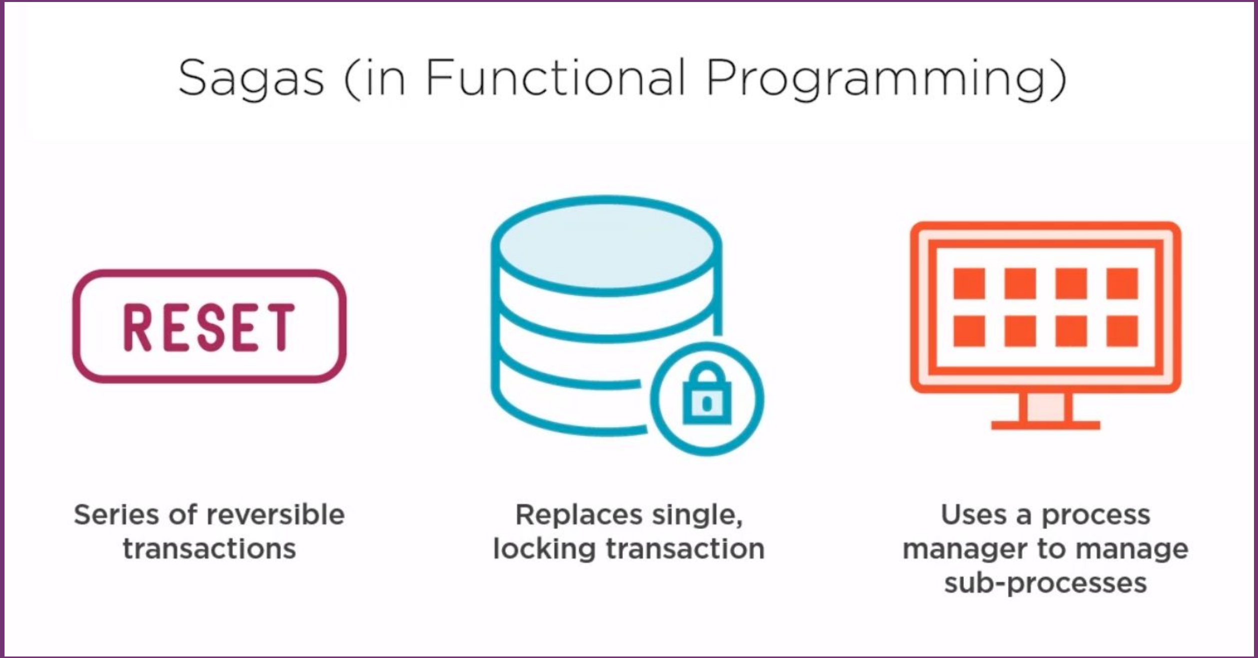
React Native
Redux Middleware (Extra)

Isumi
Batam, Jan 2020

GOALS

1. Intro about Saga & Redux Saga
2. Why Use & Learn Redux Saga?
3. Redux Thunk vs Redux Saga
4. Async ES6 (Yield)
5. Redux Saga Effects

1. Intro about :: Saga ::

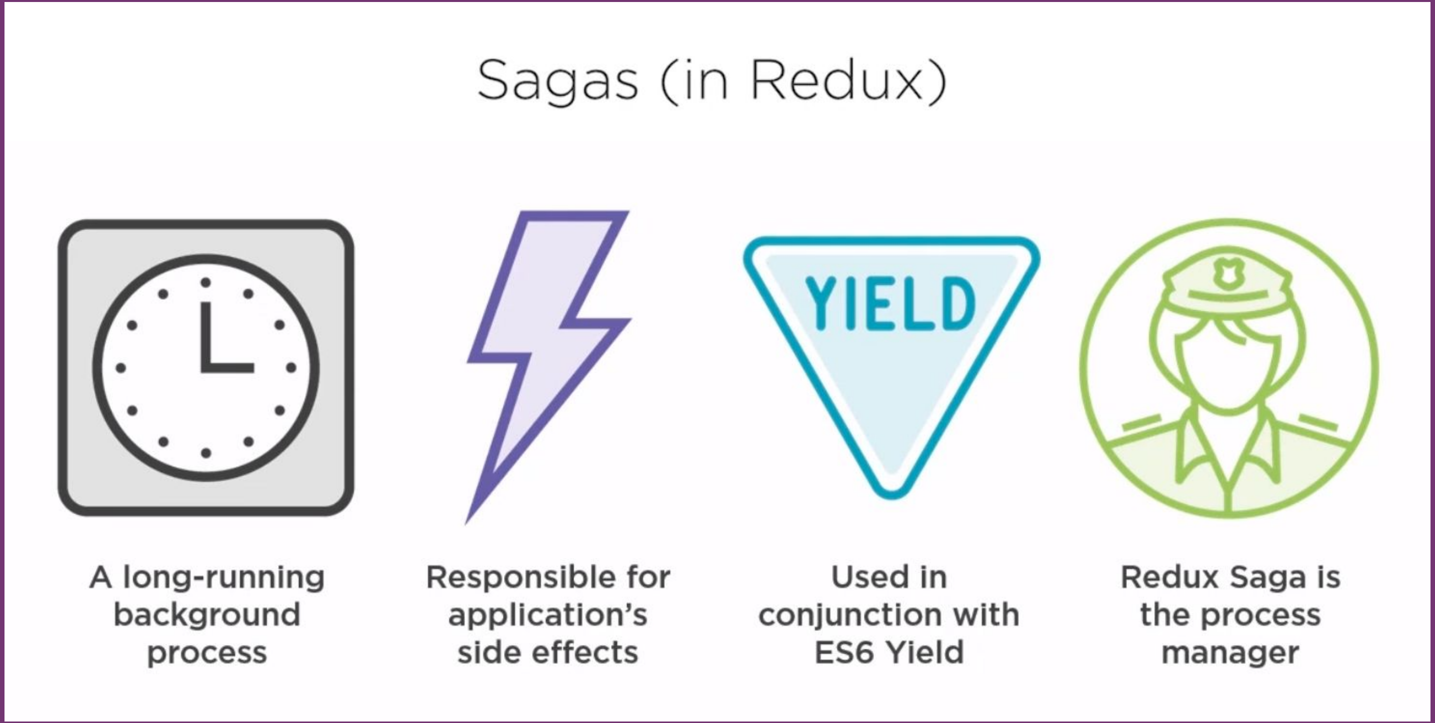


Sagas (in Functional Programming)

Series of reversible transactions

Replaces single, locking transaction

Uses a process manager to manage sub-processes



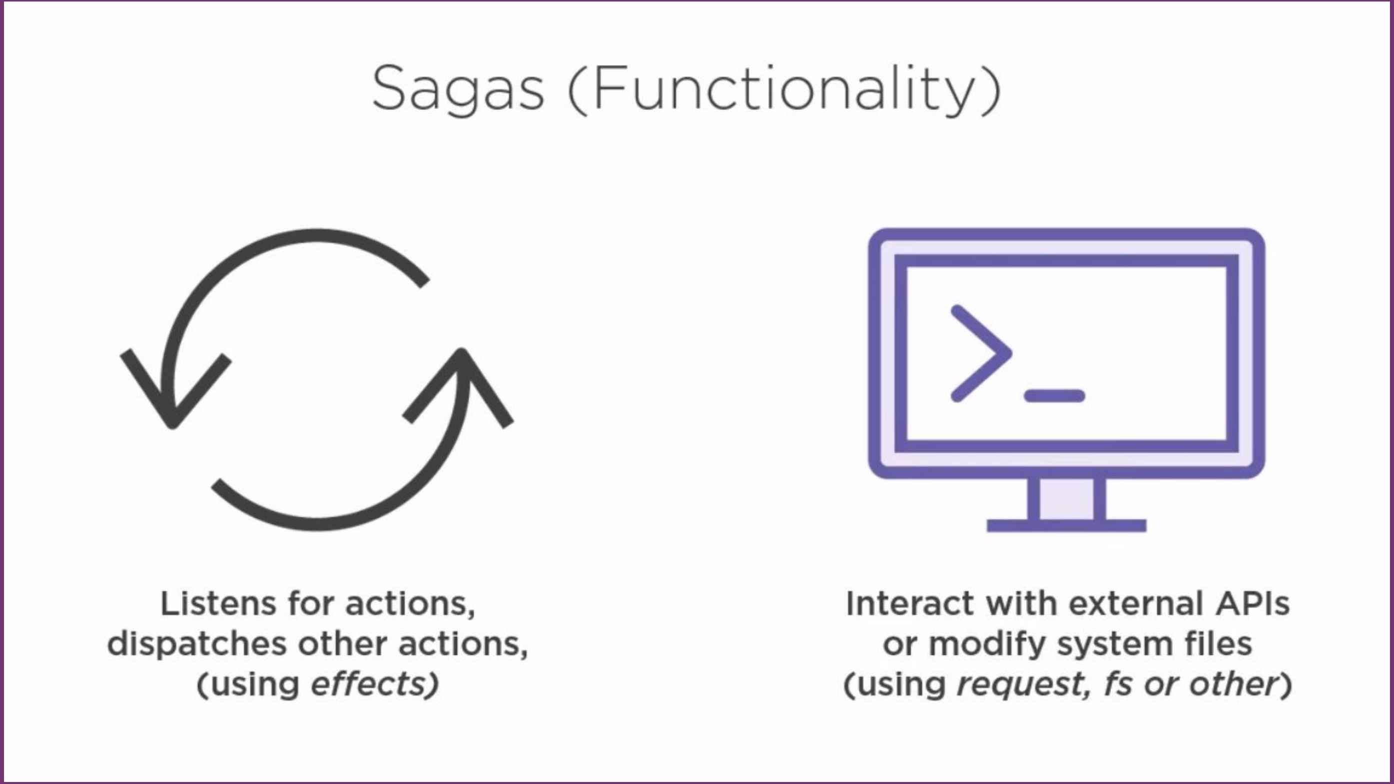
Sagas (in Redux)

A long-running background process

Responsible for application's side effects

Used in conjunction with ES6 Yield

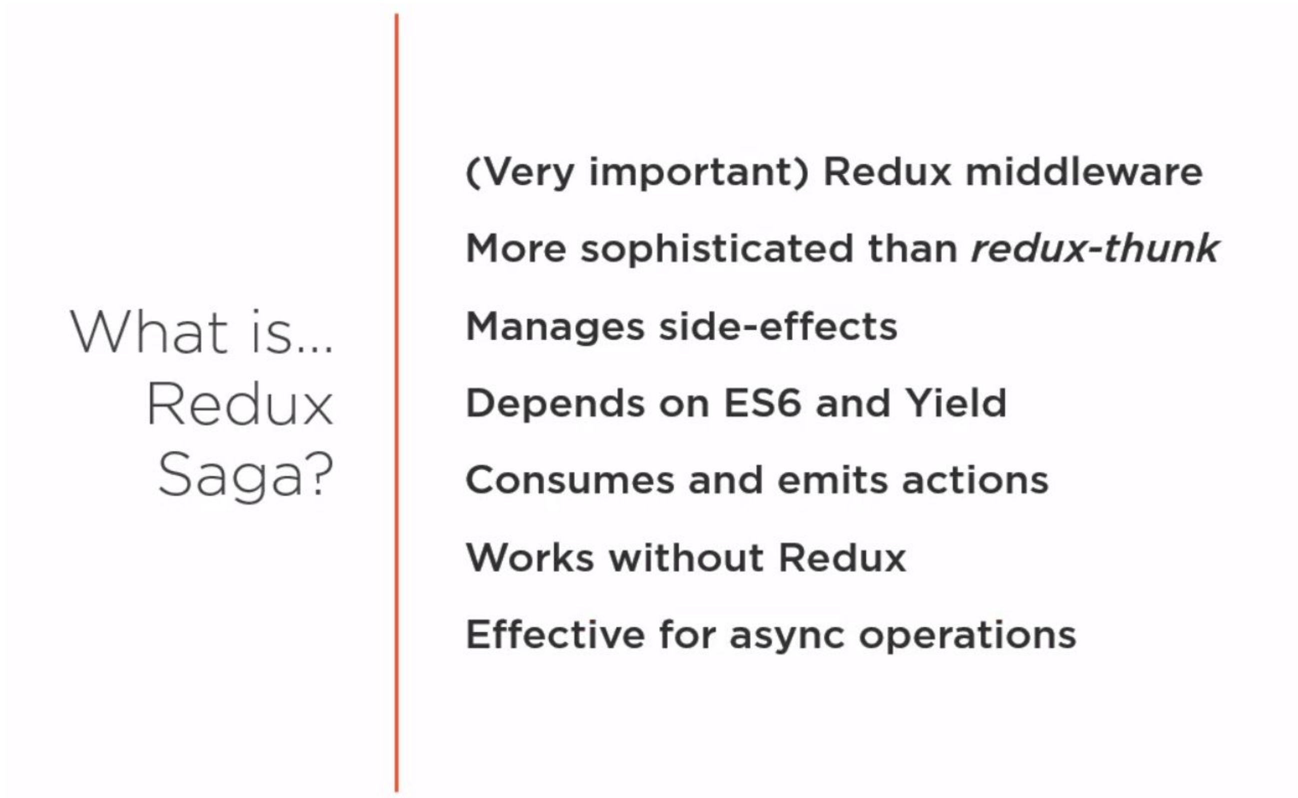
Redux Saga is the process manager



Sagas (Functionality)

Listens for actions,
dispatches other actions,
(using *effects*)

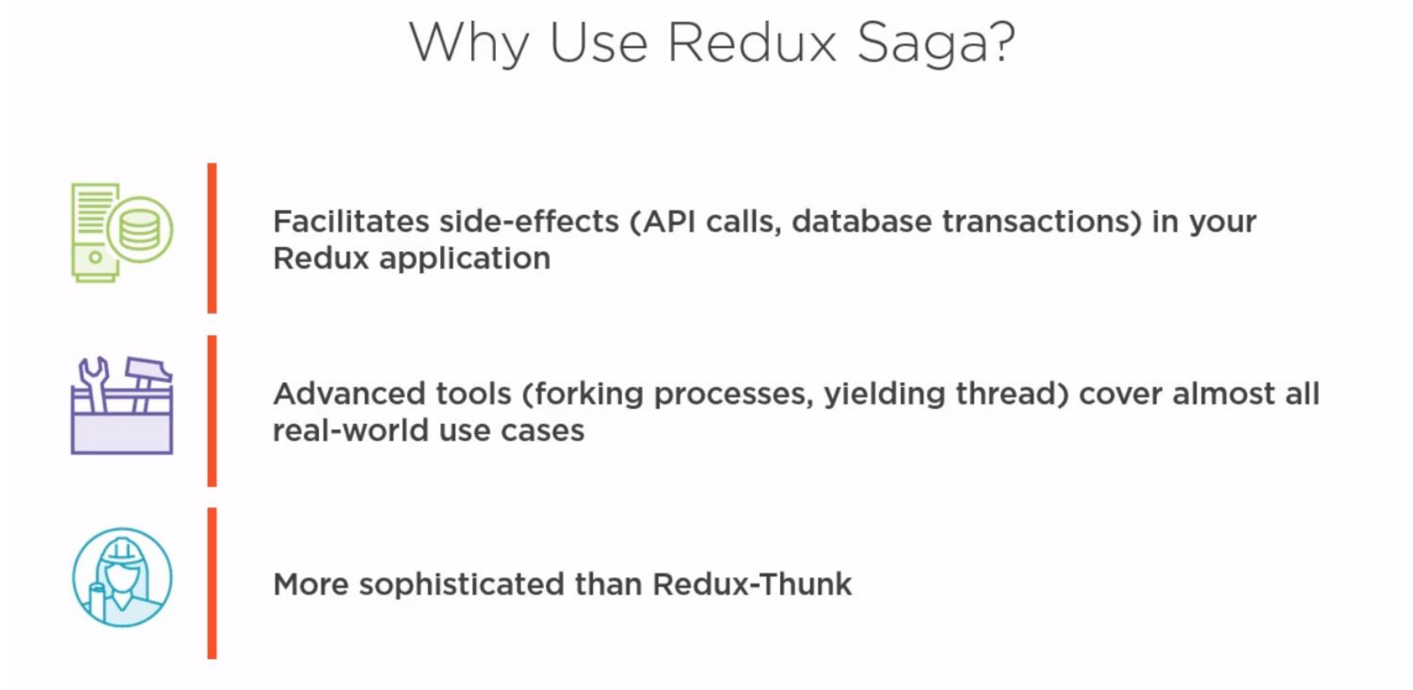
Interact with external APIs
or modify system files
(using *request, fs or other*)



What is... Redux Saga?

