FE CONF

{ React ♡ Recoil }

상태관리, 이제 Recoil 하세요!

김 현 태 @MASS_ADOPTION

woody@mass-aoption.com

안녕하세요!
김현태 라고합니다

- 블록체인 기술기반 디지털보증 솔루션, 명품자산관리 앱 개발
- Fin-Tech, O2O, e-Commerce 등 Full-Stack 개발 13년+
- .NET(C#), Python, **JavaScript, React, Node.js,** Go…

👫 🐈 🏕️ 🛠️ 🎸

MASS
ADOPTION

Chief Technology Officer
김 현 태 (Woody Kim)

Index

상태관리 라이브러리에 대한 고민

🤩 새로운 기술을 도입할 수 있는 절호의 기회

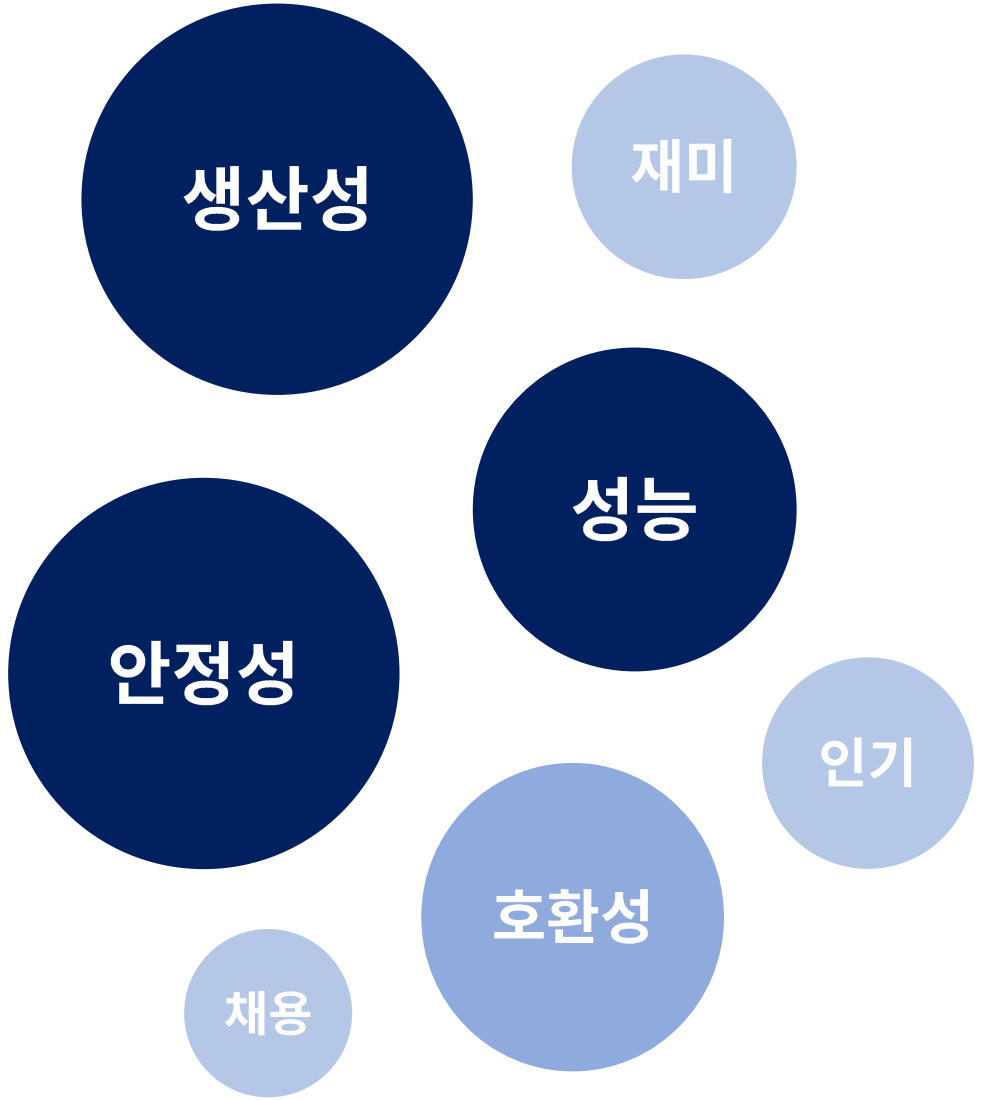
😭 무조건 빠르게 개발해야 한다

🤔 최대한 네이티브 앱스럽게(?) 만들자

😨 갑자기 라이브러리 운영이 중단된다면?

😱 서비스가 폭발적으로 성장할 것만 같은 착각

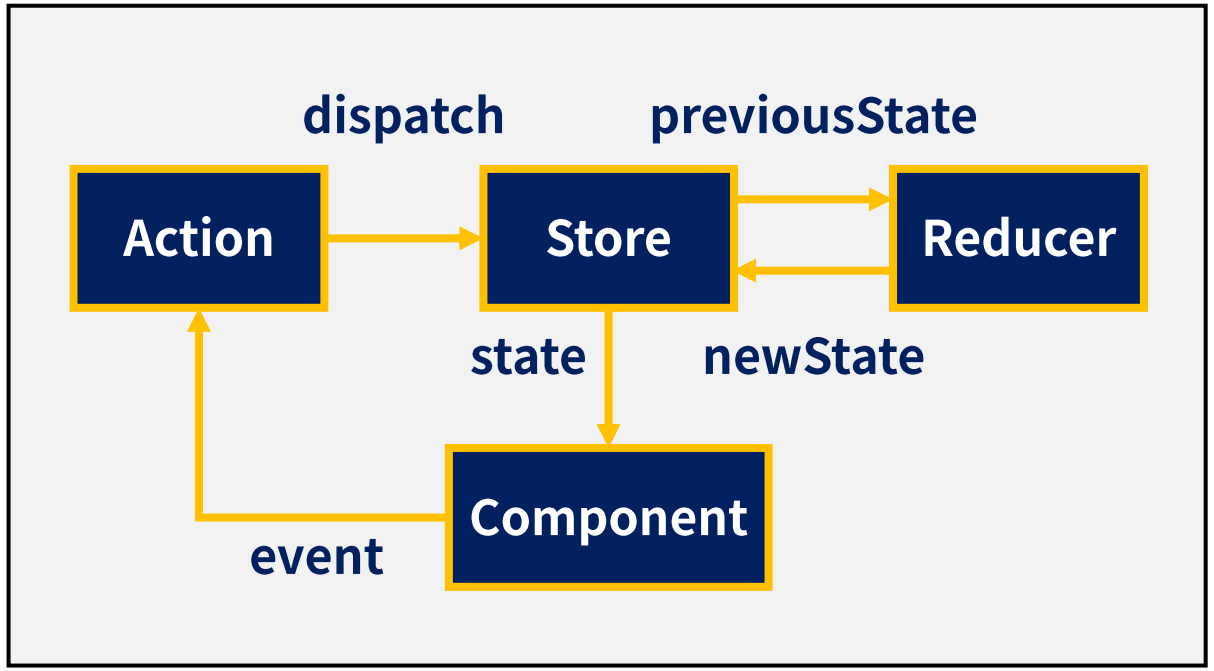
😎 그리고 힙해야 한다

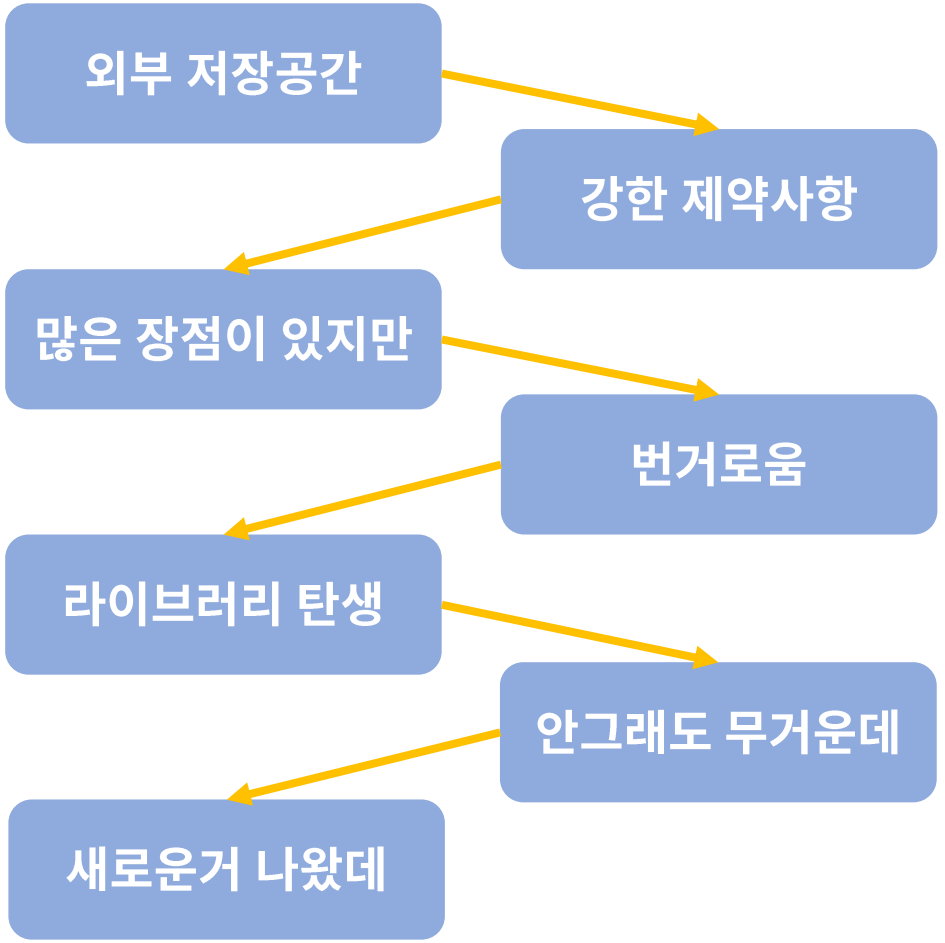


너도 나도 쓰는 Redux

"귀찮고 번거로워요.."

