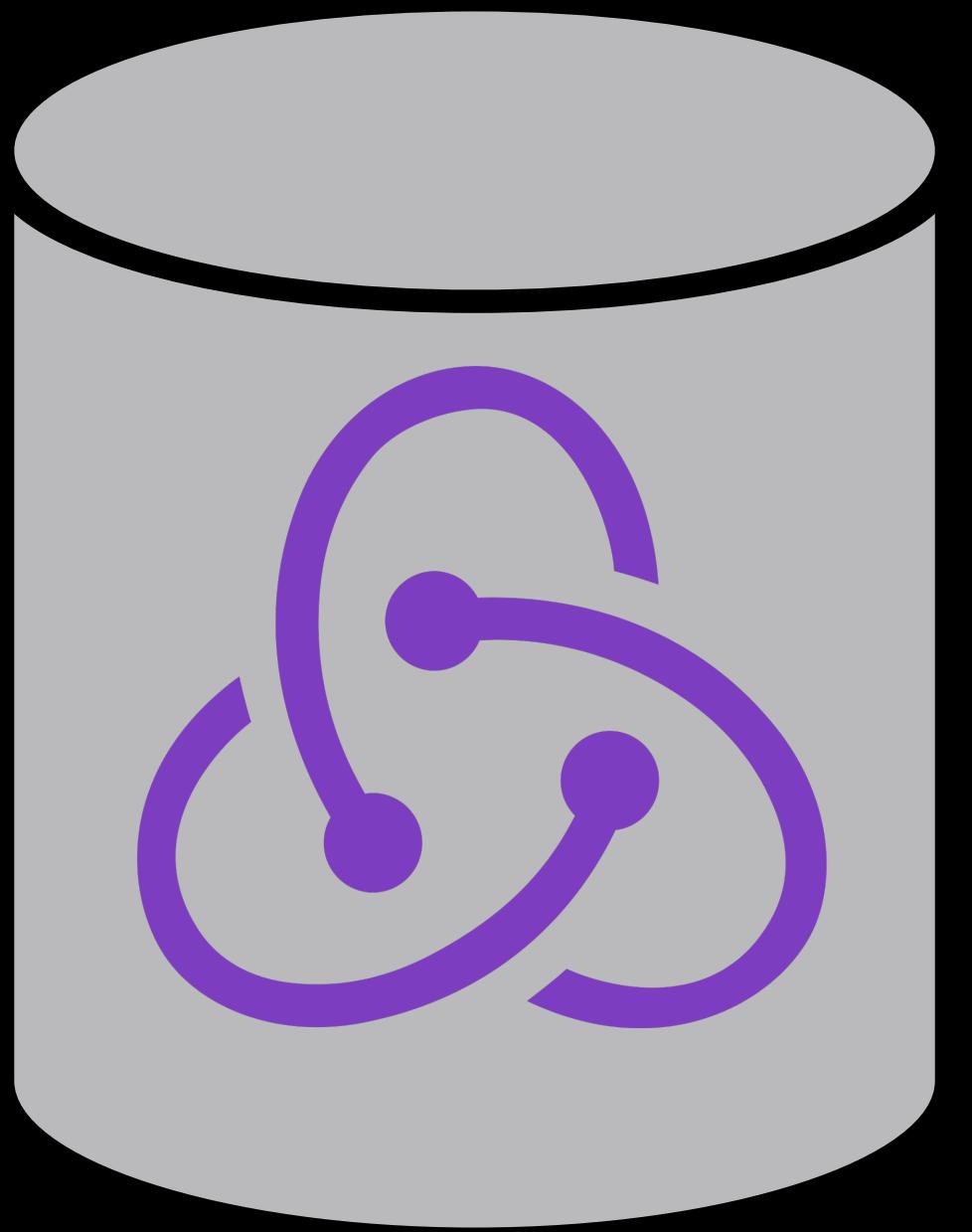




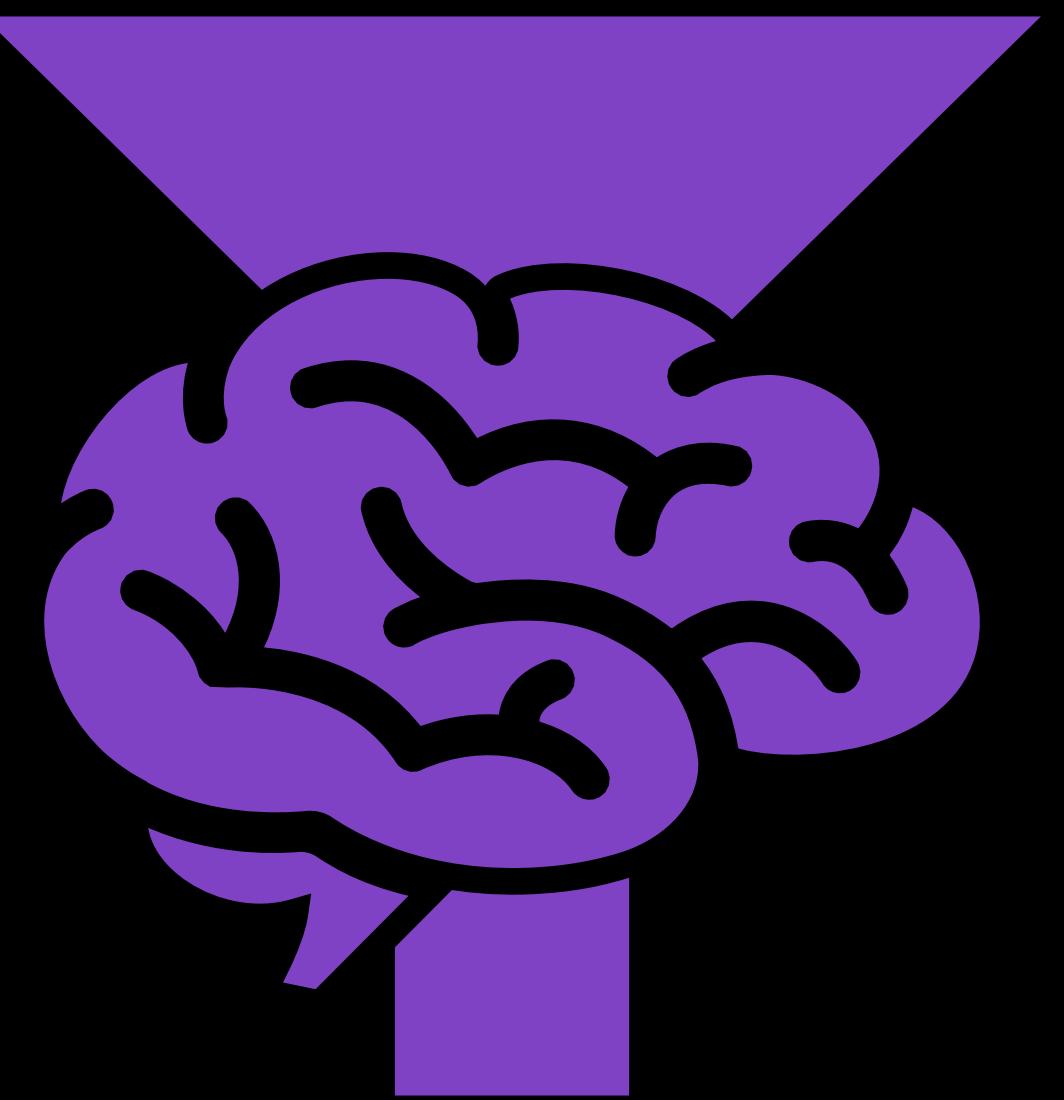
Be a Better Mario!



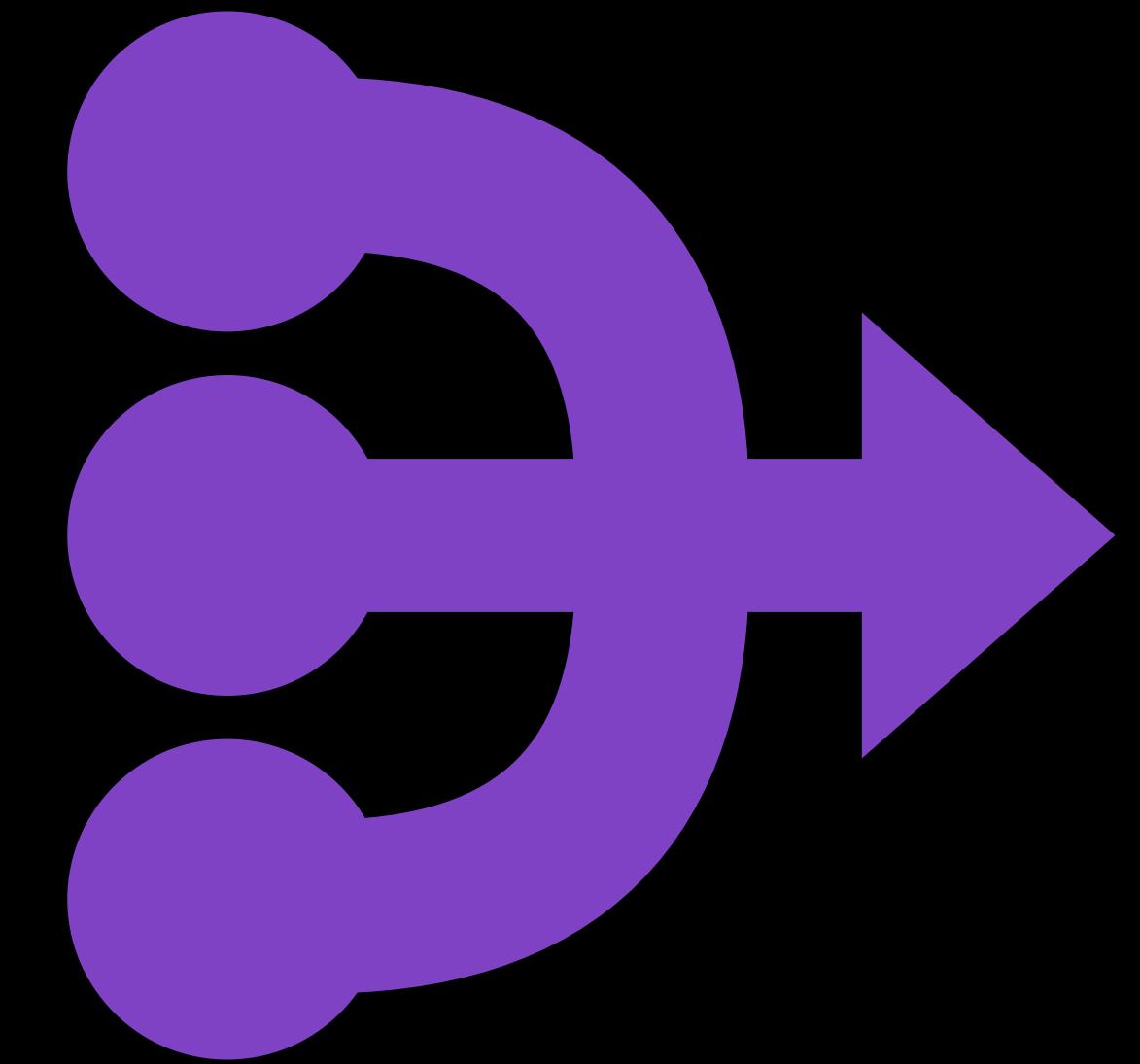
Agenda



Store



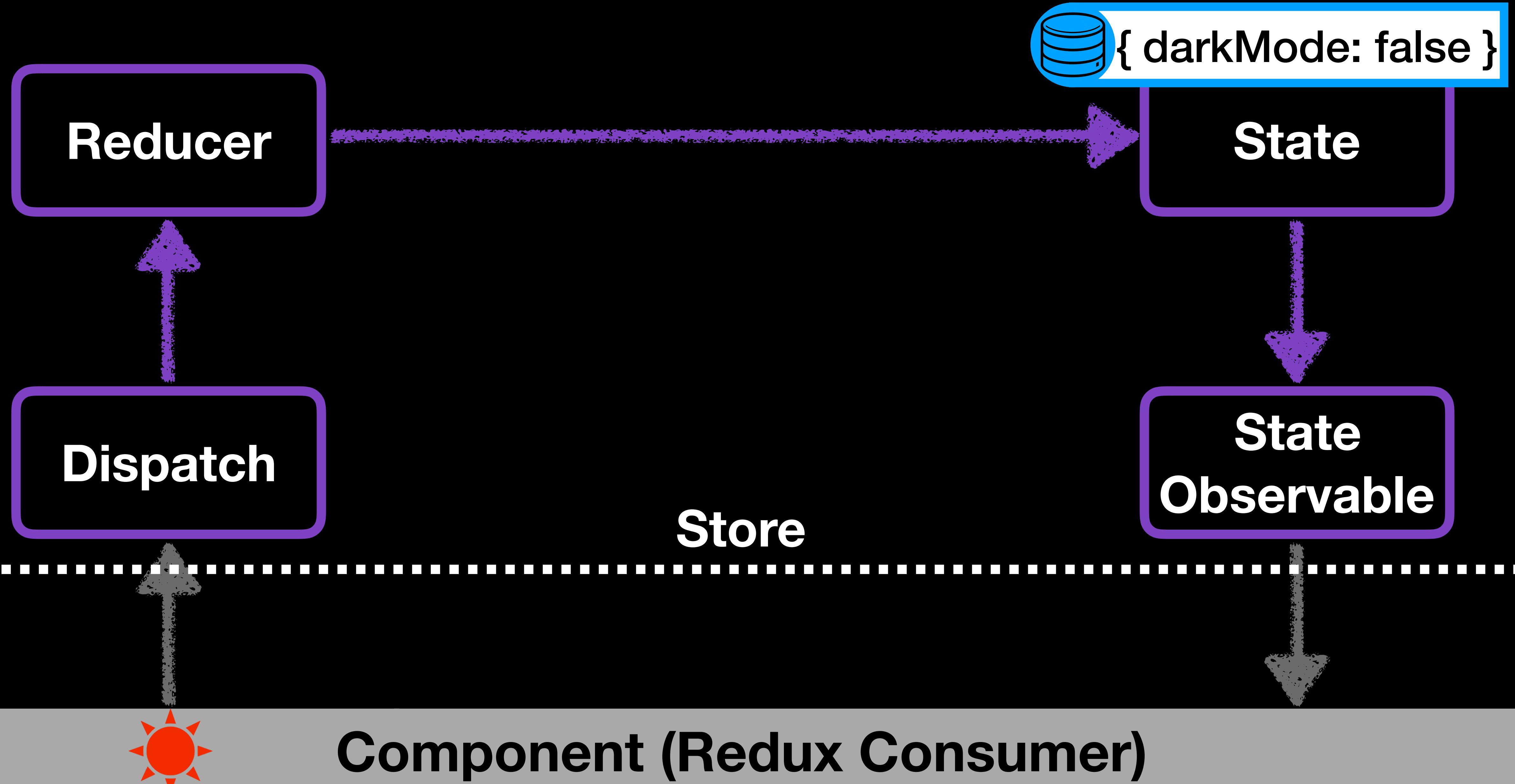
Memoized
Selector



Combine
Reducers



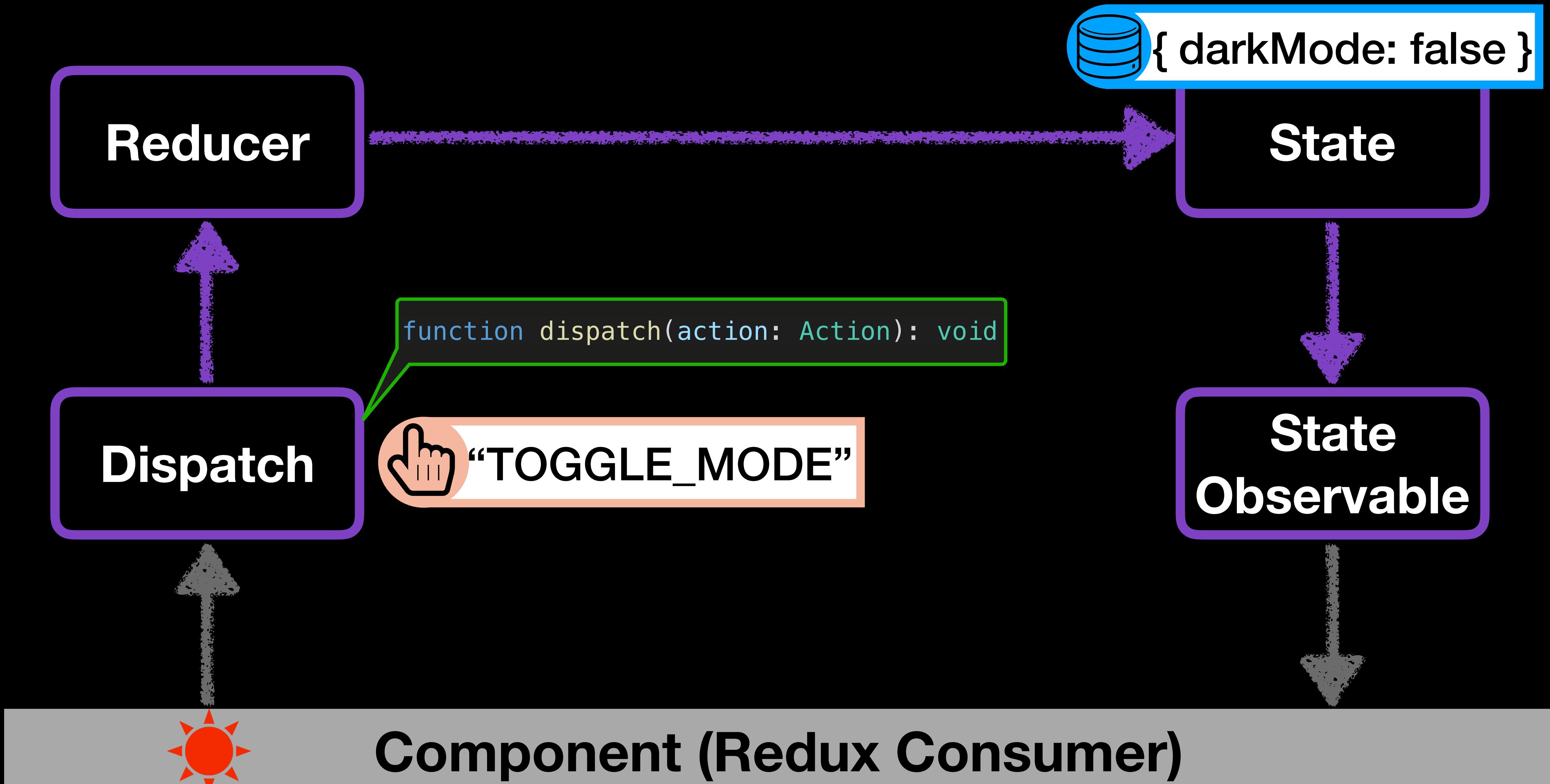
Redux Data Flow



Component (Redux Consumer)