- (Very important) Redux middleware
- More sophisticated than *redux-thunk*
- Manages side-effects
- Depends on ES6 and Yield
- Consumes and emits actions
- Works without Redux
- Effective for async operations

2. Why Use Redux Saga?



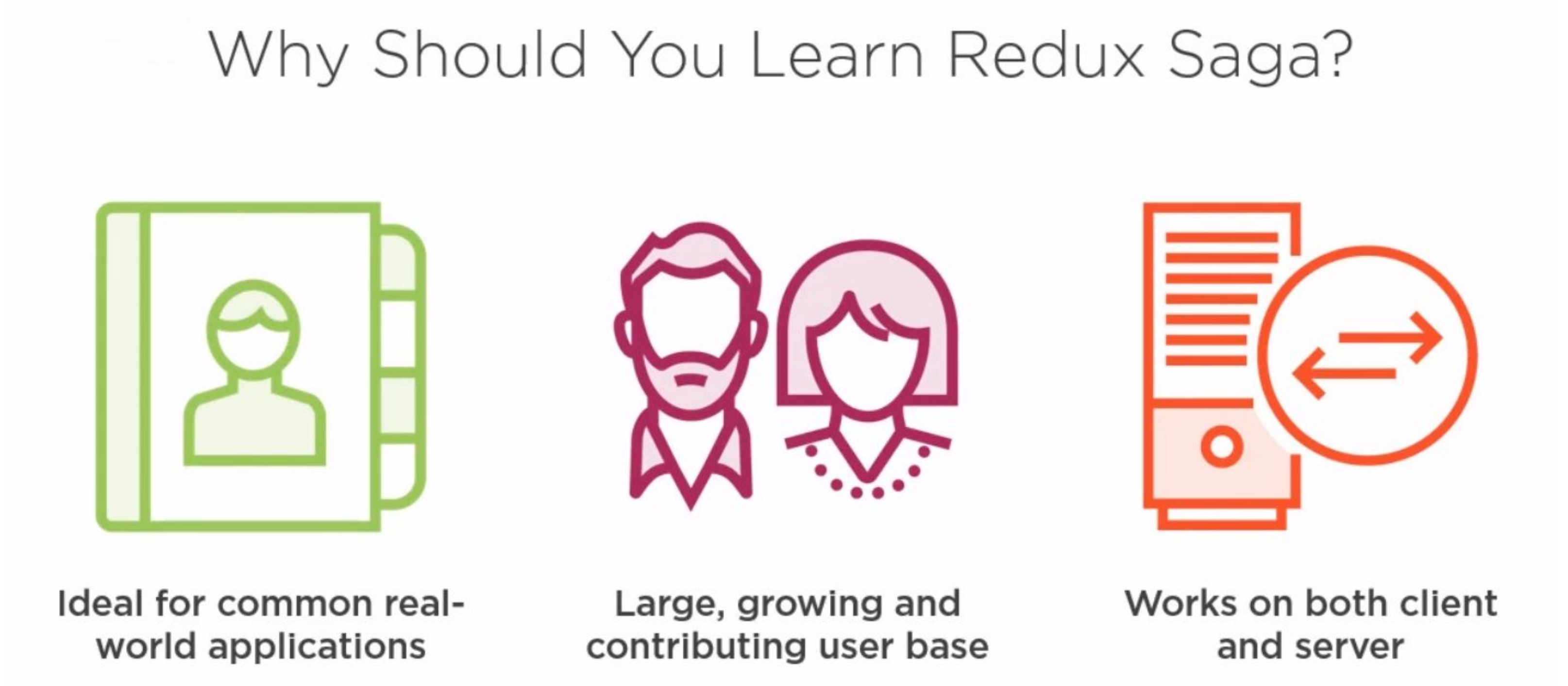
Why Use Redux Saga?

Facilitates side-effects (API calls, database transactions) in your Redux application

Advanced tools (forking processes, yielding thread) cover almost all real-world use cases

More sophisticated than Redux-Thunk

Why Learn Redux Saga?



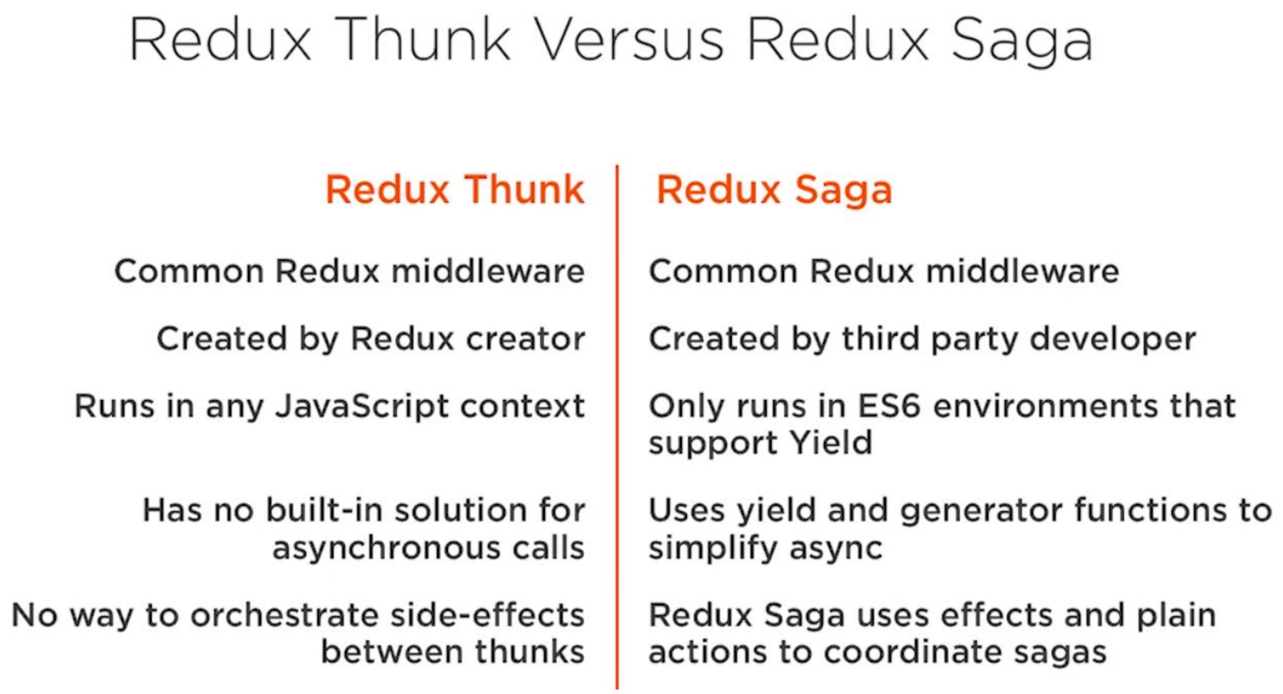
Why Should You Learn Redux Saga?

Ideal for common real-world applications

Large, growing and contributing user base

Works on both client and server

3. Redux Thunk vs Redux Saga



Redux Thunk Versus Redux Saga

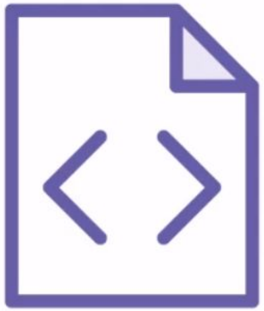
Redux Thunk	Redux Saga
Common Redux middleware	Common Redux middleware
Created by Redux creator	Created by third party developer
Runs in any JavaScript context	Only runs in ES6 environments that support Yield
Has no built-in solution for asynchronous calls	Uses yield and generator functions to simplify async
No way to orchestrate side-effects between thunks	Redux Saga uses effects and plain actions to coordinate sagas

4. Async ES 6 (Yield)

What Is Yield?

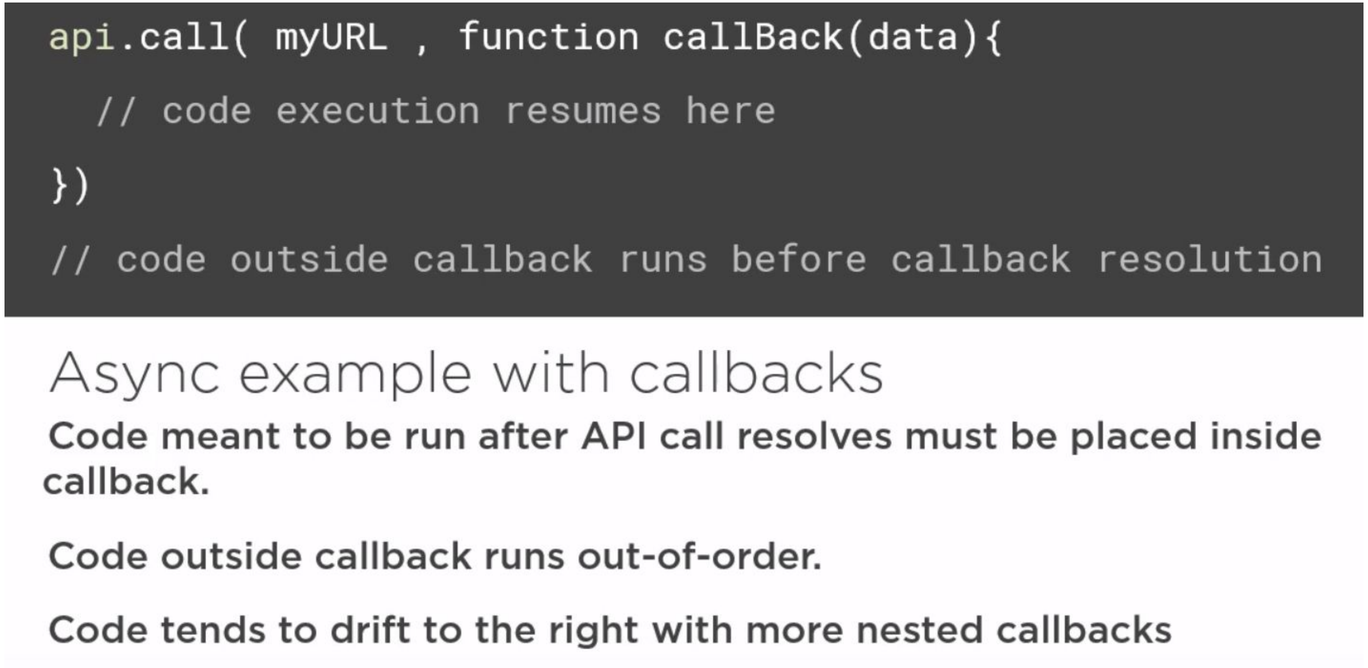
Special keyword that can delay the execution of subsequent code

Only works inside generator functions



Works with promises and condenses code surrounding them

Before Yield: Callback



```
api.call( myURL , function callBack(data){
  // code execution resumes here
})
// code outside callback runs before callback resolution
```

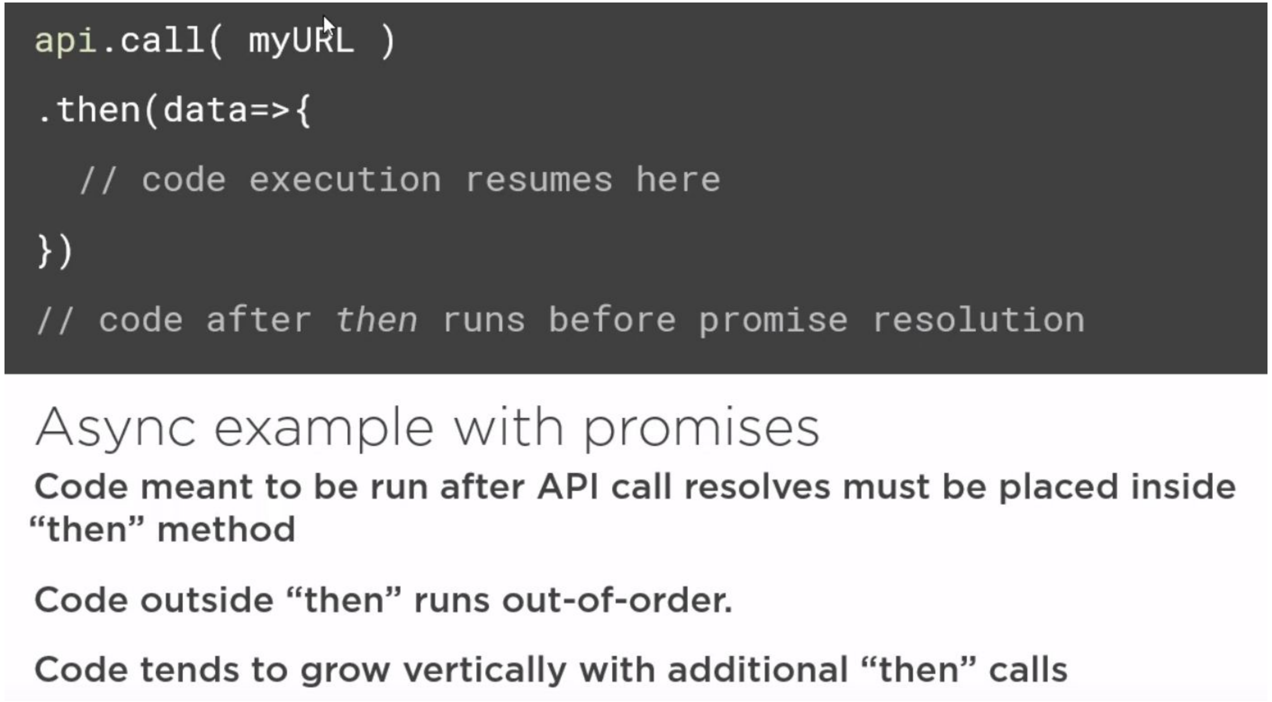
Async example with callbacks

Code meant to be run after API call resolves must be placed inside callback.

Code outside callback runs out-of-order.

Code tends to drift to the right with more nested callbacks

Before Yield: Promise



```
api.call( myURL )
.then(data=>{
  // code execution resumes here
})
// code after then runs before promise resolution
```

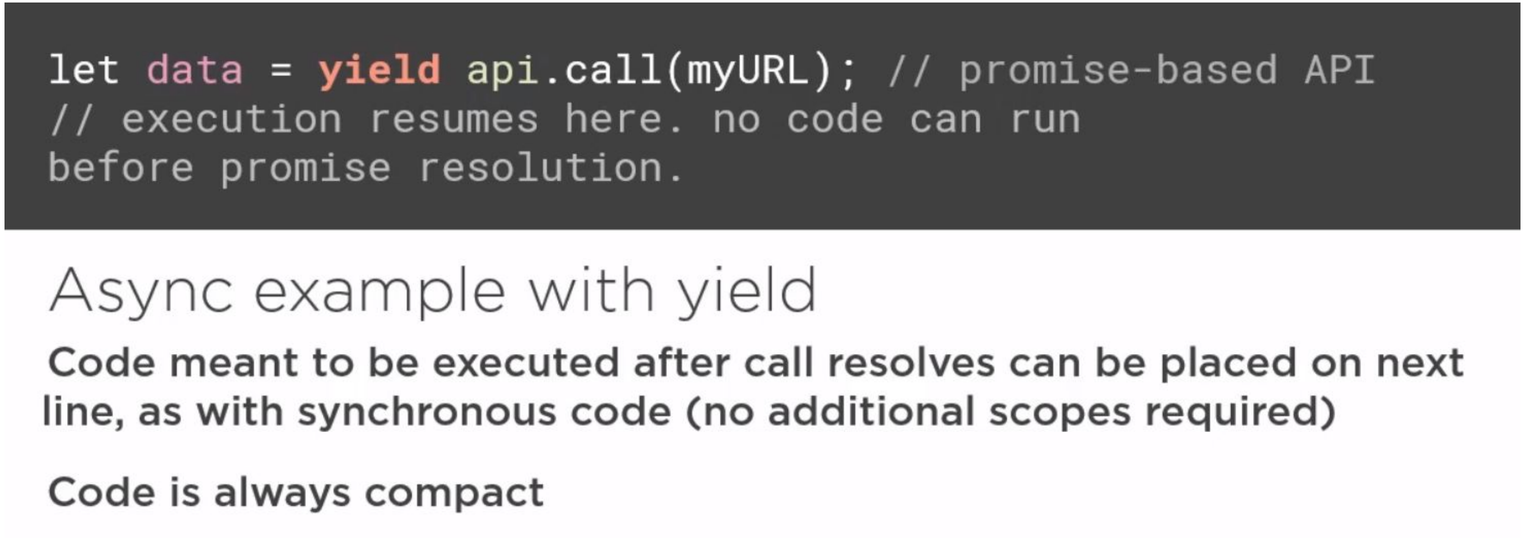
Async example with promises

Code meant to be run after API call resolves must be placed inside “then” method

Code outside “then” runs out-of-order.

Code tends to grow vertically with additional “then” calls

Using Yield



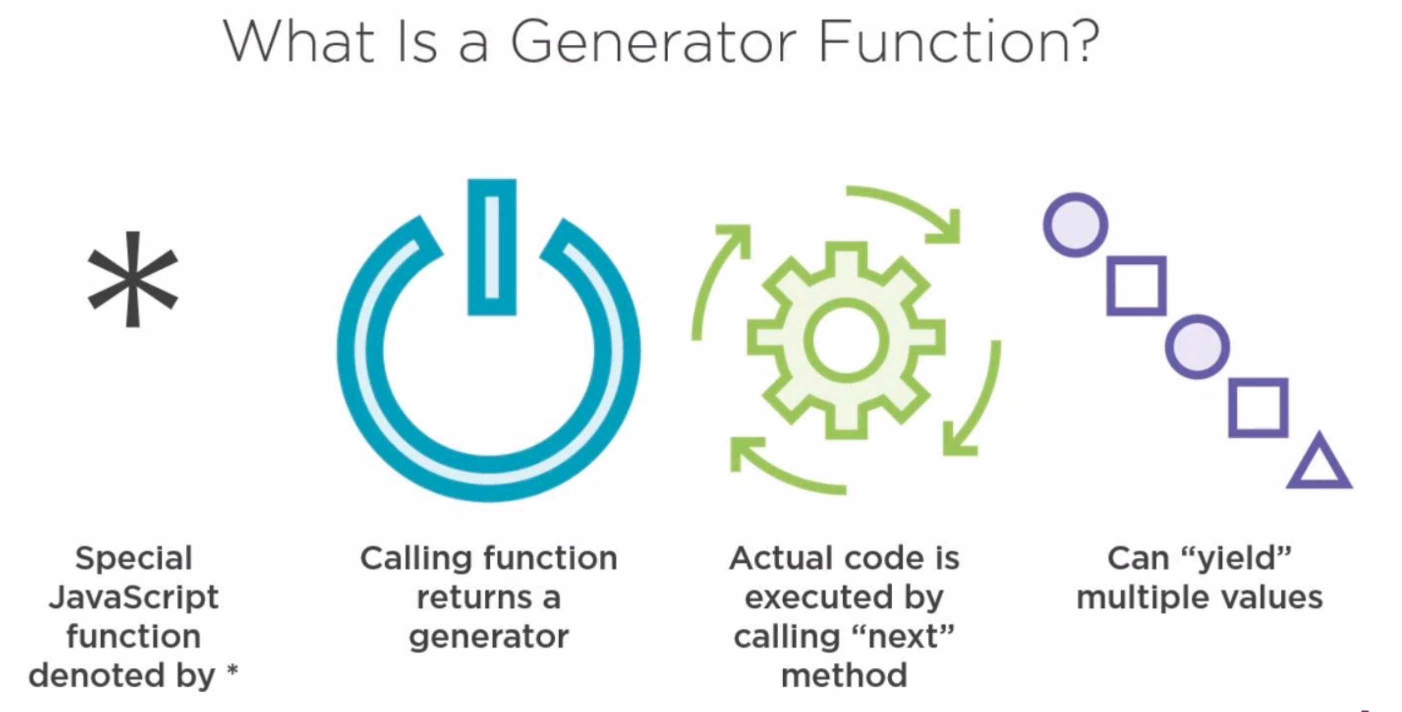
```
let data = yield api.call(myURL); // promise-based API
// execution resumes here. no code can run
before promise resolution.
```

Async example with yield

Code meant to be executed after call resolves can be placed on next line, as with synchronous code (no additional scopes required)

Code is always compact

Generator Function



What Is a Generator Function?

