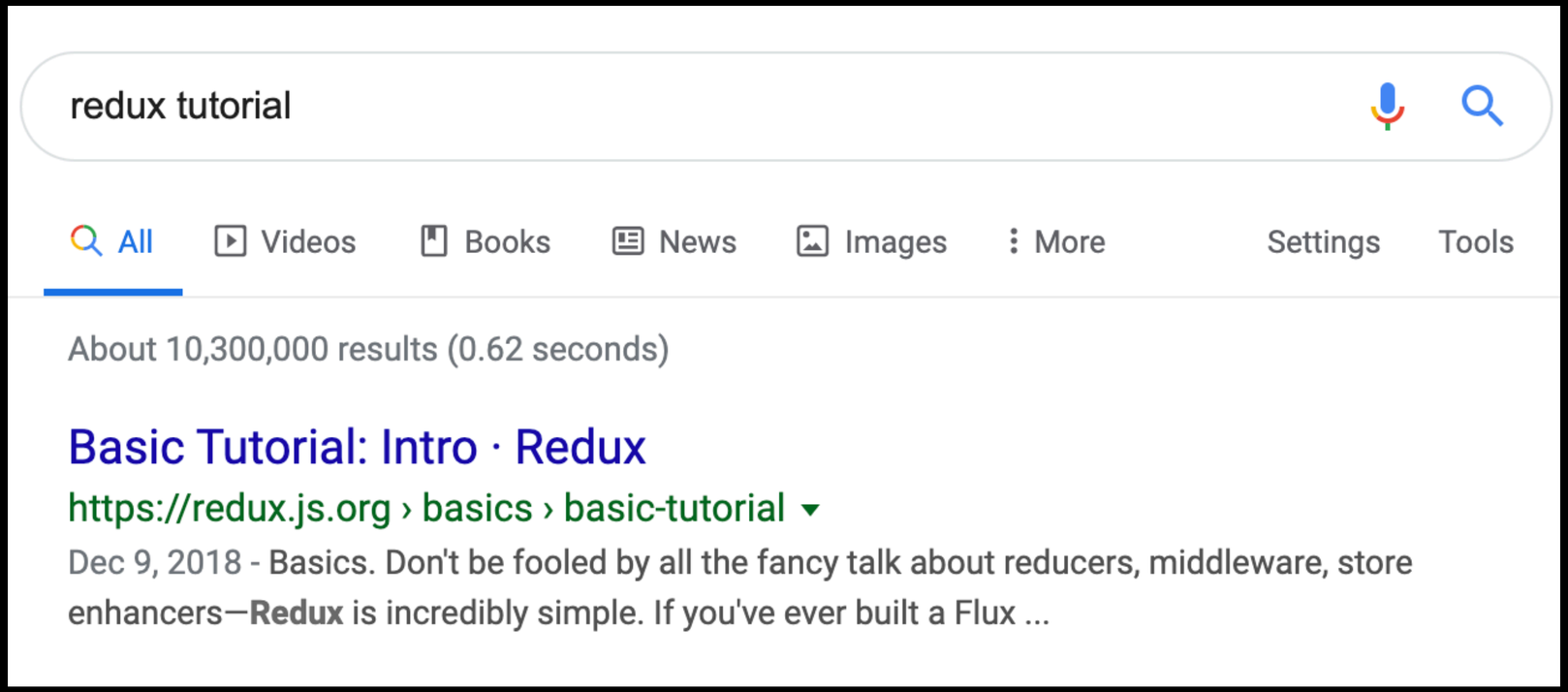
Understand Redux by Implementing It

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 ...

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

NGXS

@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.

Redux

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

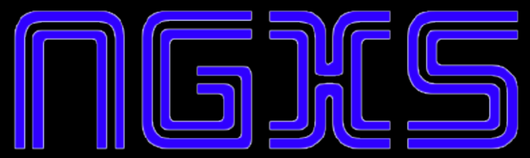
Start by installing the latest `@ngxs/store` package from npm.

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


Cranky's Cab

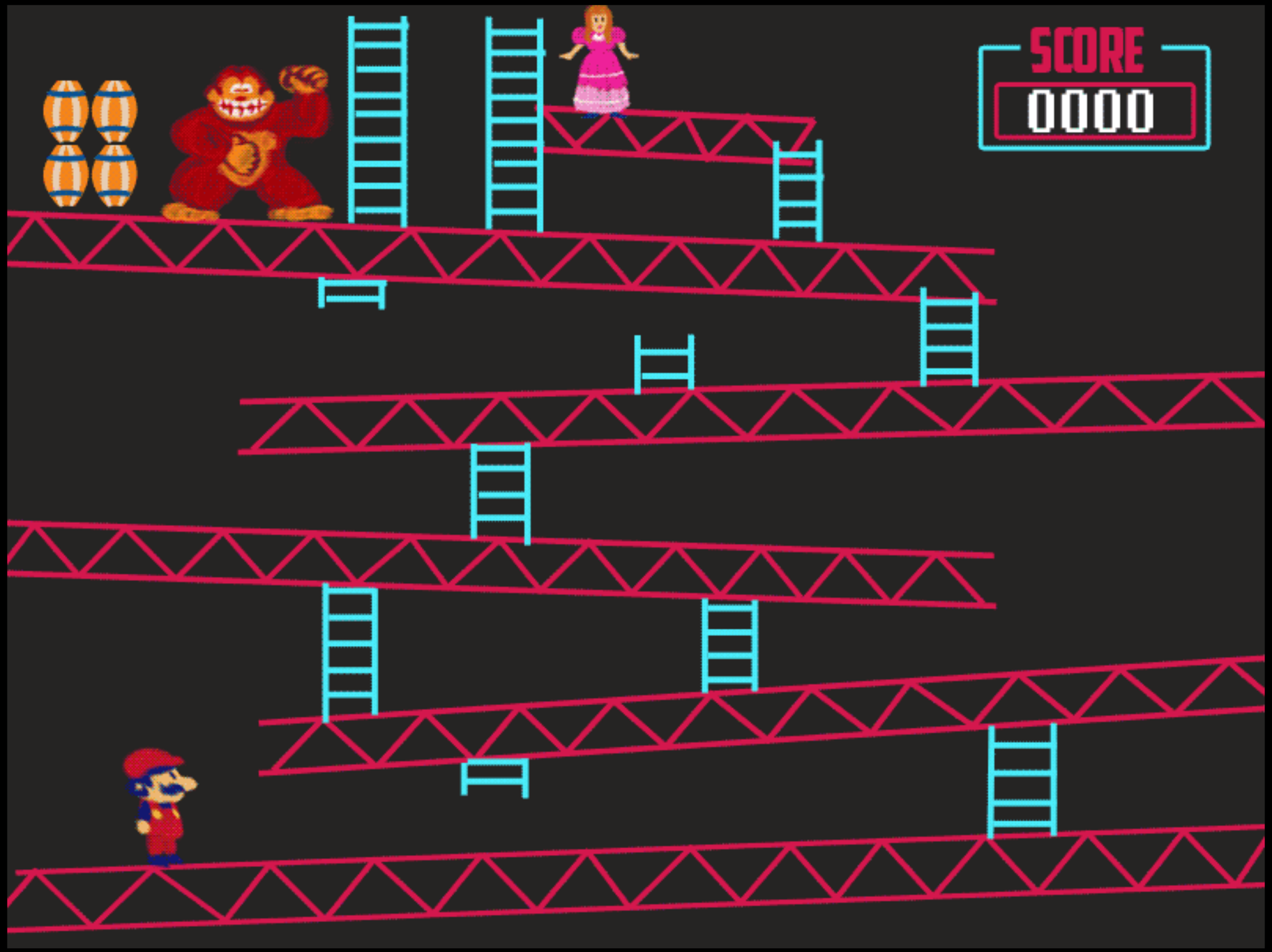
#Goals

Independent



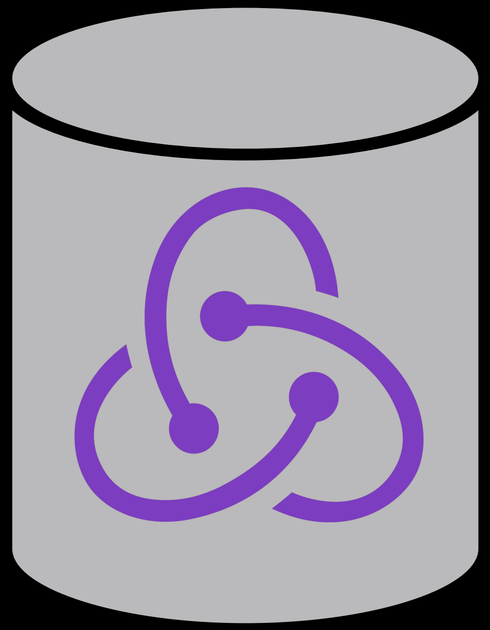
Many more!

Fearless

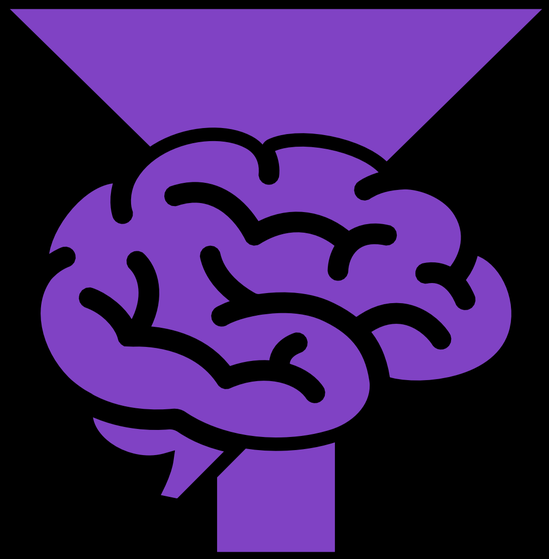

SCORE
0000

Be a Better Mario!