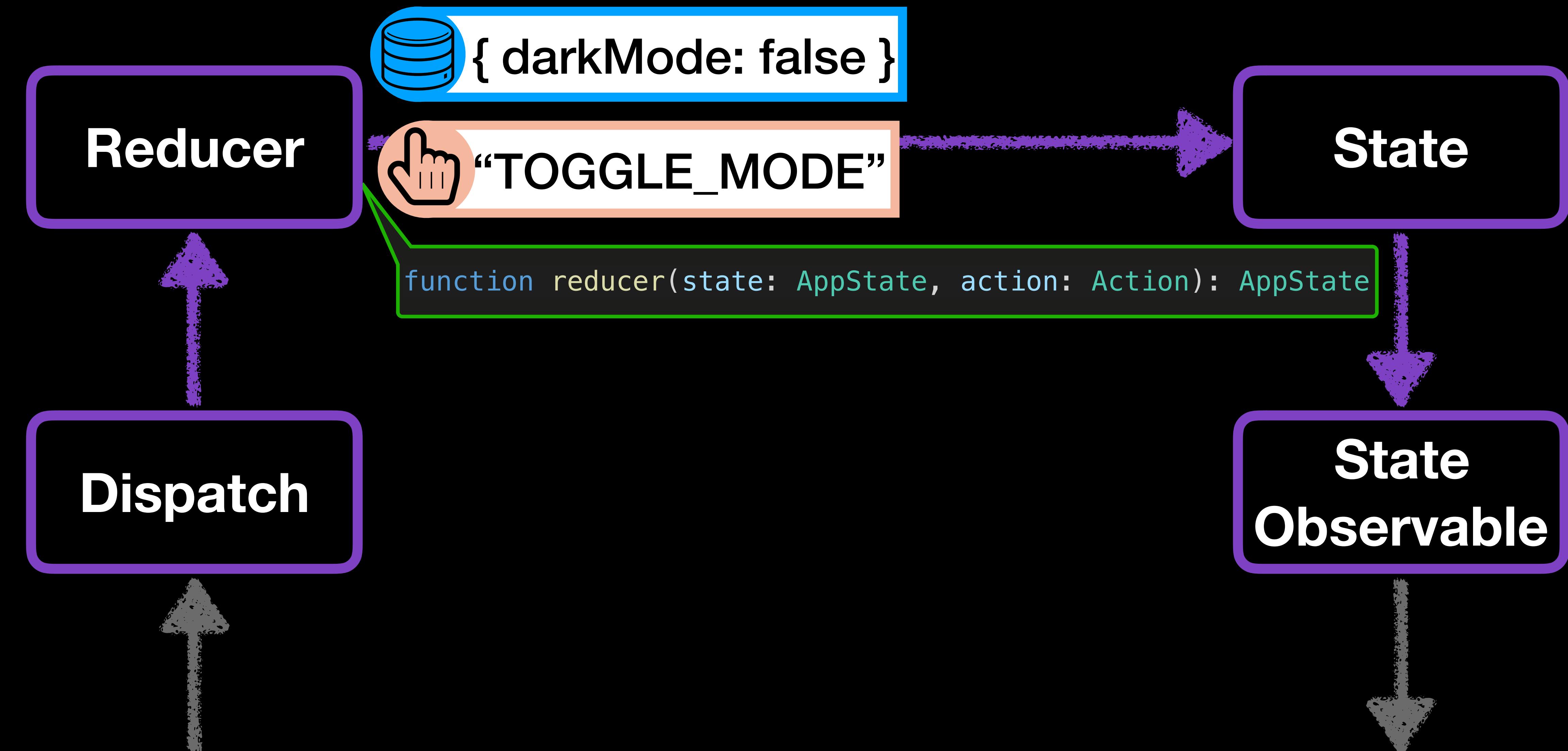


Redux Data Flow





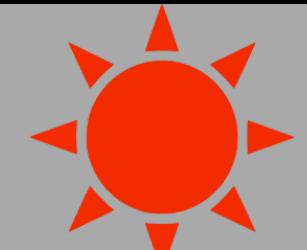
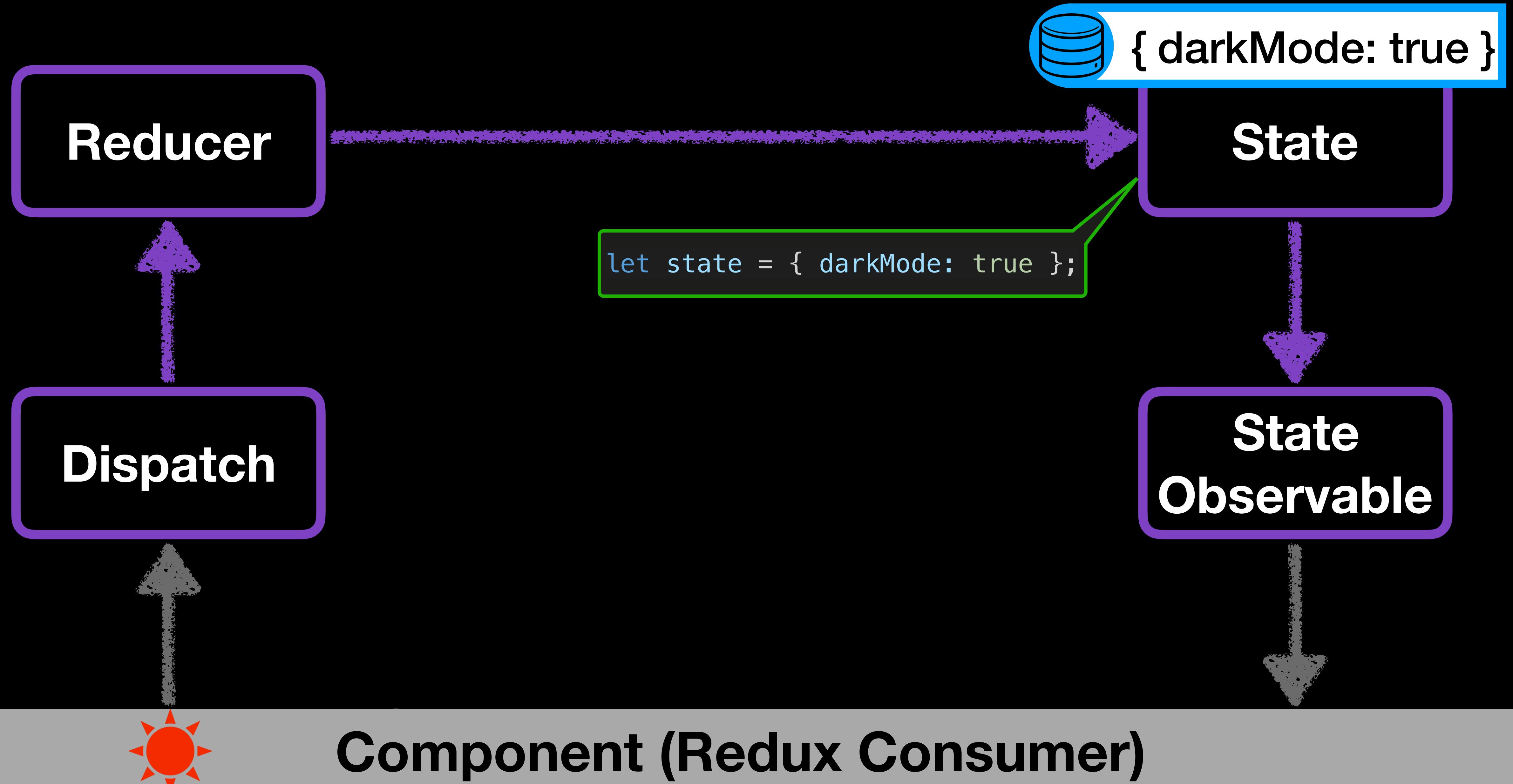
Redux Data Flow



Component (Redux Consumer)



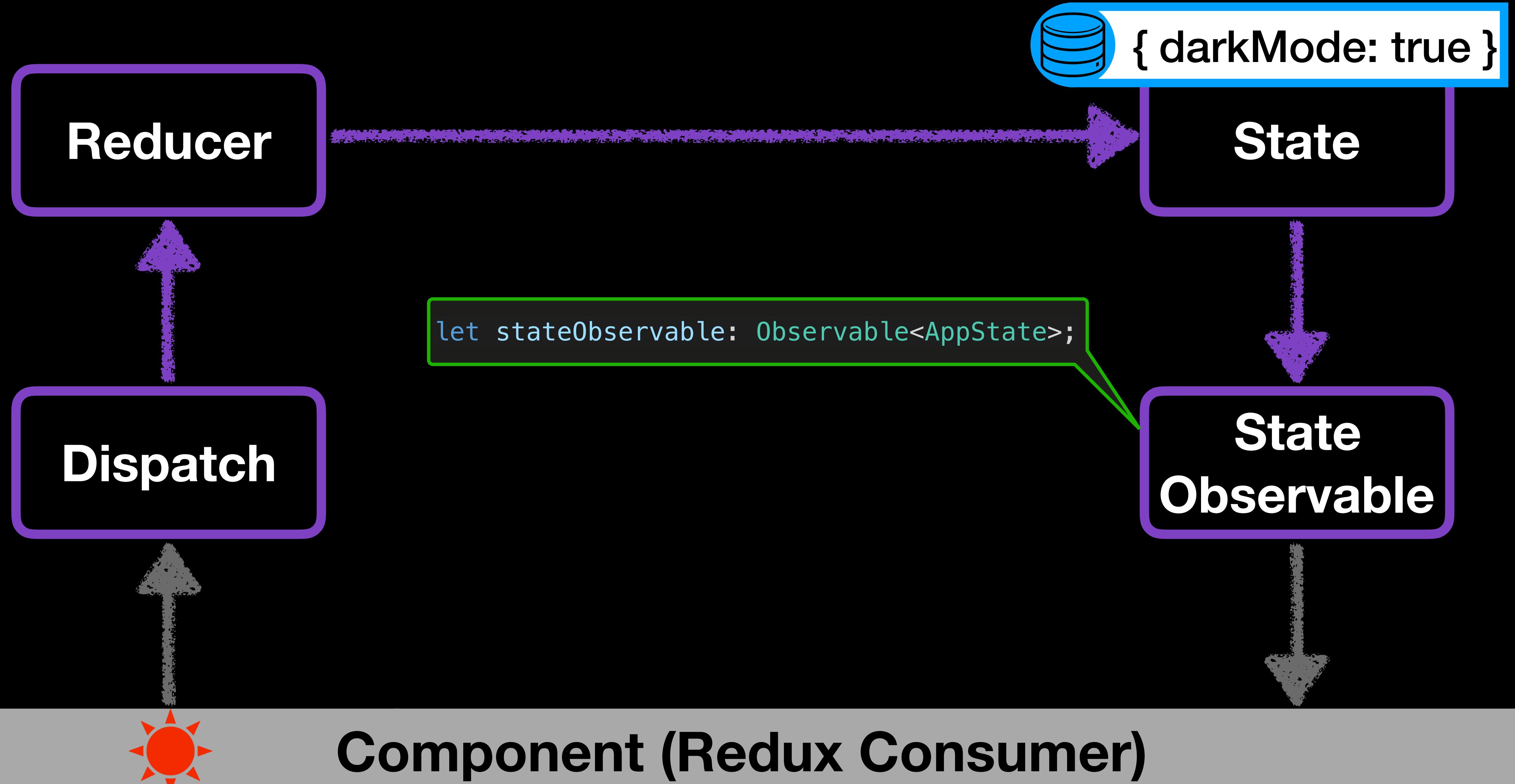
Redux Data Flow



Component (Redux Consumer)

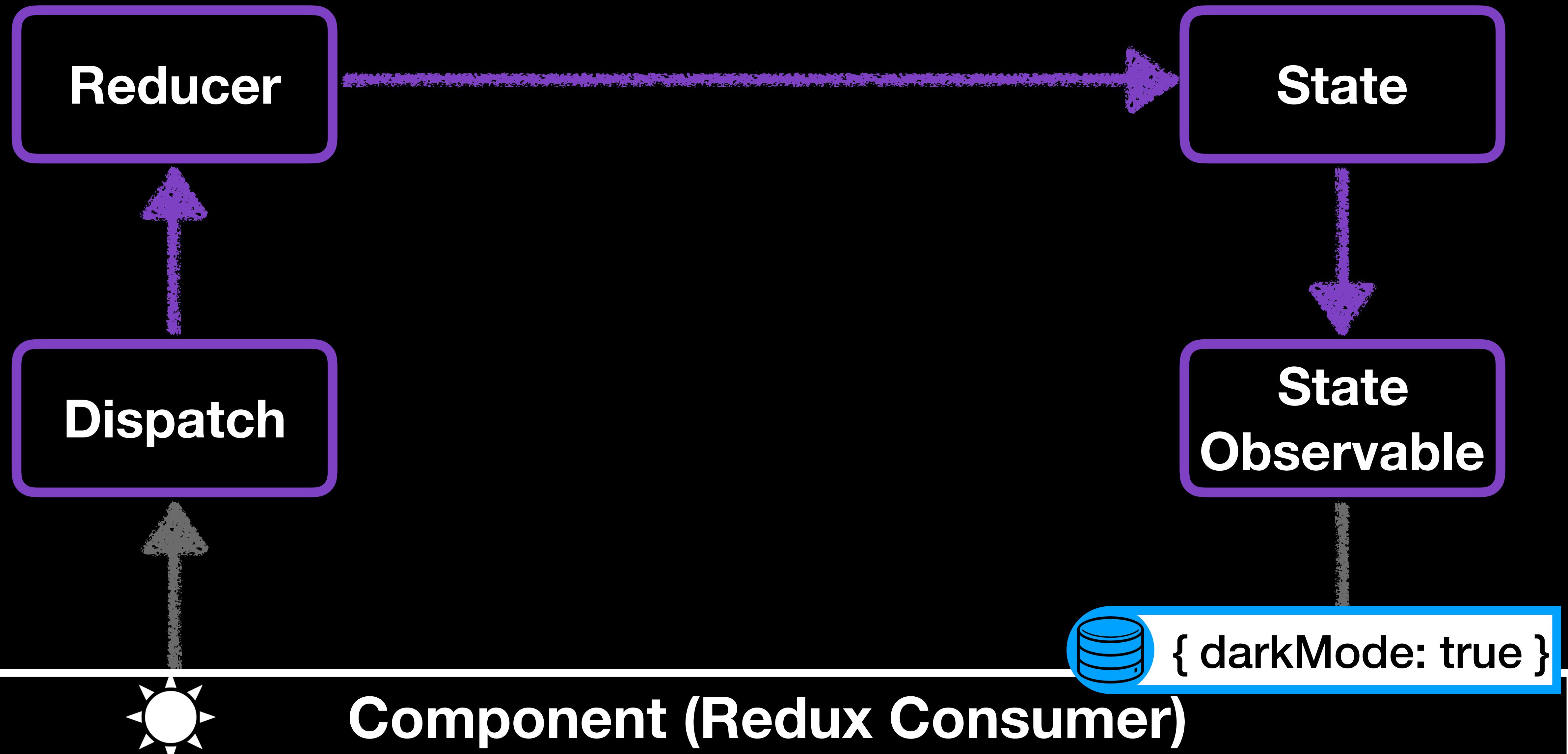


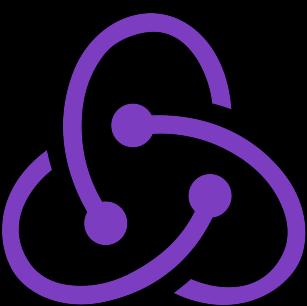
Redux Data Flow



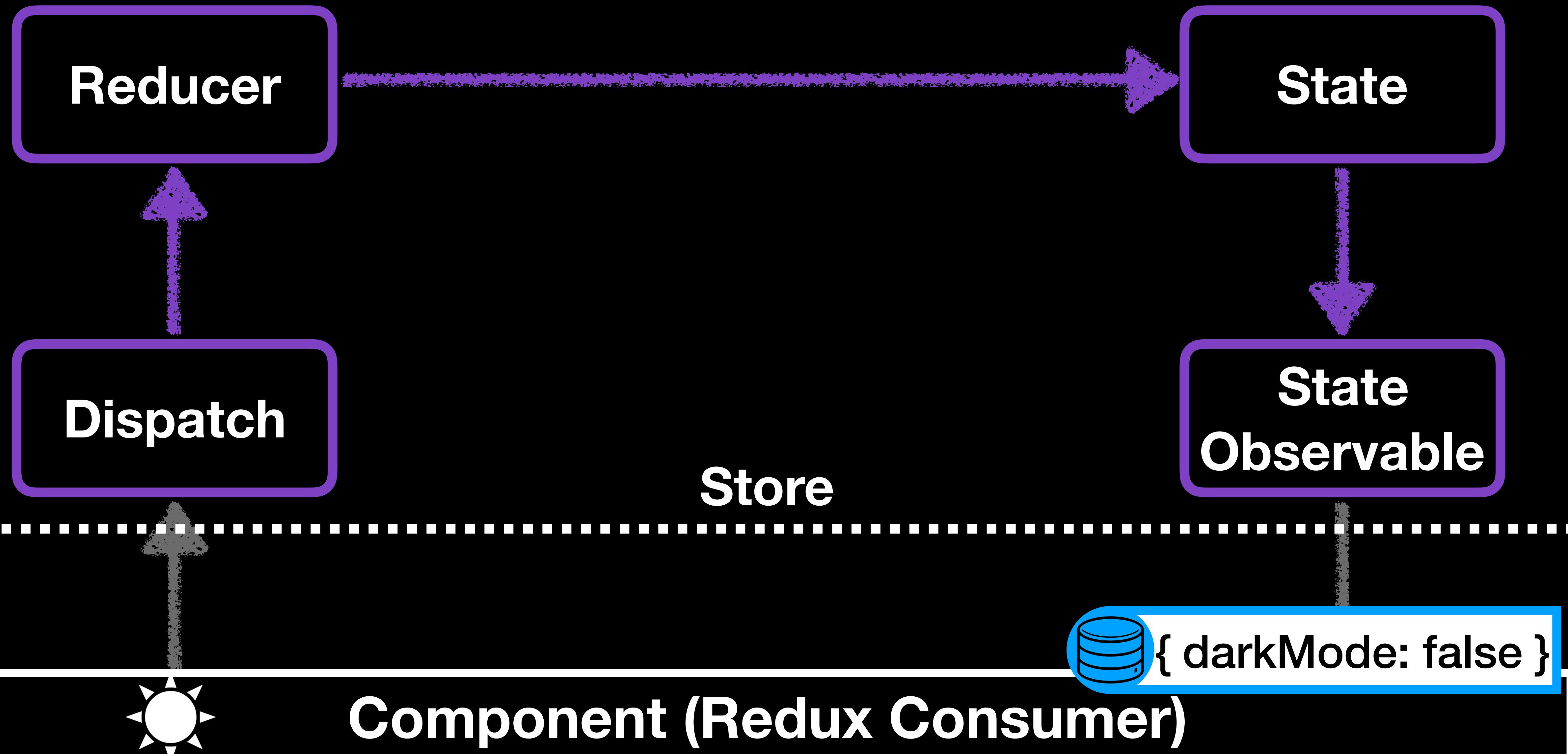


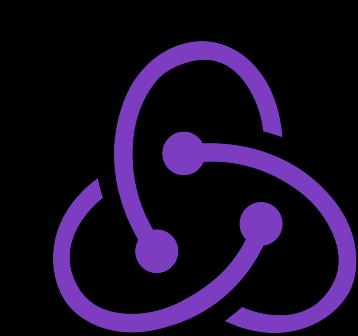
Redux Data Flow





Redux Data Flow

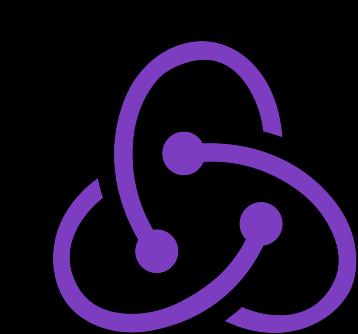




Redux Definitions

```
export interface Action {  
  type: string;  
}
```

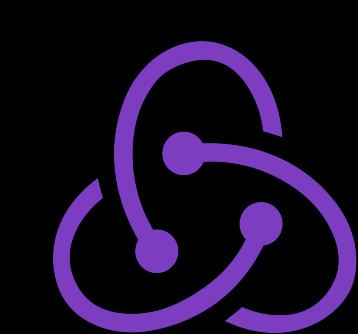
```
type ReducerFunction<S> = (state: S, action: Action) => S;
```



Redux Store Implementation

```
export class Store<S> {
```

```
}
```



Redux Store Implementation

```
export class Store<S> {  
  // Implementation details...  
  dispatch(action: Action) {  
    // Implementation details...  
  }  
}
```



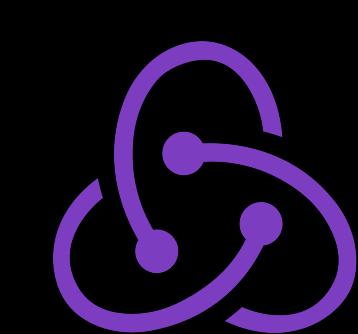
Redux Store Implementation

```
export class Store<S> {  
  
  constructor(private reducer: ReducerFunction<S>) {  
  
  }  
  
  dispatch(action: Action) {  
  
  }  
}
```



Redux Store Implementation

```
export class Store<S> {  
  
  constructor(private reducer: ReducerFunction<S>) {  
  
  }  
  
  dispatch(action: Action) {  
    this.state = this.reducer(this.state, action);  
  
  }  
}
```

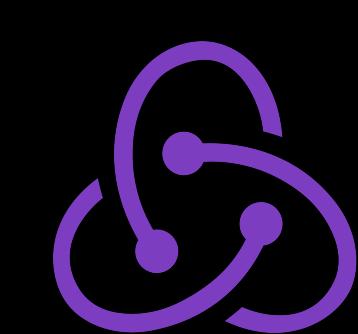


Redux Store Implementation

```
export class Store<S> {
    private state: S;

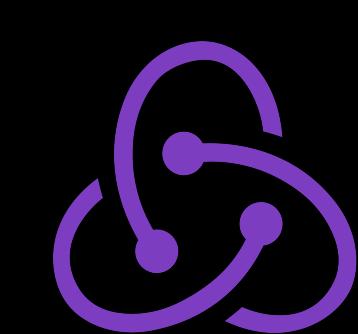
    constructor(private reducer: ReducerFunction<S>) {
        this.state = initialS;
    }

    dispatch(action: Action) {
        this.state = this.reducer(this.state, action);
    }
}
```



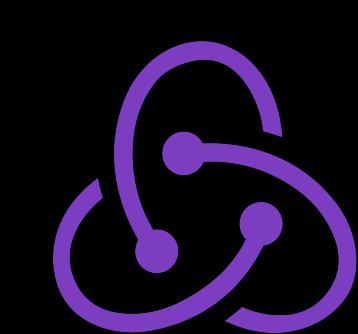
Redux Store Implementation

```
export class Store<S> {  
    private state: S;  
    private stateSubject: BehaviorSubject<S>;  
  
    constructor(private reducer: ReducerFunction<S>) {  
  
    }  
  
    dispatch(action: Action) {  
        this.state = this.reducer(this.state, action);  
  
    }  
}
```



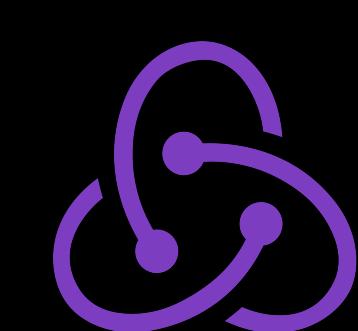
Redux Store Implementation

```
export class Store<S> {  
    private state: S;  
    private stateSubject: BehaviorSubject<S>;  
  
    constructor(private reducer: ReducerFunction<S>) {  
        this.stateSubject = new BehaviorSubject(this.state);  
  
    }  
  
    dispatch(action: Action) {  
        this.state = this.reducer(this.state, action);  
  
    }  
}
```



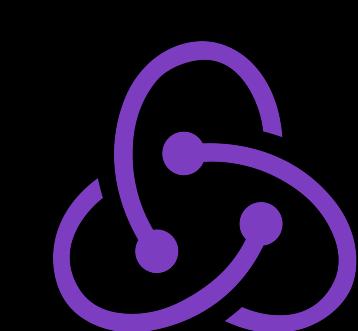
Redux Store Implementation

```
export class Store<S> {
    private state: S;
    private stateSubject: BehaviorSubject<S>;
    constructor(private reducer: ReducerFunction<S>) {
        this.stateSubject = new BehaviorSubject(this.state);
    }
    dispatch(action: Action) {
        this.state = this.reducer(this.state, action);
        this.stateSubject.next(this.state);
    }
}
```



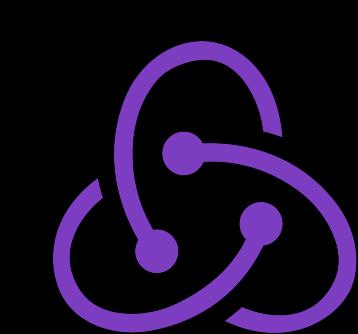
Redux Store Implementation

```
export class Store<S> {
  private state: S;
  private stateSubject: BehaviorSubject<S>;
  readonly stateObservable: Observable<S>;
  constructor(private reducer: ReducerFunction<S>) {
    this.stateSubject = new BehaviorSubject(this.state);
  }
  dispatch(action: Action) {
    this.state = this.reducer(this.state, action);
    this.stateSubject.next(this.state);
  }
}
```



Redux Store Implementation

```
export class Store<S> {
    private state: S;
    private stateSubject: BehaviorSubject<S>;
    readonly stateObservable: Observable<S>;
    constructor(private reducer: ReducerFunction<S>) {
        this.stateSubject = new BehaviorSubject(this.state);
        this.stateObservable = this.stateSubject.asObservable();
    }
    dispatch(action: Action) {
        this.state = this.reducer(this.state, action);
        this.stateSubject.next(this.state);
    }
}
```

