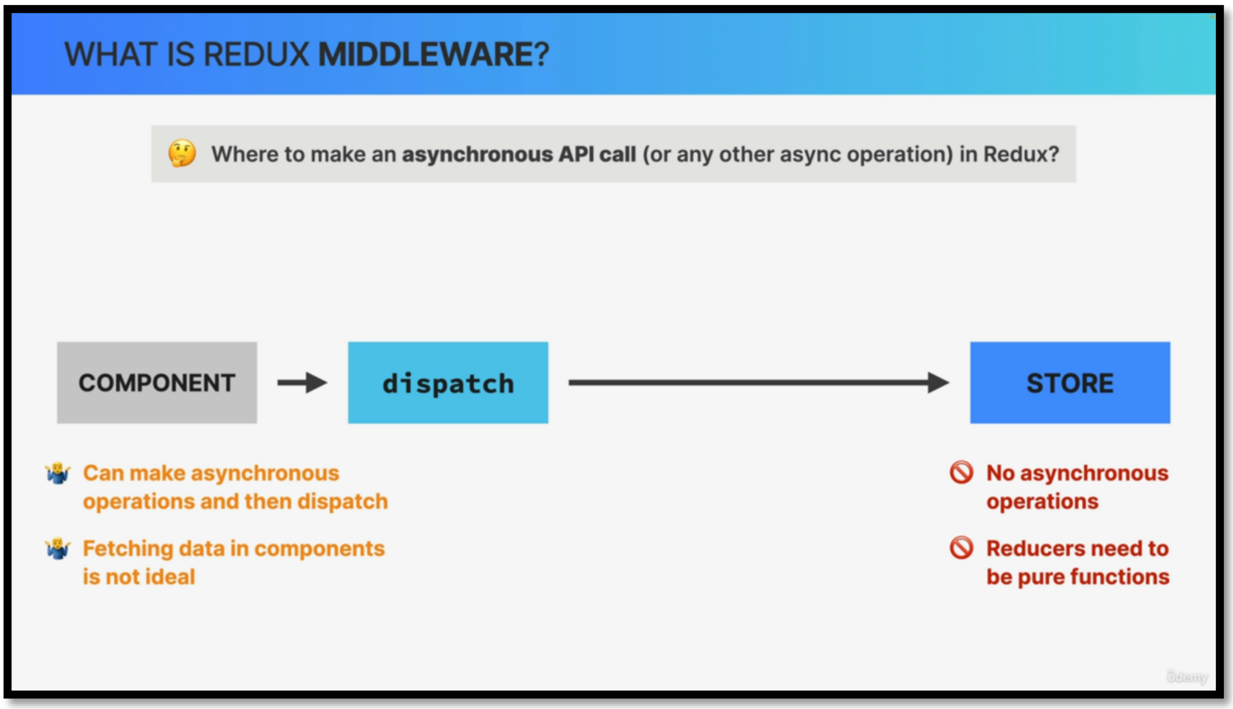


WHAT IS REDUX **MIDDLEWARE?**

Where to make an **asynchronous API call** (or any other async operation) in Redux?

COMPONENT
dispatch
STORE
Can make asynchronous operations and then dispatch
Fetching data in components is not ideal
No asynchronous operations
Reducers need to be pure functions

Let's say that we wanted to make an asynchronous call to some API. So where could we actually do that in Redux?

Well, we can definitely not make the API call inside a reducer because reducers need to be pure functions with no side effects.