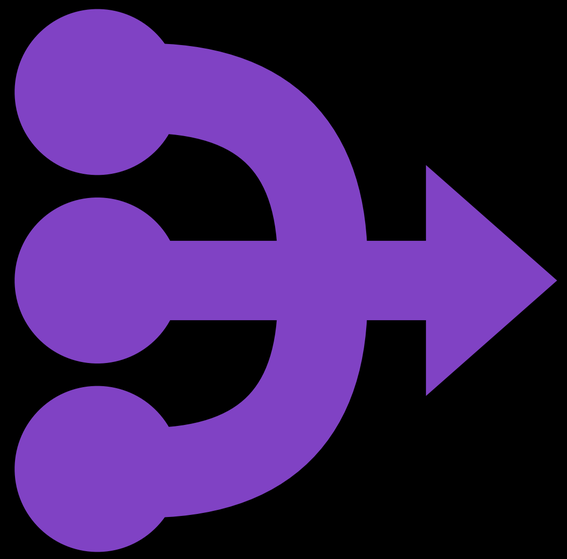
Agenda



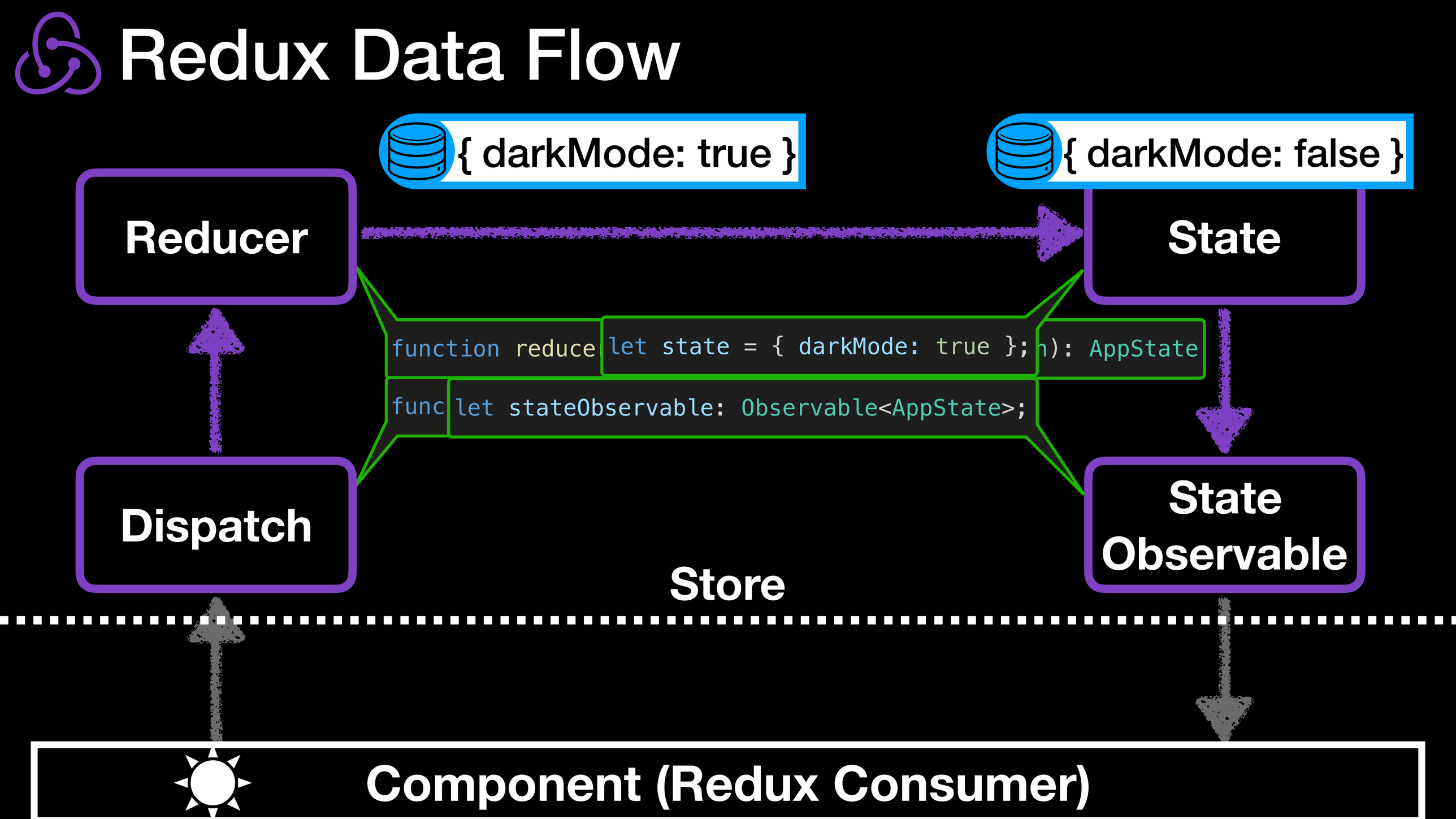
Store



Memoized Selector



Combine Reducers


Redux Data Flow
{ darkMode: true }
{ darkMode: false }
Reducer
State
function reduce
let state = { darkMode: true };
n): AppState
func
let stateObservable: Observable<AppState>;
Dispatch
State
Observable
Store
Component (Redux Consumer)

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> {

  dispatch(action: Action) {

  }
}
```

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>) {



  }

  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);
  }
}
```

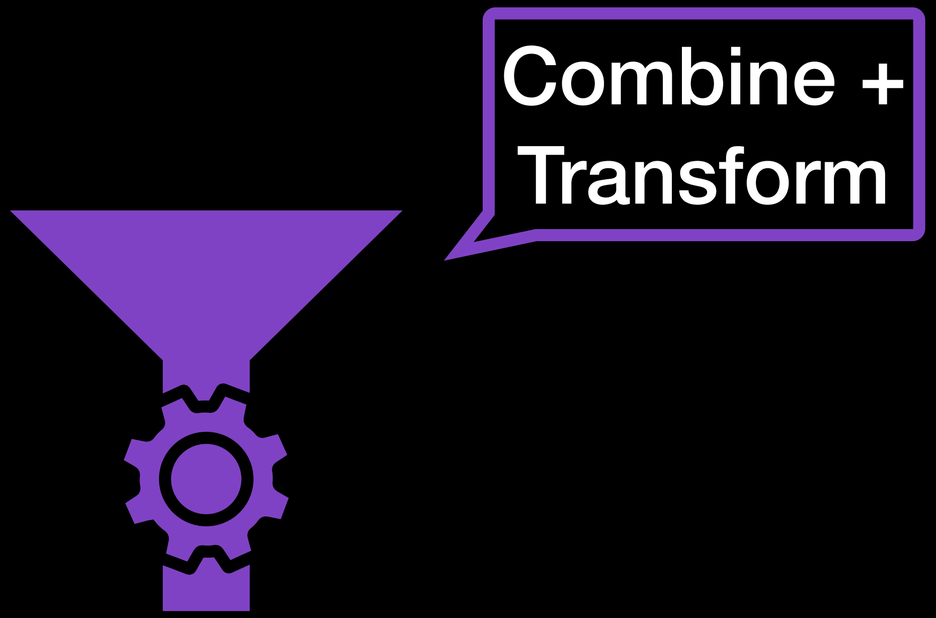
Store Demo

createSelector / Reselect

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

createSelector / Reselect

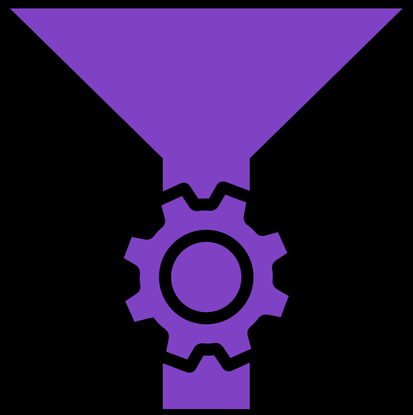
```
export const getTotalCostSelector = createSelector(
  [
    getCartSelector,
    getTaxRateSelector
  ],
  (cart: number[], taxRate: number) =>  getTotalCost(cart, taxRate)
);
```



createSelector / Reselect

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

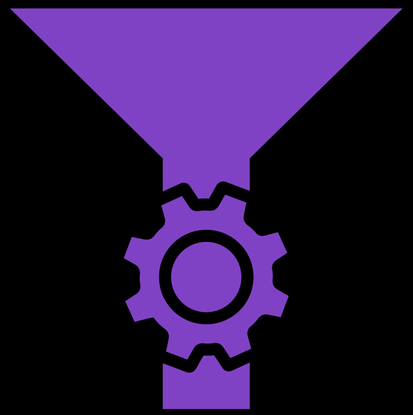
Combine + Transform



createSelector / Reselect

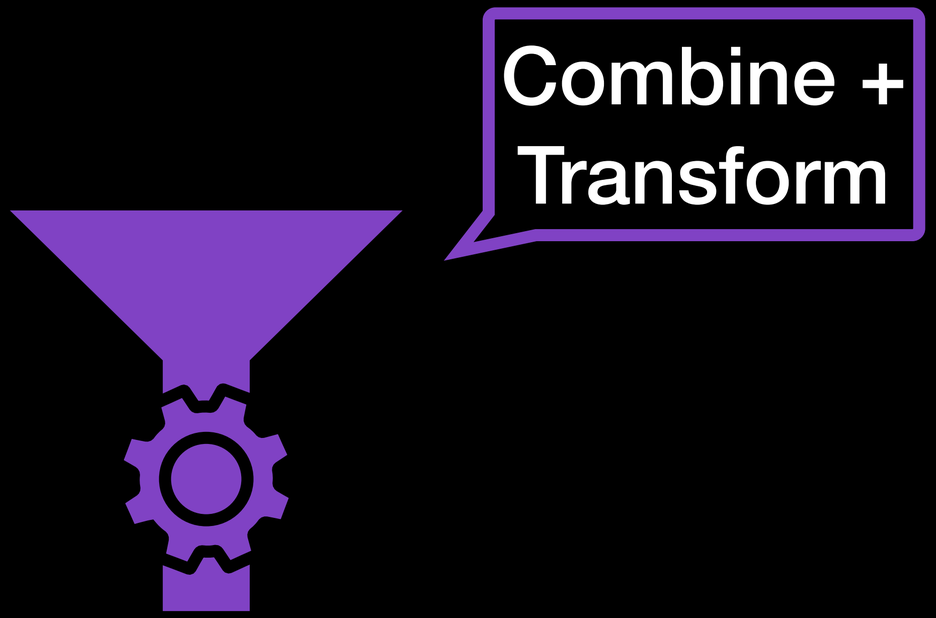
```
export const getTotalCostSelector = createSelector(
  [
    getCartSelector,
    getTaxRateSelector
  ],
  (cart: number[], taxRate: number) =>  getTotalCost(cart, taxRate)
);
```

