Revisiting x State x Management

Mohamed EL AYADI

@incepterr

React

A JavaScript library

React

A JavaScript library for building user interfaces

React

A JavaScript library

A UI runtime that run components

React as a UI Runtime

February 2, 2019 · 37 min read

An in-depth description of the React programming model.

React

A JavaScript library

A UI runtime that run components

A set of packages and renderers of different hosts environments

whoami

Mohamed EL AYADI

Senior software engineer with a decade of experience

Java & JavaScript ecosystems

@incepter

@incepterr

Plan

- What x Is x State ?
- State crimes x And x State police
- State x Management x Aspects
- Conclusion x And x Takeaways

What x Is x State ?

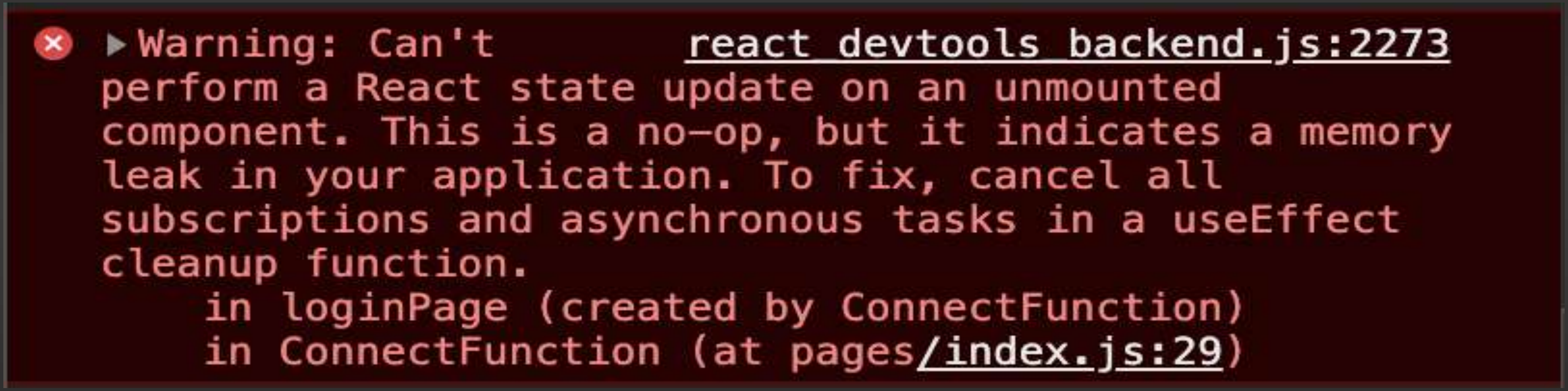
- It is the internal memory of a react component
- State setter is the only trigger of the update phase in react
    - OK, may be useSyncExternalStore also, but not trivial
- Some states need to be shared in a tree
- Shared by prop drilling or wired via context Api.
- Asynchronous states update state later.

State crimes x And x State police

- Cannot update state of unmounted component
- Showing results of earlier request
- No cancellations
- No pending states
- Poor error handling
- Useless useEffects
- Flashing old state
- No sharing
- The list doesn’t end here…

State crimes x And x State police

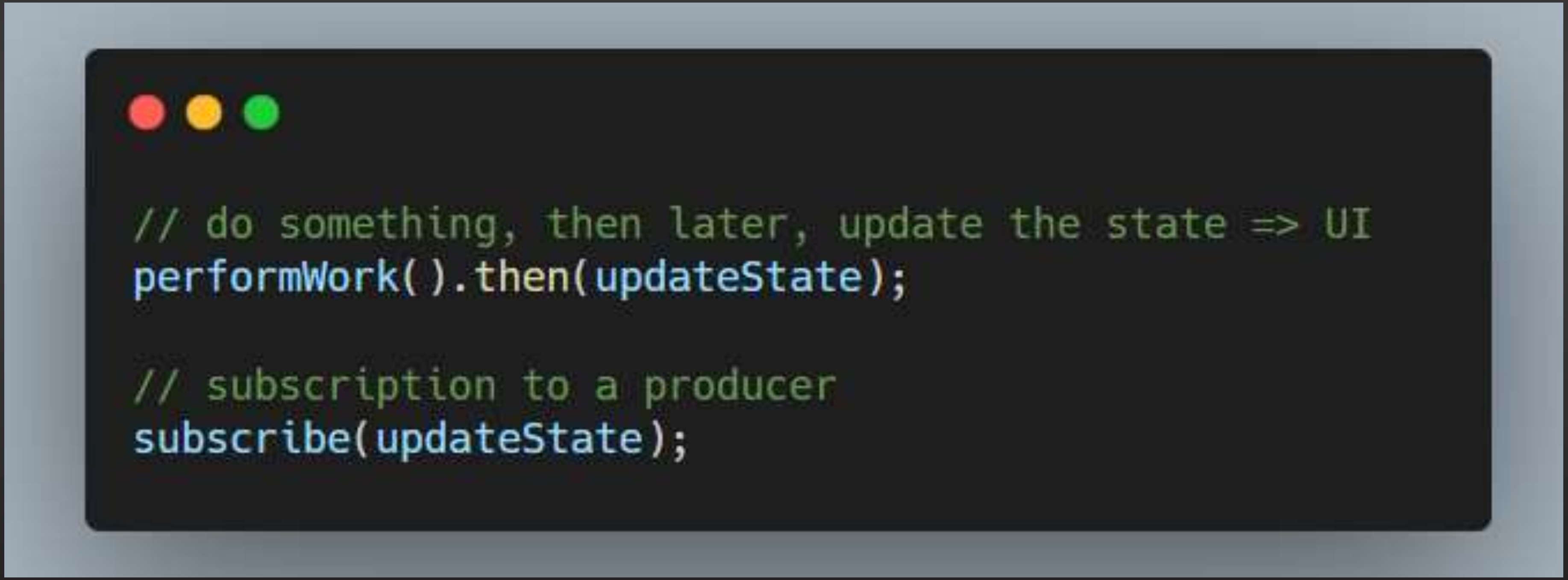
- Cannot update state of unmounted component



```
▶Warning: Can't          react_devtools_backend.js:2273
perform a React state update on an unmounted
component. This is a no-op, but it indicates a memory
leak in your application. To fix, cancel all
subscriptions and asynchronous tasks in a useEffect
cleanup function.
    in loginPage (created by ConnectFunction)
    in ConnectFunction (at pages/index.js:29)
```

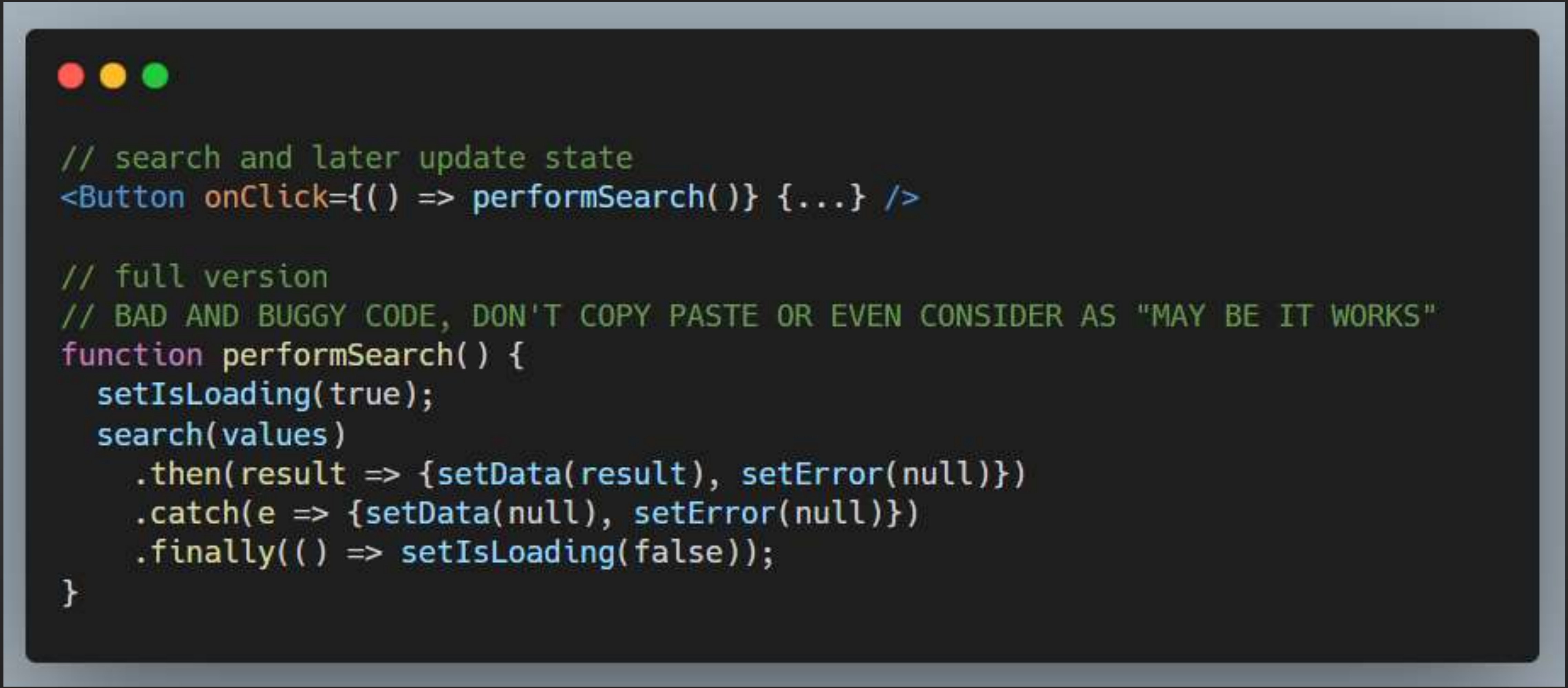
State crimes x And x State police

- Cannot update state of unmounted component



```
// do something, then later, update the state => UI
performWork().then(updateState);

// subscription to a producer
subscribe(updateState);
```

State crimes x And x State police



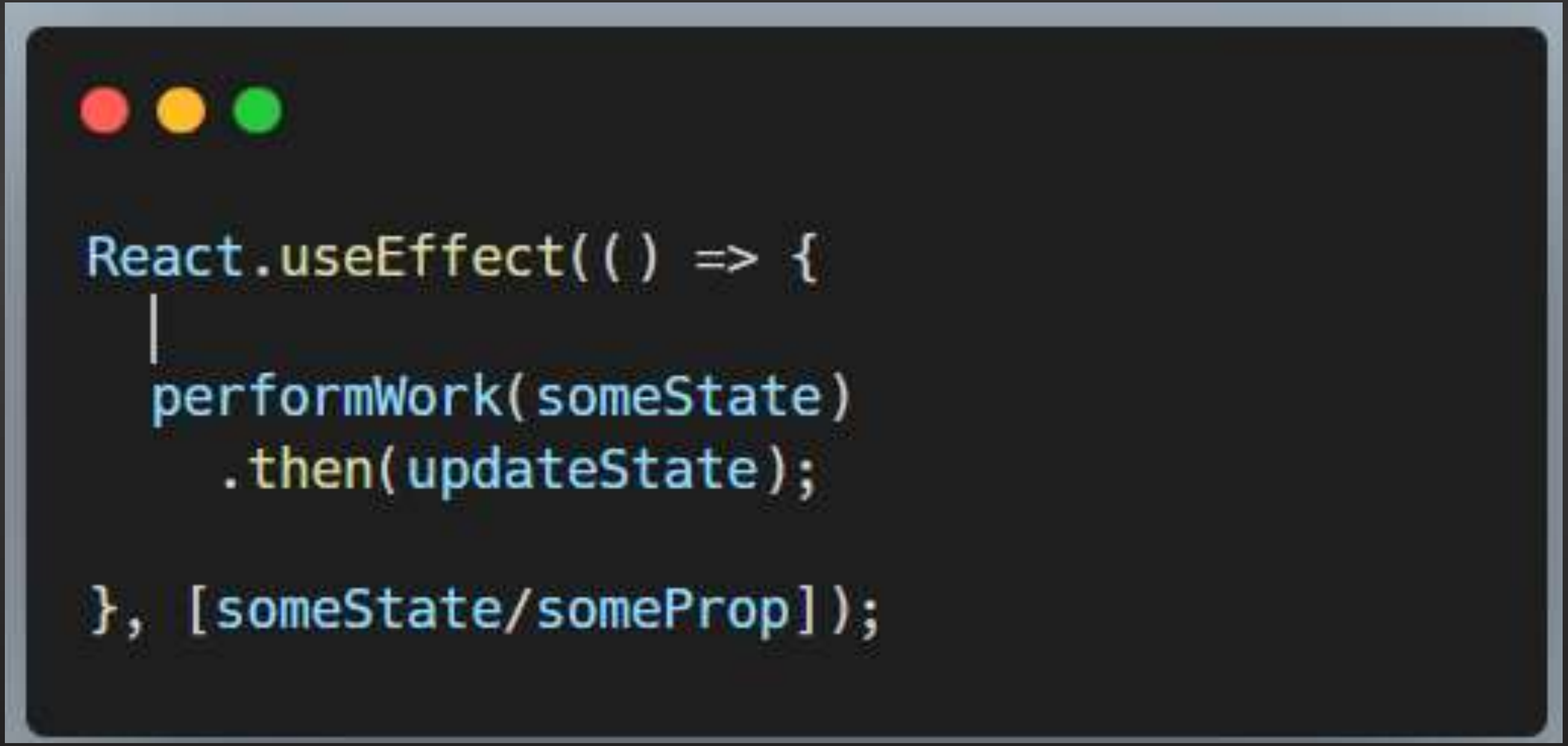
```
// search and later update state
<Button onClick={() => performSearch()} {...} />

// full version
// BAD AND BUGGY CODE, DON'T COPY PASTE OR EVEN CONSIDER AS "MAY BE IT WORKS"
function performSearch() {
  setIsLoading(true);
  search(values)
    .then(result => {setData(result), setError(null)})
    .catch(e => {setData(null), setError(null)})
    .finally(() => setIsLoading(false));
}
```