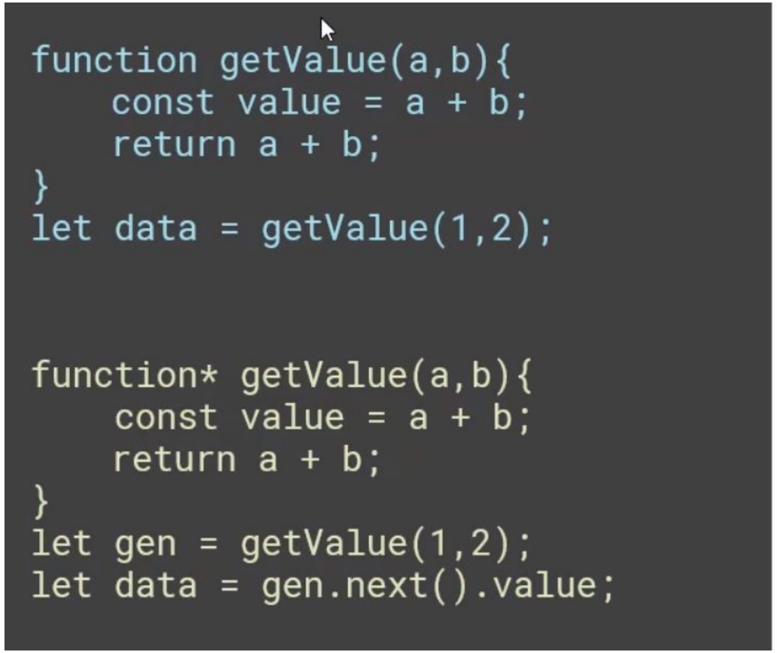
Special JavaScript function denoted by *

Calling function returns a generator

Actual code is executed by calling "next" method

Can "yield" multiple values

Normal Vs Generator Function



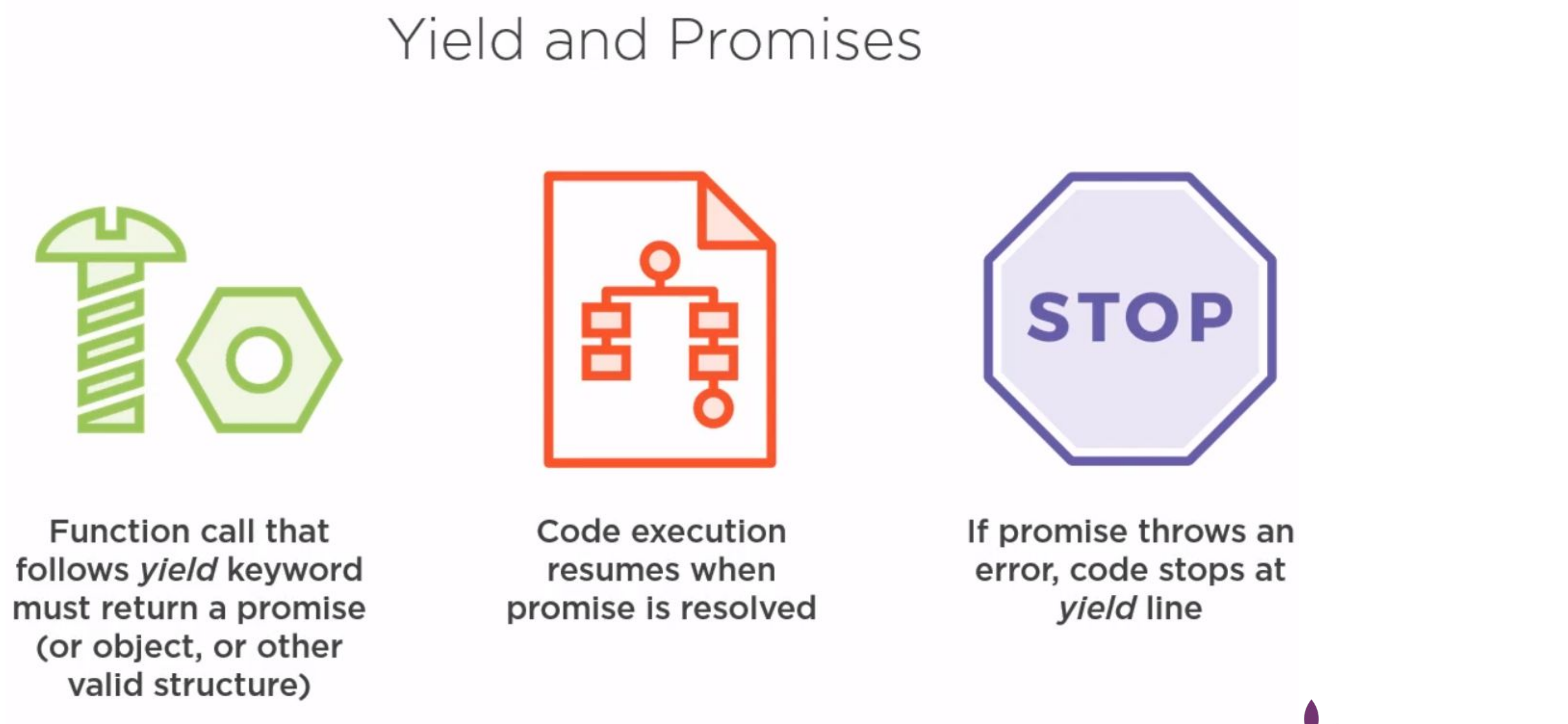
```
function getValue(a,b){
    const value = a + b;
    return a + b;
}
let data = getValue(1,2);


function* getValue(a,b){
    const value = a + b;
    return a + b;
}
let gen = getValue(1,2);
let data = gen.next().value;
```

- With normal function, invocation of function returns final value
- With generator function, invocation returns a generator
- “Next must be called to get final value”

BINAR ACADEMY

Yield and Promise



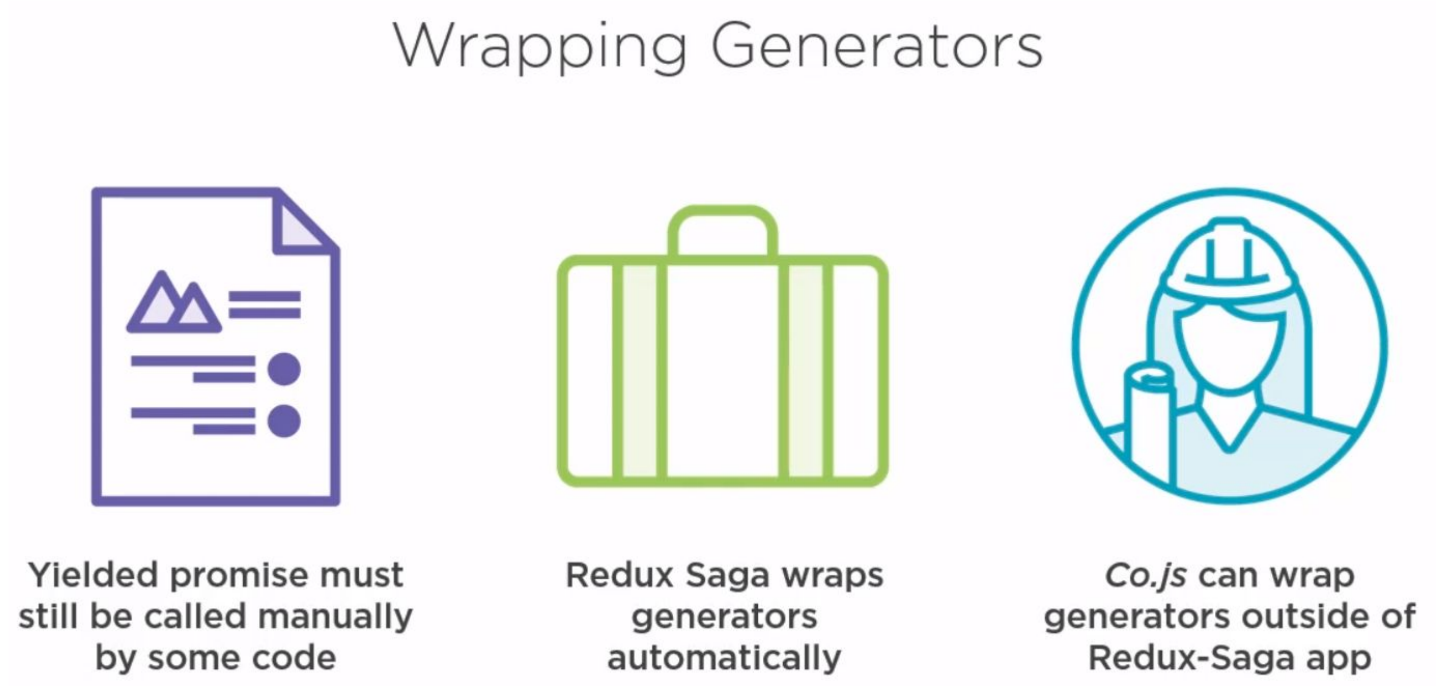
Yield and Promises

Function call that follows *yield* keyword must return a promise (or object, or other valid structure)

Code execution resumes when promise is resolved

If promise throws an error, code stops at *yield* line

Wrapping Generators



Wrapping Generators

Yielded promise must still be called manually by some code

Redux Saga wraps generators automatically

Co.js can wrap generators outside of Redux-Saga app

Redux Saga Wrapped Generators

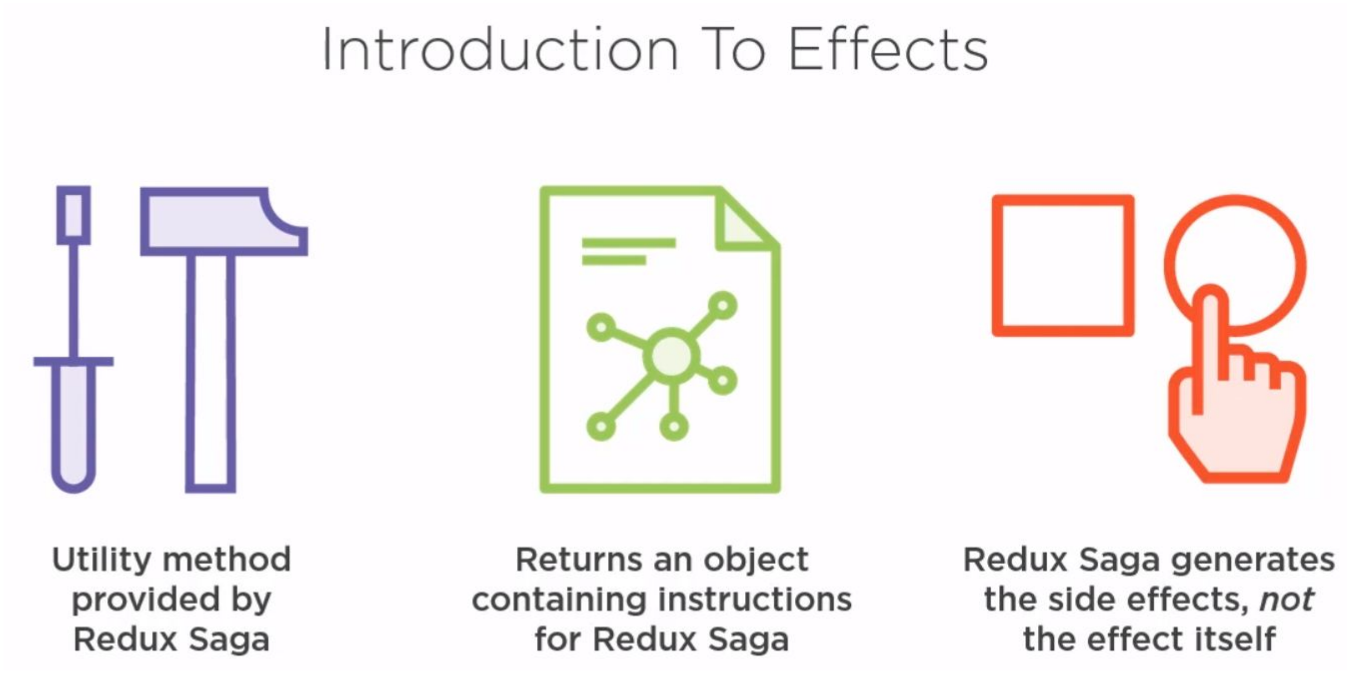
```
function* getData(){
  let data =
    yield api.call('/cart');
  return data + 5;
}


let gen = getData();
let promise = gen.next();
promise.then(data=>{
  let value = gen.next(data);
})

function* mySaga() {
  yield delay(500);
  yield delay(700);
  console.log("Saga complete");
};
```

- Generator function needs value returned from API call to proceed
- Wrapper code still needs *.then* somewhere to capture the response from API and pass it to generator
- Sagas are wrapped by redux-saga, *.then()* is never manually called