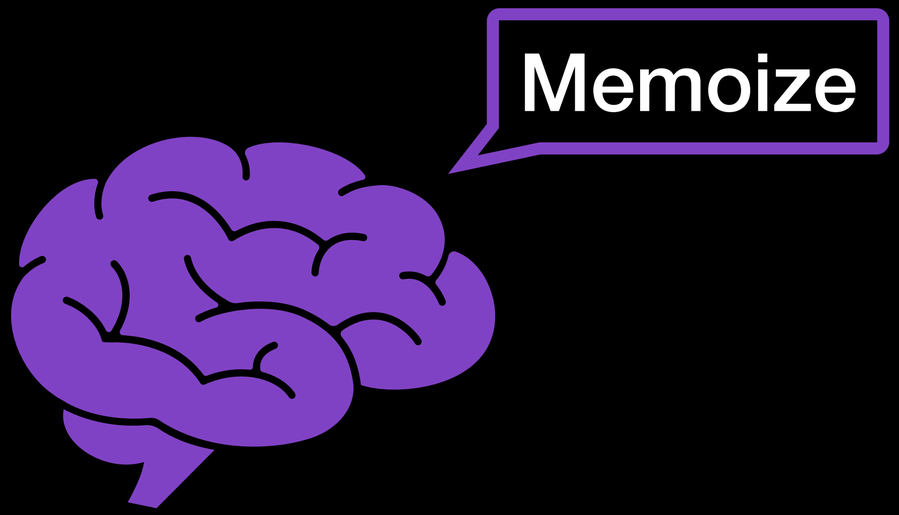
Combine + Transform



createSelector / Reselect

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



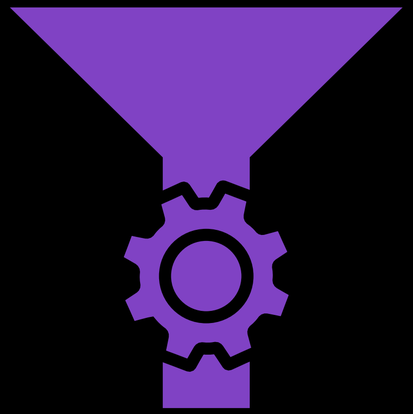


createSelector / Reselect

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

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

Combine + Transform

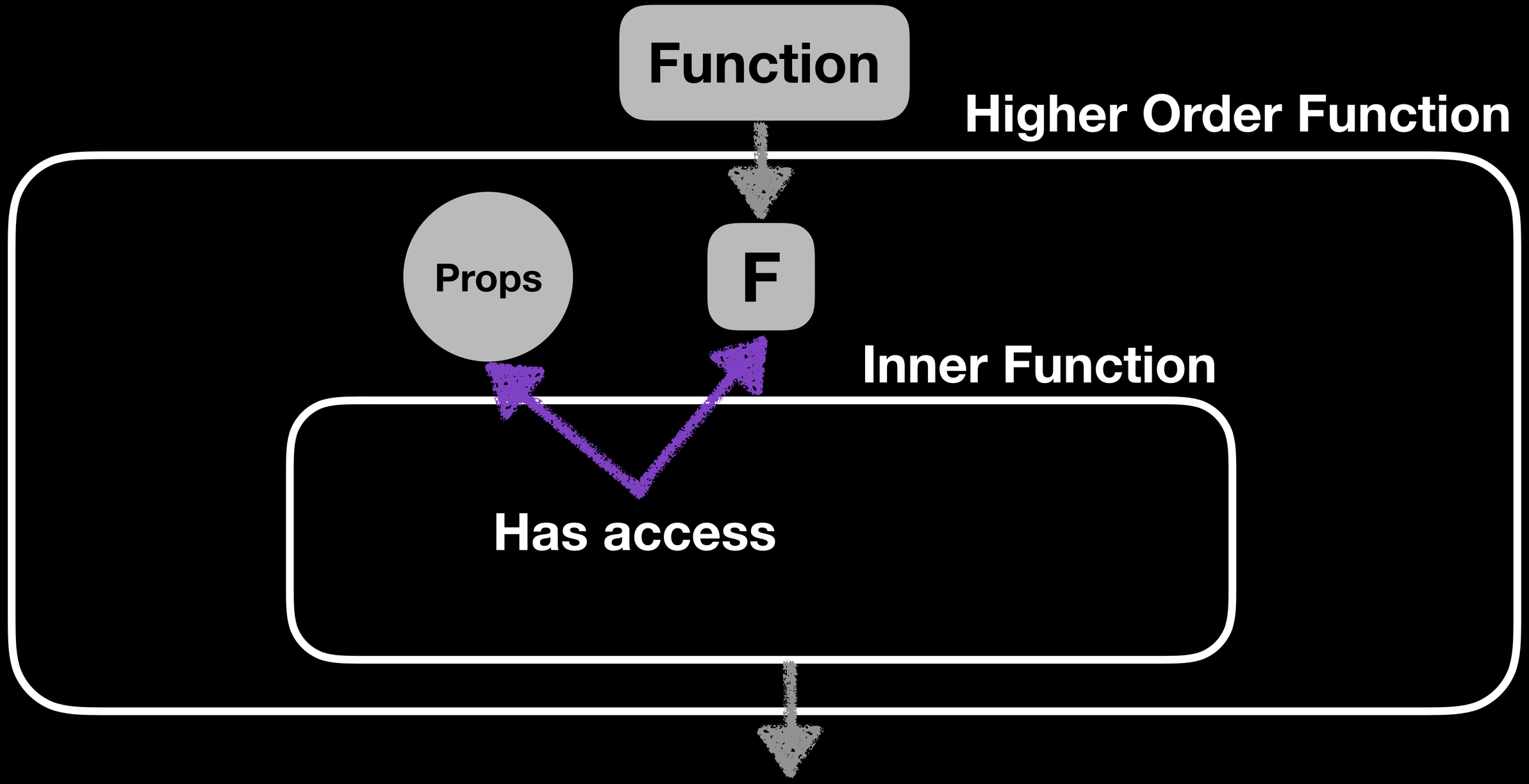


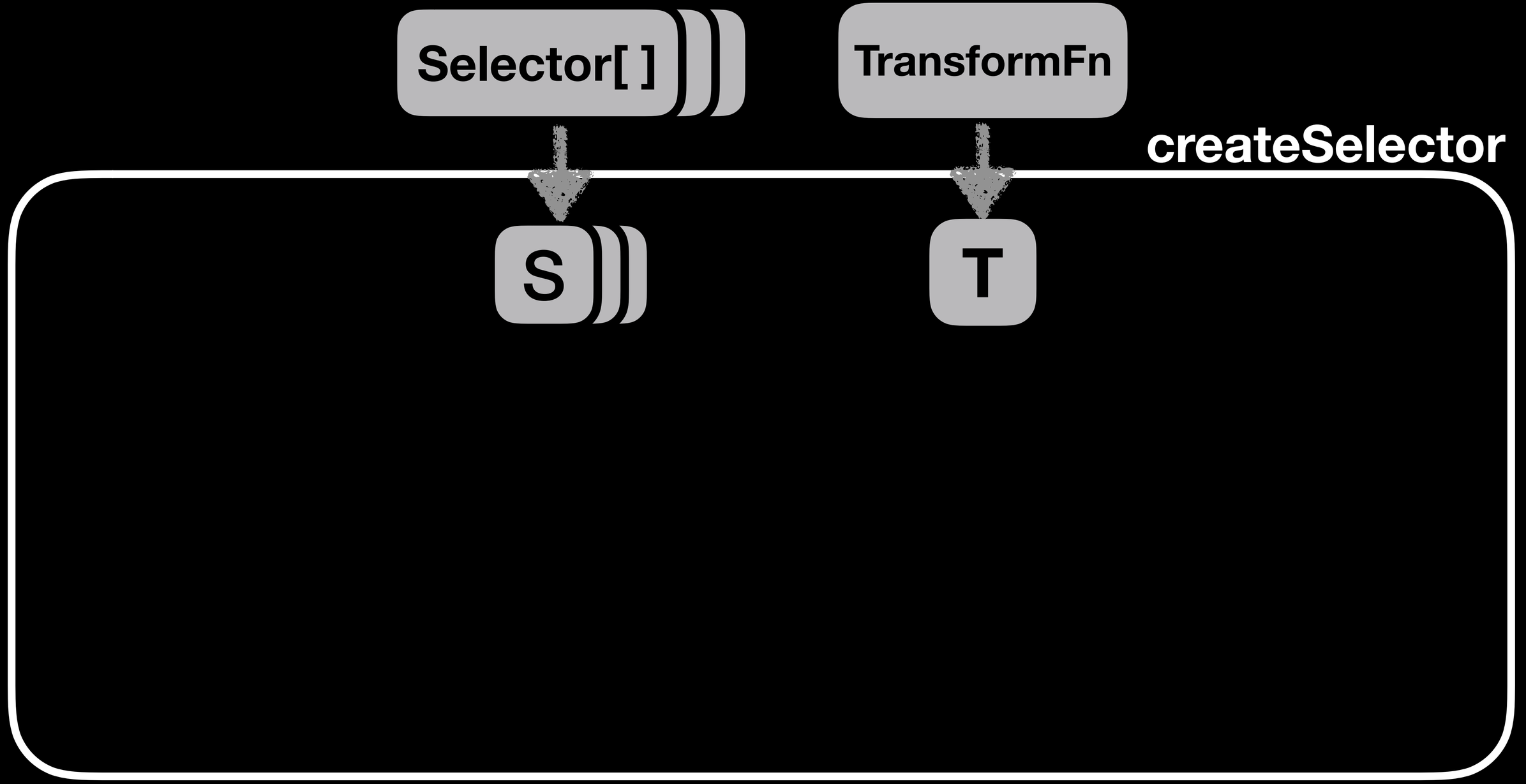
Memoize

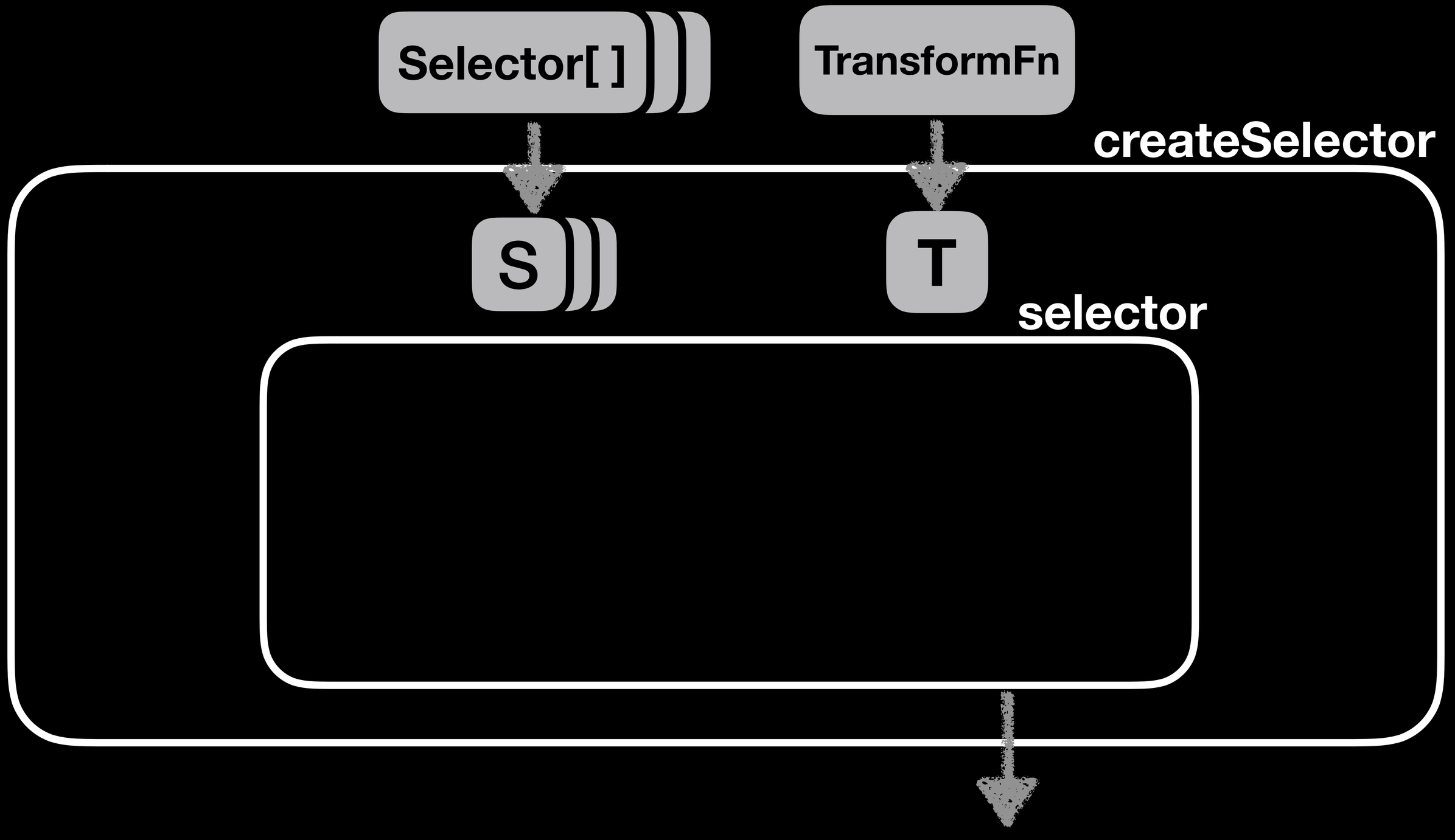
createSelector / Reselect

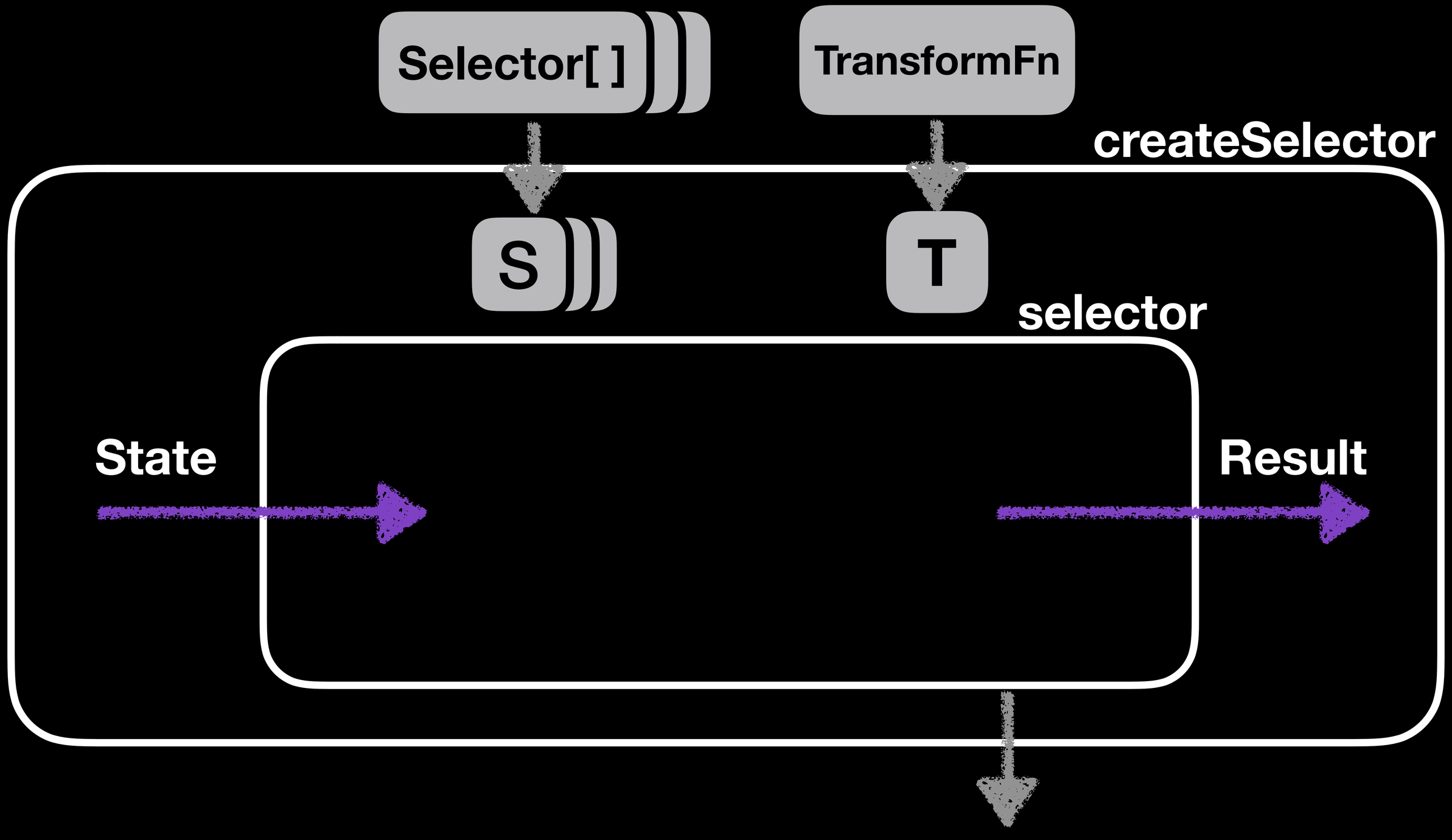
```
function createSelector(
