Problem 3: Messy React — Analysis & Refactor

Issues (Anti-Patterns & Inefficiencies)

1. **Broken/unsafe types**
   - `WalletBalance` lacks `blockchain`, yet it is used.
   - `getPriority(blockchain: any)` uses `any`.
   - Items treated as `FormattedWalletBalance` without `formatted`.
2. **Incorrect logic/bugs**
   - Filter references undefined `lhsPriority`; predicate keeps `amount <= 0`; otherwise returns `false`.
   - Sort callback omits returning `0` for equality.
3. **Misused memoization**
   - `useMemo` depends on `prices` (unused) and omits `getPriority`, which is recreated each render.
4. **Redundant work**
   - `formattedBalances` computed but never used; list is mapped twice.
5. **Rendering anti-patterns**
   - Unstable `key={index}`. Use a stable identifier.
6. **Formatting/robustness**
   - `toFixed()` defaults to 0 decimals.
   - `prices[currency]` may be `undefined` ⇒ `NaN`.
7. **Minor**
   - Destructures `children` but doesn't use it.
   - `getPriority` defined inside component each render.

Refactored Version

Listing 1: Typed, correct, and efficient version

```
import React, { useMemo } from 'react';
import type { BoxProps } from '@mui/system';

interface WalletBalance {
  currency: string;
```

```
  amount: number;
  blockchain: 'Osmosis' | 'Ethereum' | 'Arbitrum' | 'Zilliqa' | 'Neo' |
      string;
}

interface FormattedWalletBalance extends WalletBalance {
  formatted: string;
}

interface PricesMap {
  [currency: string]: number | undefined;
}

interface Props extends BoxProps {}

const CHAIN_PRIORITY: Record<string, number> = {
  Osmosis: 100,
  Ethereum: 50,
  Arbitrum: 30,
  Zilliqa: 20,
  Neo: 20,
};

const getPriority = (blockchain: string): number =>
  CHAIN_PRIORITY[blockchain] ?? -99;

const formatAmount = (amount: number) =>
  new Intl.NumberFormat(undefined, { maximumFractionDigits: 6 }).format
     (amount);

export const WalletPage: React.FC<Props> = (props) => {
  const { /* children, */ ...rest } = props;
  const balances = useWalletBalances() as WalletBalance[];
  const prices = usePrices() as PricesMap;

  const formattedBalances: FormattedWalletBalance[] = useMemo(() => {
    const filtered = balances.filter(
      (b) => getPriority(b.blockchain) > -99 && b.amount > 0
    );

    filtered.sort((a, b) => {
      const pa = getPriority(a.blockchain);
      const pb = getPriority(b.blockchain);
      if (pa > pb) return -1;
      if (pa < pb) return 1;
      return 0;
    });

    return filtered.map((b) => ({
      ...b,
      formatted: formatAmount(b.amount),
    }));
  }, [balances]);

  return (
    <div {...rest}>
      {formattedBalances.map((b) => {
        const price = prices?.[b.currency] ?? 0;
```

```
62          const usdValue = price * b.amount;
63          return (
64            <WalletRow
65              className={classes.row}
66              key={`${b.blockchain}:${b.currency}`}
67              amount={b.amount}
68              usdValue={usdValue}
69              formattedAmount={b.formatted}
70            />
71          );
72        })}
73      </div>
74    );
75  };
```

Why This is Better

- Strong typing; no `any`; all used fields exist.
- Correct filter and total ordering; guards against missing prices.
- Effective memoization (stable `getPriority`, minimal deps).
- Single pass for formatting; no redundant mapping.
- Stable `key` and sensible number formatting.