"원래 그렇게 사용하라고 설계되었으니까요"



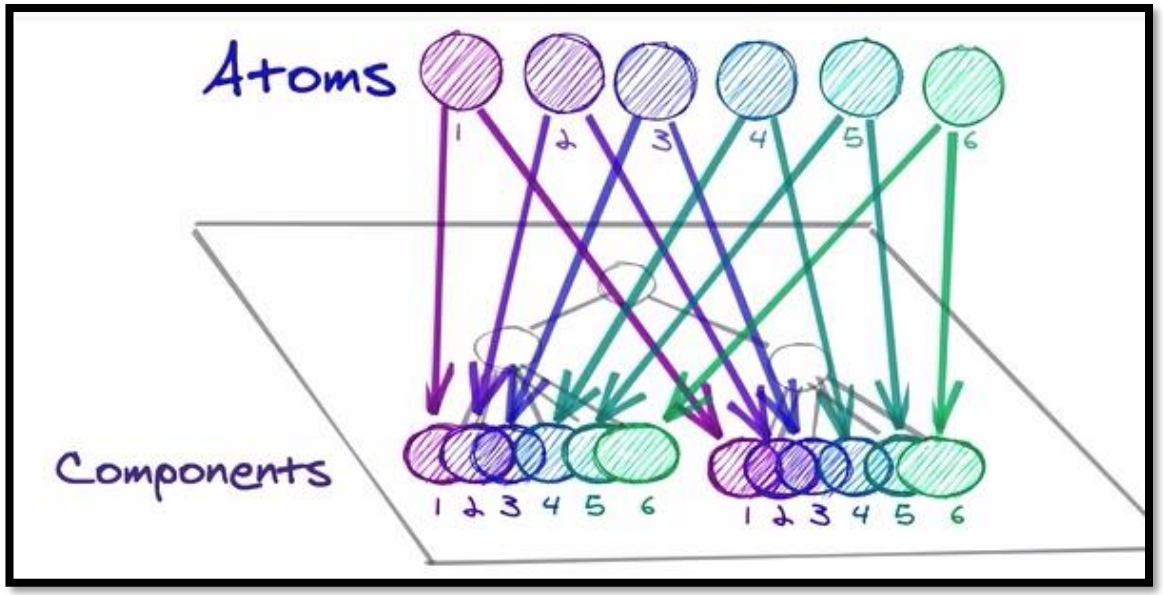


꼭 이렇게 까지 해야만..

이제는 Redux만이 답이 아닙니다

Flux	**Redux,** Zustand	익숙하지만 여전히 번거롭다..
Proxy	**MobX,** Valtio	쉽지만 리액트답지 못하달까..
Context	**Context API**	복잡성이 커지면 Context 분리 어려움
Server Cache	**React-query,** SWR	이것 만으로는 부족하지..
Atomic	**Recoil,** Jotai	**페이스북이 만들었다고? 간결하다!**

Recoil의 핵심 컨셉



(https://recoiljs.org)

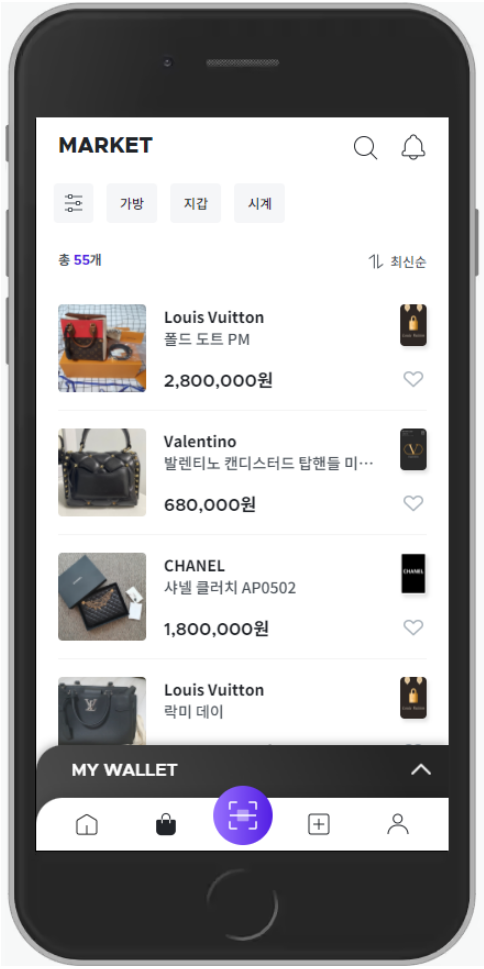
- ✓ 오직 React만을 위해 React처럼
- ✓ React 내부상태만 이용
- ✓ 작은 Atom 단위로 관리
- ✓ 순수함수 Selector
- ✓ Re-Render 최소화
- ✓ 데이터 흐름을 따라서
- ✓ 곧 새로운 React 기능과 호환성

