How to unleash the power of

# Typescript

in your project

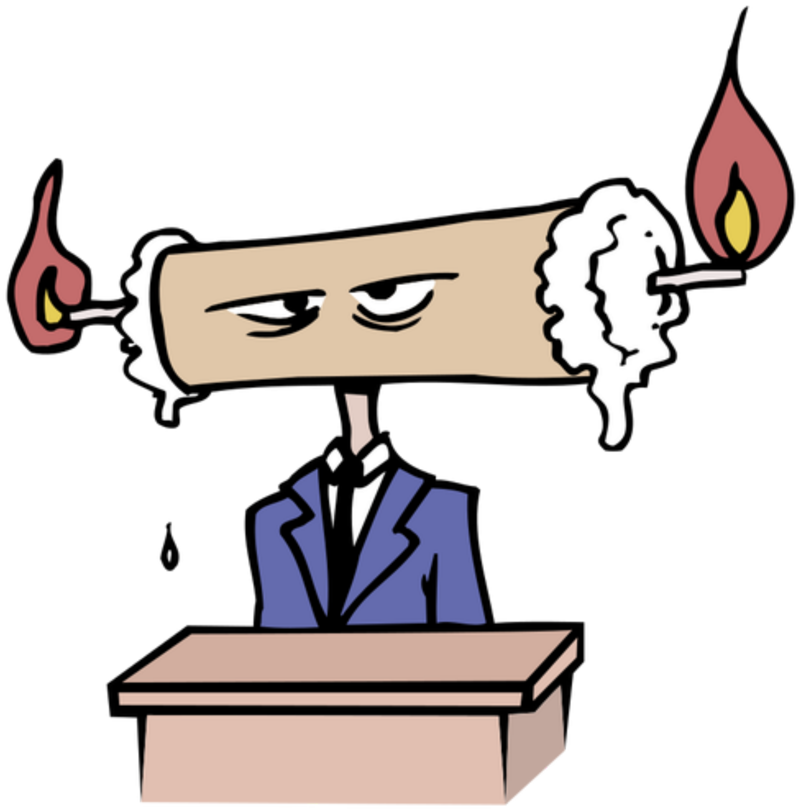
JĘDRZEJ
LEWANDOWSKI

cicha
obecność

ASK SOLI DEO

AMERYKA

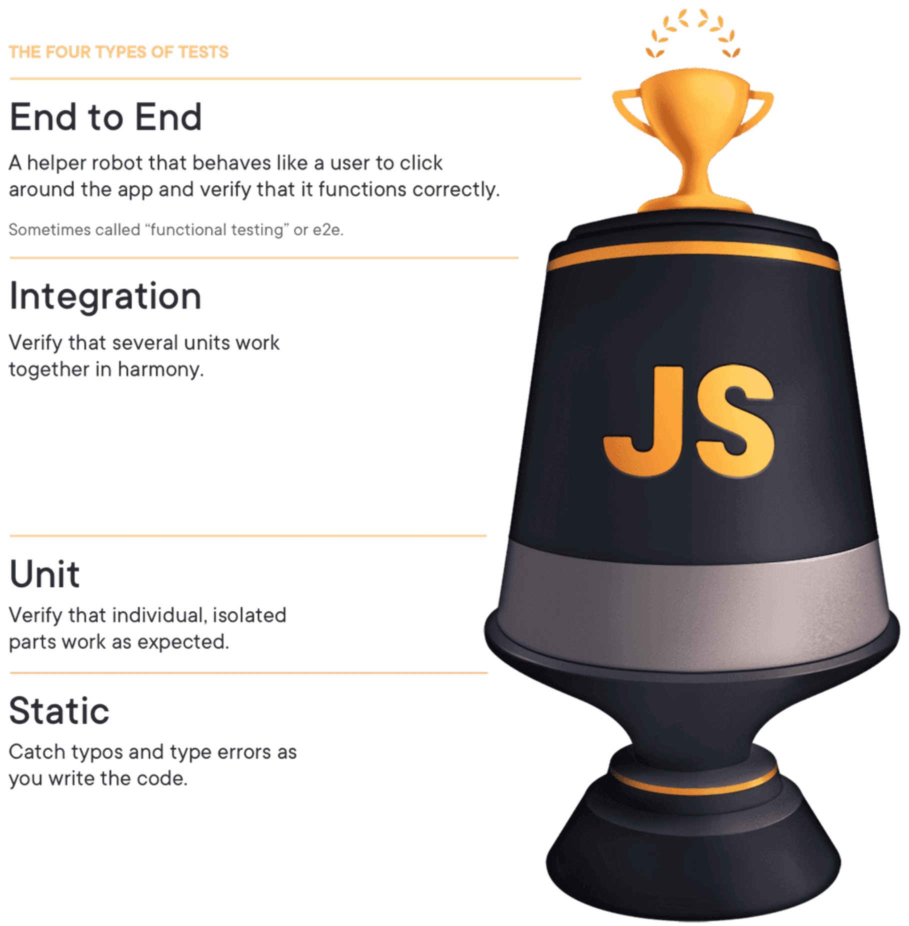
wise team



On the edge of 20’s… remember 00’s

## Benefits of using typescript

## Testing Trophy by Kent C. Dodds



THE FOUR TYPES OF TESTS

**End to End**

A helper robot that behaves like a user to click around the app and verify that it functions correctly.

Sometimes called "functional testing" or e2e.

**Integration**

Verify that several units work together in harmony.

**Unit**

Verify that individual, isolated parts work as expected.

**Static**

Catch typos and type errors as you write the code.

## End to End

A helper robot that behaves like a user to click around the app and verify that it functions correctly.

Sometimes called "functional testing" or e2e.

## Integration

Verify that several units work together in harmony.

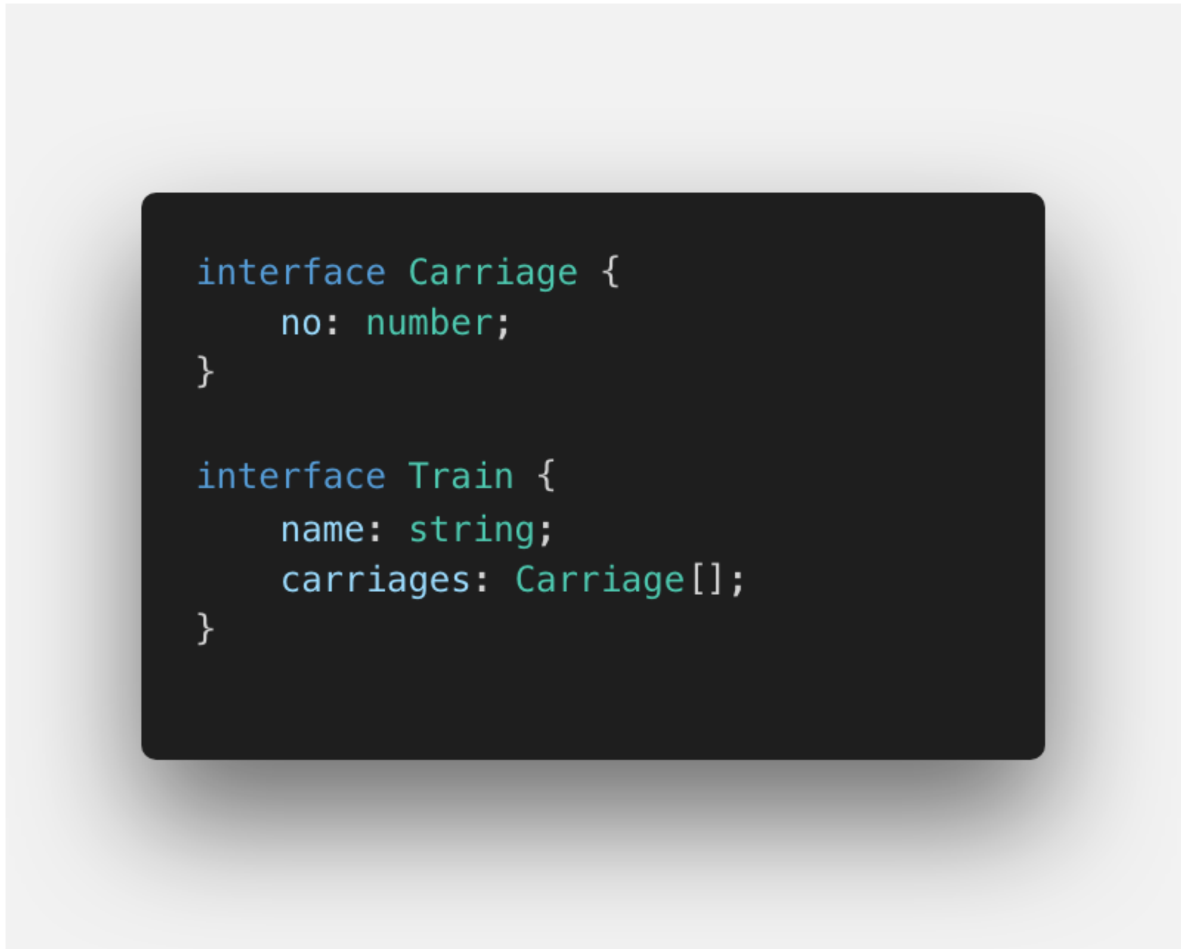
## Unit

Verify that individual, isolated parts work as expected.

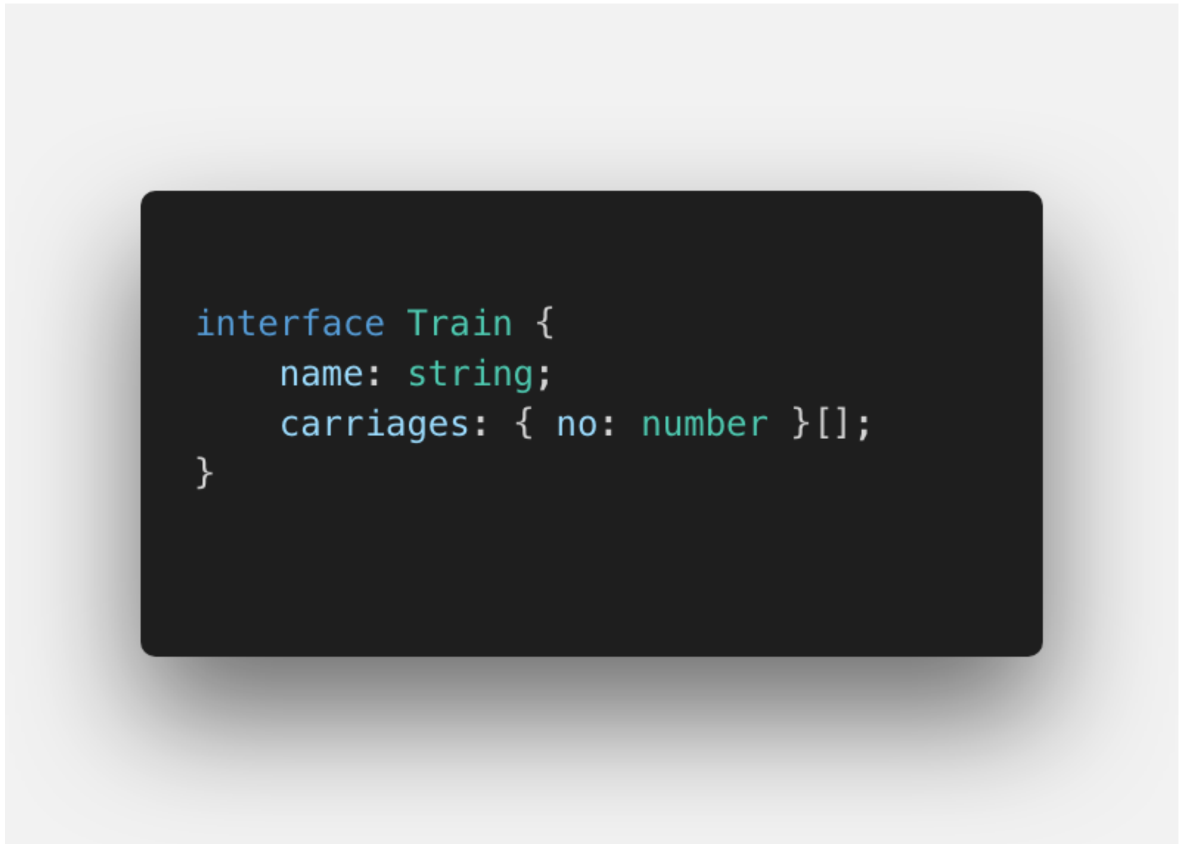
## Static

Catch typos and type errors as you write the code.

## Structural types = interoperability



```
interface Carriage {
    no: number;
}

interface Train {
    name: string;
    carriages: Carriage[];
}
```



```
interface Train {
    name: string;
    carriages: { no: number }[];
}
```

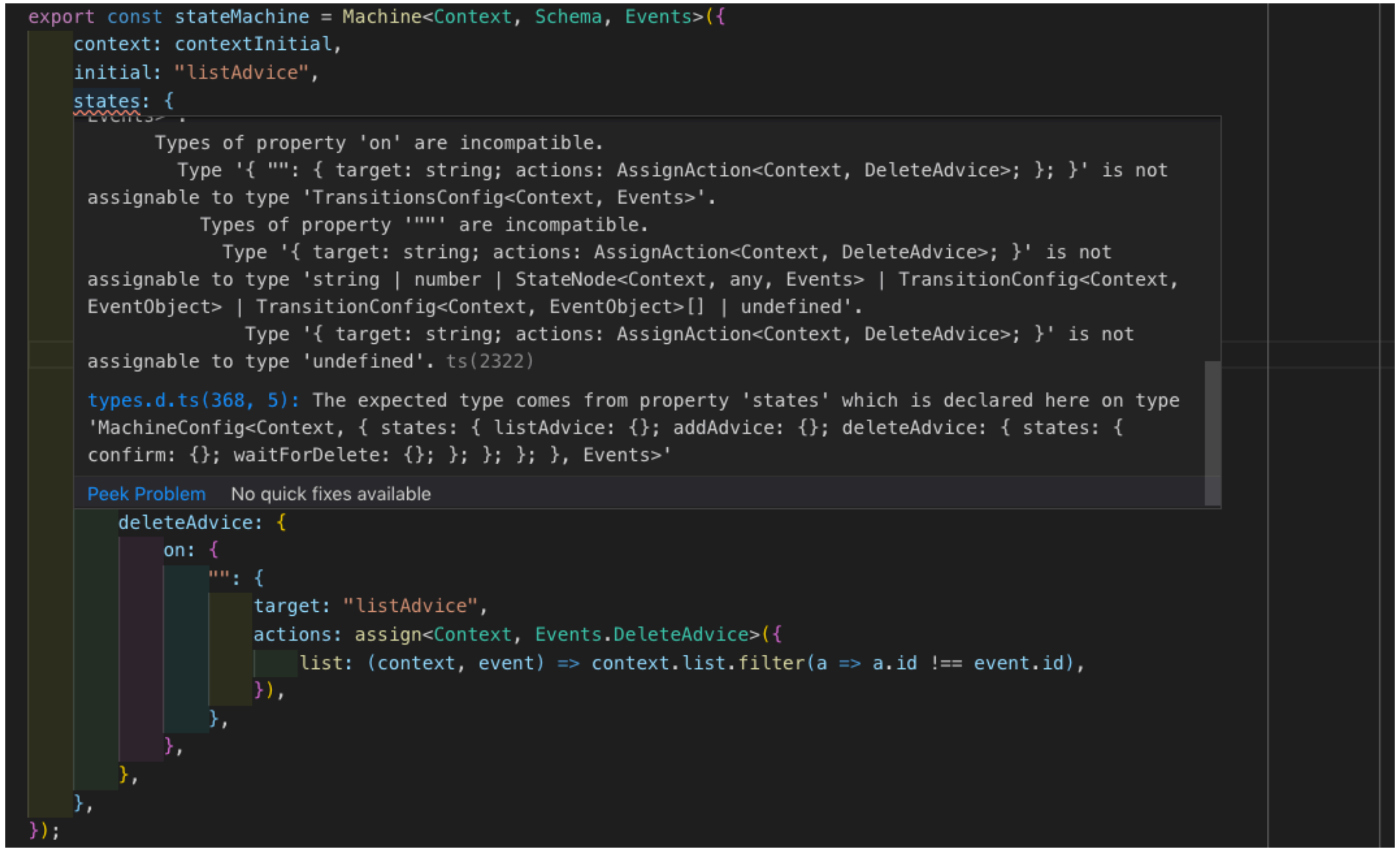
## Typescript related obstacles

- False type security
- Verbosity
- Repetitions
- Impact on bundle size

Fight…

## False type security

# Hidden type loss ⬄ wasting time on debugging

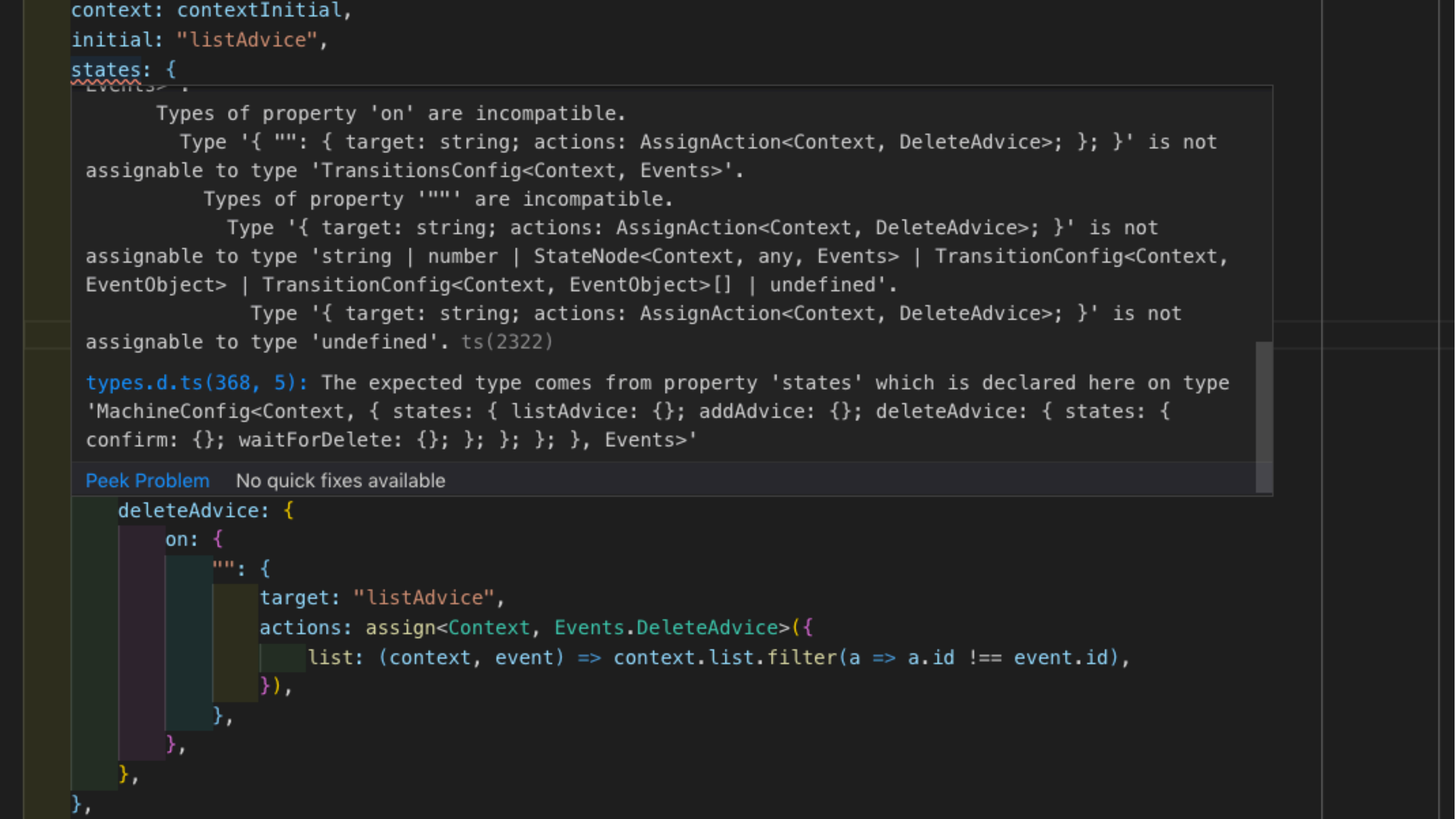


```
export const stateMachine = Machine<Context, Schema, Events>({
    context: contextInitial,
    initial: "listAdvice",
    states: {
```

```
      Types of property 'on' are incompatible.
        Type '{ "": { target: string; actions: AssignAction<Context, DeleteAdvice>; }; }' is not
assignable to type 'TransitionsConfig<Context, Events>'.
          Types of property '""' are incompatible.
            Type '{ target: string; actions: AssignAction<Context, DeleteAdvice>; }' is not
assignable to type 'string | number | StateNode<Context, any, Events> | TransitionConfig<Context,
EventObject> | TransitionConfig<Context, EventObject>[] | undefined'.
              Type '{ target: string; actions: AssignAction<Context, DeleteAdvice>; }' is not
assignable to type 'undefined'. ts(2322)

types.d.ts(368, 5): The expected type comes from property 'states' which is declared here on type
'MachineConfig<Context, { states: { listAdvice: {}; addAdvice: {}; deleteAdvice: { states: {
confirm: {}; waitForDelete: {}; }; }; }; }, Events>'
```