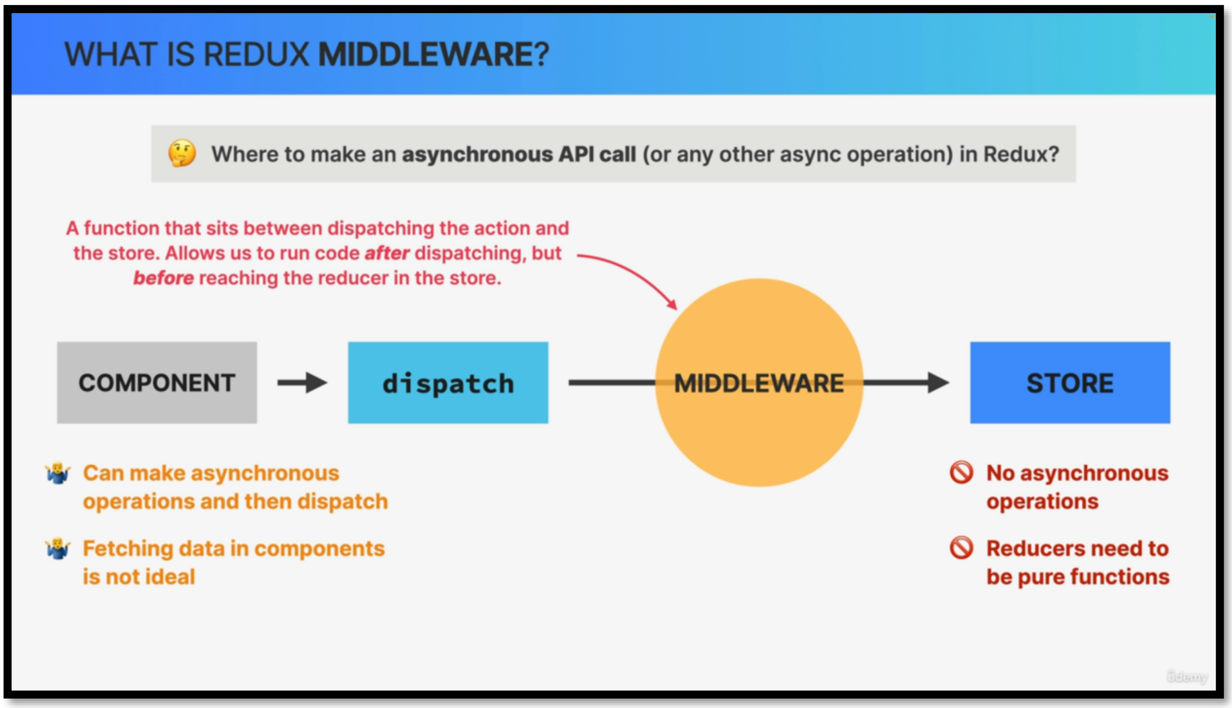
So by itself, a Redux store doesn't know anything about performing asynchronous logic like this. It only knows how to synchronously dispatch actions and update the state. Therefore, any asynchronous operations like that API call need to happen outside a reducer.

So instead, should we maybe fetch the data inside the component and then dispatch an action to the store with that received data?

Well, that is actually possible, but it's not an ideal solution and the reason for that is that we usually want to keep our components clean and free of data fetching and we also want our important data fetching logic encapsulated somewhere, so all in one place and not have it spread all over the application.

Therefore, fetching data inside components, like we have been doing all this time, is not ideal.

But if not in the store and not in the components, then where do we perform asynchronous actions?



WHAT IS REDUX **MIDDLEWARE**?

🤔 Where to make an **asynchronous API call** (or any other async operation) in Redux?

A function that sits between dispatching the action and the store. Allows us to run code *after* dispatching, but *before* reaching the reducer in the store.

