How to reduce unnecessary WebSocket reconnections in React

JongChan Choi

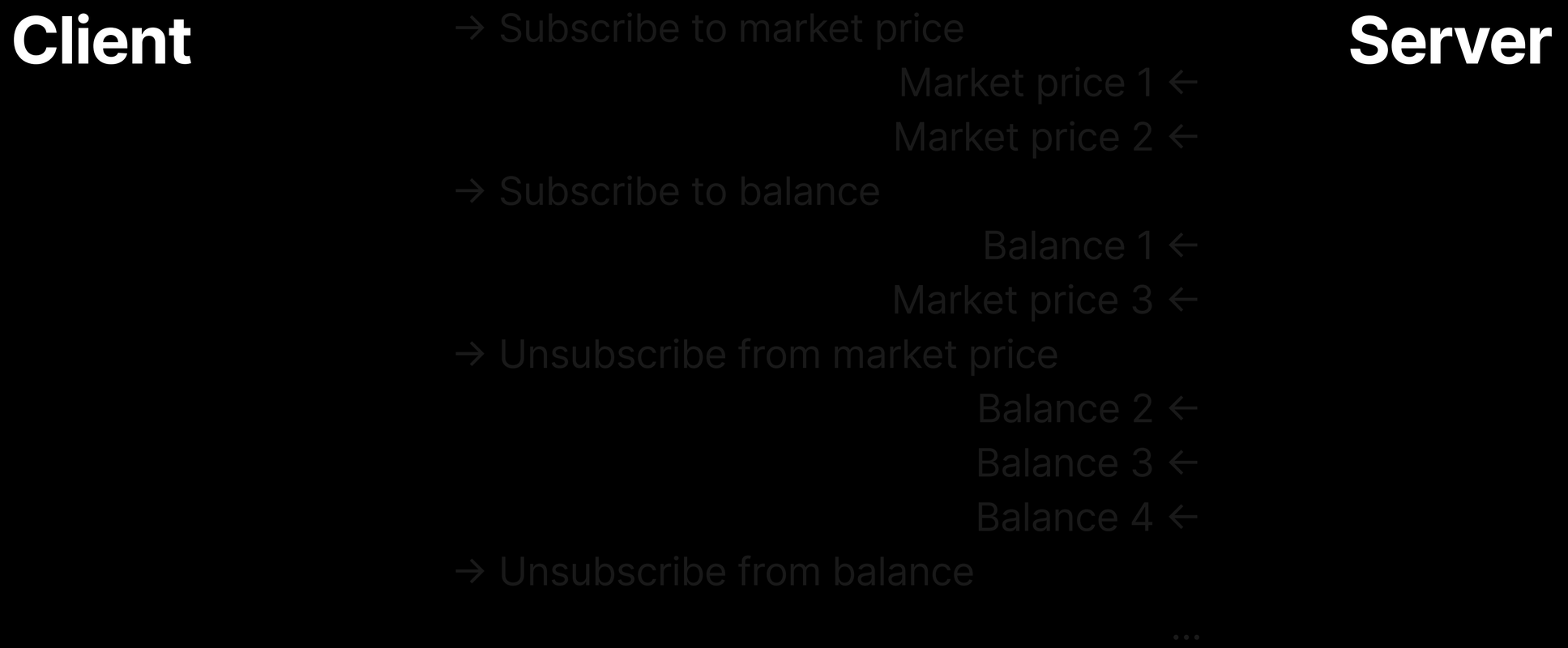

Flipster


Flipster
WebSocket

WebSocket?

- Unlike the “request-response” exchange of information in an HTTP API, messages can be freely exchanged between the client and server while connected.
- It can be designed to subscribe to multiple topics within a single WebSocket connection.
- It is necessary to manage the lifetime (start and end) of WebSocket connections and topic subscriptions.

The content exchanged in a WebSocket



The content exchanged in a WebSocket

Client

→ Subscribe to market price

Market price 1 ←

Market price 2 ←

→ Subscribe to balance

Balance 1 ←

Market price 3 ←

→ Unsubscribe from market price

Balance 2 ←

Balance 3 ←

Balance 4 ←

→ Unsubscribe from balance

...

Server

The content exchanged in a WebSocket

Client

→ Subscribe to market price

Market price 1 ←

Market price 2 ←

→ Subscribe to balance

Balance 1 ←

Market price 3 ←

→ Unsubscribe from market price

Balance 2 ←

Balance 3 ←

Balance 4 ←

→ Unsubscribe from balance

...

Server

The content exchanged in a WebSocket

Client **Server**

→ Subscribe to market price

Market price 1 ←

Market price 2 ←

→ Subscribe to balance

Balance 1 ←

Market price 3 ←

→ Unsubscribe from market price

Balance 2 ←

Balance 3 ←

Balance 4 ←

→ Unsubscribe from balance

...

The content exchanged in a WebSocket

Client **Server**

→ Subscribe to market price

Market price 1 ←

Market price 2 ←

→ Subscribe to balance

Balance 1 ←

Market price 3 ←

→ Unsubscribe from market price

Balance 2 ←

Balance 3 ←

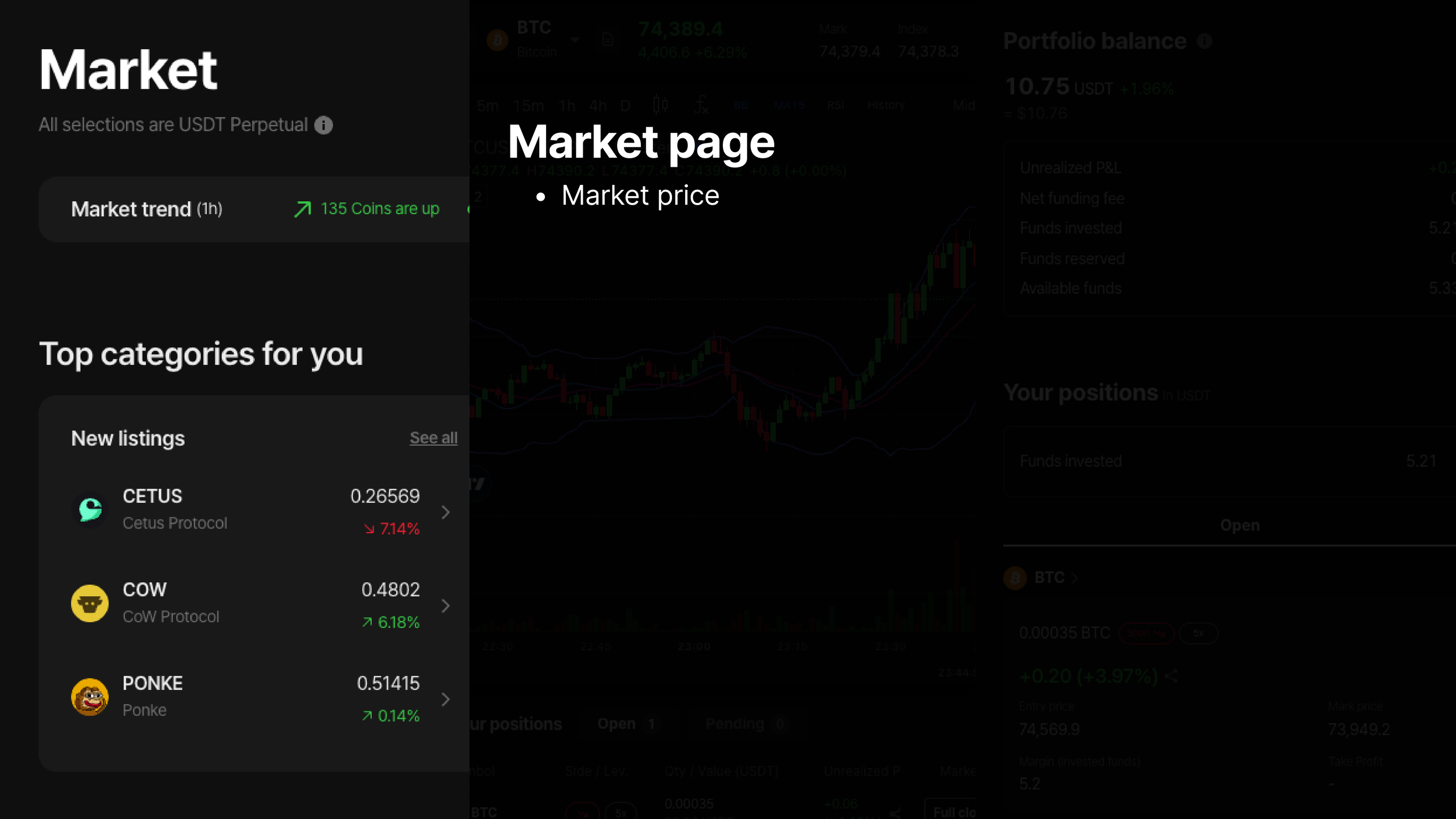
Balance 4 ←

→ Unsubscribe from balance

...

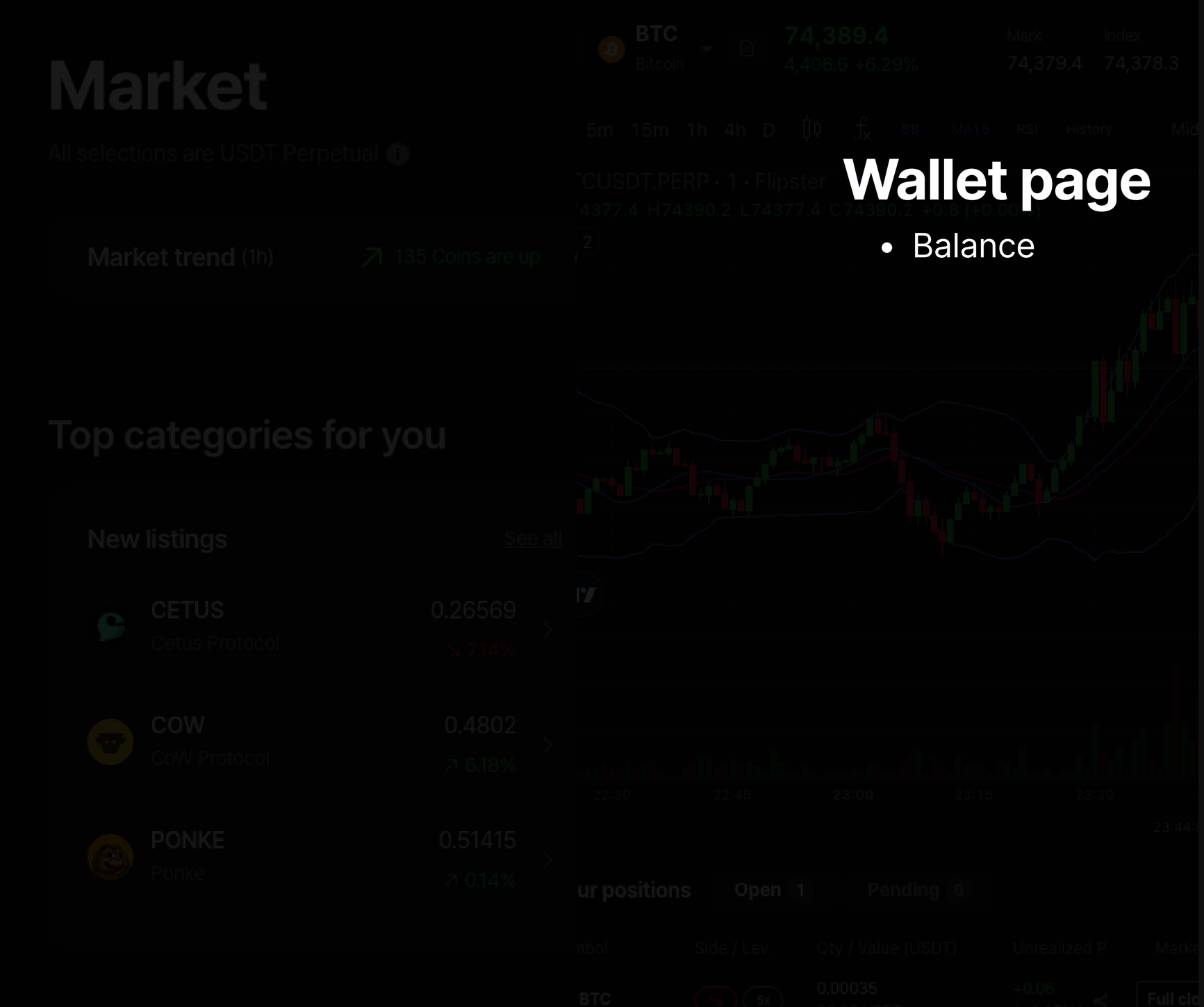
Market page

- Market price



Trade page

- Chart
- Balance



Wallet page

- Balance

Portfolio balance

10.75 USDT +1.96%

≈ $10.76

Unrealized P&L	+0.2
Net funding fee	0
Funds invested	5.21
Funds reserved	0
Available funds	5.33