Peek Problem    No quick fixes available

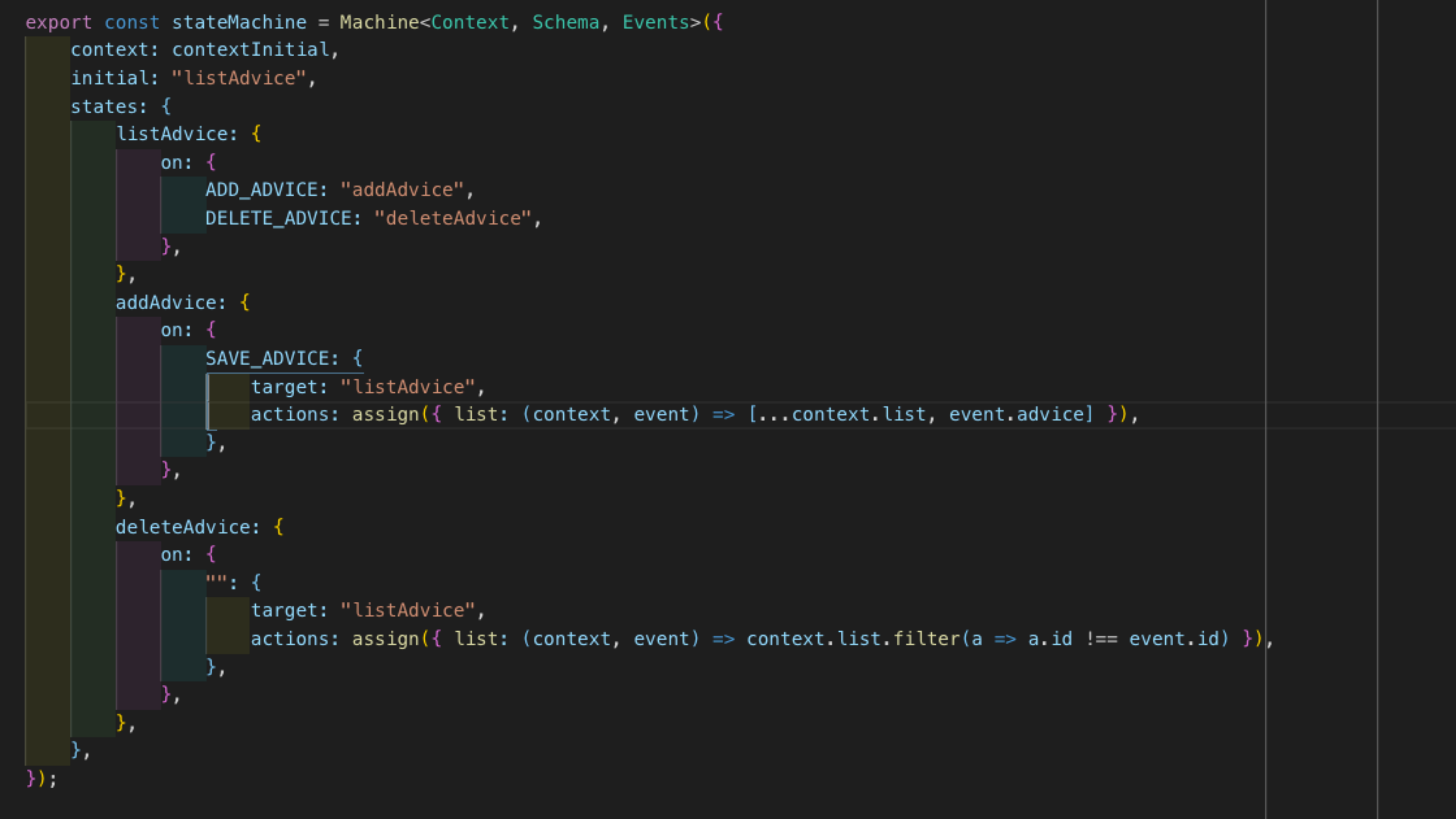
```
        deleteAdvice: {
            on: {
                "": {
                    target: "listAdvice",
                    actions: assign<Context, Events.DeleteAdvice>({
                        list: (context, event) => context.list.filter(a => a.id !== event.id),
                    }),
                },
            },
        },
    },
});
```



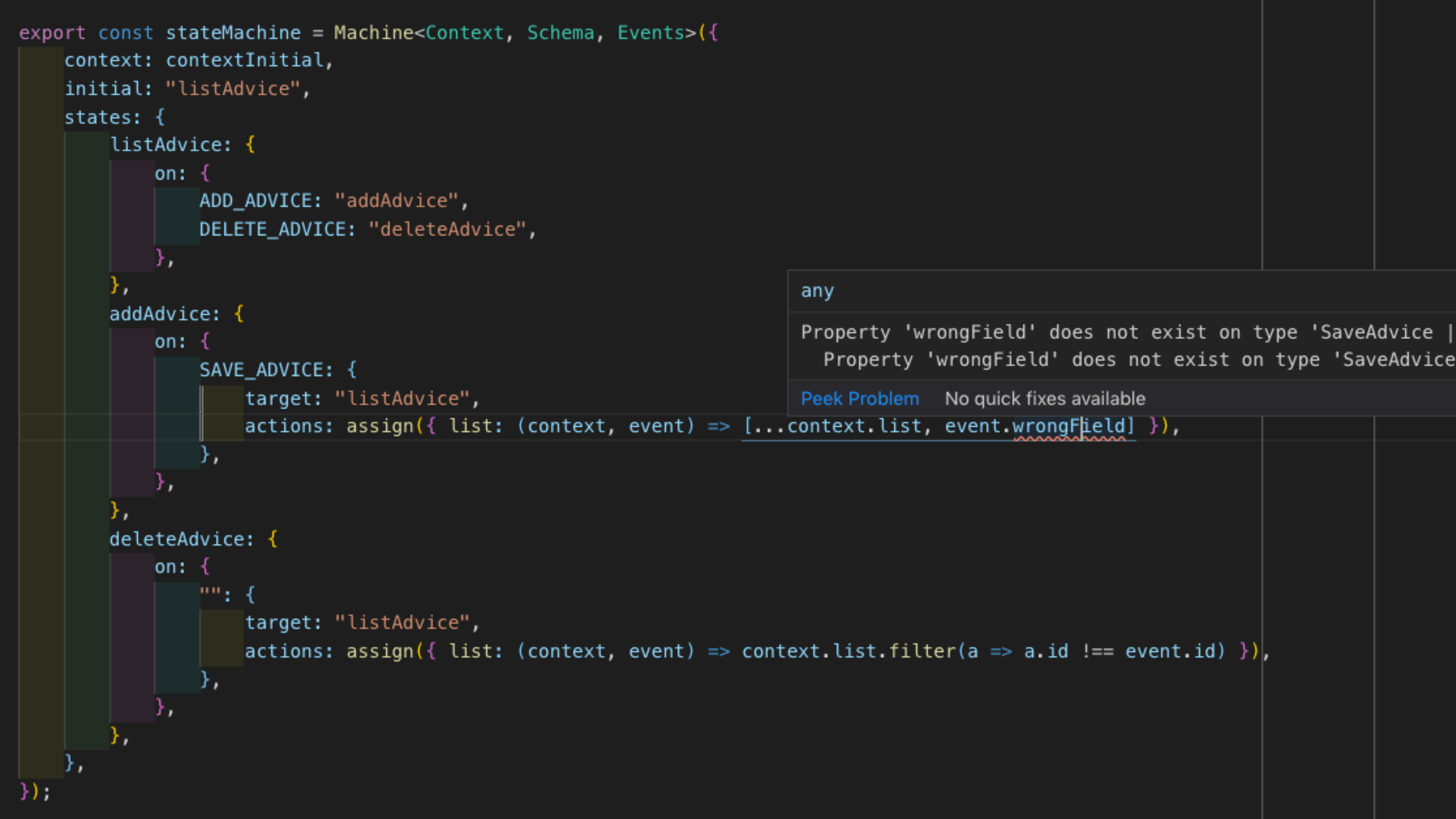
```
context: contextInitial,
initial: "listAdvice",
states: {
```

```
      Types of property 'on' are incompatible.
        Type '{ "": { target: string; actions: AssignAction<Context, DeleteAdvice>; }; }' is not
assignable to type 'TransitionsConfig<Context, Events>'.
          Types of property '""' are incompatible.
            Type '{ target: string; actions: AssignAction<Context, DeleteAdvice>; }' is not
assignable to type 'string | number | StateNode<Context, any, Events> | TransitionConfig<Context,
EventObject> | TransitionConfig<Context, EventObject>[] | undefined'.
              Type '{ target: string; actions: AssignAction<Context, DeleteAdvice>; }' is not
assignable to type 'undefined'. ts(2322)

types.d.ts(368, 5): The expected type comes from property 'states' which is declared here on type
'MachineConfig<Context, { states: { listAdvice: {}; addAdvice: {}; deleteAdvice: { states: {
confirm: {}; waitForDelete: {}; }; }; }; }, Events>'
```

```
  deleteAdvice: {
    on: {
      "": {
        target: "listAdvice",
        actions: assign<Context, Events.DeleteAdvice>({
          list: (context, event) => context.list.filter(a => a.id !== event.id),
        }),
      },
    },
  },
},
```

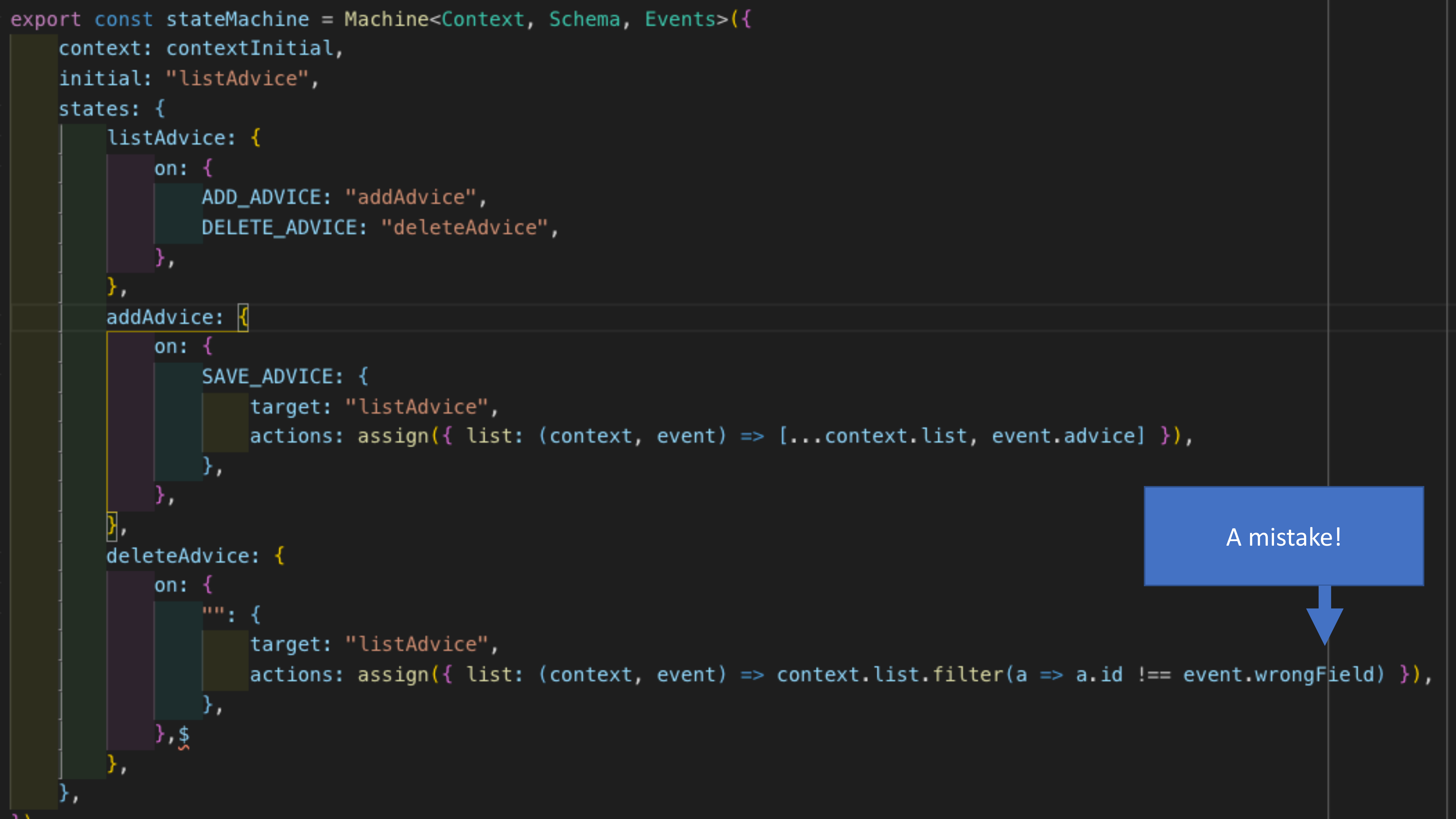


```
export const stateMachine = Machine<Context, Schema, Events>({
    context: contextInitial,
    initial: "listAdvice",
    states: {
        listAdvice: {
            on: {
                ADD_ADVICE: "addAdvice",
                DELETE_ADVICE: "deleteAdvice",
            },
        },
        addAdvice: {
            on: {
                SAVE_ADVICE: {
                    target: "listAdvice",
                    actions: assign({ list: (context, event) => [...context.list, event.advice] }),
                },
            },
        },
        deleteAdvice: {
            on: {
                "": {
                    target: "listAdvice",
                    actions: assign({ list: (context, event) => context.list.filter(a => a.id !== event.id) }),
                },
            },
        },
    },
});
```



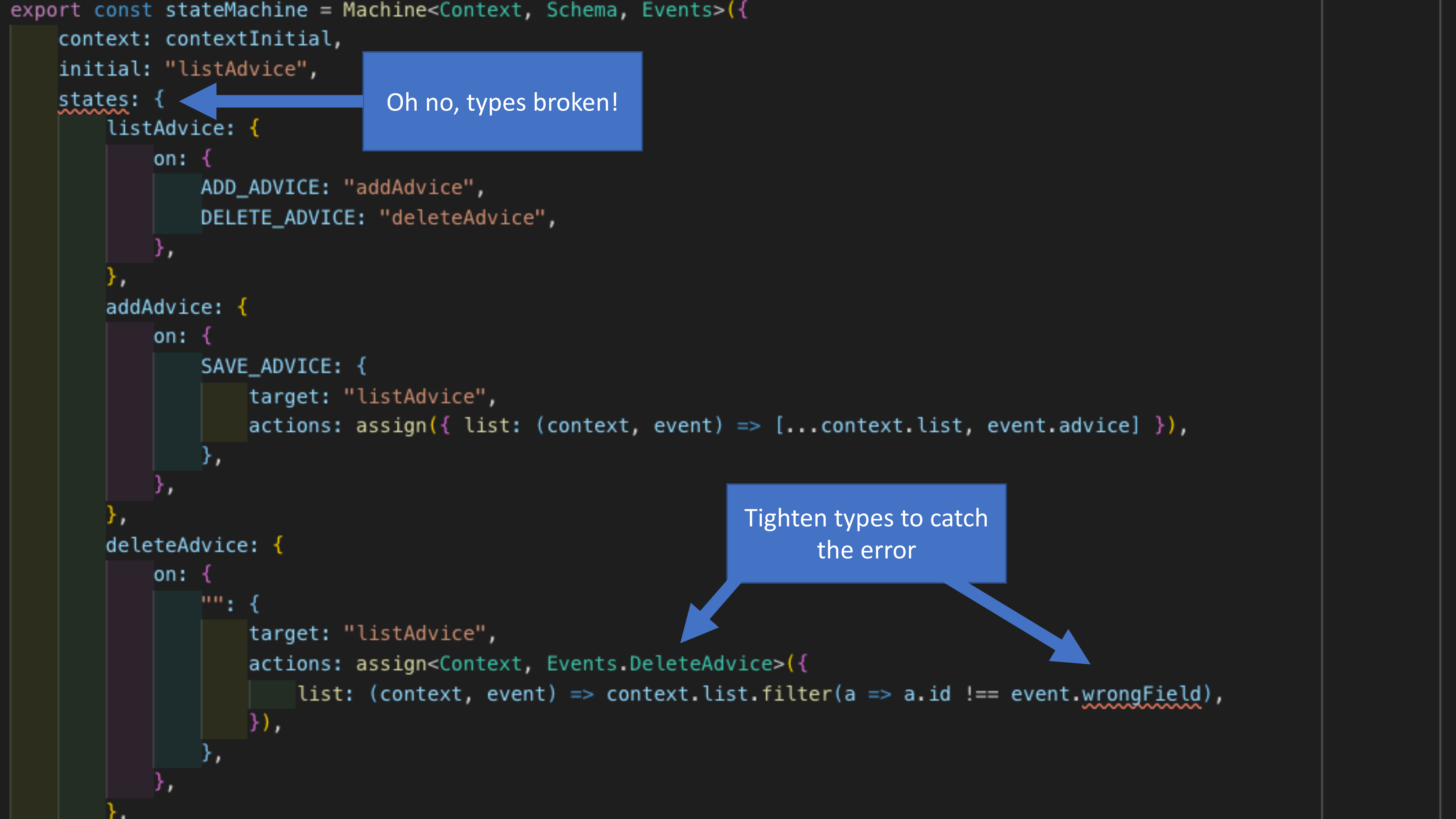
```
export const stateMachine = Machine<Context, Schema, Events>({
  context: contextInitial,
  initial: "listAdvice",
  states: {
    listAdvice: {
      on: {
        ADD_ADVICE: "addAdvice",
        DELETE_ADVICE: "deleteAdvice",
      },
    },
    addAdvice: {
      on: {
        SAVE_ADVICE: {
          target: "listAdvice",
          actions: assign({ list: (context, event) => [...context.list, event.wrongField] }),
        },
      },
    },
    deleteAdvice: {
      on: {
        "": {
          target: "listAdvice",
          actions: assign({ list: (context, event) => context.list.filter(a => a.id !== event.id) }),
        },
      },
    },
  },
});
```

```
any
Property 'wrongField' does not exist on type 'SaveAdvice |
  Property 'wrongField' does not exist on type 'SaveAdvice
```