COMPONENT → dispatch → MIDDLEWARE → STORE

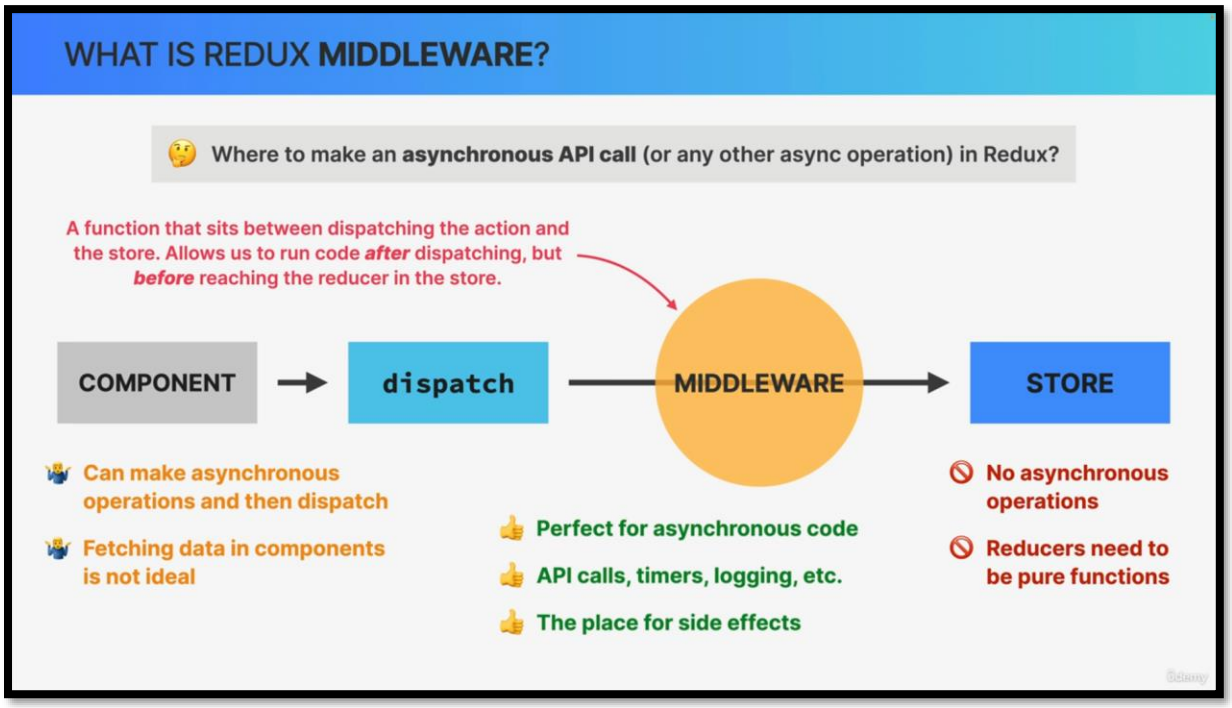
Can make asynchronous operations and then dispatch

Fetching data in components is not ideal

🚫 No asynchronous operations

🚫 Reducers need to be pure functions

Well, that's where Middleware comes into action. So, in Redux, Middleware is basically a function that sits between the dispatching and the store.



WHAT IS REDUX **MIDDLEWARE**?

🤔 Where to make an **asynchronous API call** (or any other async operation) in Redux?

A function that sits between dispatching the action and the store. Allows us to run code *after* dispatching, but *before* reaching the reducer in the store.

COMPONENT → dispatch → MIDDLEWARE → STORE

Can make asynchronous operations and then dispatch

Fetching data in components is not ideal

👍 Perfect for asynchronous code

👍 API calls, timers, logging, etc.