Your positions In USDT

Funds invested	5.21

Open

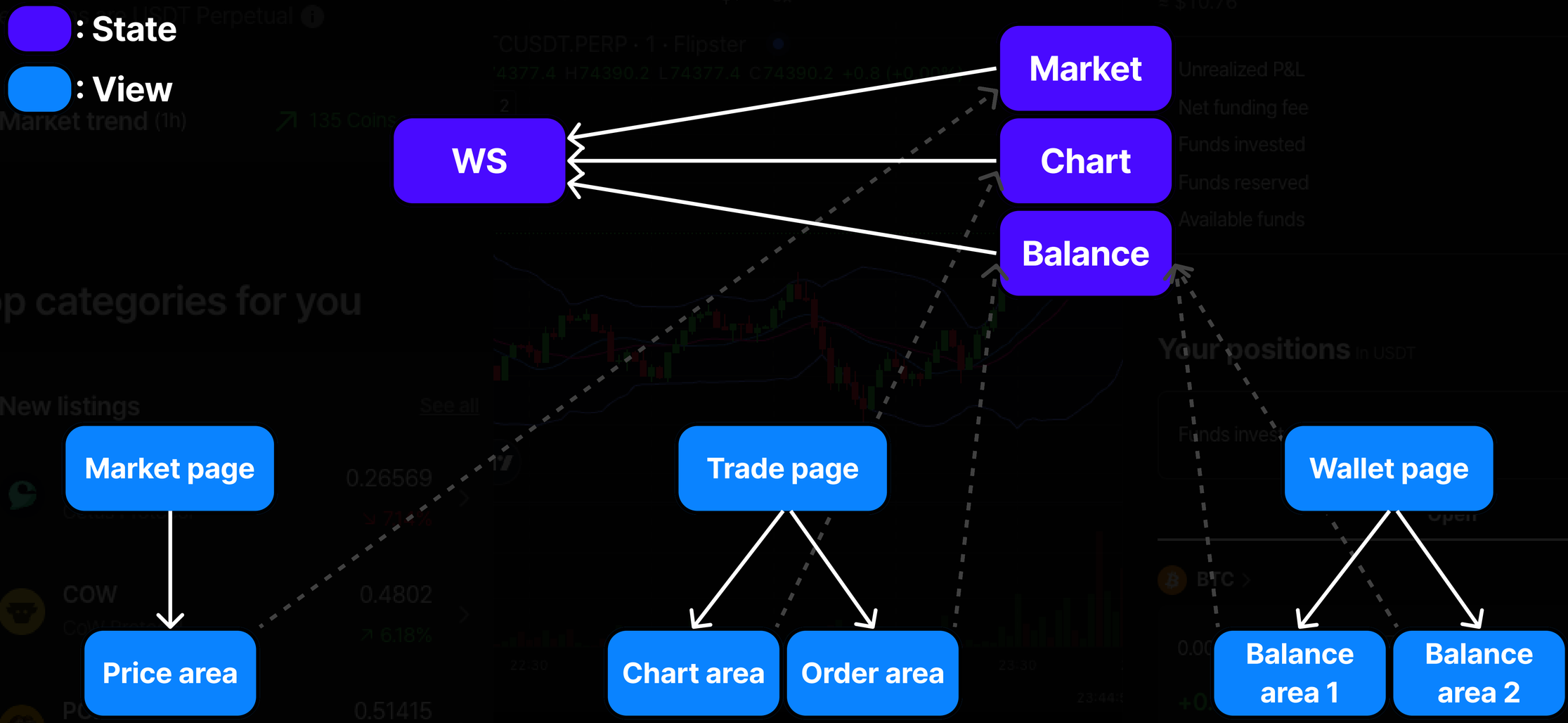
BTC

0.00035 BTC Short 5x

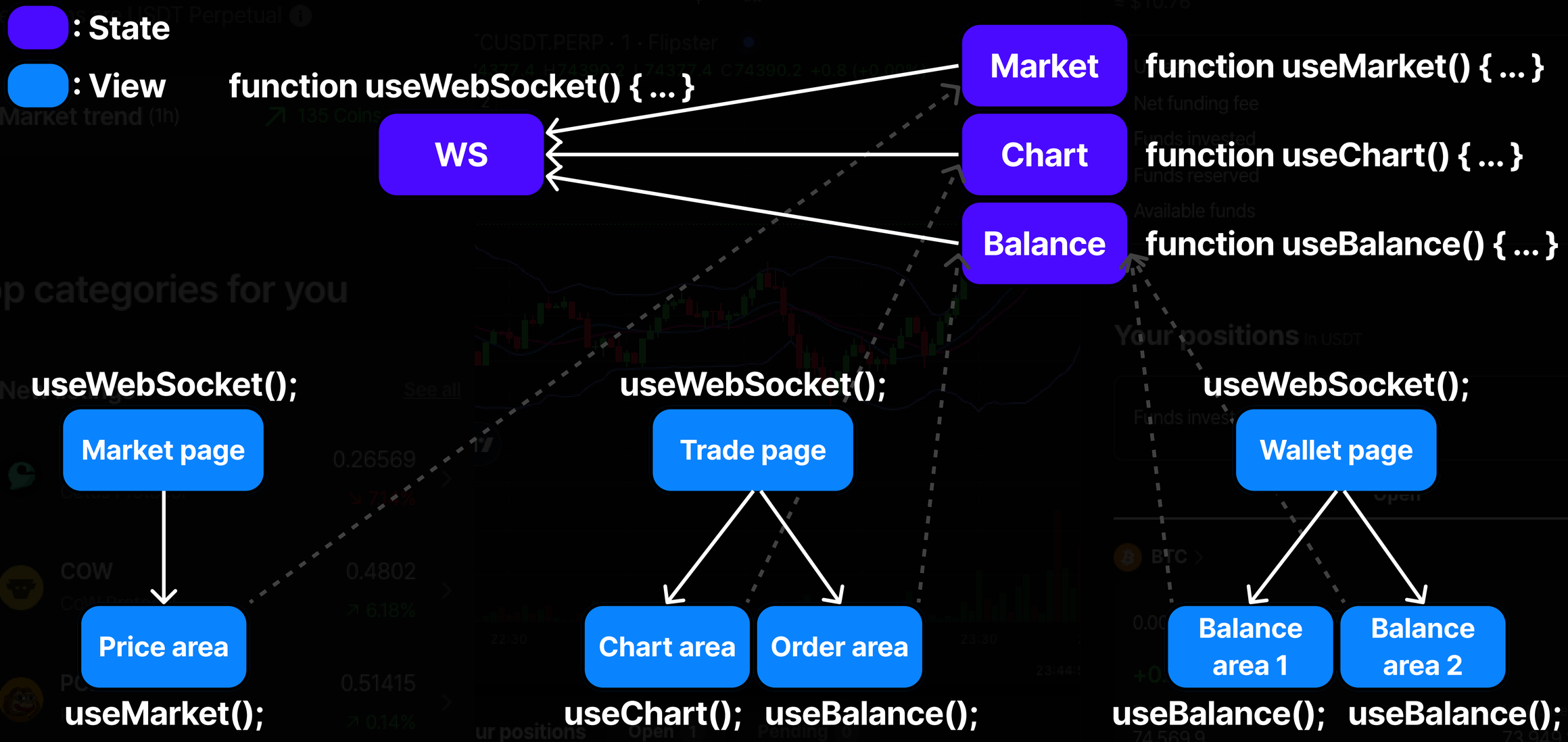
+0.20 (+3.97%)

Entry price	Mark price
74,569.9	73,949.2
Margin (Invested funds)	Take Profit
5.2	-

State-view dependency



Original code (naive implementation)



Original code (naive implementation)

Market page

```
function MarketPage() {
 useWebSocket();
 return <PriceArea />;
}

function PriceArea() {
 useMarket();
 return ...;
}
```

Trade page

```
function TradePage() {
 useWebSocket();
 return <>
  <ChartArea />
  <OrderArea />
 </>;
}

function ChartArea() {
 useChart();
 return ...;
}
function OrderArea() {
 useBalance();
 return ...;
}
```

Wallet page

```
function WalletPage() {
 useWebSocket();
 return <>
  <BalanceArea1 />
  <BalanceArea2 />
 </>;
}

function BalanceArea1() {
 useBalance();
 return ...;
}
function BalanceArea2() {
 useBalance();
 return ...;
}
```

Original code (naive implementation)

WS

```
function useWebSocket() {
 useEffect(() => {
  connectWS();
  return () => disconnectWS();
 });
}
```

Market

```
function useMarket() {
 useEffect(() => {
  subMarket();
  return () => unsubMarket();
 });
}
```

Chart

```
function useChart() {
 useEffect(() => {
  subChart();
  return () => unsubChart();
 });
}
```