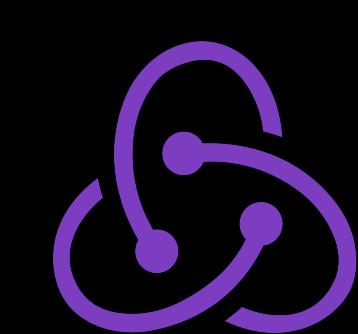


Redux Store Implementation

```
export class Store<S> {
    private state: S;
    private stateSubject: BehaviorSubject<S>;
    readonly stateObservable: Observable<S>;
    constructor(private reducer: ReducerFunction<S>) {
        this.stateSubject = new BehaviorSubject(this.state);
        this.stateObservable = this.stateSubject.asObservable();
        this.dispatch({ type: 'INIT' });
    }
    dispatch(action: Action) {
        this.state = this.reducer(this.state, action);
        this.stateSubject.next(this.state);
    }
}
```

Store Demo





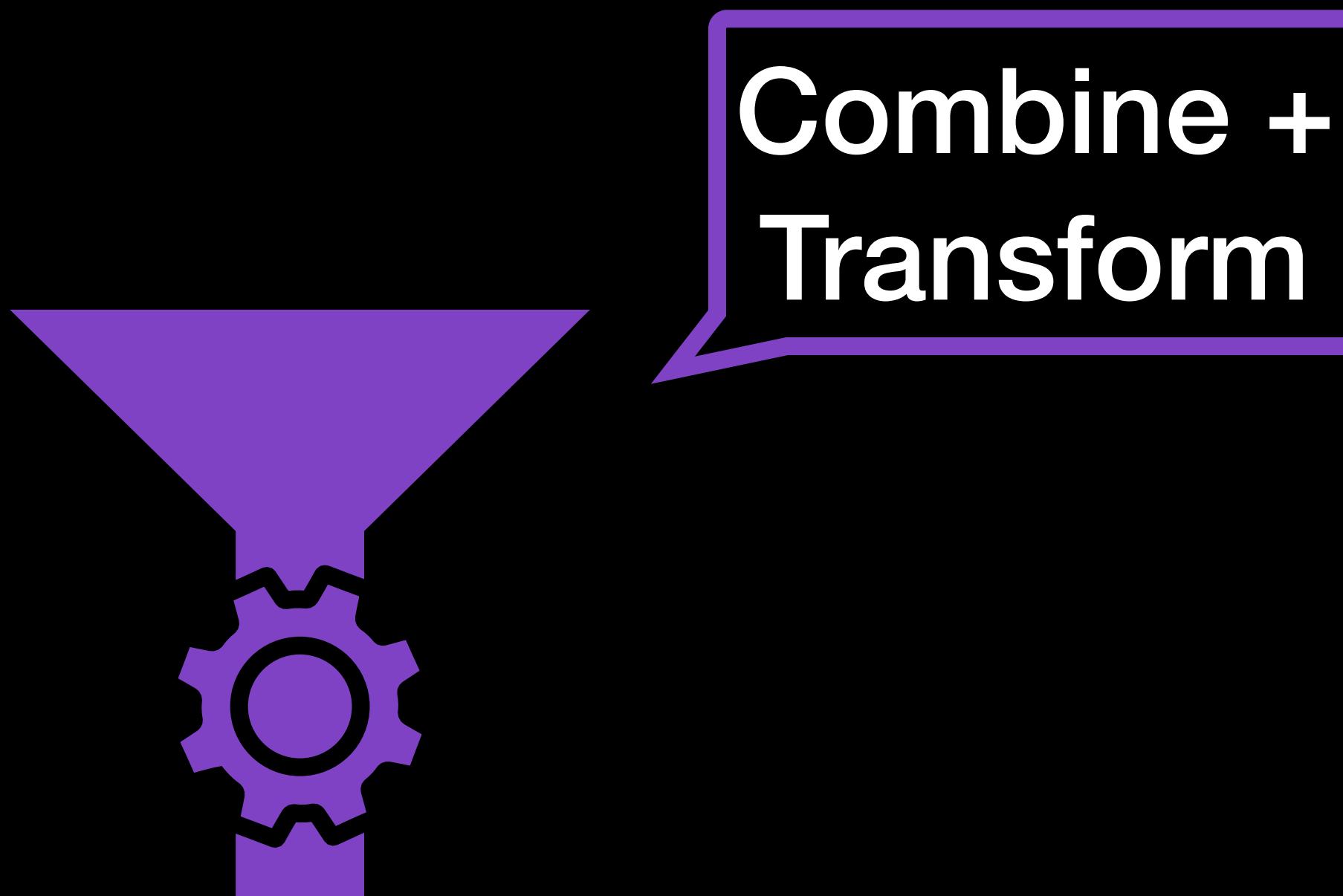
createSelector / Reselect

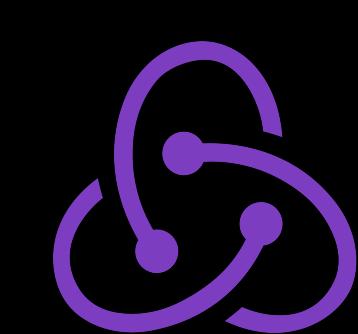
```
export const getTotalCostSelector = createSelector(  
  [  
    getCartSelector,  
    getTaxRateSelector  
  ],  
  (cart: number[], taxRate: number) => getTotalCost(cart, taxRate)  
);
```



createSelector / Reselect

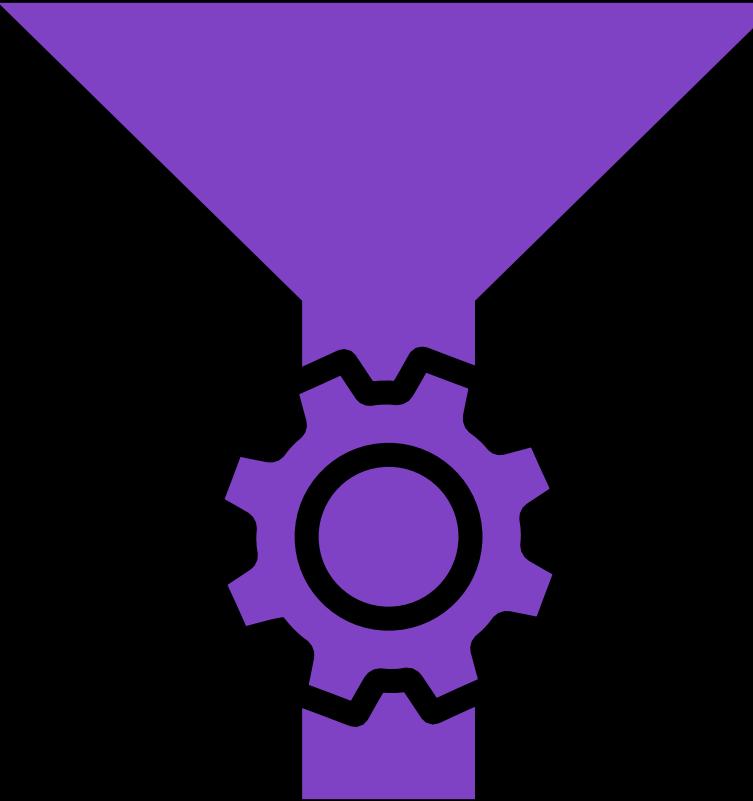
```
export const getTotalCostSelector = createSelector(  
  [  
    getCartSelector,  
    getTaxRateSelector  
  ],  
  (cart: number[], taxRate: number) => getTotalCost(cart, taxRate)  
);
```



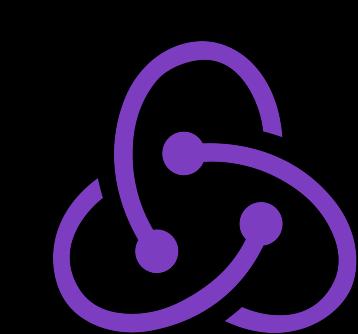


createSelector / Reselect

```
export const getTotalCostSelector = createSelector(  
  [  
    getCartSelector,  
    getTaxRateSelector  
  ],  
  (cart: number[], taxRate: number) => getTotalCost(cart, taxRate)  
);
```

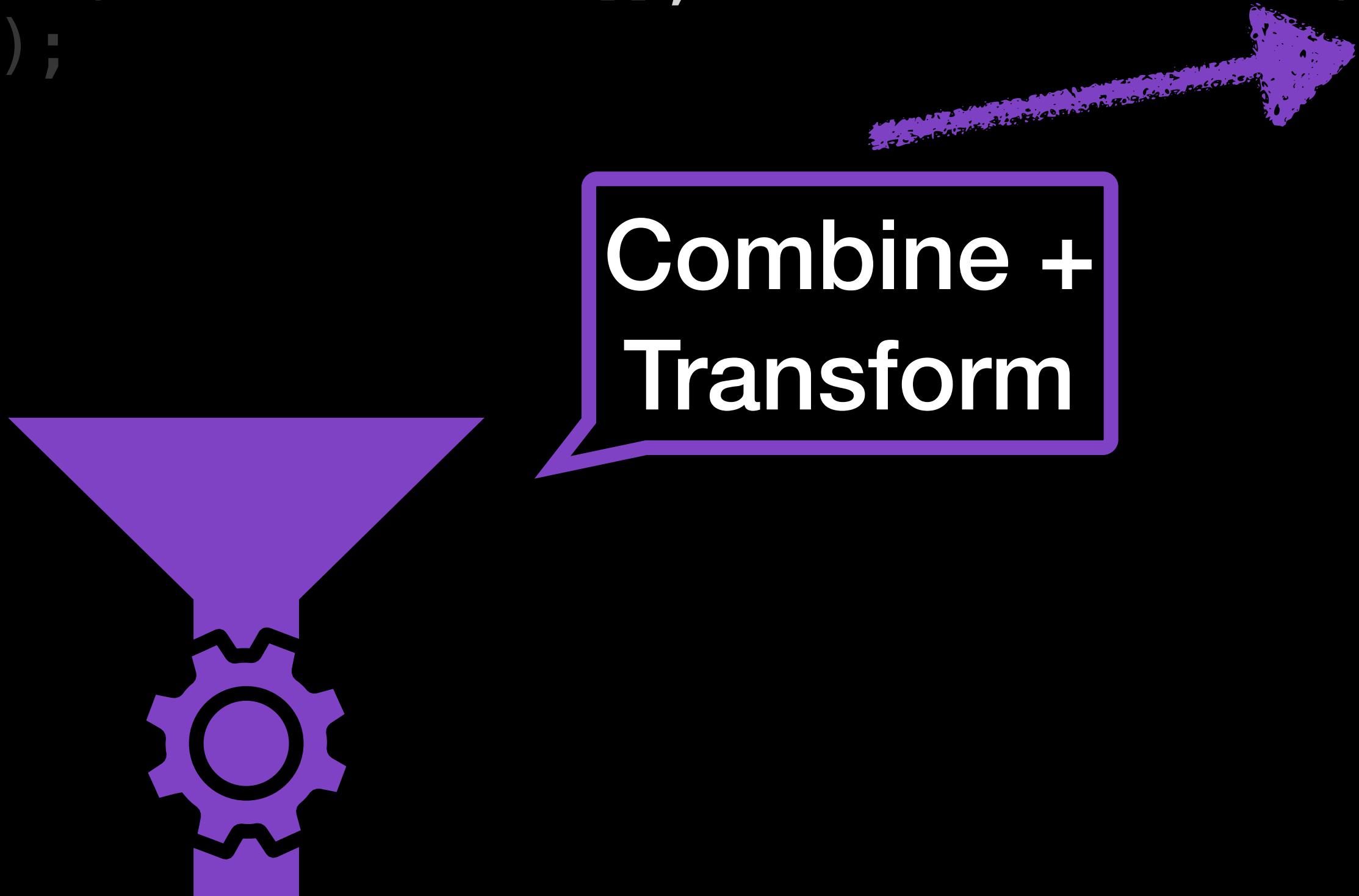


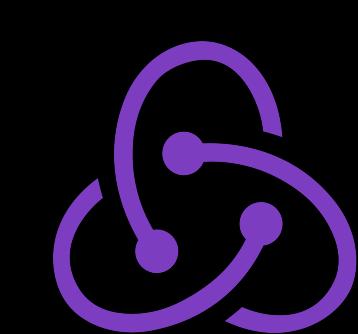
Combine +
Transform



createSelector / Reselect

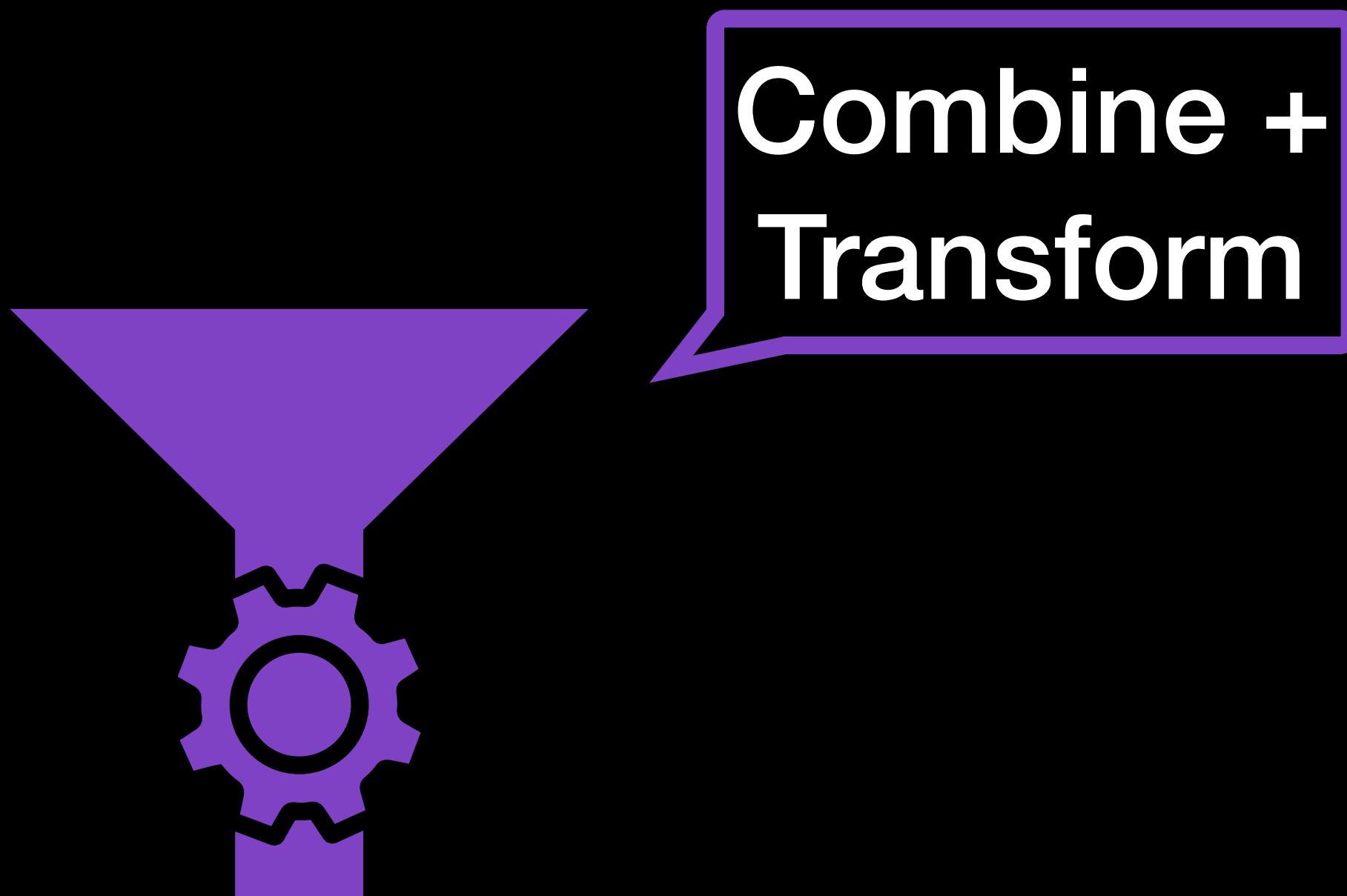
```
export const getTotalCostSelector = createSelector(  
  [  
    getCartSelector,  
    getTaxRateSelector  
  ],  
  (cart: number[], taxRate: number) => getTotalCost(cart, taxRate)  
);
```



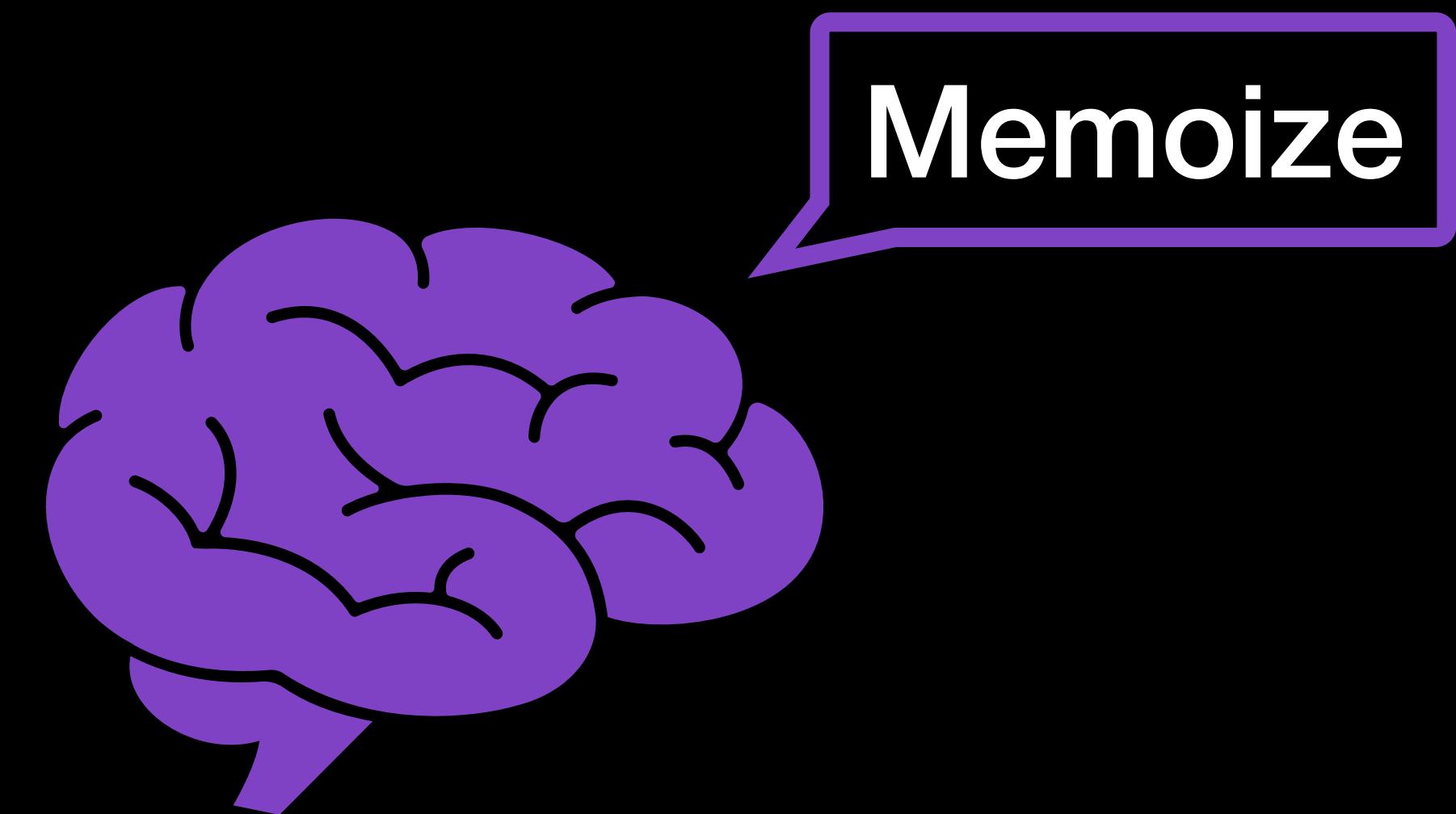


createSelector / Reselect

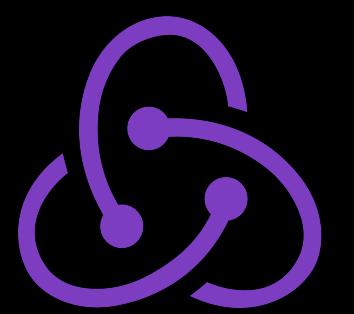
```
export const getTotalCostSelector = createSelector(  
  [  
    getCartSelector,  
    getTaxRateSelector  
  ],  
  (cart: number[], taxRate: number) => getTotalCost(cart, taxRate)  
);
```