👍 The place for side effects

🚫 No asynchronous operations

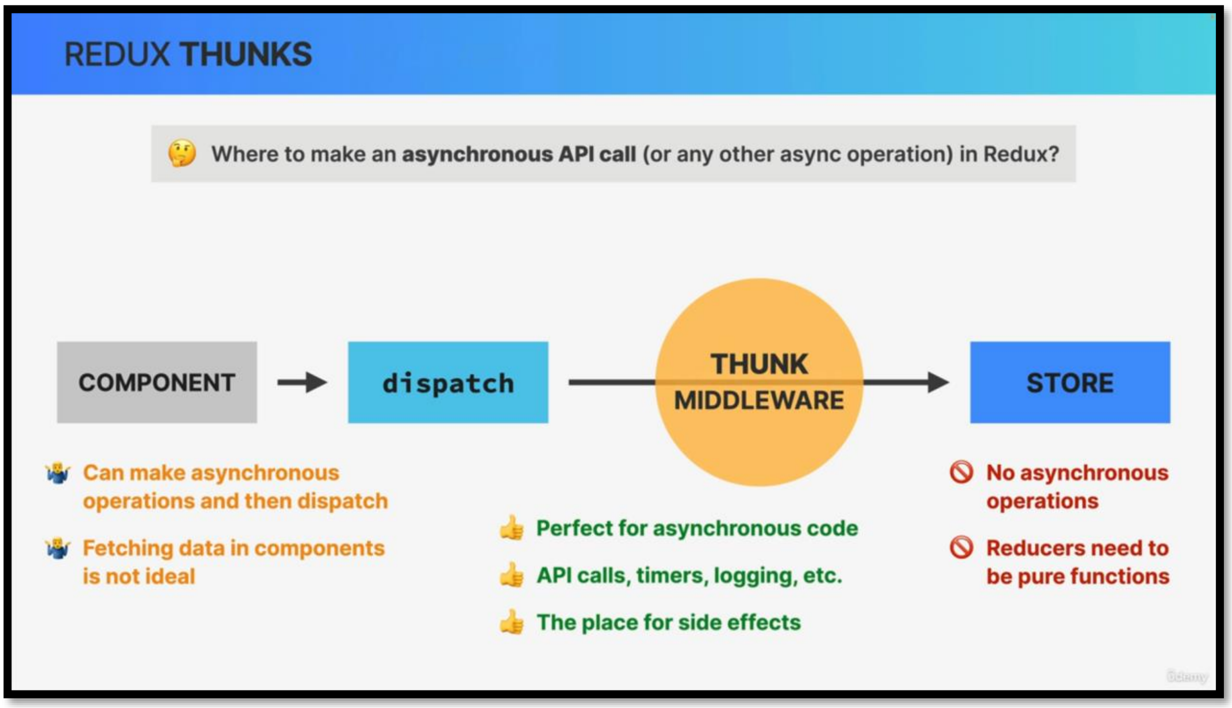
🚫 Reducers need to be pure functions

This means that a Middleware allows developers to run some code after dispatching an action, but before that action reaches the reducer in the store.

So again, usually after we dispatch, the action immediately reaches the reducer and the state is updated.

But with a Middleware, we can do something with the action before that action actually gets into the reducer and therefore, this is the perfect place for our asynchronous API call, as well as other operations, such as setting timers, logging to the console, or even pausing and canceling the action altogether.

So in essence, Middleware is the go-to place for side effects in the Redux cycle.