  selectorFns: SelectorFn[],
  transformationFn: AnyFn
): SelectorFn
```

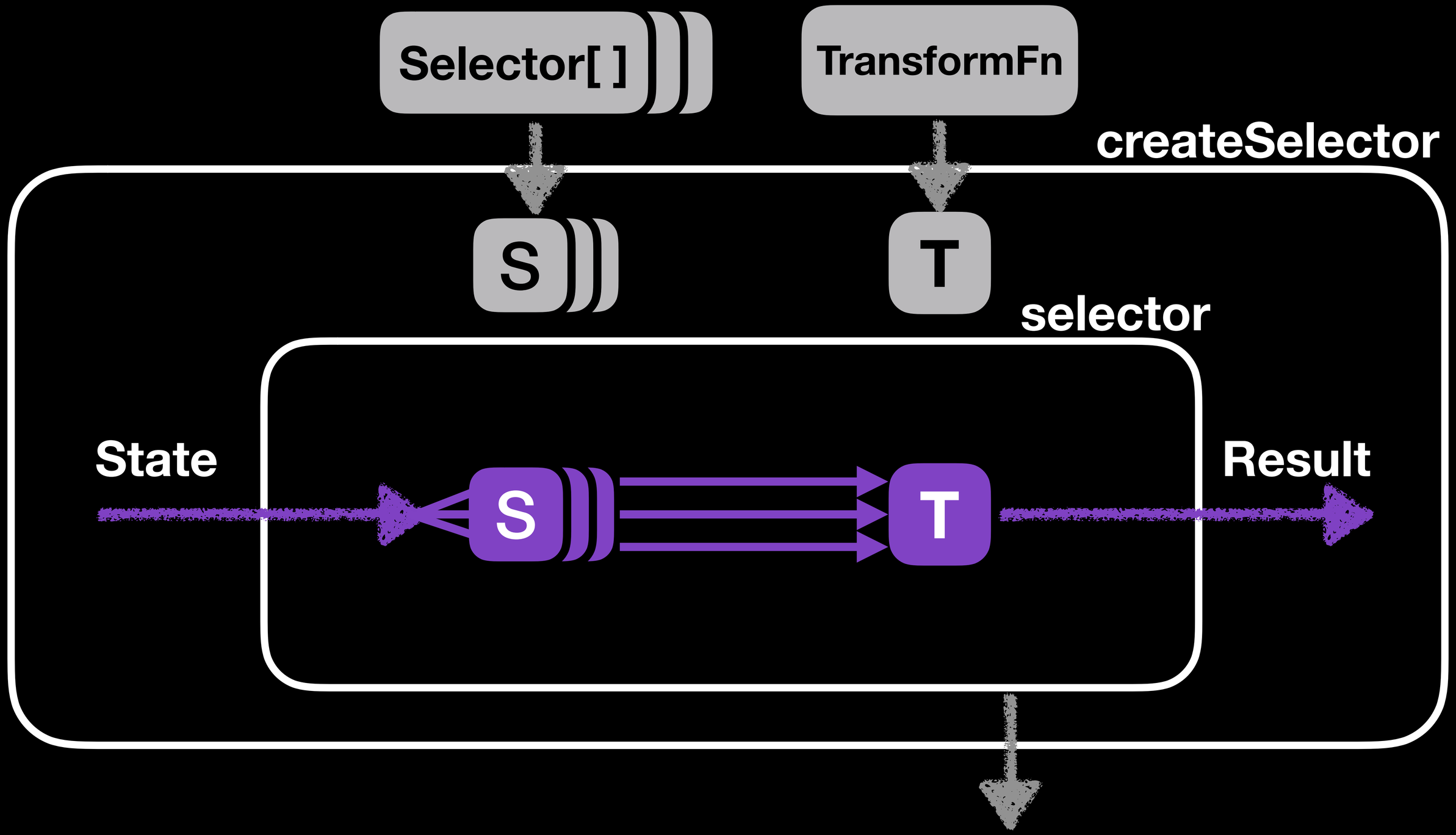
Higher Order Function




Selector[ ]
TransformFn
createSelector
S
T


Selector[ ]
TransformFn
createSelector
S
T
selector


Selector[ ]
TransformFn
createSelector
S
T
selector
State
Result


Selector[ ]
TransformFn
createSelector
S
T
selector
State
S
T
Result

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

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

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

expensiveCalculation(5, 10);