{ React ♡ Recoil }

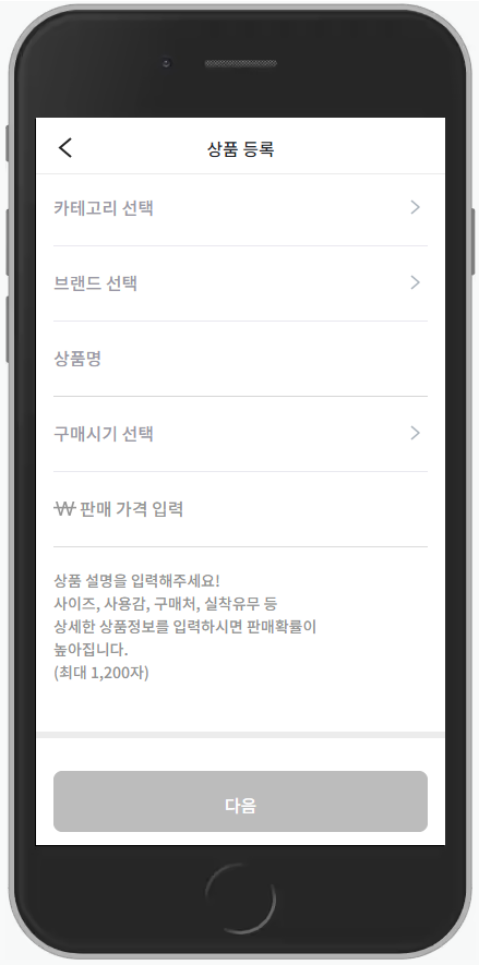
예제로 보는 Recoil 활용기

Atom/Selector 를 활용한 상품목록 만들기

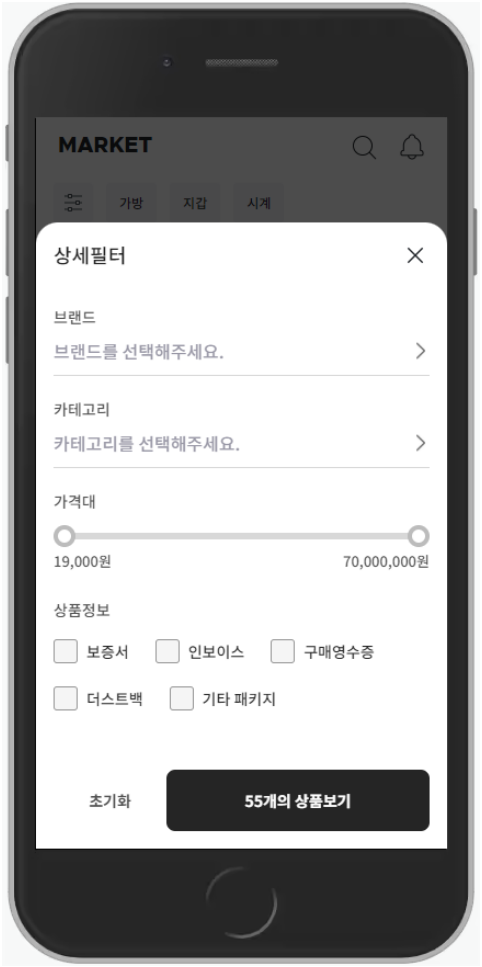
List

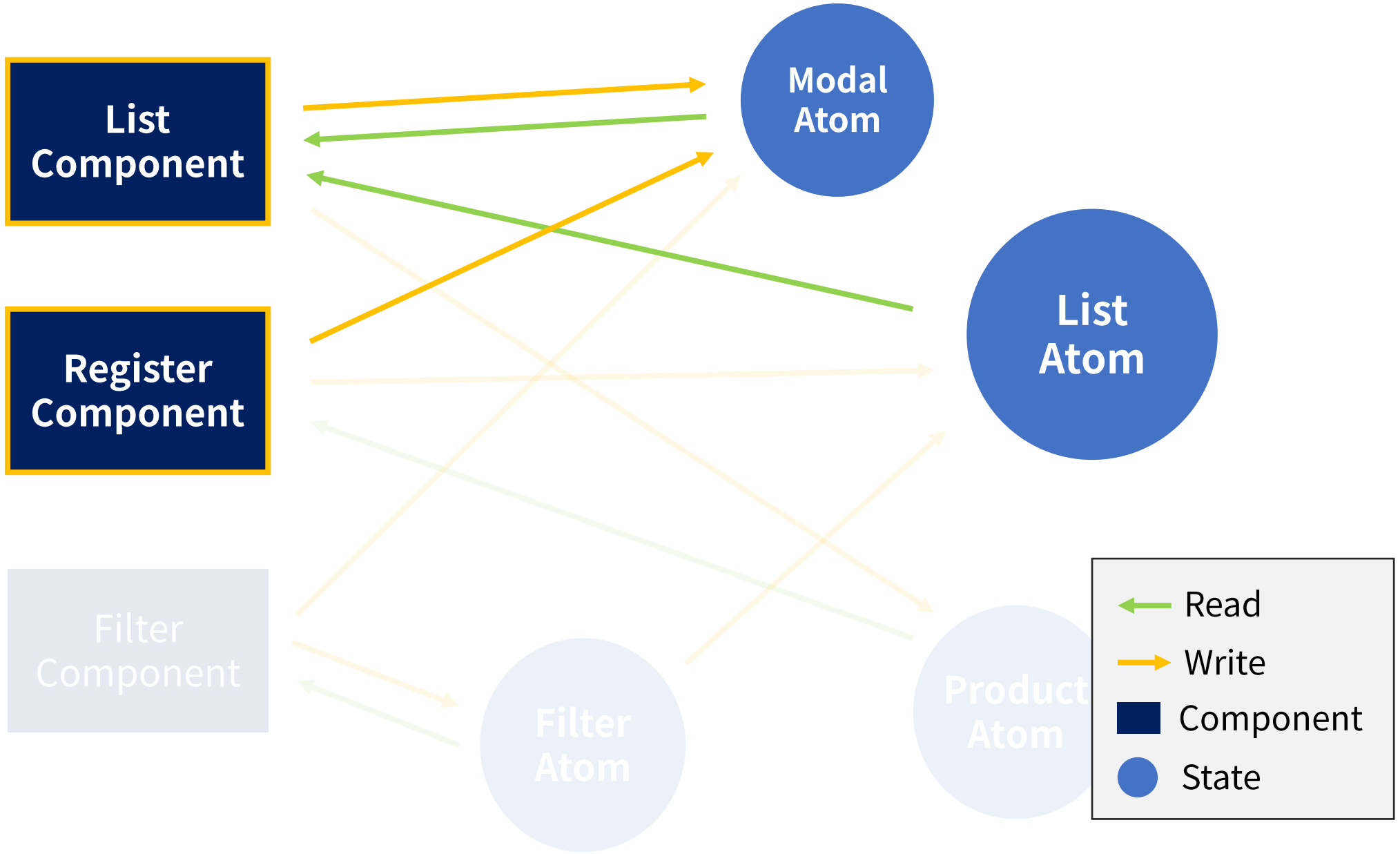


Register



Filtering




List Component
Register Component
Filter Component
Modal Atom
List Atom
Filter Atom
Product Atom
Read
Write
Component
State

Modal

```
import {atom} from 'recoil';

export const modalState = atom({
    key: 'modalState',
    default: false,
});
```

```
import {useRecoilState} from 'recoil';
import {modalState} from '../store/atoms';

function List({data}) {
    const [modalOpen, setModalOpen] = useRecoilState(modalState);

    const handleRegister = () => {
        setModalOpen(true);
    };

    return (
        <button onClick={handleRegister}>상품 추가</article>
        {modalOpen && (<RegisterModal />)}
    );
}

function RegisterModal() {
    const [modalOpen, setModalOpen] = useRecoilState(modalState);

    const handleCancel = () => {
        setModalOpen(false);
    };

    ...
}
```

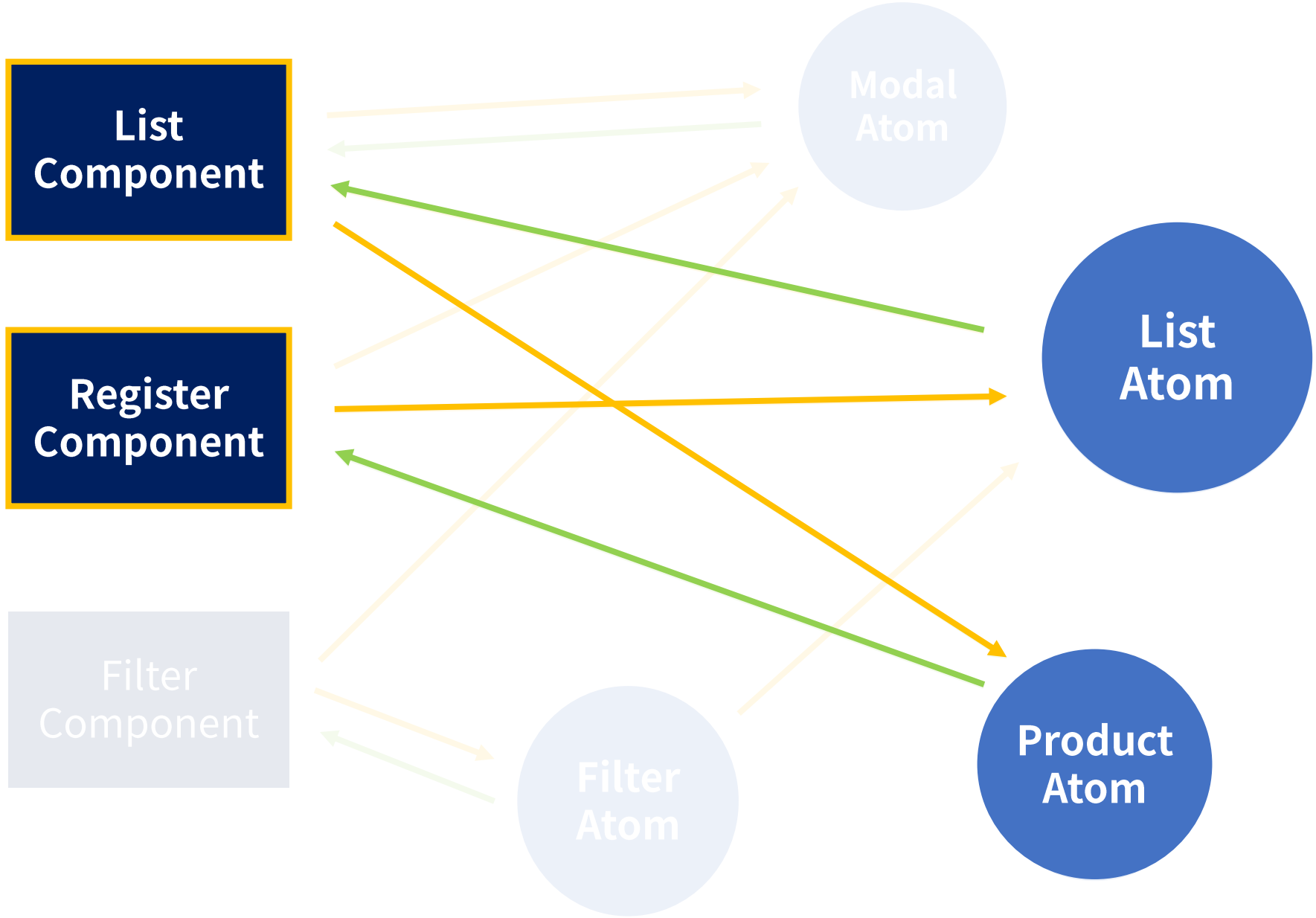
Hooks 처럼 호출해서 사용

List

```
import {atom} from 'recoil';

export const productsState = atom({
    key: 'productsState',
    default: [],
});
```

```
function ProductList() {
    const list = useRecoilValue(productsState);

    return list && (
        <div className="product-list">
            {list.map((row) => (
                <Product key={row.idx} data={row} />
            ))}
        </div>
    );
}

function Product({data}) {
    return (
        data && (
            <article className="product">
                <img src={data.img} alt={data.name} />
                <dl>
                    <dt>[{data.brand}] {data.name}</dt>
                    <dd>{data.category}</dd>
                    <dd>{data.price}원</dd>
                </dl>
```

읽기 전용


List Component
Register Component
Filter Component
Modal Atom
List Atom
Filter Atom
Product Atom

Register

```
function RegisterModal() {
    const [formData, setFormData] = useState();
    const setList = useSetRecoilState(productsState);
    const setIsOpen = useSetRecoilState(modalState);

    const handleChange = (e) => {
        setFormData({
            ...formData,
            [e.target.name]: e.target.value,
        });
    };

    const handleSubmit = async (e) => {
        e.preventDefault();
        setList((prev) => [...prev, formData]);
        setIsOpen(false);
    };
```

쓰기전용

Detail

```
export const productState = atom({
    key: 'productState',
    default: {
        idx: 0,
        name: '',
        category: '',
        brand: '',
        price: 0,
        desc: '',
    },
});
```

```
function Product({data}) {
    const list = useRecoilValue(productsState);
    const setProduct = useSetRecoilState(productState);

    ...

    const handleDetail = (idx) => {
        setProduct(list.filter((row) => row.idx === idx)[0]);
        setRegisterOpen(true);
    };

    return (
        data && (
            <article
              className="product"
              onClick={() => handleDetail(data.idx)}>
                ...
            </article>
        )
    );
}
```

Update

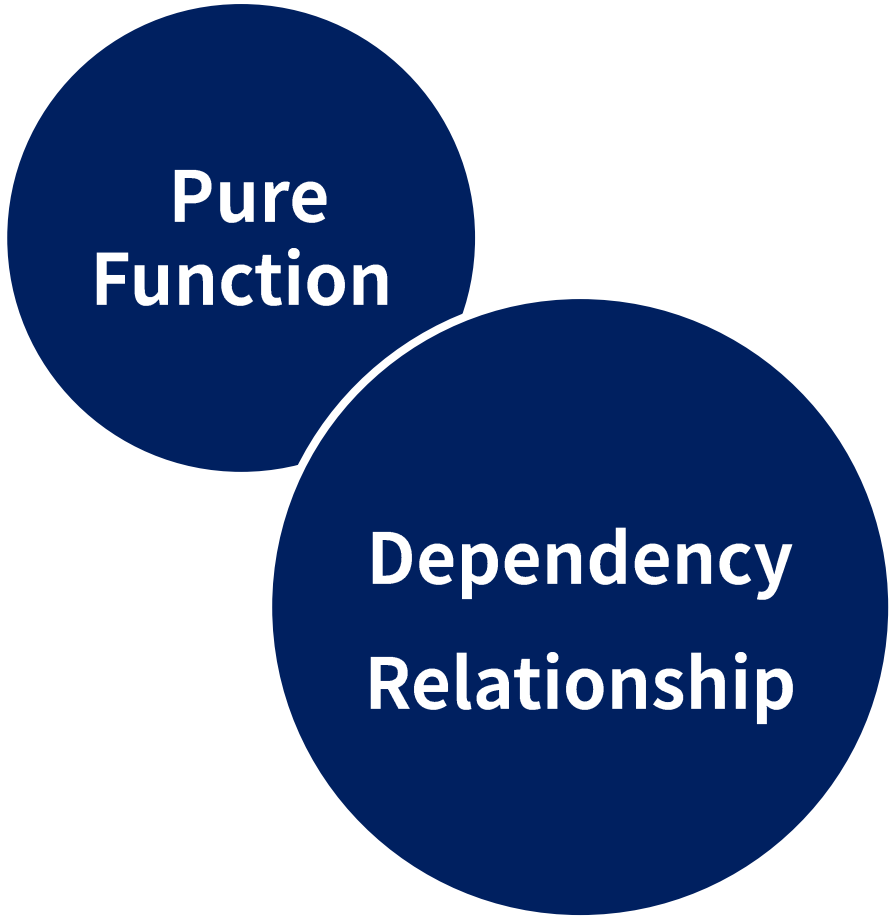
```
function RegisterModal() {
    const [list, setList] = useRecoilState(productsState);
    const product = useRecoilValue(productState);
    const resetProduct = useResetRecoilState(productState);

    ...

    const handleSubmit = async (e) => {
        e.preventDefault();

        if (product) {
            let newList = list.map((row) => {
                if (row.idx === product.idx) {
                    return product;
                } else {
                    return row;
                }
            });
            setList(newList);
        }
        setRegisterOpen(false);
    };
```

```
useEffect(() => {
    return () => {
        resetProduct();
    };
}, []);
```