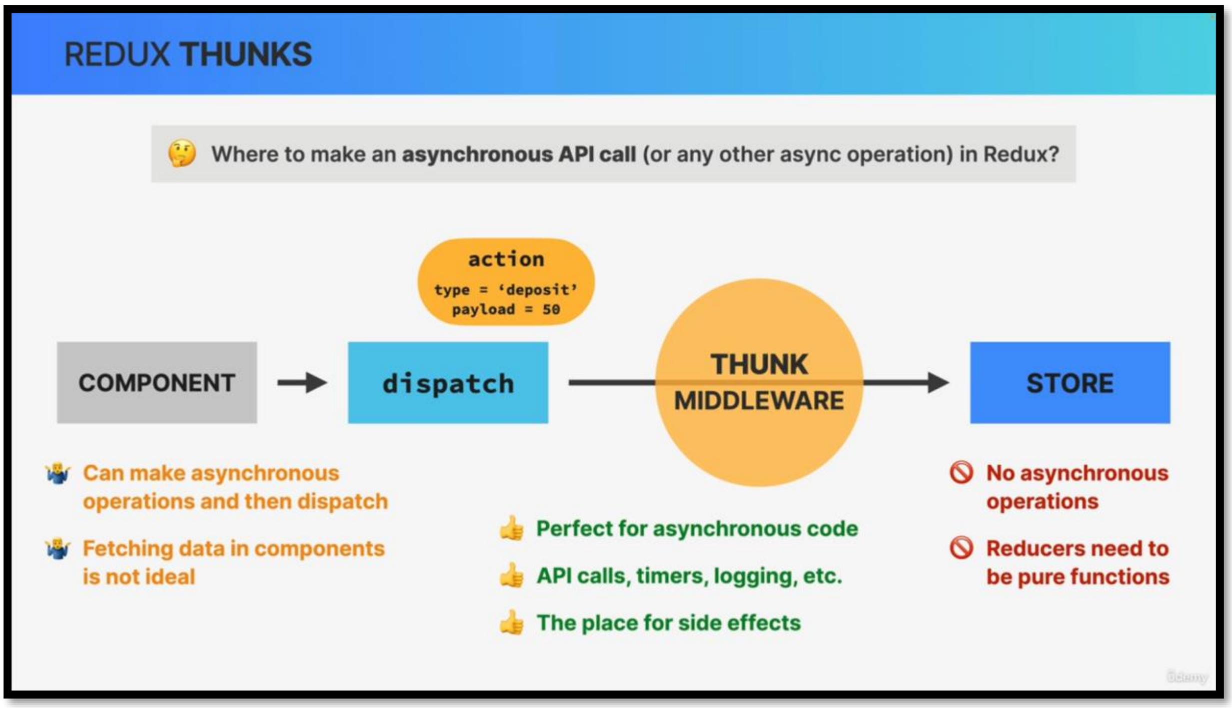

REDUX THUNKS

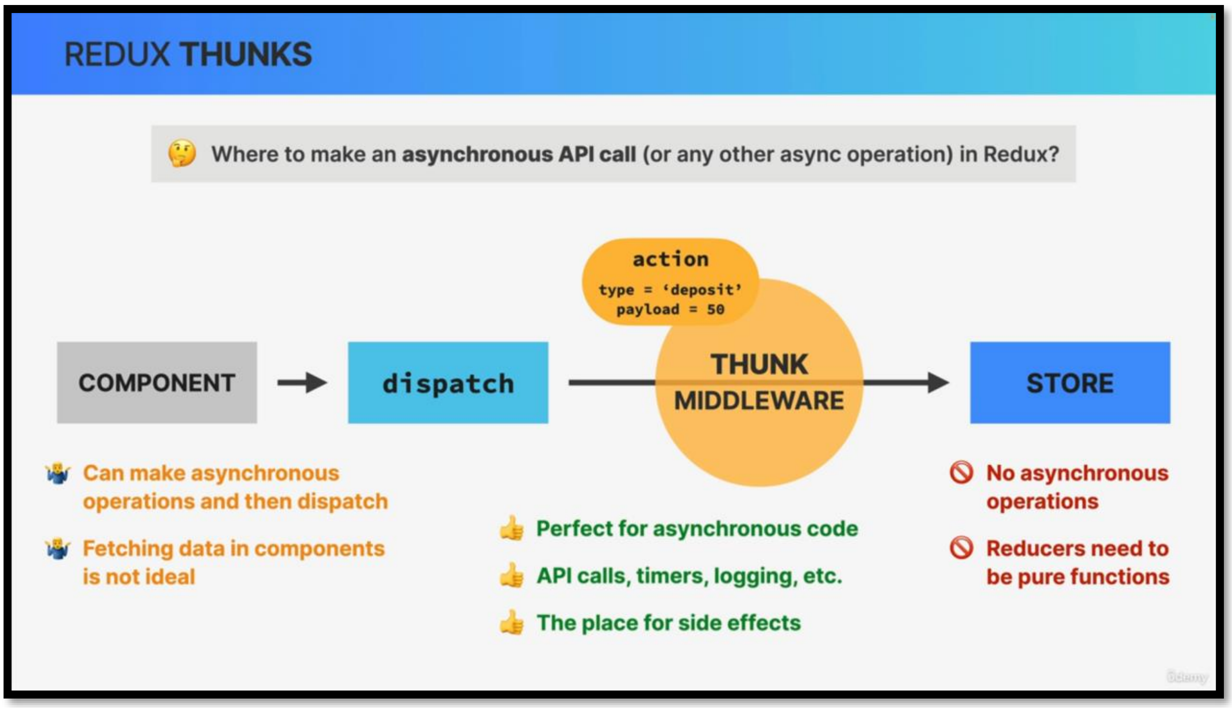
🤔 Where to make an **asynchronous API call** (or any other async operation) in Redux?

COMPONENT
dispatch
THUNK MIDDLEWARE
STORE
Can make asynchronous operations and then dispatch
Fetching data in components is not ideal
Perfect for asynchronous code
API calls, timers, logging, etc.
The place for side effects
No asynchronous operations
Reducers need to be pure functions

So now that we know what Middleware is, where does Middleware actually come from?