State crimes x And x State police

- Problems with this code
  - The loading state isn’t a Boolean
  - Three pieces of states working and updated together
  - Boolean semaphore lock rather than Counting lock!
  - Doesn’t abort the previous call

State crimes x And x State police

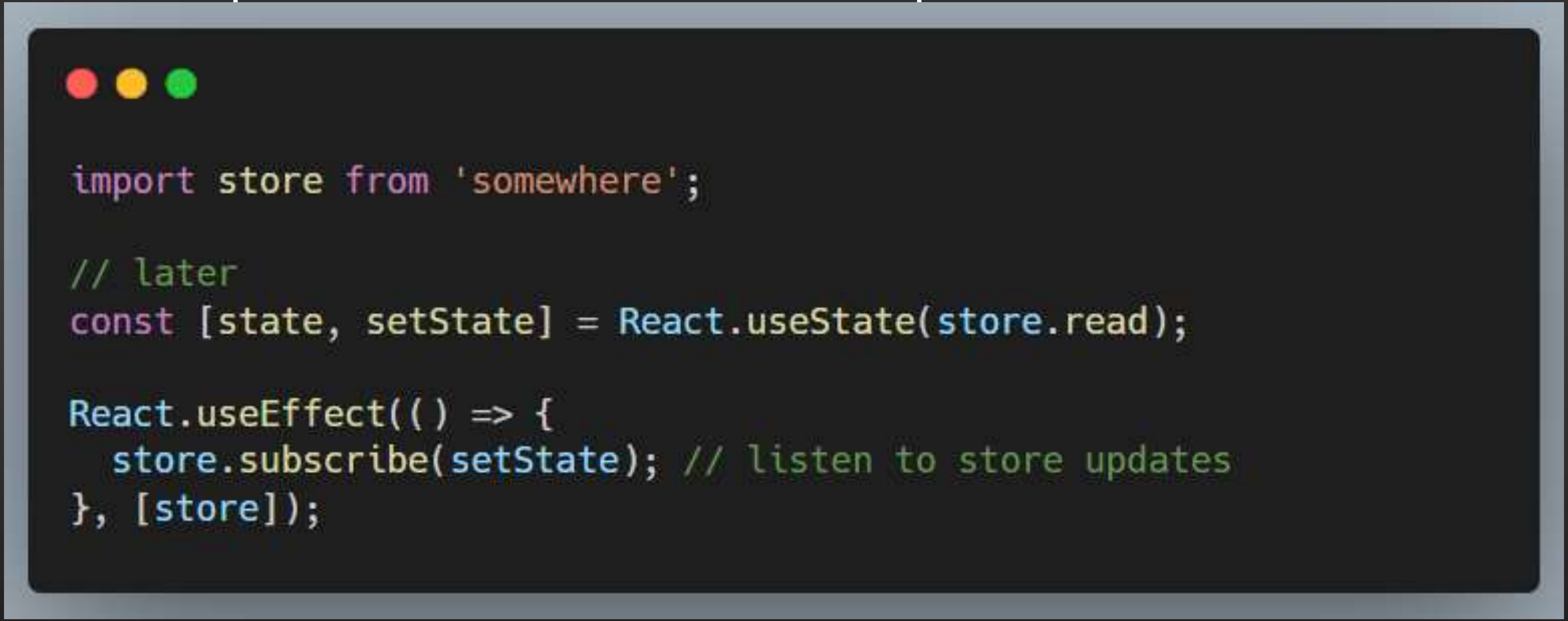
- Cannot update state of unmounted component



```
React.useEffect(() => {

  performWork(someState)
    .then(updateState);

}, [someState/someProp]);
```

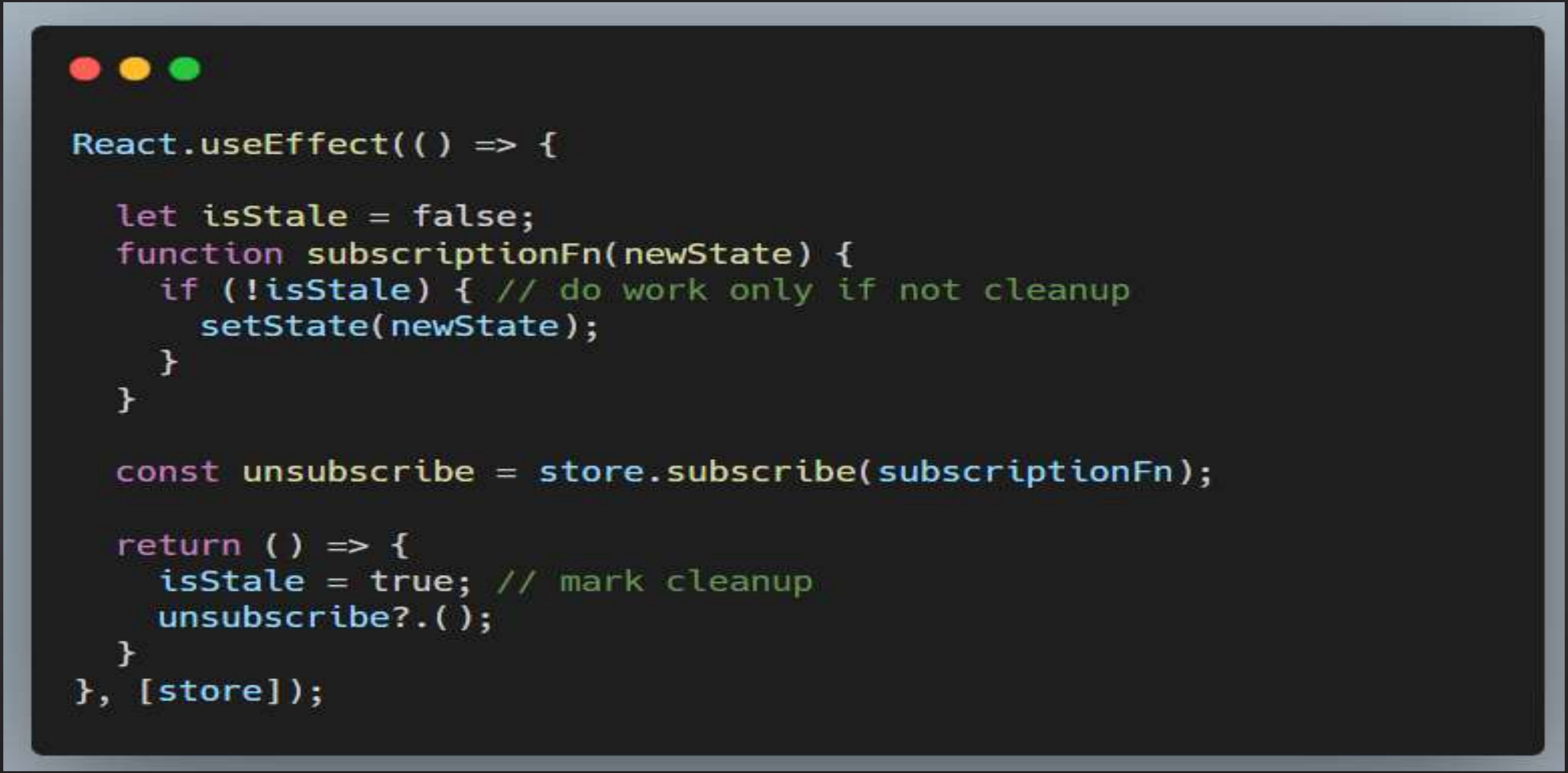
State crimes x And x State police

- Cannot update state of unmounted component



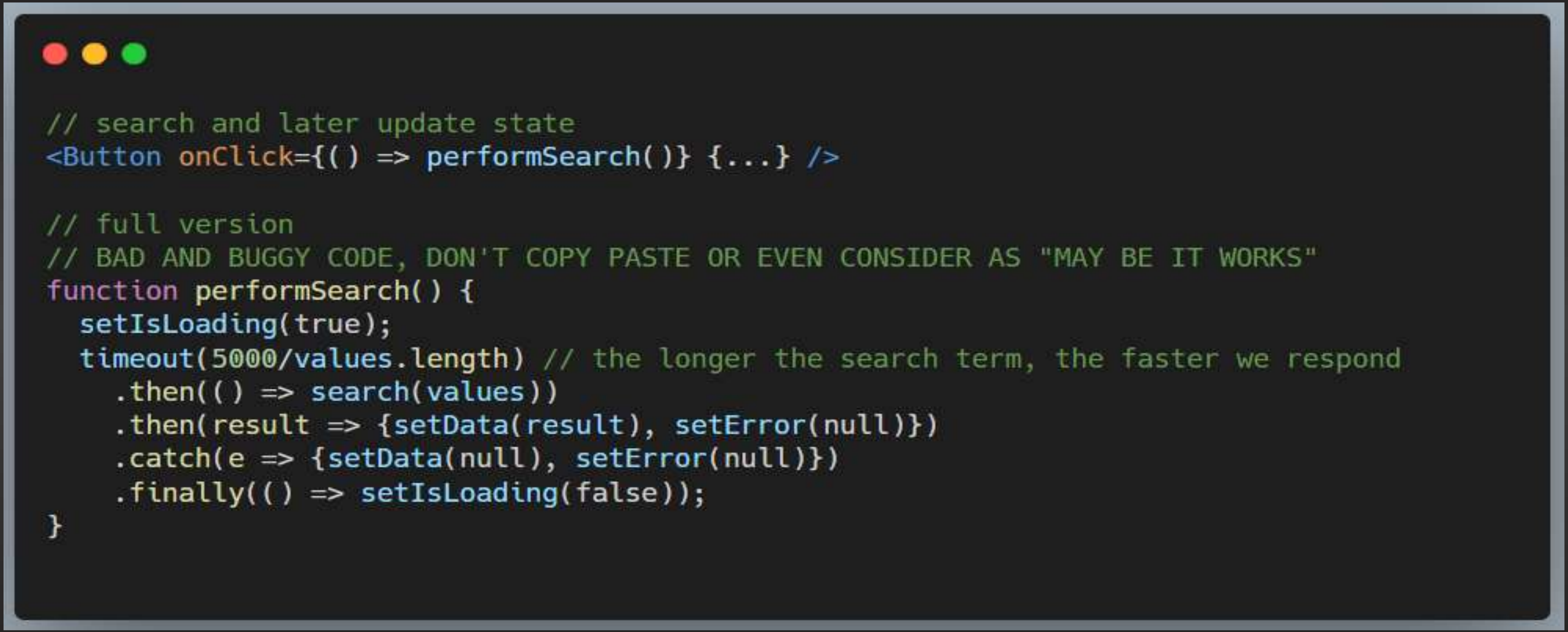
```
import store from 'somewhere';

// later
const [state, setState] = React.useState(store.read);

React.useEffect(() => {
  store.subscribe(setState); // listen to store updates
}, [store]);
```

State crimes x And x State police



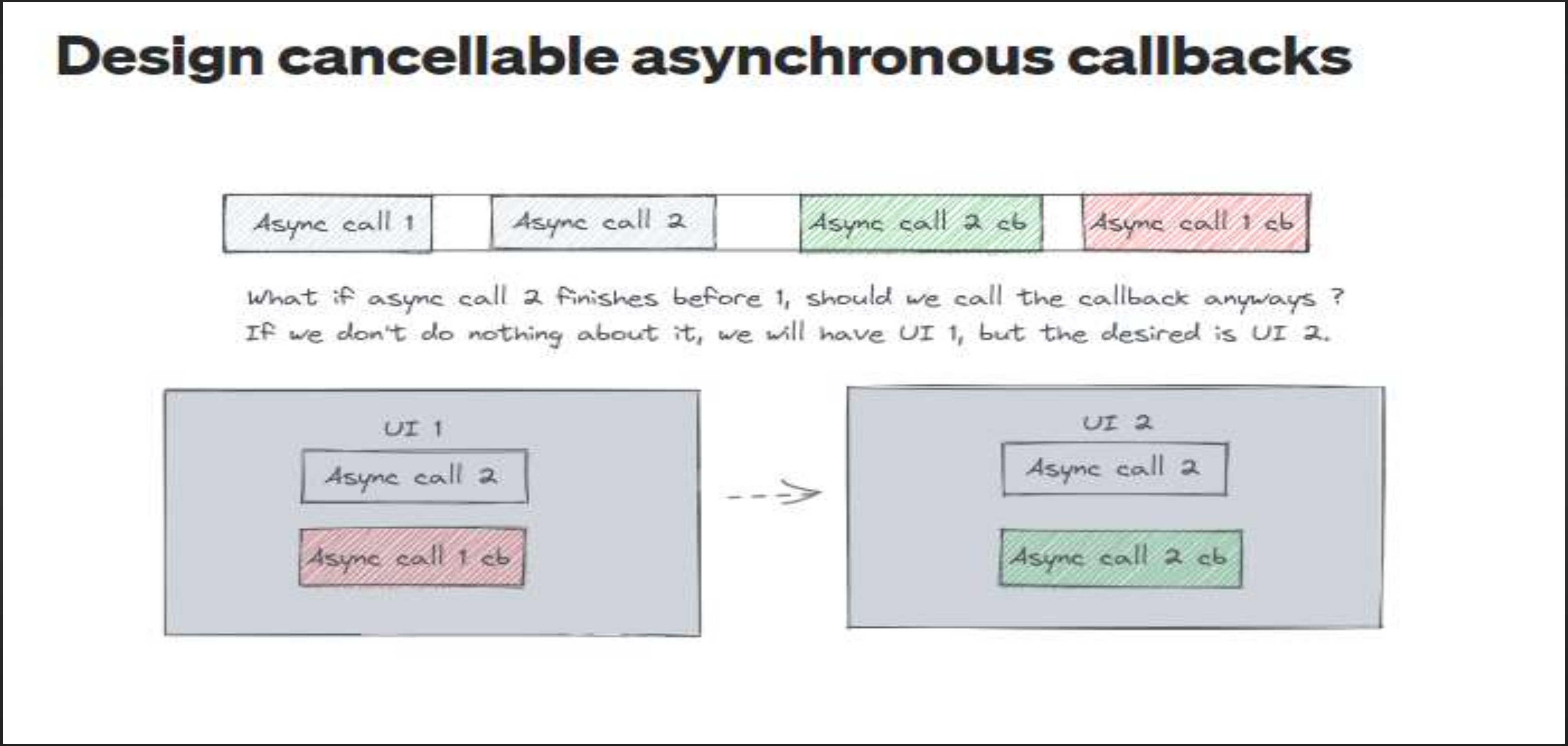
```
React.useEffect(() => {

  let isStale = false;
  function subscriptionFn(newState) {
    if (!isStale) { // do work only if not cleanup
      setState(newState);
    }
  }

  const unsubscribe = store.subscribe(subscriptionFn);

  return () => {
    isStale = true; // mark cleanup
    unsubscribe?.();
  }
}, [store]);
```