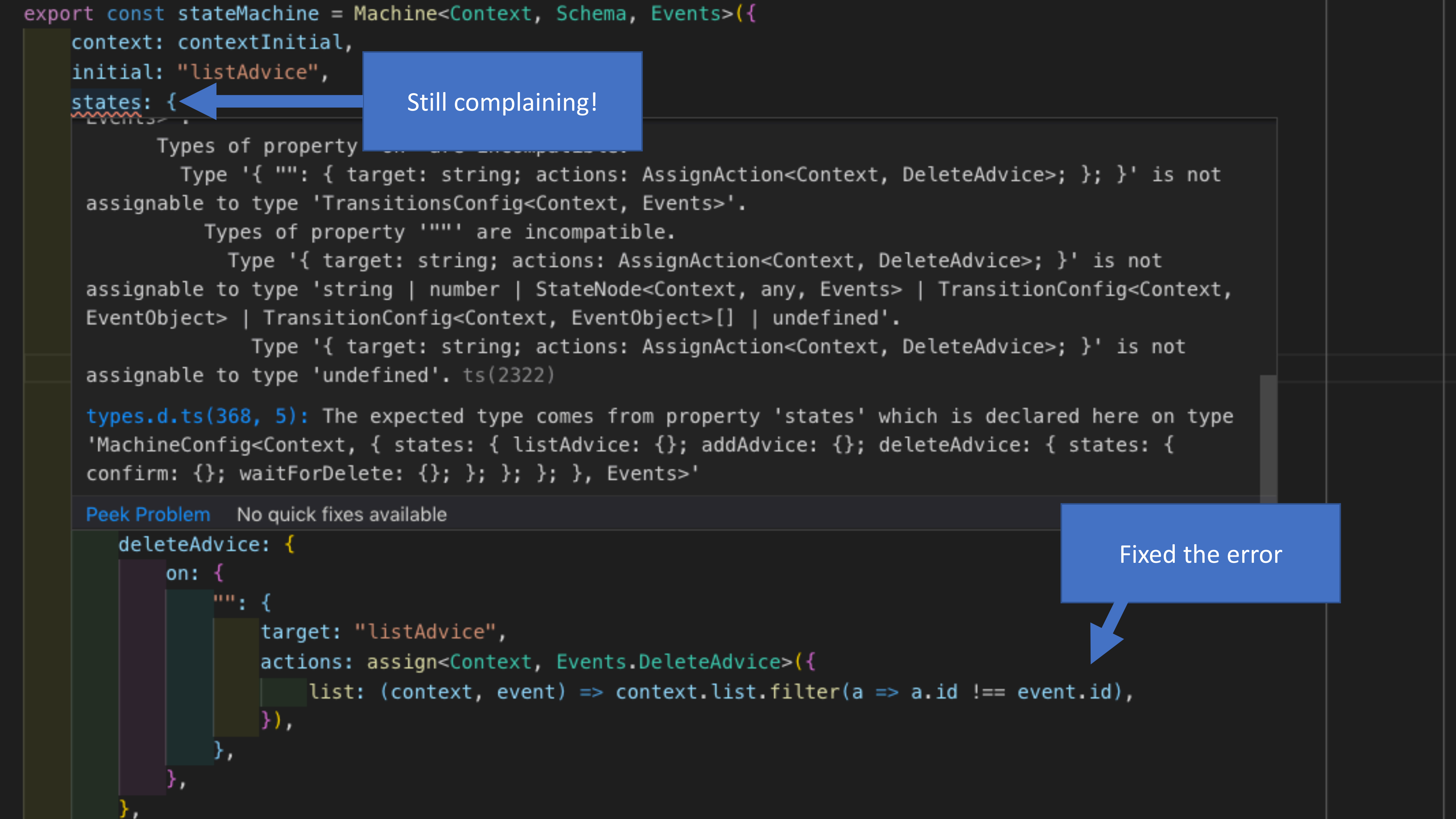


```
export const stateMachine = Machine<Context, Schema, Events>({
    context: contextInitial,
    initial: "listAdvice",
    states: {
        listAdvice: {
            on: {
                ADD_ADVICE: "addAdvice",
                DELETE_ADVICE: "deleteAdvice",
            },
        },
        addAdvice: {
            on: {
                SAVE_ADVICE: {
                    target: "listAdvice",
                    actions: assign({ list: (context, event) => [...context.list, event.advice] }),
                },
            },
        },
        deleteAdvice: {
            on: {
                "": {
                    target: "listAdvice",
                    actions: assign({ list: (context, event) => context.list.filter(a => a.id !== event.wrongField) }),
                },
            },$
        },
    },
})
```

A mistake!



```
export const stateMachine = Machine<Context, Schema, Events>({
    context: contextInitial,
    initial: "listAdvice",
    states: {
        listAdvice: {
            on: {
                ADD_ADVICE: "addAdvice",
                DELETE_ADVICE: "deleteAdvice",
            },
        },
        addAdvice: {
            on: {
                SAVE_ADVICE: {
                    target: "listAdvice",
                    actions: assign({ list: (context, event) => [...context.list, event.advice] }),
                },
            },
        },
        deleteAdvice: {
            on: {
                "": {
                    target: "listAdvice",
                    actions: assign<Context, Events.DeleteAdvice>({
                        list: (context, event) => context.list.filter(a => a.id !== event.wrongField),
                    }),
                },
            },
        },
```

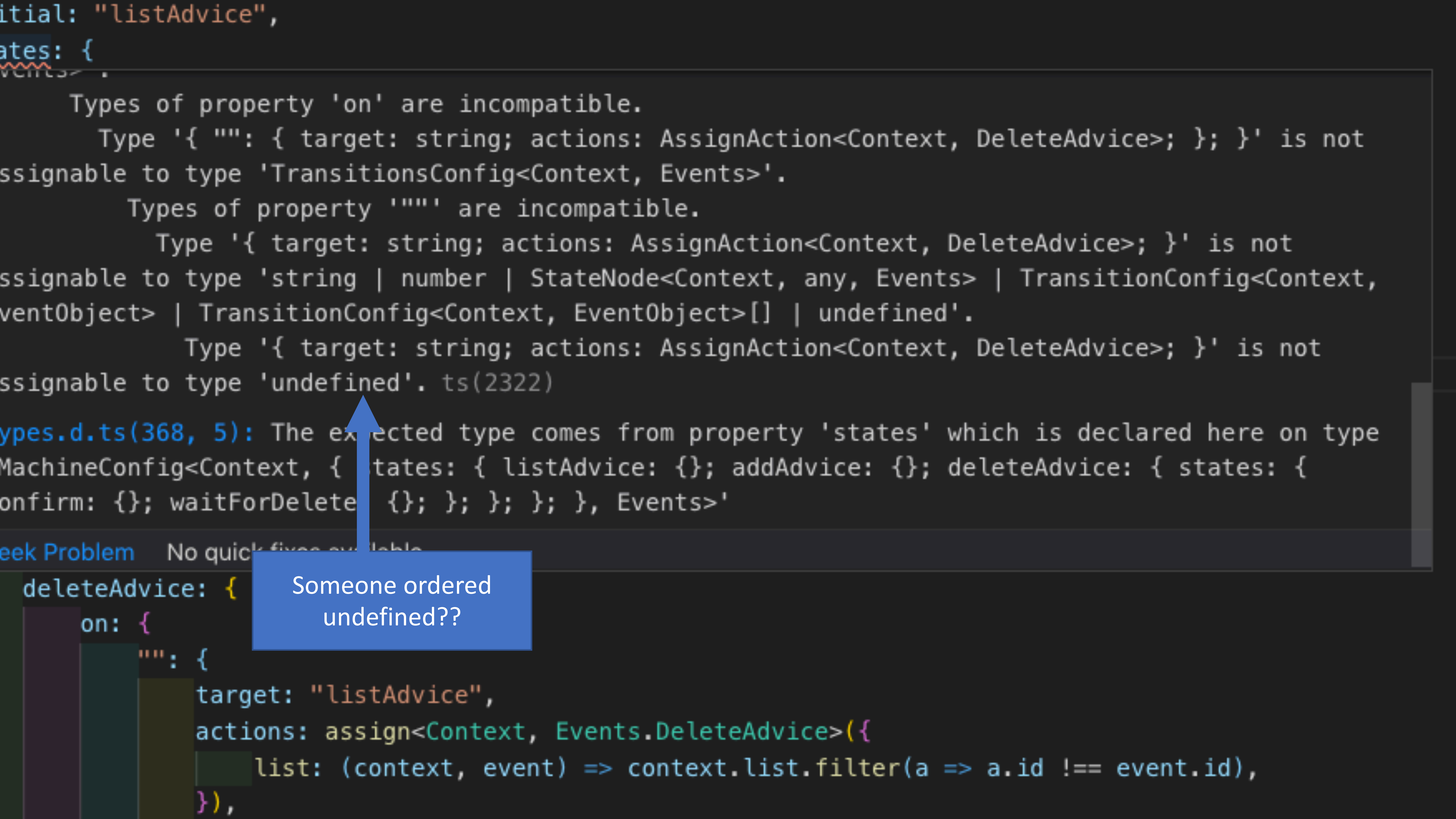


```
export const stateMachine = Machine<Context, Schema, Events>({
  context: contextInitial,
  initial: "listAdvice",
  states: {
```

```
      Types of property
        Type '{ "": { target: string; actions: AssignAction<Context, DeleteAdvice>; }; }' is not
assignable to type 'TransitionsConfig<Context, Events>'.
          Types of property '""' are incompatible.
            Type '{ target: string; actions: AssignAction<Context, DeleteAdvice>; }' is not
assignable to type 'string | number | StateNode<Context, any, Events> | TransitionConfig<Context,
EventObject> | TransitionConfig<Context, EventObject>[] | undefined'.
              Type '{ target: string; actions: AssignAction<Context, DeleteAdvice>; }' is not
assignable to type 'undefined'. ts(2322)

types.d.ts(368, 5): The expected type comes from property 'states' which is declared here on type
'MachineConfig<Context, { states: { listAdvice: {}; addAdvice: {}; deleteAdvice: { states: {
confirm: {}; waitForDelete: {}; }; }; }; }, Events>'
```

Peek Problem    No quick fixes available

```
    deleteAdvice: {
      on: {
        "": {
          target: "listAdvice",
          actions: assign<Context, Events.DeleteAdvice>({
            list: (context, event) => context.list.filter(a => a.id !== event.id),
          }),
        },
      },
    },
```



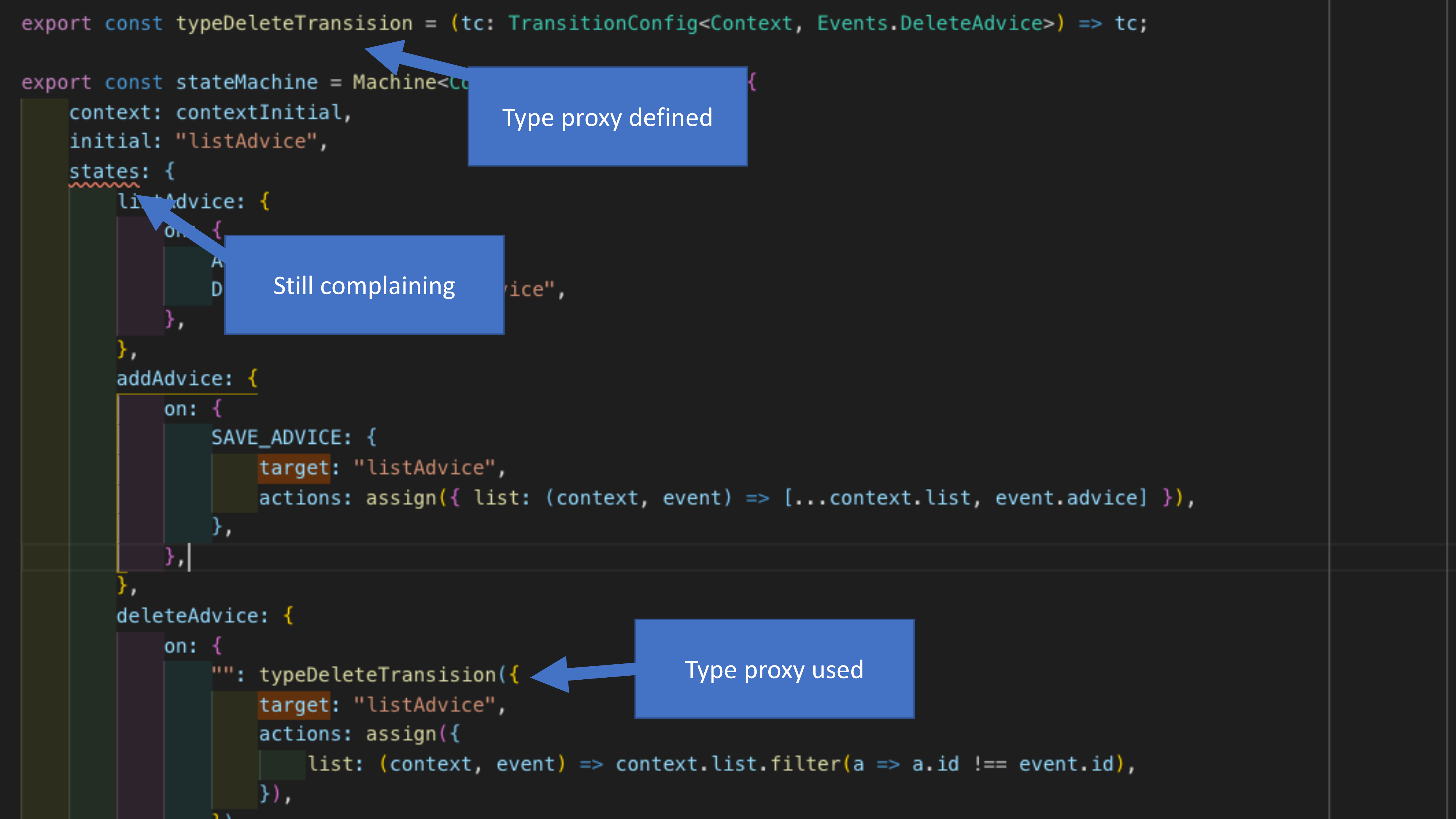
```
itial: "listAdvice",
ates: {
      Types of property 'on' are incompatible.
        Type '{ "": { target: string; actions: AssignAction<Context, DeleteAdvice>; }; }' is not
ssignable to type 'TransitionsConfig<Context, Events>'.
          Types of property '""' are incompatible.
            Type '{ target: string; actions: AssignAction<Context, DeleteAdvice>; }' is not
ssignable to type 'string | number | StateNode<Context, any, Events> | TransitionConfig<Context,
ventObject> | TransitionConfig<Context, EventObject>[] | undefined'.
              Type '{ target: string; actions: AssignAction<Context, DeleteAdvice>; }' is not
ssignable to type 'undefined'. ts(2322)

ypes.d.ts(368, 5): The expected type comes from property 'states' which is declared here on type
MachineConfig<Context, { states: { listAdvice: {}; addAdvice: {}; deleteAdvice: { states: {
onfirm: {}; waitForDelete: {}; }; }; }; }, Events>'

eek Problem    No quick fixes available

  deleteAdvice: {
      on: {
          "": {
              target: "listAdvice",
              actions: assign<Context, Events.DeleteAdvice>({
                  list: (context, event) => context.list.filter(a => a.id !== event.id),
              }),
```

Someone ordered undefined??

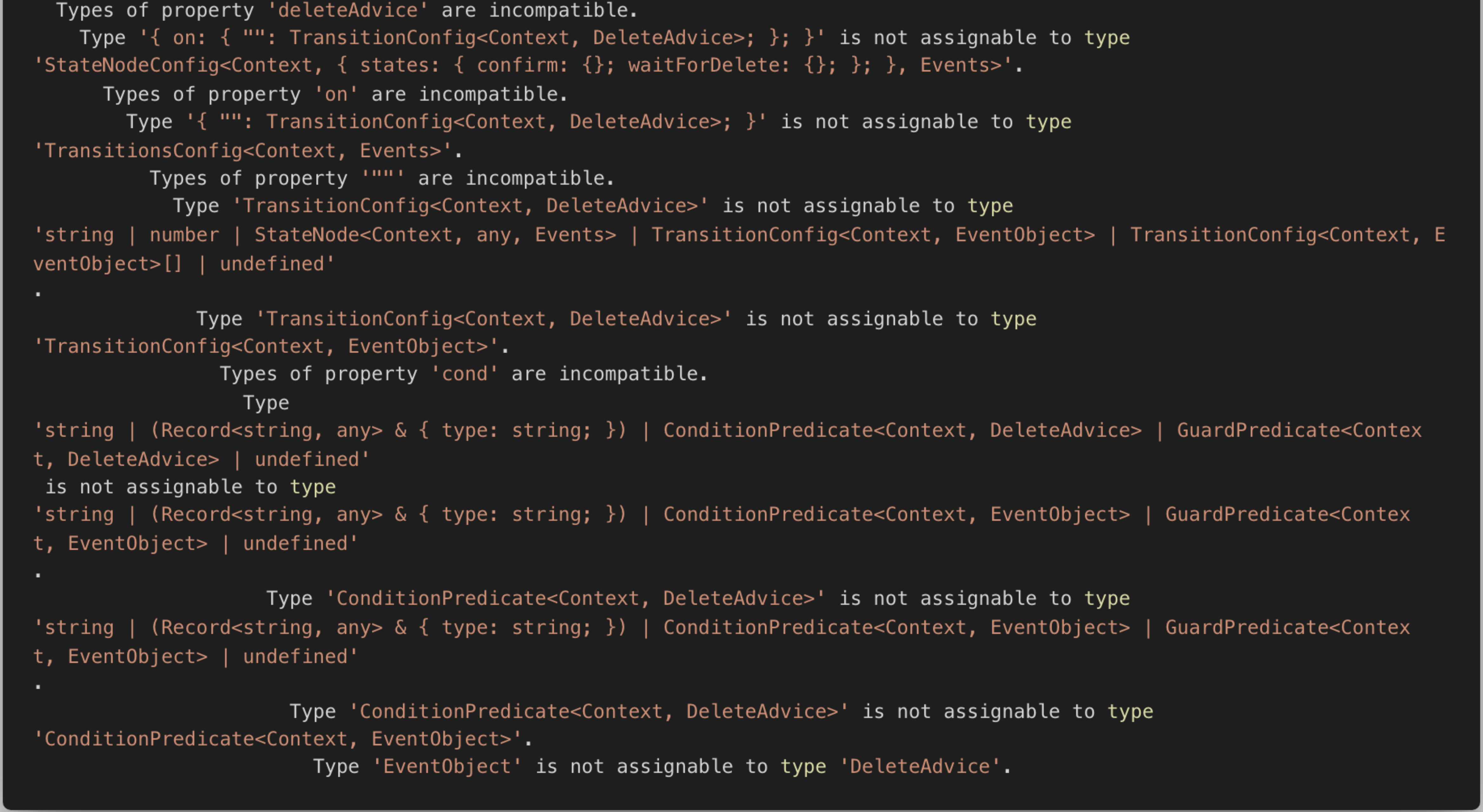
Hours spent on debugging and digging into node_modules/@types/xstate

or…

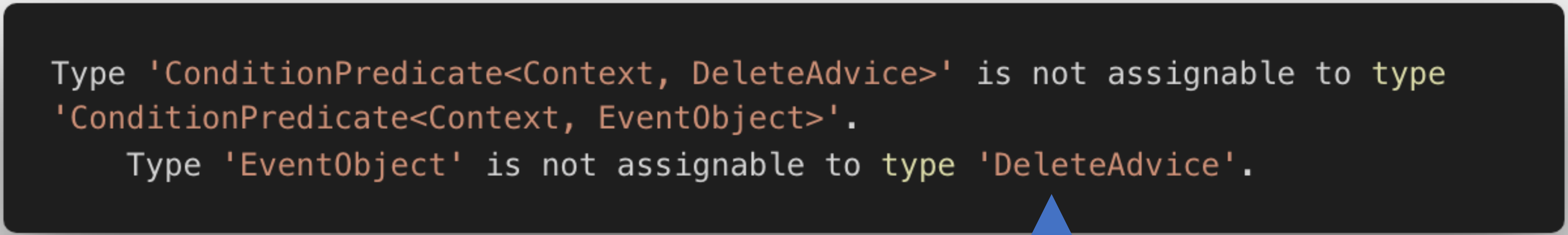
type proxy!



```
export const typeDeleteTransision = (tc: TransitionConfig<Context, Events.DeleteAdvice>) => tc;

export const stateMachine = Machine<Co                                    {
    context: contextInitial,
    initial: "listAdvice",
    states: {
        li  Advice: {
            on  {
                A
                D                        ice",
            },
        },
        addAdvice: {
            on: {
                SAVE_ADVICE: {
                    target: "listAdvice",
                    actions: assign({ list: (context, event) => [...context.list, event.advice] }),
                },
            },
        },
        deleteAdvice: {
            on: {
                "": typeDeleteTransision({
                    target: "listAdvice",
                    actions: assign({
                        list: (context, event) => context.list.filter(a => a.id !== event.id),
                    }),
```