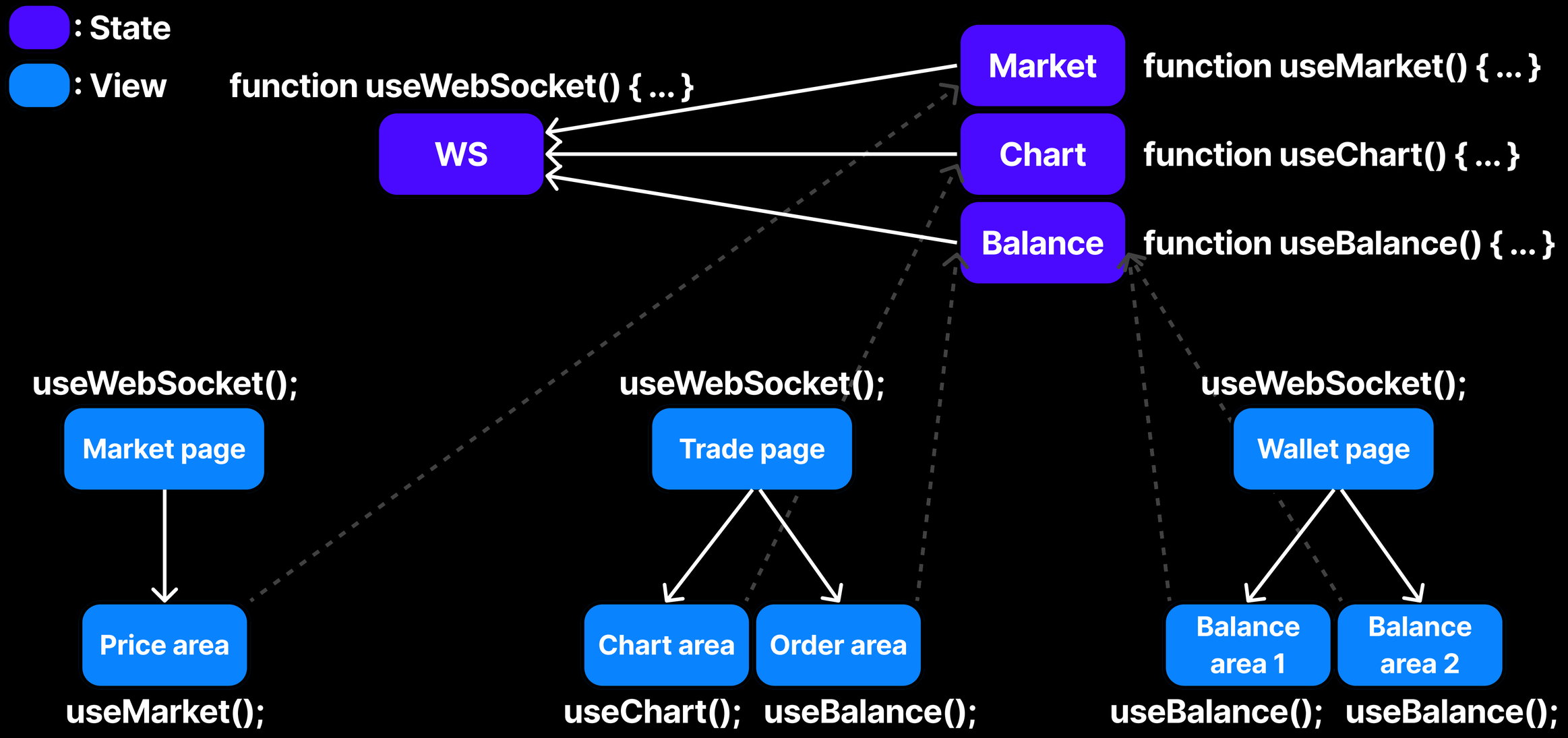
Balance

```
function useBalance() {
 useEffect(() => {
  subBalance();
  return () => unsubBalance();
 });
}
```

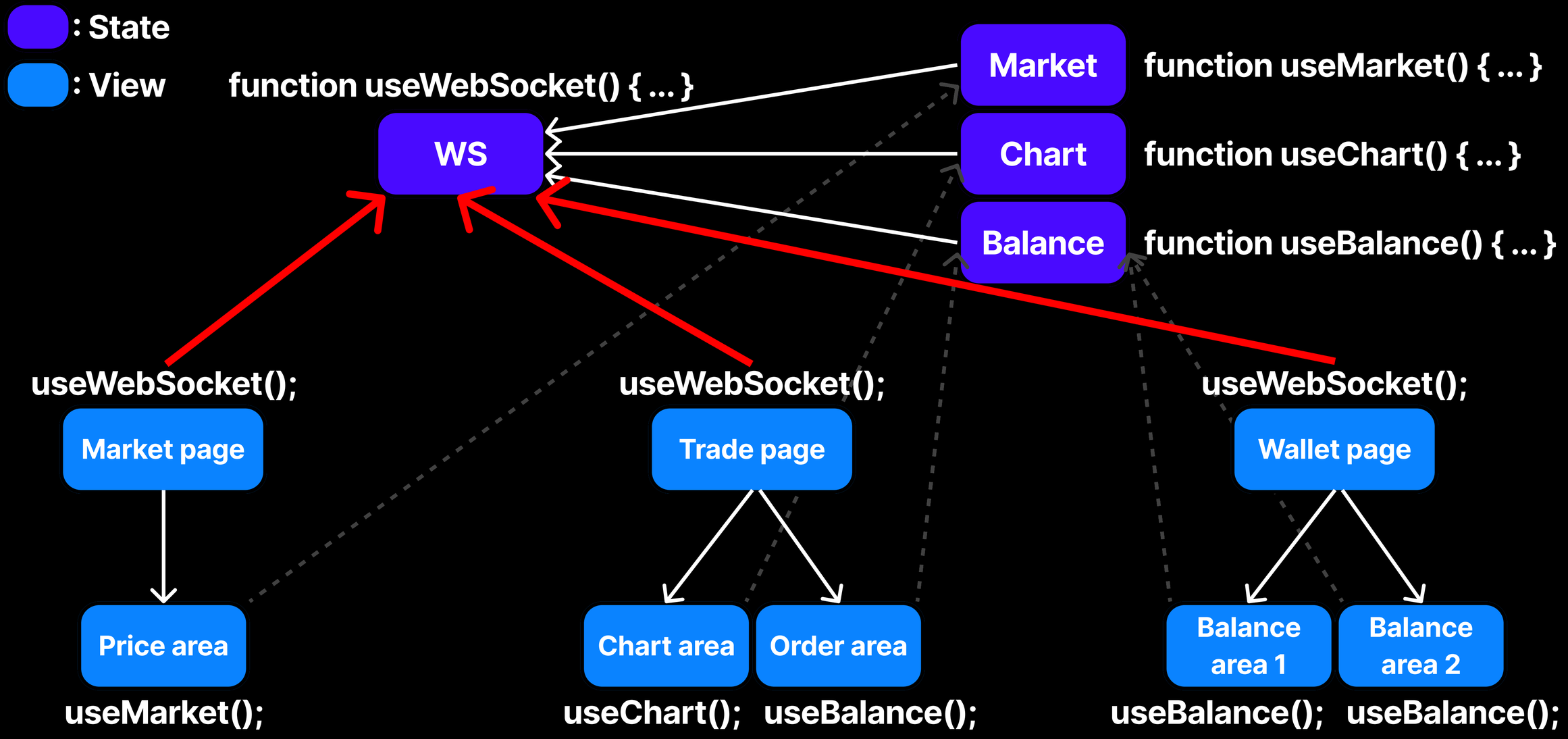
Problems with the original code

1. The place responsible for managing WebSocket connections and cleanup is separate from where the WebSocket state is used, leading to a lack of colocation among related concerns
2. Having multiple places referencing the same resource can lead to dangling subscription issues
3. Despite being an SPA, the WebSocket reconnects on every page navigation