State crimes x And x State police



```
// search and later update state
<Button onClick={() => performSearch()} {...} />

// full version
// BAD AND BUGGY CODE, DON'T COPY PASTE OR EVEN CONSIDER AS "MAY BE IT WORKS"
function performSearch() {
  setIsLoading(true);
  timeout(5000/values.length) // the longer the search term, the faster we respond
    .then(() => search(values))
    .then(result => {setData(result), setError(null)})
    .catch(e => {setData(null), setError(null)})
    .finally(() => setIsLoading(false));
}
```

State crimes x And x State police

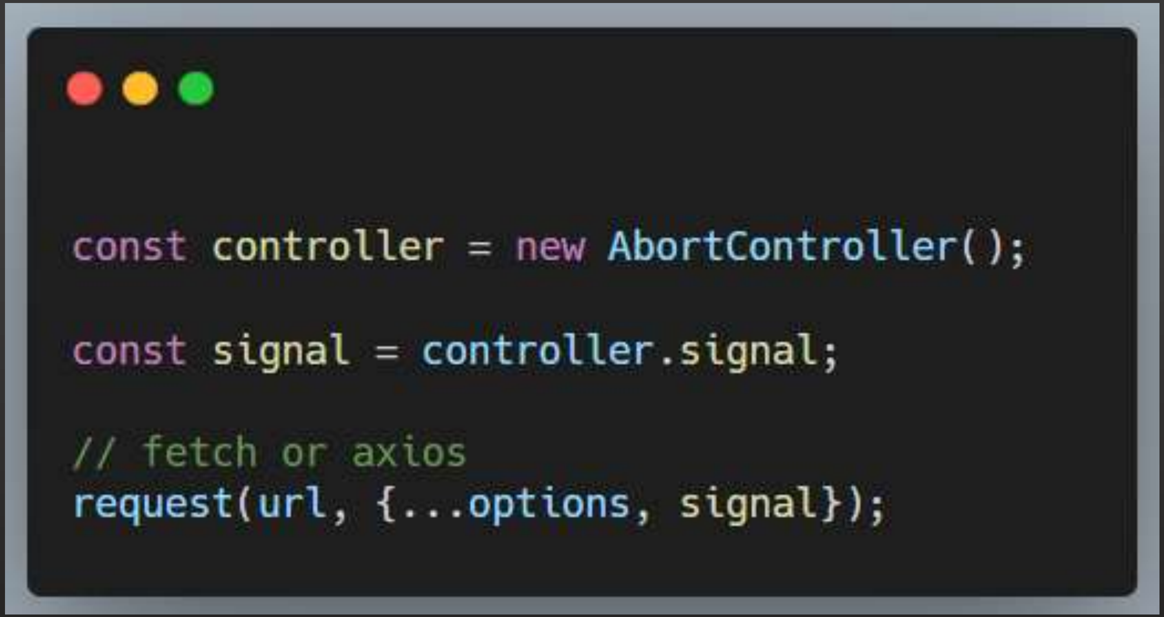


Design cancellable asynchronous callbacks

What if async call 2 finishes before 1, should we call the callback anyways ?
If we don't do nothing about it, we will have UI 1, but the desired is UI 2.

State crimes x And x State police

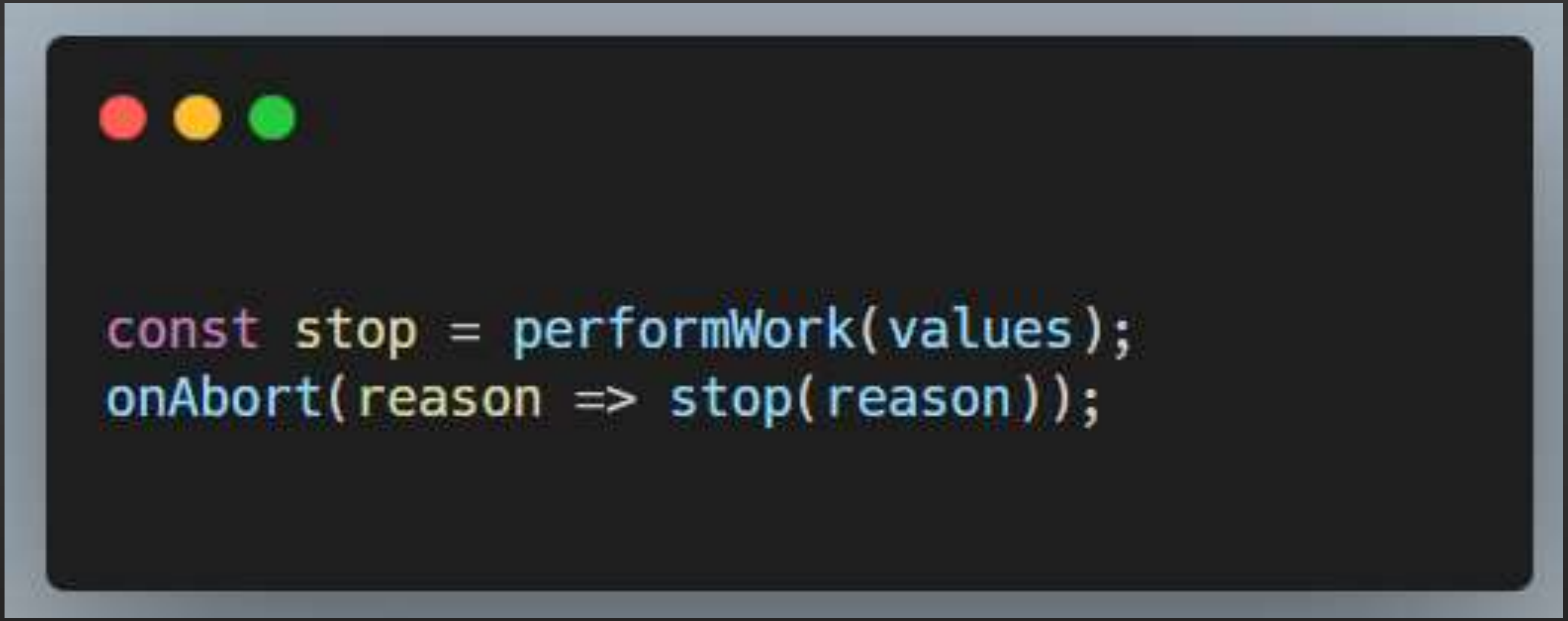
- No cancellations
  - AbortController



```
const controller = new AbortController();

const signal = controller.signal;

// fetch or axios
request(url, {...options, signal});
```

State crimes x And x State police

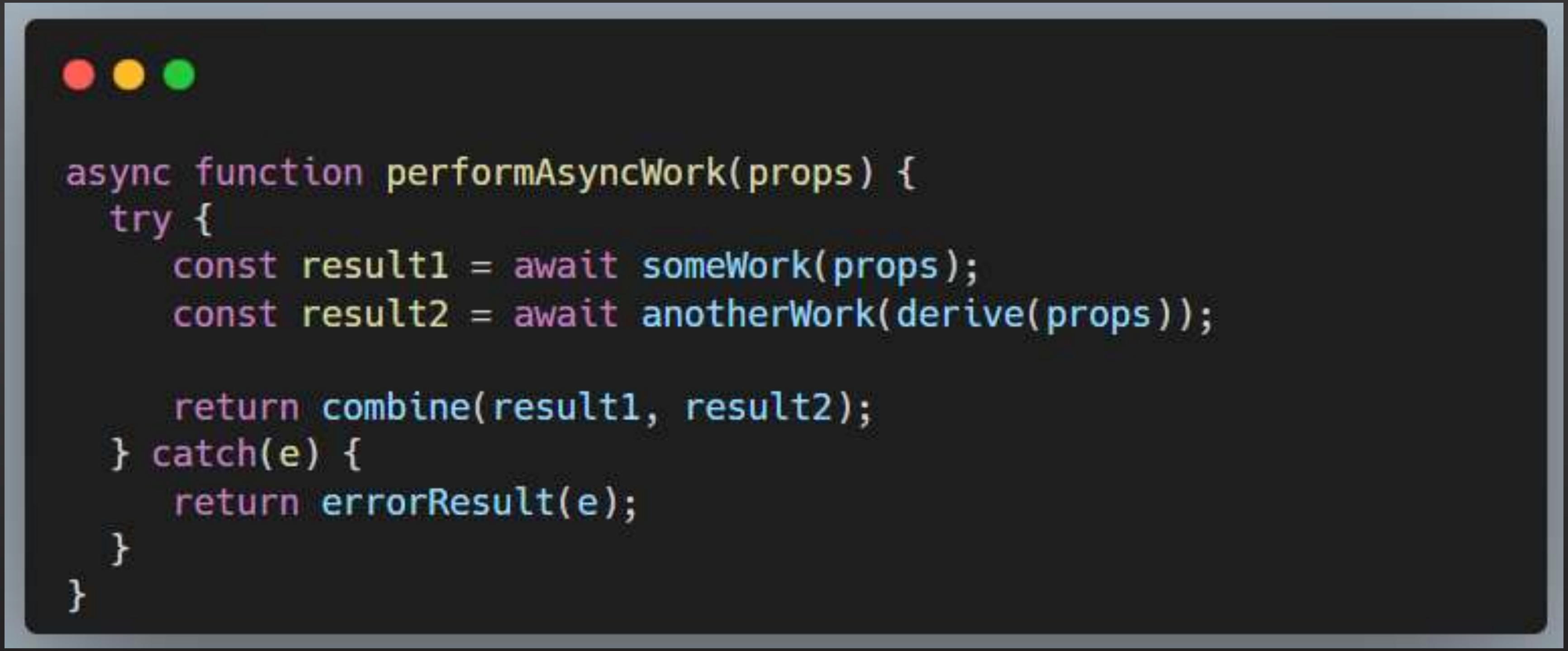
- No cancellations



```
const stop = performWork(values);
onAbort(reason => stop(reason));
```

State crimes x And x State police

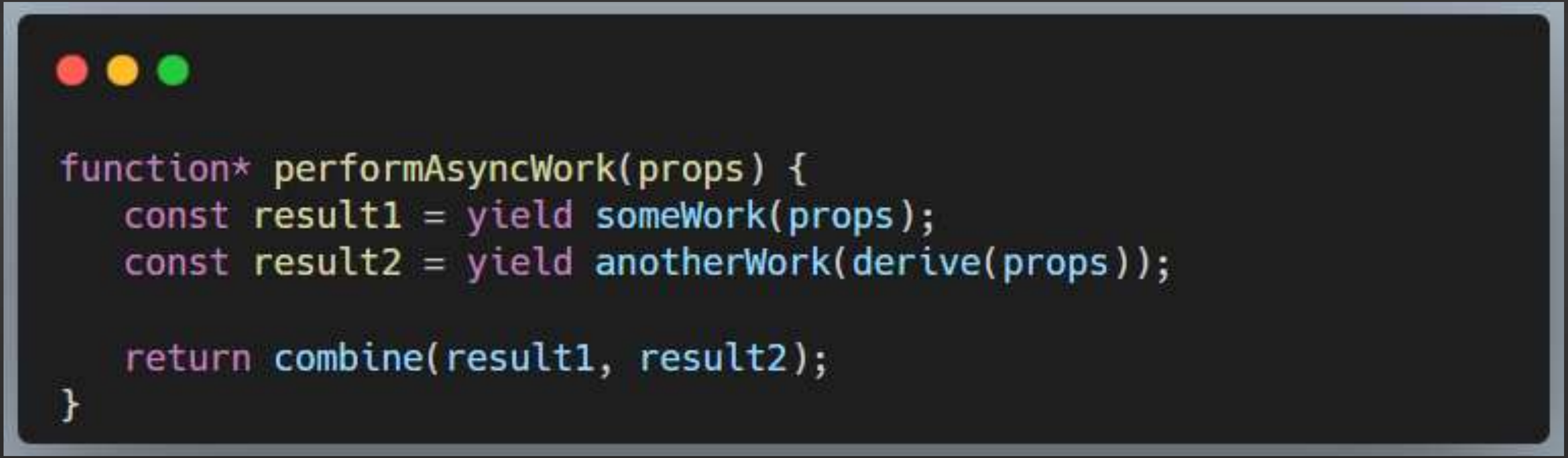
- No cancellations



```
async function performAsyncWork(props) {
  try {
     const result1 = await someWork(props);
     const result2 = await anotherWork(derive(props));

     return combine(result1, result2);
  } catch(e) {
     return errorResult(e);
  }
}
```