상태 초기화

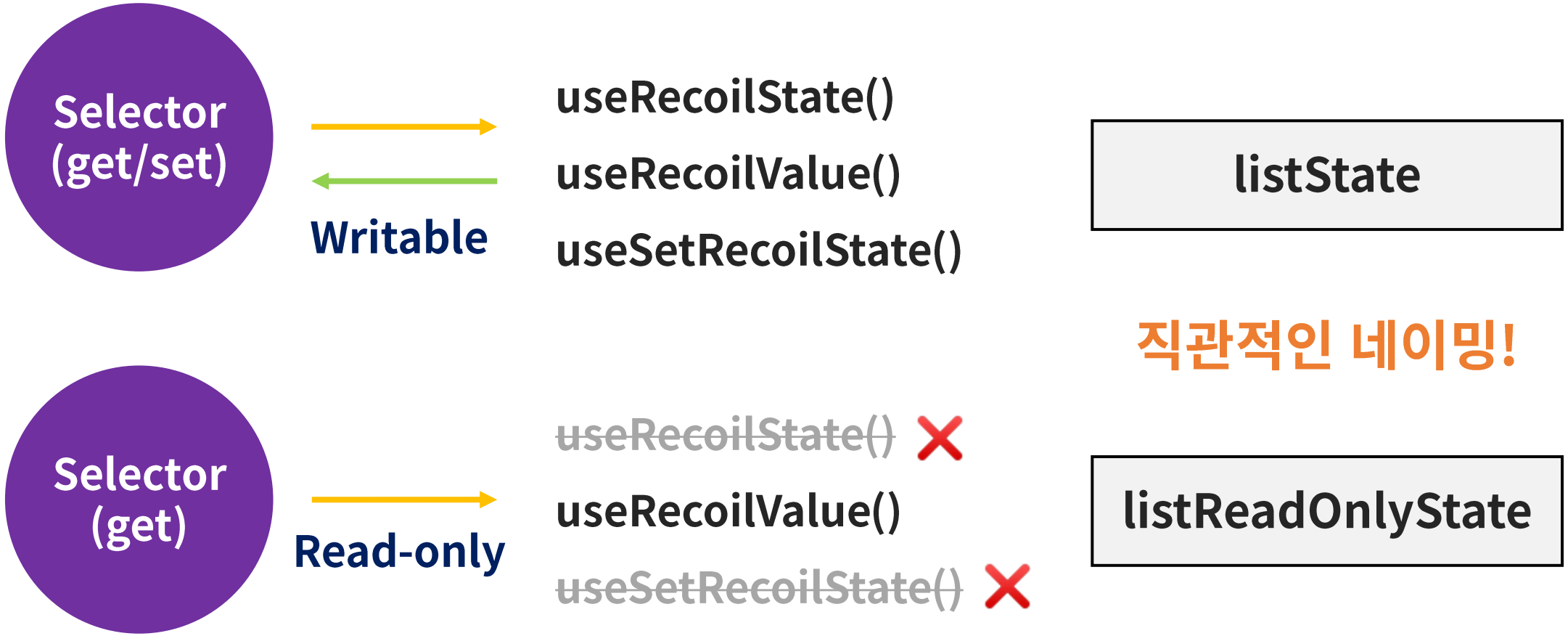
```
...


<input
    type="text"
    name="name"
    defaultValue={product?.name}
    onChange={handleChange}
/>
```

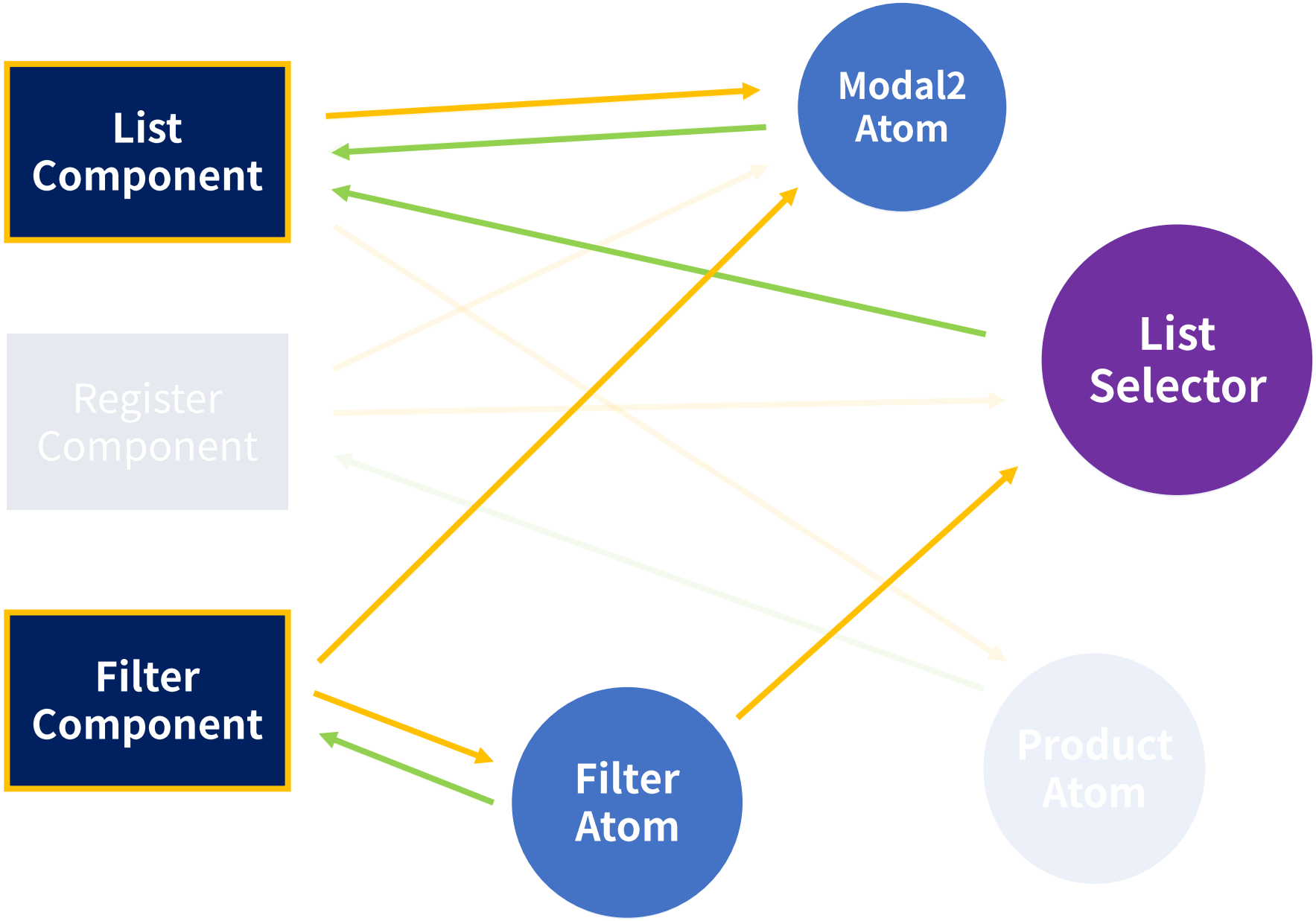
Selector




```
function selector<T>({
  key: string,

  get: ({
    get: GetRecoilValue,
    getCallback: GetCallback,
  }) => T | Promise<T> | RecoilValue<T>,

  set?: (
    {
      get: GetRecoilValue,
      set: SetRecoilState,
      reset: ResetRecoilState,
    },
    newValue: T | DefaultValue,
  ) => void,
```

Read-only / Writable




List Component
Register Component
Filter Component
Modal2 Atom
List Selector
Filter Atom
Product Atom

Filtered List

```
export const filteredProductsState = selector({
    key: 'filteredProductsState',
    get: ({get}) => {
        const filter = get(filterState);
        let list = get(productsState);

        if (filter) {
            if (filter.brand) {
                list = list.filter((row) => filter.brand === row.brand);
            }
            if (filter.category) {
                list = list.filter((row) => filter.category === row.category);
            }
        }
        return list;
    },
});
```