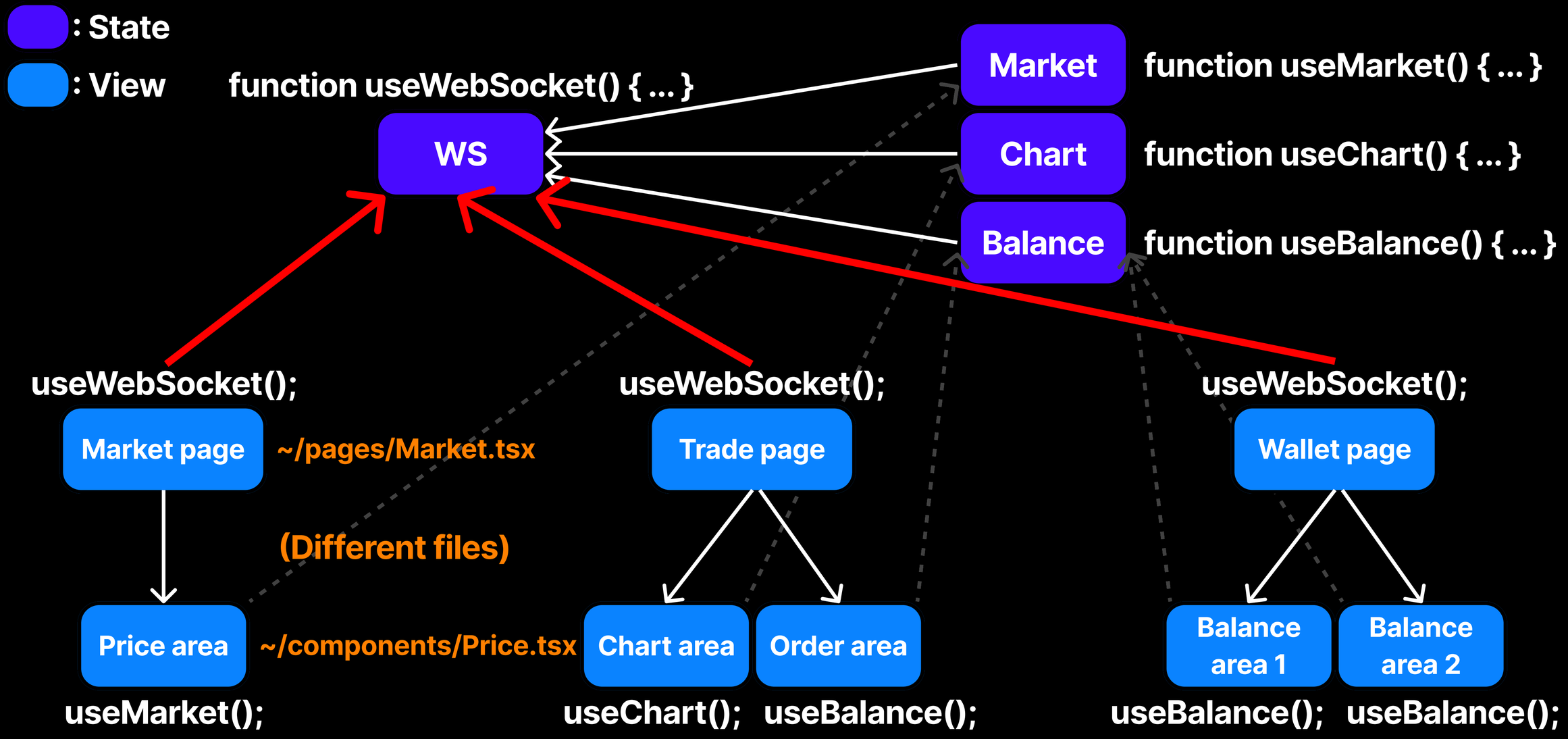
Problem 1 - Lack of colocation



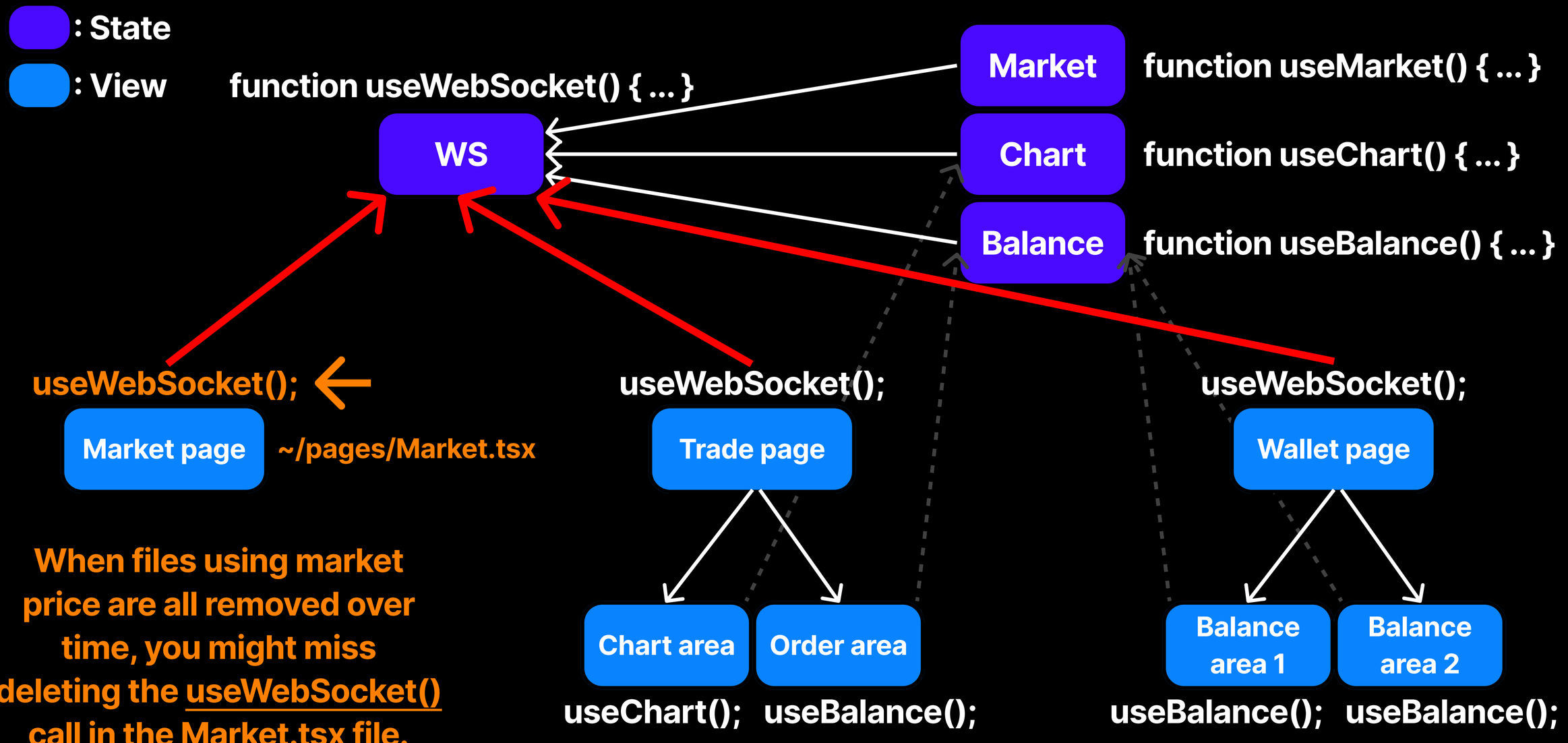
Problem 1 - Lack of colocation



Problem 1 - Lack of colocation



Problem 1 - Lack of colocation

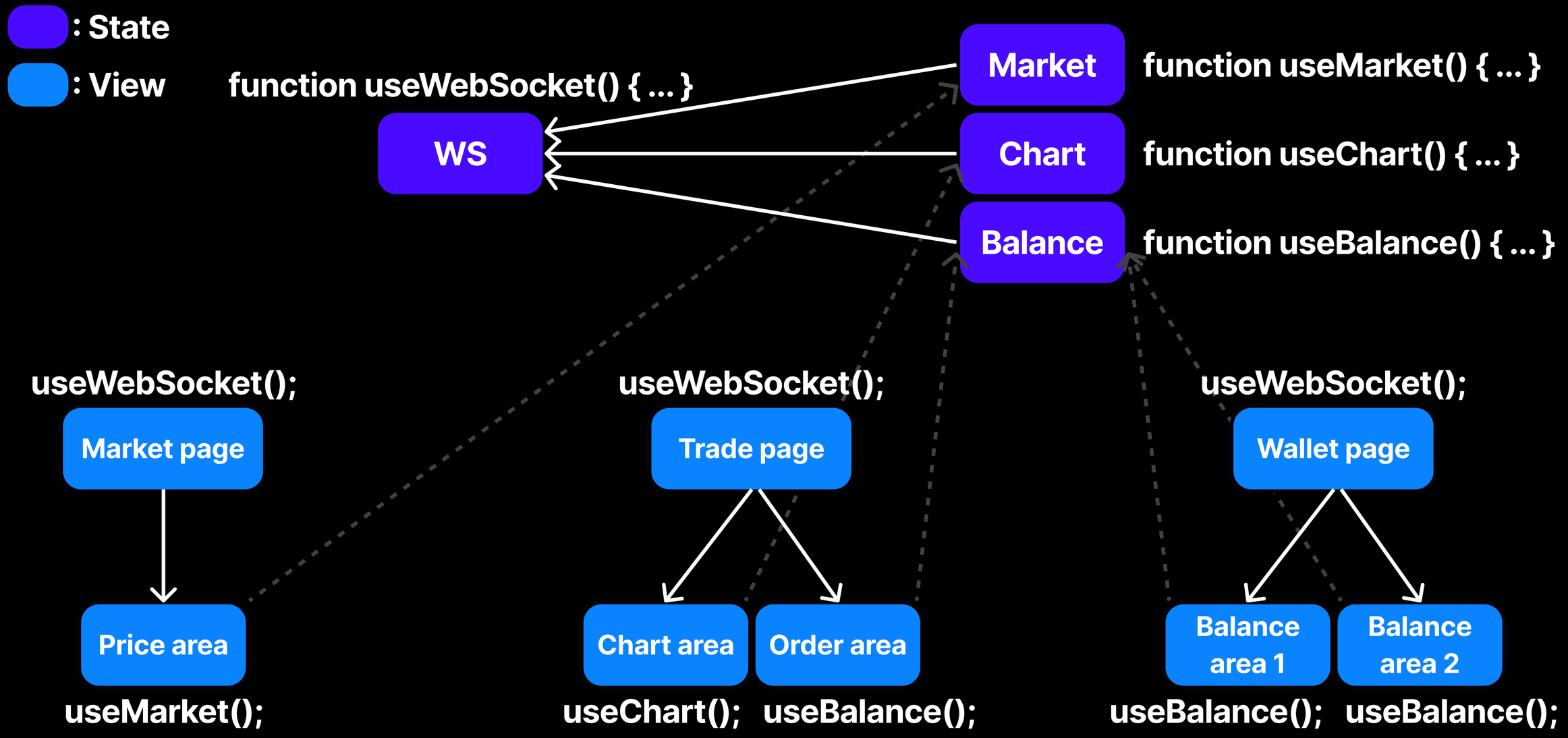


When files using market price are all removed over time, you might miss deleting the useWebSocket() call in the Market.tsx file.

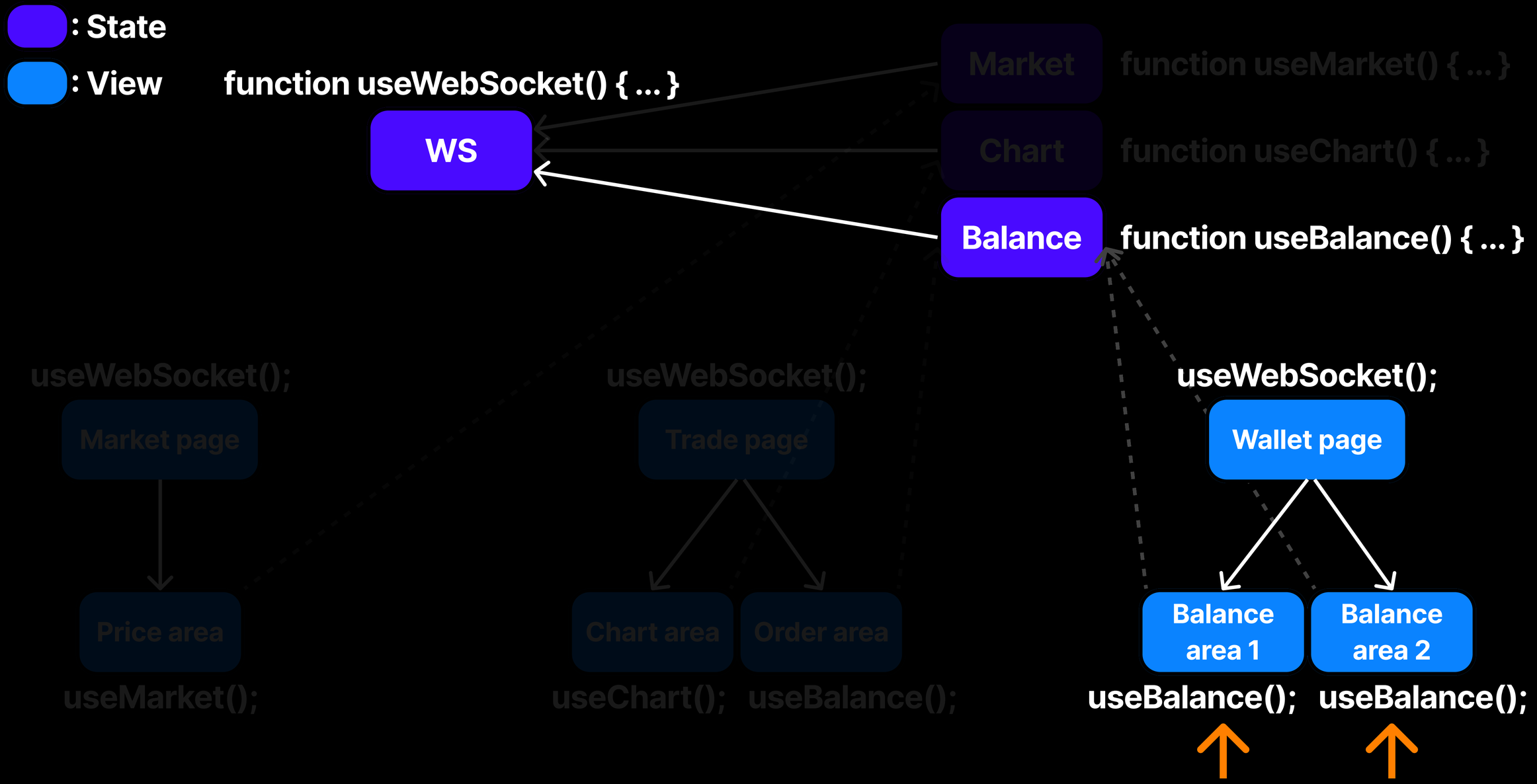
Problems with the original code

1. The place responsible for managing WebSocket connections and cleanup is separate from where the WebSocket state is used, leading to a lack of colocation among related concerns
2. Having multiple places referencing the same resource can lead to dangling subscription issues
3. Despite being an SPA, the WebSocket reconnects on every page navigation