State crimes x And x State police

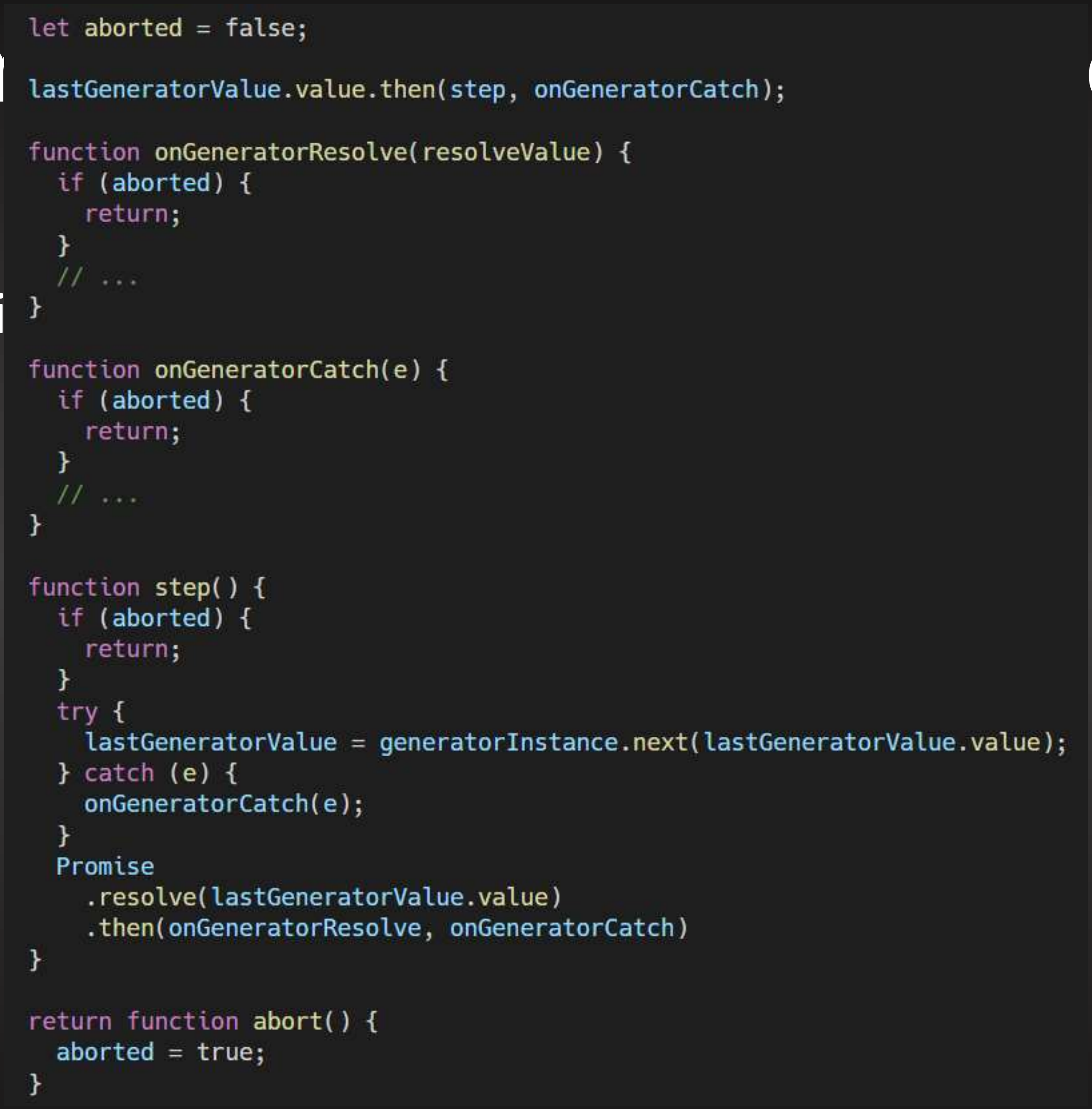
- No cancellations



```
function* performAsyncWork(props) {
   const result1 = yield someWork(props);
   const result2 = yield anotherWork(derive(props));

   return combine(result1, result2);
}
```

State crin e

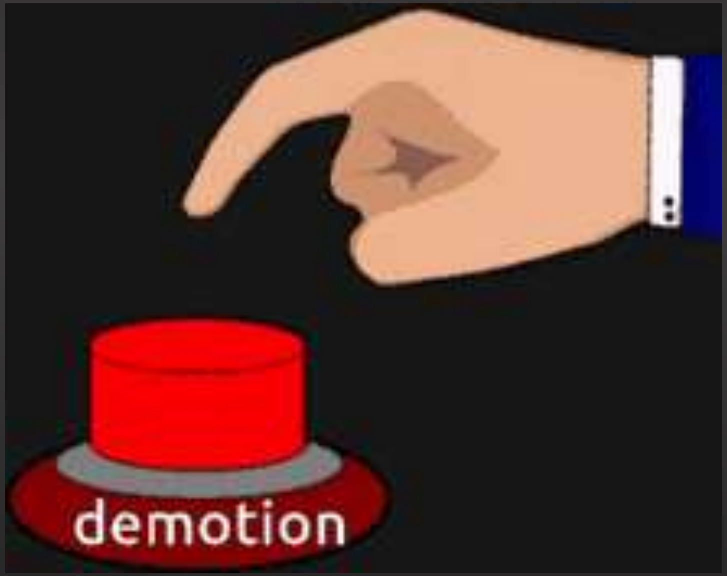
- No cancellati



```
let aborted = false;

lastGeneratorValue.value.then(step, onGeneratorCatch);

function onGeneratorResolve(resolveValue) {
  if (aborted) {
    return;
  }
  // ...
}

function onGeneratorCatch(e) {
  if (aborted) {
    return;
  }
  // ...
}

function step() {
  if (aborted) {
    return;
  }
  try {
    lastGeneratorValue = generatorInstance.next(lastGeneratorValue.value);
  } catch (e) {
    onGeneratorCatch(e);
  }
  Promise
    .resolve(lastGeneratorValue.value)
    .then(onGeneratorResolve, onGeneratorCatch)
}

return function abort() {
  aborted = true;
}
```

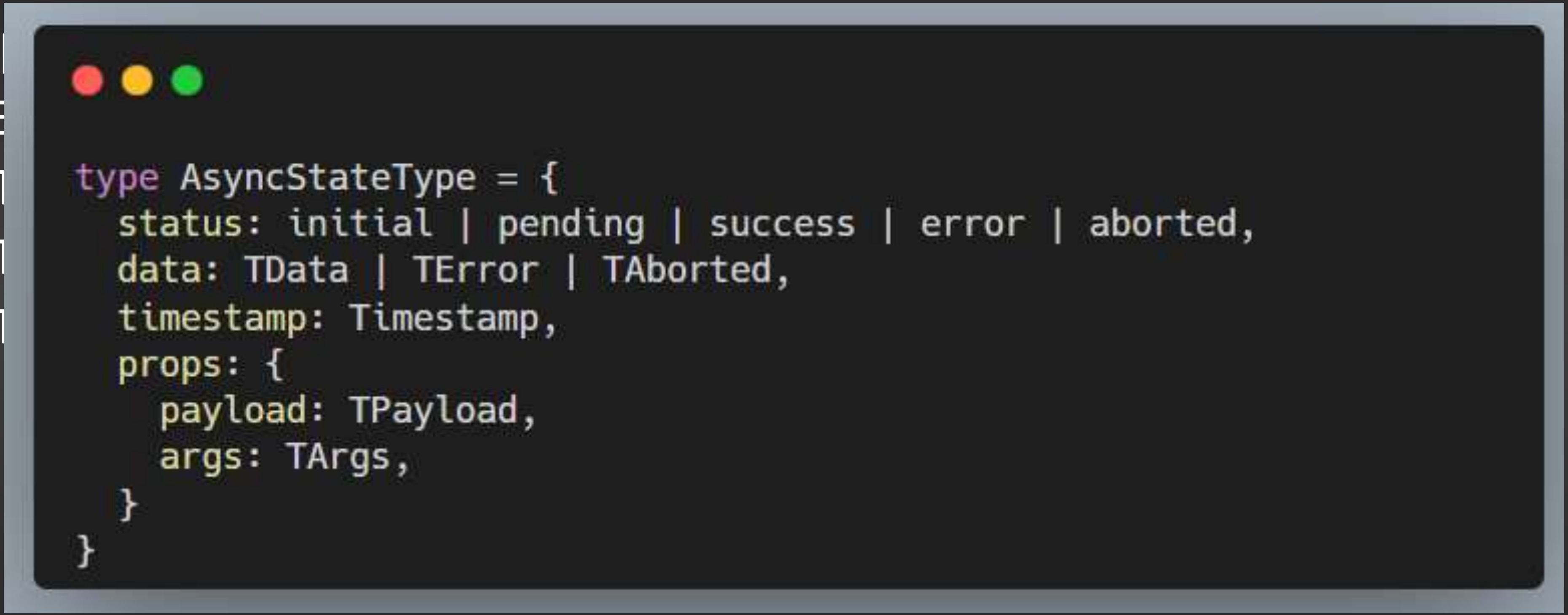
State crimes x And x State police

- No pending states



State crimes x And x State police

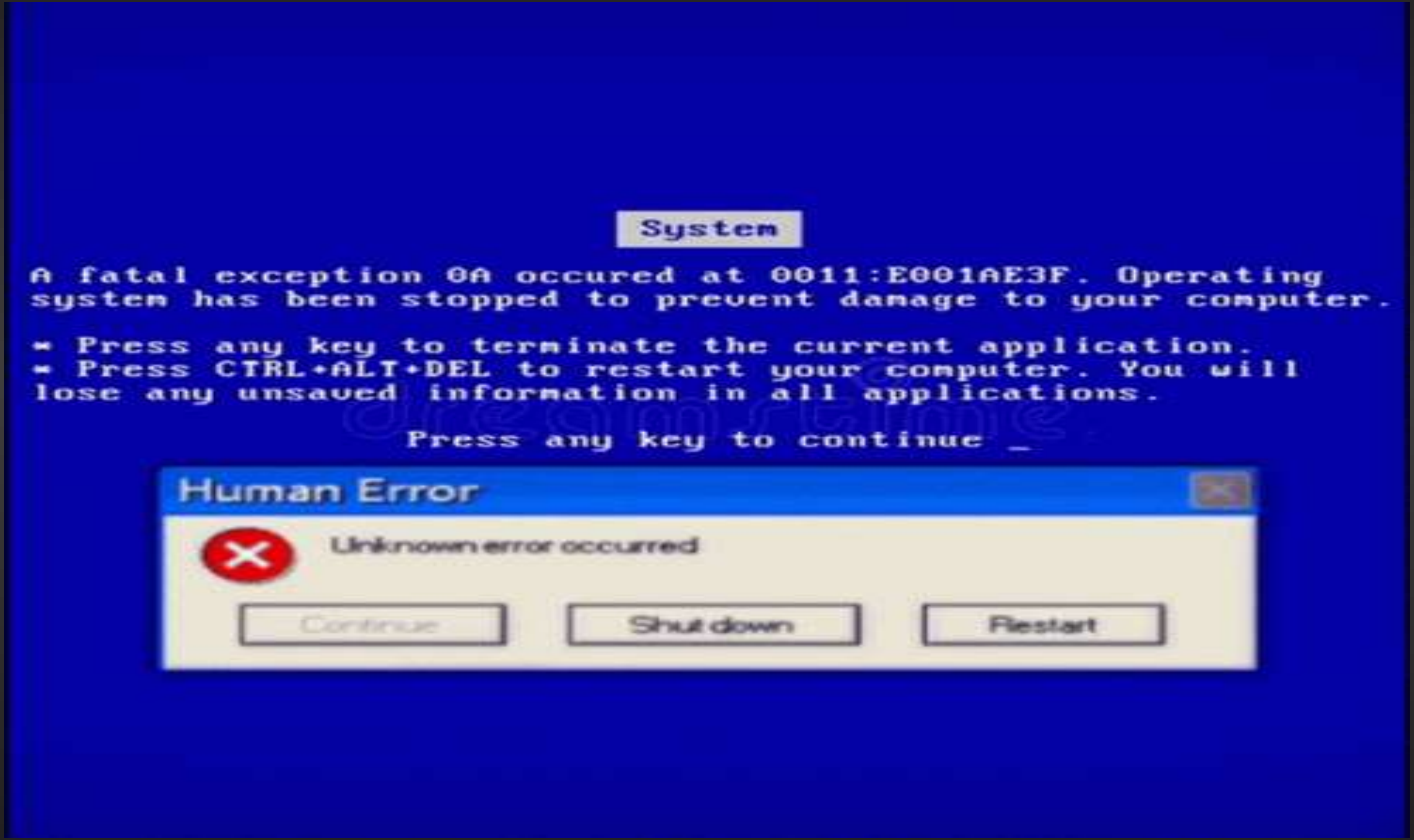
- No
  - E
  - T
  - T
  - T



```
type AsyncStateType = {
  status: initial | pending | success | error | aborted,
  data: TData | TError | TAborted,
  timestamp: Timestamp,
  props: {
    payload: TPayload,
    args: TArgs,
  }
}
```

State crimes x And x State police

- Poor e



System

A fatal exception 0A occured at 0011:E001AE3F. Operating system has been stopped to prevent damage to your computer.