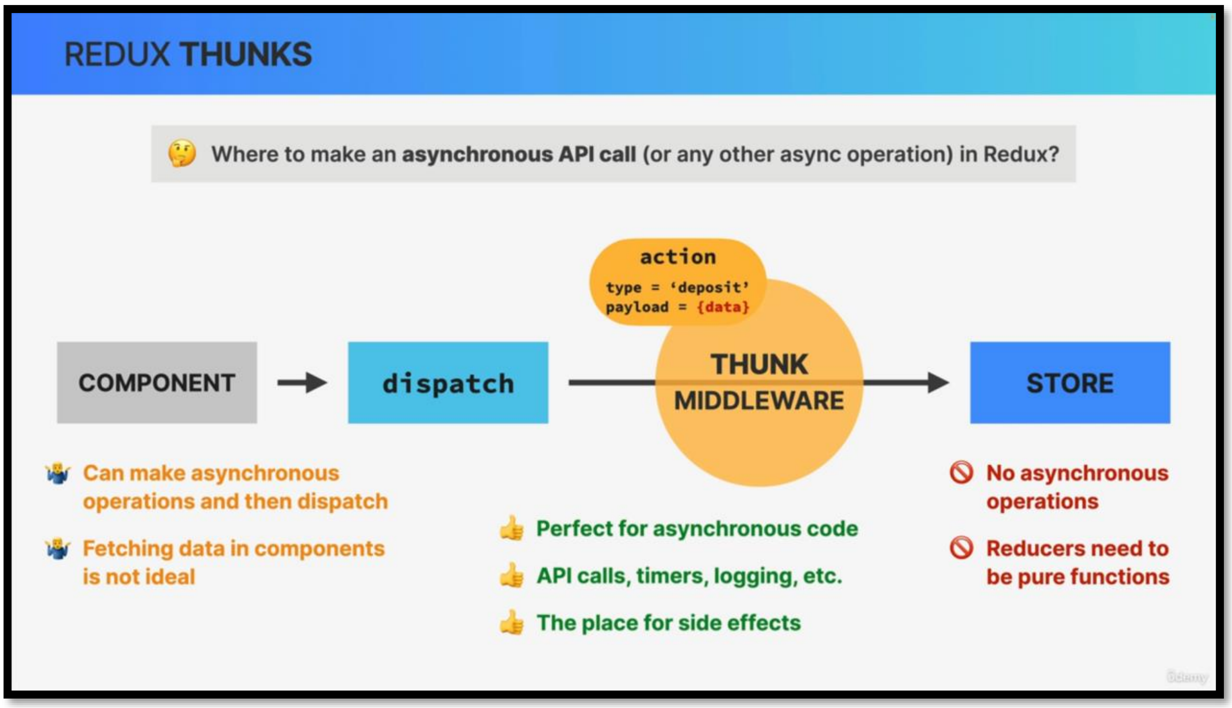
Well, we can write Middleware functions ourselves, but usually, we just use some third-party package and in the case of asynchronous operations, the most popular Middleware in Redux is called Redux Thunk.


REDUX THUNKS
Where to make an asynchronous API call (or any other async operation) in Redux?
action
type = 'deposit'
payload = 50
COMPONENT
dispatch
THUNK MIDDLEWARE
STORE
Can make asynchronous operations and then dispatch
Fetching data in components is not ideal
Perfect for asynchronous code
API calls, timers, logging, etc.
The place for side effects
No asynchronous operations
Reducers need to be pure functions


REDUX THUNKS
Where to make an asynchronous API call (or any other async operation) in Redux?
action
type = 'deposit'
payload = 50
COMPONENT
dispatch
THUNK MIDDLEWARE
STORE
Can make asynchronous operations and then dispatch
Fetching data in components is not ideal
Perfect for asynchronous code
API calls, timers, logging, etc.
The place for side effects
No asynchronous operations
Reducers need to be pure functions

So let's see how Thunks work by analyzing what happens to this action that we have seen before. So now that we have this Thunk Middleware in place, the action will no longer be immediately dispatched, but will first get into the Middleware.

Then we can start fetching some data inside the Thunk,but it could also be some other asynchronous operation but let's stick to data fetching here.



REDUX THUNKS

🤔 Where to make an **asynchronous API call** (or any other async operation) in Redux?