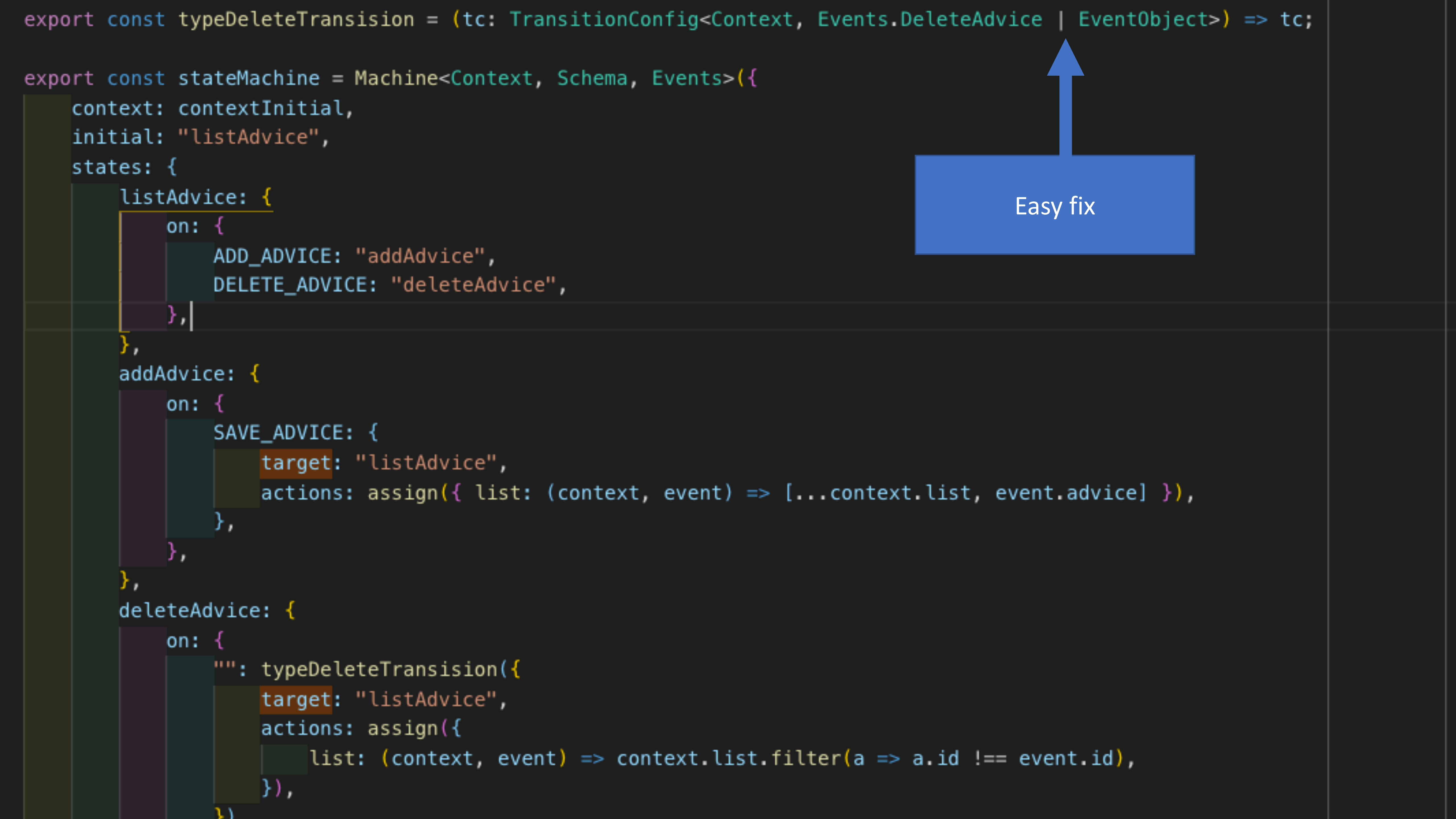


```
 Types of property 'deleteAdvice' are incompatible.
  Type '{ on: { "": TransitionConfig<Context, DeleteAdvice>; }; }' is not assignable to type
'StateNodeConfig<Context, { states: { confirm: {}; waitForDelete: {}; }; }, Events>'.
   Types of property 'on' are incompatible.
    Type '{ "": TransitionConfig<Context, DeleteAdvice>; }' is not assignable to type
'TransitionsConfig<Context, Events>'.
     Types of property '""' are incompatible.
      Type 'TransitionConfig<Context, DeleteAdvice>' is not assignable to type
'string | number | StateNode<Context, any, Events> | TransitionConfig<Context, EventObject> | TransitionConfig<Context, E
ventObject>[] | undefined'
.
       Type 'TransitionConfig<Context, DeleteAdvice>' is not assignable to type
'TransitionConfig<Context, EventObject>'.
        Types of property 'cond' are incompatible.
         Type
'string | (Record<string, any> & { type: string; }) | ConditionPredicate<Context, DeleteAdvice> | GuardPredicate<Contex
t, DeleteAdvice> | undefined'
 is not assignable to type
'string | (Record<string, any> & { type: string; }) | ConditionPredicate<Context, EventObject> | GuardPredicate<Contex
t, EventObject> | undefined'
.
          Type 'ConditionPredicate<Context, DeleteAdvice>' is not assignable to type
'string | (Record<string, any> & { type: string; }) | ConditionPredicate<Context, EventObject> | GuardPredicate<Contex
t, EventObject> | undefined'
.
           Type 'ConditionPredicate<Context, DeleteAdvice>' is not assignable to type
'ConditionPredicate<Context, EventObject>'.
            Type 'EventObject' is not assignable to type 'DeleteAdvice'.
```

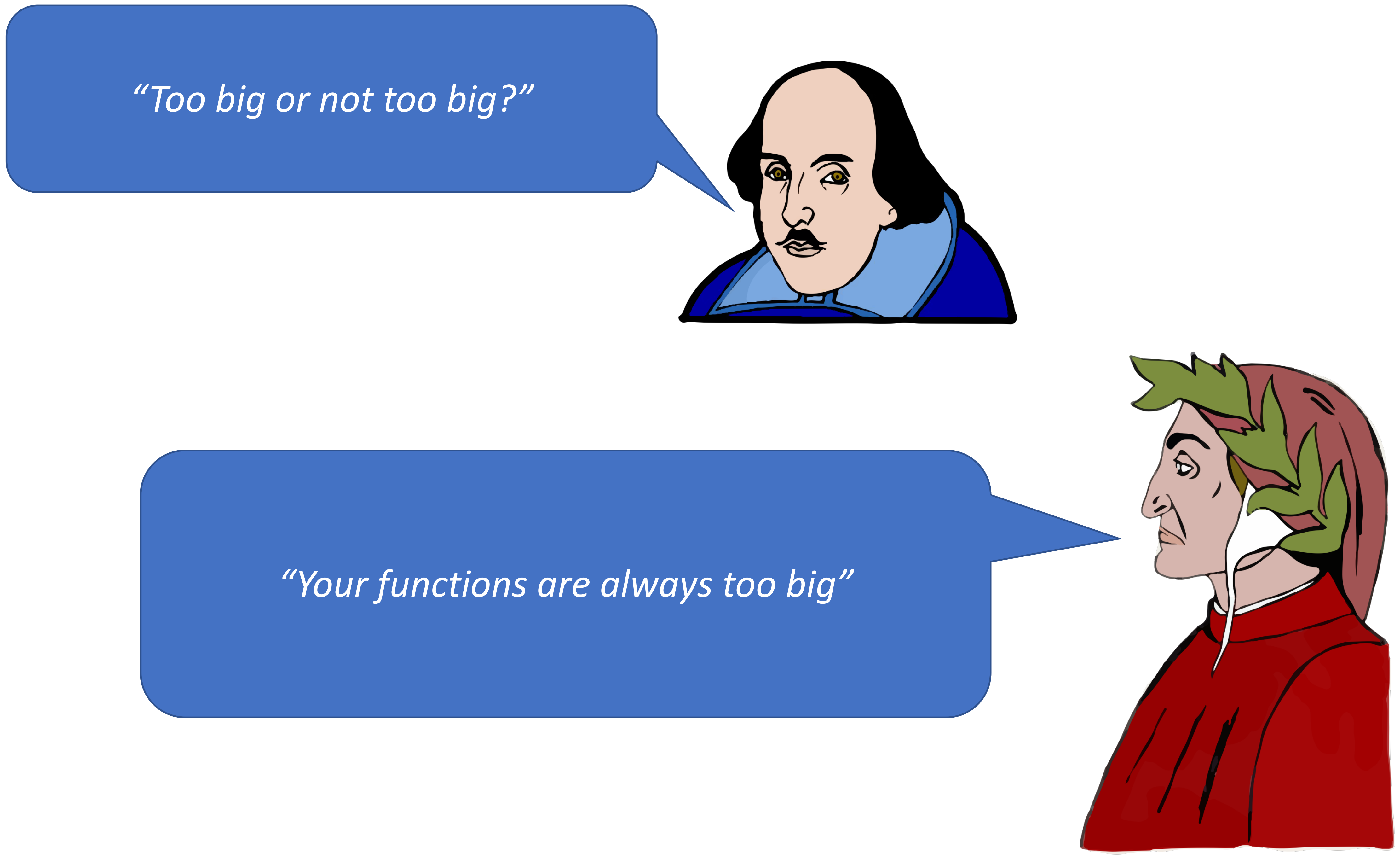


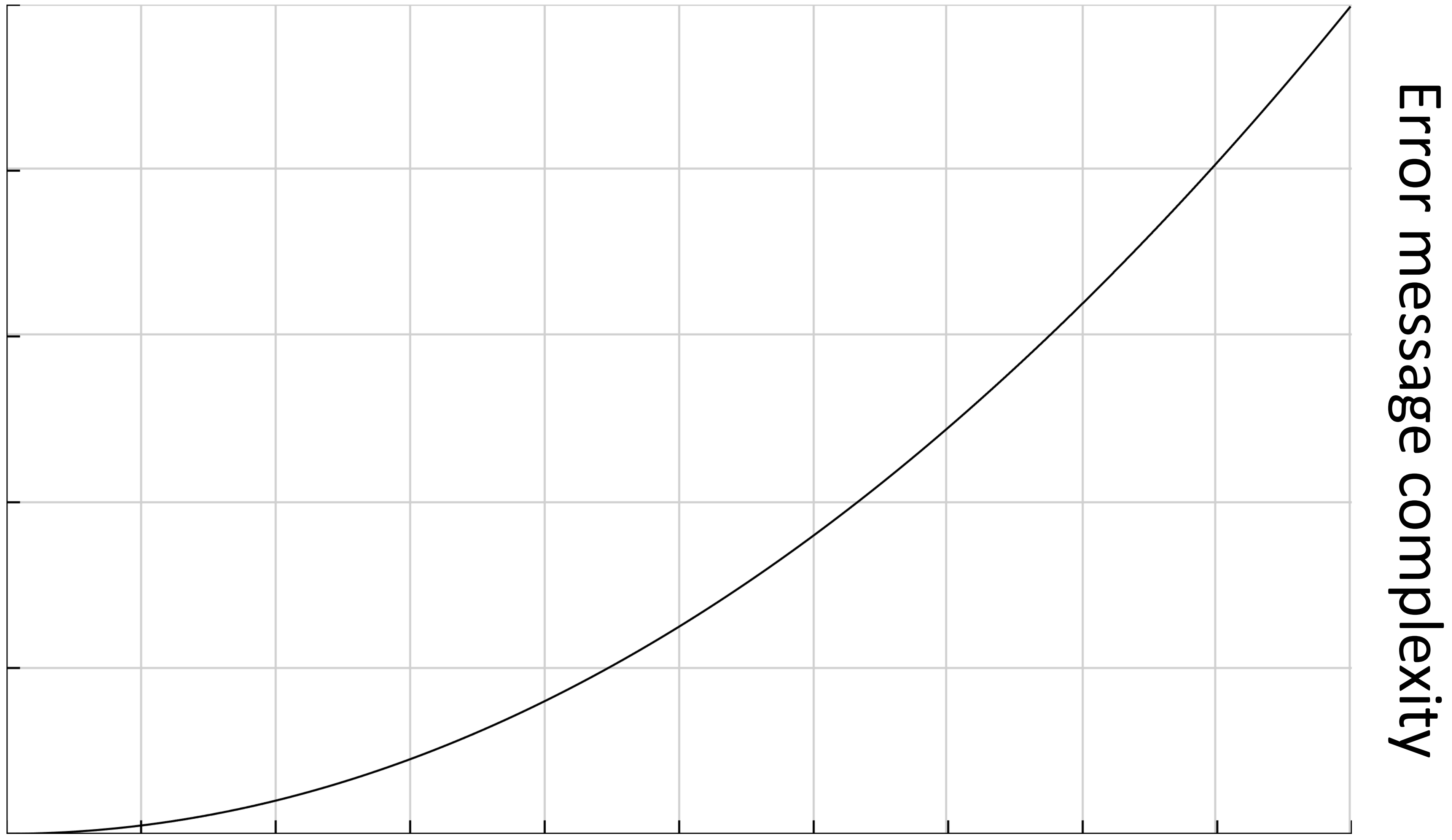
```
Type 'ConditionPredicate<Context, DeleteAdvice>' is not assignable to type
'ConditionPredicate<Context, EventObject>'.
    Type 'EventObject' is not assignable to type 'DeleteAdvice'.
```

Aha!

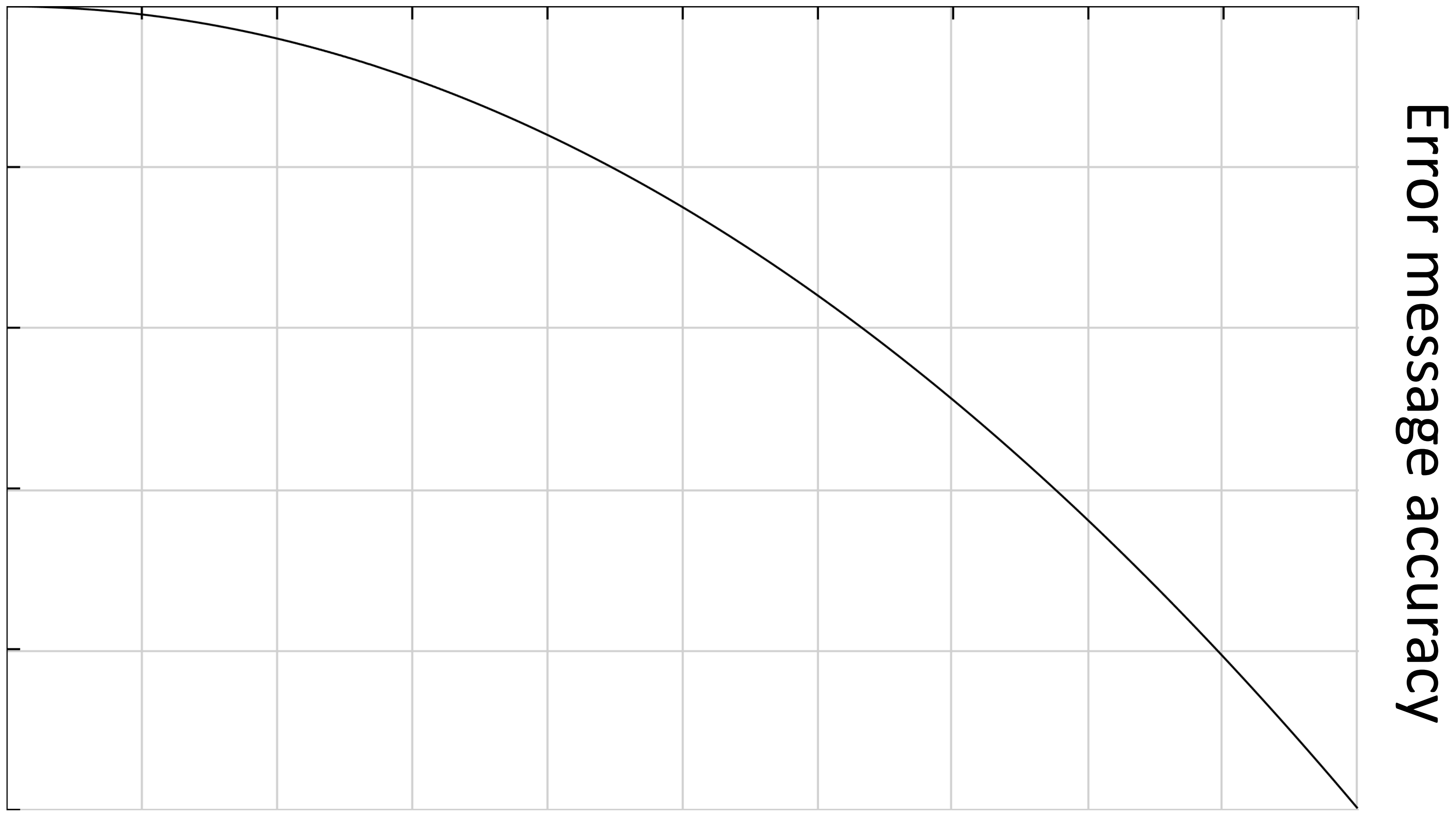


```
export const typeDeleteTransision = (tc: TransitionConfig<Context, Events.DeleteAdvice | EventObject>) => tc;

export const stateMachine = Machine<Context, Schema, Events>({
    context: contextInitial,
    initial: "listAdvice",
    states: {
        listAdvice: {
            on: {
                ADD_ADVICE: "addAdvice",
                DELETE_ADVICE: "deleteAdvice",
            },
        },
        addAdvice: {
            on: {
                SAVE_ADVICE: {
                    target: "listAdvice",
                    actions: assign({ list: (context, event) => [...context.list, event.advice] }),
                },
            },
        },
        deleteAdvice: {
            on: {
                "": typeDeleteTransision({
                    target: "listAdvice",
                    actions: assign({
                        list: (context, event) => context.list.filter(a => a.id !== event.id),
                    }),
                })
```