* Press any key to terminate the current application.
* Press CTRL+ALT+DEL to restart your computer. You will lose any unsaved information in all applications.

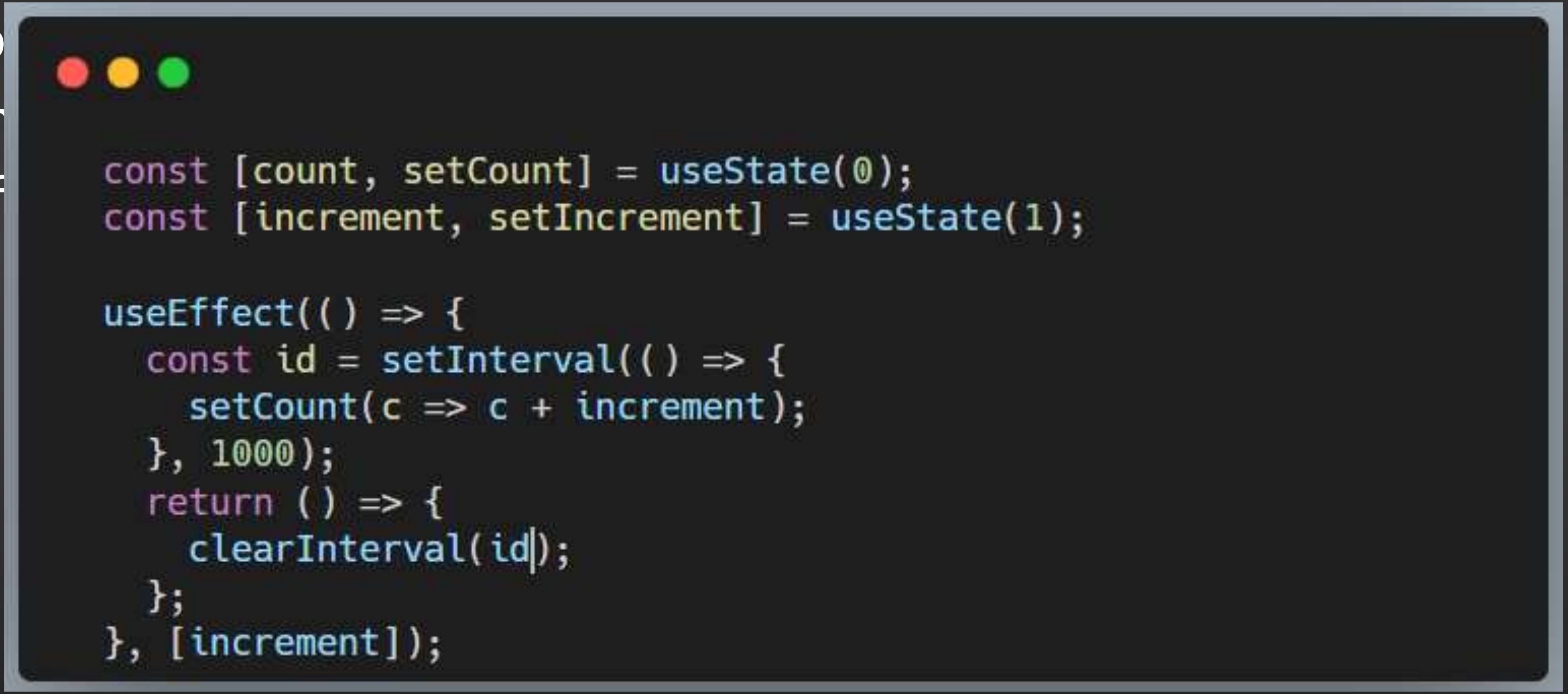
Press any key to continue _

Human Error

Unknown error occurred

State crimes x And x State police

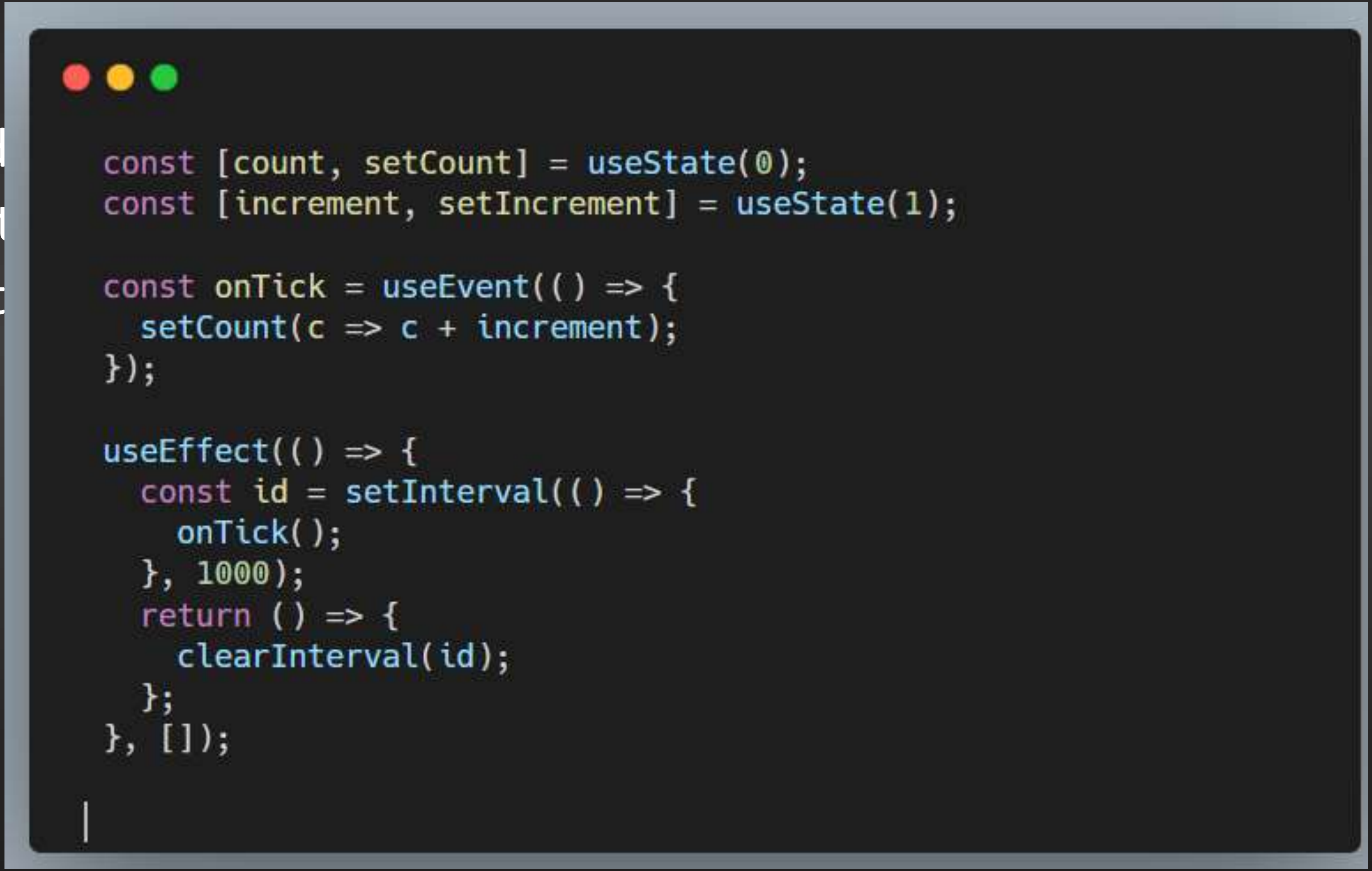
- Useless effects
  - Co
  - On
  - Eff



```
const [count, setCount] = useState(0);
const [increment, setIncrement] = useState(1);

useEffect(() => {
  const id = setInterval(() => {
    setCount(c => c + increment);
  }, 1000);
  return () => {
    clearInterval(id);
  };
}, [increment]);
```

State crimes x And x State police

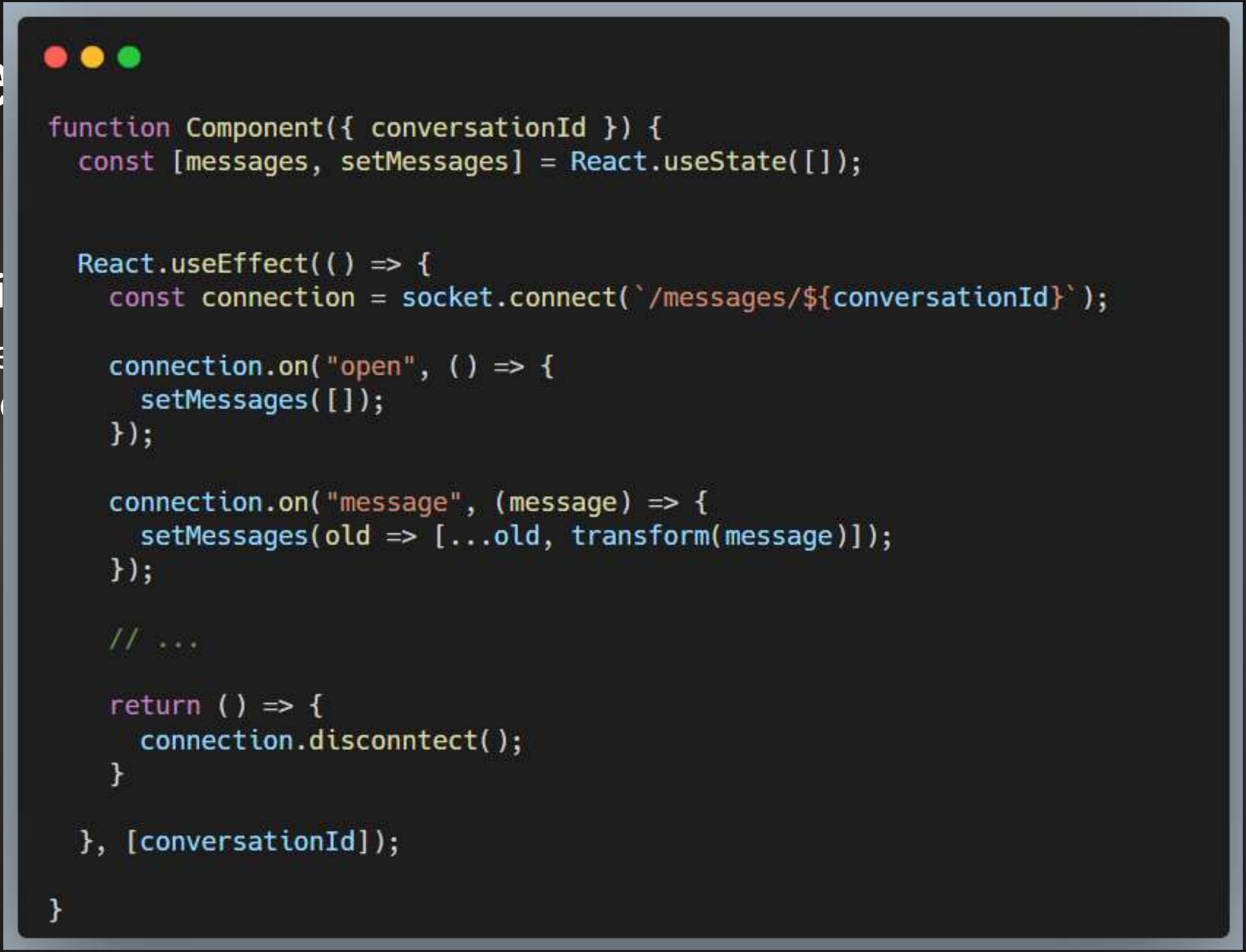
- Useless
  - Could
  - Only t
  - Effect



```
const [count, setCount] = useState(0);
const [increment, setIncrement] = useState(1);

const onTick = useEvent(() => {
  setCount(c => c + increment);
});

useEffect(() => {
  const id = setInterval(() => {
    onTick();
  }, 1000);
  return () => {
    clearInterval(id);
  };
}, []);
```