const memoizedExpensiveCalculation = memoize(expensiveCalculation);
```

Calculates on every call

createSelector: Memoize

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

expensiveCalculation(5, 10);

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

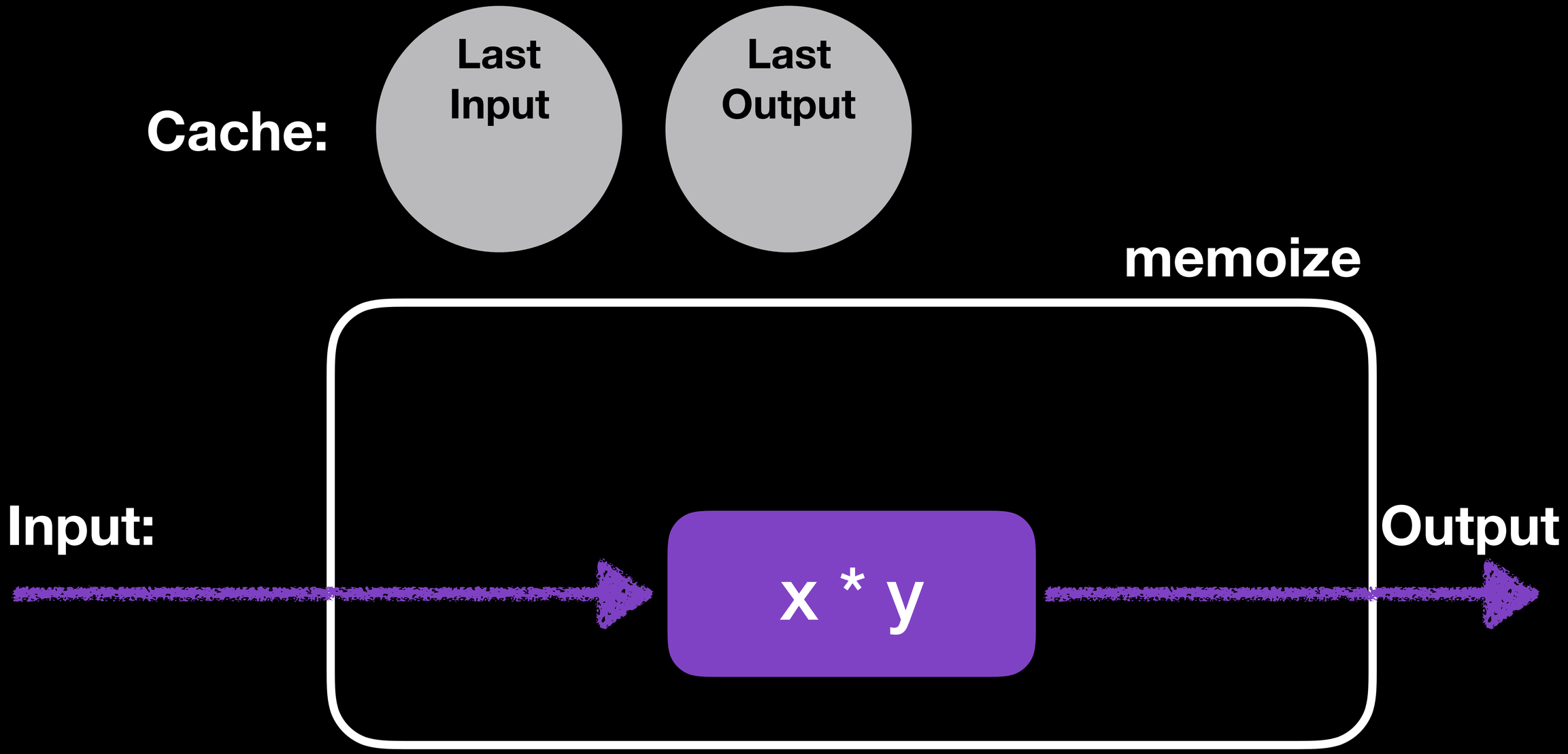
Calculates on every call

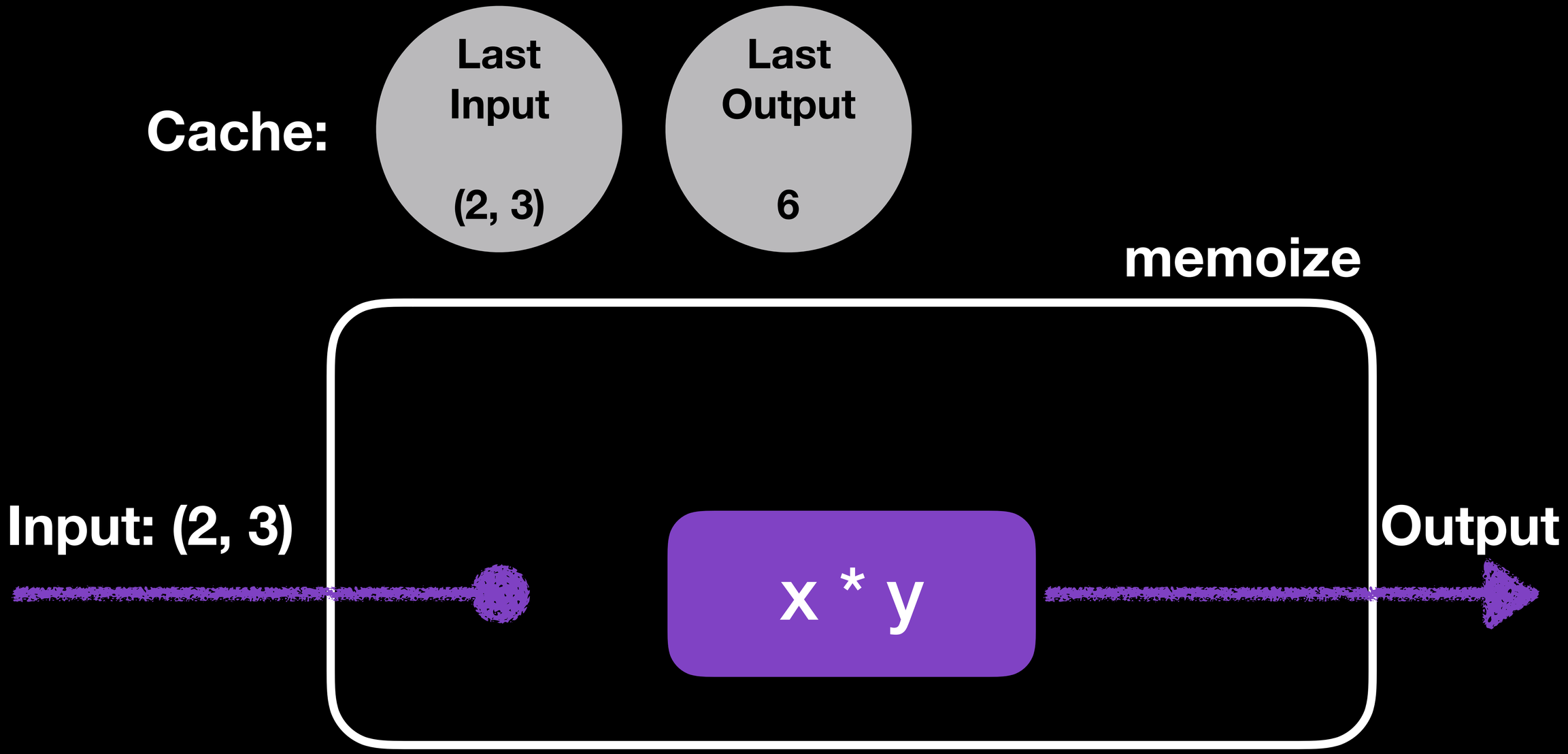
createSelector: Memoize

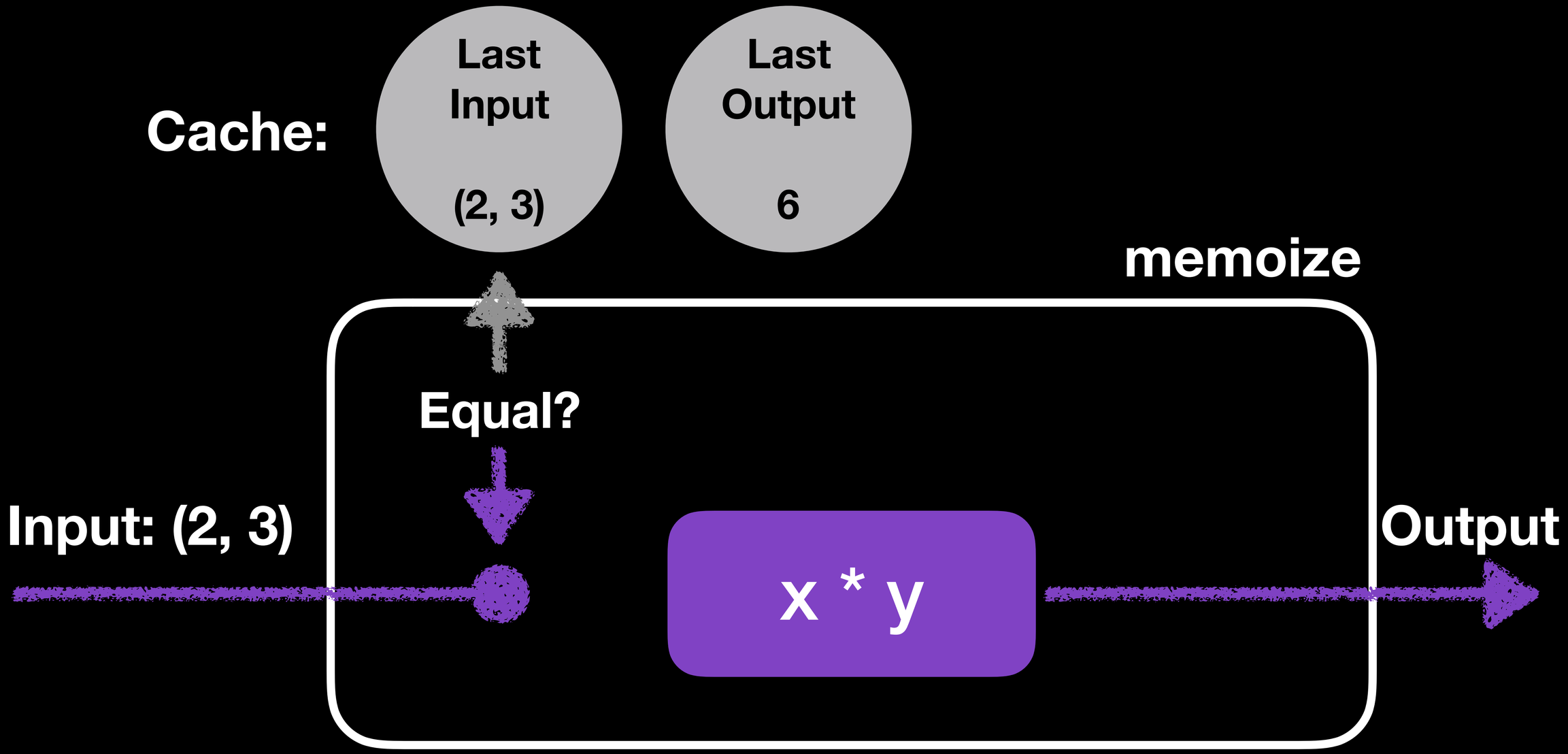
```ts
function expensiveCalculation(arg1: number, arg2: number): number {
  // ...
}

expensiveCalculation(5, 10);