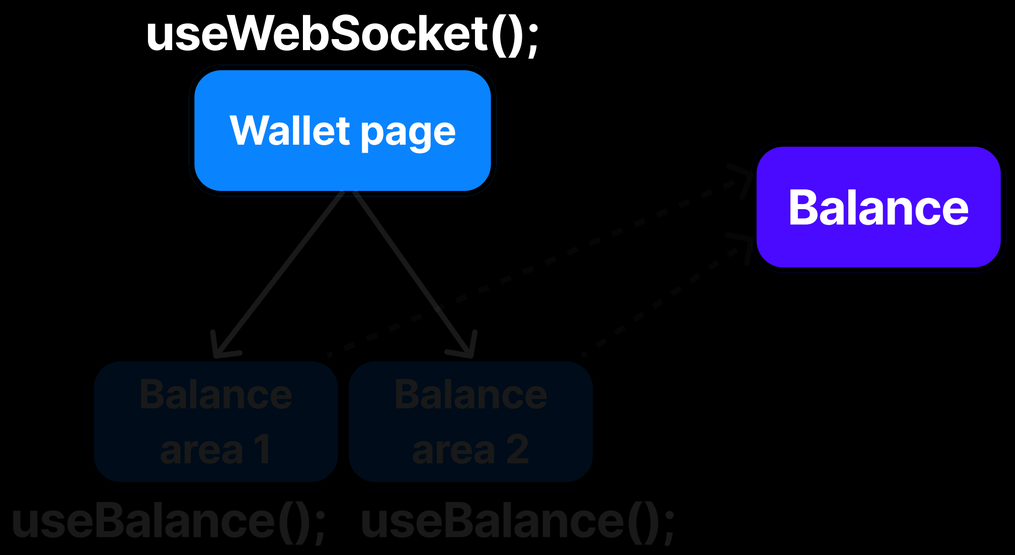
Problem 2 - Dangling subscriptions occur



Problem 2 - Dangling subscriptions occur

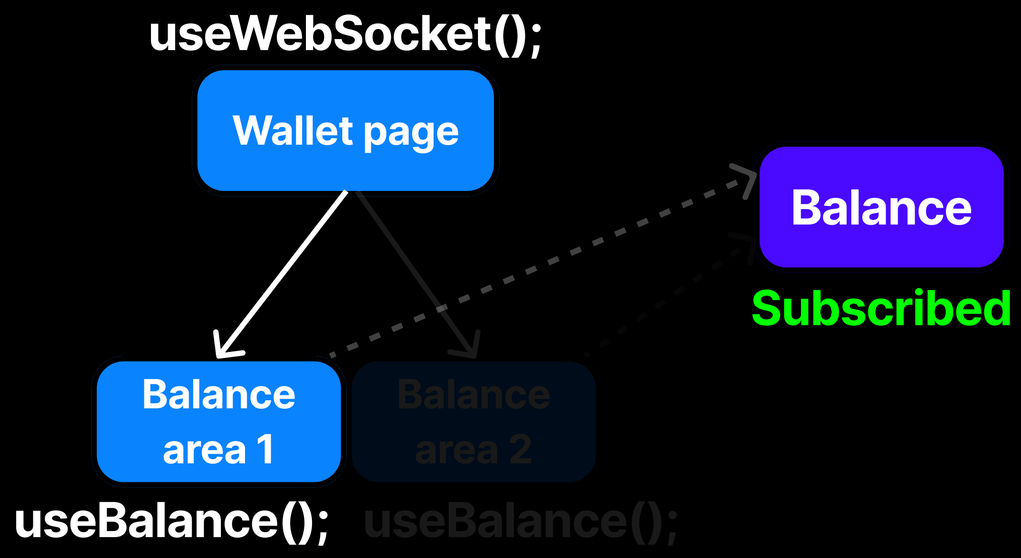


Problem 2 - Dangling subscriptions occur




```
function useBalance() {
  useEffect(() => {
    subBalance();
    return () => unsubBalance();
  });
}
```

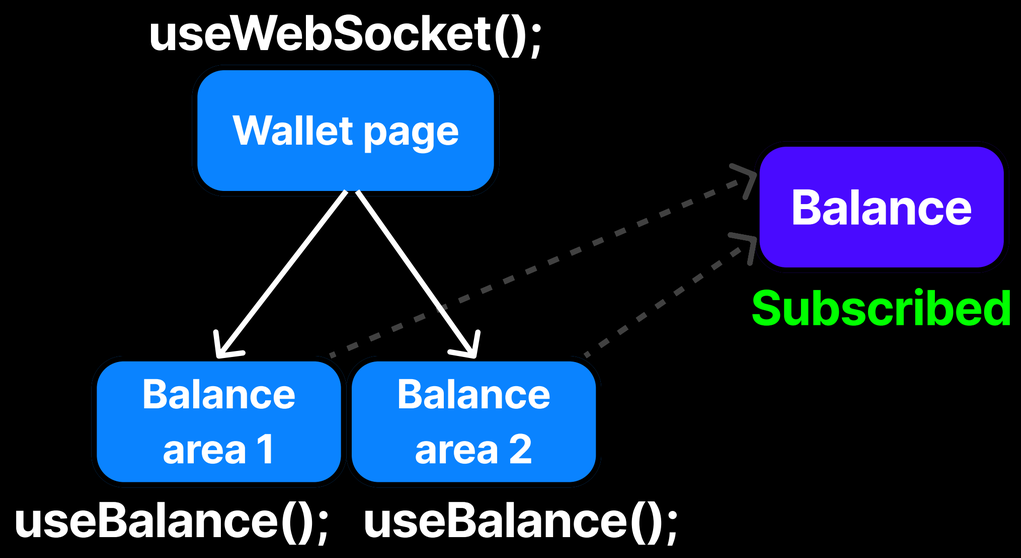
Problem 2 - Dangling subscriptions occur




```
function useBalance() {
  useEffect(() => {
    subBalance();
    return () => unsubBalance();
  });
}
```

(Balance area 1 mounted)

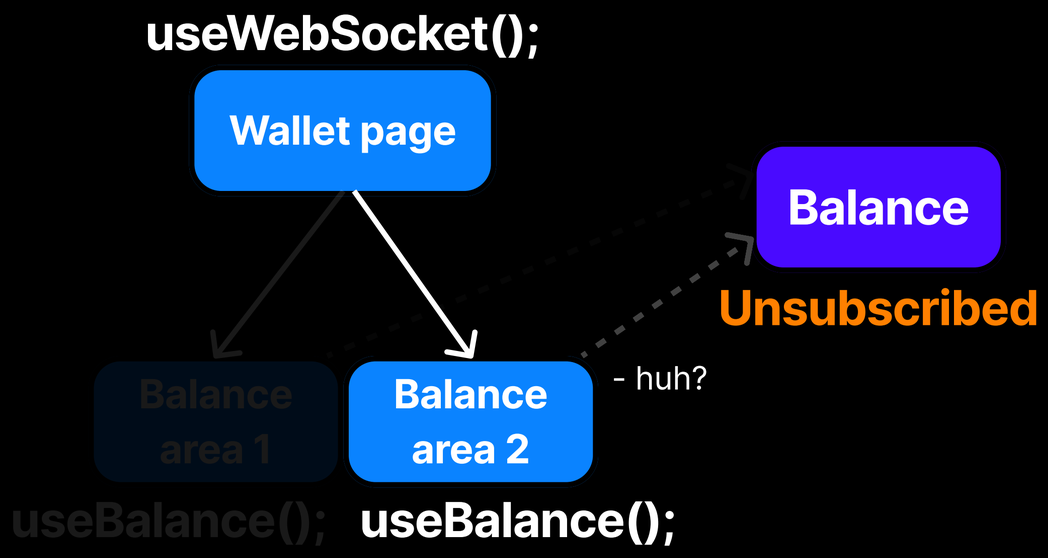
Problem 2 - Dangling subscriptions occur




```
function useBalance() {
  useEffect(() => {
    subBalance();
    return () => unsubBalance();
  });
}
```

(Balance area 2 mounted)

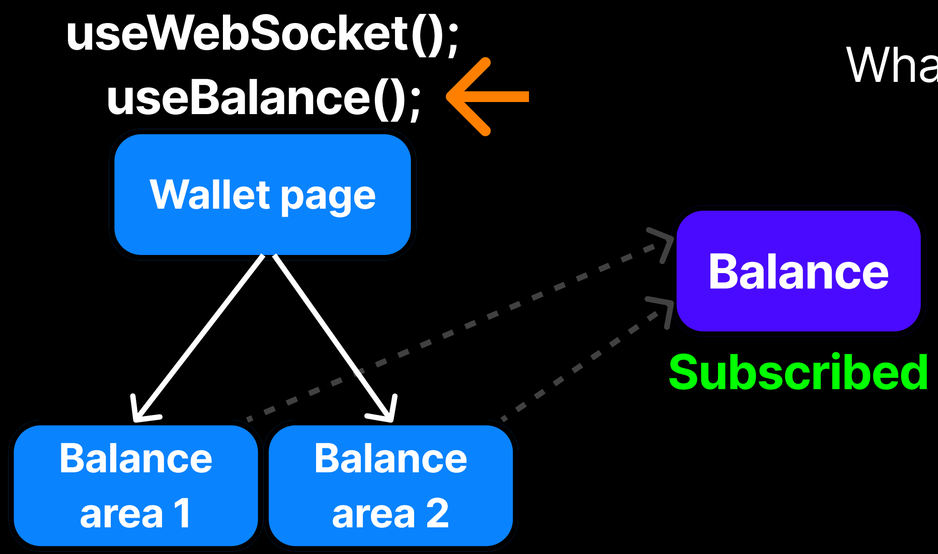
Problem 2 - Dangling subscriptions occur




```
function useBalance() {
  useEffect(() => {
    subBalance();
    return () => unsubBalance();
  });
}
```

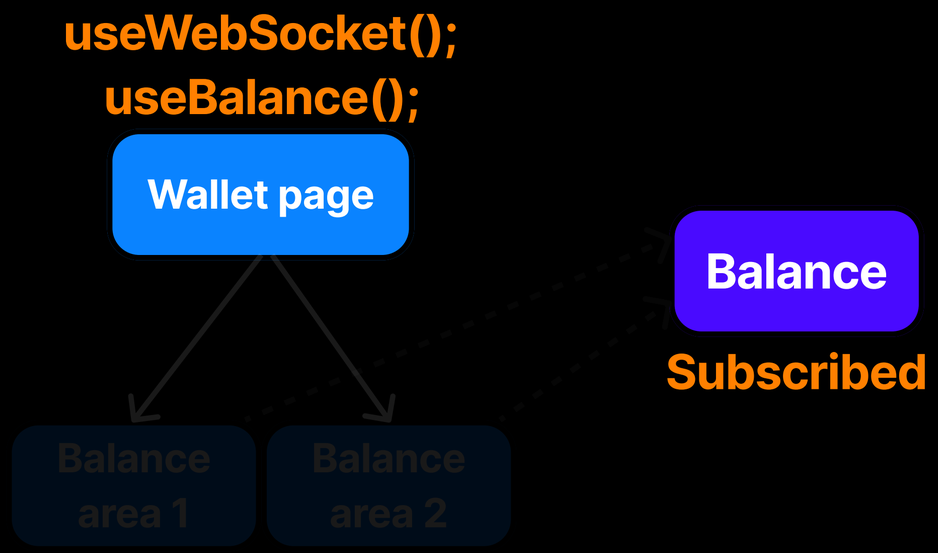
(Balance area 1 unmounted)

Problem 2 - Dangling subscriptions occur



What if the page handles all subscriptions?