State

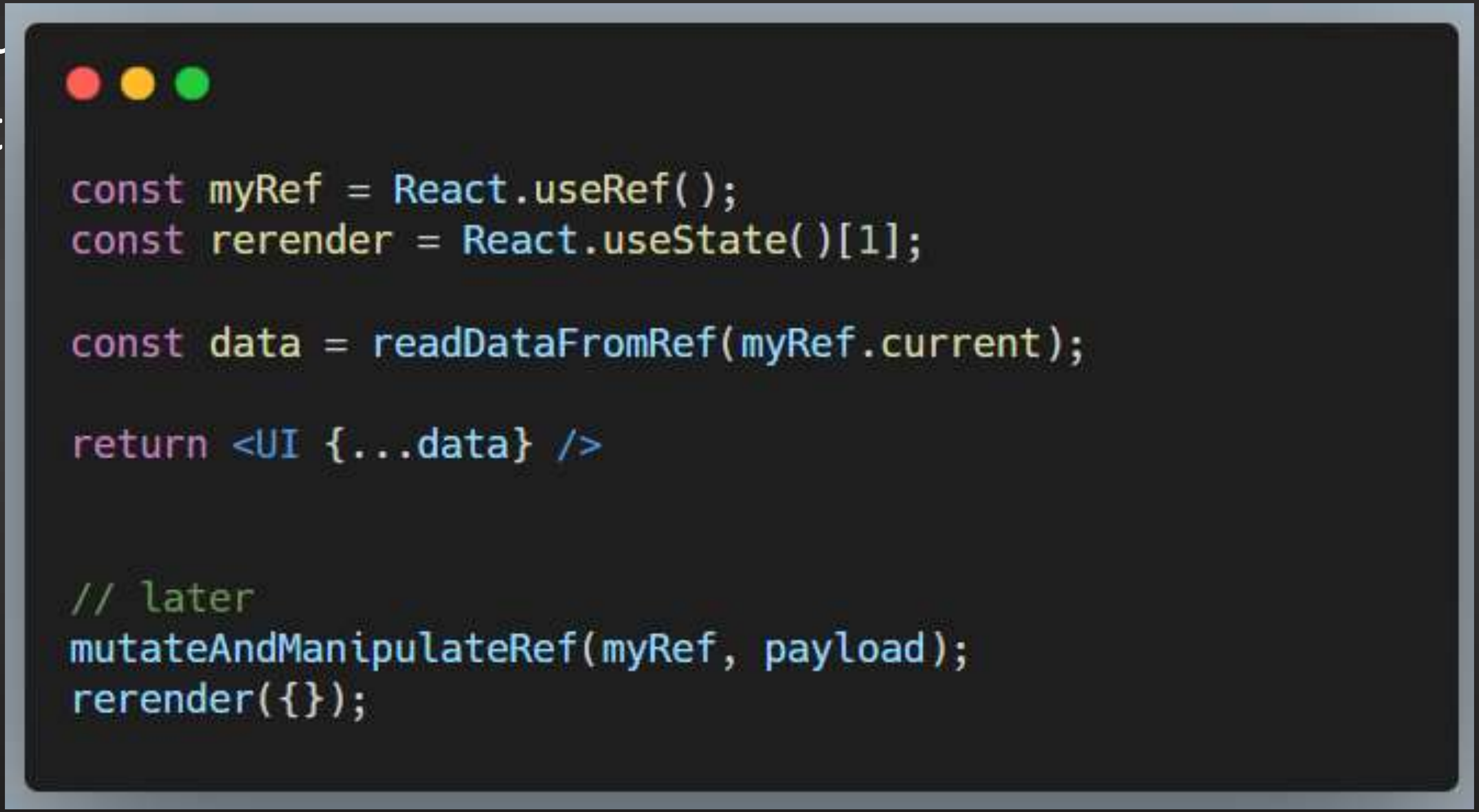
- Flashi
  - Pre   if lucky
    en



```
function Component({ conversationId }) {
  const [messages, setMessages] = React.useState([]);


  React.useEffect(() => {
    const connection = socket.connect(`/messages/${conversationId}`);

    connection.on("open", () => {
      setMessages([]);
    });

    connection.on("message", (message) => {
      setMessages(old => [...old, transform(message)]);
    });

    // ...

    return () => {
      connection.disconntect();
    }

  }, [conversationId]);

}
```

State crimes x And x State police

- Manipu
- The list



```
const myRef = React.useRef();
const rerender = React.useState()[1];

const data = readDataFromRef(myRef.current);

return <UI {...data} />


// later
mutateAndManipulateRef(myRef, payload);
rerender({});
```

State crimes x And x State police

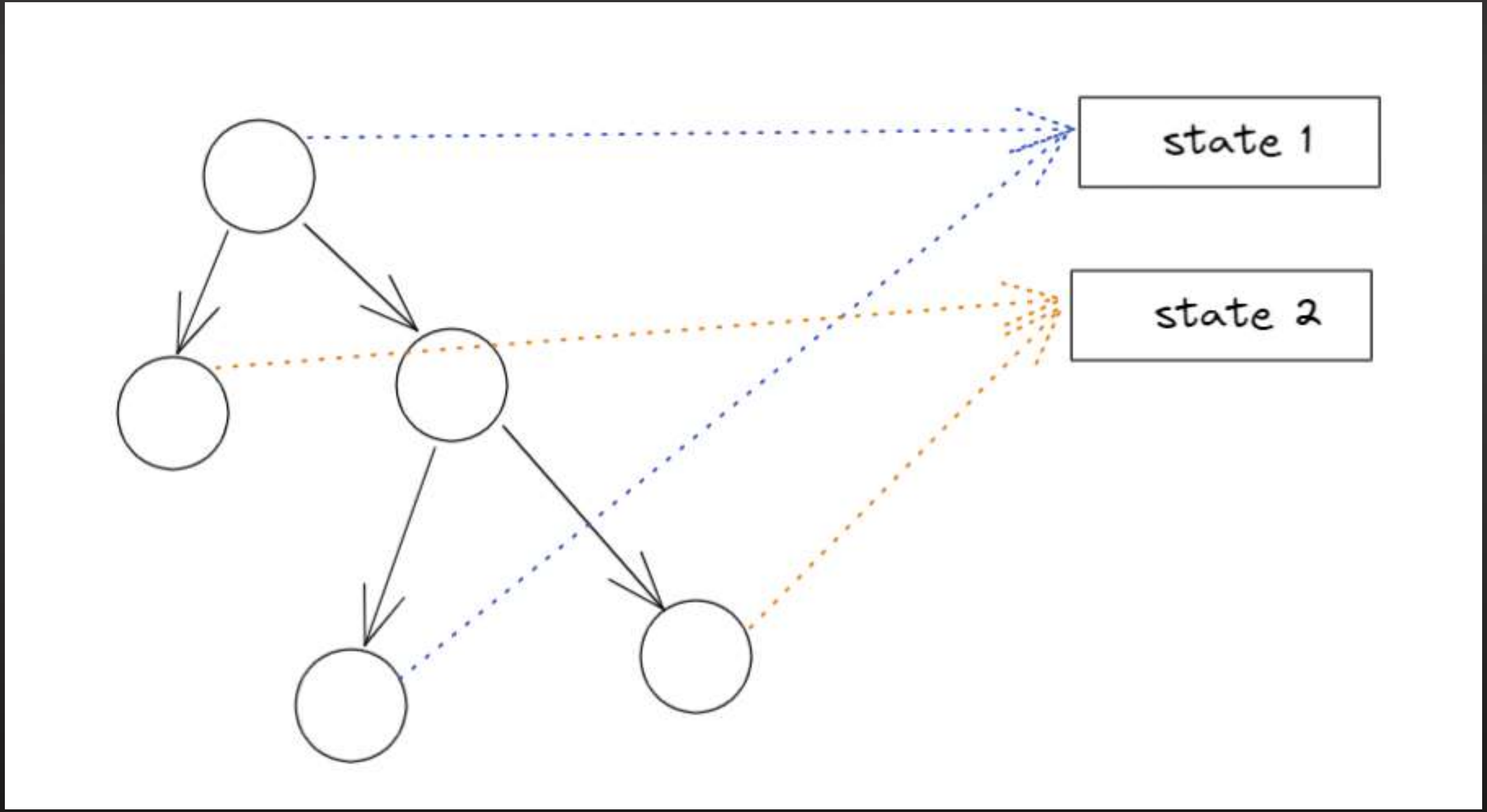
- Linters provide great help to follow the rules of hooks
  - Rules of hooks
  - Exhaustive deps
  - …
- Blog posts and tutorials try their maximum to share good practices
  - Dan’s overreacted blog
  - Kent’s blogs and learning material
- Community libraries makes our life easier
  - UI libraries
  - State managers
  - XState
- State managers leverage the complexity but create bad habits.
- React Strict Mode

State x Management x Aspects

- Sharing
- Subscriptions
- Asynchronous flows
- Cancellations
- Forks
- Effects: debounce, throttle, delay…
- Caching (with/without persistence)
- React/root independent