Combine +
Transform



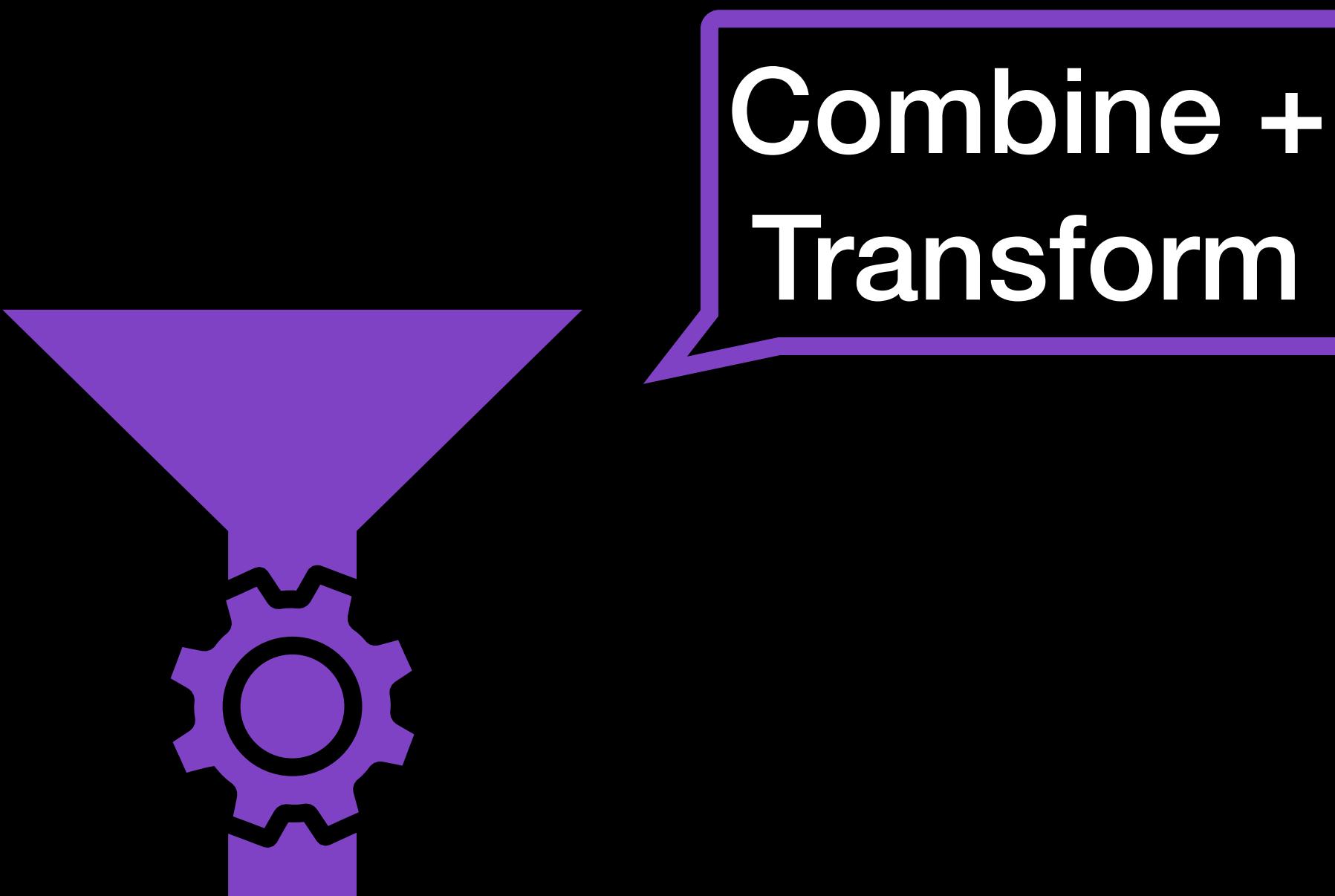
Memoize



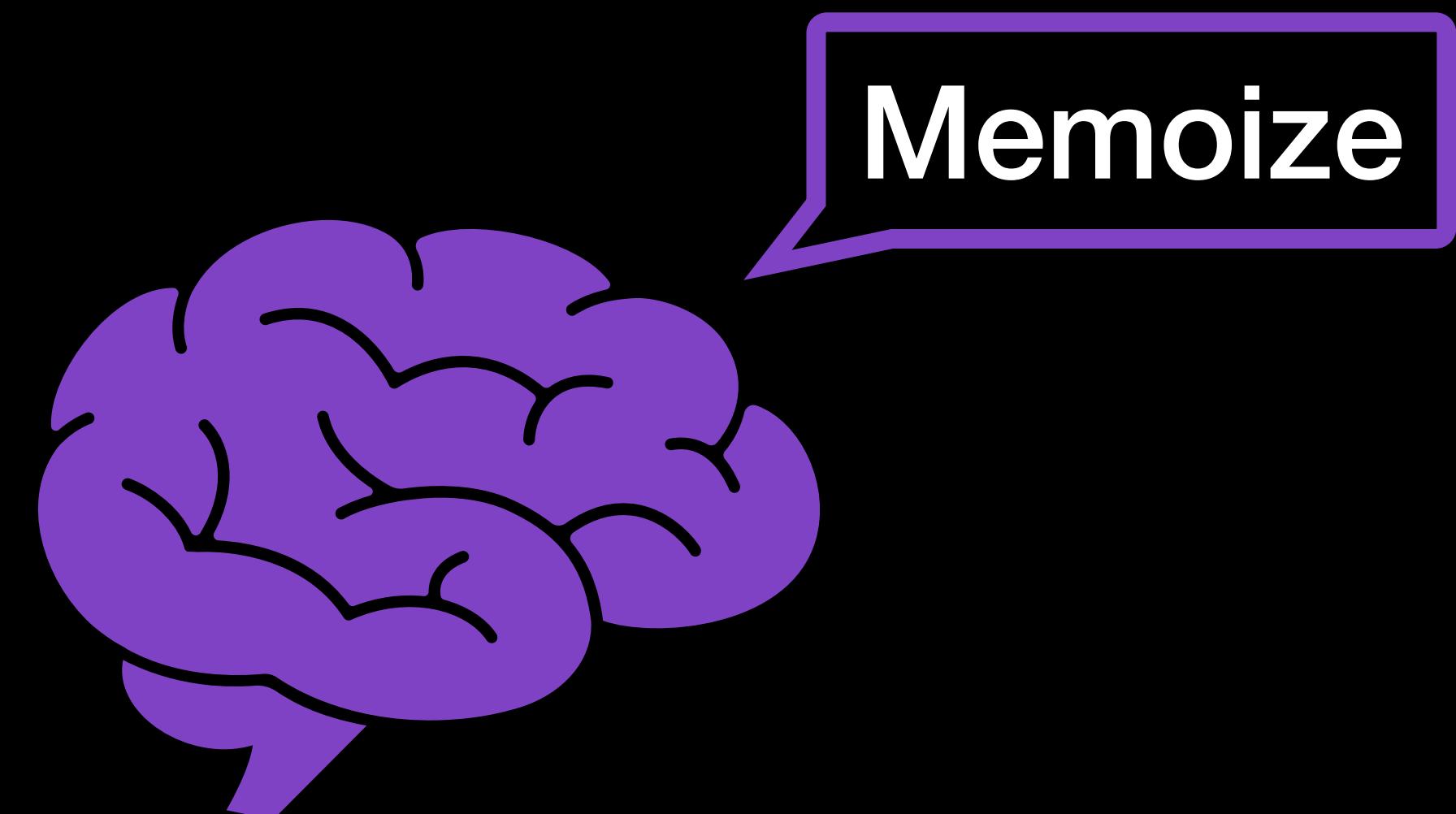
createSelector / Reselect

```
export const getTotalCostSelector = createSelector(  
  [  
    getCartSelector,  
    getTaxRateSelector  
  ],  
  (cart: number[], taxRate: number) => getTotalCost(cart, taxRate)  
);
```

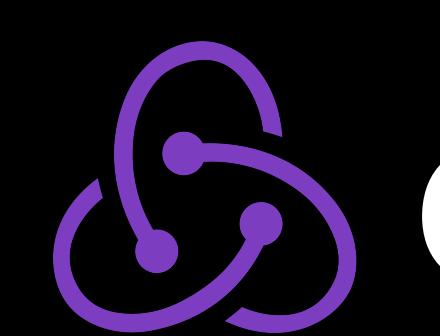
Input hasn't changed? Don't rerun the transformation!



Combine +
Transform

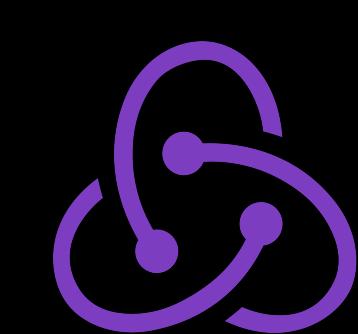


Memoize

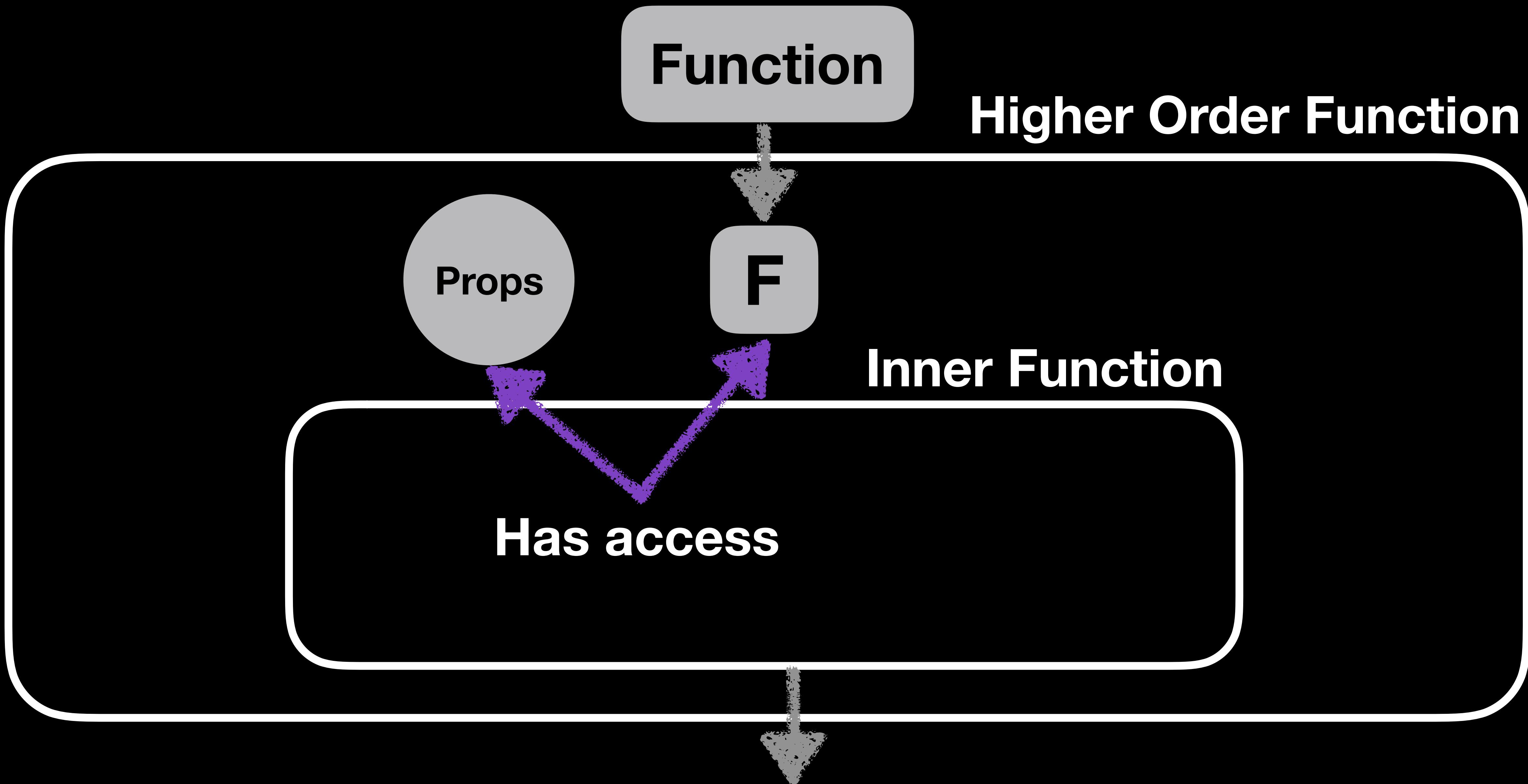


createSelector / Reselect

```
function createSelector(  
  selectorFns: SelectorFn[],  
  transformationFn: AnyFn  
) : SelectorFn
```



Higher Order Function



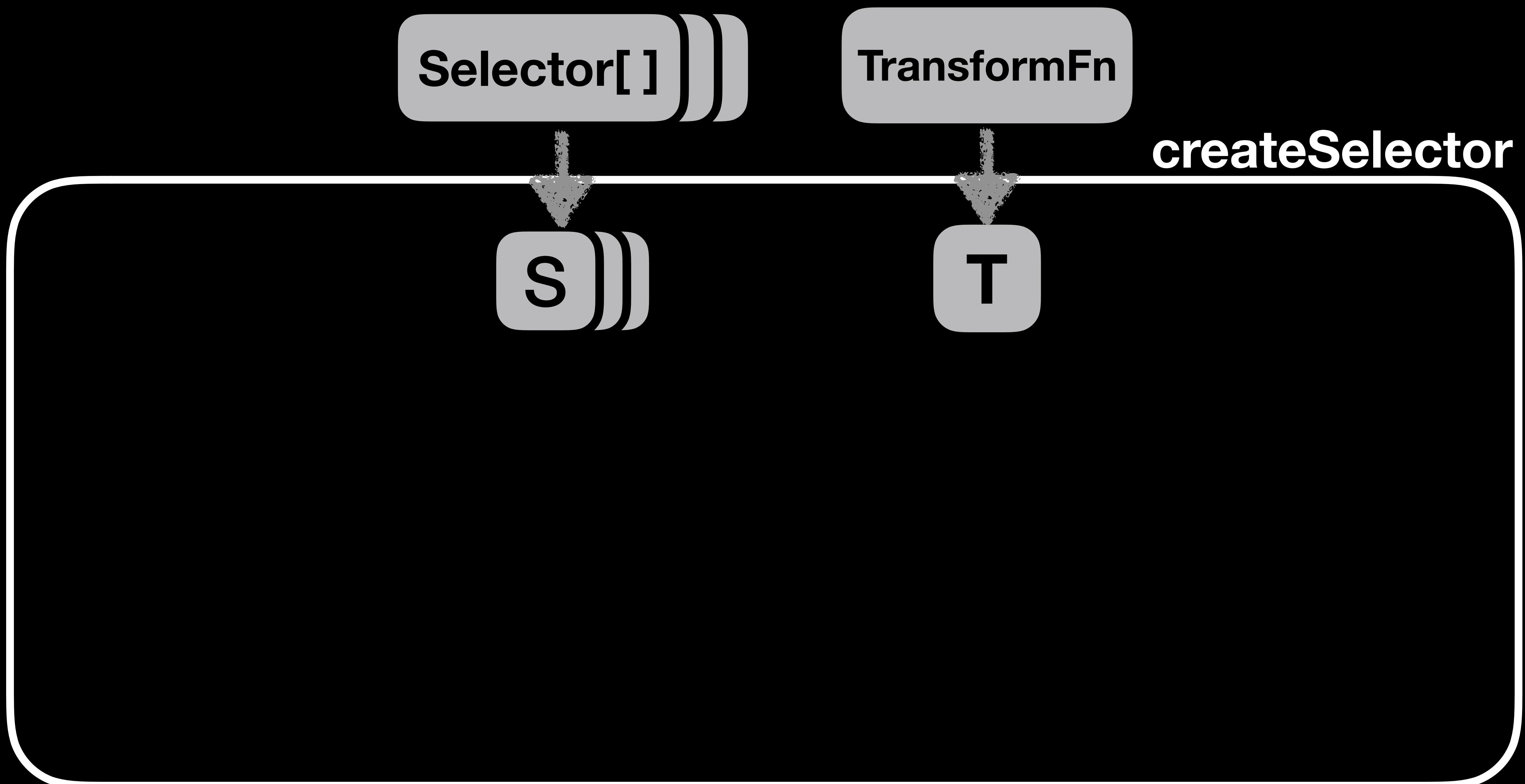
Selector[]

TransformFn

createSelector

s

T



Selector[]