5. Redux Saga Effects



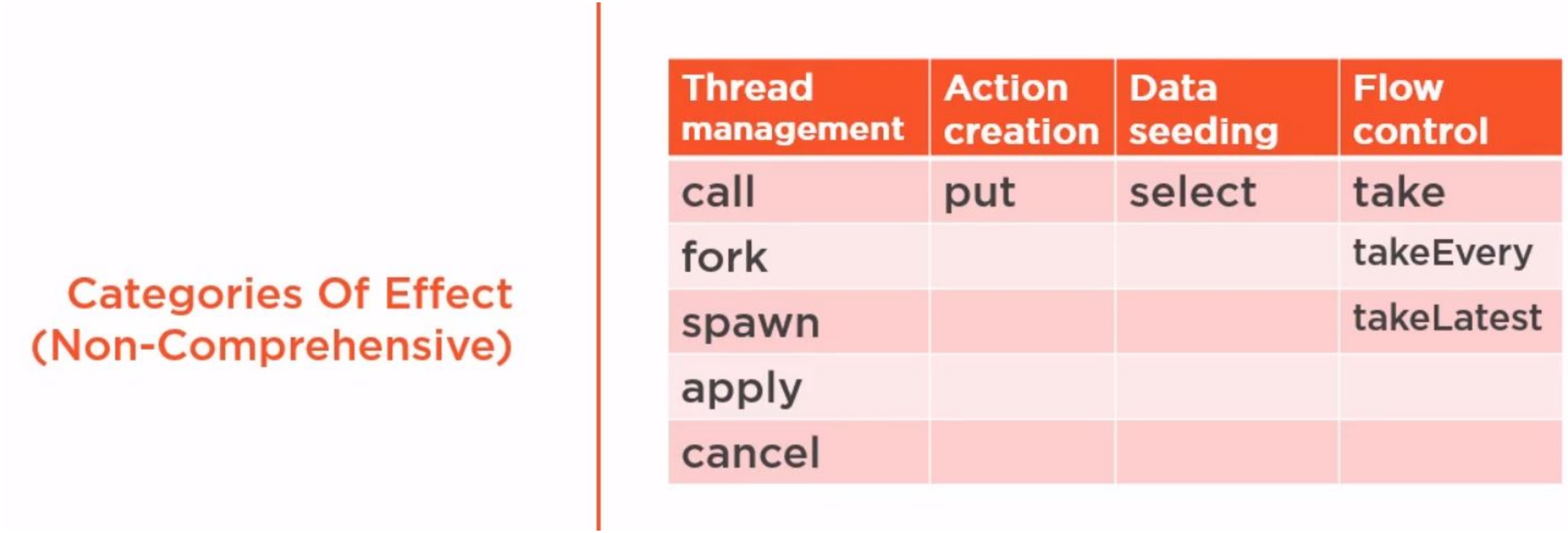
Introduction To Effects

Utility method provided by Redux Saga

Returns an object containing instructions for Redux Saga

Redux Saga generates the side effects, *not* the effect itself

Categories of Effects

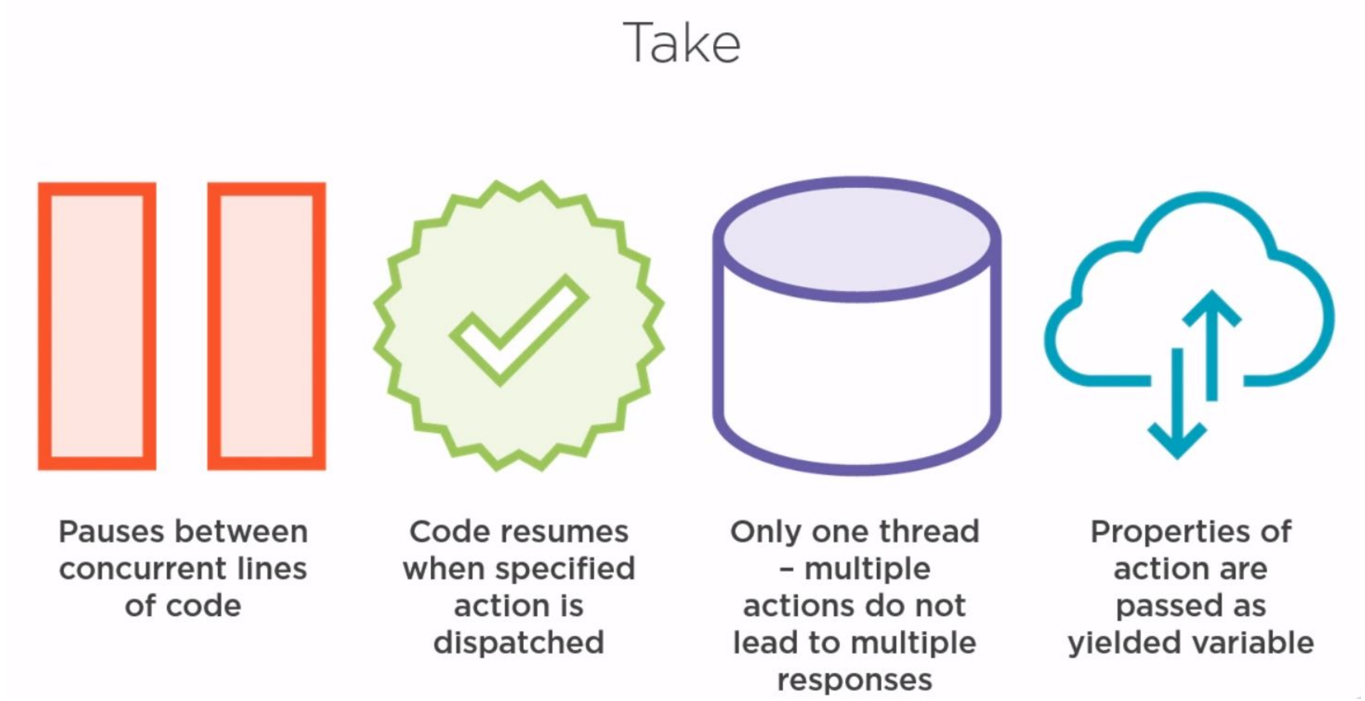

Categories Of Effect (Non-Comprehensive)

Thread management	Action creation	Data seeding	Flow control
call	put	select	take
fork			takeEvery
spawn			takeLatest
apply			
cancel			

Demo Redux Saga Sandbox:
https://github.com/danielstern/redux-saga-sandbox

Redux Saga Cart App:
https://github.com/danielstern/redux-saga-cart
Redux Saga Shopping Cart Server:
https://github.com/danielstern/redux-saga-shopping-cart-server

1) Take



Take

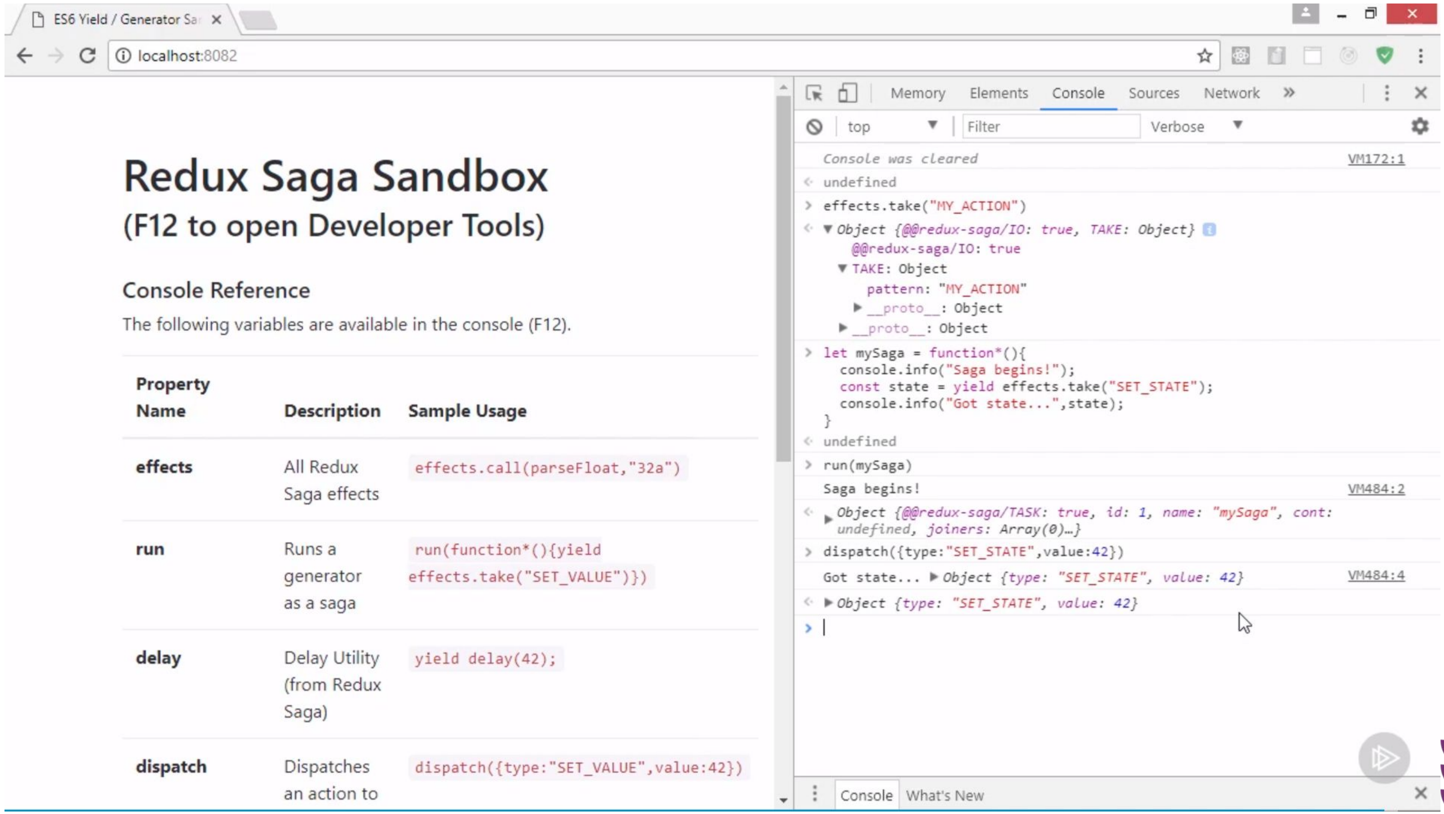
Pauses between concurrent lines of code

Code resumes when specified action is dispatched

Only one thread – multiple actions do not lead to multiple responses

Properties of action are passed as yielded variable

Implementing Take



Redux Saga Sandbox
(F12 to open Developer Tools)

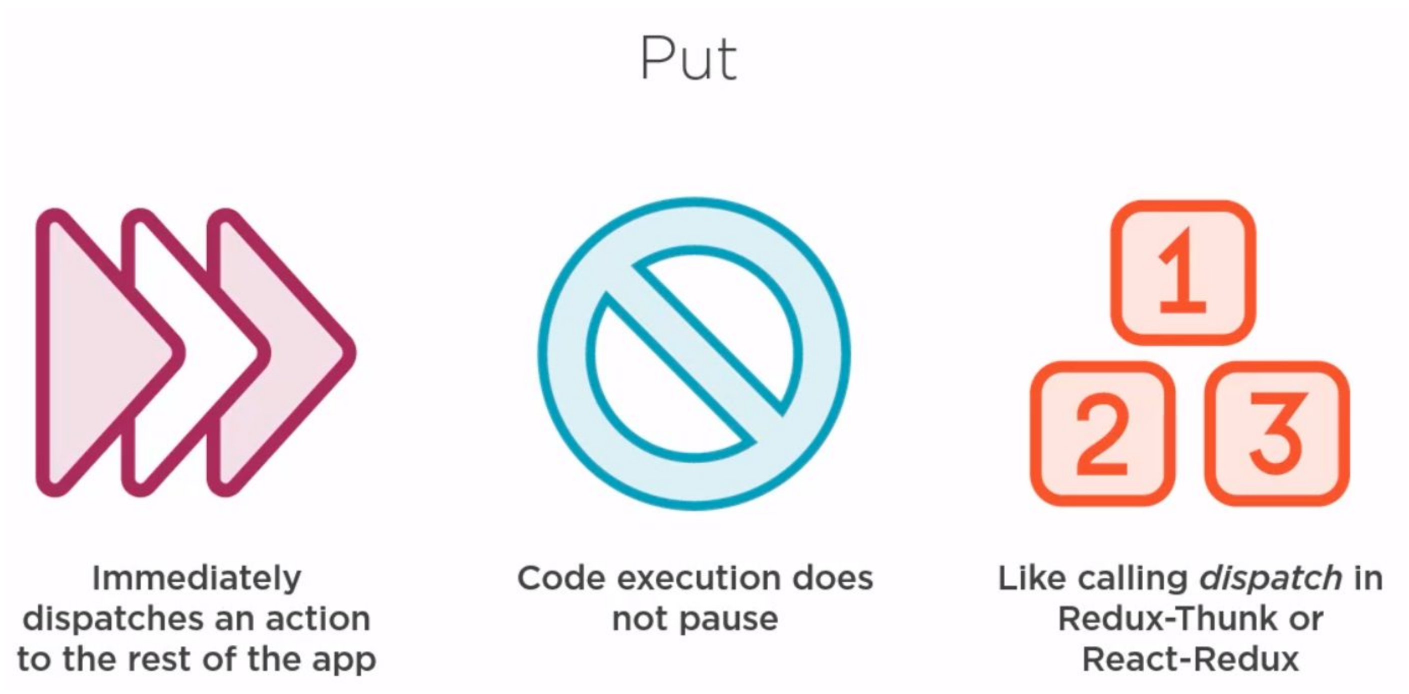
Console Reference

The following variables are available in the console (F12).

Property Name	Description	Sample Usage
effects	All Redux Saga effects	`effects.call(parseFloat,"32a")`
run	Runs a generator as a saga	`run(function*(){yield effects.take("SET_VALUE")})`
delay	Delay Utility (from Redux Saga)	`yield delay(42);`
dispatch	Dispatches an action to	`dispatch({type:"SET_VALUE",value:42})`

```
Console was cleared                                              VM172:1
< undefined
> effects.take("MY_ACTION")
< ▼Object {@@redux-saga/IO: true, TAKE: Object}
     @@redux-saga/IO: true
   ▼ TAKE: Object
       pattern: "MY_ACTION"
     ▶ __proto__: Object
   ▶ __proto__: Object
> let mySaga = function*(){
    console.info("Saga begins!");
    const state = yield effects.take("SET_STATE");
    console.info("Got state...",state);
  }
< undefined
> run(mySaga)
  Saga begins!                                                   VM484:2
< ▶Object {@@redux-saga/TASK: true, id: 1, name: "mySaga", cont:
  undefined, joiners: Array(0)…}
> dispatch({type:"SET_STATE",value:42})
  Got state... ▶Object {type: "SET_STATE", value: 42}            VM484:4
< ▶Object {type: "SET_STATE", value: 42}
>
```

2) Put



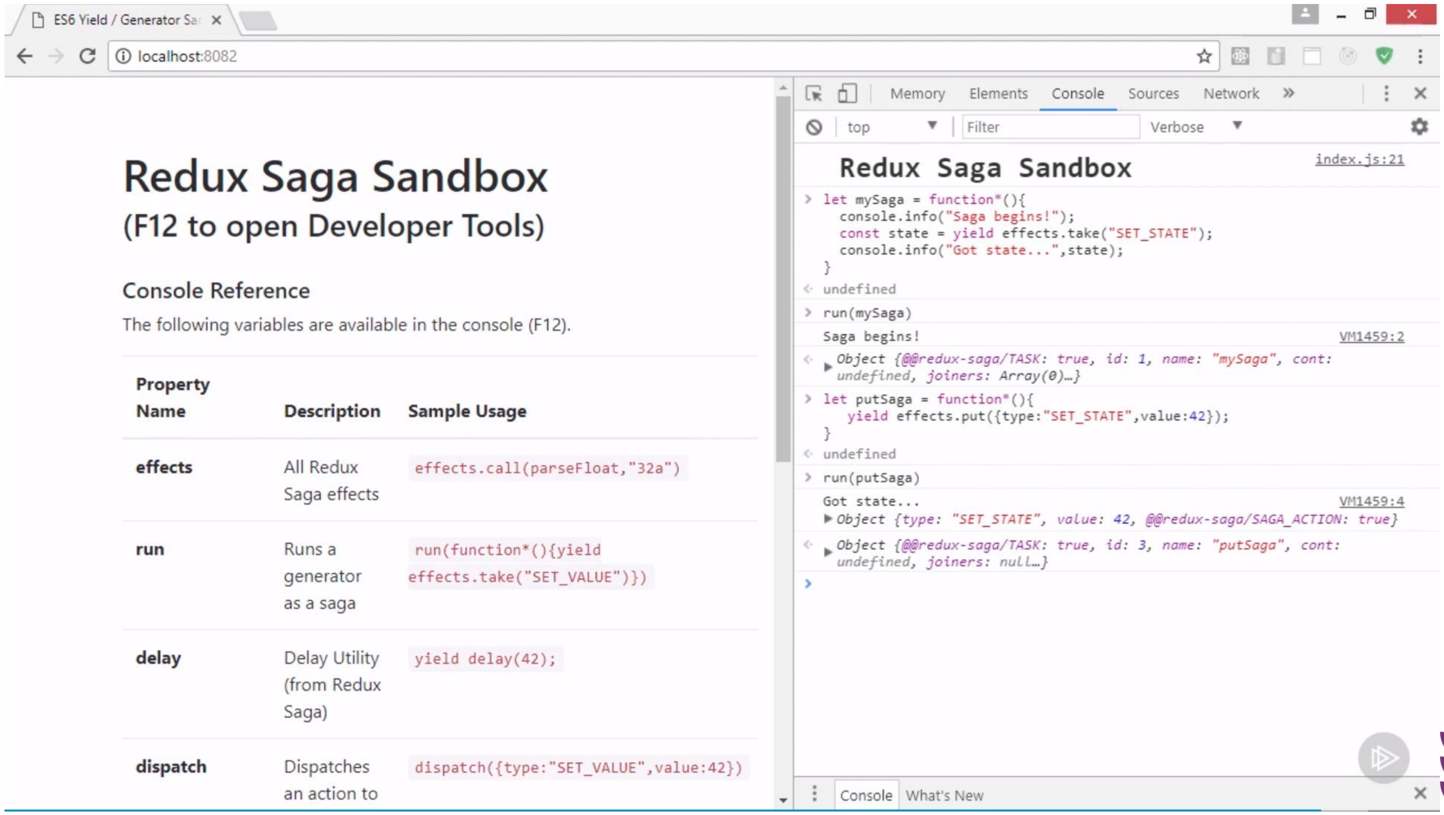
Put

Immediately dispatches an action to the rest of the app

Code execution does not pause

Like calling *dispatch* in Redux-Thunk or React-Redux

Implementing Put



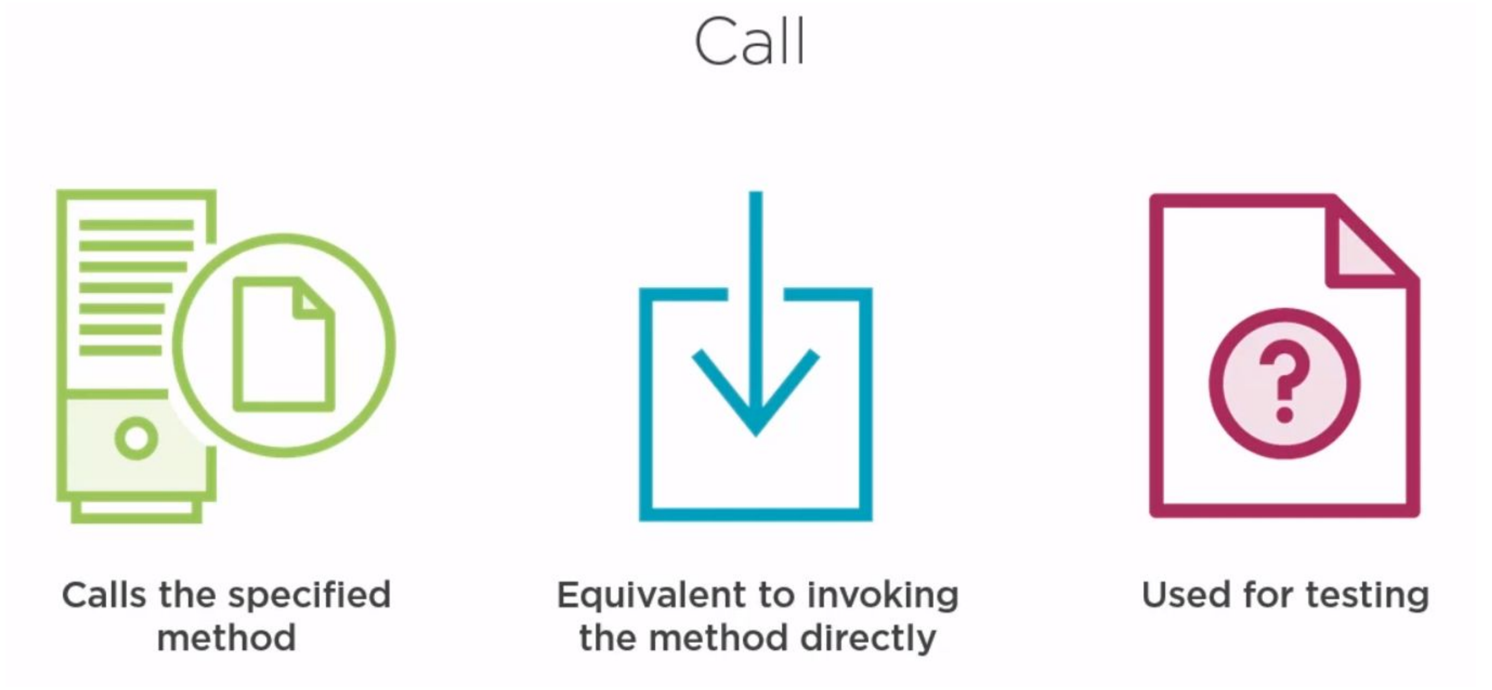
Redux Saga Sandbox
(F12 to open Developer Tools)

Console Reference
The following variables are available in the console (F12).