Problem 2 - Dangling subscriptions occur

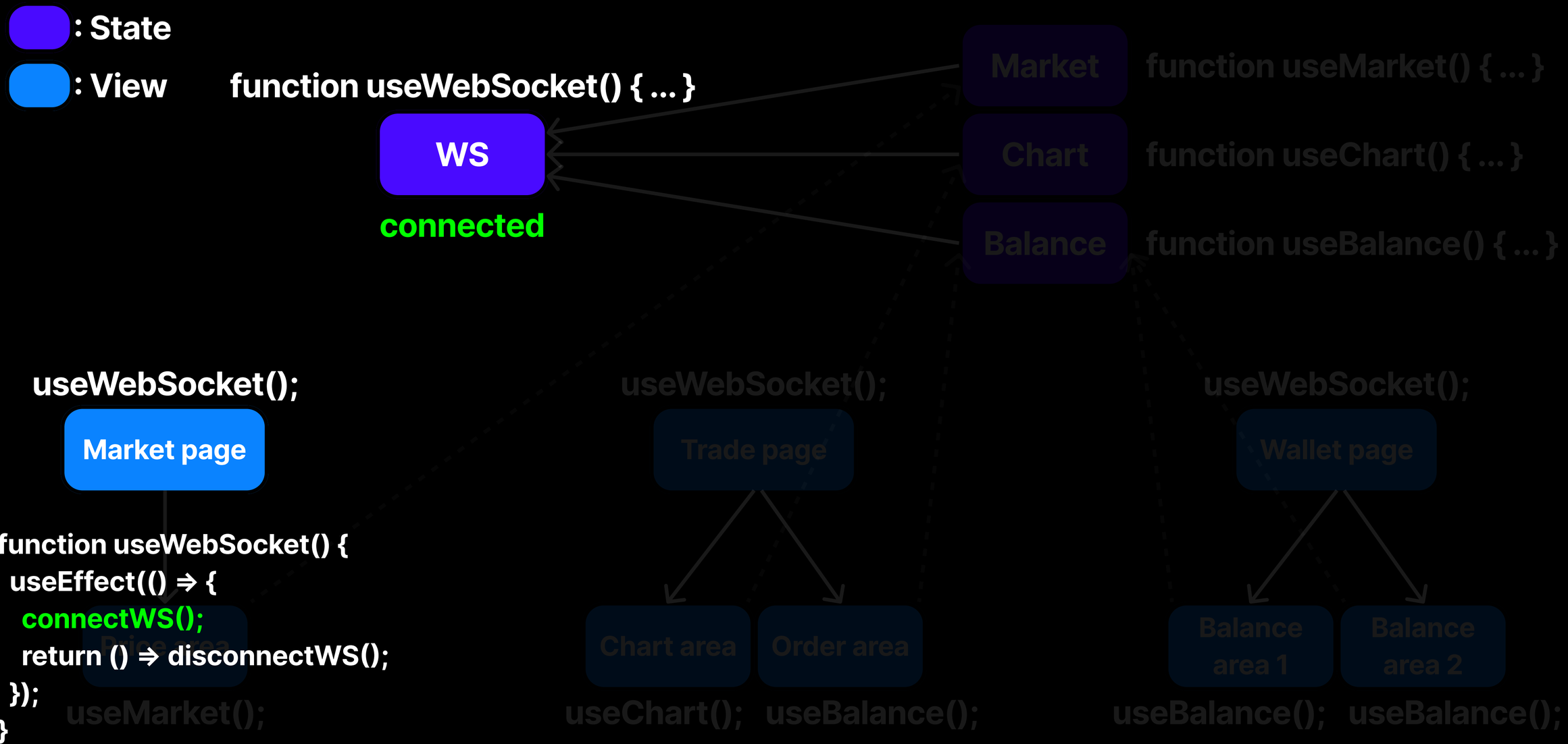


- Unnecessary subscription states may persist even when all areas viewing balance information are unmounted.
- It becomes more difficult to colocate related concerns.

Problems with the original code

1. The place responsible for managing WebSocket connections and cleanup is separate from where the WebSocket state is used, leading to a lack of colocation among related concerns
2. Having multiple places referencing the same resource can lead to dangling subscription issues
3. Despite being an SPA, the WebSocket reconnects on every page navigation

Problem 3 - WebSocket reconnects on page navigation



Problem 3 - WebSocket reconnects on page navigation

: State

: View

function useWebSocket() { ... }

WS

Disconnected & reconnected

Market

Chart

Balance

function useMarket() { ... }

function useChart() { ... }

function useBalance() { ... }

useWebSocket();

Market page

Price area

useMarket();

Market page unmounts, triggering useEffect cleanup

```
function useWebSocket() {
 useEffect(() => {
  connectWS();
  return () => disconnectWS();
 });
}
```

useWebSocket();

Trade page

Chart area

Order area

useChart(); useBalance();

```
function useWebSocket() {
 useEffect(() => {
  connectWS(); ← reconnect
  return () => disconnectWS();
 });
}
```

useWebSocket();

Wallet page

Balance area 1

Balance area 2

useBalance(); useBalance();

Problem 3 - WebSocket reconnects on page navigation

: State

: View

function useWebSocket() { ... }

WS

Disconnected &
reconnected again

Market

Chart

Balance

function useMarket() { ... }

function useChart() { ... }

function useBalance() { ... }

useWebSocket();

Market page

Price area

useMarket();

useWebSocket();

useChart();

useBalance();

Trade page unmounts,
triggering useEffect cleanup

```
function useWebSocket() {
 useEffect(() => {
  connectWS();
  return () => disconnectWS();
 });
}
```

useWebSocket();

Wallet page

useWebSocket();

useBalance();

useBalance();

```
function useWebSocket() {
 useEffect(() => {
  connectWS(); <- again
  return () => disconnectWS();
 });
}
```

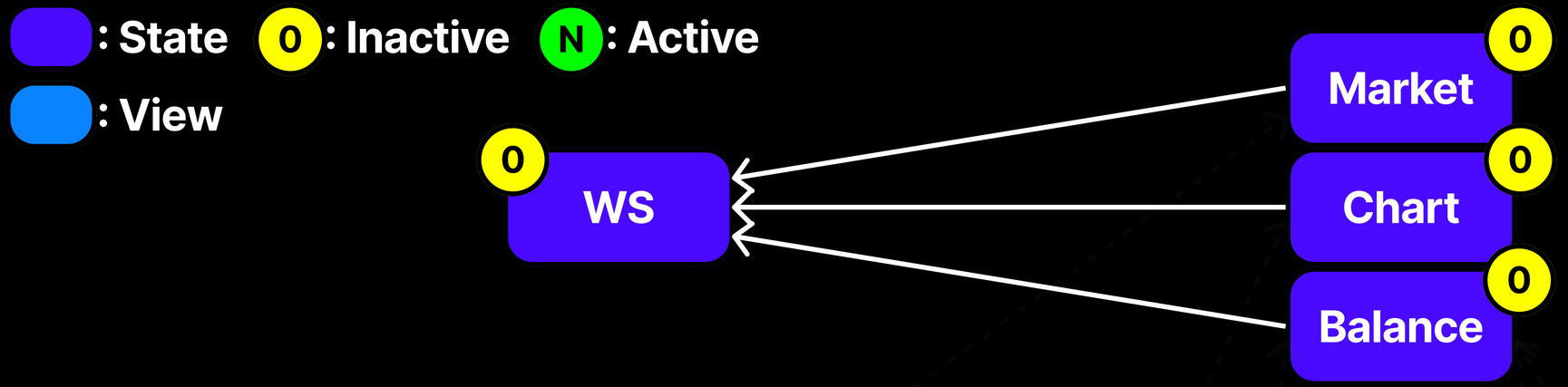
Solution

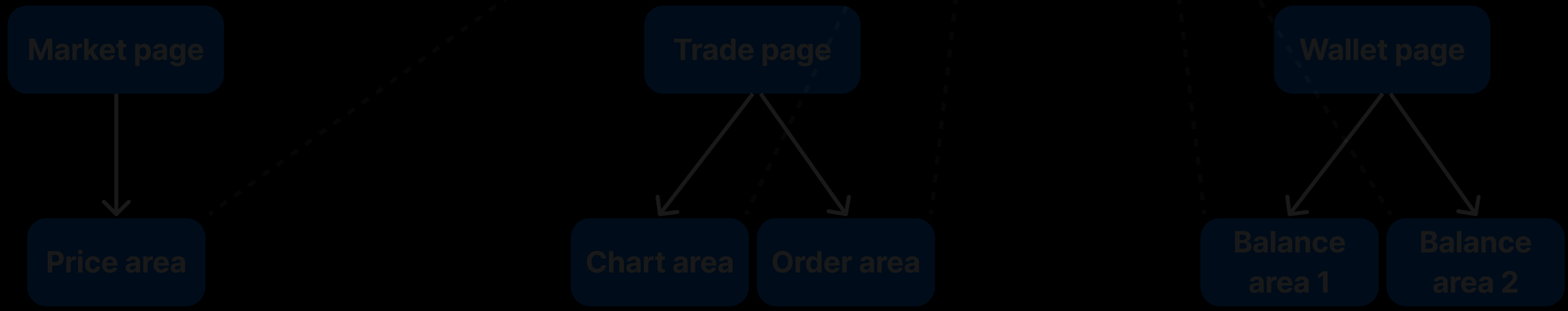
Reference Counting

- Apply reference counting to concepts with a start and end, such as WebSocket or market price information

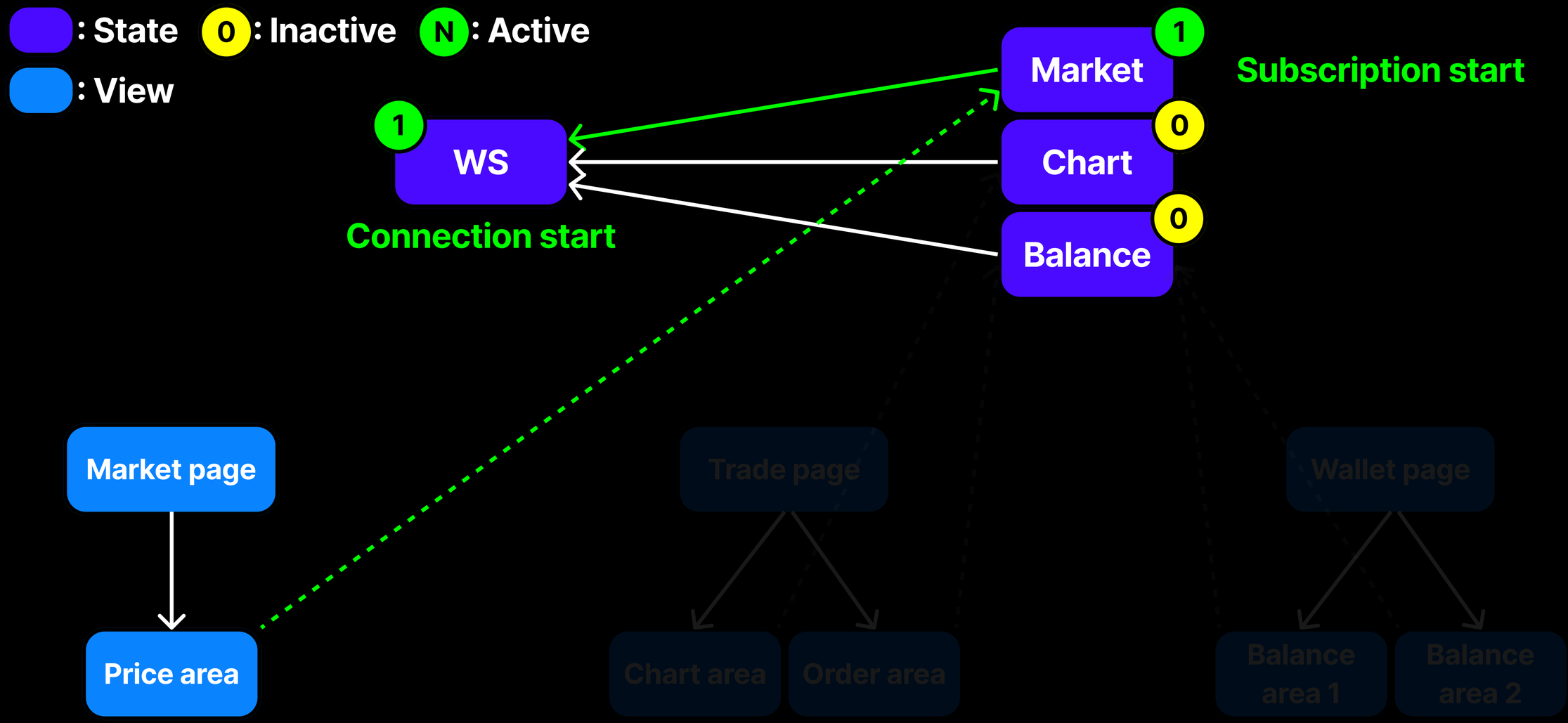
- +1 when usage begins somewhere, -1 when it ends
- If the count increases above 0, initialize the resource
- If the count returns to 0, clean up the resource