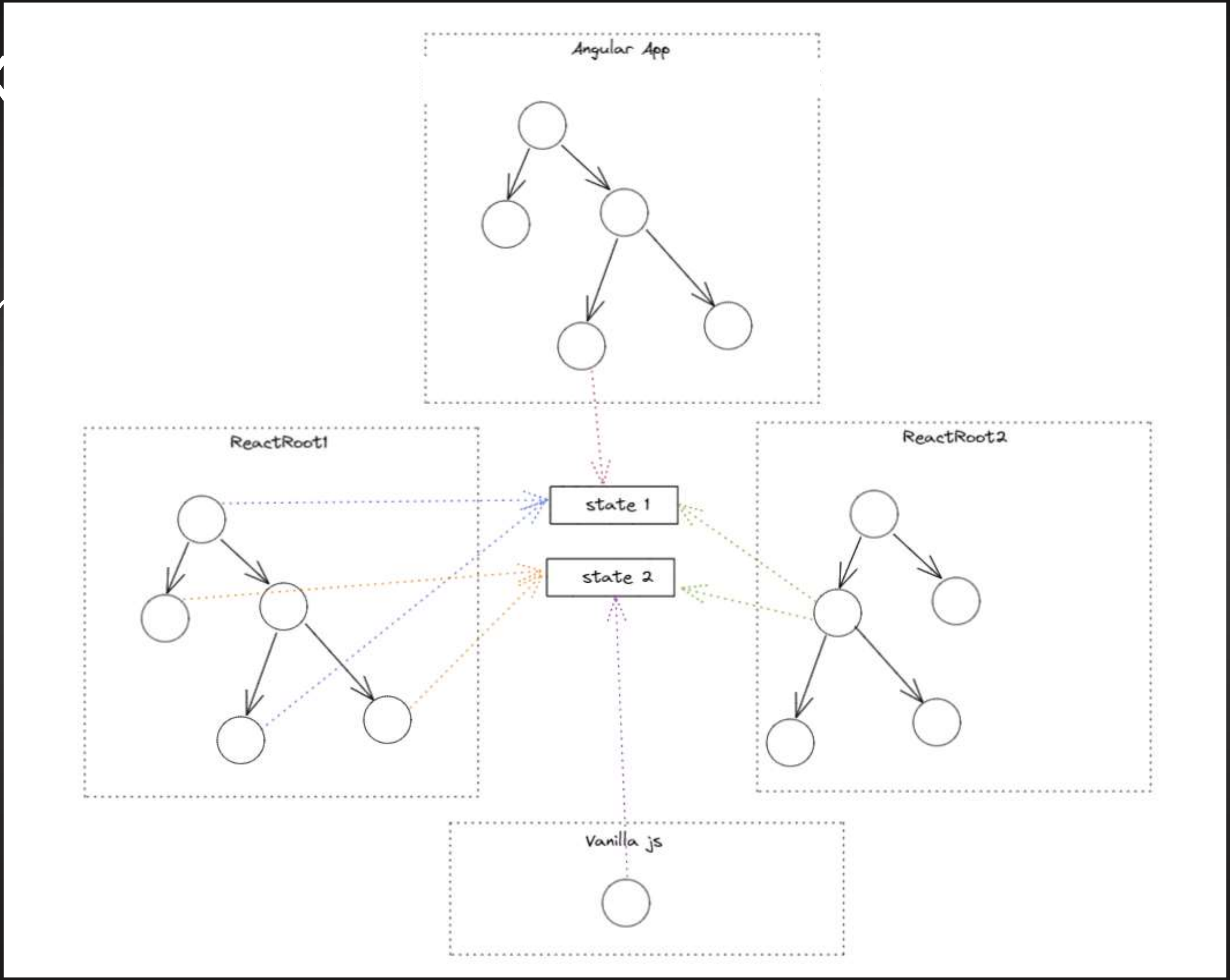
State x Management x Aspects

- Sharing



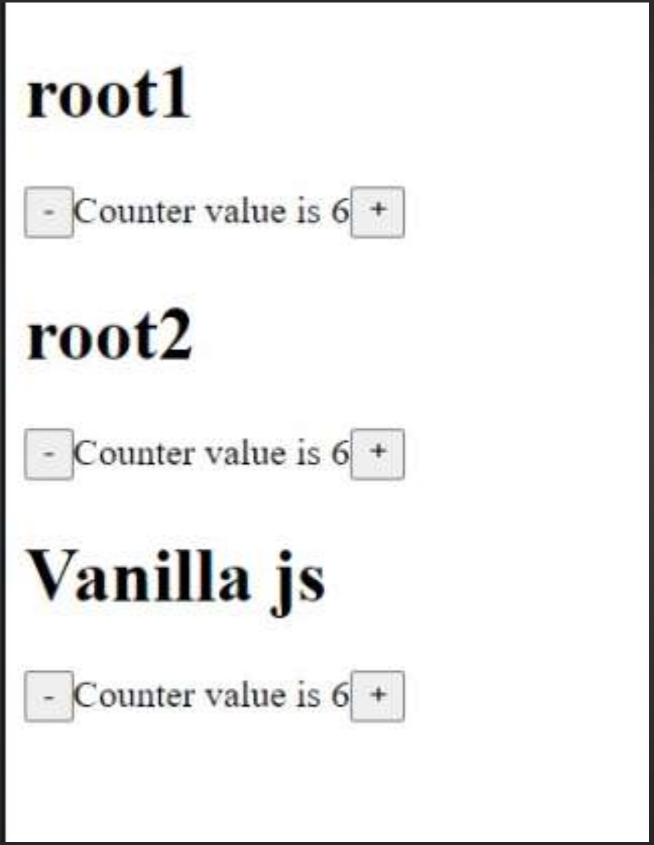
State x

- Subscr



State x Management x Aspects

- Subscriptions



root1

- Counter value is 6 +

root2

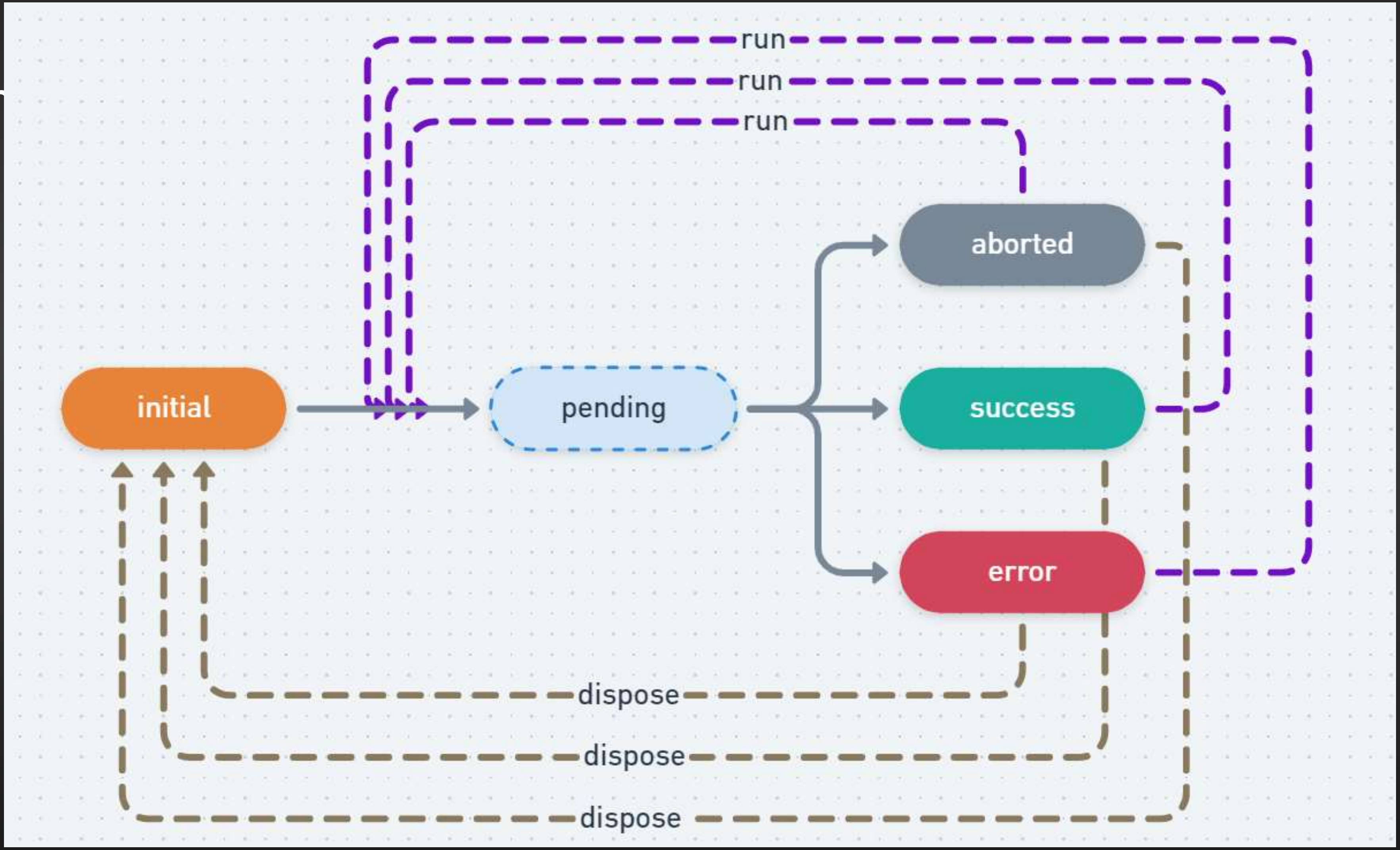
- Counter value is 6 +

Vanilla js

- Counter value is 6 +

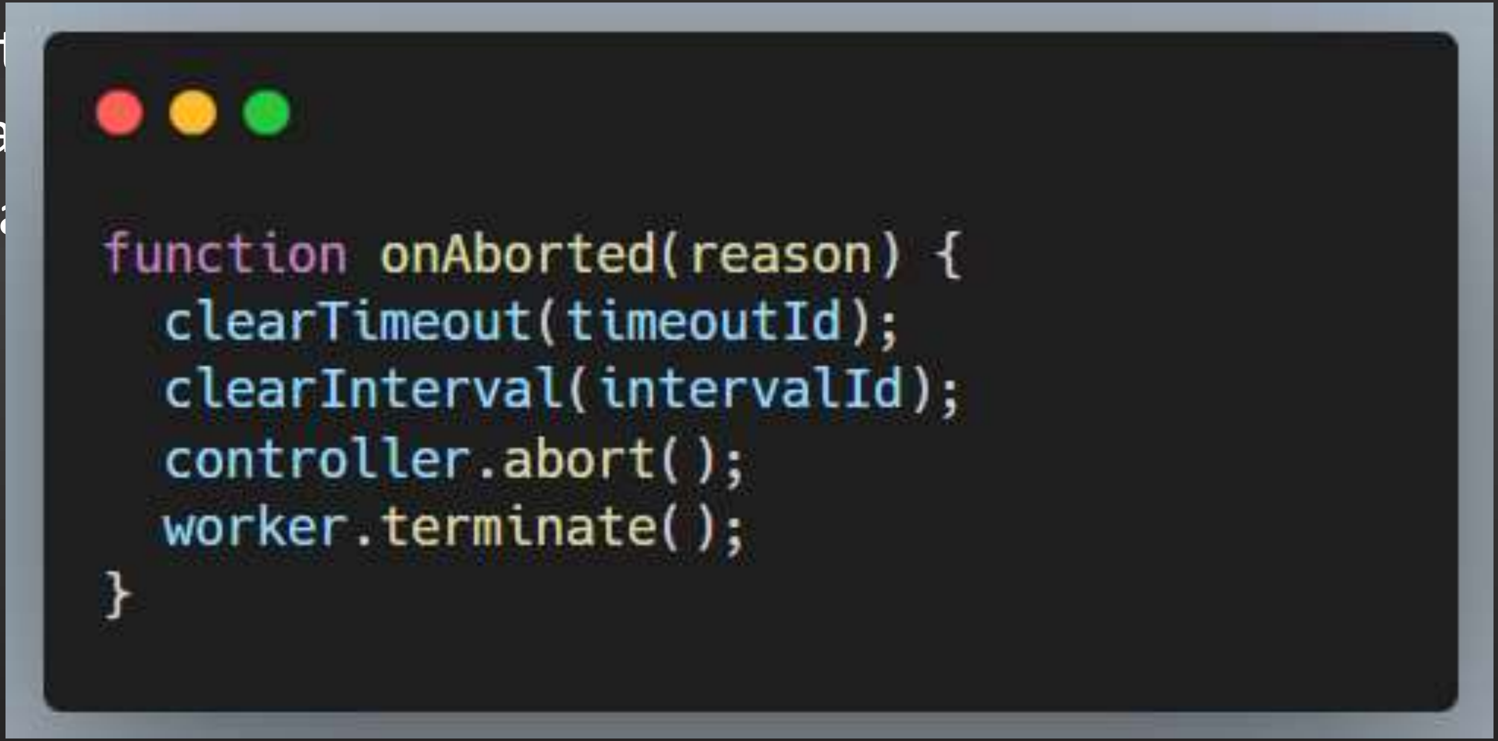
State x Management x Aspects

- Asynch



State x Management x Aspects

- Cancellations
  - Generat
  - Async/a
  - Abort ca



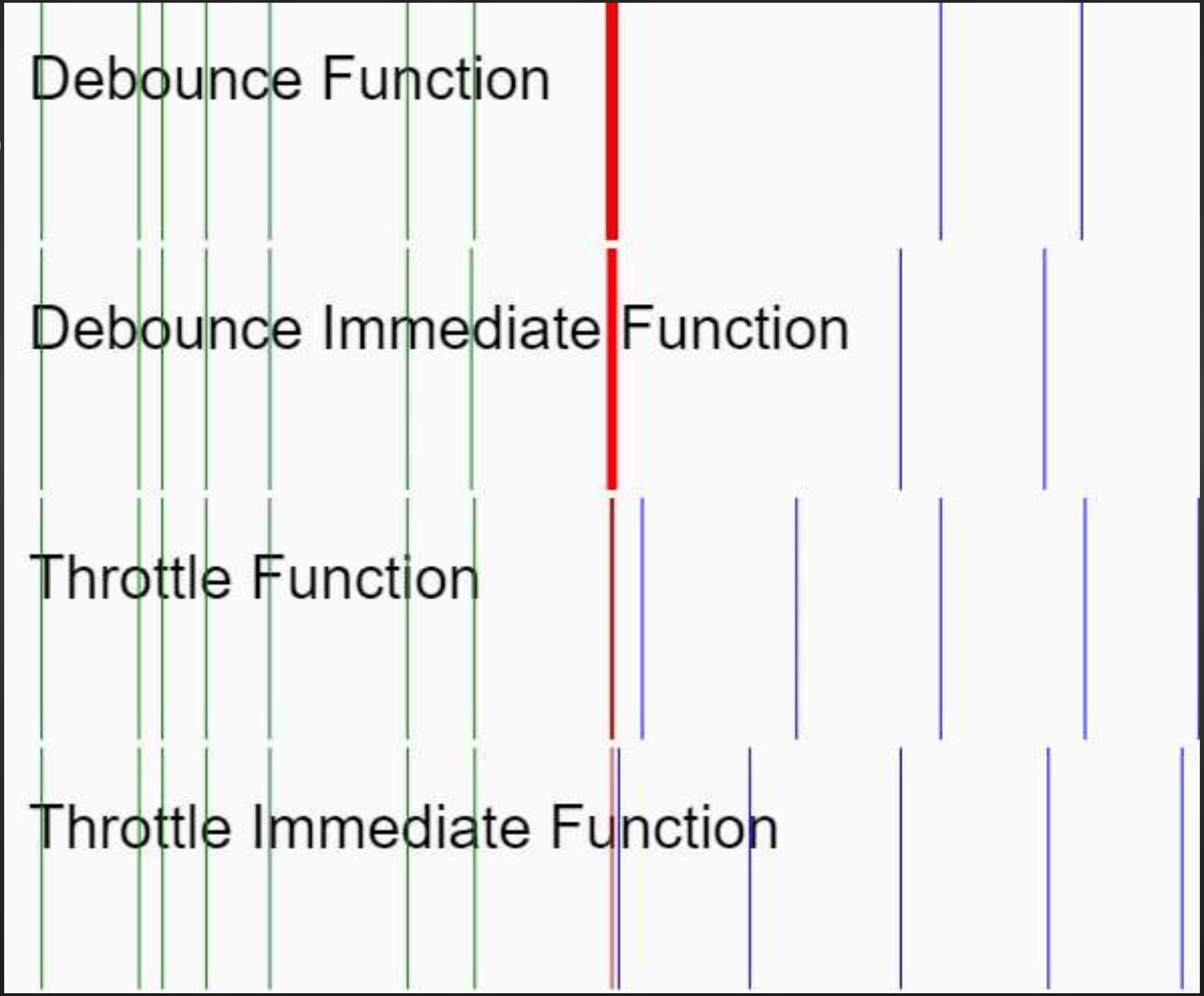
```
function onAborted(reason) {
  clearTimeout(timeoutId);
  clearInterval(intervalId);
  controller.abort();
  worker.terminate();
}
```