*“Too big or not too big?”*
*“Your functions are always too big”*



Distance between type definition and check



Distance between type definition and check

Fight… False type security …

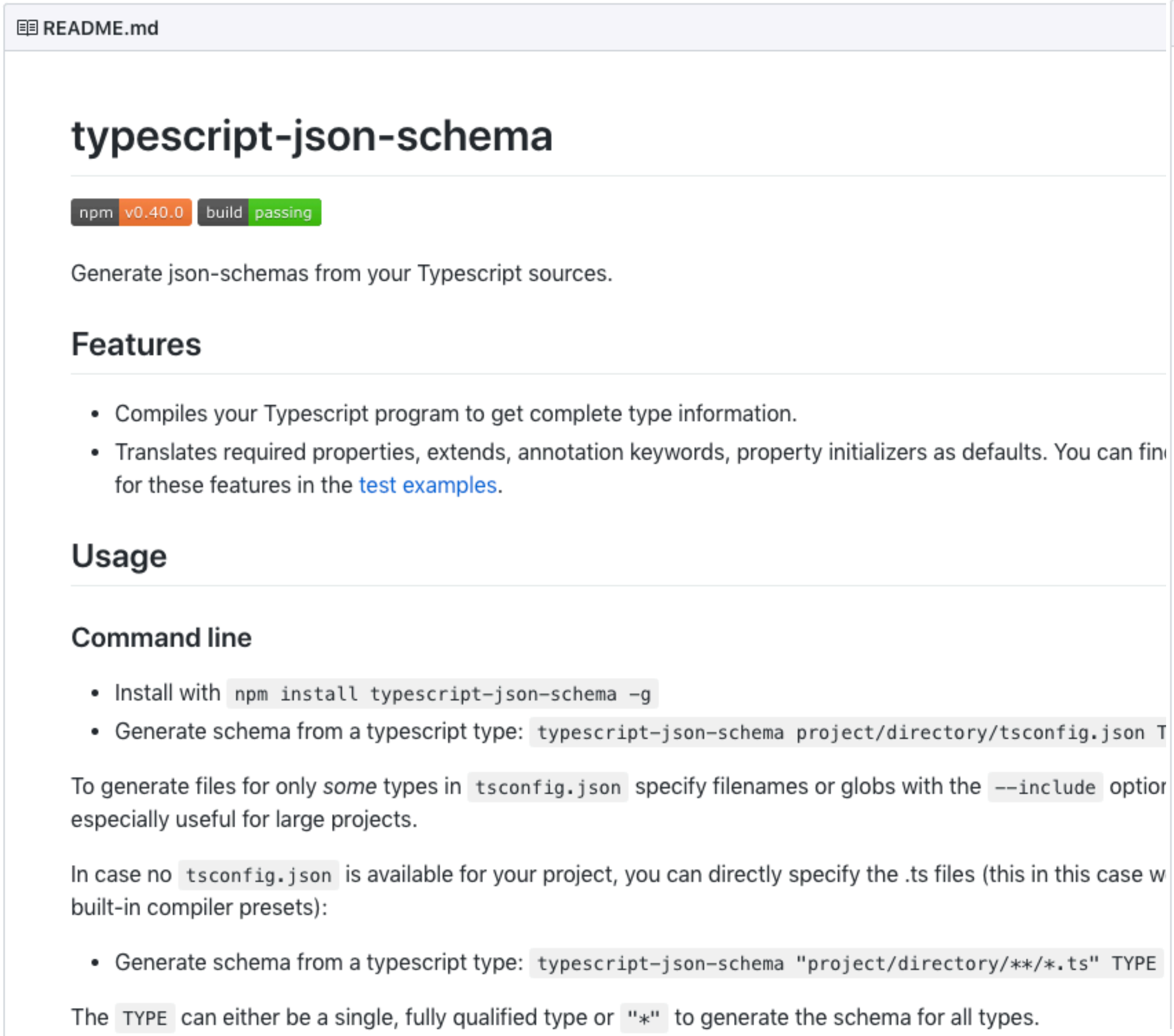
## Typescript on input data boundary

## Typescript = compile-time only

Typescript = ~~compile-time only~~

compile + runtime

# Typescript = ~~compile-time only~~
compile + runtime



README.md

## typescript-json-schema

npm v0.40.0 build passing

Generate json-schemas from your Typescript sources.

### Features

- Compiles your Typescript program to get complete type information.
- Translates required properties, extends, annotation keywords, property initializers as defaults. You can fin
  for these features in the test examples.

### Usage

#### Command line

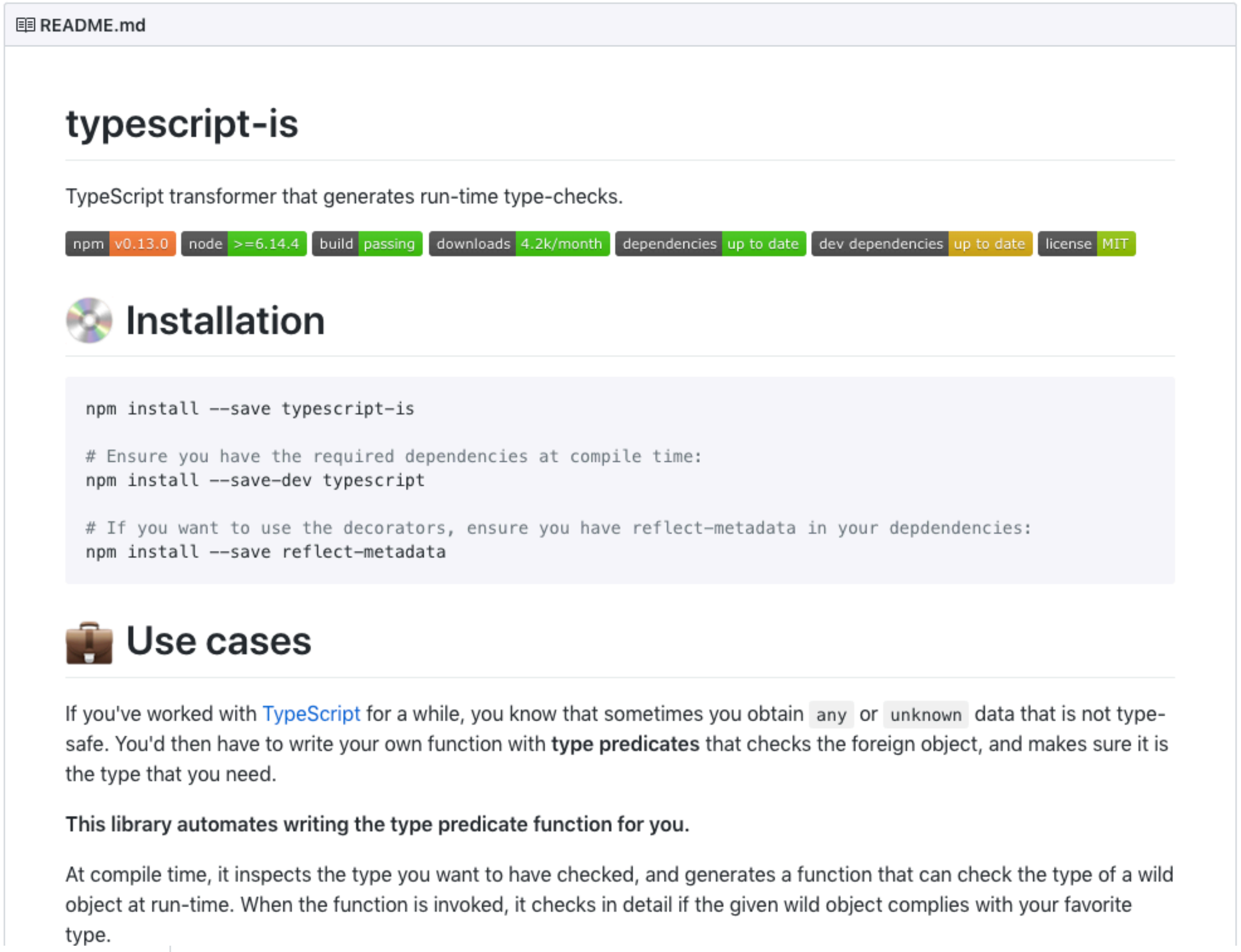
- Install with `npm install typescript-json-schema -g`
- Generate schema from a typescript type: `typescript-json-schema project/directory/tsconfig.json T`

To generate files for only *some* types in `tsconfig.json` specify filenames or globs with the `--include` optior
especially useful for large projects.

In case no `tsconfig.json` is available for your project, you can directly specify the .ts files (this in this case w
built-in compiler presets):

- Generate schema from a typescript type: `typescript-json-schema "project/directory/**/*.ts" TYPE`

The `TYPE` can either be a single, fully qualified type or `"*"` to generate the schema for all types.



README.md

## typescript-is

TypeScript transformer that generates run-time type-checks.

npm v0.13.0 node >=6.14.4 build passing downloads 4.2k/month dependencies up to date dev dependencies up to date license MIT

### Installation

```
npm install --save typescript-is

# Ensure you have the required dependencies at compile time:
npm install --save-dev typescript

# If you want to use the decorators, ensure you have reflect-metadata in your depdendencies:
npm install --save reflect-metadata
```

### Use cases

If you've worked with TypeScript for a while, you know that sometimes you obtain `any` or `unknown` data that is not type-safe. You'd then have to write your own function with **type predicates** that checks the foreign object, and makes sure it is the type that you need.

**This library automates writing the type predicate function for you.**

At compile time, it inspects the type you want to have checked, and generates a function that can check the type of a wild object at run-time. When the function is invoked, it checks in detail if the given wild object complies with your favorite type.

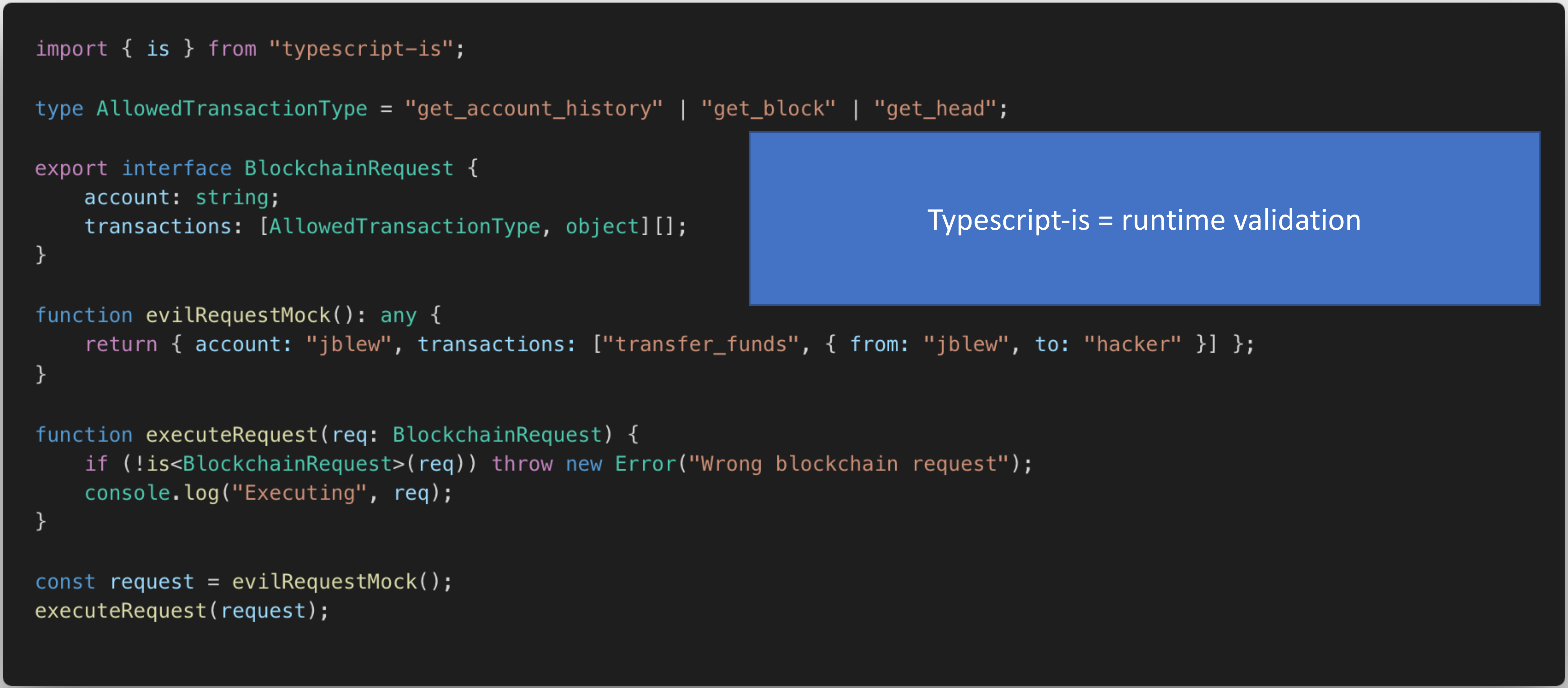
Modelling JSON-schema with Typescript interfaces

```
export type wise_operation = wise_send_voteorder_operation | wise_set_rules_operation;

export type wise_send_voteorder_operation = ["v2:send_voteorder", wise_send_voteorder];
export type wise_set_rules_operation = ["v2:set_rules", wise_set_rules];

export interface wise_set_rules {
    voter: string;
    description?: string;
    rulesets: [string, wise_rule[]][];
}

export interface wise_send_voteorder {
    delegator: string;
    ruleset: string;
    permlink: string;
    author: string;

    /**
     * Vote / flag weight
     *
     * @minimum -10000
     * @maximum 10000
     * @TJS-type integer
     */
    weight: number;
}
```

```
$ typescript-json-schema --out "schema.json" tsconfig.json "wise_operation"
```

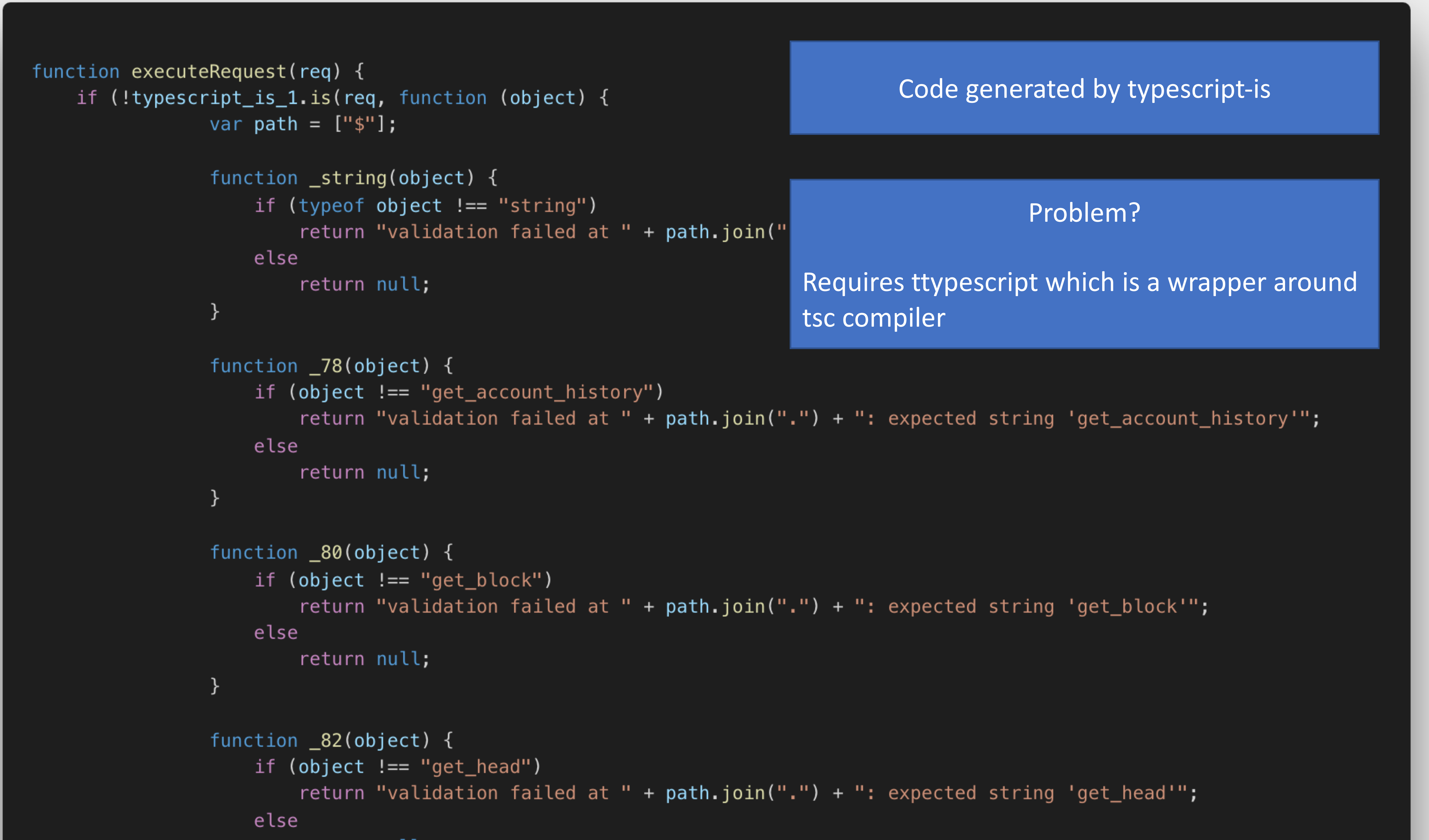
```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "anyOf": [{
      "additionalItems": {
        "anyOf": [{
            "enum": [
              "v2:send_voteorder"
            ],
            "type": "string"
          },
          {
            "$ref": "#/definitions/wise_send_voteorder"
          }
        ]
      },
      "items": [{
          "enum": [
            "v2:send_voteorder"
          ],
          "type": "string"
        },
        {
          "$ref": "#/definitions/wise_send_voteorder"
        }
      ],
      "minItems": 2,
      "type": "array"
    },
```

Generated JSON-schema can be used with any validator (available for almost all programming langs)



```typescript
import { is } from "typescript-is";

type AllowedTransactionType = "get_account_history" | "get_block" | "get_head";

export interface BlockchainRequest {
    account: string;
    transactions: [AllowedTransactionType, object][];
}

function evilRequestMock(): any {
    return { account: "jblew", transactions: ["transfer_funds", { from: "jblew", to: "hacker" }] };
}

function executeRequest(req: BlockchainRequest) {
    if (!is<BlockchainRequest>(req)) throw new Error("Wrong blockchain request");
    console.log("Executing", req);
}

const request = evilRequestMock();
executeRequest(request);
```

Typescript-is = runtime validation



```
function executeRequest(req) {
    if (!typescript_is_1.is(req, function (object) {
                var path = ["$"];

                function _string(object) {
                    if (typeof object !== "string")
                        return "validation failed at " + path.join("
                    else
                        return null;
                }

                function _78(object) {
                    if (object !== "get_account_history")
                        return "validation failed at " + path.join(".") + ": expected string 'get_account_history'";
                    else
                        return null;
                }

                function _80(object) {
                    if (object !== "get_block")
                        return "validation failed at " + path.join(".") + ": expected string 'get_block'";
                    else
                        return null;
                }

                function _82(object) {
                    if (object !== "get_head")
                        return "validation failed at " + path.join(".") + ": expected string 'get_head'";
                    else
```

Code generated by typescript-is

Problem?