Property Name	Description	Sample Usage
effects	All Redux Saga effects	`effects.call(parseFloat,"32a")`
run	Runs a generator as a saga	`run(function*(){yield effects.take("SET_VALUE")})`
delay	Delay Utility (from Redux Saga)	`yield delay(42);`
dispatch	Dispatches an action to	`dispatch({type:"SET_VALUE",value:42})`

```
   Redux Saga Sandbox                                              index.js:21
> let mySaga = function*(){
    console.info("Saga begins!");
    const state = yield effects.take("SET_STATE");
    console.info("Got state...",state);
  }
< undefined
> run(mySaga)
  Saga begins!                                                     VM1459:2
< ▸Object {@@redux-saga/TASK: true, id: 1, name: "mySaga", cont:
   undefined, joiners: Array(0)…}
> let putSaga = function*(){
     yield effects.put({type:"SET_STATE",value:42});
  }
< undefined
> run(putSaga)
  Got state...                                                     VM1459:4
  ▸Object {type: "SET_STATE", value: 42, @@redux-saga/SAGA_ACTION: true}
< ▸Object {@@redux-saga/TASK: true, id: 3, name: "putSaga", cont:
   undefined, joiners: null…}
>
```

3) Call



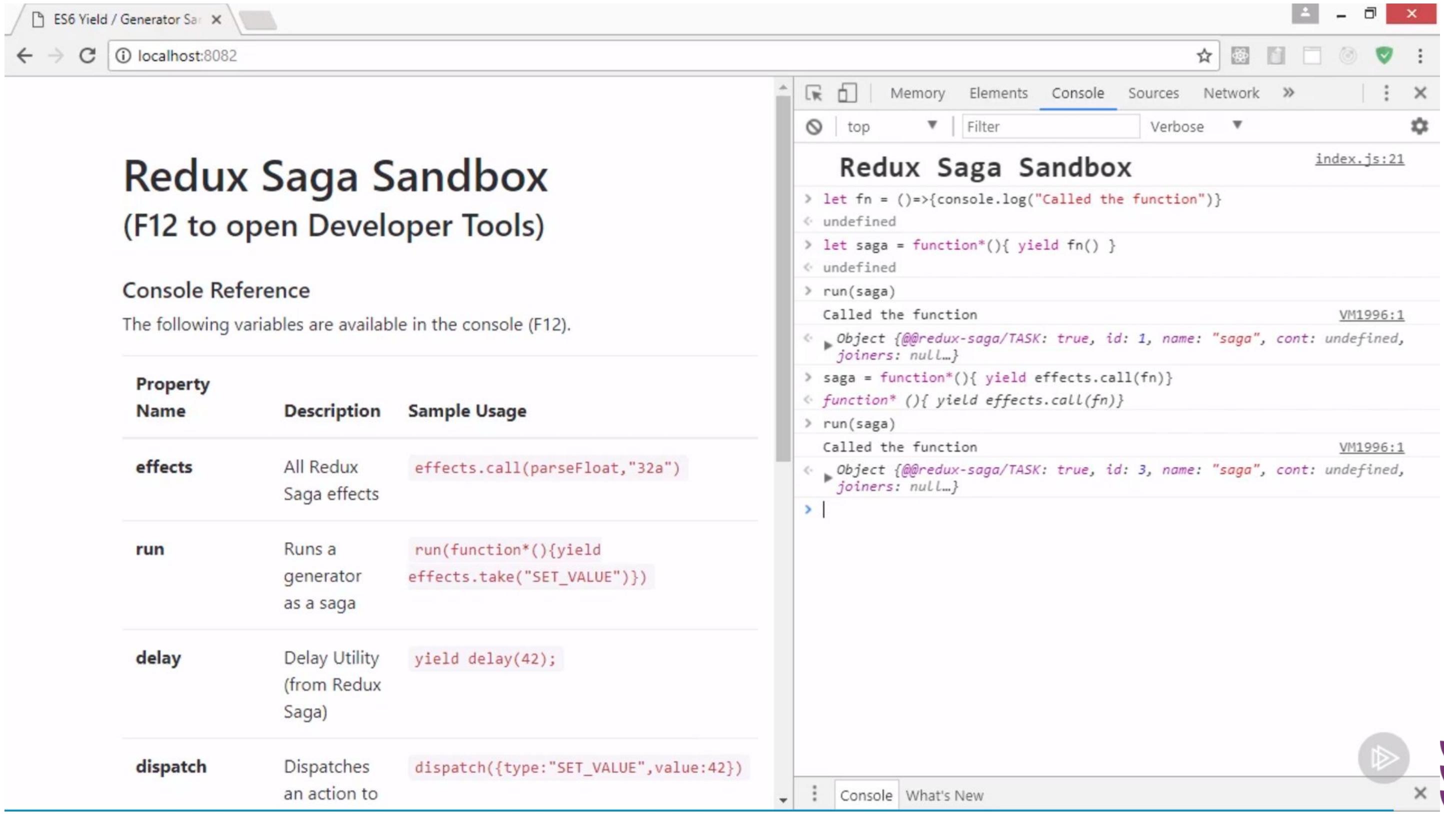
Call

Calls the specified method

Equivalent to invoking the method directly

Used for testing

Implementing Call



Redux Saga Sandbox
(F12 to open Developer Tools)

Console Reference

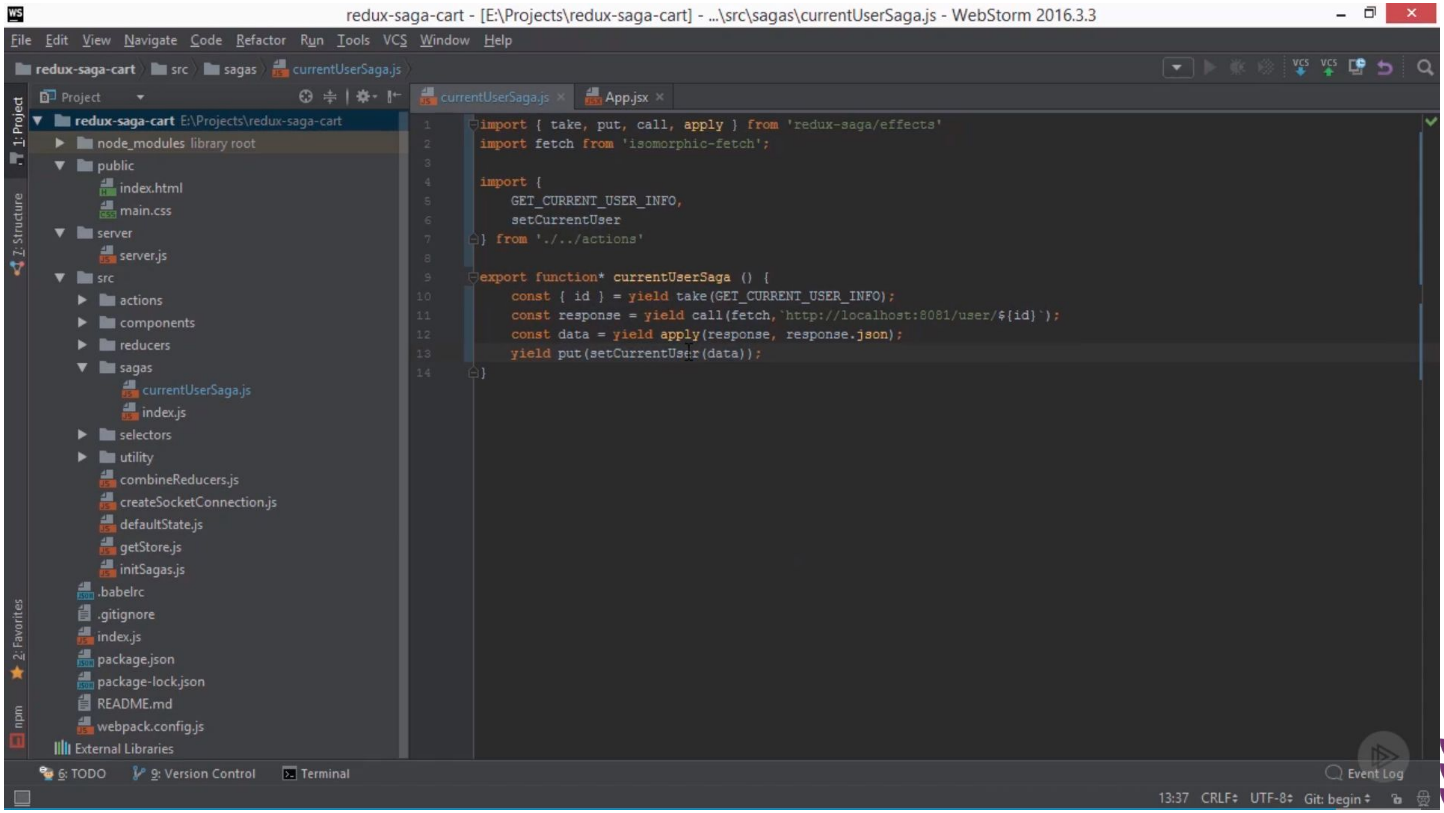
The following variables are available in the console (F12).

Property Name	Description	Sample Usage
effects	All Redux Saga effects	`effects.call(parseFloat,"32a")`
run	Runs a generator as a saga	`run(function*(){yield effects.take("SET_VALUE")})`
delay	Delay Utility (from Redux Saga)	`yield delay(42);`
dispatch	Dispatches an action to	`dispatch({type:"SET_VALUE",value:42})`

```
Redux Saga Sandbox                                              index.js:21
> let fn = ()=>{console.log("Called the function")}
< undefined
> let saga = function*(){ yield fn() }
< undefined
> run(saga)
  Called the function                                           VM1996:1
< ▶Object {@@redux-saga/TASK: true, id: 1, name: "saga", cont: undefined,
  joiners: null…}
> saga = function*(){ yield effects.call(fn)}
< function* (){ yield effects.call(fn)}
> run(saga)
  Called the function                                           VM1996:1
< ▶Object {@@redux-saga/TASK: true, id: 3, name: "saga", cont: undefined,
  joiners: null…}
> |
```

BINAR ACADEMY

Implementing Take-Put-Call in the App



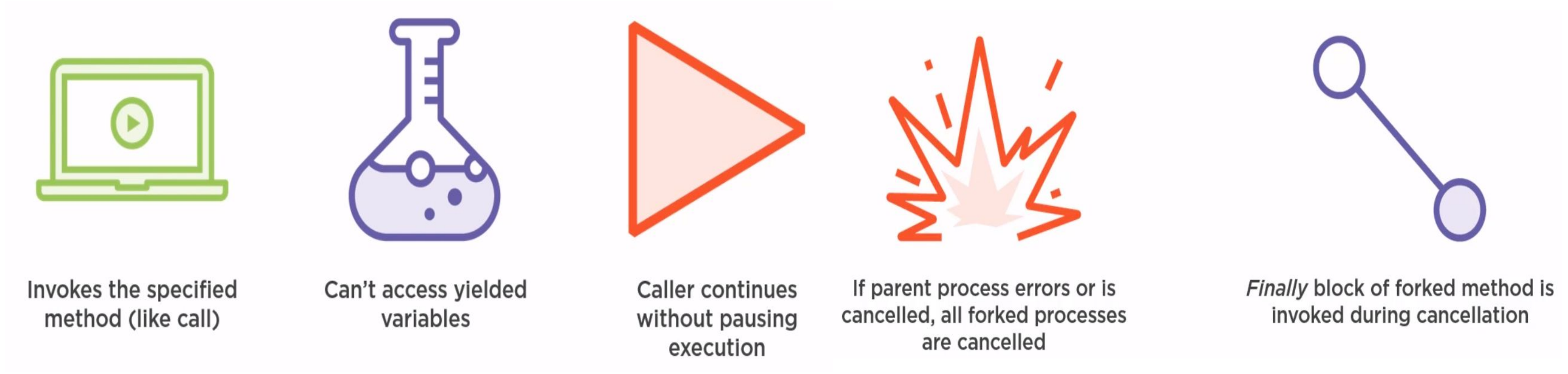
```
import { take, put, call, apply } from 'redux-saga/effects'
import fetch from 'isomorphic-fetch';

import {
    GET_CURRENT_USER_INFO,
    setCurrentUser
} from './../actions'

export function* currentUserSaga () {
    const { id } = yield take(GET_CURRENT_USER_INFO);
    const response = yield call(fetch,`http://localhost:8081/user/${id}`);
    const data = yield apply(response, response.json);
    yield put(setCurrentUser(data));
}
```

BINAR ACADEMY

4) Fork



Invokes the specified method (like call)

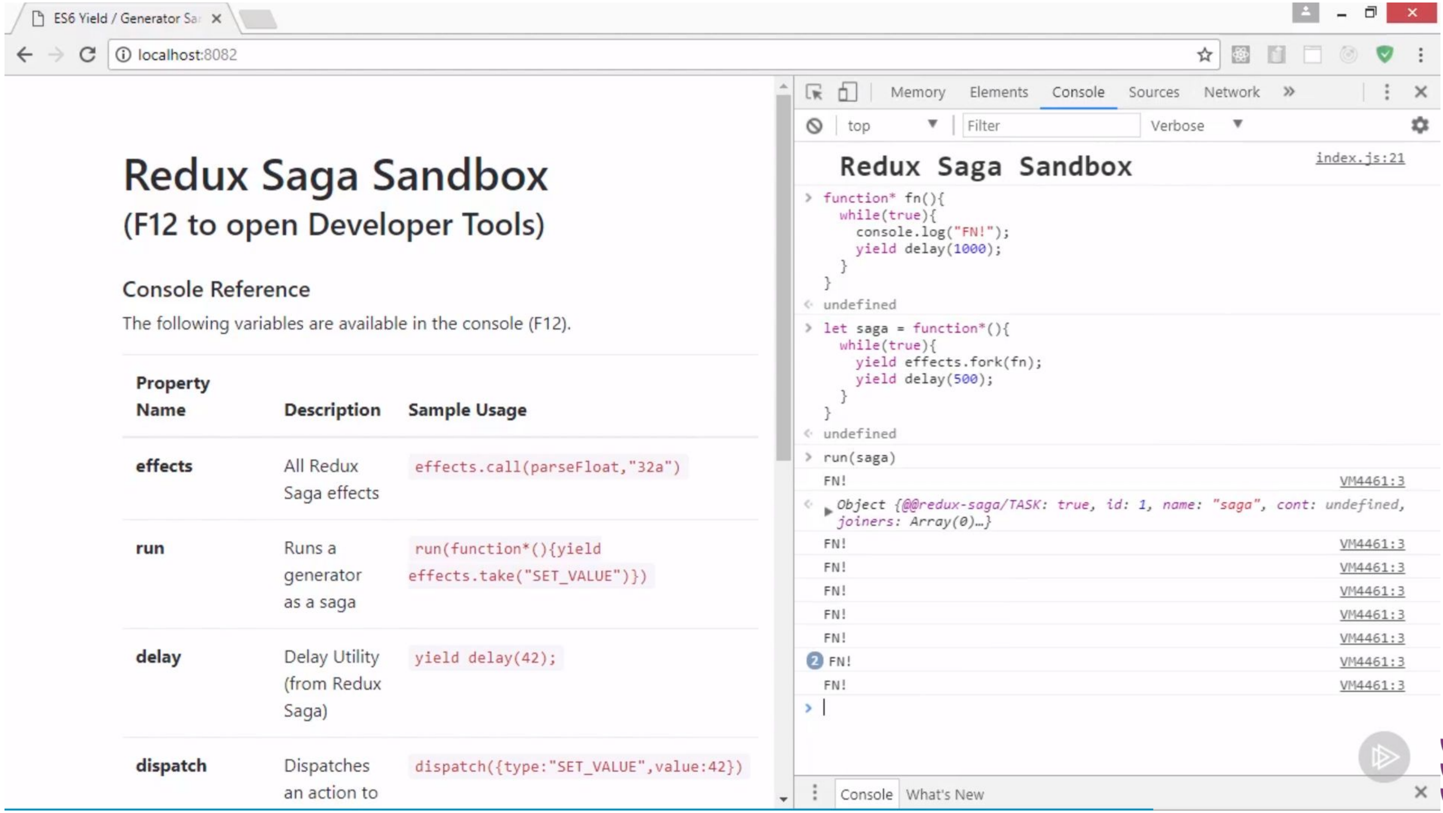
Can't access yielded variables

Caller continues without pausing execution

If parent process errors or is cancelled, all forked processes are cancelled

Finally block of forked method is invoked during cancellation

Implementing Fork



Redux Saga Sandbox
(F12 to open Developer Tools)