Solution - Reference Counting

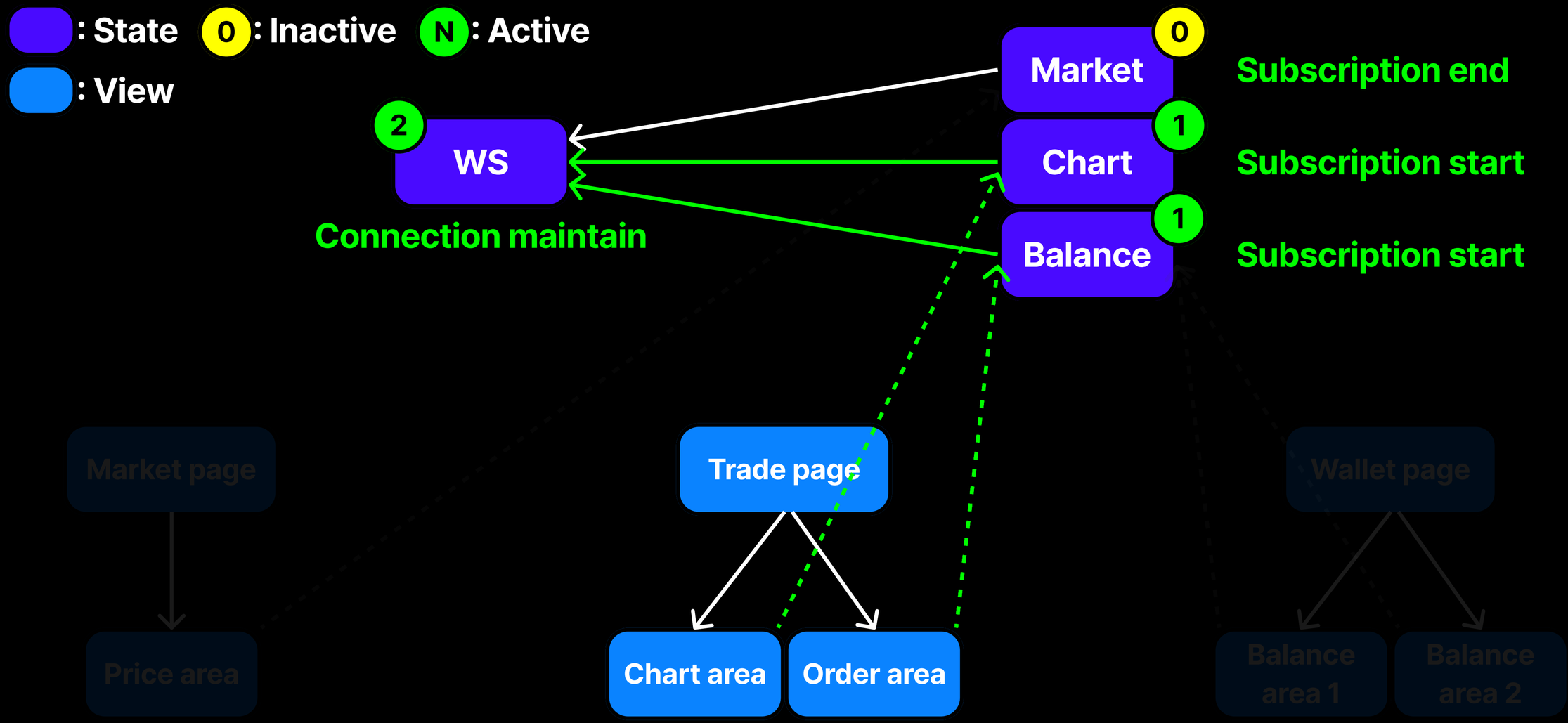




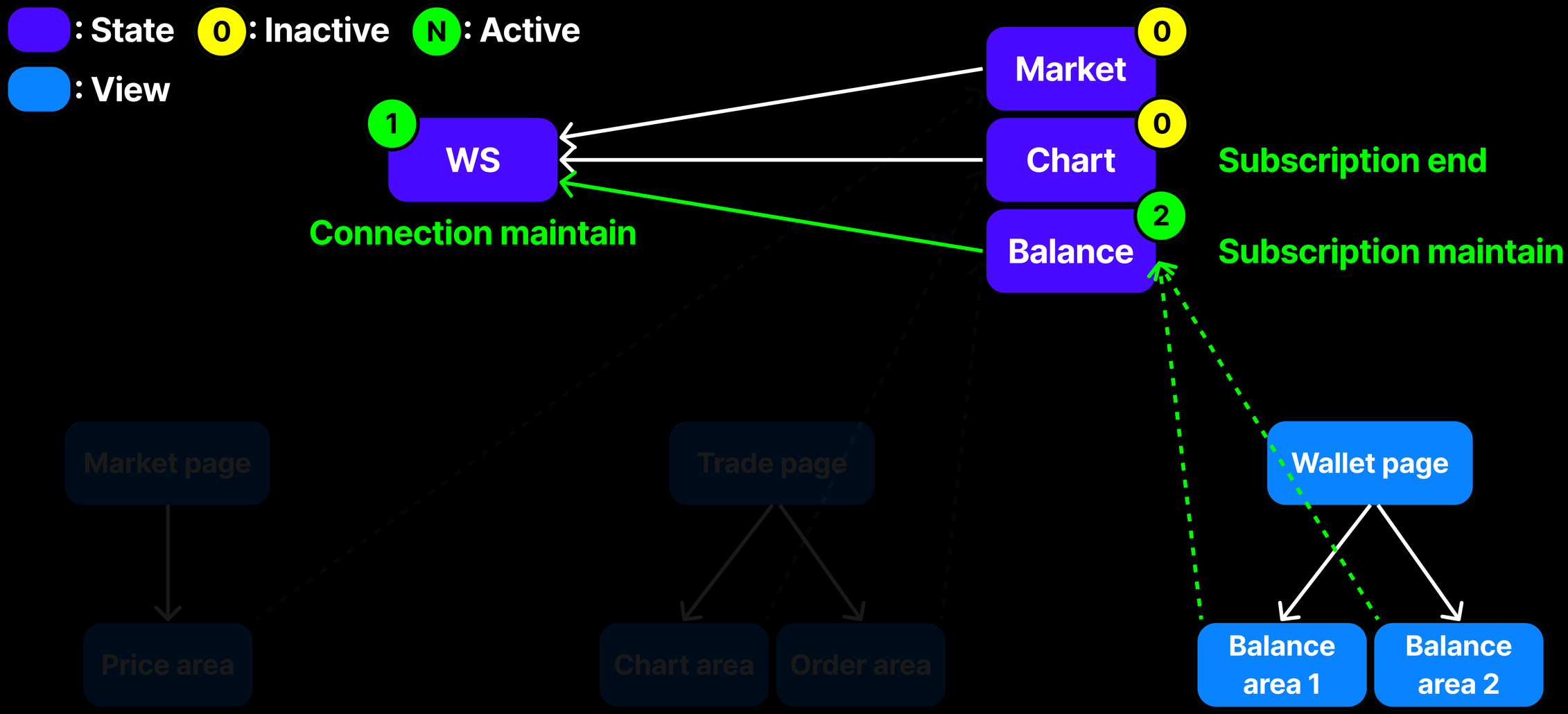
Solution - Reference Counting



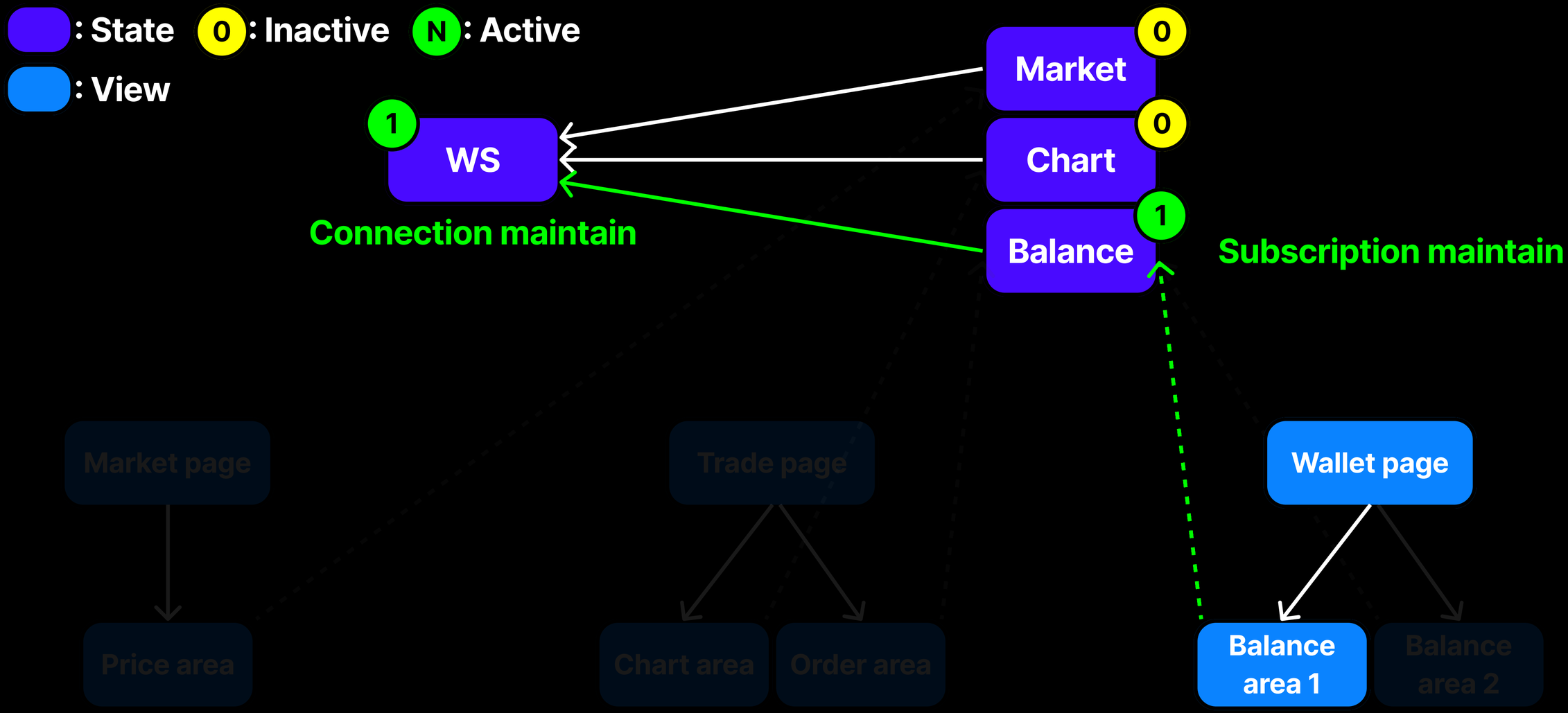
Solution - Reference Counting



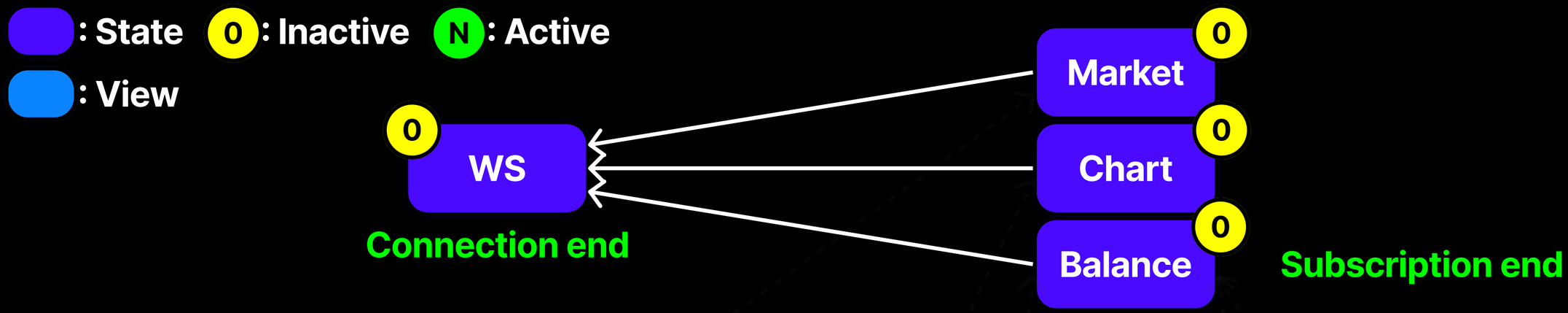
Solution - Reference Counting

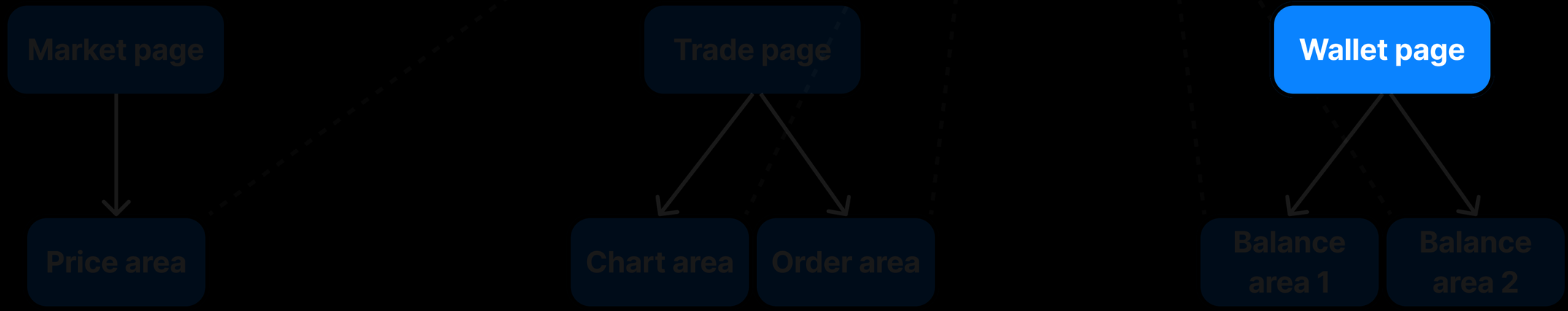


Solution - Reference Counting



Solution - Reference Counting





Library

Bunja

- Bunja (分子 / 분자) - Korean for molecule, member or element.

- Provides bunja and useBunja Functions
  - bunja – A state unit capable of describing dependency relationships
  - **useBunja – A hook where reference counting is automatically managed**

- Inspired by the Dependency Injection Library bunshi
- A metaphor where atoms combine to form molecules
  - Works well alongside libraries like Jotai

Code Using Bunja

WS

```
const WS = bunja([], () => ({
  [bunja.effect]() {
    connectWS();
    return () => disconnectWS();
  }
}));
```

Market

```
const Market = bunja([WS], () => ({
  [bunja.effect]() {
    subMarket();
    return () => unsubMarket();
  }
}));
```

Chart

```
const Chart = bunja([WS], () => ({
  [bunja.effect]() {
    subChart();
    return () => unsubChart();
  }
}));
```

Balance

```
const Balance = bunja([WS], () => ({
  [bunja.effect]() {
    subBalance();
    return () => unsubBalance();
  }
}));
```

Code Using Bunja

Market page

```
function MarketPage() {
 return <PriceArea />;
}

function PriceArea() {
 useBunja(Market);
 return ...;
}
```

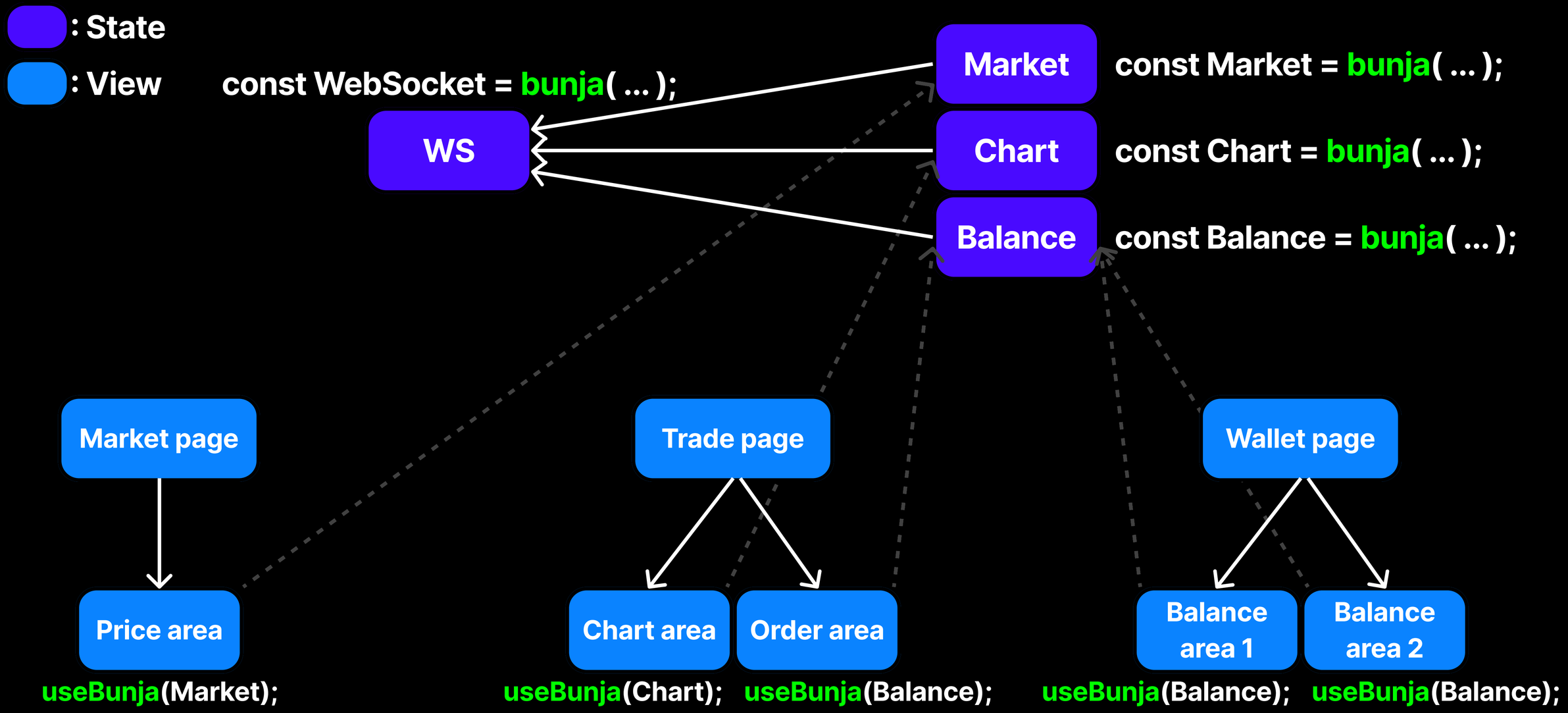
Trade page

```
function TradePage() {
 return <>
  <ChartArea />
  <OrderArea />
 </>;
}

function ChartArea() {
 useBunja(Chart);
 return ...;
}
function OrderArea() {
 useBunja(Balance);
 return ...;
}
```

Wallet page

```
function WalletPage() {
 return <>
  <BalanceArea1 />
  <BalanceArea2 />
 </>;
}

function BalanceArea1() {
 useBunja(Balance);
 return ...;
}
function BalanceArea2() {
 useBunja(Balance);
 return ...;
}
```

Code Using Bunja



Problems Solved by Bunja

1. The place responsible for managing WebSocket connections and cleanup is separate from where the WebSocket state is used, leading to a lack of colocation among related concerns
2. Having multiple places referencing the same resource can lead to dangling subscription issues
3. Despite being an SPA, the WebSocket reconnects on every page navigation

Problems Solved by Bunja