State x Management x Aspects

- Forks
  - Replicate behavior for a new tree

State x Management x Aspects

- Effects: deb

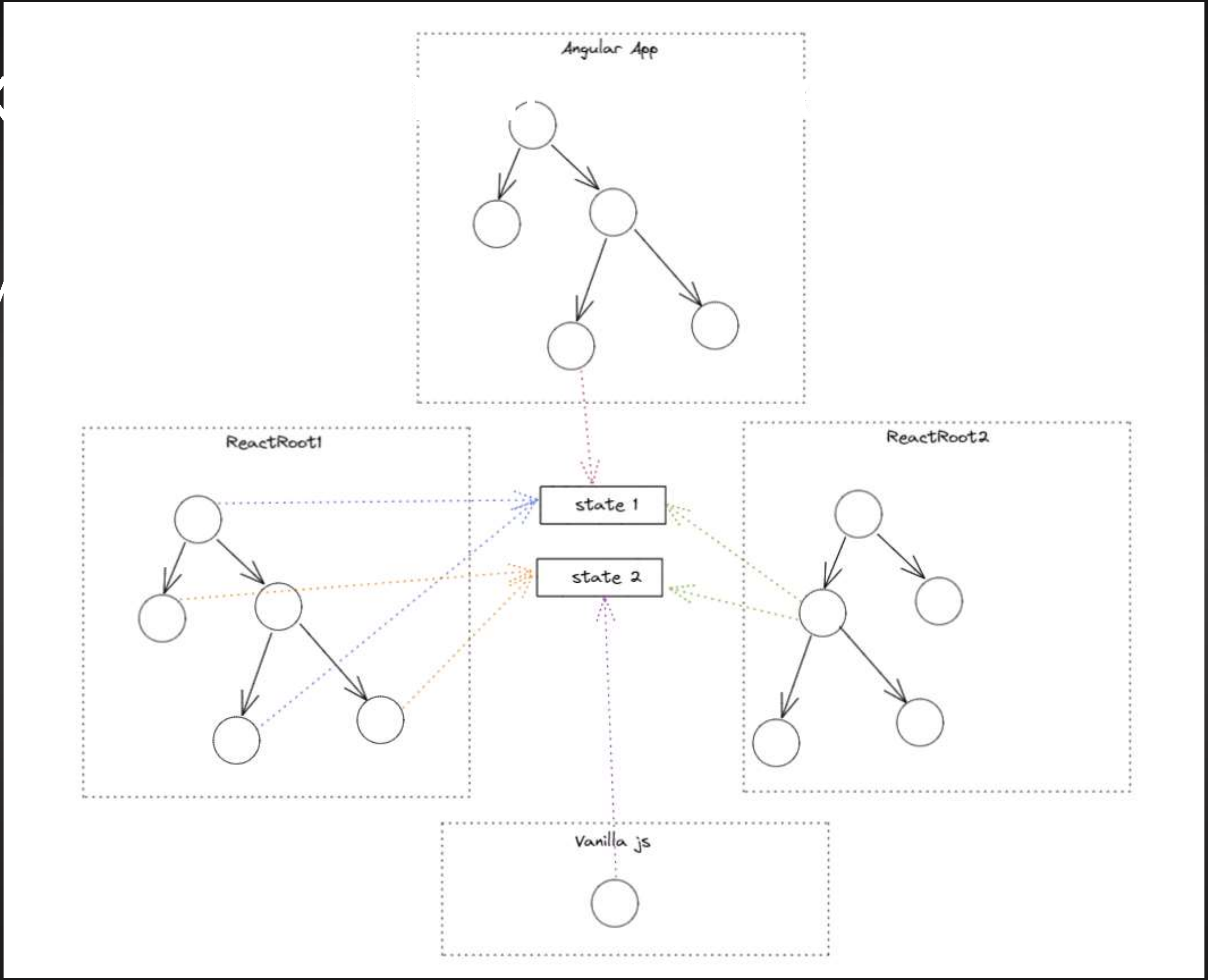


State x Management x Aspects

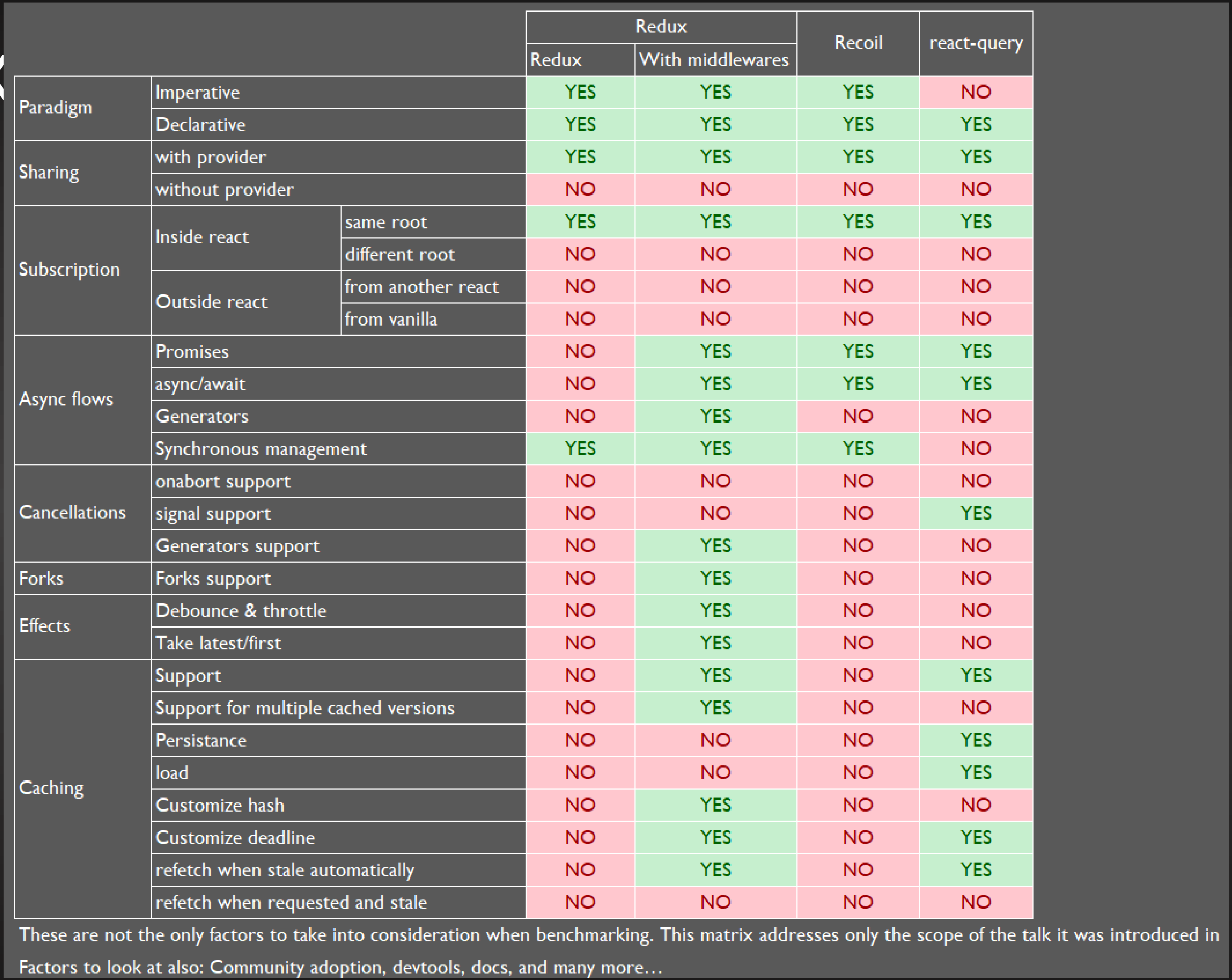
- Caching (with/without persistence)
  - Invalidate and refetch when timed-out
  - Only invalidate when requested and state

State x

- React/



State x

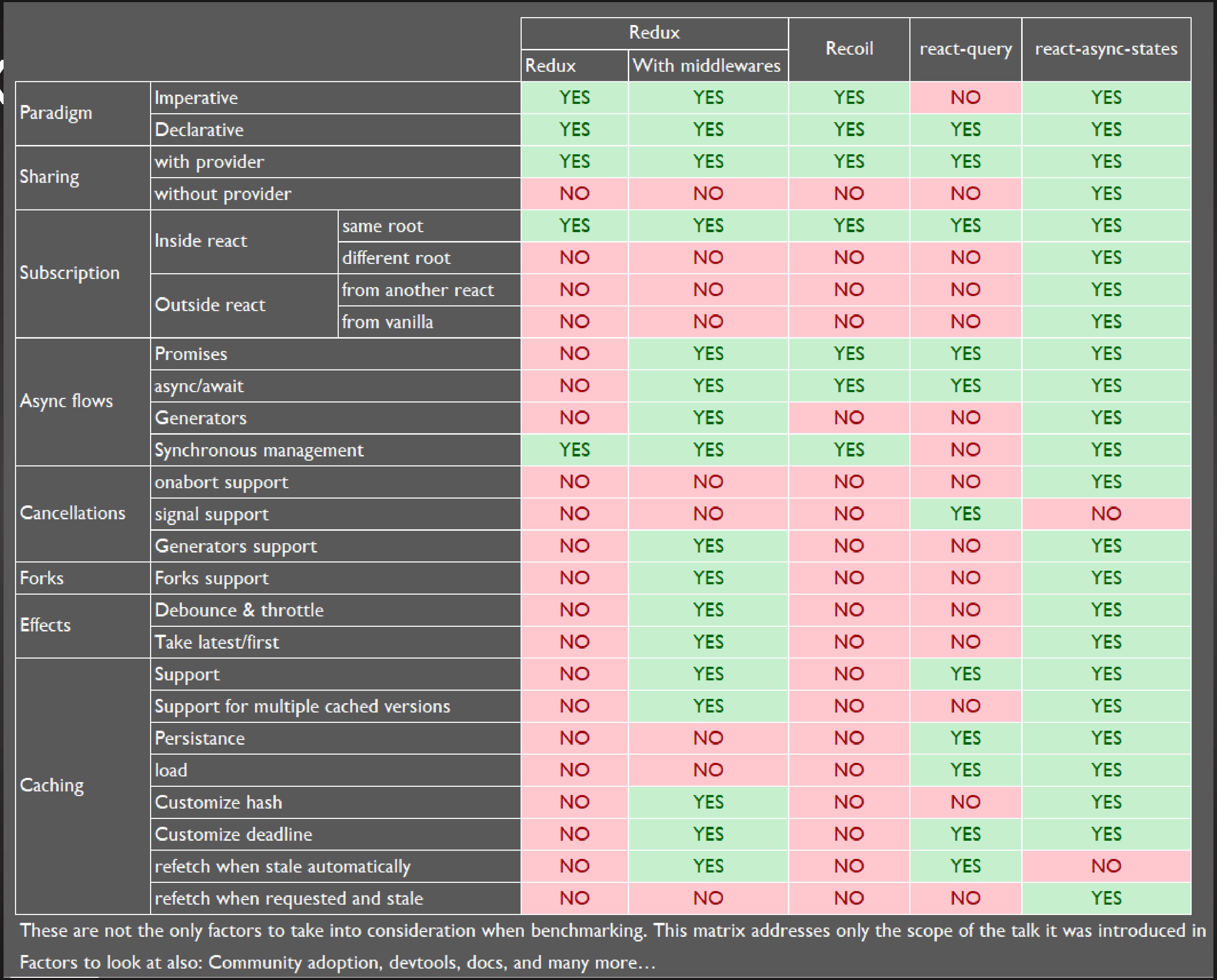


			Redux		Recoil	react-query
			Redux	With middlewares		
Paradigm	Imperative		YES	YES	YES	NO
	Declarative		YES	YES	YES	YES
Sharing	with provider		YES	YES	YES	YES
	without provider		NO	NO	NO	NO
Subscription	Inside react	same root	YES	YES	YES	YES
		different root	NO	NO	NO	NO
	Outside react	from another react	NO	NO	NO	NO
		from vanilla	NO	NO	NO	NO
Async flows	Promises		NO	YES	YES	YES
	async/await		NO	YES	YES	YES
	Generators		NO	YES	NO	NO
	Synchronous management		YES	YES	YES	NO
Cancellations	onabort support		NO	NO	NO	NO
	signal support		NO	NO	NO	YES
	Generators support		NO	YES	NO	NO
Forks	Forks support		NO	YES	NO	NO
Effects	Debounce & throttle		NO	YES	NO	NO
	Take latest/first		NO	YES	NO	NO
Caching	Support		NO	YES	NO	YES
	Support for multiple cached versions		NO	YES	NO	NO
	Persistance		NO	NO	NO	YES
	load		NO	NO	NO	YES
	Customize hash		NO	YES	NO	NO
	Customize deadline		NO	YES	NO	YES
	refetch when stale automatically		NO	YES	NO	YES
	refetch when requested and stale		NO	NO	NO	NO

These are not the only factors to take into consideration when benchmarking. This matrix addresses only the scope of the talk it was introduced in

Factors to look at also: Community adoption, devtools, docs, and many more…

State x



			Redux		Recoil	react-query	react-async-states
			Redux	With middlewares			
Paradigm	Imperative		YES	YES	YES	NO	YES
	Declarative		YES	YES	YES	YES	YES
Sharing	with provider		YES	YES	YES	YES	YES
	without provider		NO	NO	NO	NO	YES
Subscription	Inside react	same root	YES	YES	YES	YES	YES
		different root	NO	NO	NO	NO	YES
	Outside react	from another react	NO	NO	NO	NO	YES
		from vanilla	NO	NO	NO	NO	YES
Async flows	Promises		NO	YES	YES	YES	YES
	async/await		NO	YES	YES	YES	YES
	Generators		NO	YES	NO	NO	YES
	Synchronous management		YES	YES	YES	NO	YES
Cancellations	onabort support		NO	NO	NO	NO	YES
	signal support		NO	NO	NO	YES	NO
	Generators support		NO	YES	NO	NO	YES
Forks	Forks support		NO	YES	NO	NO	YES
Effects	Debounce & throttle		NO	YES	NO	NO	YES
	Take latest/first		NO	YES	NO	NO	YES
Caching	Support		NO	YES	NO	YES	YES
	Support for multiple cached versions		NO	YES	NO	NO	YES
	Persistance		NO	NO	NO	YES	YES
	load		NO	NO	NO	YES	YES
	Customize hash		NO	YES	NO	NO	YES
	Customize deadline		NO	YES	NO	YES	YES
	refetch when stale automatically		NO	YES	NO	YES	NO
	refetch when requested and stale		NO	NO	NO	NO	YES

These are not the only factors to take into consideration when benchmarking. This matrix addresses only the scope of the talk it was introduced in

Factors to look at also: Community adoption, devtools, docs, and many more…

Conclusion x And x Takeaways

- State is the only trigger of updates in react (re-renders). Ok, useSES!
- State setter must be secured from old and stale closures, not only unmount.
- If React.useCallback could just invalidate the previous callback when deps change!
- Status is mandatory when dealing with asynchronous states
- useEffect should not be abused and events should perform more work
- State managers offers great help dealing with state
- Community resources are a gem that should be considered more
- We are all state criminals

Thank you

Mohamed EL AYADI

@incepterr