Requires ttypescript which is a wrapper around tsc compiler

## Typescript-json-schema

- Great interoperability
- Use by external validators

- Additional build step
- Slow
- Big footprint of validation libraries (ajv is 300kB)

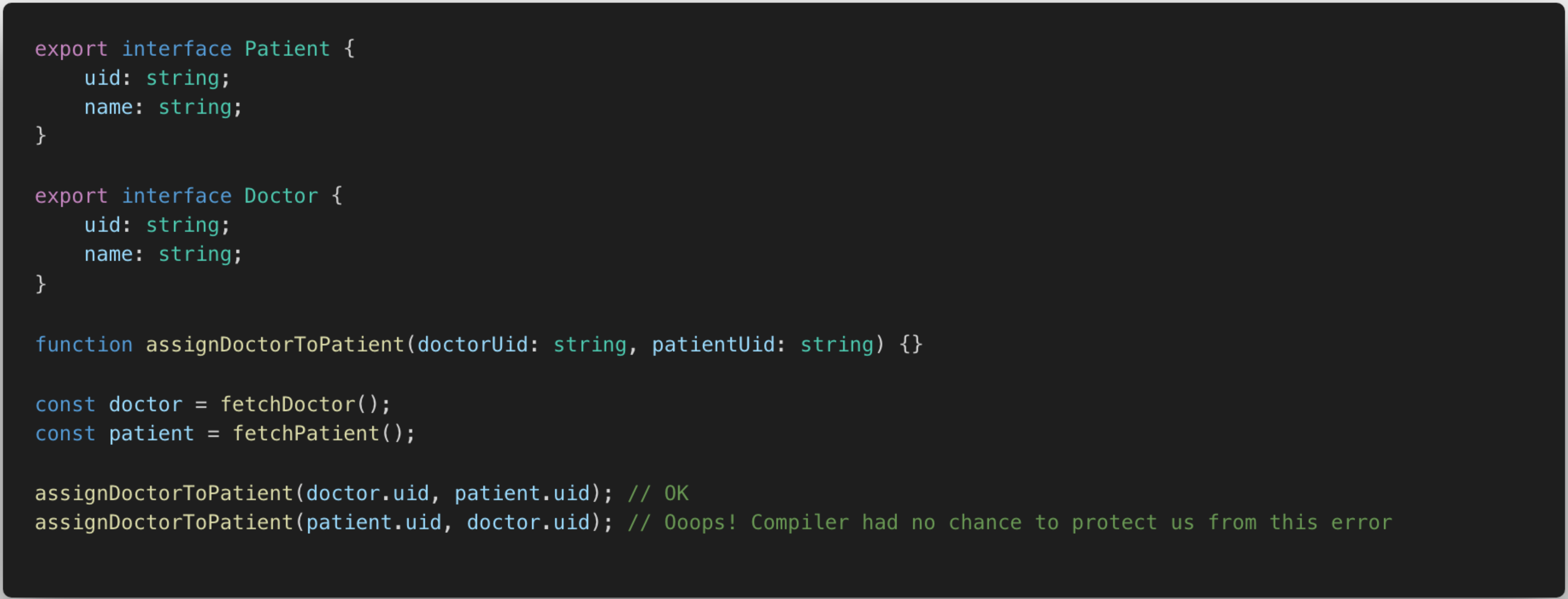
## Typescript-is

- Fast
- Small footprint (generated code + typescript-is = 1.8kB)

- Only inside ttypescript enabled projects

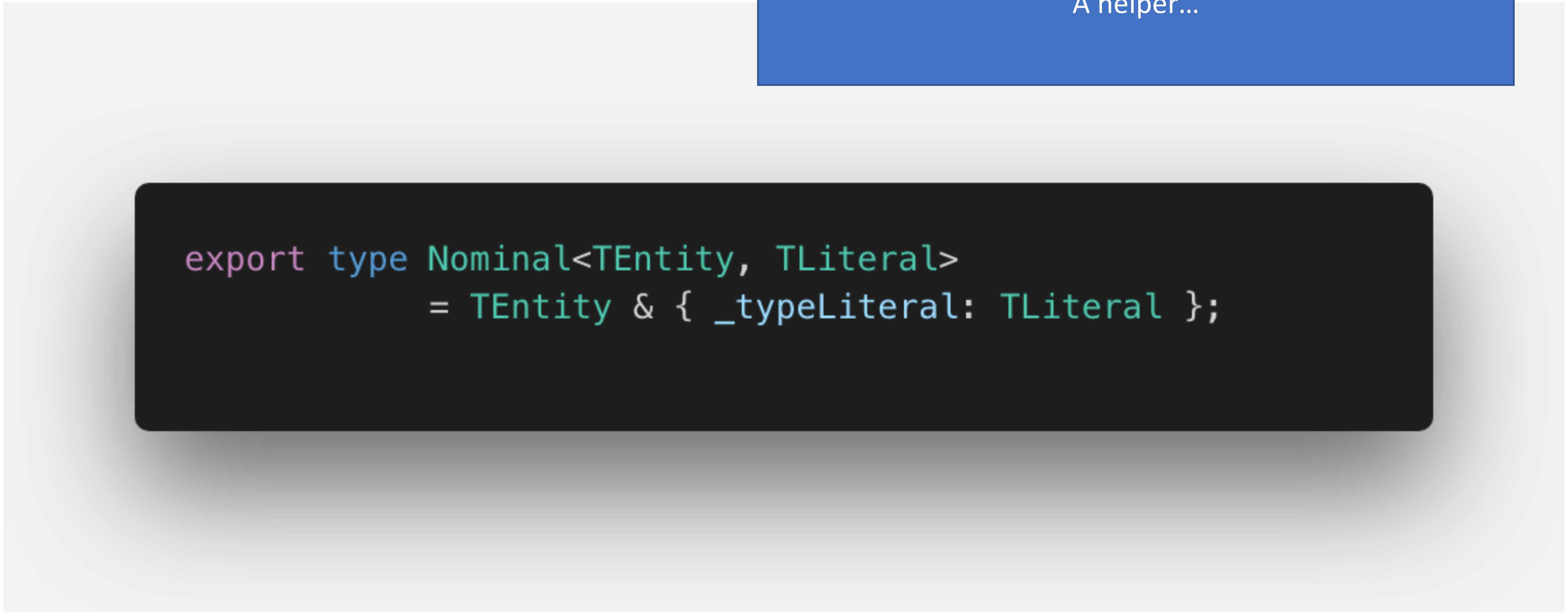
Fight false type security with…

## Nominal typing

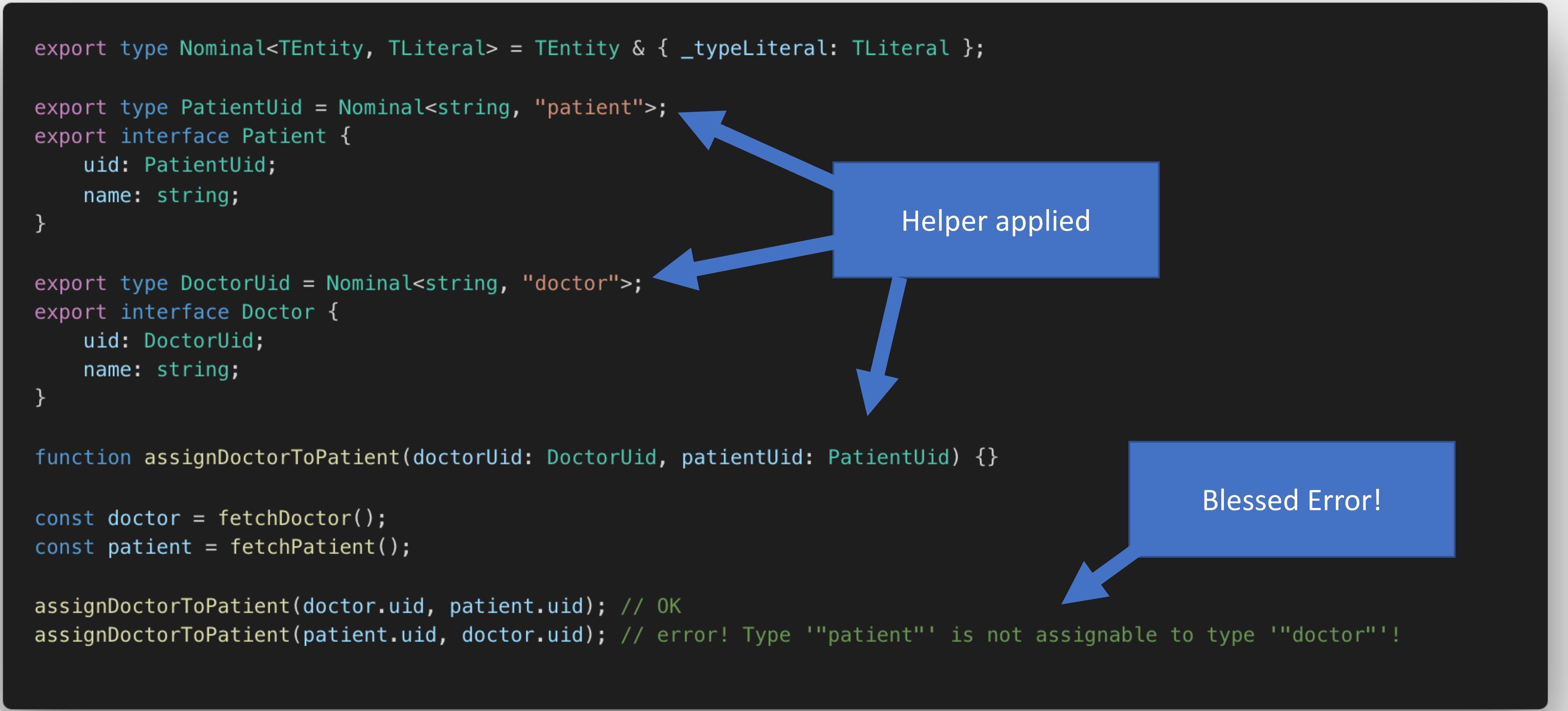


```
export interface Patient {
    uid: string;
    name: string;
}

export interface Doctor {
    uid: string;
    name: string;
}

function assignDoctorToPatient(doctorUid: string, patientUid: string) {}

const doctor = fetchDoctor();
const patient = fetchPatient();

assignDoctorToPatient(doctor.uid, patient.uid); // OK
assignDoctorToPatient(patient.uid, doctor.uid); // Ooops! Compiler had no chance to protect us from this error
```

A helper…



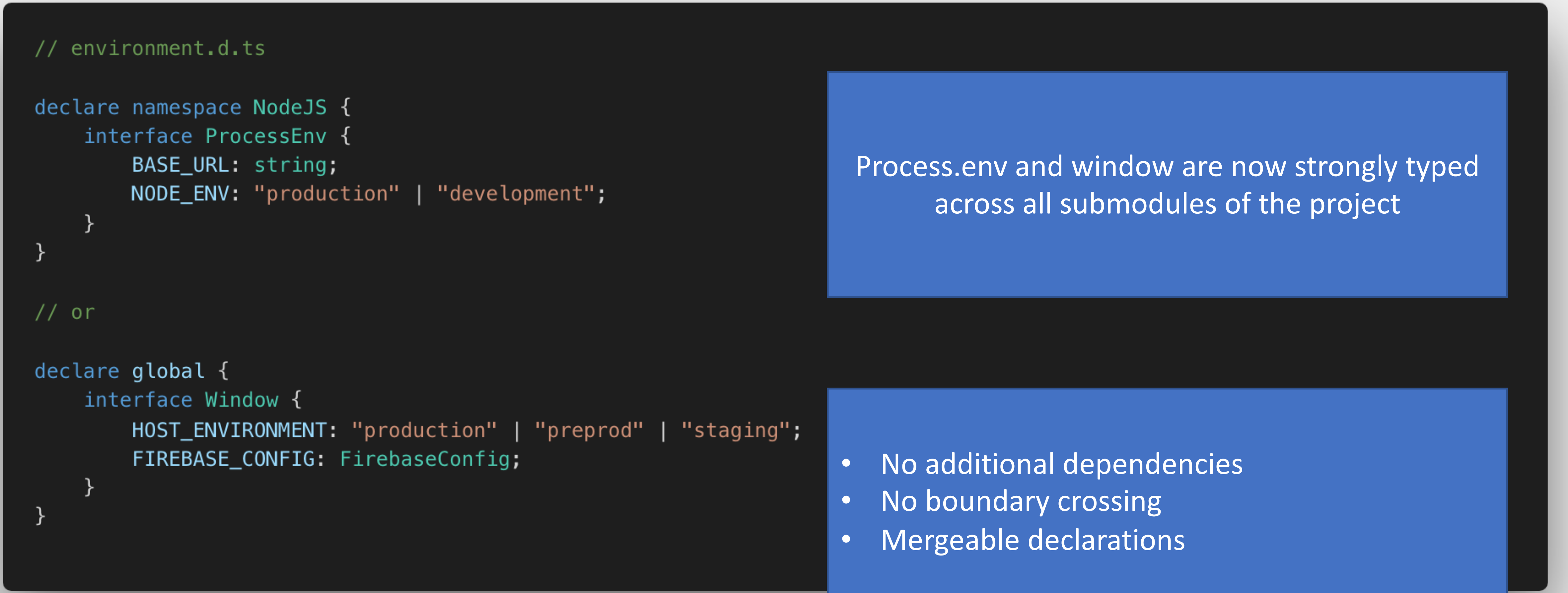
```
export type Nominal<TEntity, TLiteral>
            = TEntity & { _typeLiteral: TLiteral };
```



```
export type Nominal<TEntity, TLiteral> = TEntity & { _typeLiteral: TLiteral };

export type PatientUid = Nominal<string, "patient">;
export interface Patient {
    uid: PatientUid;
    name: string;
}

export type DoctorUid = Nominal<string, "doctor">;
export interface Doctor {
    uid: DoctorUid;
    name: string;
}

function assignDoctorToPatient(doctorUid: DoctorUid, patientUid: PatientUid) {}

const doctor = fetchDoctor();
const patient = fetchPatient();

assignDoctorToPatient(doctor.uid, patient.uid); // OK
assignDoctorToPatient(patient.uid, doctor.uid); // error! Type '"patient"' is not assignable to type '"doctor"'!
```

Fight false type security with…

## Typing globals in ambient space



```
// environment.d.ts

declare namespace NodeJS {
    interface ProcessEnv {
        BASE_URL: string;
        NODE_ENV: "production" | "development";
    }
}

// or

declare global {
    interface Window {
        HOST_ENVIRONMENT: "production" | "preprod" | "staging";
        FIREBASE_CONFIG: FirebaseConfig;
    }
}
```

Process.env and window are now strongly typed across all submodules of the project

- No additional dependencies
- No boundary crossing
- Mergeable declarations

Fight…

## Verbosity

Fight verbosity

## The problem of enums

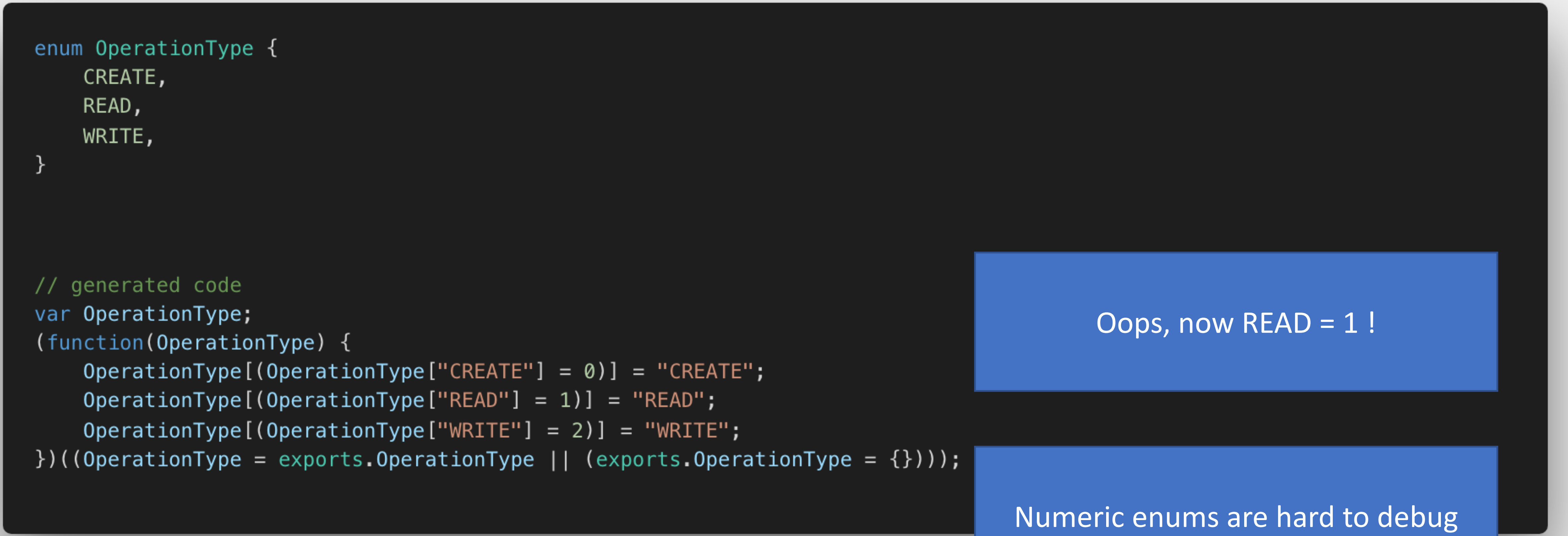
```
enum OperationType {
    READ,
    WRITE,
}
```

Elegant

```
// generated code
var OperationType;
(function(OperationType) {
    OperationType[(OperationType["READ"] = 0)] = "READ";
    OperationType[(OperationType["WRITE"] = 1)] = "WRITE";
})((OperationType = exports.OperationType || (exports.OperationType = {})));
```