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

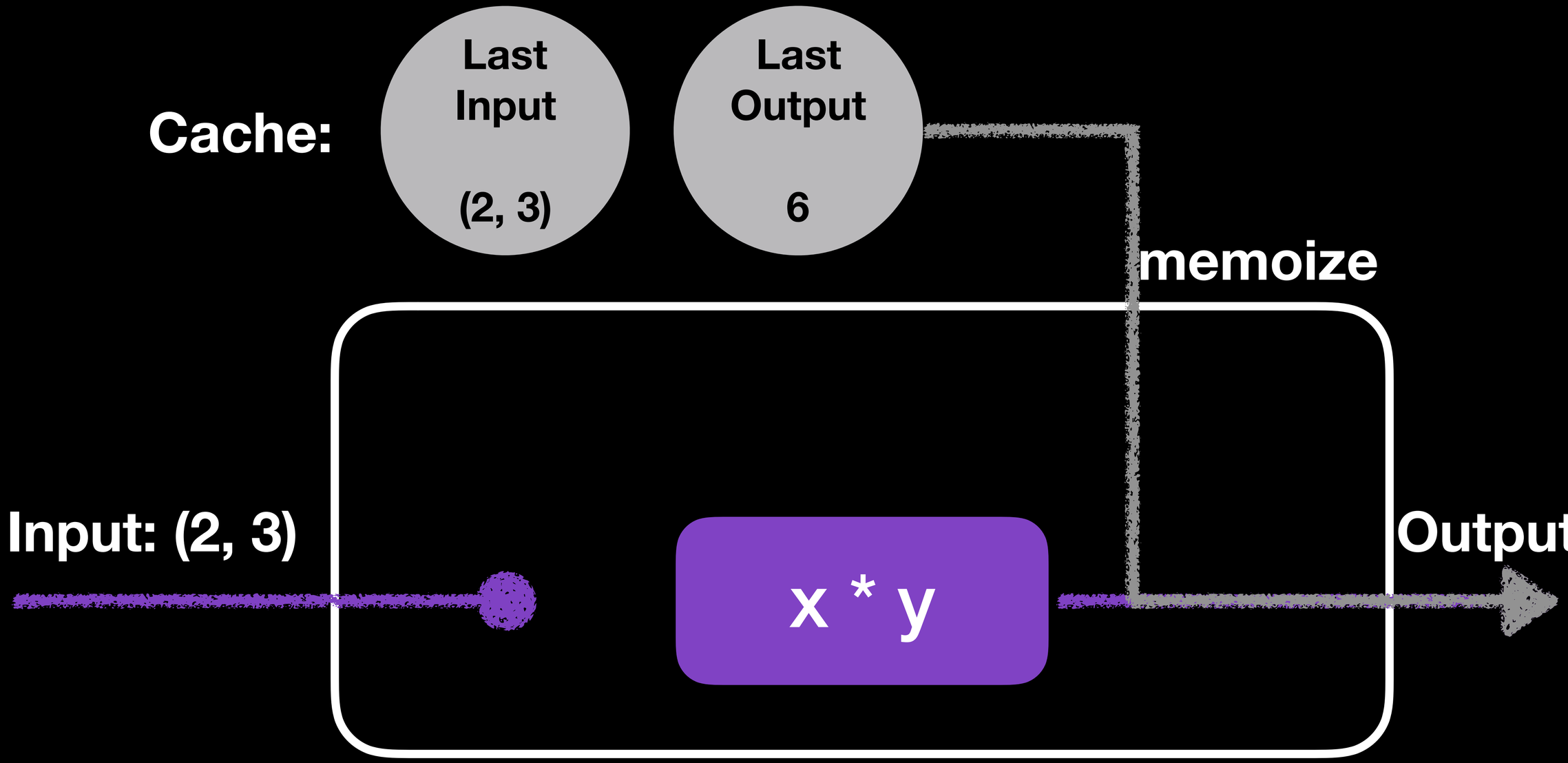
Calculates on every call

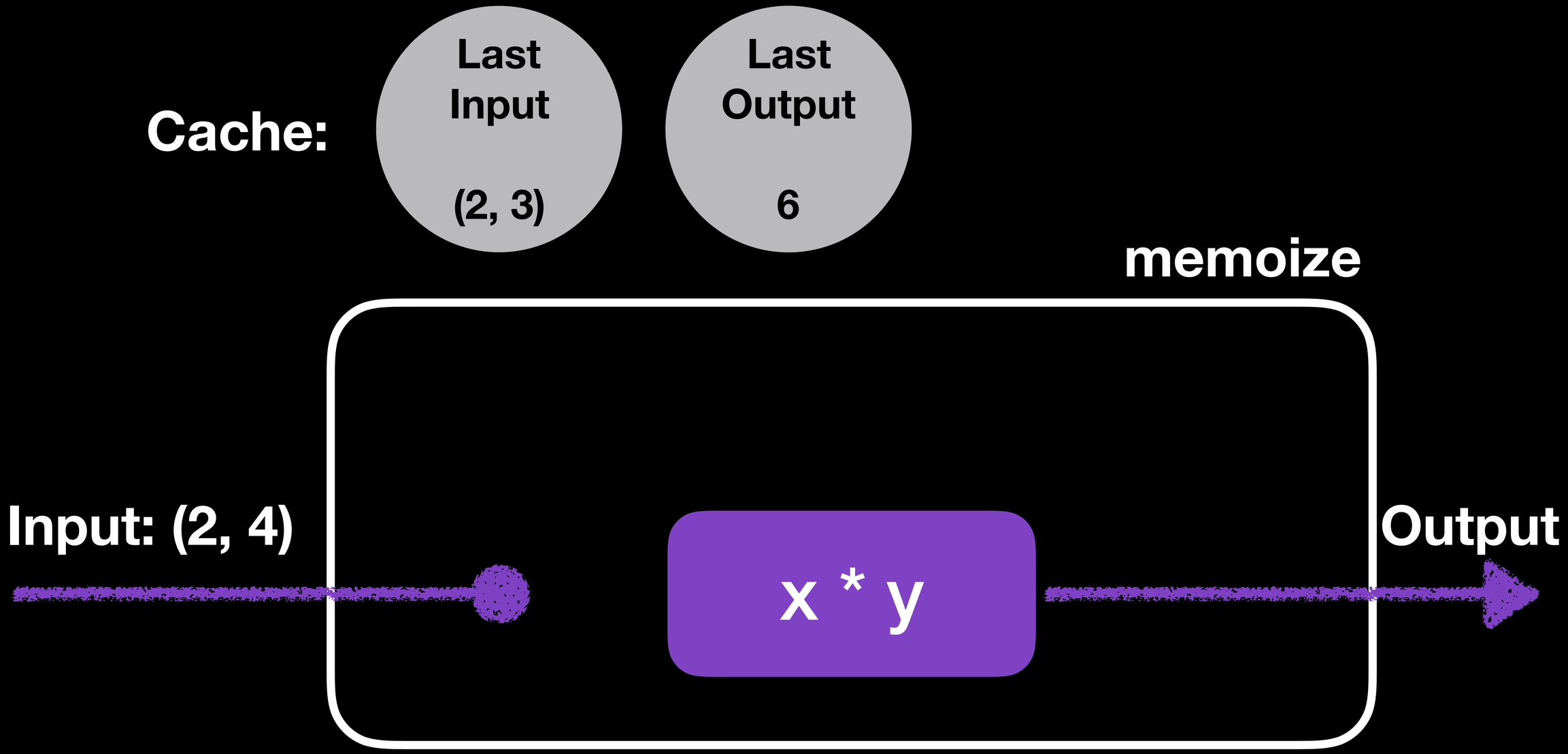
Calculates if input differs from last call