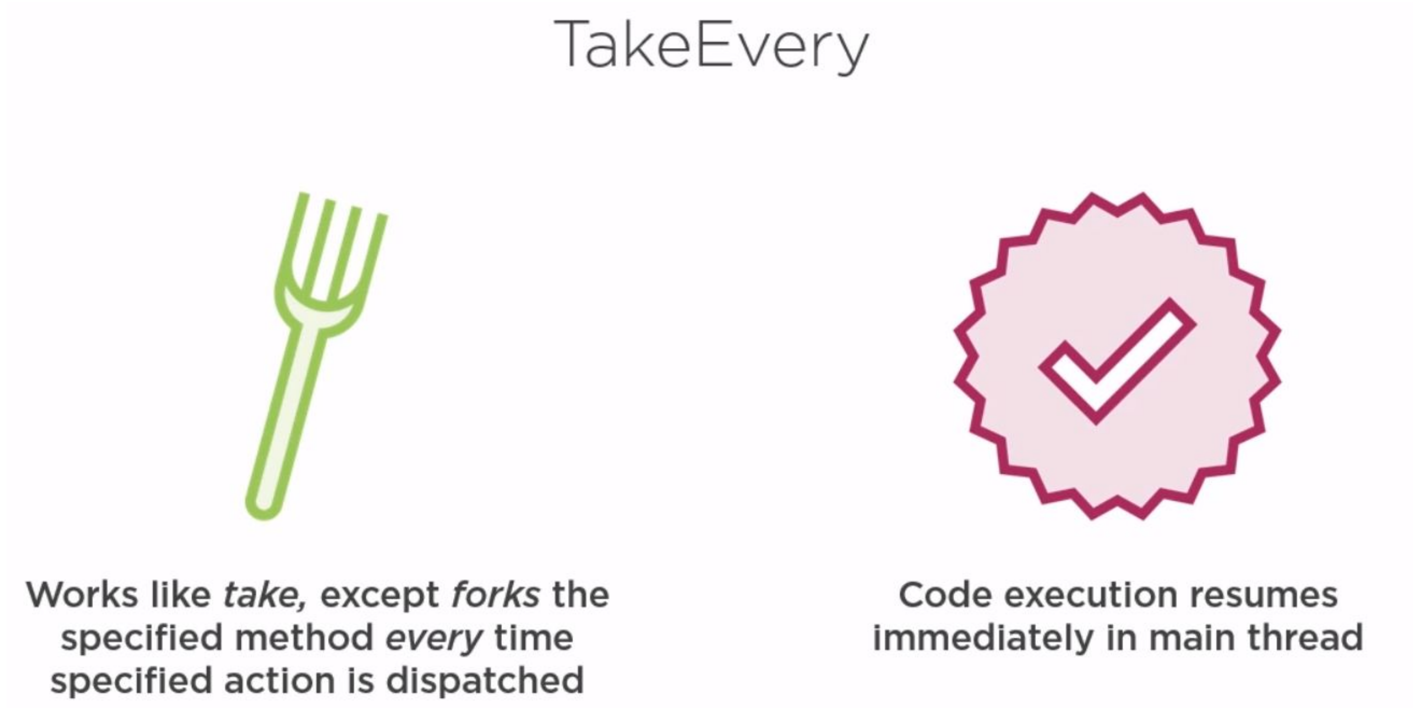
Console Reference

The following variables are available in the console (F12).

Property Name	Description	Sample Usage
effects	All Redux Saga effects	`effects.call(parseFloat,"32a")`
run	Runs a generator as a saga	`run(function*(){yield effects.take("SET_VALUE")})`
delay	Delay Utility (from Redux Saga)	`yield delay(42);`
dispatch	Dispatches an action to	`dispatch({type:"SET_VALUE",value:42})`

```
Redux Saga Sandbox                                              index.js:21
> function* fn(){
    while(true){
      console.log("FN!");
      yield delay(1000);
    }
  }
< undefined
> let saga = function*(){
    while(true){
      yield effects.fork(fn);
      yield delay(500);
    }
  }
< undefined
> run(saga)
  FN!                                                           VM4461:3
< ▸Object {@@redux-saga/TASK: true, id: 1, name: "saga", cont: undefined,
  joiners: Array(0)…}
  FN!                                                           VM4461:3
  FN!                                                           VM4461:3
  FN!                                                           VM4461:3
  FN!                                                           VM4461:3
  FN!                                                           VM4461:3
2 FN!                                                           VM4461:3
  FN!                                                           VM4461:3
>
```

5) Take Every

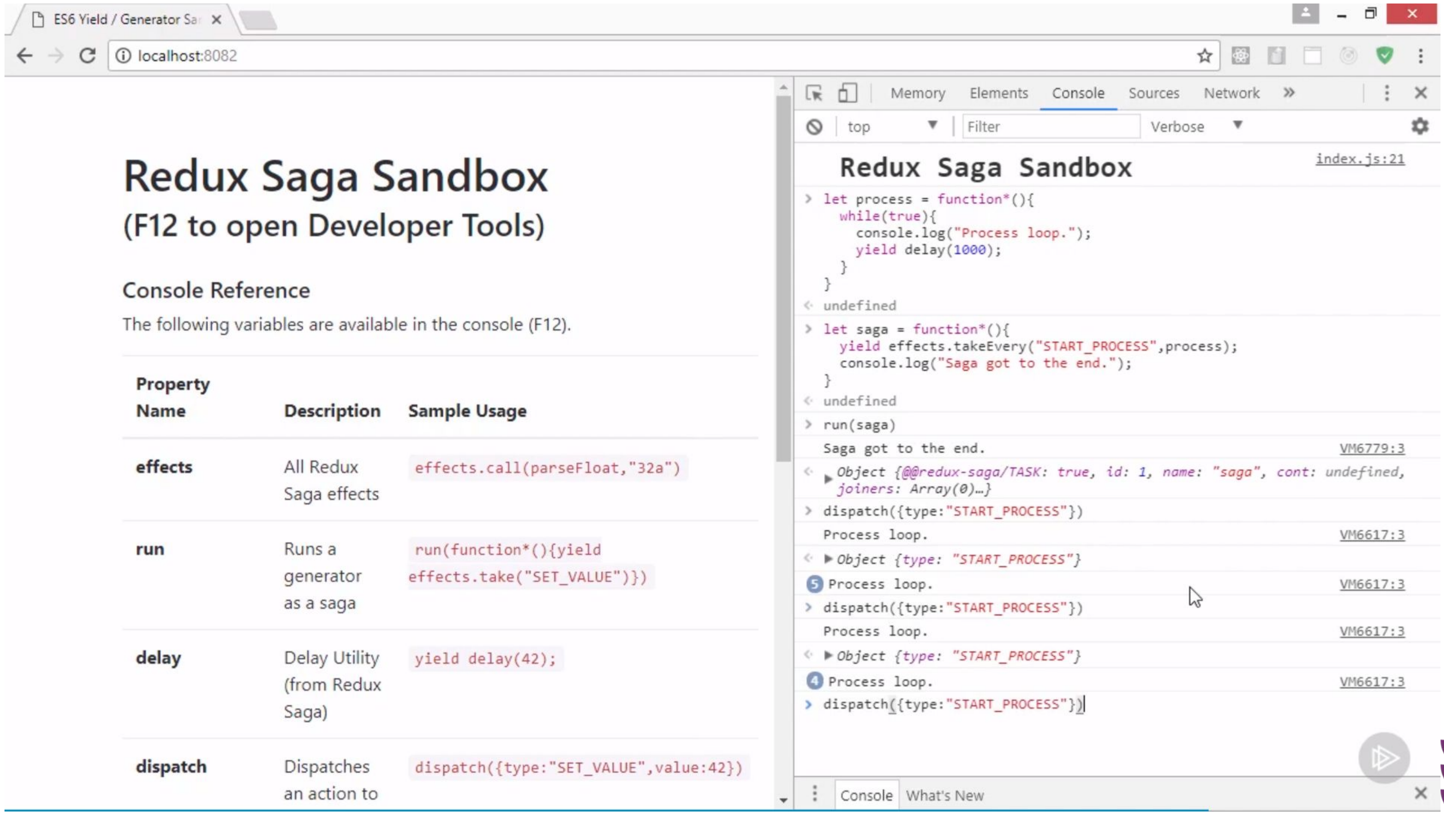


TakeEvery

Works like *take,* except *forks* the specified method *every* time specified action is dispatched

Code execution resumes immediately in main thread

Implementing Take Every



ES6 Yield / Generator Sa

localhost:8082

Redux Saga Sandbox

(F12 to open Developer Tools)

Console Reference

The following variables are available in the console (F12).

Property Name	Description	Sample Usage
effects	All Redux Saga effects	`effects.call(parseFloat,"32a")`
run	Runs a generator as a saga	`run(function*(){yield effects.take("SET_VALUE")})`
delay	Delay Utility (from Redux Saga)	`yield delay(42);`
dispatch	Dispatches an action to	`dispatch({type:"SET_VALUE",value:42})`

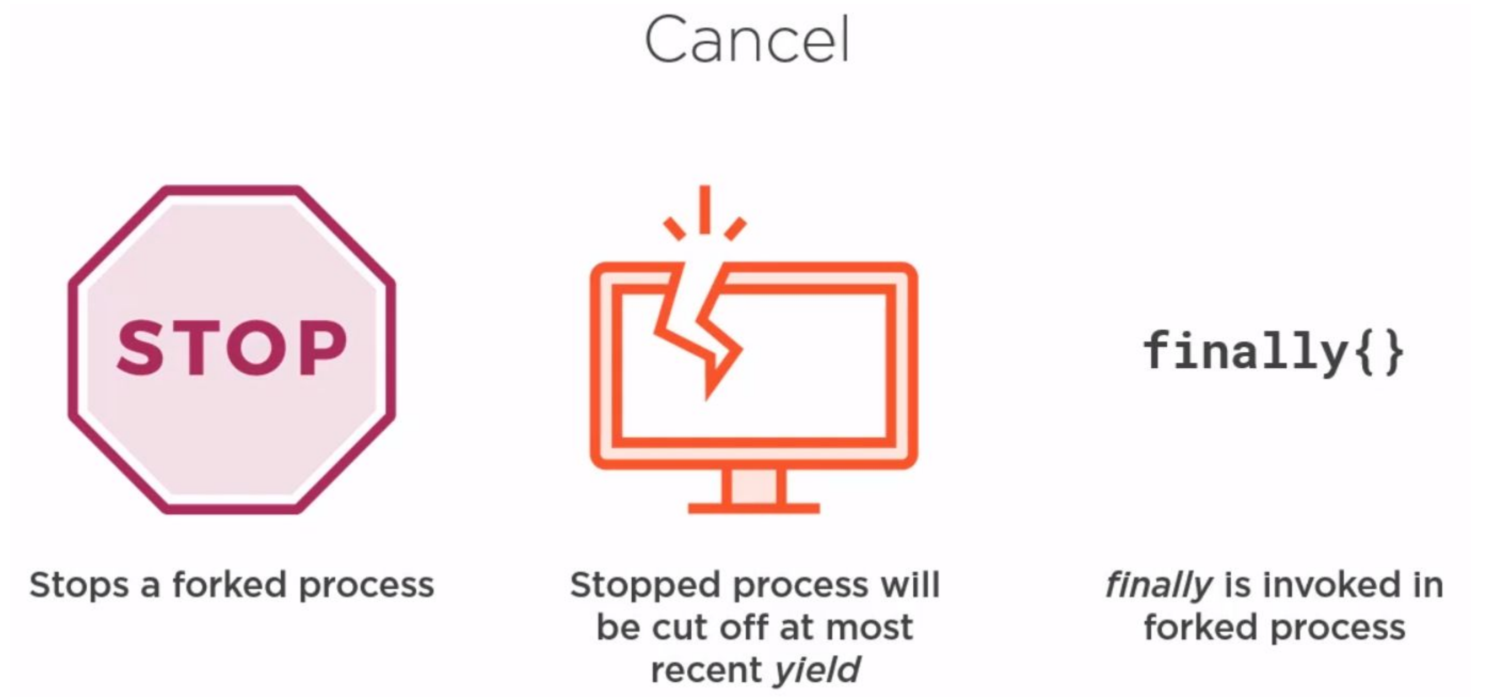
Memory Elements Console Sources Network

top Filter Verbose

```
Redux Saga Sandbox                                              index.js:21
> let process = function*(){
    while(true){
      console.log("Process loop.");
      yield delay(1000);
    }
  }
< undefined
> let saga = function*(){
    yield effects.takeEvery("START_PROCESS",process);
    console.log("Saga got to the end.");
  }
< undefined
> run(saga)
  Saga got to the end.                                          VM6779:3
< ▶Object {@@redux-saga/TASK: true, id: 1, name: "saga", cont: undefined,
  joiners: Array(0)…}
> dispatch({type:"START_PROCESS"})
  Process loop.                                                 VM6617:3
< ▶Object {type: "START_PROCESS"}
5 Process loop.                                                 VM6617:3
> dispatch({type:"START_PROCESS"})
  Process loop.                                                 VM6617:3
< ▶Object {type: "START_PROCESS"}
4 Process loop.                                                 VM6617:3
> dispatch({type:"START_PROCESS"})
```