1. The place responsible for managing WebSocket connections and cleanup is separate from where the WebSocket state is used, leading to a lack of colocation among related concerns

    - By defining dependency relationships between resources, React components can focus solely on consuming the state without worrying about its lifetime management

    - In useBunja(Bunja) code, it’s easy to navigate to dependent bunjas using "go to definition"
      In code utilizing bunja, initialization and cleanup logic can be easily traced from anywhere

Problems Solved by Bunja

1. The place responsible for managing WebSocket connections and cleanup is separate from where the WebSocket state is used, leading to a lack of colocation among related concerns
2. Having multiple places referencing the same resource can lead to dangling subscription issues
3. Despite being an SPA, the WebSocket reconnects on every page navigation

Problems Solved by Bunja

2. Having multiple places referencing the same resource can lead to dangling subscription issues

- As long as useBunja(Bunja) call exists somewhere in the render tree, the resource remains active
- When all components calling useBunja(Bunja) disappear from the render tree, the resource is automatically disposed

Problems Solved by Bunja

1. The place responsible for managing WebSocket connections and cleanup is separate from where the WebSocket state is used, leading to a lack of colocation among related concerns
2. Having multiple places referencing the same resource can lead to dangling subscription issues
3. Despite being an SPA, the WebSocket reconnects on every page navigation

Problems Solved by Bunja

3. Despite being an SPA, the WebSocket reconnects on every page navigation

- Just like how it resolves the dangling subscription issue, as long as useBunja(Bunja) is used somewhere in the render tree, the WebSocket connection remains active.

- In fact, if only the reference counting concept exists, resources are immediately cleaned up when the count reaches zero, which can still lead to reconnection issues during page navigation.

    - In bunja, when the reference count decreases, it waits for one tick using setTimeout, and if the count is still zero at that point, the resource is cleaned up.

End

Questions?

https://github.com/disjukr/bunja

https://www.npmjs.com/package/bunja

https://jsr.io/@disjukr/bunja