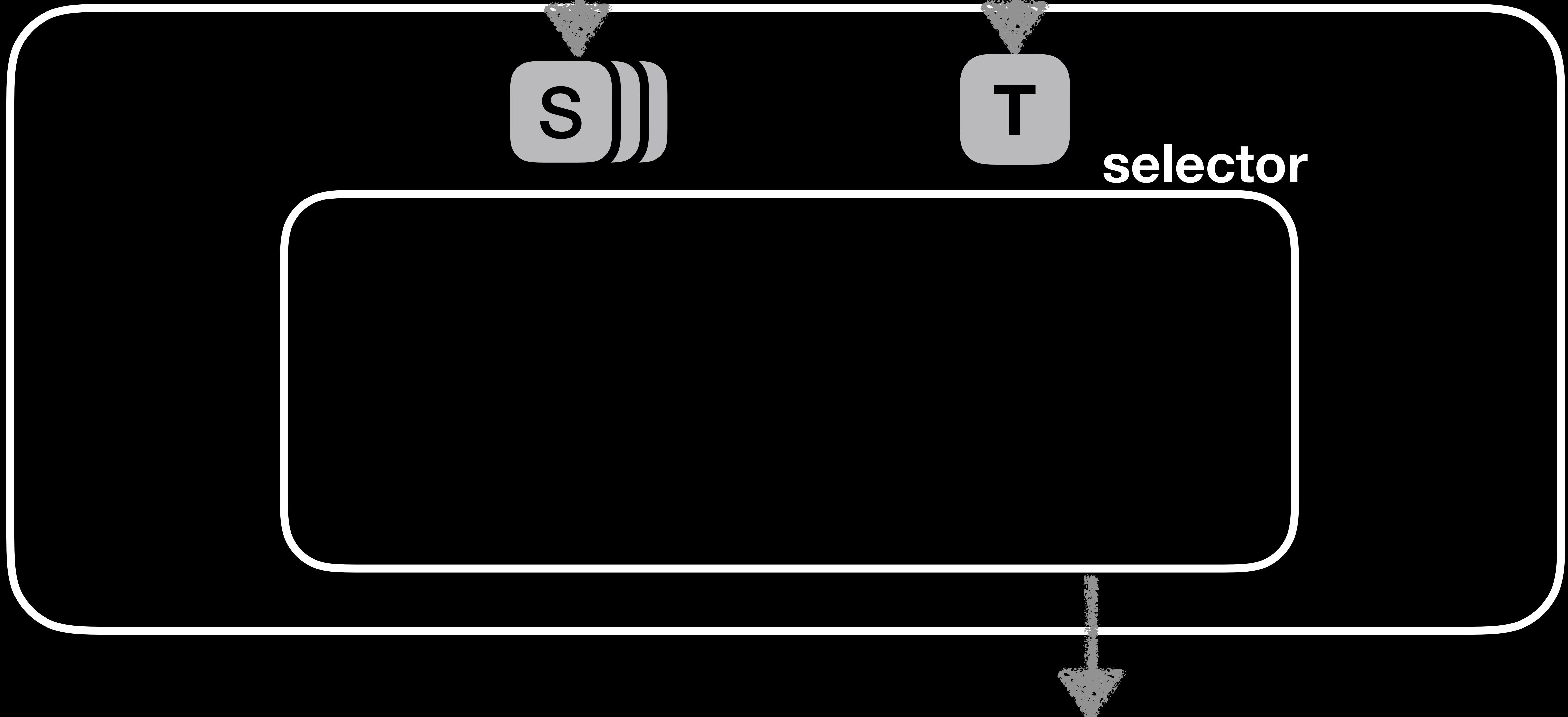
TransformFn

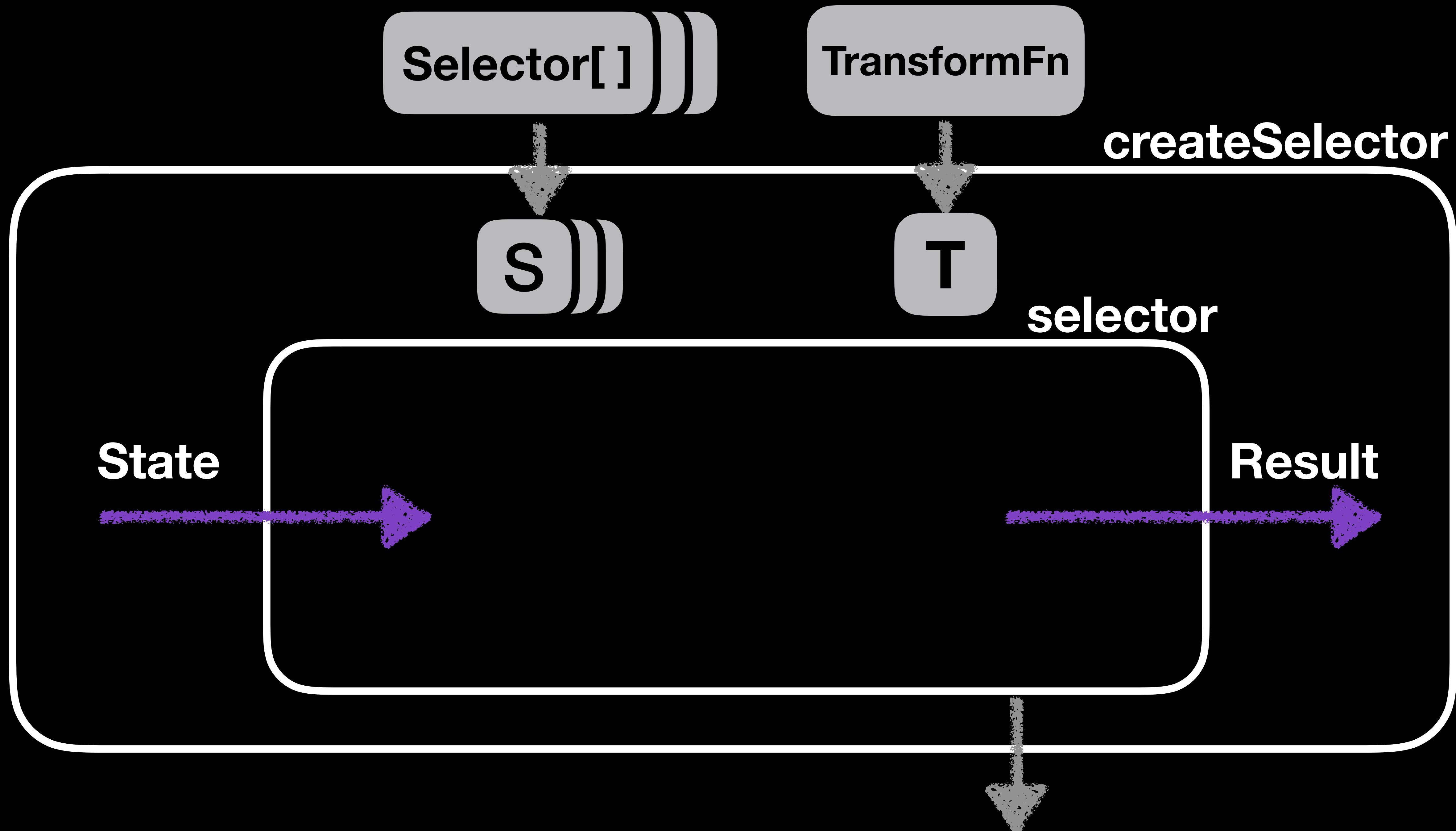
createSelector

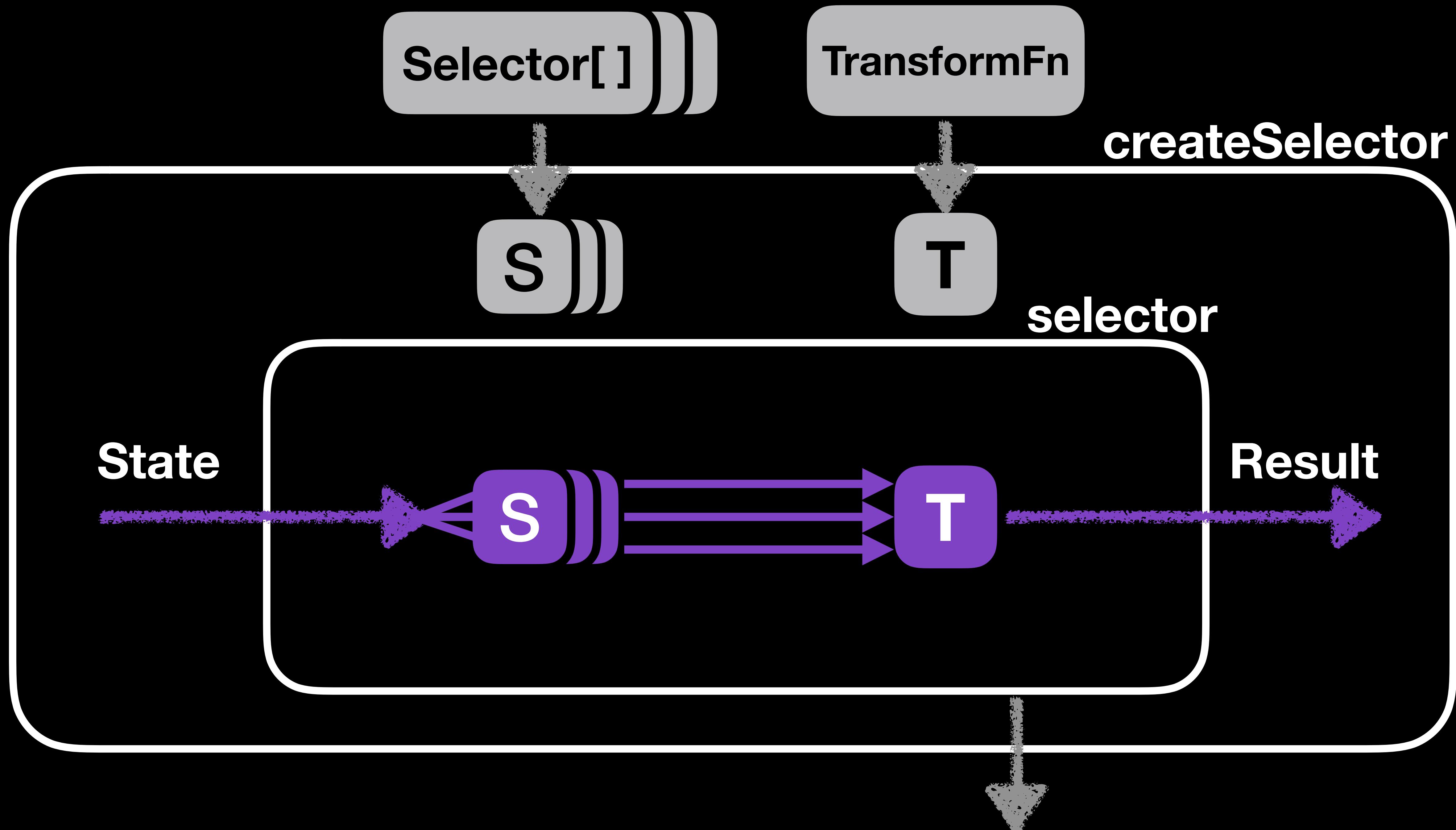
S

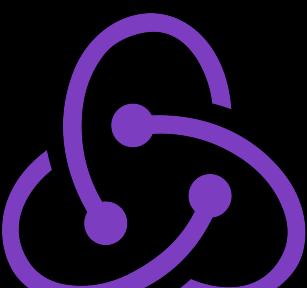
T

selector









createSelector: Combine + Transform

```
function createSelector(selectorFns: SelectorFn[], transformationFn: AnyFn): SelectorFn {  
}
```



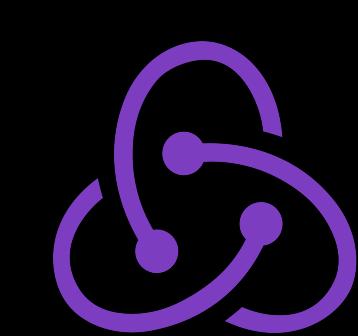
createSelector: Combine + Transform

```
function createSelector(selectorFns: SelectorFn[], transformationFn: AnyFn): SelectorFn {  
  function selector(state): any {  
    }  
    return selector;  
}
```



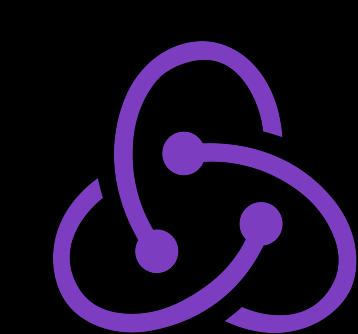
createSelector: Combine + Transform

```
function createSelector(selectorFns: SelectorFn[], transformationFn: AnyFn): SelectorFn {  
  function selector(state): any {  
    const selectorFnResults = selectorFns.map(selectorFn => selectorFn(state));  
  
  }  
  return selector;  
}
```



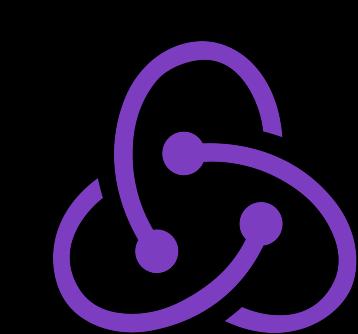
createSelector: Combine + Transform

```
function createSelector(selectorFns: SelectorFn[], transformationFn: AnyFn): SelectorFn {  
  function selector(state): any {  
    const selectorFnResults = selectorFns.map(selectorFn => selectorFn(state));  
    return transformationFn(...selectorFnResults);  
  }  
  return selector;  
}
```



createSelector: Memoize

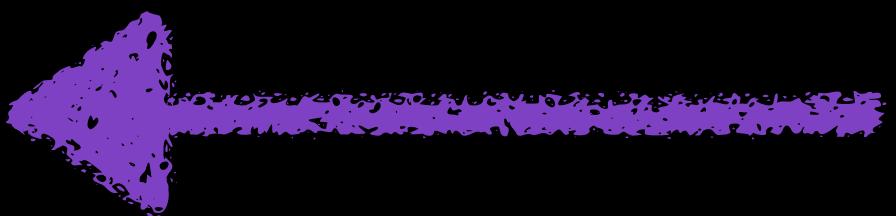
```
function expensiveCalculation(arg1: number, arg2: number): number {  
    // ...  
}  
  
expensiveCalculation(5, 10);
```



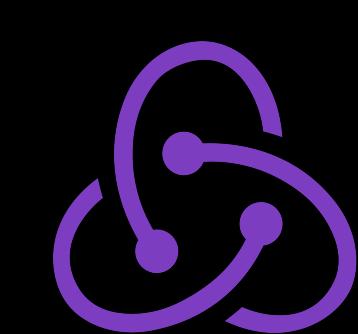
createSelector: Memoize

```
function expensiveCalculation(arg1: number, arg2: number): number {  
    // ...  
}
```

```
expensiveCalculation(5, 10);
```



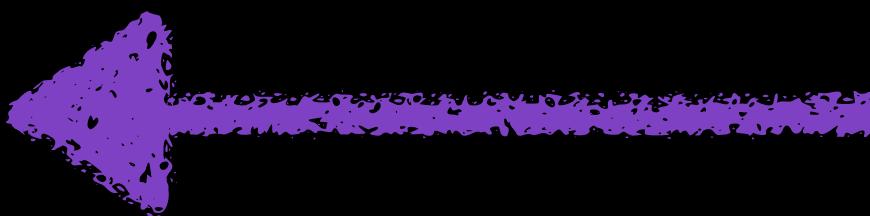
Calculates on every call



createSelector: Memoize

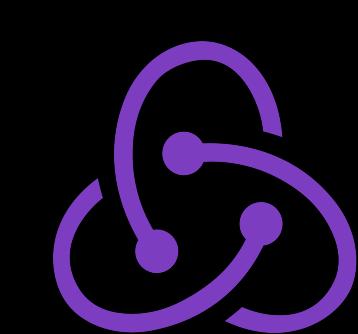
```
function expensiveCalculation(arg1: number, arg2: number): number {  
    // ...  
}
```

```
expensiveCalculation(5, 10);
```



Calculates on every call

```
const memoizedExpensiveCalculation = memoize(expensiveCalculation);
```



createSelector: Memoize

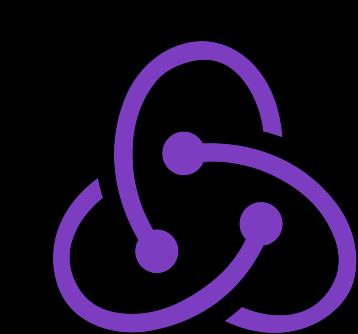
```
function expensiveCalculation(arg1: number, arg2: number): number {  
  // ...  
}
```

```
expensiveCalculation(5, 10);
```



Calculates on every call

```
const memoizedExpensiveCalculation = memoize(expensiveCalculation);  
memoizedExpensiveCalculation(5, 10);
```



createSelector: Memoize

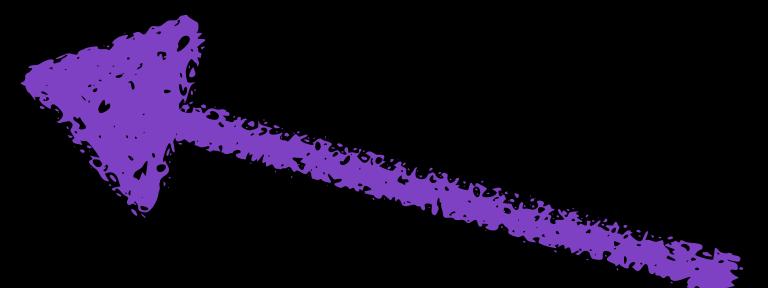
```
function expensiveCalculation(arg1: number, arg2: number): number {  
    // ...  
}
```

```
expensiveCalculation(5, 10);
```



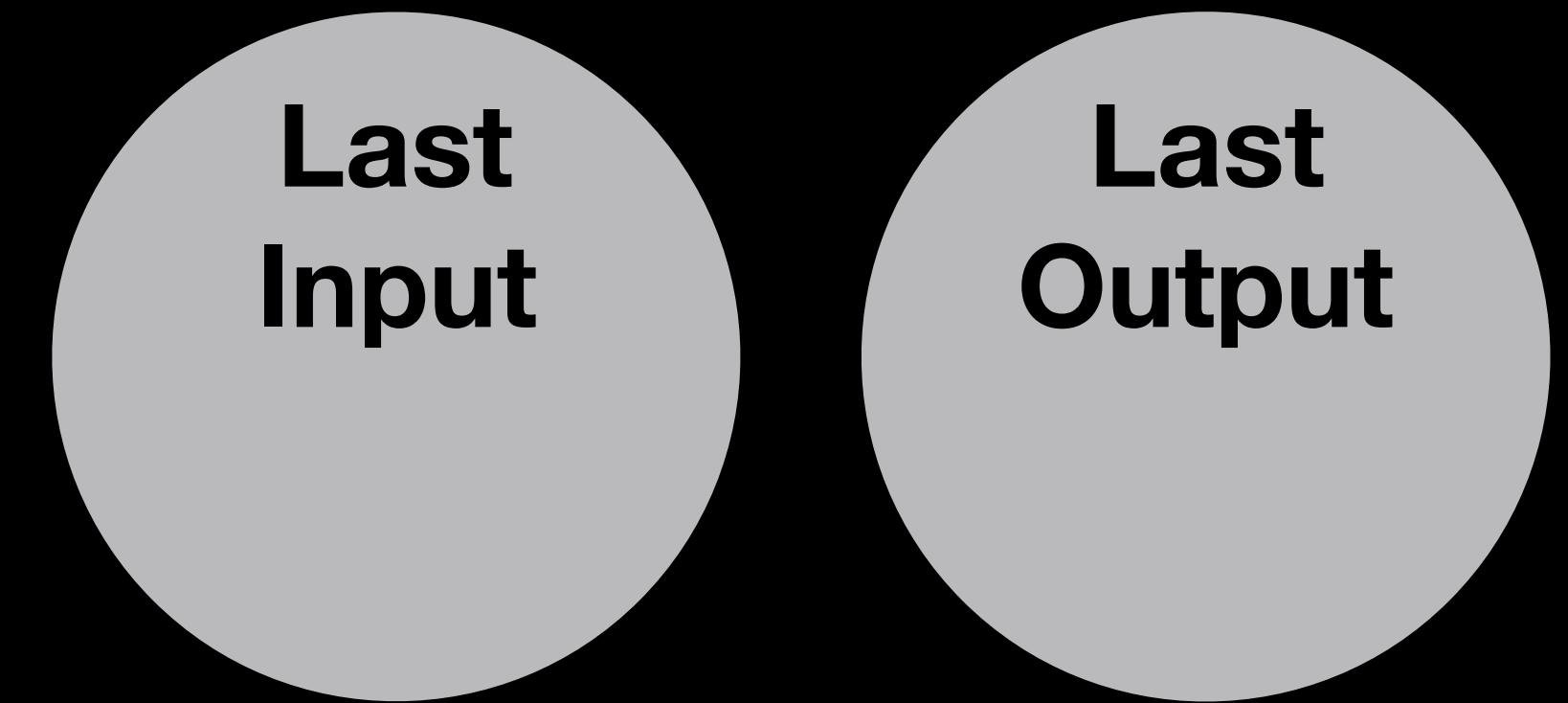
Calculates on every call

```
const memoizedExpensiveCalculation = memoize(expensiveCalculation);  
memoizedExpensiveCalculation(5, 10);
```



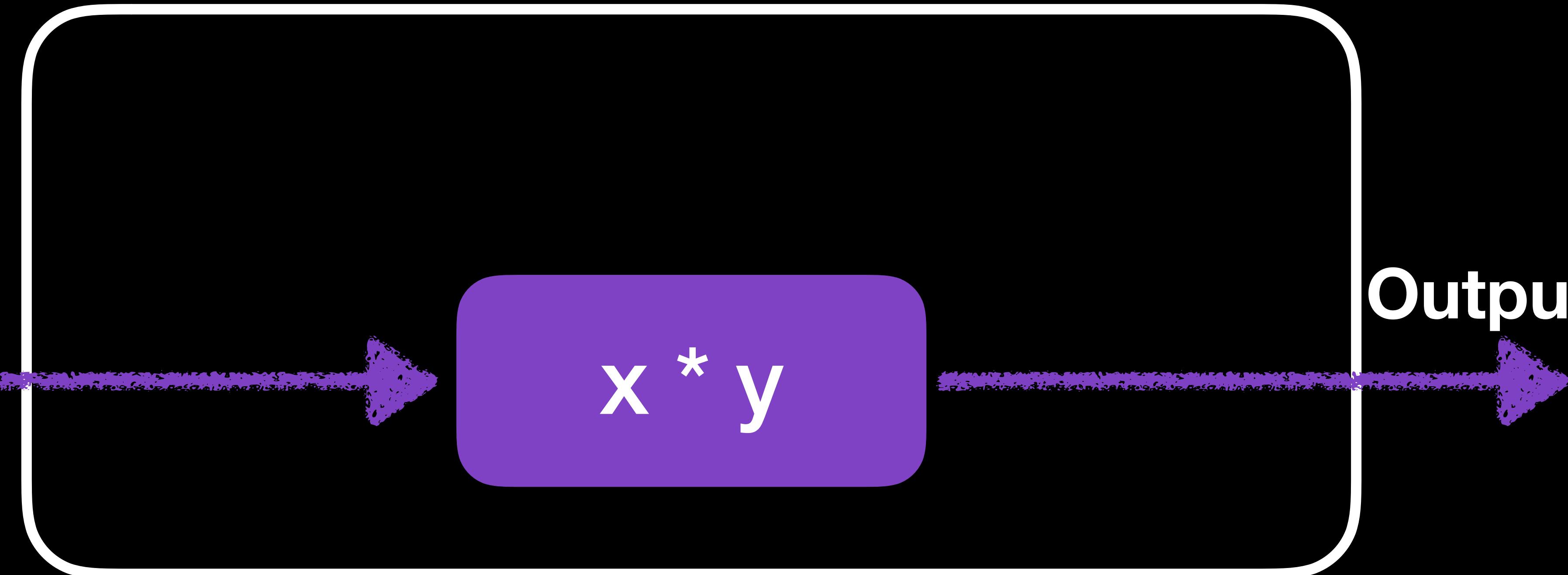
**Calculates if input
differs from last call**

Cache:

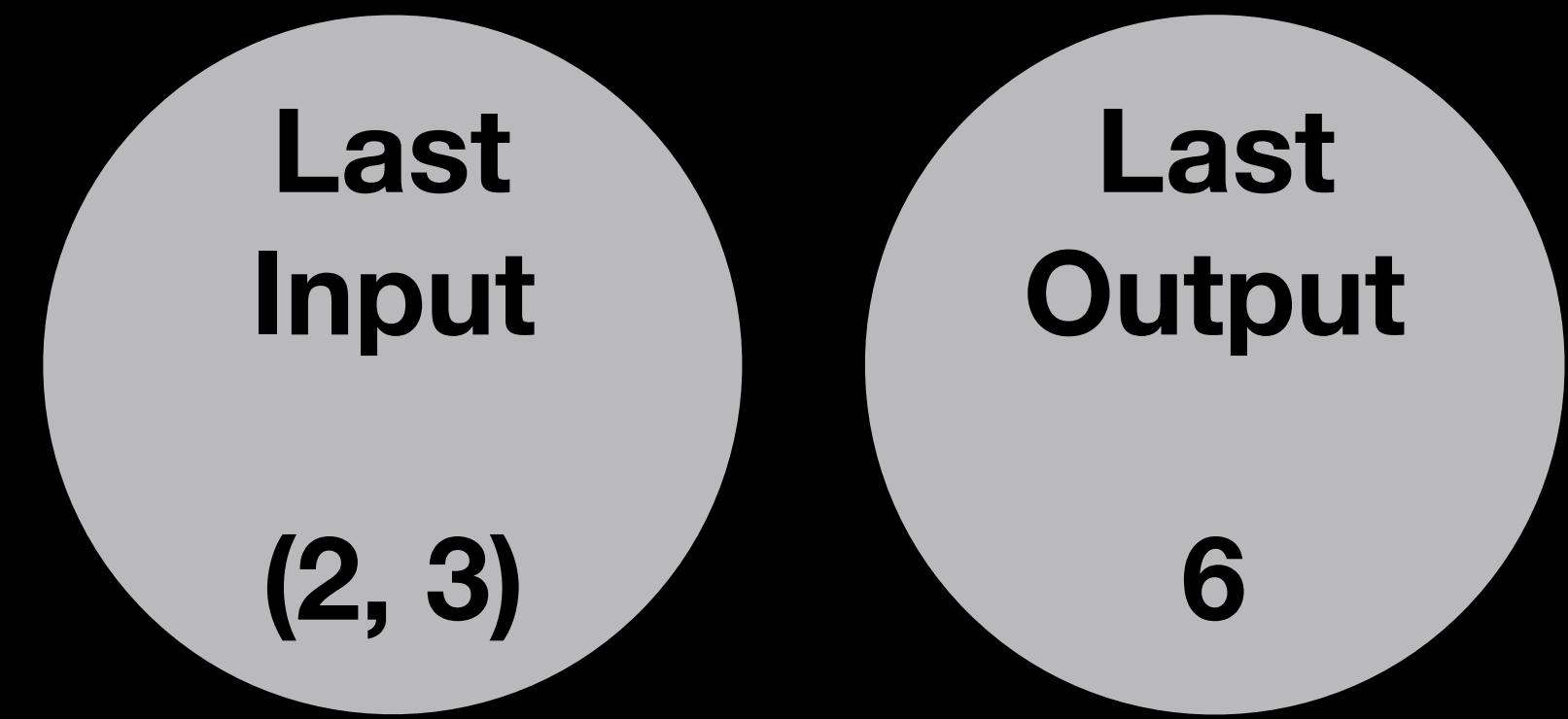


memoize

Input:



Cache:

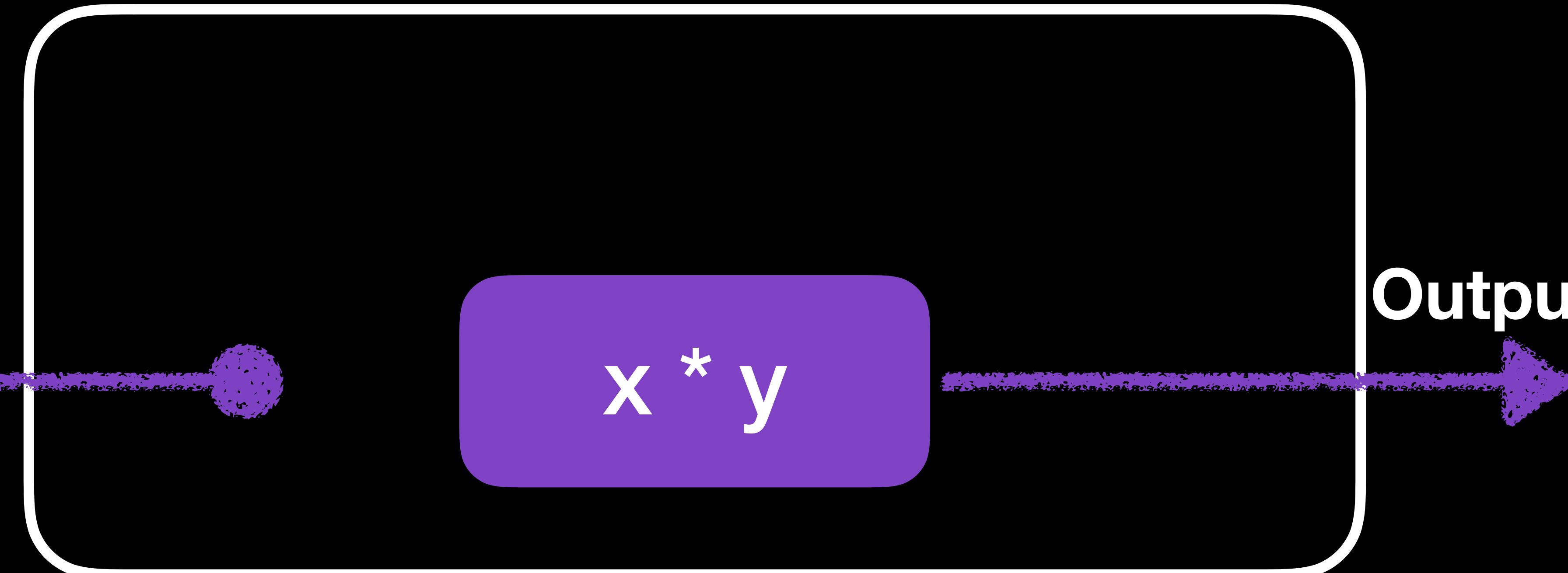


memoize

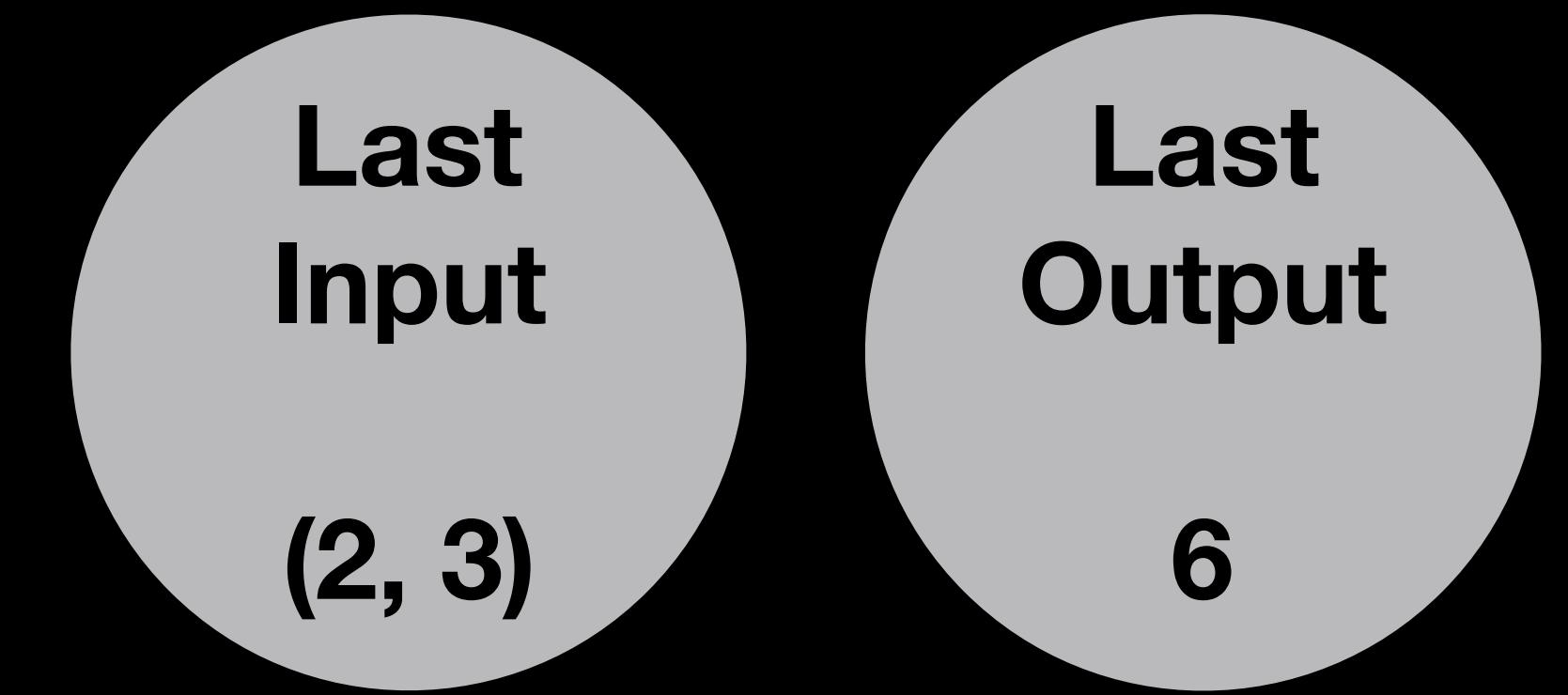
Input: (2, 3)

$x * y$

Output



Cache:



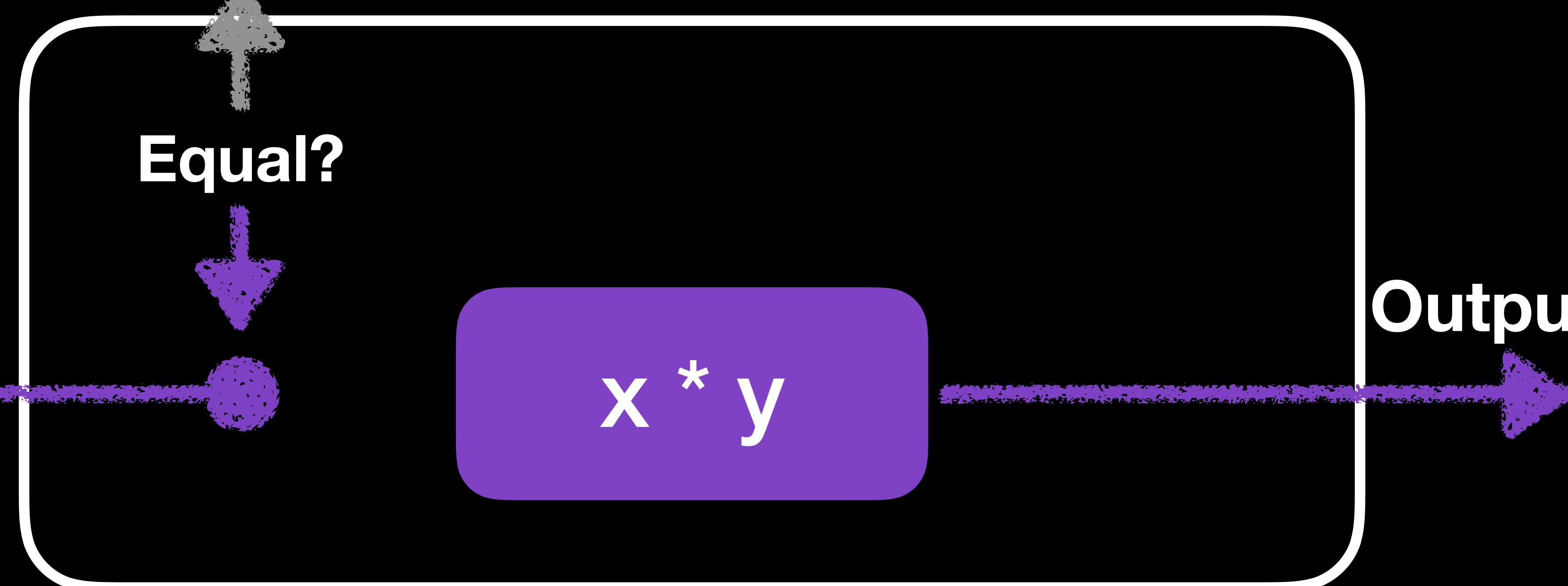
Input: (2, 3)

Equal?

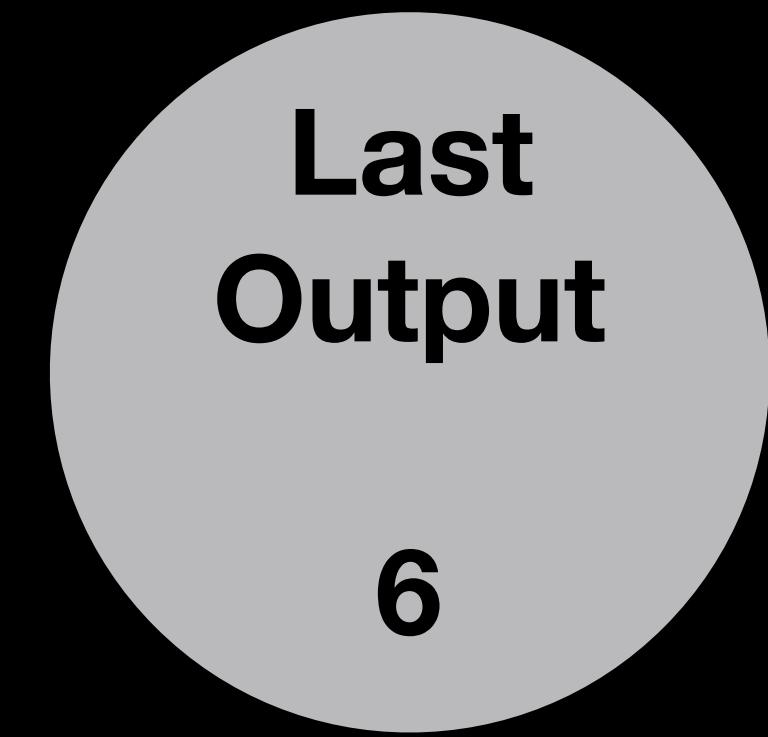
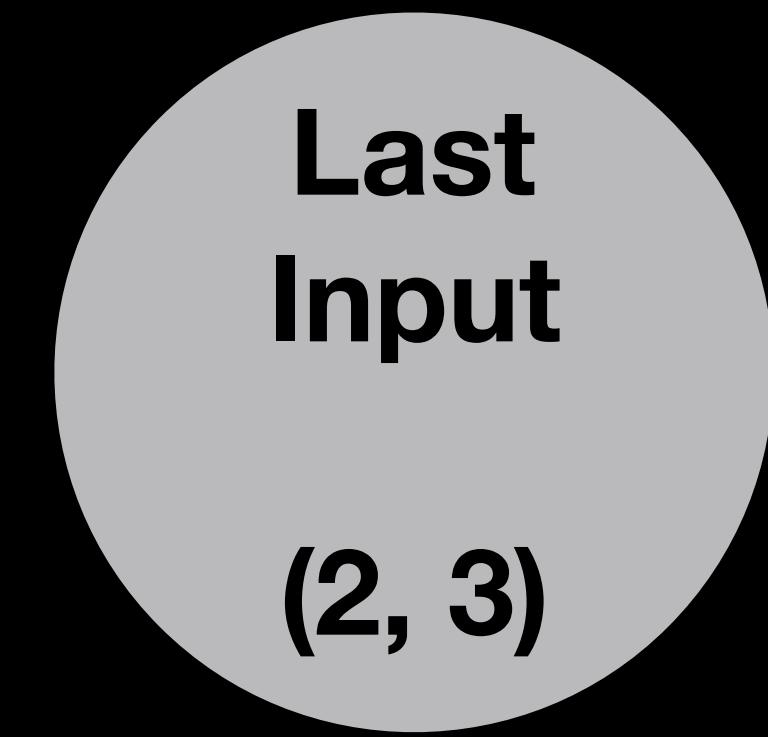
$x * y$

memoize

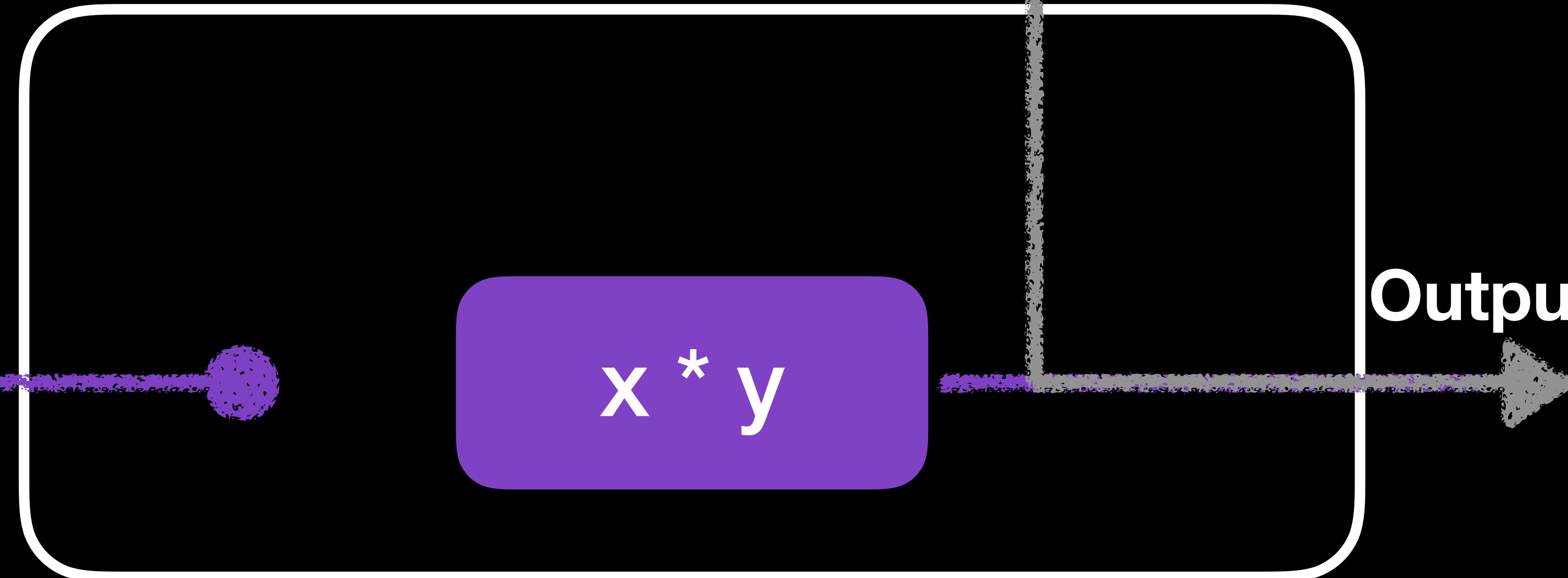
Output



Cache:

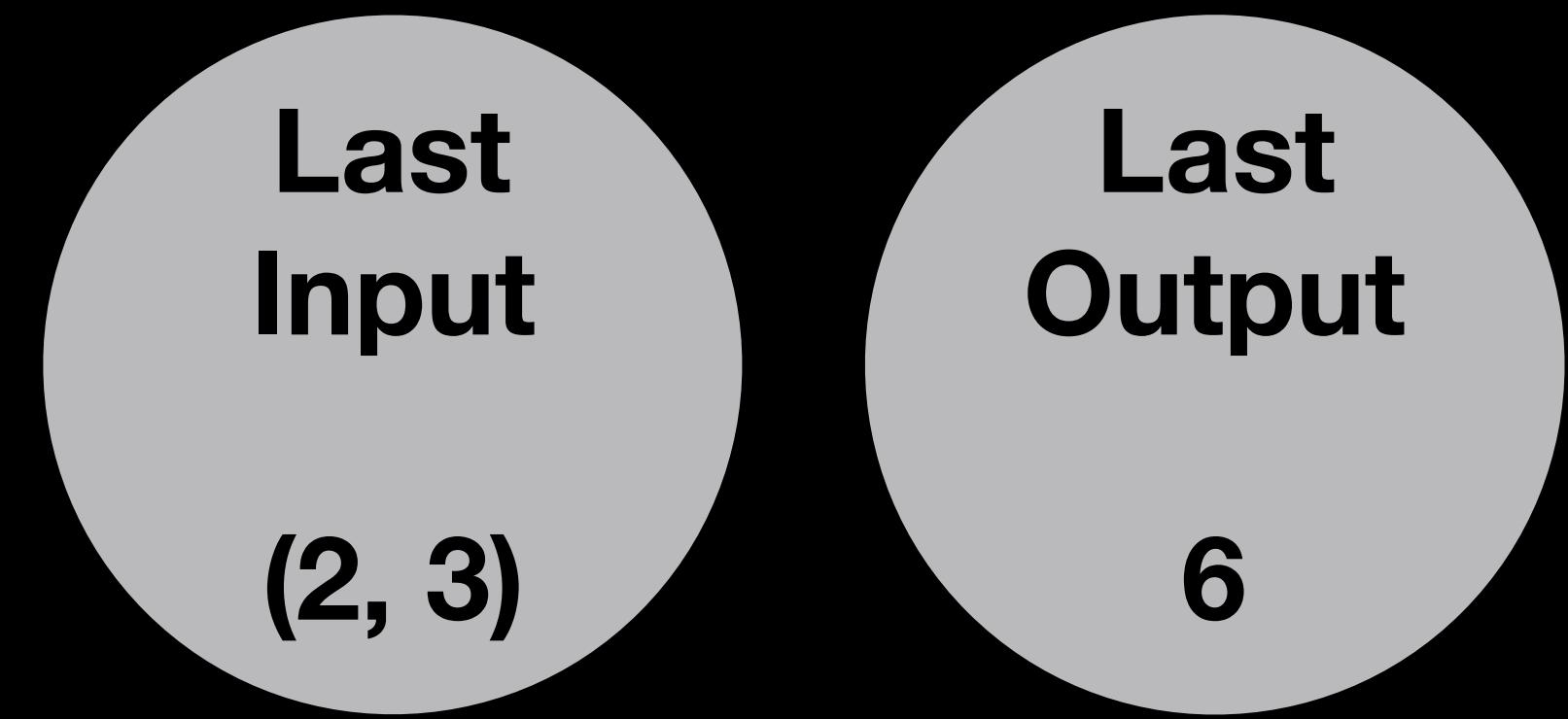


Input: (2, 3)



memoize

Cache:

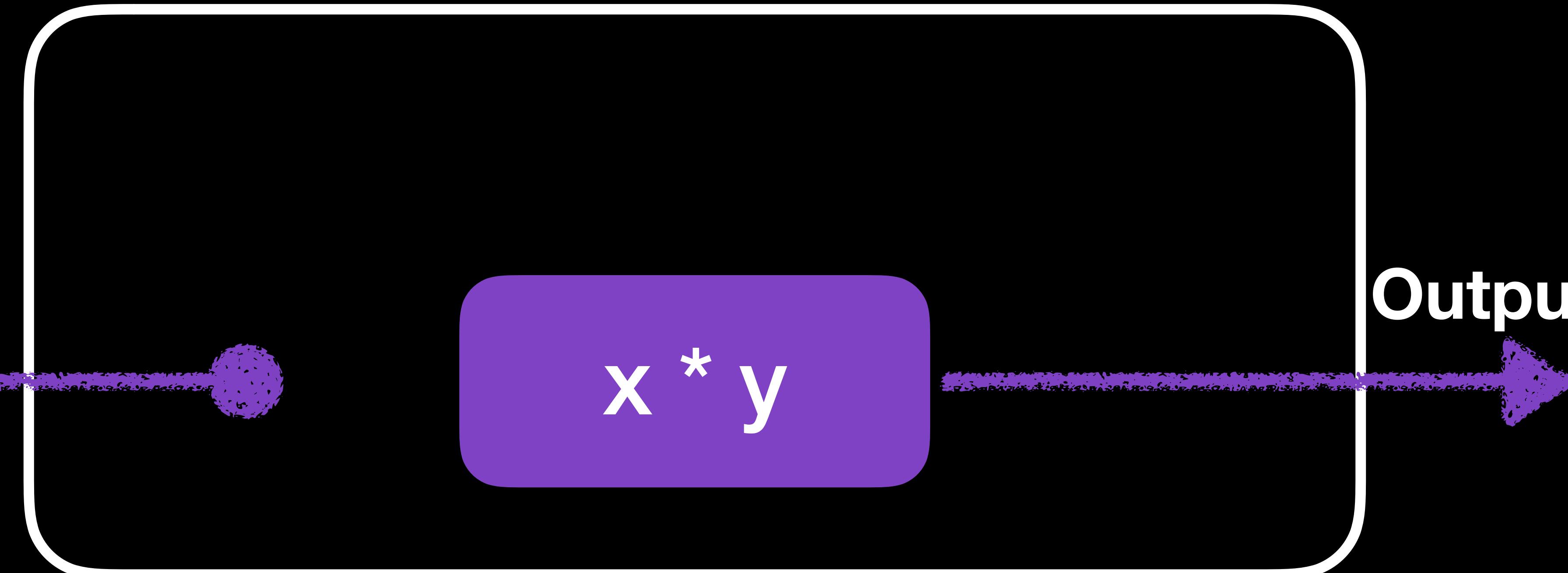


memoize

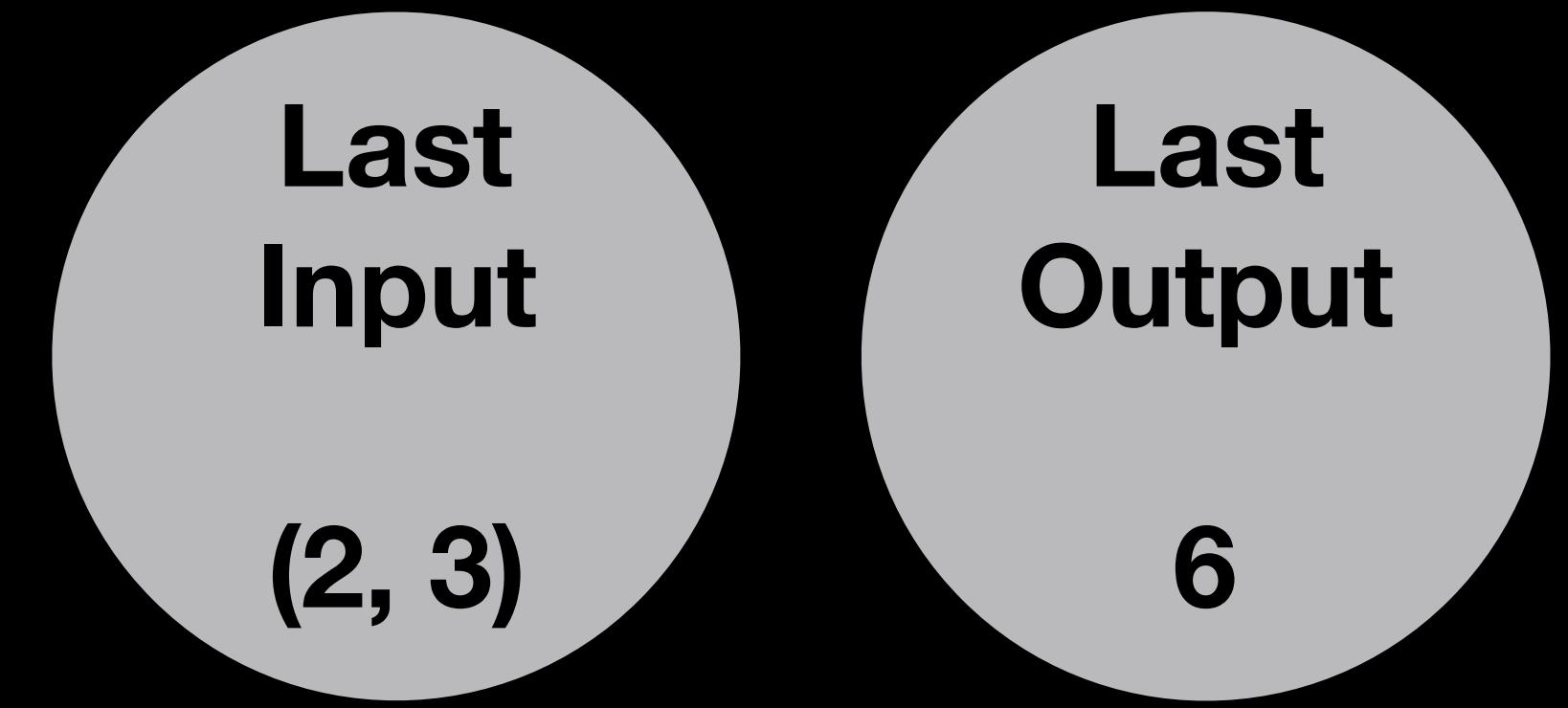
Input: (2, 4)

$x * y$

Output



Cache:



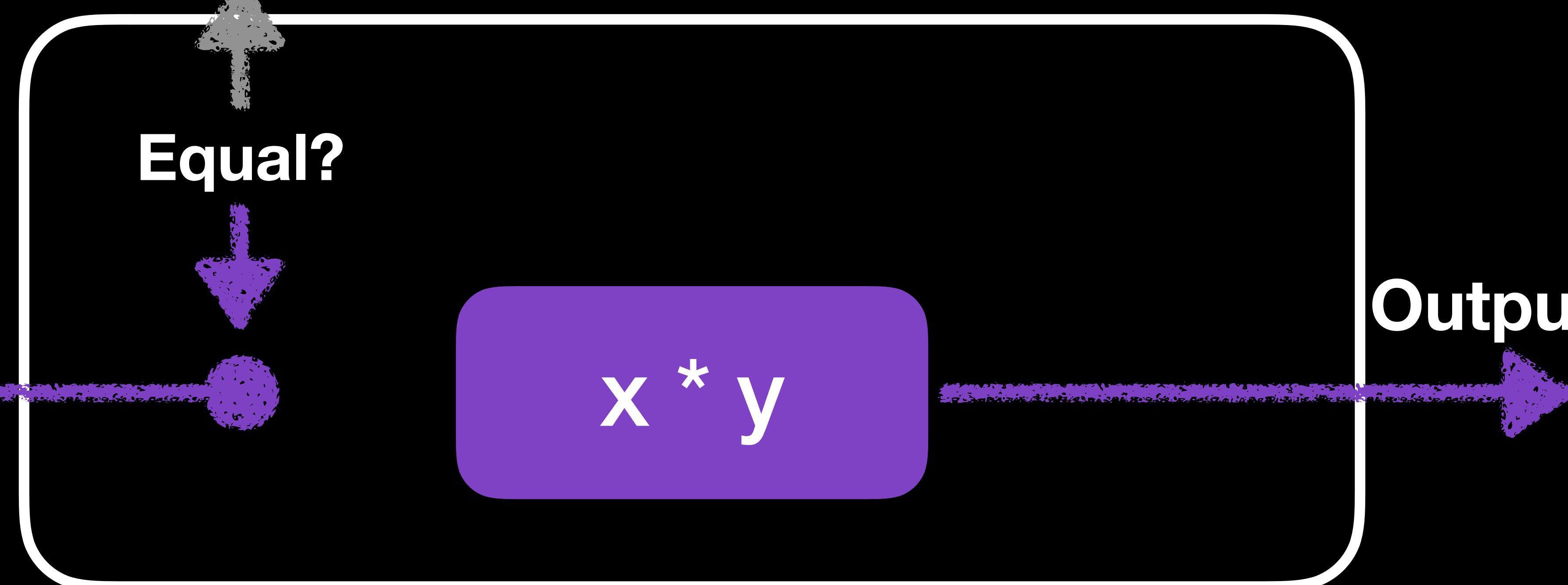
Input: (2, 4)

Equal?