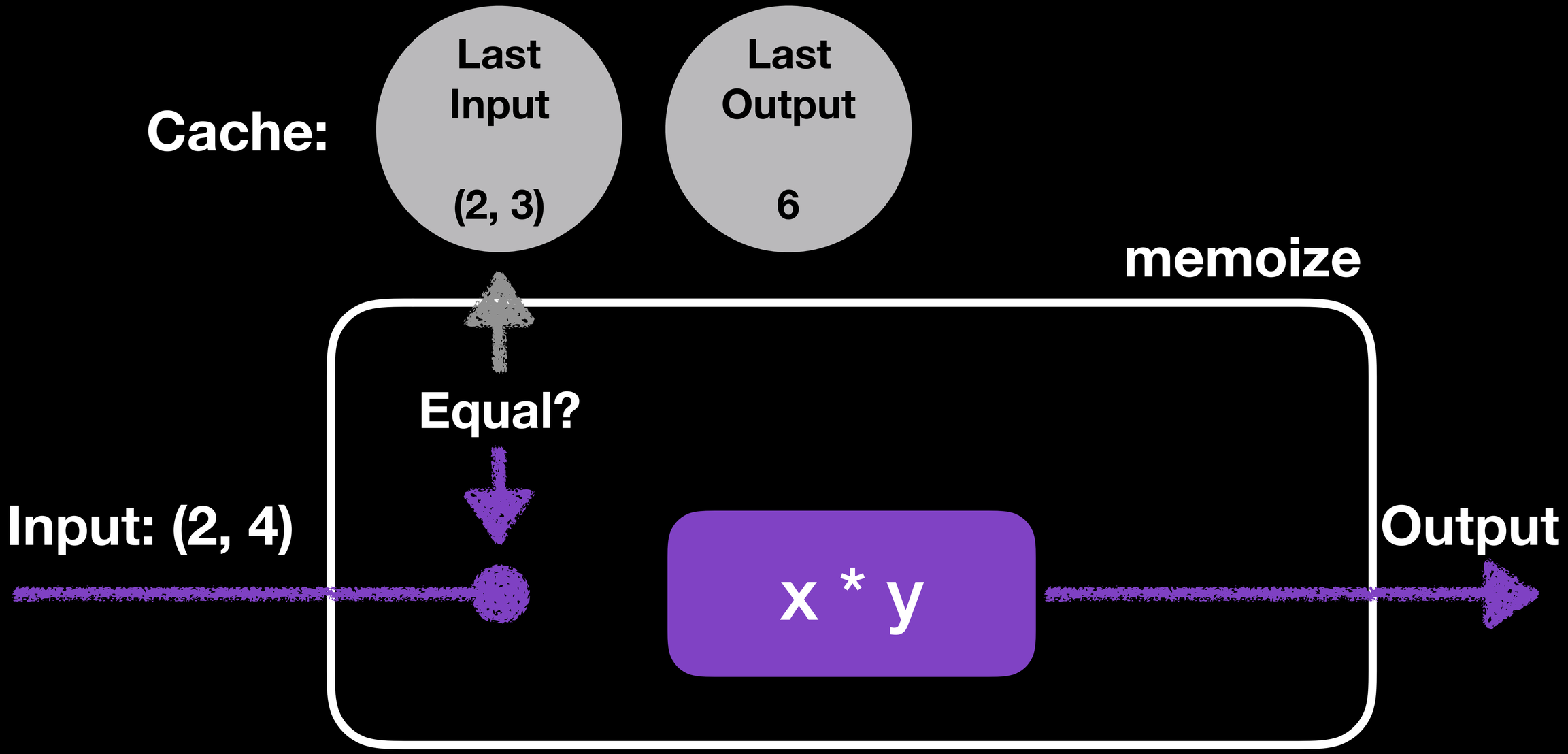

Cache:
Last Input
Last Output
memoize
Input:
x * y
Output

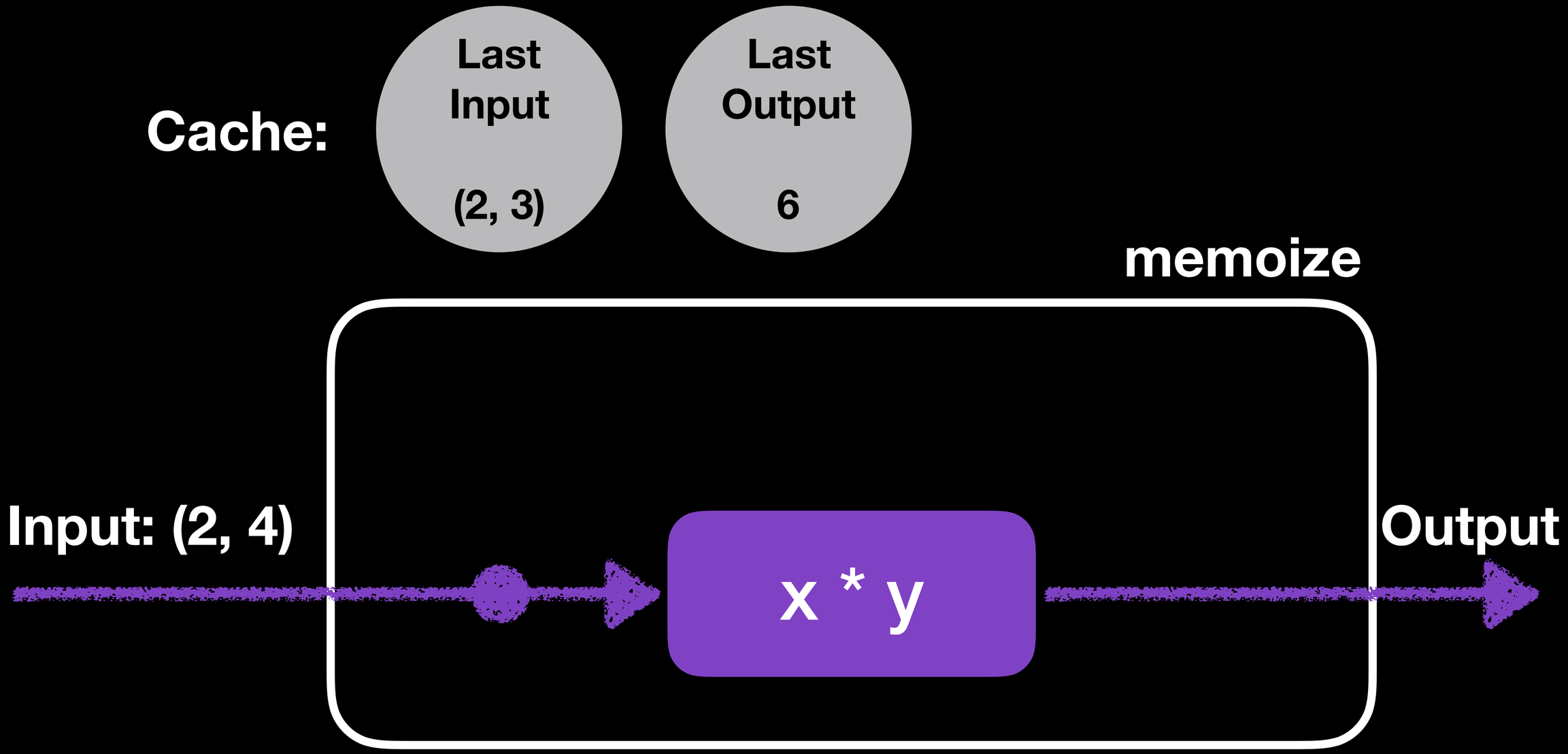

Cache:
Last Input
(2, 3)
Last Output
6
memoize
Input: (2, 3)
x * y
Output

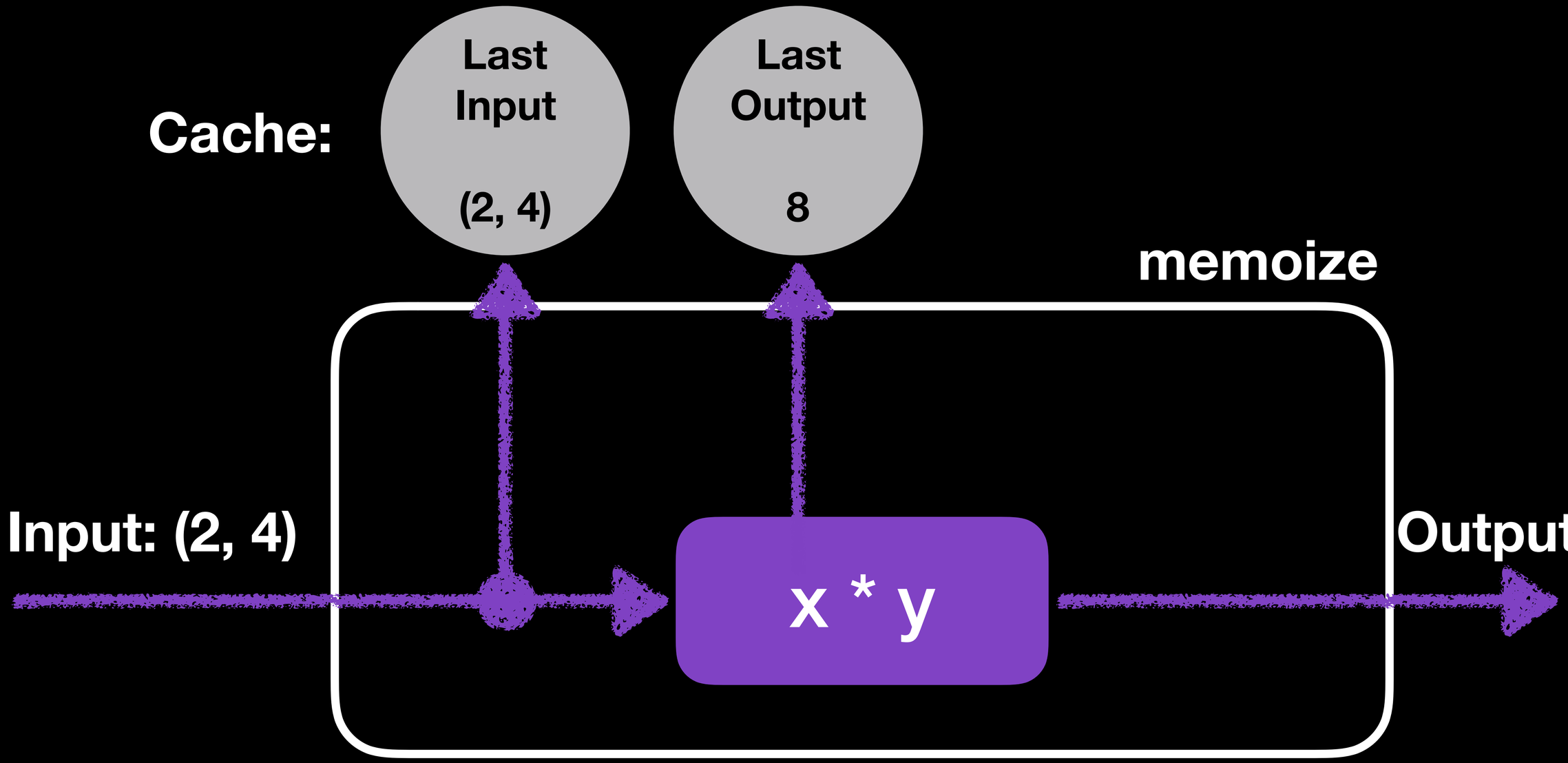

Cache:
Last Input
(2, 3)
Last Output
6
memoize
Equal?
Input: (2, 3)
x * y
Output


Cache:
Last Input
(2, 3)
Last Output
6
memoize
Input: (2, 3)
x * y
Output: 6


Cache:
Last Input
(2, 3)
Last Output
6
memoize
Input: (2, 4)
x * y
Output


Cache:
Last Input
(2, 3)
Last Output
6
memoize
Equal?
Input: (2, 4)
x * y
Output


Cache:
Last Input
(2, 3)
Last Output
6
memoize
Input: (2, 4)
x * y
Output


Cache:
Last Input
(2, 4)
Last Output
8
memoize
Input: (2, 4)
x * y
Output: 8

createSelector: Memoize

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

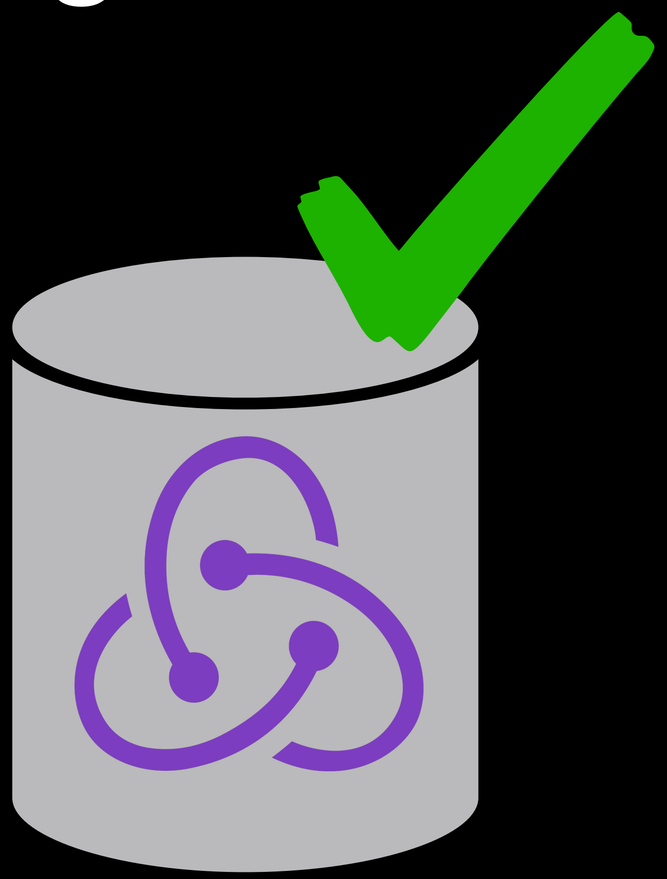
createSelector: Memoize

```
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);
}
```

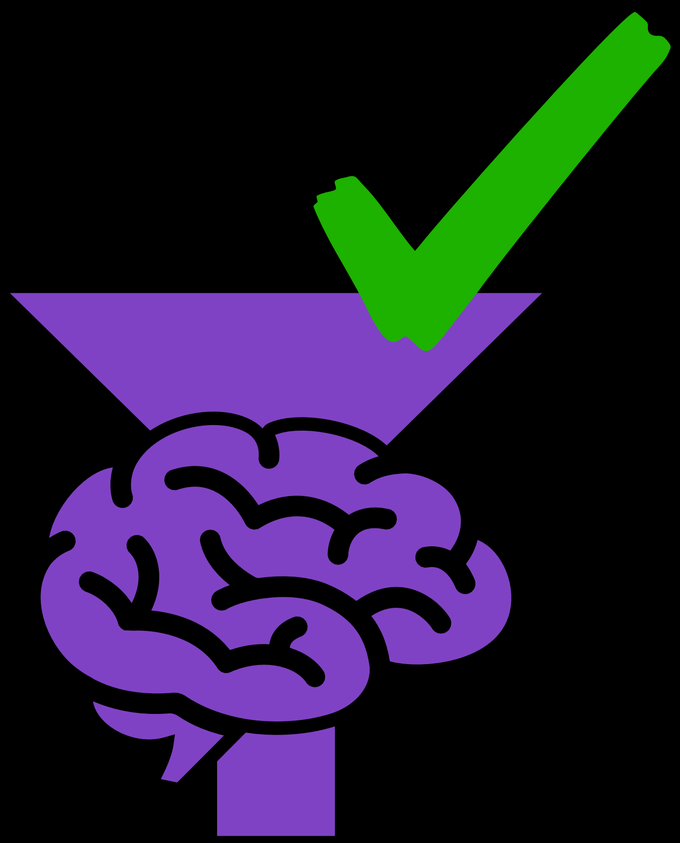
createSelector: Memoize

```
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);
}
```

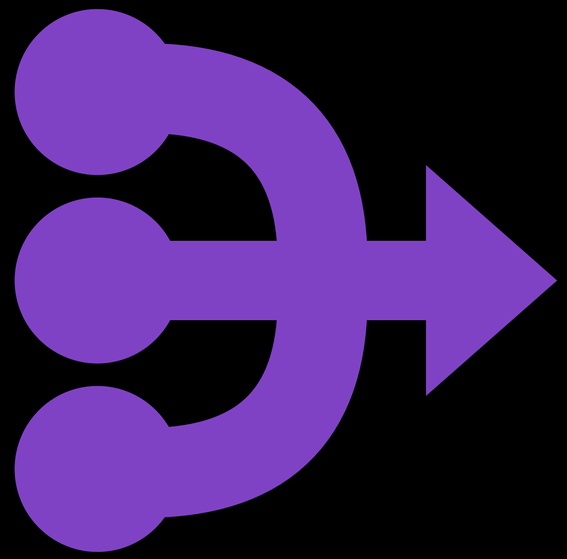
Agenda



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

substate reducer

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

substate reducer

combineReducers()

state reducer

StoreModule.forRoot(reducer)

NgRx

constructor(reducer)

Store

substate reducer

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

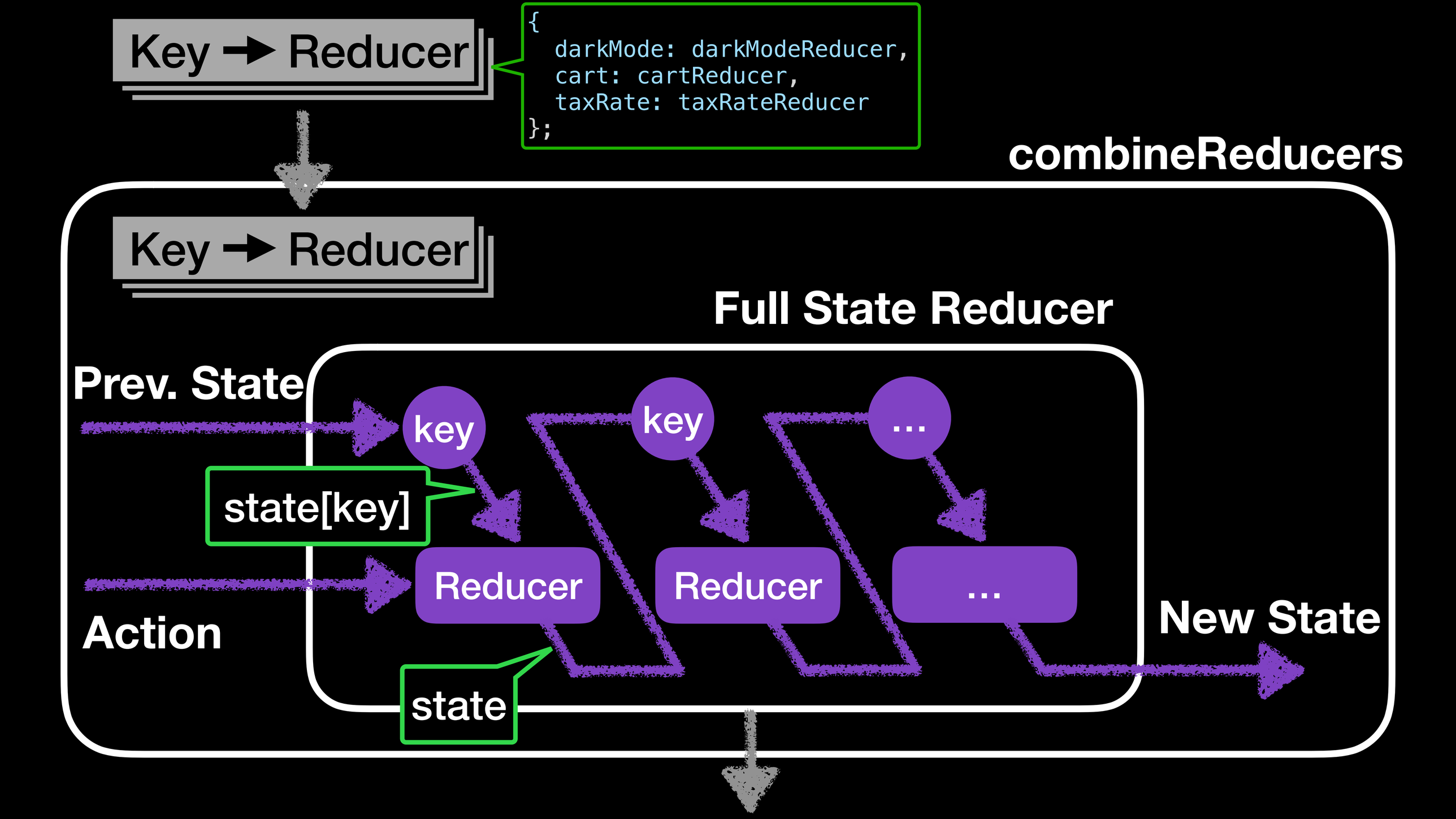
Combine Reducers

```
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
key
...
state[key]
Reducer
Reducer
...
Action
New State
state

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

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

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

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

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

}
```

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

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

```ts
function combineReducers<S>(reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {
  function reducer(previousState = initialState, action: Action): S {









  }
  return reducer;