- Much boilerplate
- Impact on bundle size

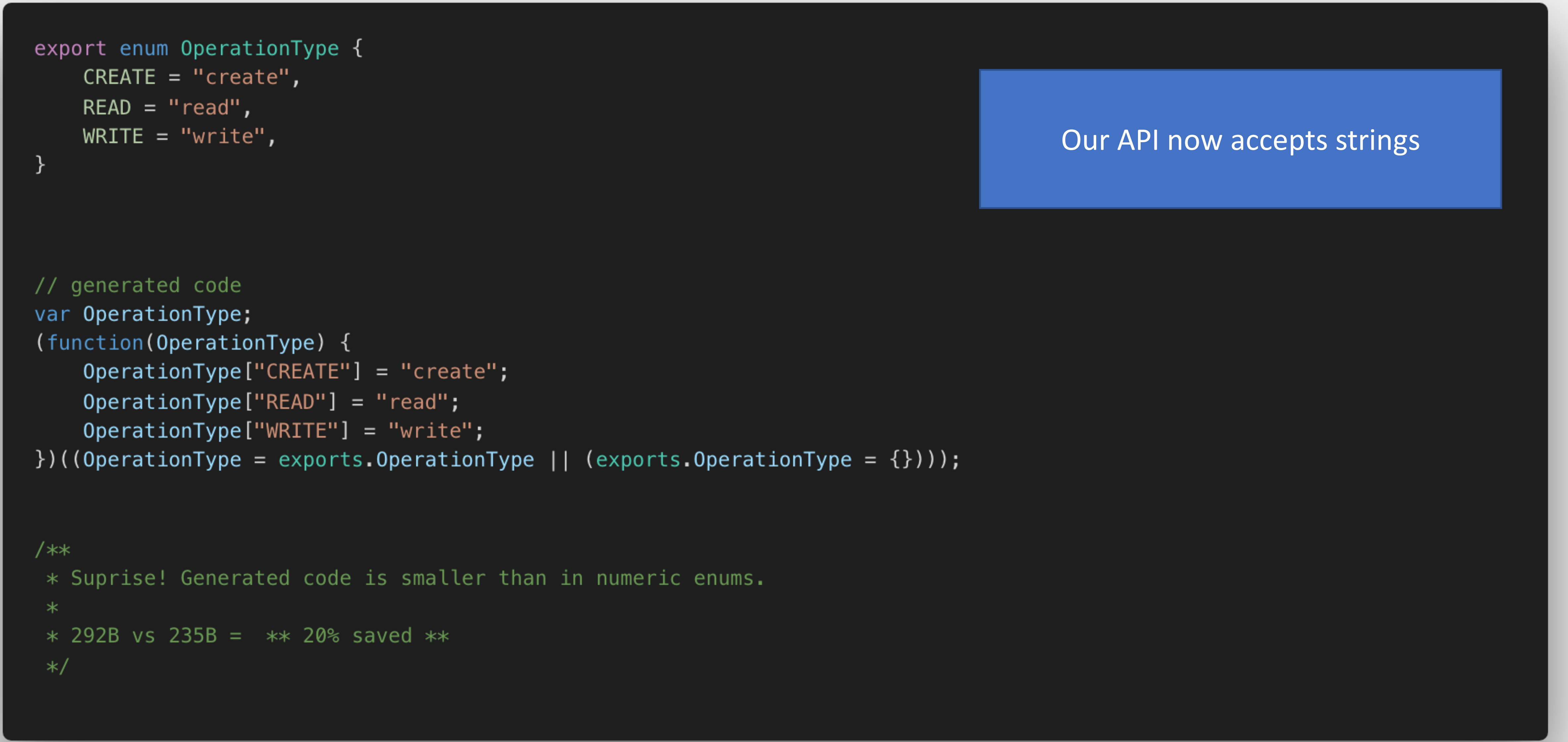
```
// effective value of OperationType object
var OperationType = { "0": "READ", "1": "WRITE", READ: 0, WRITE: 1 };
```



```
enum OperationType {
    CREATE,
    READ,
    WRITE,
}



// generated code
var OperationType;
(function(OperationType) {
    OperationType[(OperationType["CREATE"] = 0)] = "CREATE";
    OperationType[(OperationType["READ"] = 1)] = "READ";
    OperationType[(OperationType["WRITE"] = 2)] = "WRITE";
})((OperationType = exports.OperationType || (exports.OperationType = {})));
```

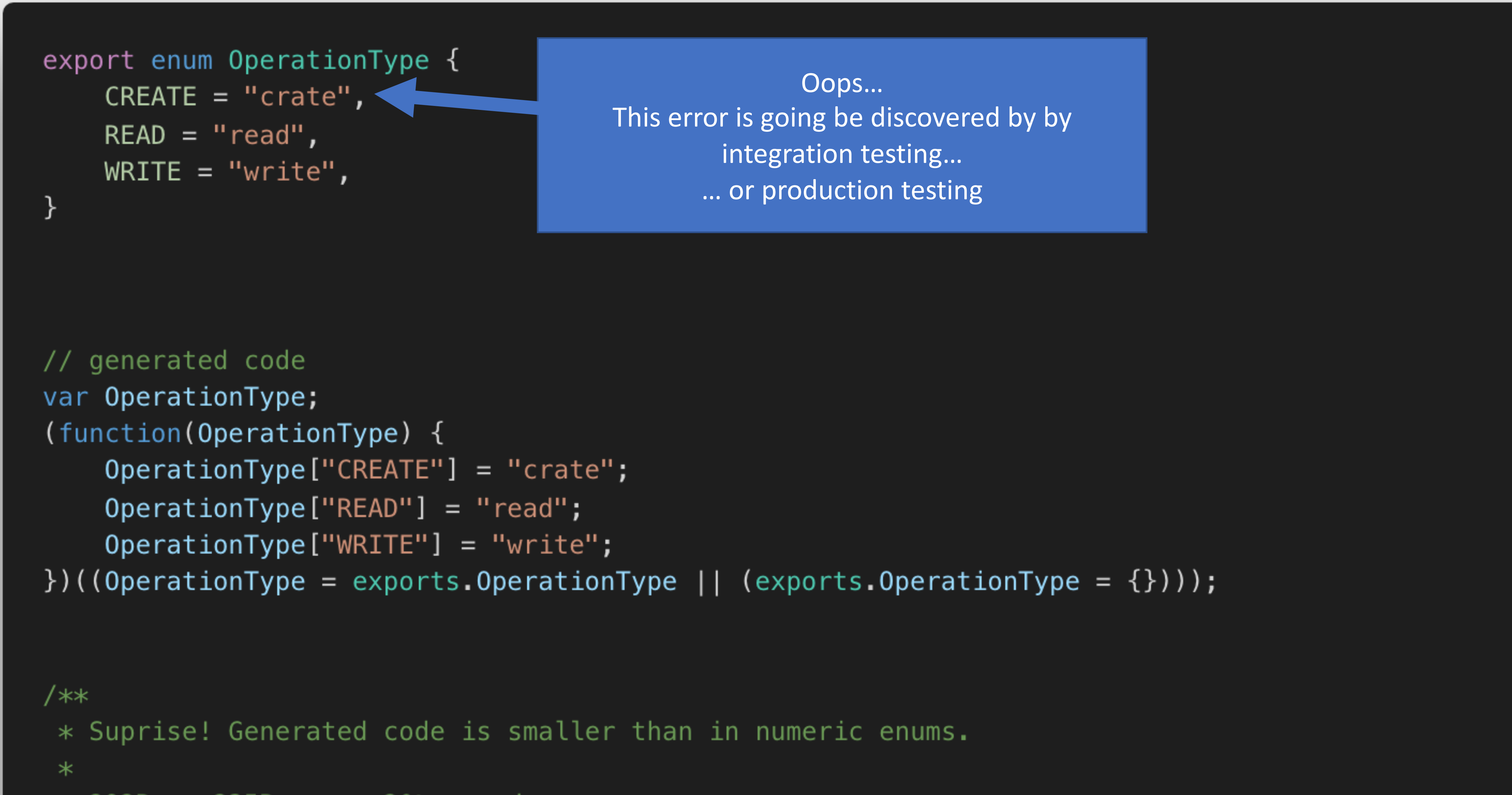
Oops, now READ = 1 !

Numeric enums are hard to debug



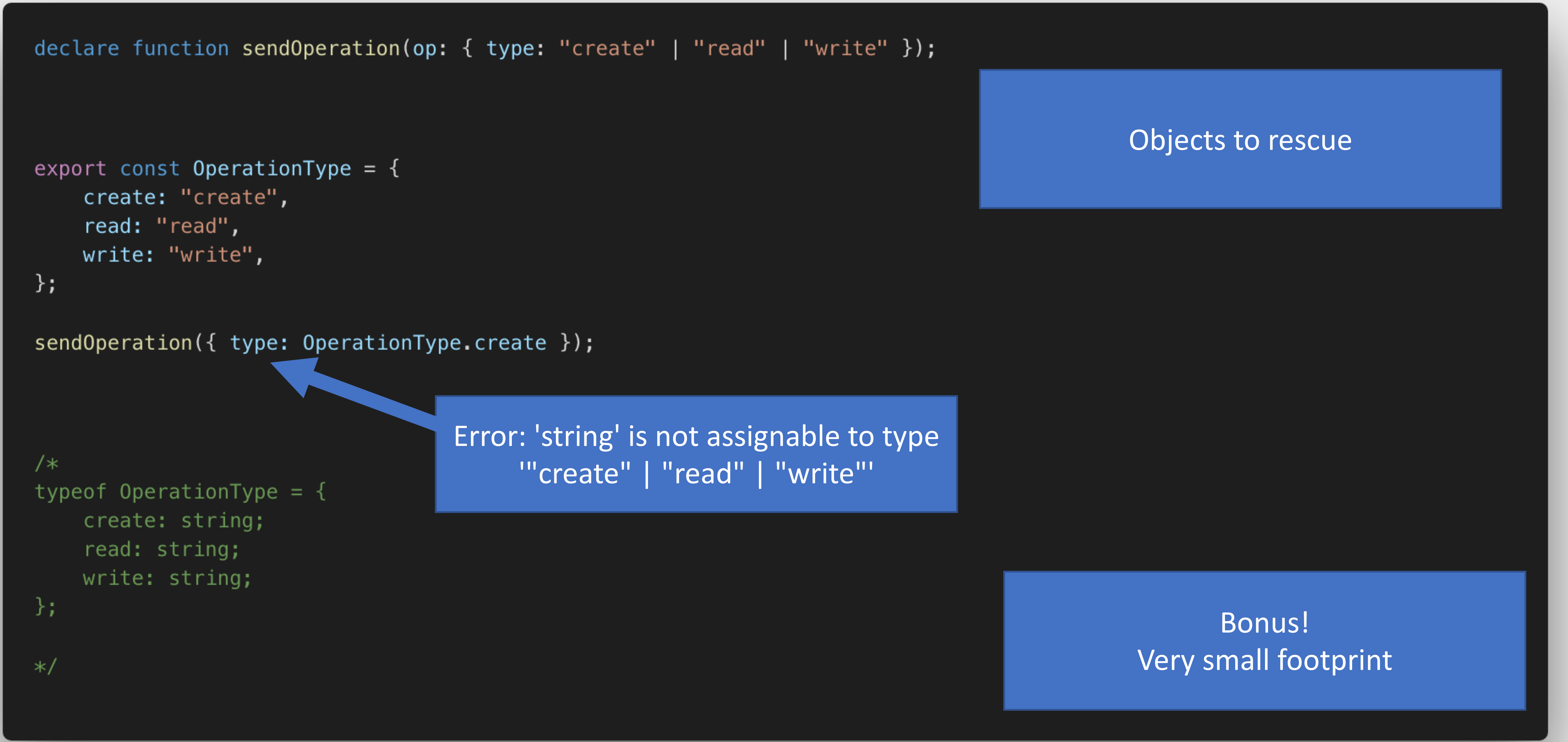
```
export enum OperationType {
    CREATE = "create",
    READ = "read",
    WRITE = "write",
}



// generated code
var OperationType;
(function(OperationType) {
    OperationType["CREATE"] = "create";
    OperationType["READ"] = "read";
    OperationType["WRITE"] = "write";
})((OperationType = exports.OperationType || (exports.OperationType = {})));


/**
 * Suprise! Generated code is smaller than in numeric enums.
 *
 * 292B vs 235B =  ** 20% saved **
 */
```

Our API now accepts strings



```
export enum OperationType {
    CREATE = "crate",
    READ = "read",
    WRITE = "write",
}



// generated code
var OperationType;
(function(OperationType) {
    OperationType["CREATE"] = "crate";
    OperationType["READ"] = "read";
    OperationType["WRITE"] = "write";
})((OperationType = exports.OperationType || (exports.OperationType = {})));


/**
 * Suprise! Generated code is smaller than in numeric enums.
 *
```

Oops…
This error is going be discovered by by integration testing…
… or production testing

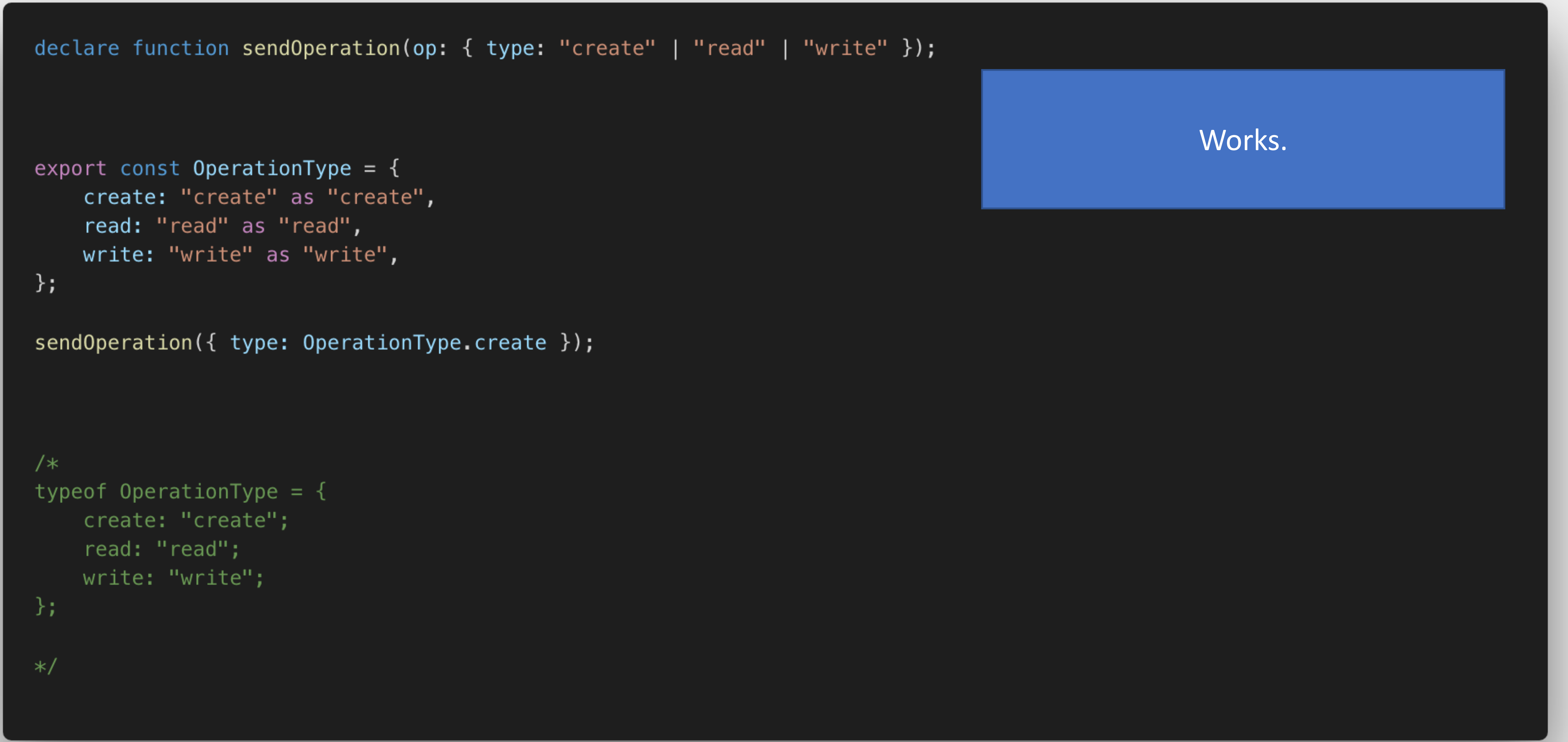


```typescript
declare function sendOperation(op: { type: "create" | "read" | "write" });



export const OperationType = {
    create: "create",
    read: "read",
    write: "write",
};

sendOperation({ type: OperationType.create });



/*
typeof OperationType = {
    create: string;
    read: string;
    write: string;
};

*/
```

Objects to rescue

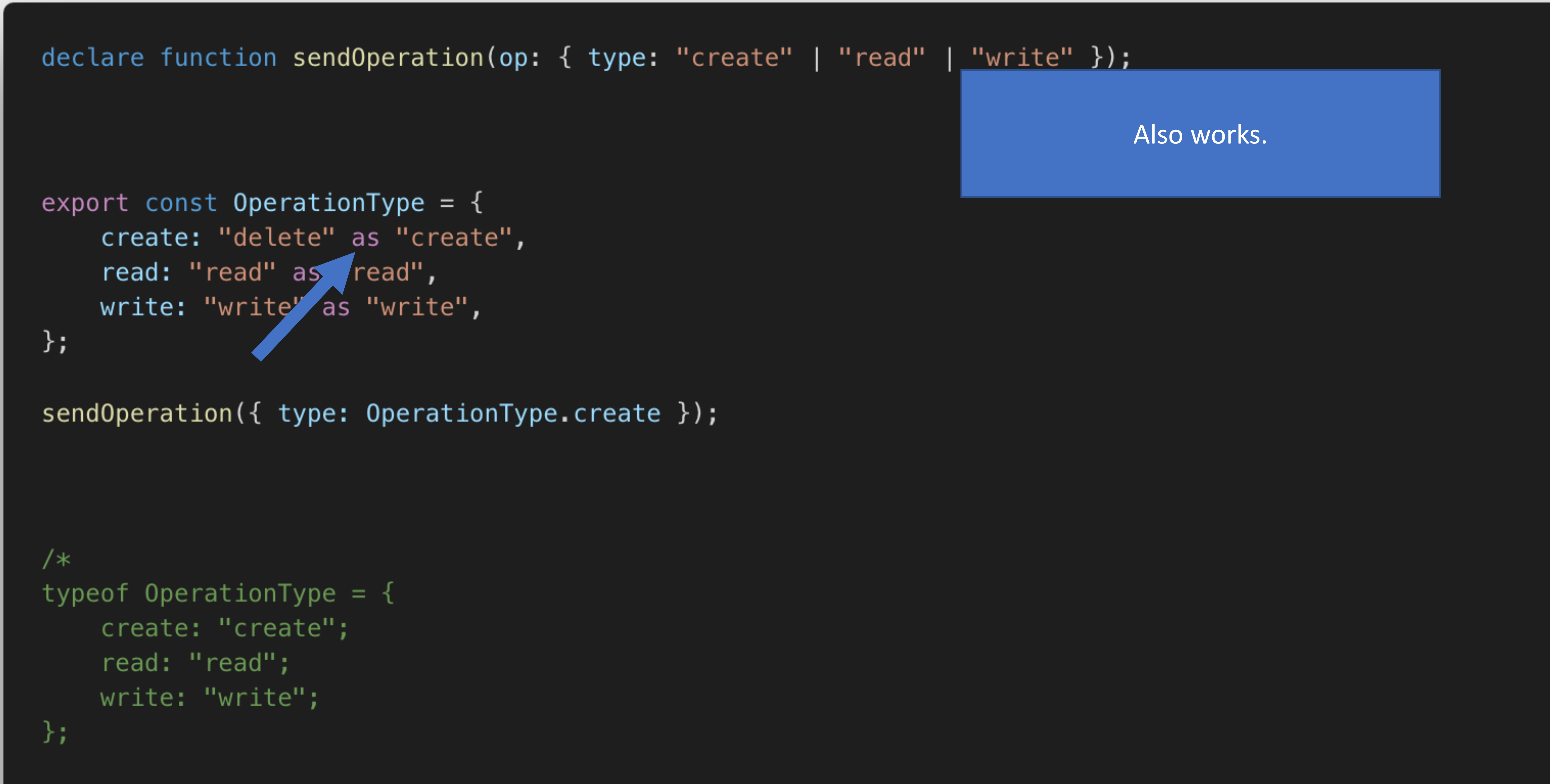
Error: 'string' is not assignable to type '"create" | "read" | "write"'