상호 의존성

구독중인 State가 변경

→ Selector 재평가

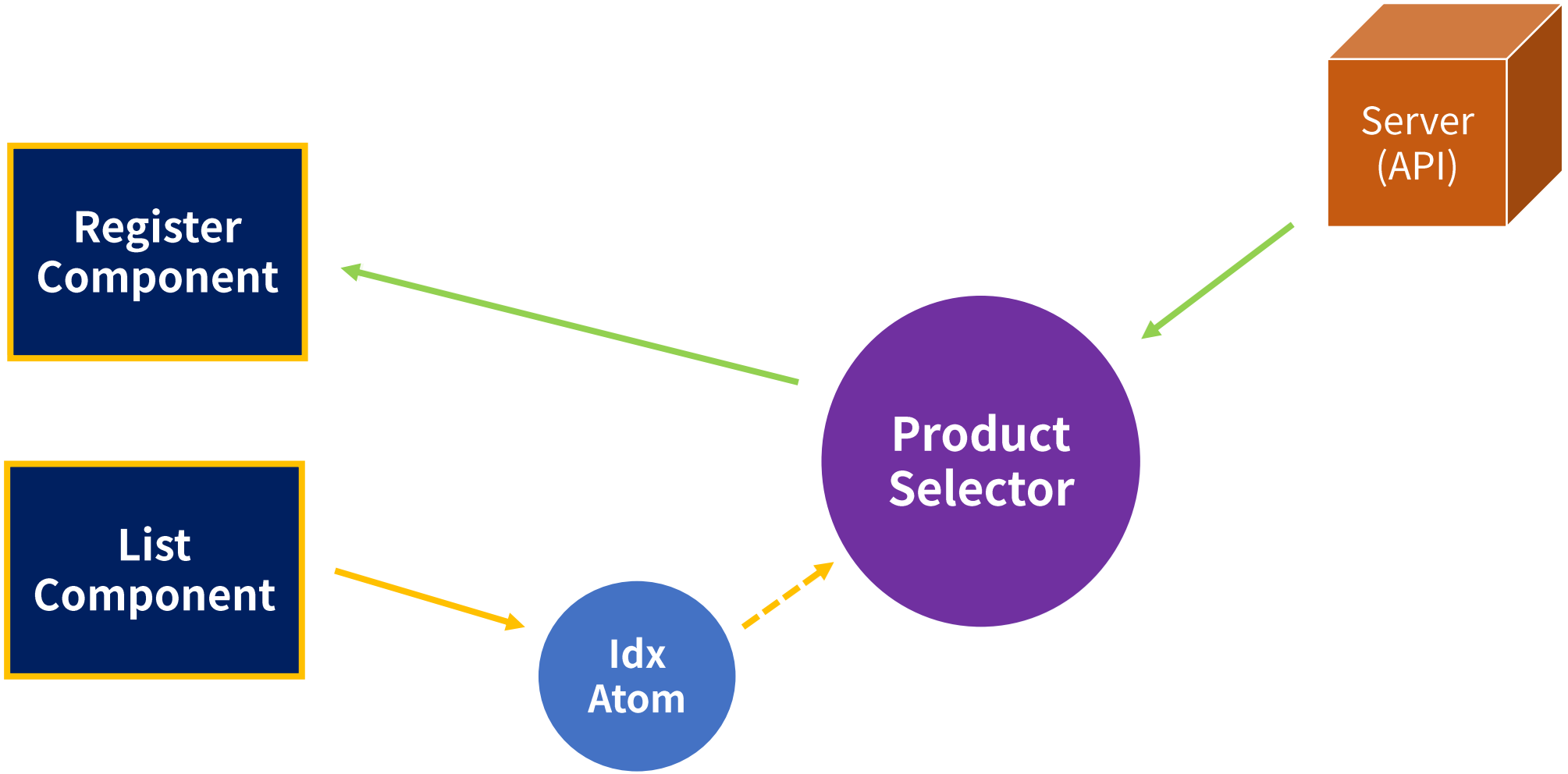
→ 참조 컴포넌트 Re-Render

비동기 데이터 다루기

조금은 불편한 Recoil 비동기 데이터 쿼리

실제 서버 데이터를 가져와 보자

```
const productsAsyncState = selector({
    key: 'productsAsyncState',
    get: async ({get}) => {
        const filter = get(filterState);
        return await getProductList(filter);
    }
});

const getProductList = async (_filter) => {
    let list = [];
    const result = await axios('sample.products.json');
    if (result && result.data) {
        list = result.data;
        // 대충, 선택된 필터 값으로 데이터를 가공하는 로직
    }
    return list;
};
```


Register Component
List Component
Idx Atom
Product Selector
Server (API)

상세 데이터 조회

```
export const productIdxState = atom({
    key: 'productIdxState',
    default: 0,
});

export const productAsyncState = selector({
    key: 'productAsyncState',
    get: async ({get}) => {
        const idx = get(productIdxState);
        return idx > 0 ? await getProductDetail(idx) : null;
    },
});
```

동일한 의존성

Family는 매개변수 전달가능

```
export const productAsyncState = selectorFamily({
    key: 'productAsyncState',
    get: ({idx}) => async ({get}) => {
        return idx > 0 ? await getProductDetail(idx) : null;
    },
});
```

Suspense는 실험적? Recoil에서는 적극 권장

```
▼Uncaught Error: ProductList react-dom.development.js:20349
suspended while rendering, but no fallback UI was specified.

Add a <Suspense fallback=...> component higher in the tree
to provide a loading indicator or placeholder to display.
    at throwException (react-dom.development.js:20349)
    at handleError (react-dom.development.js:22558)
    at renderRootSync (react-dom.development.js:22673)
    at performSyncWorkOnRoot (react-dom.development.js:2229
3)
    at scheduleUpdateOnFiber (react-dom.development.js:2188
1)
    at updateContainer (react-dom.development.js:25482)
    at react-dom.development.js:26021
    at unbatchedUpdates (react-dom.development.js:22431)
```

- **Indicator 컴포넌트를 미리 만들어 두자**
- **State 호출하는 컴포넌트를 감싸면 끝**

  (Suspense가 싫다면 Loadable 이용)

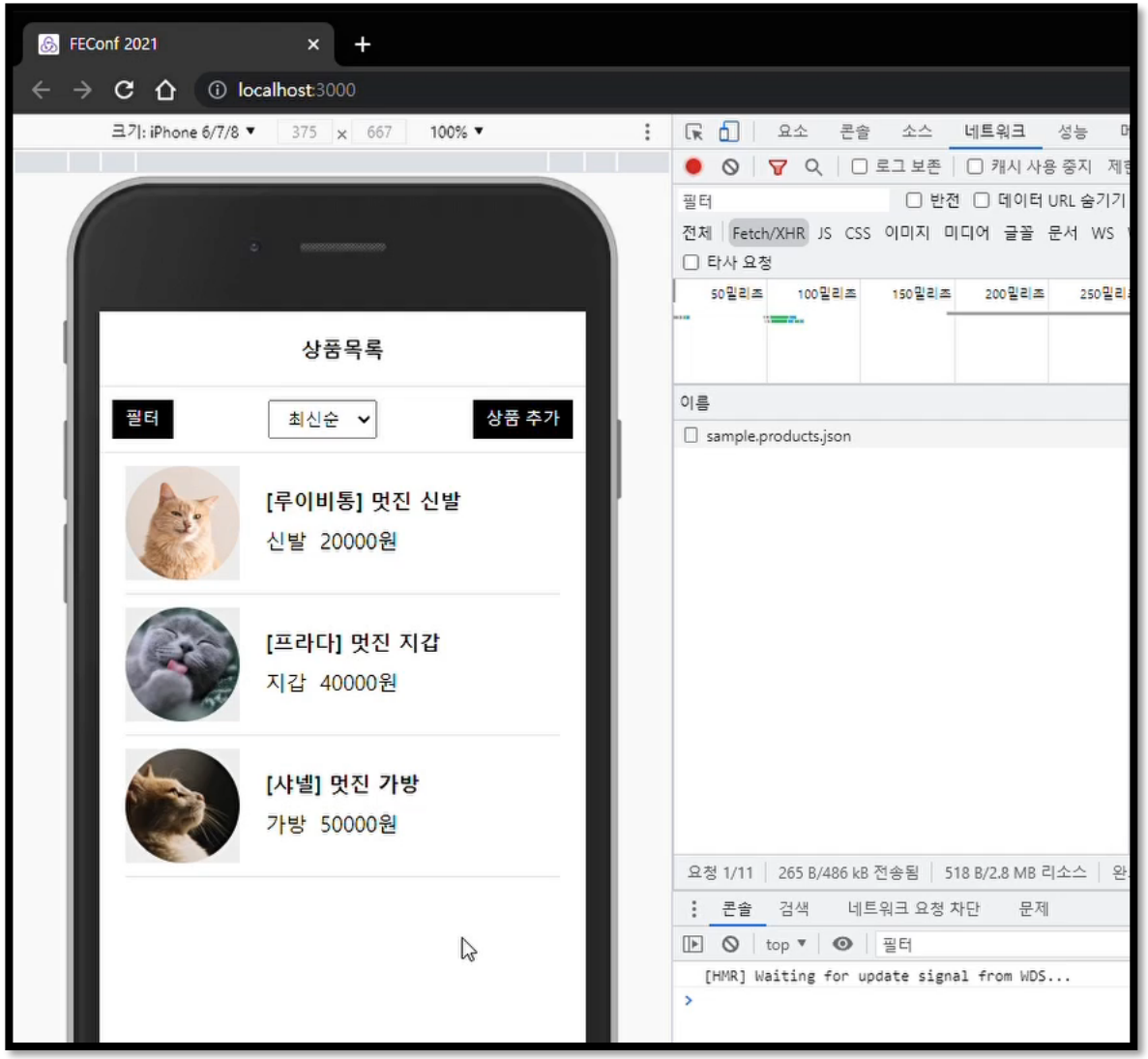
```jsx
function Indicator({type}) {
    return (
        <div className={`indicator ${type}`}>
            Loading...
        </div>
    );
}

...

<section className="body">
    <Suspense fallback={<Indicator type="small" />}>
        <ProductList />
    </Suspense>
</section>

{registerOpen && (
    <Suspense fallback={<Indicator type="full" />}>
        <RegisterModal />
    </Suspense>
)}
```

Error Handling

```
<RecoilRoot>

  <ErrorBoundary>
    <Suspense fallback={<Indicator />}>
      <ProductList />
    </Suspense>
  </ErrorBoundary>

  <ErrorBoundary>
    <Suspense fallback={<Indicator />}>
      <FilterModal />
    </Suspense>
  </ErrorBoundary>

</RecoilRoot>
```

```
const listLoadable =
      useRecoilValueLoadable(listAsyncState);

switch (listLoadable.state) {
  case 'hasValue':
    return listLoadable.contents;
  case 'loading':
    return <Indicator />;
  case 'hasError':
    throw <Error />;
}
```

한 번 사용한 API를 다시 호출한 경우



"오... 한번 조회했던 데이터는
서버 요청 자체를 하지 않는군"

"트래픽 비용이 엄청 절약되겠는걸!"