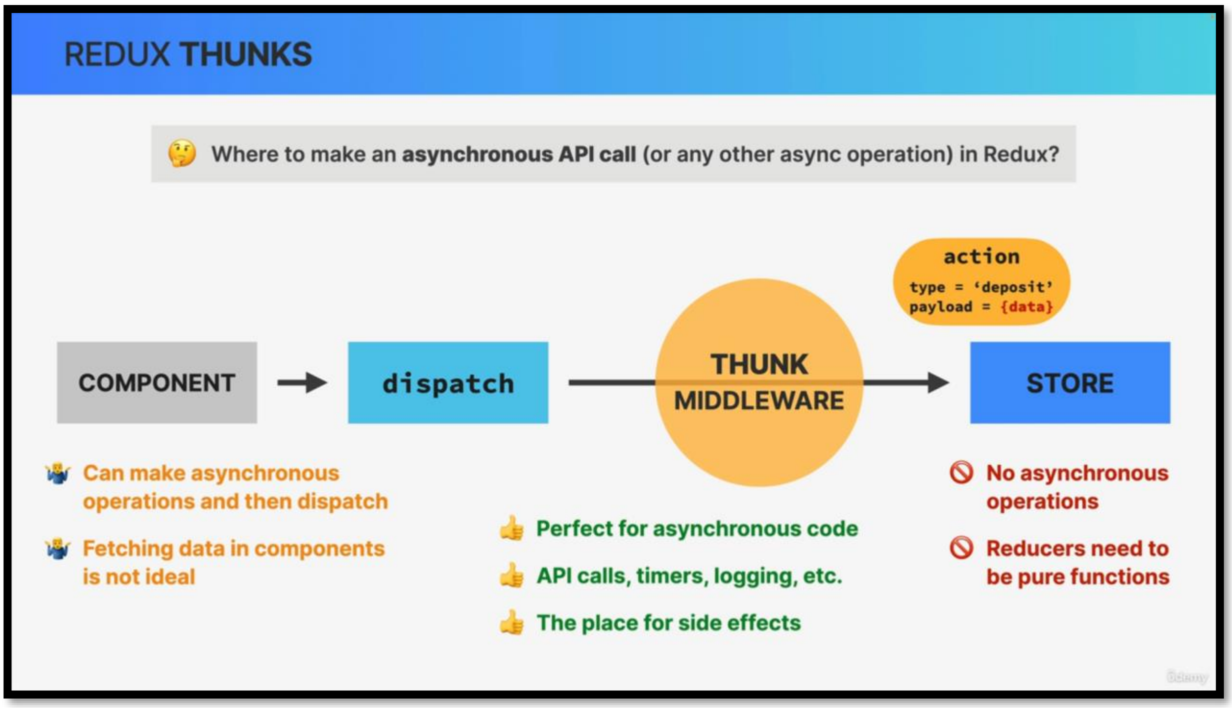
COMPONENT → dispatch → THUNK MIDDLEWARE → STORE

action
type = 'deposit'
payload = {data}

- Can make asynchronous operations and then dispatch
- Fetching data in components is not ideal

- 👍 Perfect for asynchronous code
- 👍 API calls, timers, logging, etc.
- 👍 The place for side effects

- 🚫 No asynchronous operations
- 🚫 Reducers need to be pure functions



REDUX THUNKS

🤔 Where to make an **asynchronous API call** (or any other async operation) in Redux?

COMPONENT → dispatch → THUNK MIDDLEWARE → STORE

action
type = 'deposit'
payload = {data}

- Can make asynchronous operations and then dispatch
- Fetching data in components is not ideal

- 👍 Perfect for asynchronous code
- 👍 API calls, timers, logging, etc.
- 👍 The place for side effects

- 🚫 No asynchronous operations
- 🚫 Reducers need to be pure functions

Now, as soon as the data arrives, we place it into the actions payload and then we finally dispatch the action into the store, where the state will then immediately get updated.

So basically, the Thunk allows Redux to wait before dispatching the fetch data into the store.

In other words, we have used the Thunk in order to defer dispatching into the future. So to the point in which the data that we need has actually arrived.