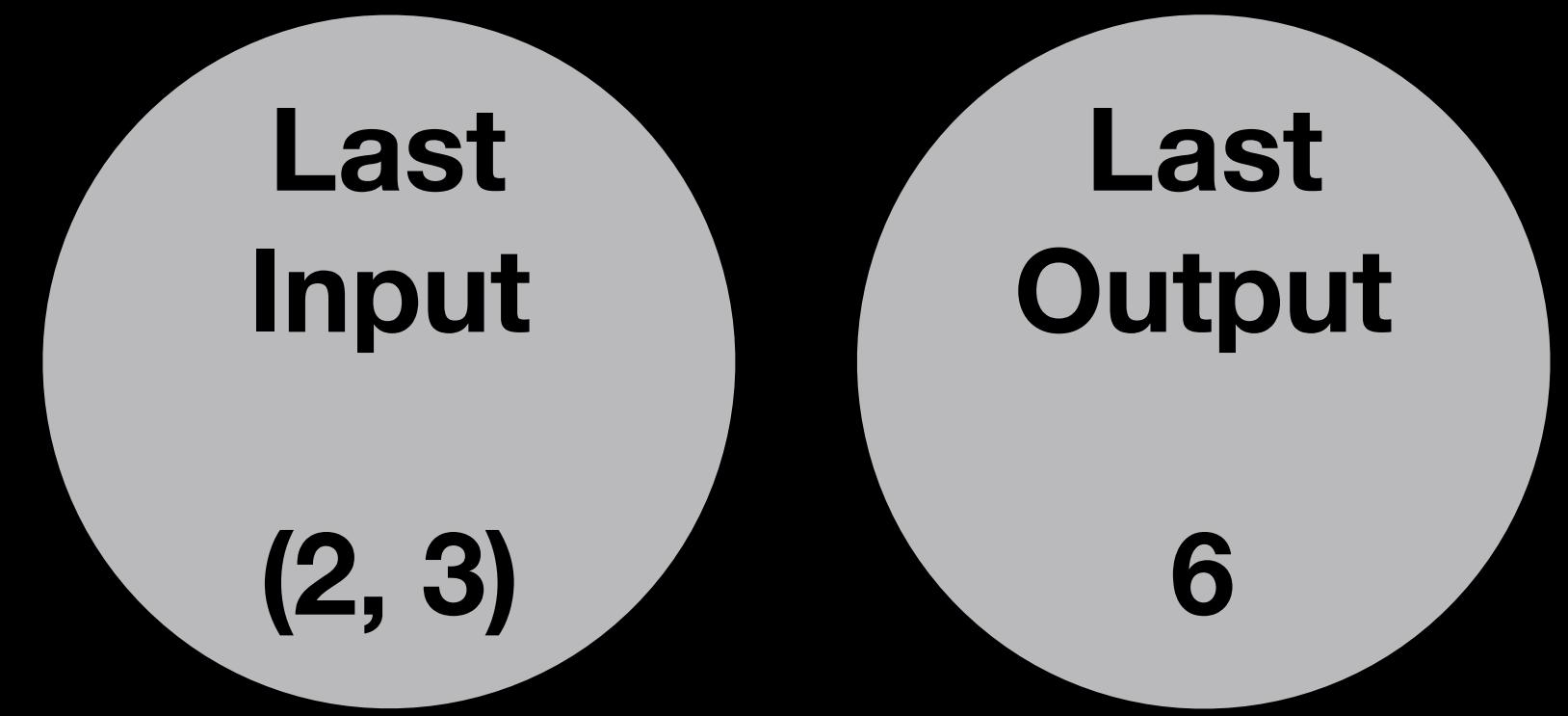
$x * y$

memoize

Output

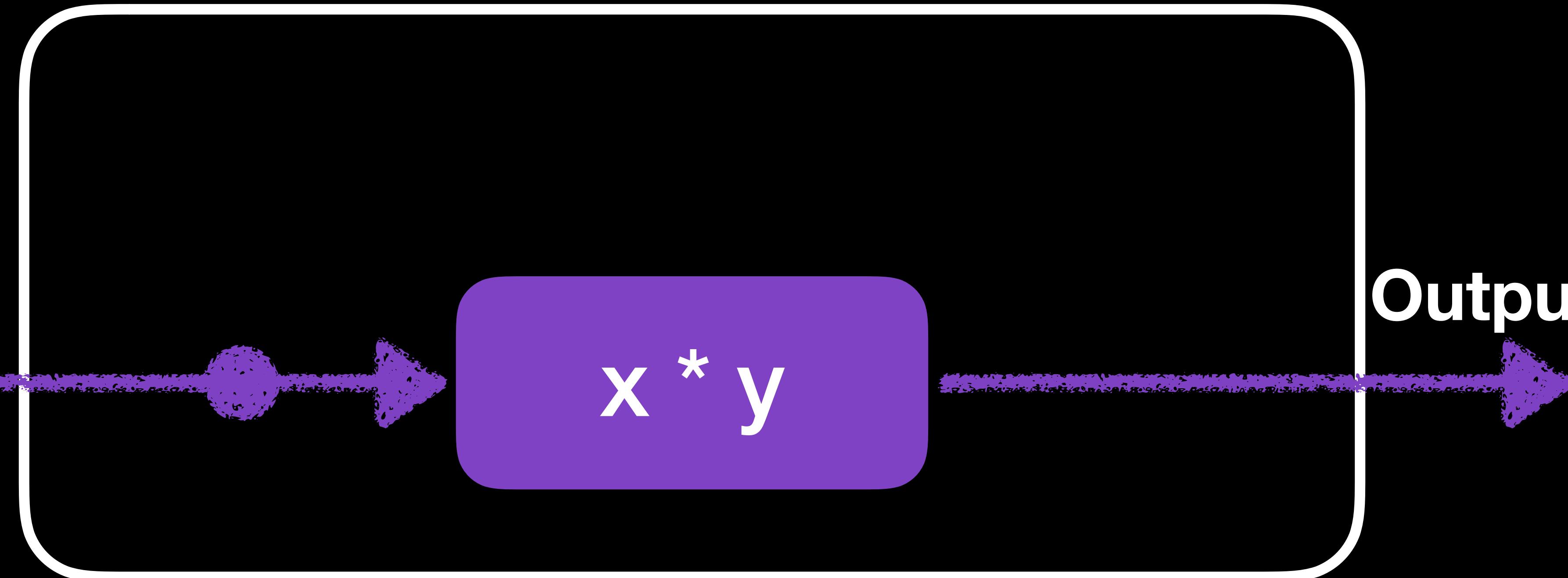


Cache:

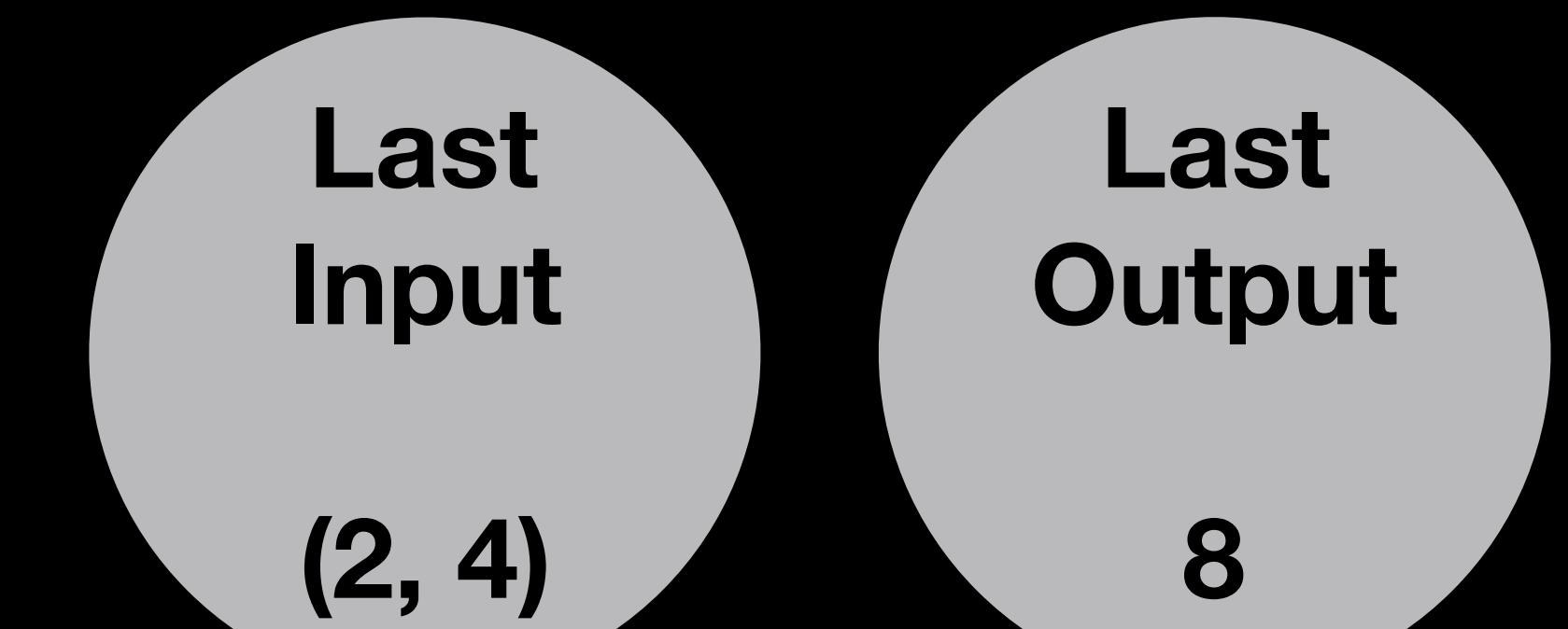


memoize

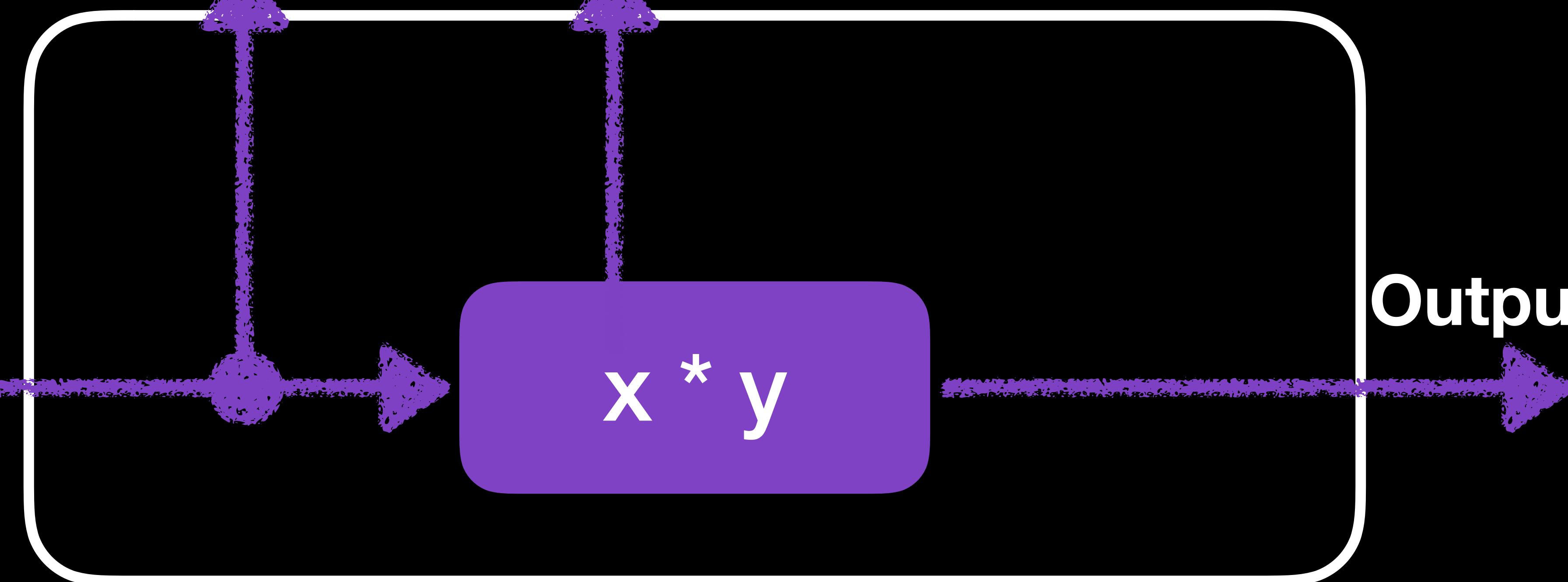
Input: (2, 4)



Cache:



Input: (2, 4)

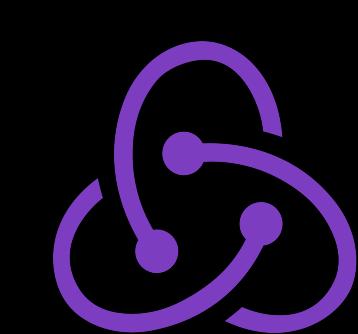


Output: 8



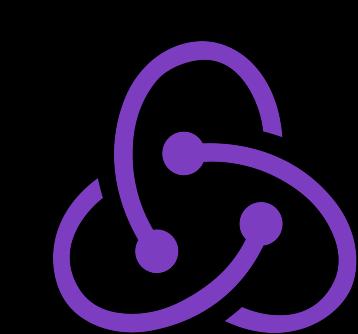
createSelector: Memoize

```
function memoize(originalFn: AnyFn): AnyFn {  
  let lastInput: any[] = null;  
  let lastOutput: any = null;  
  function memoizedFn(...args: any[]) {  
    if (!areInputsEqual(lastInput, args)) {  
      lastOutput = originalFn(...args);  
      lastInput = args;  
    }  
    return lastOutput;  
  }  
  return memoizedFn;  
}
```



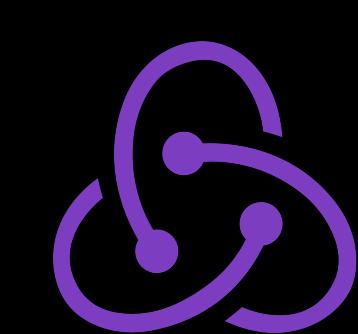
createSelector: Memoize

```
function createSelector(selectorFns: SelectorFn[], transformationFn: AnyFn): SelectorFn {  
  
  function selector(state) {  
    const selectorFnResults = selectorFns.map(selectorFn => selectorFn(state));  
    return transformationFn(...selectorFnResults);  
  }  
  return selector;  
}
```



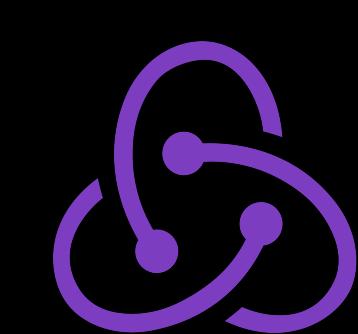
createSelector: Memoize

```
function createSelector(selectorFns: SelectorFn[], transformationFn: AnyFn): SelectorFn {  
  const memoizedTransformationFn = memoize(transformationFn);  
  
  function selector(state) {  
    const selectorFnResults = selectorFns.map(selectorFn => selectorFn(state));  
    return transformationFn(...selectorFnResults);  
  }  
  return selector;  
}
```



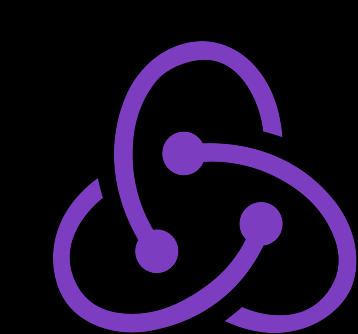
createSelector: Memoize

```
function createSelector(selectorFns: SelectorFn[], transformationFn: AnyFn): SelectorFn {  
  const memoizedTransformationFn = memoize(transformationFn);  
  
  function selector(state) {  
    const selectorFnResults = selectorFns.map(selectorFn => selectorFn(state));  
    return transformationFn(...selectorFnResults);  
  }  
  return selector;  
}
```



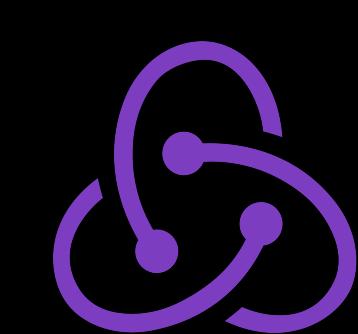
createSelector: Memoize

```
function createSelector(selectorFns: SelectorFn[], transformationFn: AnyFn): SelectorFn {  
  const memoizedTransformationFn = memoize(transformationFn);  
  
  function selector(state) {  
    const selectorFnResults = selectorFns.map(selectorFn => selectorFn(state));  
    return memoizedTransformationFn(...selectorFnResults);  
  }  
  return selector;  
}
```



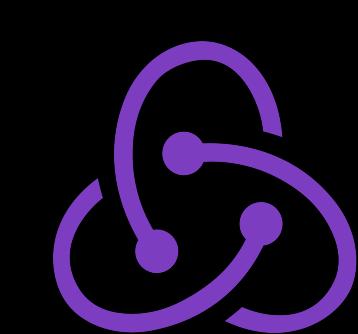
createSelector: Memoize

```
function createSelector(selectorFns: SelectorFn[], transformationFn: AnyFn): SelectorFn {  
  const memoizedTransformationFn = memoize(transformationFn);  
  
  function selector(state) {  
    const selectorFnResults = selectorFns.map(selectorFn => selectorFn(state));  
    return memoizedTransformationFn(...selectorFnResults);  
  }  
  return selector;  
}
```



createSelector: Memoize

```
function createSelector(selectorFns: SelectorFn[], transformationFn: AnyFn): SelectorFn {  
  const memoizedTransformationFn = memoize(transformationFn);  
  
  function selector(state) {  
    const selectorFnResults = selectorFns.map(selectorFn => selectorFn(state));  
    return memoizedTransformationFn(...selectorFnResults);  
  }  
  return memoize(selector);  
}
```