"잠깐만...
이 데이터는 매번 새로운
값을 보여줘야 하는데…"

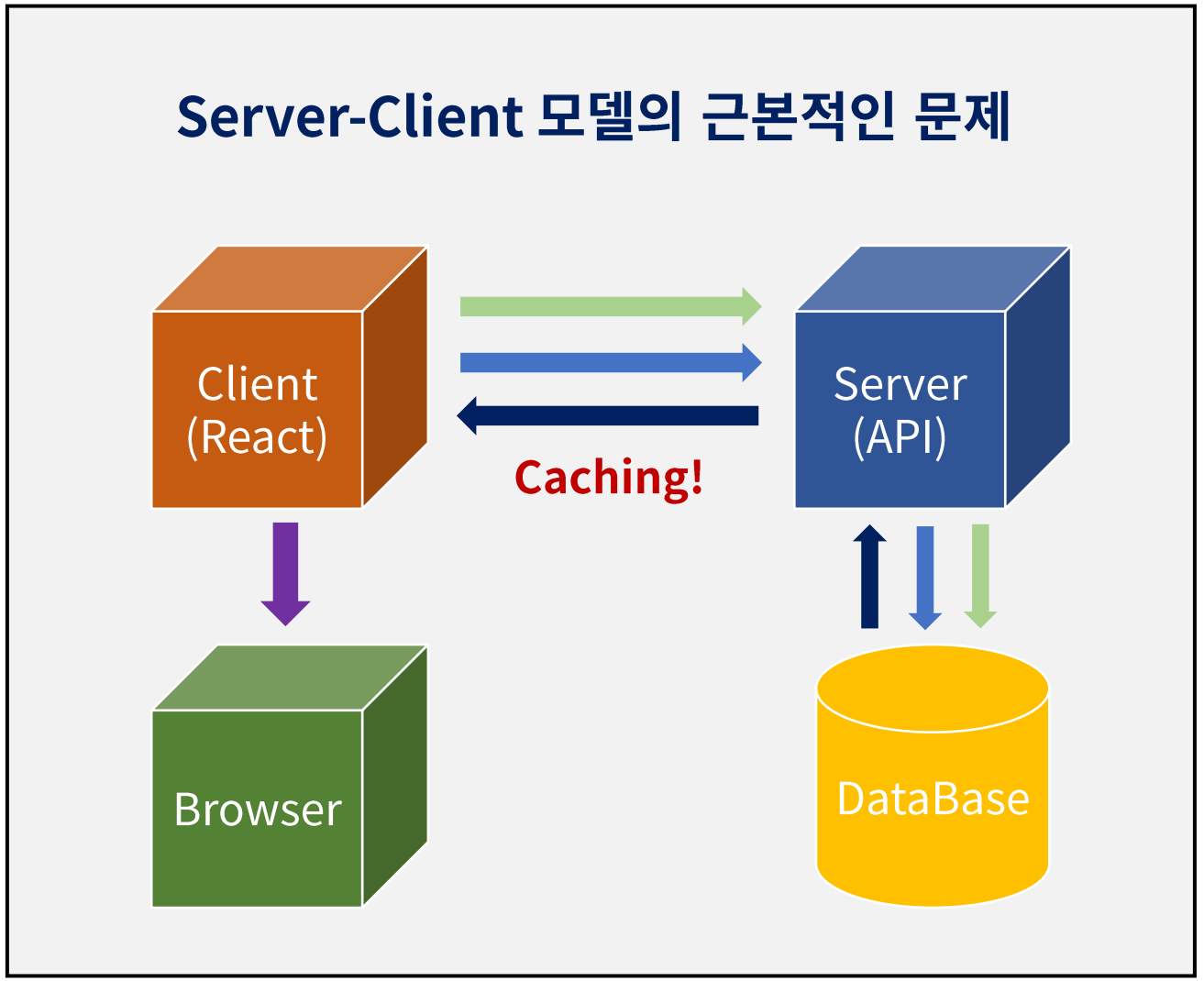
"갱신하는 방법이 어디있더라..."

동일한 데이터를 여러 API에서 사용하는 경우

찜한 상품이 마이페이지에 반영이 안되요!

다른 사람이 추가한 게시글이 내 앱에는 안보여요!

배송중이라는 문자를 받았는데 앱에서는 아직도 준비중이네요?



비동기 데이터를 갱신하는 방법들

점점 더 편해질 것이라 믿어요

비동기 데이터 갱신에 영향을 주는 요소

내부에서 구독중인

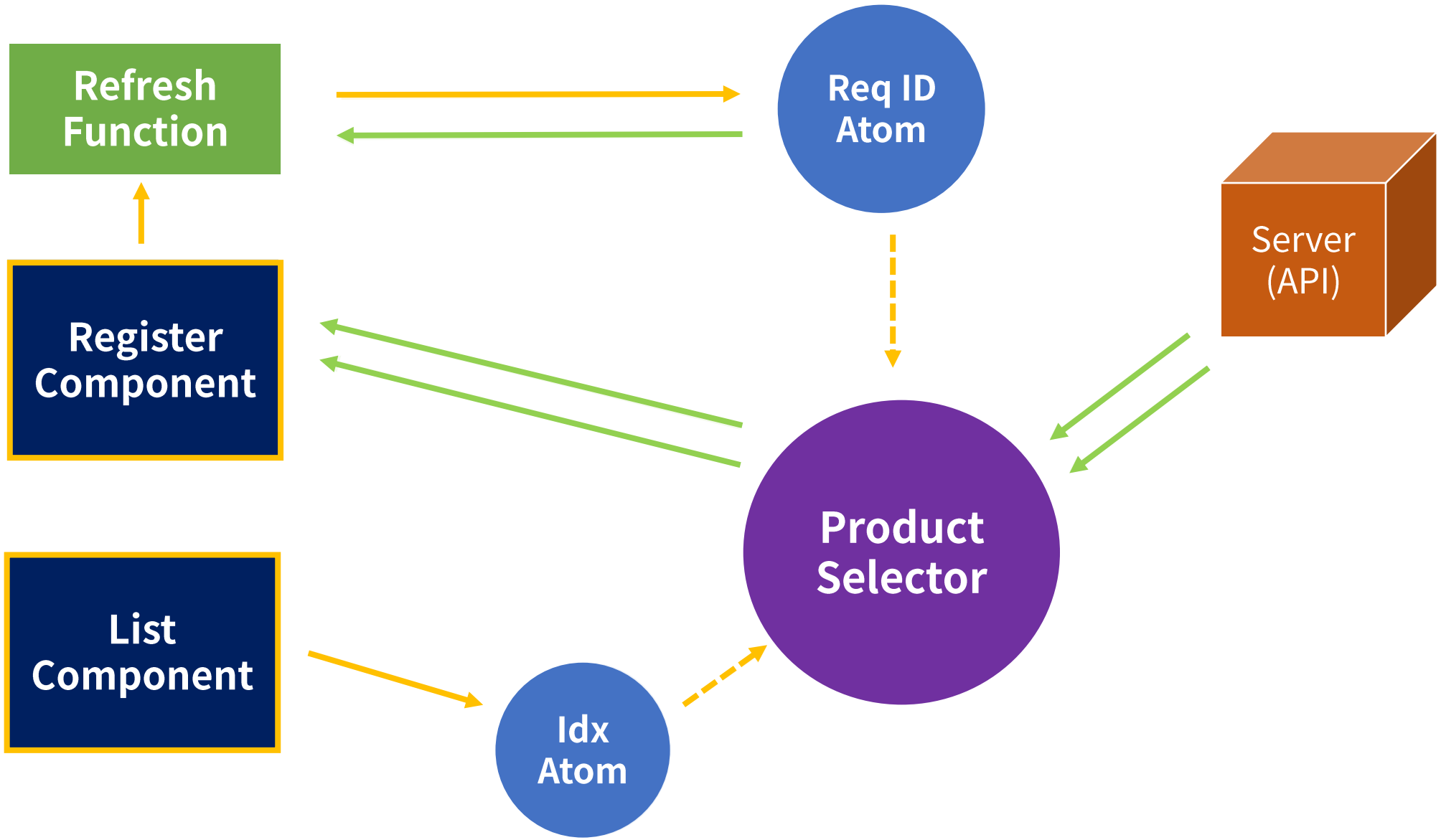
다른 Recoil state의

변경을 감지한 경우

요청 파라메터가

완전 새로운 값으로

변경된 경우


Refresh Function
Req ID Atom
Server (API)
Register Component
Product Selector
List Component
Idx Atom

Recoil이 제안한 Request ID를 이용한 명시적 갱신

```
export const productReqIDState = atom({
    key: 'productReqIDState',
    default: 1,
});

export const productAsyncState = selector({
    key: 'productAsyncState',
    get: async ({get}) => {
        const idx = get(productIdxState);
        get(productReqIDState);
        return idx > 0 ? await getProductDetail(idx) : null;
    },
});
```

의존성 주입

```
function useRefreshProductAsyncState() {
    const setProductID = useSetRecoilState(productReqIDState);
    return () => {
        setProductID((id) => id + 1);
    };
}
```

Refresh 용 함수

```
function RegisterModal() {
    const refreshProductState = useRefreshProductAsyncState();

    const handleSubmit = async (e) => {
        e.preventDefault();

        if (data.idx > 0) {
            /* 수정하는 로직 */

            refreshProductState();
        }
        setRegisterOpen(false);
    };

    {/* 중략 */}
```