Console What's New

6) Cancel & Cancelled



Cancel

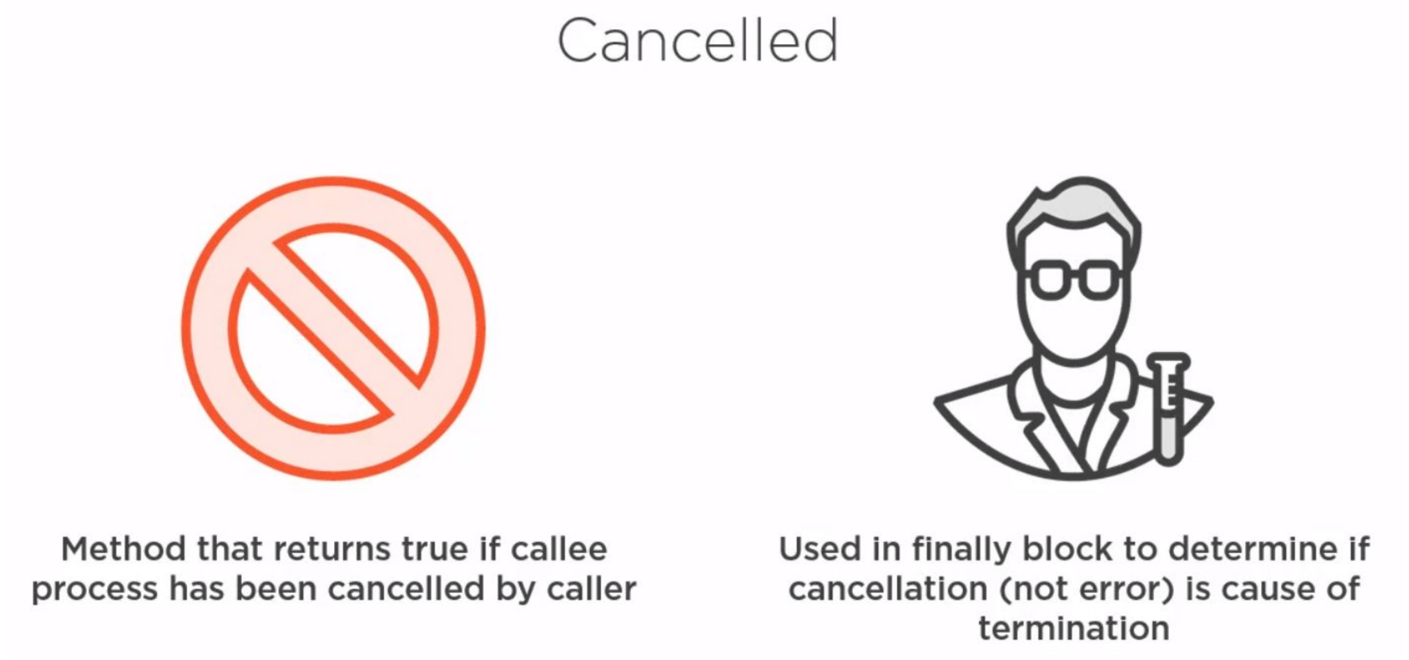
Stops a forked process

Stopped process will be cut off at most recent *yield*

```
finally{}
```

finally is invoked in forked process

Cancelled

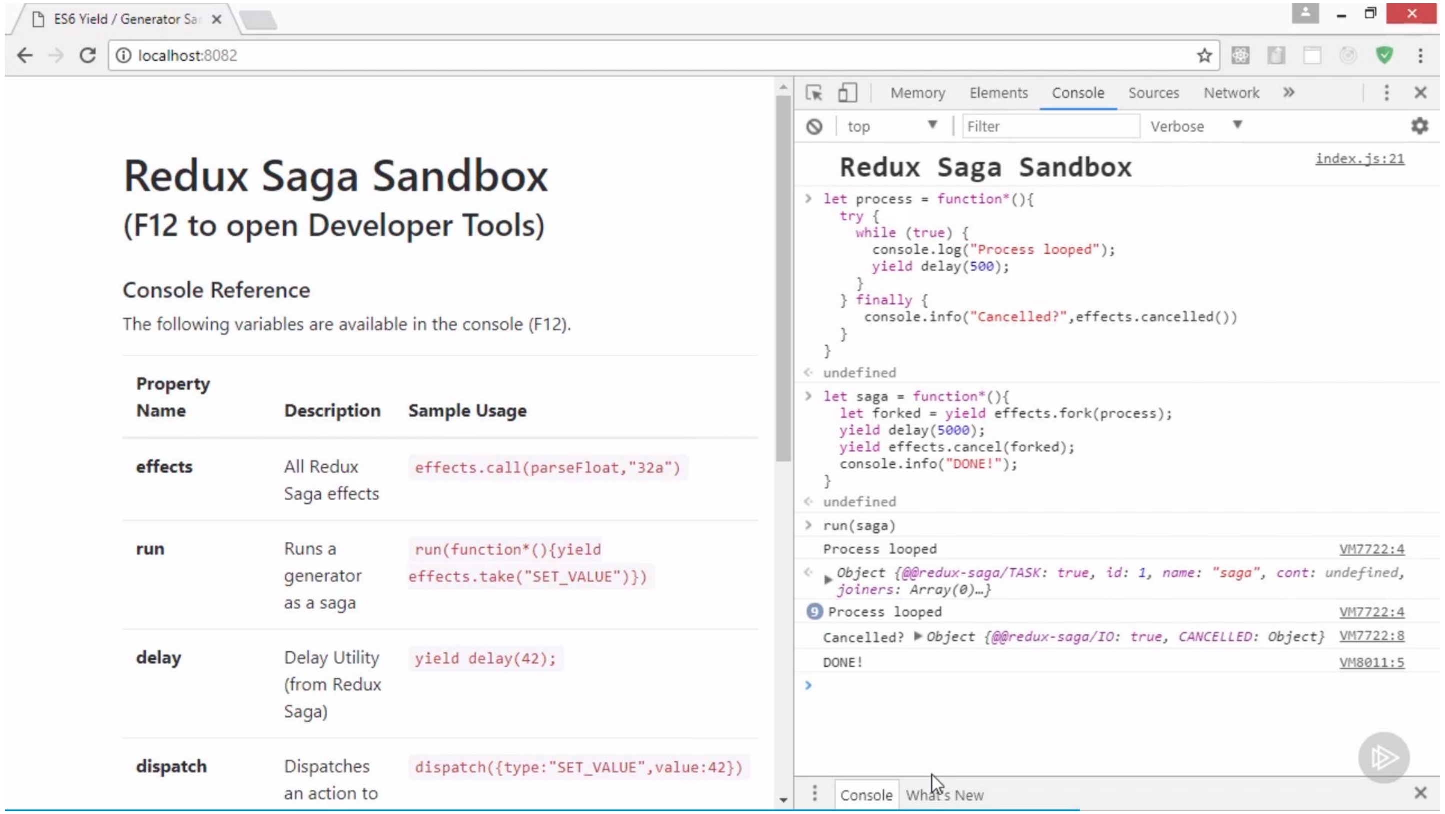


Cancelled

Method that returns true if callee process has been cancelled by caller

Used in finally block to determine if cancellation (not error) is cause of termination

Implementing Cancel & Cancelled (1)



ES6 Yield / Generator Sa
localhost:8082

Redux Saga Sandbox
(F12 to open Developer Tools)

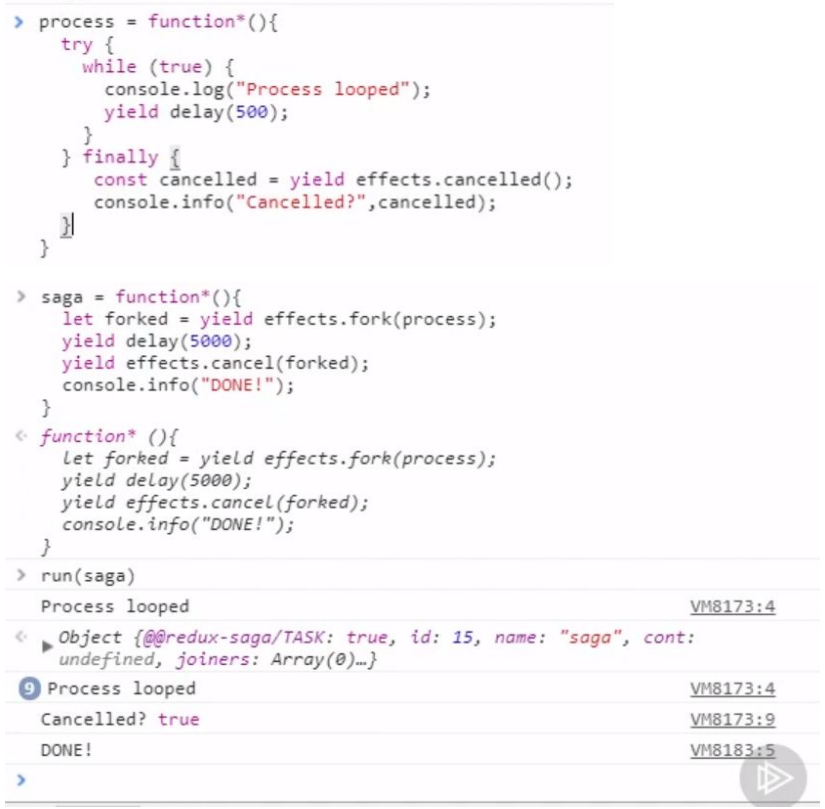
Console Reference

The following variables are available in the console (F12).

Property Name	Description	Sample Usage
effects	All Redux Saga effects	`effects.call(parseFloat,"32a")`
run	Runs a generator as a saga	`run(function*(){yield effects.take("SET_VALUE")})`
delay	Delay Utility (from Redux Saga)	`yield delay(42);`
dispatch	Dispatches an action to	`dispatch({type:"SET_VALUE",value:42})`

```
Redux Saga Sandbox                                              index.js:21
> let process = function*(){
    try {
      while (true) {
        console.log("Process looped");
        yield delay(500);
      }
    } finally {
      console.info("Cancelled?",effects.cancelled())
    }
  }
< undefined
> let saga = function*(){
    let forked = yield effects.fork(process);
    yield delay(5000);
    yield effects.cancel(forked);
    console.info("DONE!");
  }
< undefined
> run(saga)
  Process looped                                                VM7722:4
< ▶Object {@@redux-saga/TASK: true, id: 1, name: "saga", cont: undefined,
   joiners: Array(0)…}
9 Process looped                                                VM7722:4
  Cancelled? ▶Object {@@redux-saga/IO: true, CANCELLED: Object} VM7722:8
  DONE!                                                         VM8011:5
>
```

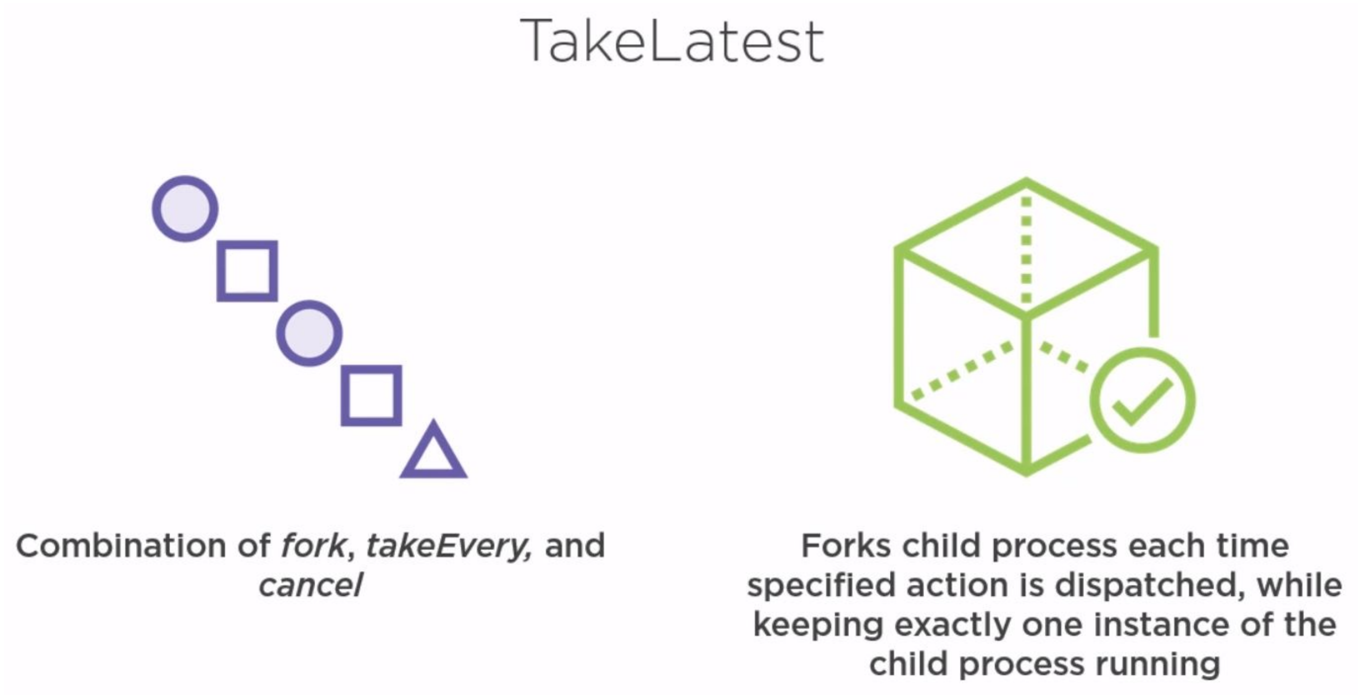
Implementing Cancel & Cancelled (2)



```
> process = function*(){
    try {
      while (true) {
        console.log("Process looped");
        yield delay(500);
      }
    } finally {
      const cancelled = yield effects.cancelled();
      console.info("Cancelled?",cancelled);
    }
  }

> saga = function*(){
    let forked = yield effects.fork(process);
    yield delay(5000);
    yield effects.cancel(forked);
    console.info("DONE!");
  }
< function* (){
    let forked = yield effects.fork(process);
    yield delay(5000);
    yield effects.cancel(forked);
    console.info("DONE!");
  }
> run(saga)
  Process looped                                     VM8173:4
< ▶ Object {@@redux-saga/TASK: true, id: 15, name: "saga", cont:
  undefined, joiners: Array(0)…}
9 Process looped                                     VM8173:4
  Cancelled? true                                    VM8173:9
  DONE!                                              VM8183:5
>
```

7) Take Latest

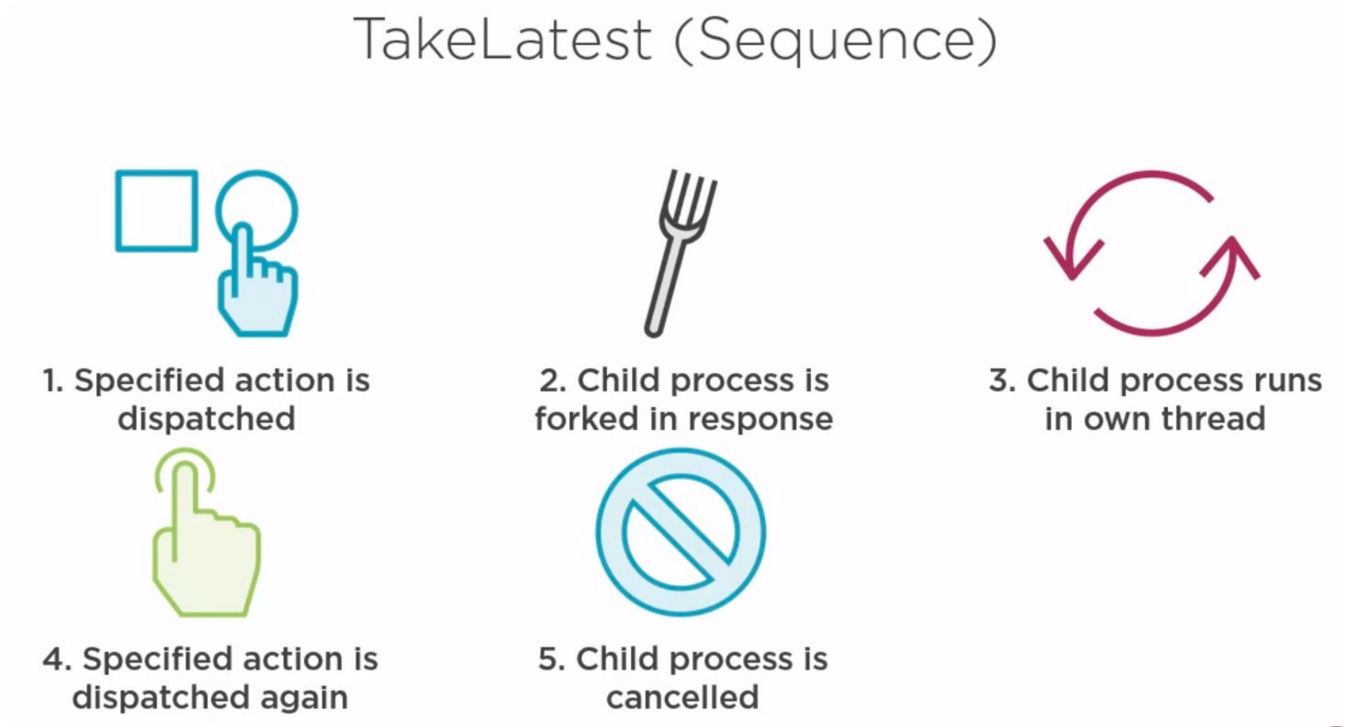


TakeLatest

Combination of *fork*, *takeEvery,* and *cancel*

Forks child process each time specified action is dispatched, while keeping exactly one instance of the child process running

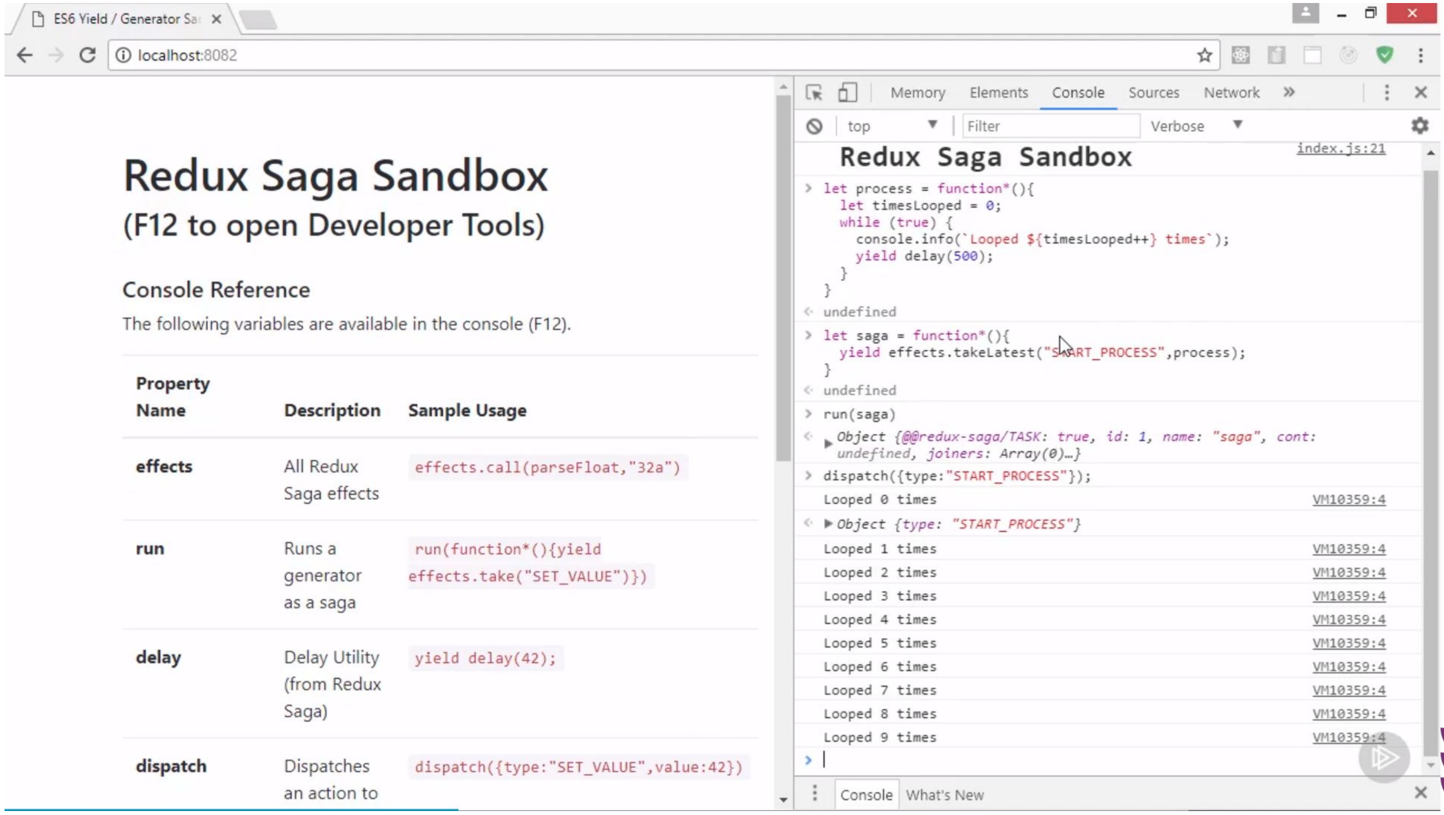
Take Latest (Sequence)