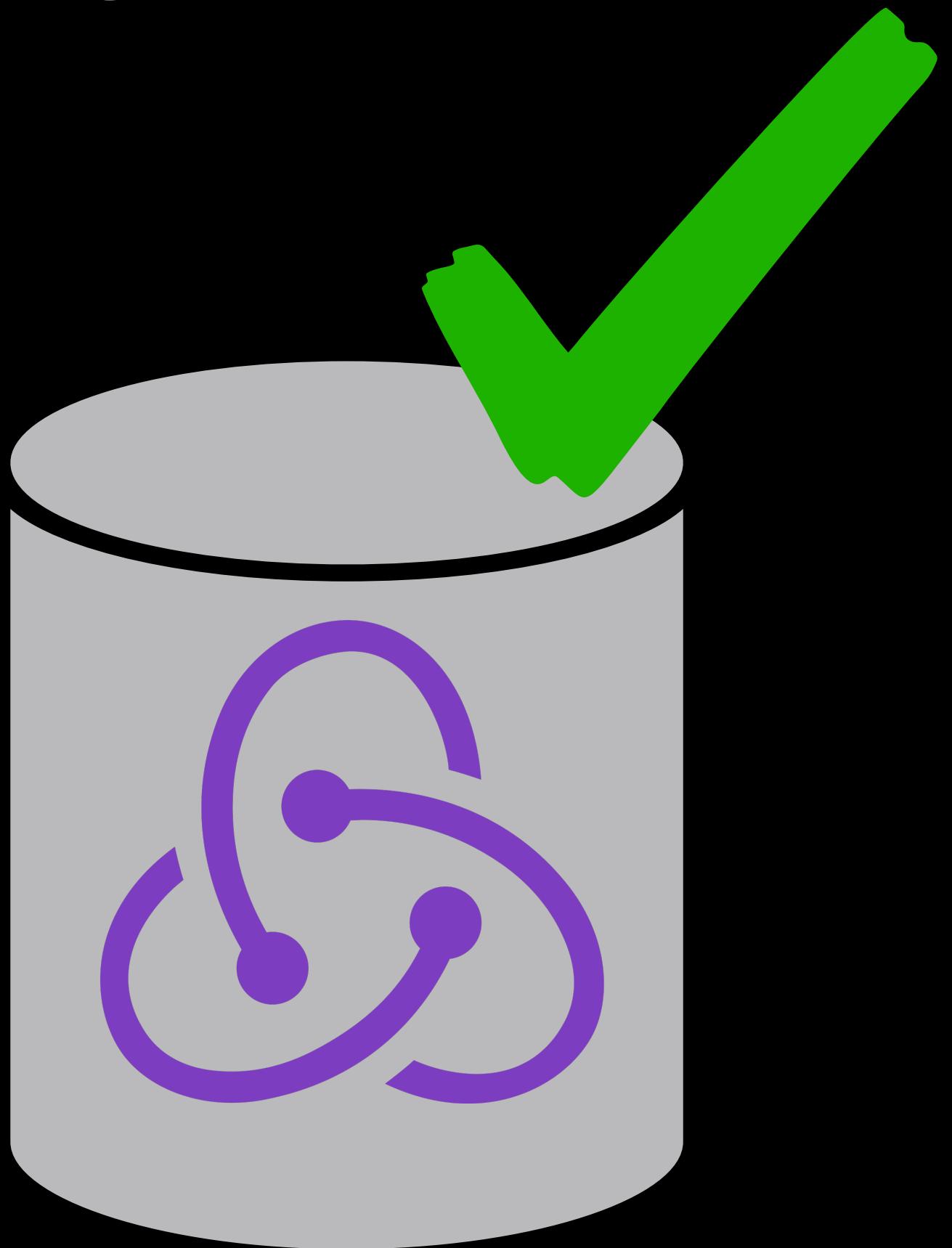


createSelector: Memoize

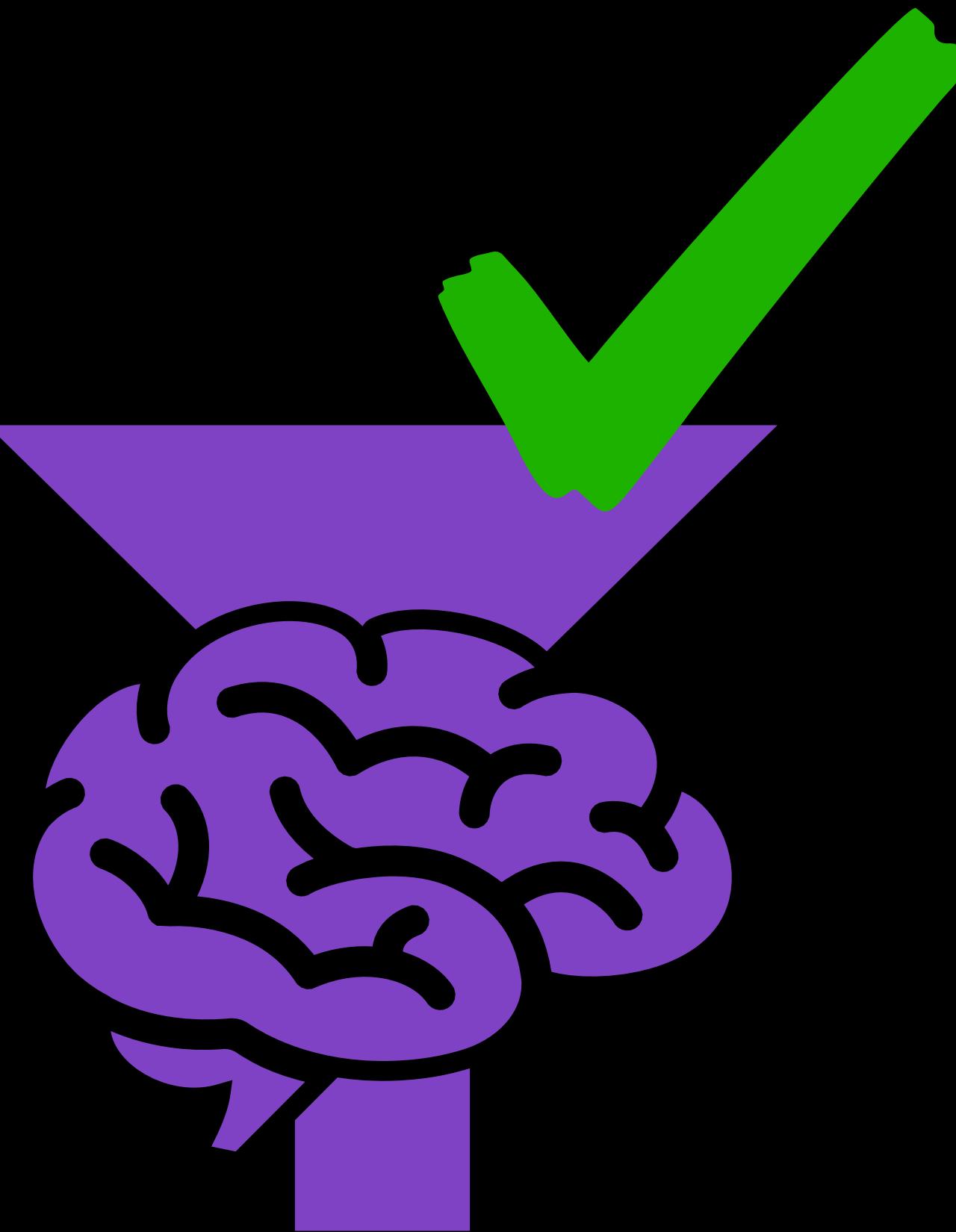
```
function createSelector(selectorFns: SelectorFn[], transformationFn: AnyFn): SelectorFn {  
  const memoizedTransformationFn = memoize(transformationFn);  
  
  function selector(state) {  
    const selectorFnResults = selectorFns.map(selectorFn => selectorFn(state));  
    return memoizedTransformationFn(...selectorFnResults);  
  }  
  return memoize(selector);  
}
```



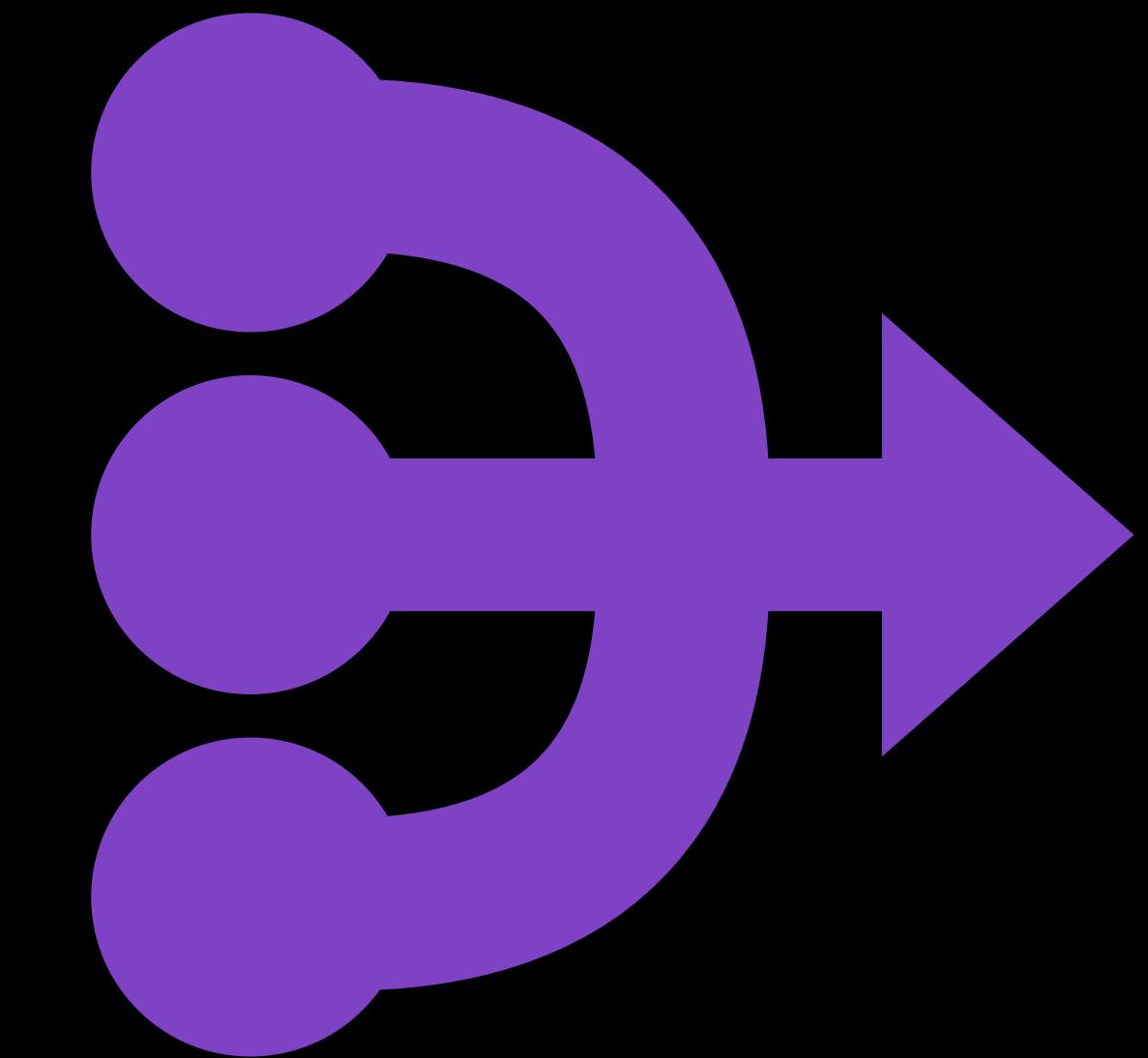
Agenda



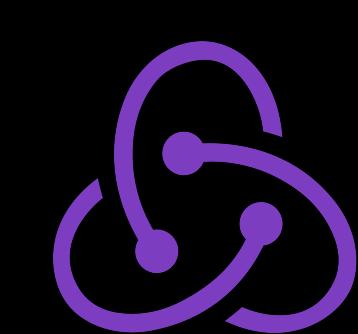
Store



Memoized
Selector

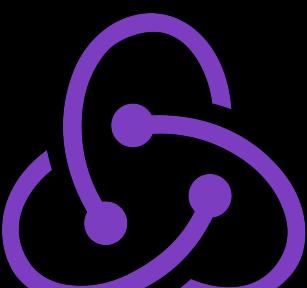


Combine
Reducers



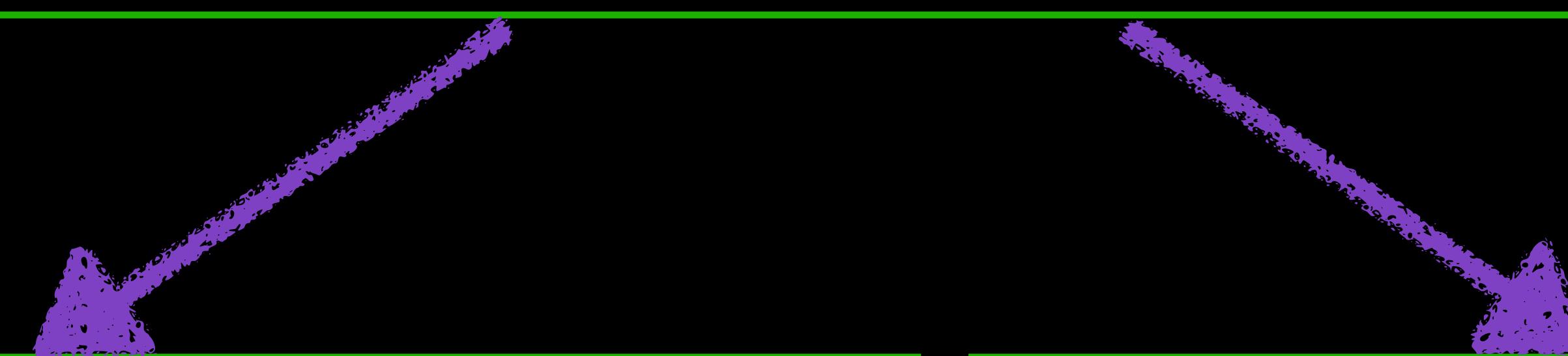
Combine Reducers: Motivation

```
export function donkeyKongAppReducer(state = initialState, action: any): DonkeyKongAppState {
  switch (action.type) {
    case 'ADD_ITEM':
      return {
        ...state,
        cart: state.cart.concat(action.price)
      };
    case 'TOGGLE_DARK_MODE':
      return {
        ...state,
        darkMode: !state.darkMode
      };
    default:
      return state;
  }
}
```



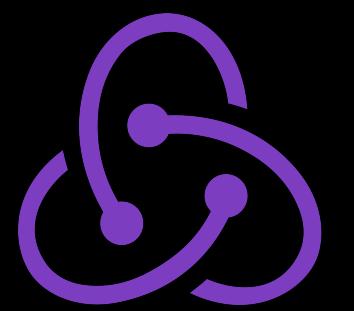
Combine Reducers: Motivation

```
export function donkeyKongAppReducer(state = initialState, action: any): DonkeyKongAppState {
  switch (action.type) {
    case 'ADD_ITEM':
      return {
        ...state,
        cart: state.cart.concat(action.price)
      };
    case 'TOGGLE_DARK_MODE':
      return {
        ...state,
        darkMode: !state.darkMode
      };
    default:
      return state;
  }
}
```



```
export function darkModeReducer(state: boolean, action: ToggleDarkMode) {
  switch (action.type) {
    case 'TOGGLE_DARK_MODE':
      return state = !state;
    default:
      return state;
  }
}
```

```
export function cartReducer(state: number[], action: AddItem) {
  switch (action.type) {
    case 'ADD_ITEM':
      return state.concat(action.price);
    default:
      return state;
  }
}
```



Combine Reducers



I'll reduce the "darkMode" section of the state tree!

combineReducers()

I'll reduce the "cart" section of the state tree!



StoreModule.forRoot(reducer)

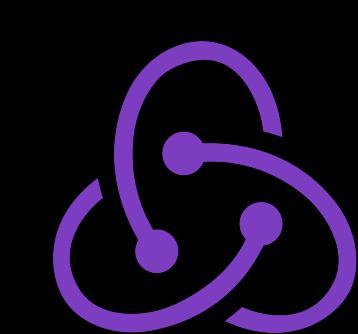
constructor(reducer)





Combine Reducers

```
function combineReducers<S>(  
  reducerMap: ReducerMap,  
  initialState: S  
) : ReducerFunction<S>
```



Combine Reducers

```
function combineReducers<S>(  
  reducerMap: ReducerMap,  
  initialState: S  
) : ReducerFunction<S>
```

```
export const donkeyKongAppReducer = combineReducers(  
{  
  darkMode: darkModeReducer,  
  cart: cartReducer,  
  taxRate: taxRateReducer  
,  
  initialState  
});
```

Key → Reducer

```
{  
  darkMode: darkModeReducer,  
  cart: cartReducer,  
  taxRate: taxRateReducer  
};
```

combineReducers

Key → Reducer

Full State Reducer

Prev. State

key

state[key]

key

...

Reducer

...

Action

state

New State

```
interface ReducerMap {  
  [key: string]: ReducerFunction<any>;  
}
```

```
const appReducers: ReducersMap = {  
  darkMode: darkModeReducer,  
  cart: cartReducer,  
  taxRate: taxRateReducer  
};
```

```
interface ReducerMap {  
  [key: string]: ReducerFunction<any>;  
}  
  
const appReducers: ReducersMap = {  
  darkMode: darkModeReducer,  
  cart: cartReducer,  
  taxRate: taxRateReducer  
};
```

```
function combineReducers<S>( reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {  
}
```

```
interface ReducerMap {  
  [key: string]: ReducerFunction<any>;  
}  
  
const appReducers: ReducersMap = {  
  darkMode: darkModeReducer,  
  cart: cartReducer,  
  taxRate: taxRateReducer  
};  
  
function combineReducers<S>(reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {  
  function reducer(previousState = initialState, action: Action): S {  
    }  
    return reducer;  
}
```

```
interface ReducerMap {  
  [key: string]: ReducerFunction<any>;  
}  
  
const appReducers: ReducersMap = {  
  darkMode: darkModeReducer,  
  cart: cartReducer,  
  taxRate: taxRateReducer  
};  
  
function combineReducers<S>(reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {  
  function reducer(previousState = initialState, action: Action): S {  
    let newState = previousState;  
  
    for (const key in reducerMap) {  
      if (reducerMap[key].handleAction) {  
        const result = reducerMap[key].handleAction(newState, action);  
        if (result !== undefined) {  
          newState = result;  
        }  
      }  
    }  
  
    return newState;  
  }  
  
  return reducer;  
}
```

```
interface ReducerMap {  
  [key: string]: ReducerFunction<any>;  
}  
  
const appReducers: ReducersMap = {  
  darkMode: darkModeReducer,  
  cart: cartReducer,  
  taxRate: taxRateReducer  
};
```

```
function combineReducers<S>(reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {  
  function reducer(previousState = initialState, action: Action): S {  
    const reducerMapKeys: string[] = Object.keys(reducerMap);  
  
    let newState = initialState;  
  
    for (let i = 0; i < reducerMapKeys.length; i++) {  
      const key = reducerMapKeys[i];  
      const reducerFunction = reducerMap[key];  
      const actionType = action.type;  
  
      if (reducerFunction === undefined || actionType === undefined) {  
        continue;  
      }  
  
      const result = reducerFunction(previousState, action);  
      if (result !== undefined) {  
        newState = result;  
      }  
    }  
  
    return newState;  
  }  
  
  return reducer;  
}
```

```
interface ReducerMap {  
  [key: string]: ReducerFunction<any>;  
}  
  
const appReducers: ReducersMap = {  
  darkMode: darkModeReducer,  
  cart: cartReducer,  
  taxRate: taxRateReducer  
};
```

```
function combineReducers<S>(reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {  
  function reducer(previousState = initialState, action: Action): S {  
    const reducerMapKeys: string[] = Object.keys(reducerMap);  
  
    const newState: S = reducerMapKeys.reduce(  
      (state, key) => {  
        if (reducerMap[key]) {  
          state = reducerMap[key](state, action);  
        }  
        return state;  
      }, initialState  
    );  
  
    return newState;  
  }  
  return reducer;  
}
```

```
interface ReducerMap {  
  [key: string]: ReducerFunction<any>;  
}  
  
function combineReducers<S>(reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {  
  function reducer(previousState = initialState, action: Action): S {  
    const reducerMapKeys: string[] = Object.keys(reducerMap);  
  
    const newState: S = reducerMapKeys.reduce(  
      (accumulatedState: S, key: string) => {  
          
      },  
  
    );  
    return newState;  
  }  
  return reducer;  
}
```

```
const appReducers: ReducersMap = {  
  darkMode: darkModeReducer,  
  cart: cartReducer,  
  taxRate: taxRateReducer  
};
```

```
interface ReducerMap {  
  [key: string]: ReducerFunction<any>;  
}  
  
function combineReducers<S>(reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {  
  function reducer(previousState = initialState, action: Action): S {  
    const reducerMapKeys: string[] = Object.keys(reducerMap);  
  
    const newState: S = reducerMapKeys.reduce(  
      (accumulatedState: S, key: string) => {  
        const substateReducer: ReducerFunction<any> = reducerMap[key];  
  
        },  
      );  
    return newState;  
  }  
  return reducer;  
}
```

```
const appReducers: ReducersMap = {  
  darkMode: darkModeReducer,  
  cart: cartReducer,  
  taxRate: taxRateReducer  
};
```

```
interface ReducerMap {  
  [key: string]: ReducerFunction<any>;  
}  
  
const appReducers: ReducersMap = {  
  darkMode: darkModeReducer,  
  cart: cartReducer,  
  taxRate: taxRateReducer  
};  
  
function combineReducers<S>(reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {  
  function reducer(previousState = initialState, action: Action): S {  
    const reducerMapKeys: string[] = Object.keys(reducerMap);  
  
    const newState: S = reducerMapKeys.reduce(  
      (accumulatedState: S, key: string) => {  
        const substateReducer: ReducerFunction<any> = reducerMap[key];  
        const previousSubstate: any = previousState[key];  
  
        },  
      );  
    return newState;  
  }  
  return reducer;  
}
```

```
interface ReducerMap {  
  [key: string]: ReducerFunction<any>;  
}  
  
function combineReducers<S>(reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {  
  function reducer(previousState = initialState, action: Action): S {  
    const reducerMapKeys: string[] = Object.keys(reducerMap);  
  
    const newState: S = reducerMapKeys.reduce(  
      (accumulatedState: S, key: string) => {  
        const substateReducer: ReducerFunction<any> = reducerMap[key];  
        const previousSubstate: any = previousState[key];  
        accumulatedState[key] = substateReducer(previousSubstate, action);  
  
      },  
    );  
    return newState;  
  }  
  return reducer;  
}
```

```
const appReducers: ReducersMap = {  
  darkMode: darkModeReducer,  
  cart: cartReducer,  
  taxRate: taxRateReducer  
};
```

```
interface ReducerMap {  
  [key: string]: ReducerFunction<any>;  
}  
  
const appReducers: ReducersMap = {  
  darkMode: darkModeReducer,  
  cart: cartReducer,  
  taxRate: taxRateReducer  
};  
  
function combineReducers<S>(reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {  
  function reducer(previousState = initialState, action: Action): S {  
    const reducerMapKeys: string[] = Object.keys(reducerMap);  
  
    const newState: S = reducerMapKeys.reduce(  
      (accumulatedState: S, key: string) => {  
        const substateReducer: ReducerFunction<any> = reducerMap[key];  
        const previousSubstate: any = previousState[key];  
        accumulatedState[key] = substateReducer(previousSubstate, action);  
        return accumulatedState;  
      },  
      initialState  
    );  
    return newState;  
  }  
  return reducer;  
}
```

Store Demo



Slides: <https://johncrowson.com/assets/angular-mix.pdf>

Thank You!