Setter를 활용한 개선 버전

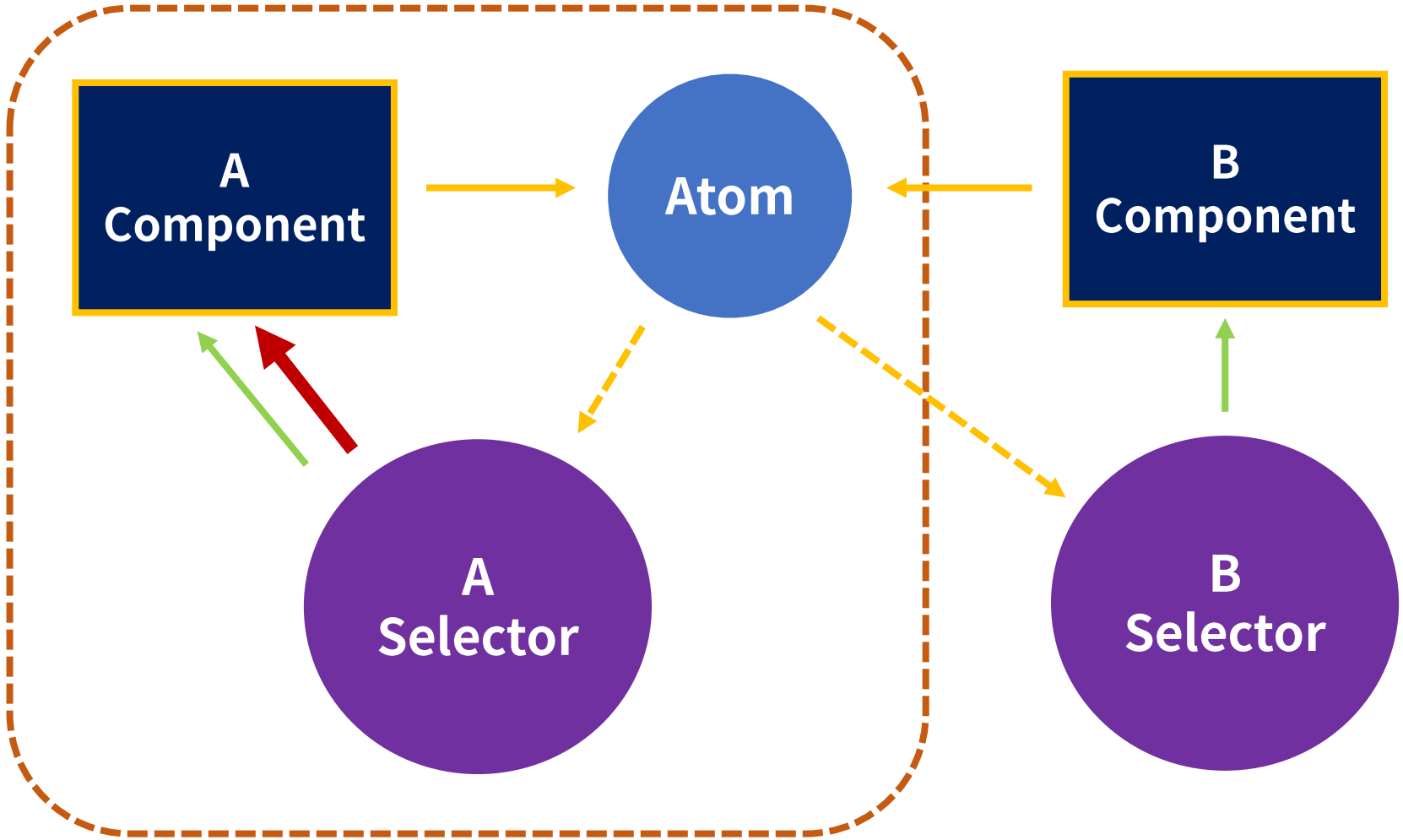
```
export const productReqIDState = atom({
    key: 'productReqIDState',
    default: 1,
});

export const productAsyncState = selector({
    key: 'productAsyncState',
    get: async ({get}) => {
        const idx = get(productIdxState);
        get(productReqIDState);
        return idx > 0 ? await getProductDetail(idx) : null;
    },
    set: ({set}) => {
        set(productReqIDState, (id) => id + 1);
    },
});
```

Refresh Hooks 대신 setter

```
const [productState, setProductState] =
        useRecoilState(productAsyncState);

const handleSubmit = async (e) => {
    e.preventDefault();

    if (data.idx > 0) {
        /* 수정하는 로직 */

        setProductState(productState);
    }
    setRegisterOpen(false);
};
```

상태 변경이 없는 set 실행

Atom 구독 공유 시 주의점



- Private으로 정의
- RecoilRoot 분리
- Family로 Grouping

다른 방법은 없나요...?

Use an Atom

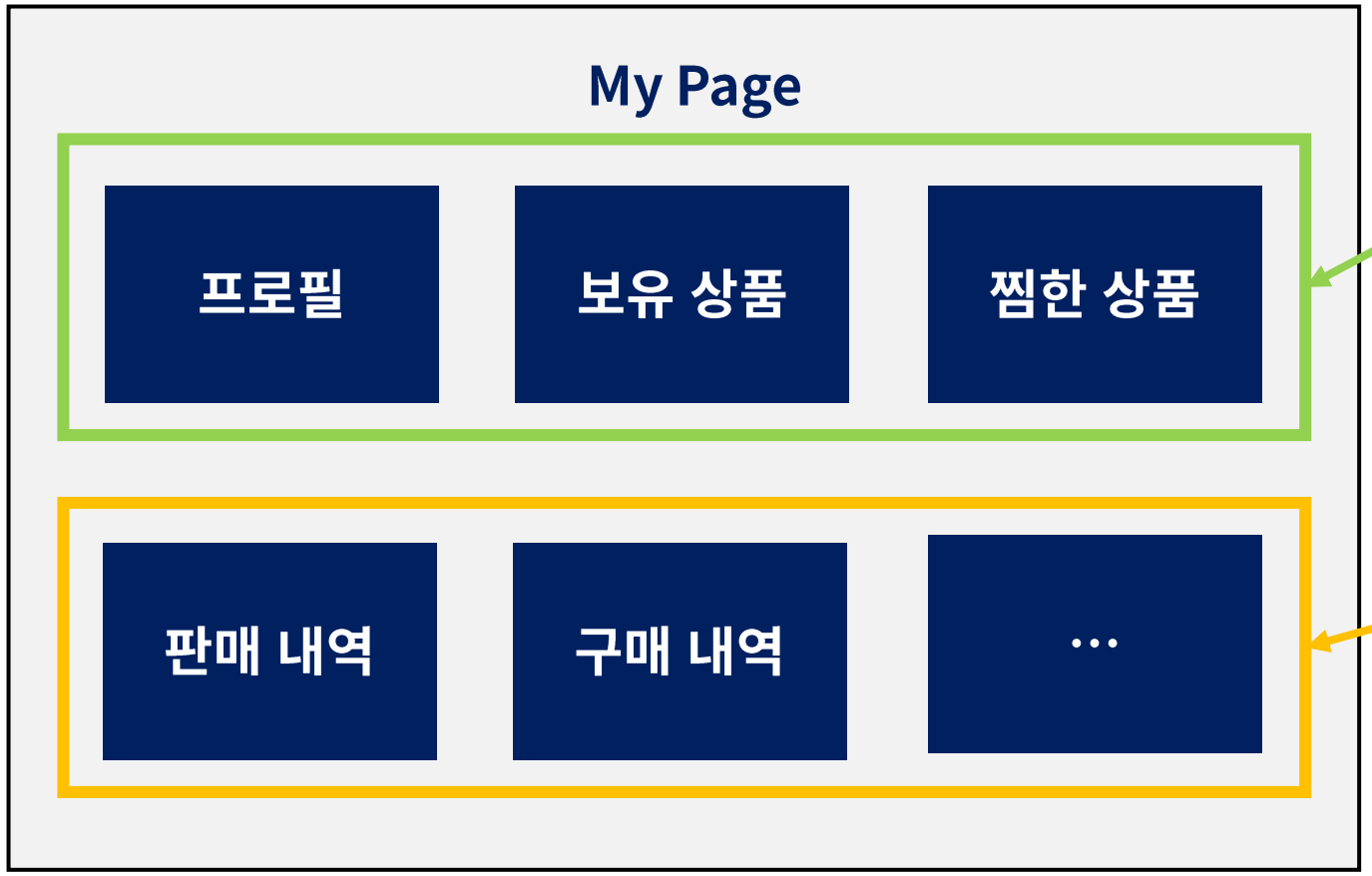
Another option is to use an atom, instead of a selector, to model the query results. You can imperatively update the atom state with the new query results based on your refresh policy.

```
const userInfoState = atomFamily({
  key: 'UserInfo',
  default: userID => fetch(userInfoURL(userID)),
});

// React component to refresh query
function RefreshUserInfo({userID}) {
  const refreshUserInfo = useRecoilCallback(({set}) => async id => {
    const userInfo = await myDBQuery({userID});
    set(userInfoState(userID), userInfo);
  }, [userID]);

  // Refresh user info every second
  useEffect(() => {
    const intervalID = setInterval(refreshUserInfo, 1000);
    return () => clearInterval(intervalID);
  }, [refreshUserInfo]);

  return null;
}
```