}
```

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

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

```ts
function combineReducers<S>(reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {
  function reducer(previousState = initialState, action: Action): S {

    return newState;
  }
  return reducer;
}
```

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

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

```ts
function combineReducers<S>(reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {
  function reducer(previousState = initialState, action: Action): S {
    const reducerMapKeys: string[] = Object.keys(reducerMap);

    return newState;
  }
  return reducer;
}
```

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

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

```ts
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(









    return newState;
  }
  return reducer;
}
```

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

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

```ts
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;
}
```

```
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];



      },

    );
    return newState;
  }
  return reducer;
}
```

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

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

```ts
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;
}
```

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

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

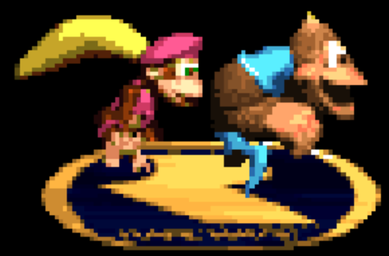
```ts
function combineReducers<S>(reducerMap: ReducerMap, initialState: S): ReducerFunction<S> {
  function reducer(previousState = initialState, action: Action): S {
    const reducerMapKeys: string[] = Object.keys(reducerMap);

    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;
}
```

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

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

```ts
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;
      },
      previousState
    );
    return newState;
  }
  return reducer;
}
```

Store Demo



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


Thank
You!