TakeLatest (Sequence)

1. Specified action is dispatched
2. Child process is forked in response
3. Child process runs in own thread
4. Specified action is dispatched again
5. Child process is cancelled

Implementing Take Latest



ES6 Yield / Generator Sa

localhost:8082

Redux Saga Sandbox

(F12 to open Developer Tools)

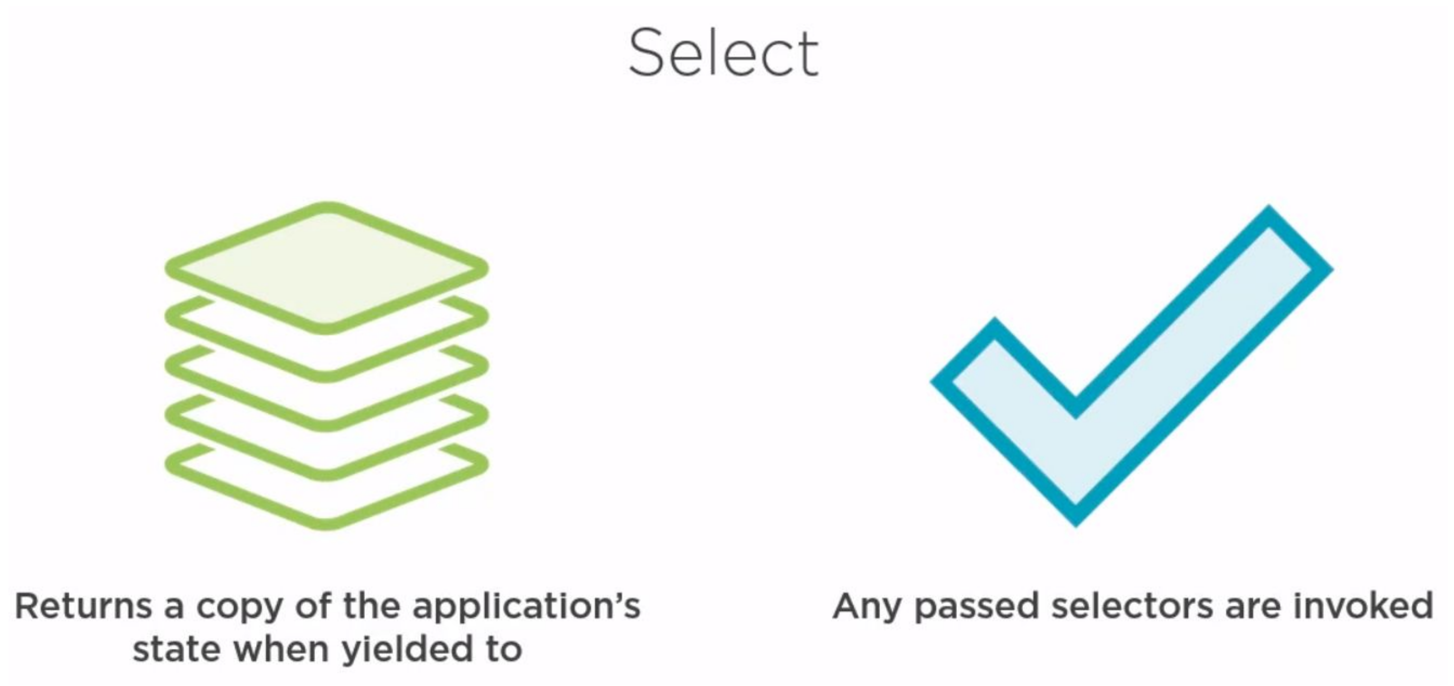
Console Reference

The following variables are available in the console (F12).

Property Name	Description	Sample Usage
effects	All Redux Saga effects	`effects.call(parseFloat,"32a")`
run	Runs a generator as a saga	`run(function*(){yield effects.take("SET_VALUE")})`
delay	Delay Utility (from Redux Saga)	`yield delay(42);`
dispatch	Dispatches an action to	`dispatch({type:"SET_VALUE",value:42})`

```
Redux Saga Sandbox                                           index.js:21
> let process = function*(){
    let timesLooped = 0;
    while (true) {
      console.info(`Looped ${timesLooped++} times`);
      yield delay(500);
    }
  }
< undefined
> let saga = function*(){
    yield effects.takeLatest("START_PROCESS",process);
  }
< undefined
> run(saga)
< ▶Object {@@redux-saga/TASK: true, id: 1, name: "saga", cont:
   undefined, joiners: Array(0)…}
> dispatch({type:"START_PROCESS"});
  Looped 0 times                                            VM10359:4
< ▶Object {type: "START_PROCESS"}
  Looped 1 times                                            VM10359:4
  Looped 2 times                                            VM10359:4
  Looped 3 times                                            VM10359:4
  Looped 4 times                                            VM10359:4
  Looped 5 times                                            VM10359:4
  Looped 6 times                                            VM10359:4
  Looped 7 times                                            VM10359:4
  Looped 8 times                                            VM10359:4
  Looped 9 times                                            VM10359:4
>
```

8) Select

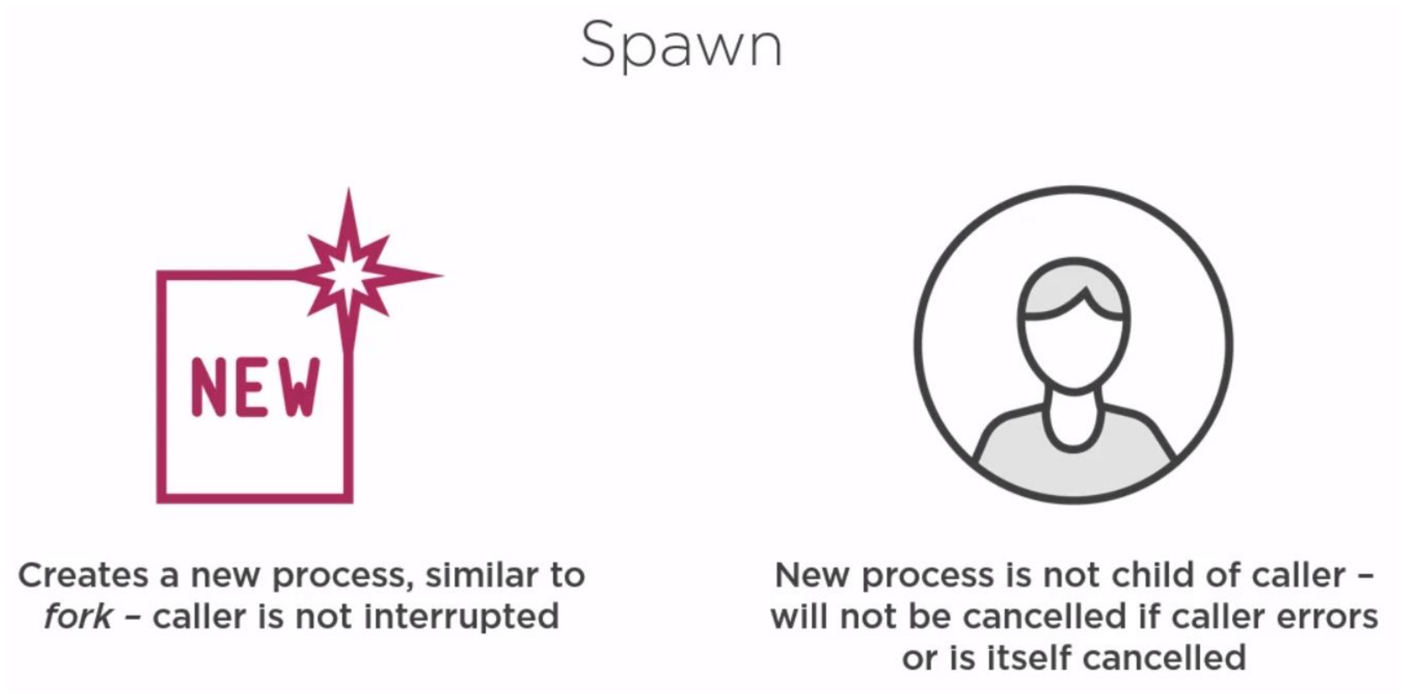


Select

Returns a copy of the application's state when yielded to

Any passed selectors are invoked

9) Spawn

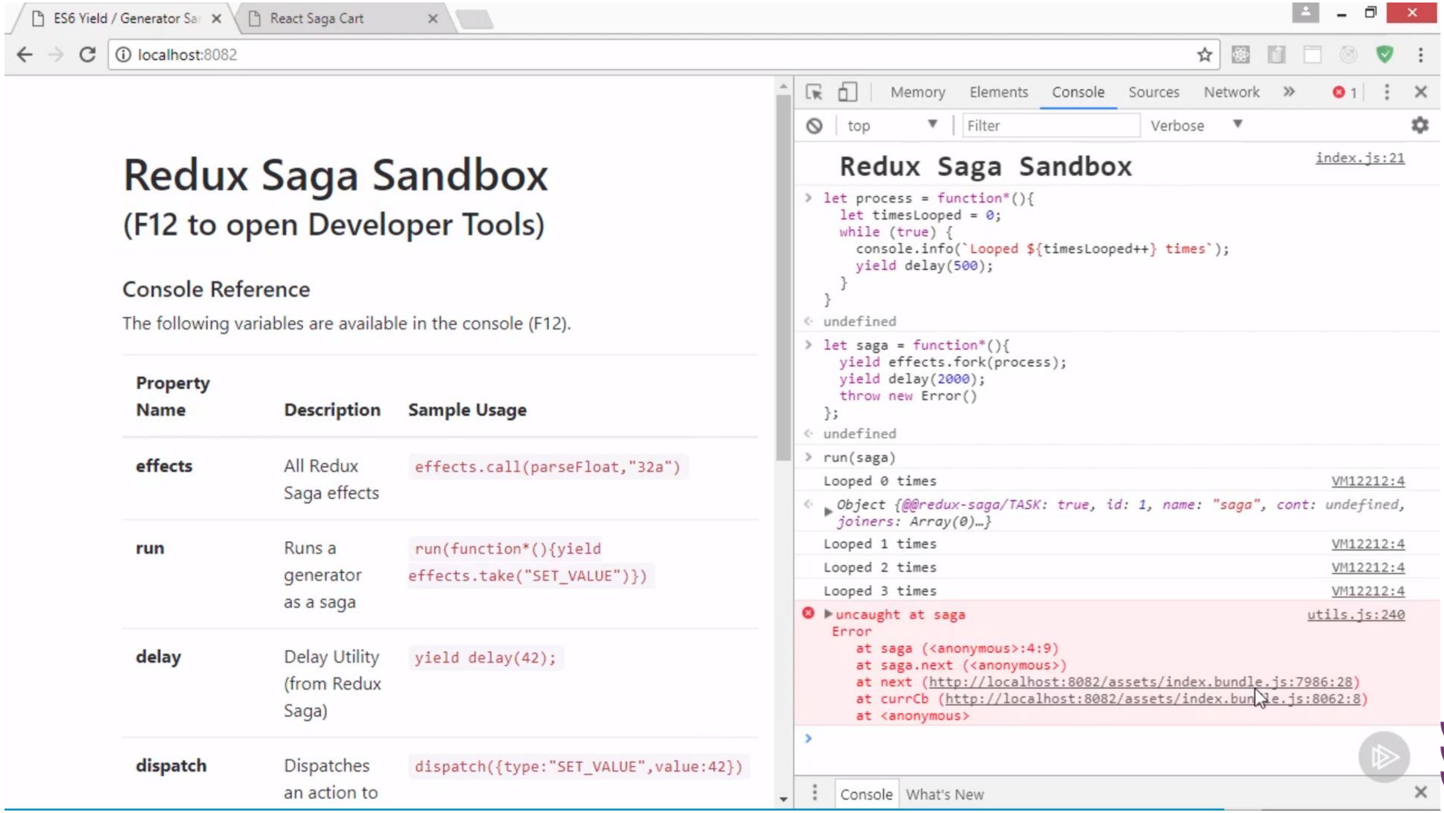


Spawn

Creates a new process, similar to *fork* – caller is not interrupted

New process is not child of caller – will not be cancelled if caller errors or is itself cancelled

Implementing Spawn



Redux Saga Sandbox

(F12 to open Developer Tools)

Console Reference

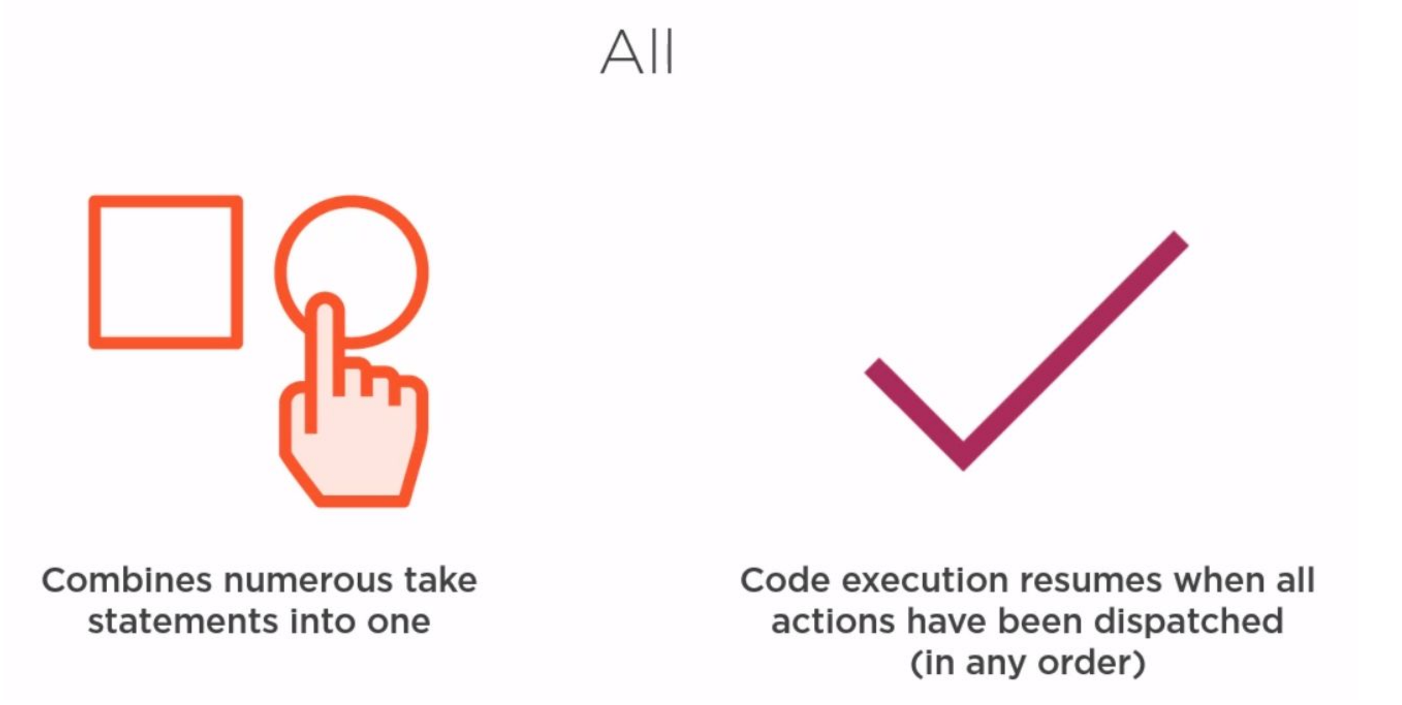
The following variables are available in the console (F12).

Property Name	Description	Sample Usage
effects	All Redux Saga effects	`effects.call(parseFloat,"32a")`
run	Runs a generator as a saga	`run(function*(){yield effects.take("SET_VALUE")})`
delay	Delay Utility (from Redux Saga)	`yield delay(42);`
dispatch	Dispatches an action to	`dispatch({type:"SET_VALUE",value:42})`

```
Redux Saga Sandbox                                                  index.js:21
> let process = function*(){
    let timesLooped = 0;
    while (true) {
      console.info(`Looped ${timesLooped++} times`);
      yield delay(500);
    }
  }
< undefined
> let saga = function*(){
    yield effects.fork(process);
    yield delay(2000);
    throw new Error()
  };
< undefined
> run(saga)
  Looped 0 times                                                  VM12212:4
< ▶Object {@@redux-saga/TASK: true, id: 1, name: "saga", cont: undefined,
   joiners: Array(0)…}
  Looped 1 times                                                  VM12212:4
  Looped 2 times                                                  VM12212:4
  Looped 3 times                                                  VM12212:4
✖ ▶uncaught at saga                                               utils.js:240
   Error
      at saga (<anonymous>:4:9)
      at saga.next (<anonymous>)
      at next (http://localhost:8082/assets/index.bundle.js:7986:28)
      at currCb (http://localhost:8082/assets/index.bundle.js:8062:8)
      at <anonymous>
>
```

BINAR ACADEMY

10) All



All

Combines numerous take statements into one

Code execution resumes when all actions have been dispatched (in any order)

Redux Saga Effects Summary

Effects create plain objects – Redux Saga interprets them and executes processes

Take, TakeEvery and TakeLatest wait for a specific kind of action to create a new process

Call, Fork and Spawn create different kinds of new processes

Forked processes are cancelled when their parent is cancelled or errors

Take and *Call* pause the execution of caller process