Selector 대신 Atom으로 데이터 관리

useEffect에 설정한 조건에 따라
useRecoilCallback 을 이용해
데이터 갱신하는 로직

명시적 / 주기적 업데이트



내 Event만 영향주는 데이터

→ 명시적 업데이트 필요

언제 변경될지 모르는 데이터

→ 실시간/주기적 업데이트

or **No-Cache** or **Recoil X**

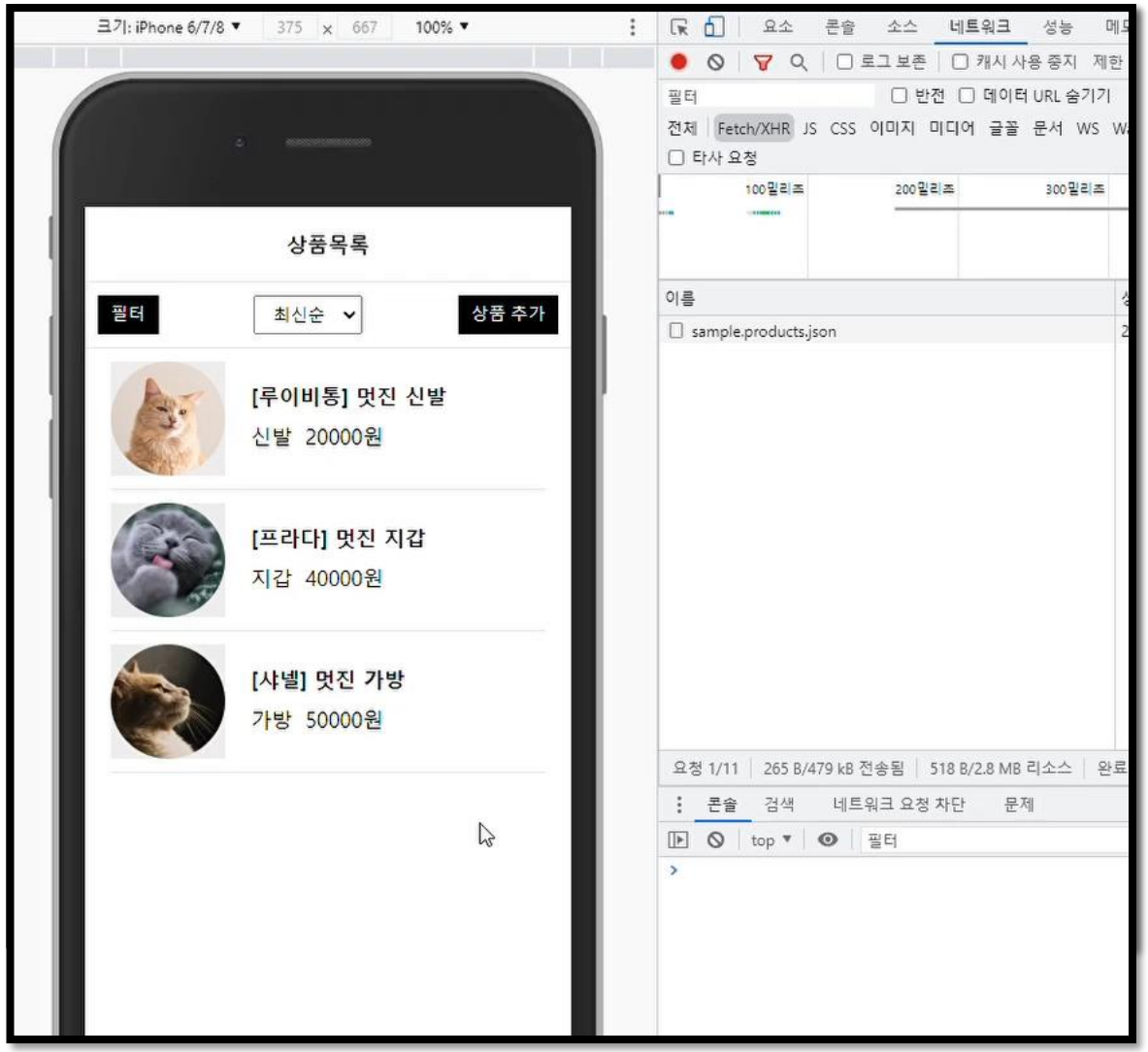
Version Param 을 이용한 주기적 갱신

```
export const productAsyncState = selectorFamily({
    key: 'productAsyncState',
    get:
        ({ver}) =>
        async ({get}) => {
            const idx = get(productIdxState);
            return idx > 0 ? await getProductDetail(idx, ver) : null;
        },
});
```

```
function getVersion() {
    const now = new Date();
    const Y = now.getFullYear(),
        M = now.getMonth(),
        D = now.getDate(),
        h = now.getHours(),
        m = now.getMinutes(),
        s = now.getSeconds();
    return `${Y}${M}${D}${h}${m}${s}`;
}

const data = useRecoilValue(
    productAsyncState({ver: getVersion()})
);
```

- 고전적이지만 확실한 Cache Control
- 1분 / 30초 / 1시간 세부 설정 가능

Version Param 을 이용한 주기적 갱신



- useEffect로 version 변경

  → 데이터 변경 시점 차이 발생

  → 더블링 발생!

- New Data(), Timestamp

  → 무한 요청..

손꼽아 기다렸어요

Recoil 0.4

July 30, 2021 · 4 min read

configurable selector caches, improved
izations and fixes.

s

d selector families allows you to configure
roperty can be useful for reducing memory
electors that have a large number of

erty:

Configurable selector caches

The new `cachePolicy_UNSTABLE` property in selectors and selector families allows you to configure the caching behavior of a selector's internal cache. This property can be useful for reducing memory in applications that have a large number of selectors or selectors that have a large number of changing dependencies.

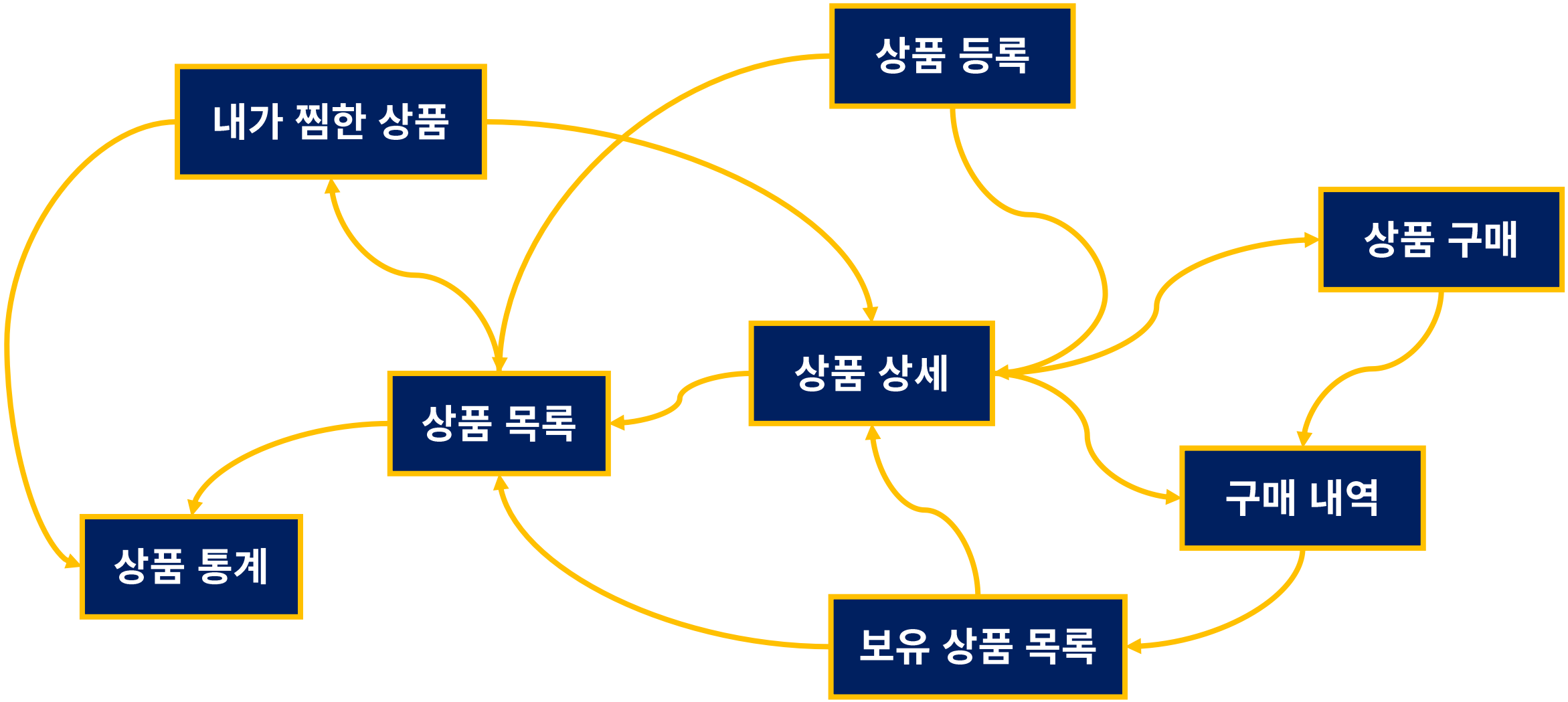
```
const clockState = selector({
  key: 'clockState',
  get: ({get}) => {
    const hour = get(hourState);
    const minute = get(minuteState);
    const second = get(secondState); // will re-run every second

    return `${hour}:${minute}:${second}`;
  },
  cachePolicy_UNSTABLE: {
    // Only store the most recent set of dependencies and their values
    eviction: 'most-recent',
  },
});
```

왜 아직 실험적 인가요... 😭

빨리 릴리즈 해주세여. 현기증 난단 말이에요..

마무리

Recoil 앞으로 더 기대됩니다


상품 등록
내가 찜한 상품
상품 구매
상품 상세
상품 목록
구매 내역
상품 통계
보유 상품 목록

초기 설계 시 고려할 점

- ✓ **꼭 전역으로 관리되야만 하는가?**

  → 간단한 건 Hooks 와 Props 로 충분

- ✓ **여러 상태값을 사용 용도에 따라 분류**

  → UI 전용 상태 / 폼 데이터 처리 상태 / 서버 데이터 조회용 상태

- ✓ **어떤 데이터를 캐싱하고, 실시간으로 반영되야 하는지 파악**

  → 캐싱: 코드성 데이터, 사용자 정보 등 / 주기적: 자주 바뀌지 않는 정보 / 실시간: 주요 데이터

- ✓ **데이터가 어느 시점에서 변경되고 어떤 부분에 영향을 주는지 예측**

그 밖에 유용한

- **constSelector** : 상수 전용 셀렉터
- **errorSelector :** 에러 전용 셀렉터
- **Multiple <RecoilRoot>** : 영역 분리와 중첩(상속 가능)
- **Concurrent Request - waitFor*()** : 비동기 데이터 동시 병렬요청
- **Pre-Fetching - Snapshot** : 랜더링 전 미리 데이터 조회
- 그 밖에 실험적 기능들..

마지막으로..

WE ARE HIRING

- Job Description : https://bit.ly/3zL6GXP
- Contact : hr@mass-adoption.com

시청해 주셔서 대단히 고맙습니다

- Recoil Homepage : https://recoiljs.org
- Demo & Source : https://codesandbox.io/s/woody-feconf2021-dwi9r

FE CONF

MASS ADOPTION

Chief Technology Officer

김 현 태 (Woody Kim)

 woody@mass-adoption.com

 www.linkedin.com/in/0gusxo0

 www.facebook.com/0gusxo0

 www.instagram.com/0gusxo0