Bonus!
Very small footprint



```
declare function sendOperation(op: { type: "create" | "read" | "write" });



export const OperationType = {
    create: "create" as "create",
    read: "read" as "read",
    write: "write" as "write",
};

sendOperation({ type: OperationType.create });



/*
typeof OperationType = {
    create: "create";
    read: "read";
    write: "write";
};

*/
```

Works.



```
declare function sendOperation(op: { type: "create" | "read" | "write" });


export const OperationType = {
    create: "delete" as "create",
    read: "read" as "read",
    write: "write" as "write",
};

sendOperation({ type: OperationType.create });



/*
typeof OperationType = {
    create: "create";
    read: "read";
    write: "write";
};
```

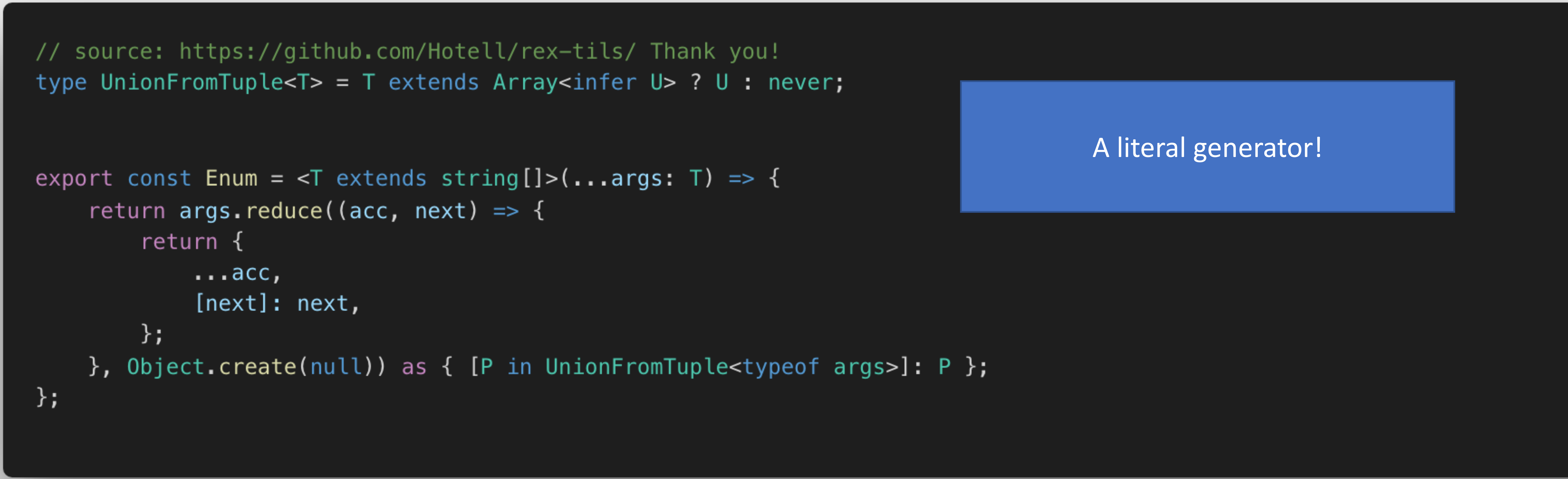
Also works.

```
declare function sendOperation(op: { type: "create" | "read" | "write" });



export const OperationType = {
    create: "create",
    read: "read",
    write: "write",
} as const;

sendOperation({ type: OperationType.create });



/*
typeof OperationType = {
    create: "create";
    read: "read";
    write: "write";
};

*/
```

Less repetitions. “delete” no longer possible

Still verbose…

## Is that all we can achieve?



```
// source: https://github.com/Hotell/rex-tils/ Thank you!
type UnionFromTuple<T> = T extends Array<infer U> ? U : never;


export const Enum = <T extends string[]>(...args: T) => {
    return args.reduce((acc, next) => {
        return {
            ...acc,
            [next]: next,
        };
    }, Object.create(null)) as { [P in UnionFromTuple<typeof args>]: P };
};
```

A literal generator!

```
declare function sendOperation(op: { type: OperationType });



export const OperationType = Enum( "create", "read", "write" )
export type OperationType = keyof typeof OperationType

sendOperation({ type: OperationType.create });


/*
typeof OperationType = {
    create: "create";
    read: "read";
    write: "write";
};

type OperationType = "create" | "read" | "write"

*/
```

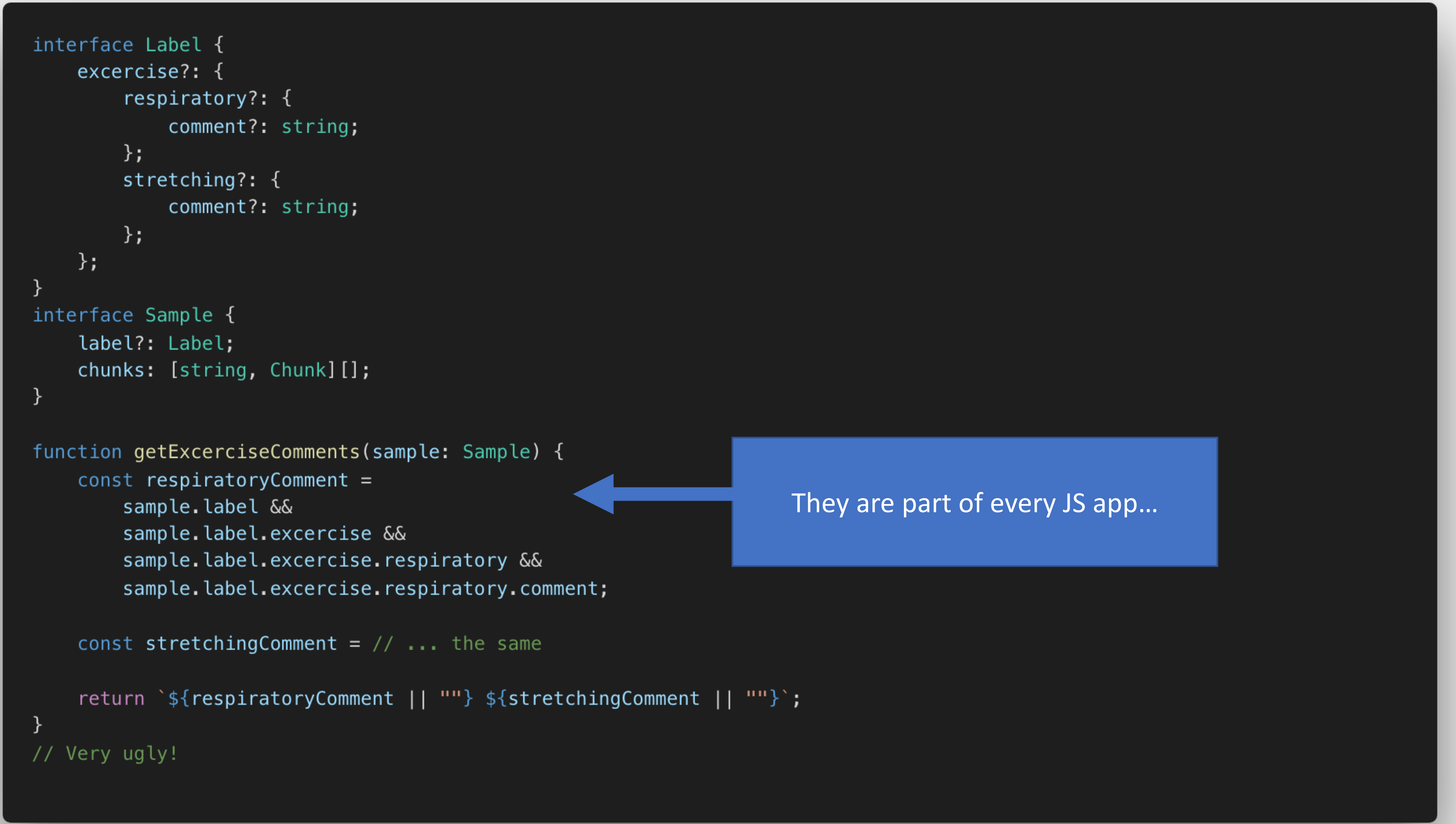
So clean! So short!

Smallest footprint if app has many enums!

All mistakes are visible

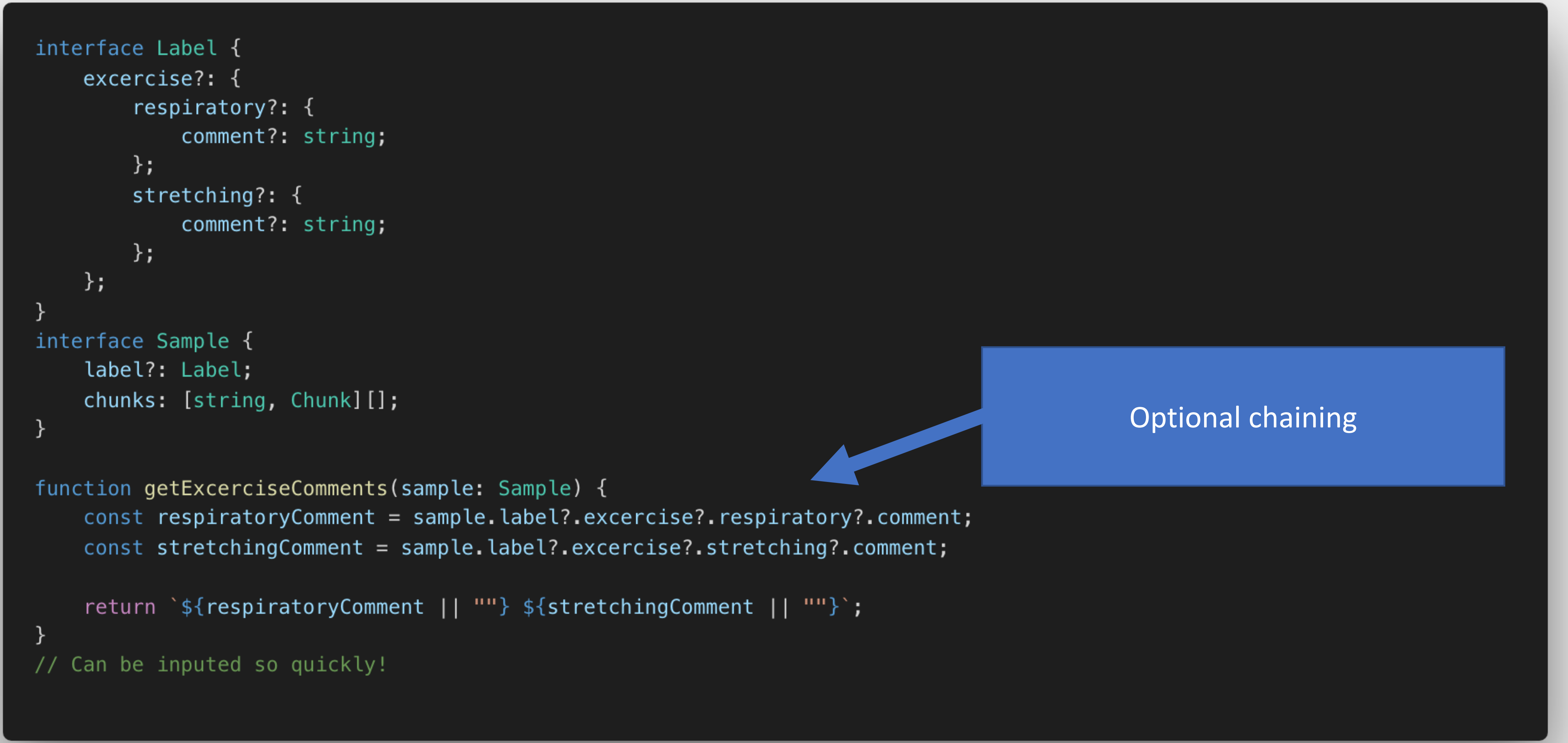
Fight verbosity with…

## Optional chaining (TS 3.7)

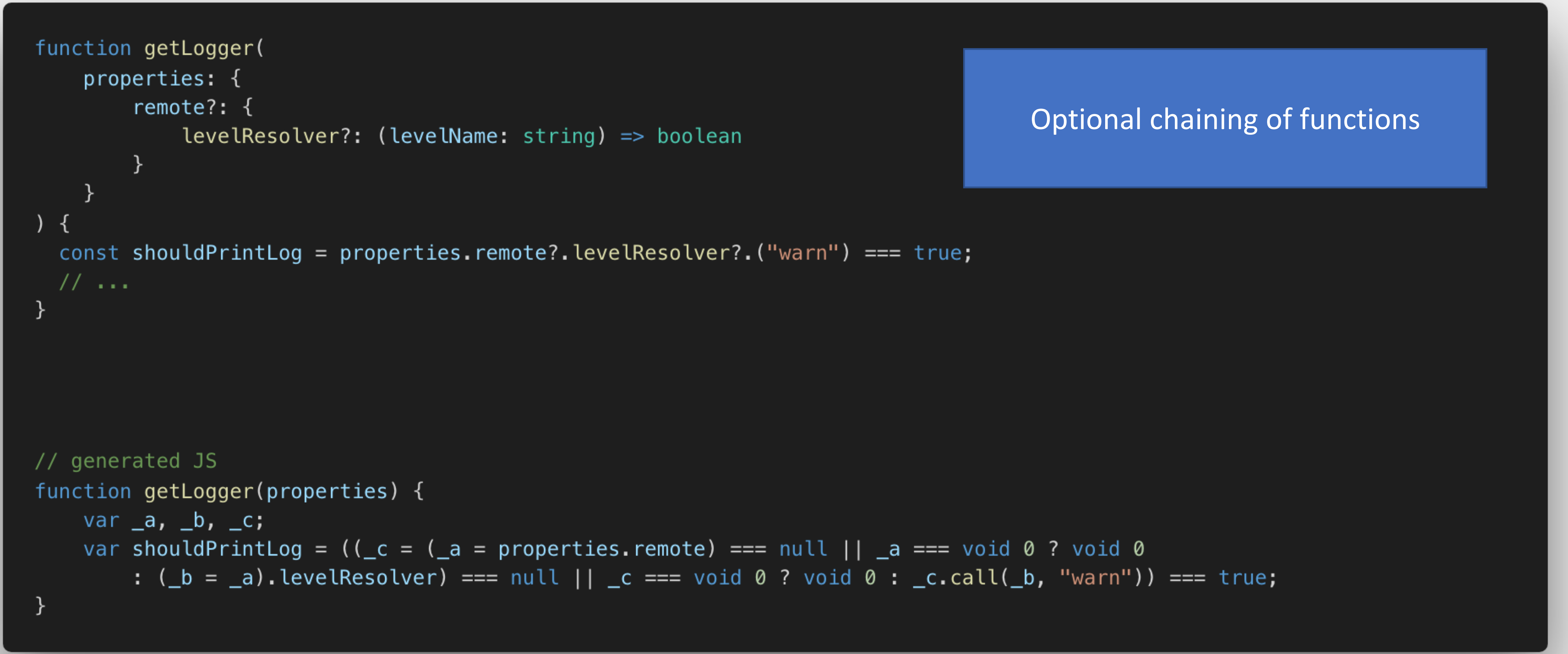


```
interface Label {
    excercise?: {
        respiratory?: {
            comment?: string;
        };
        stretching?: {
            comment?: string;
        };
    };
}
interface Sample {
    label?: Label;
    chunks: [string, Chunk][];
}

function getExcerciseComments(sample: Sample) {
    const respiratoryComment =
        sample.label &&
        sample.label.excercise &&
        sample.label.excercise.respiratory &&
        sample.label.excercise.respiratory.comment;

    const stretchingComment = // ... the same

    return `${respiratoryComment || ""} ${stretchingComment || ""}`;
}
// Very ugly!
```

They are part of every JS app…



```
interface Label {
    excercise?: {
        respiratory?: {
            comment?: string;
        };
        stretching?: {
            comment?: string;
        };
    };
}
interface Sample {
    label?: Label;
    chunks: [string, Chunk][];
}

function getExcerciseComments(sample: Sample) {
    const respiratoryComment = sample.label?.excercise?.respiratory?.comment;
    const stretchingComment = sample.label?.excercise?.stretching?.comment;

    return `${respiratoryComment || ""} ${stretchingComment || ""}`;
}
// Can be inputed so quickly!
```

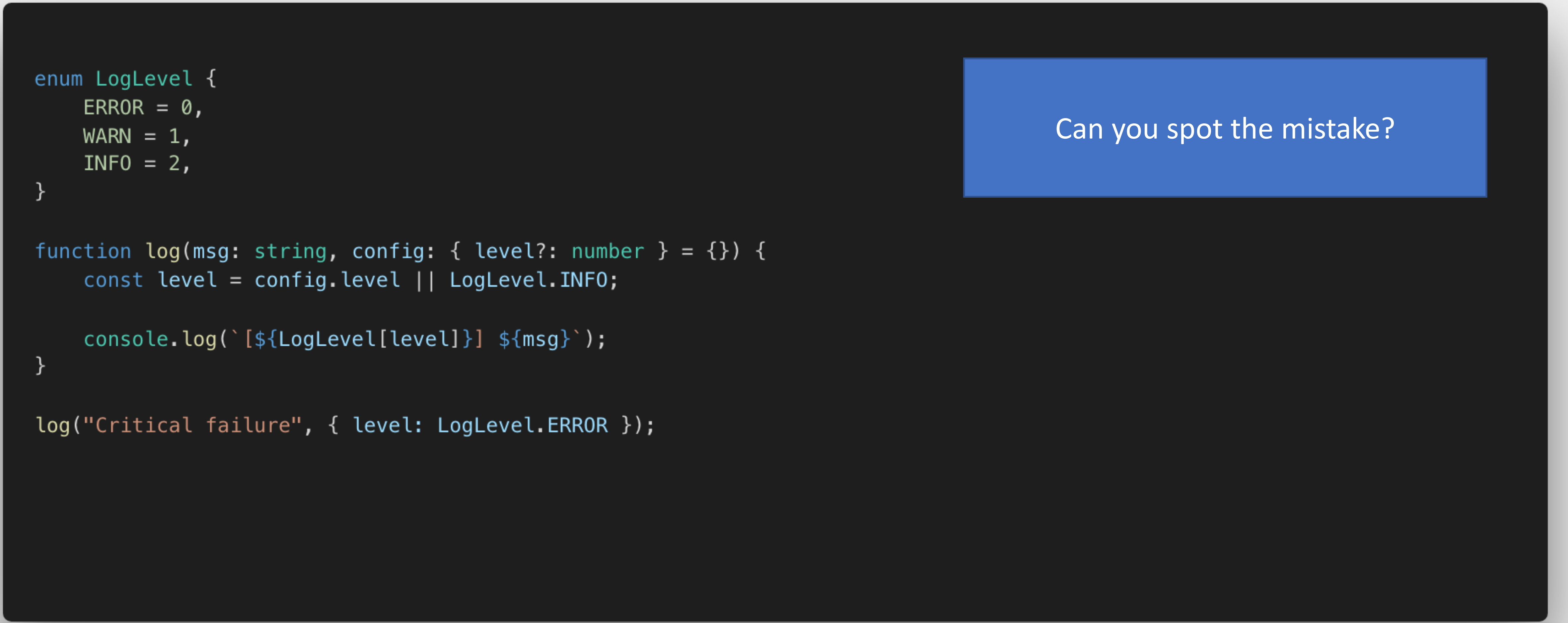


```
function getLogger(
    properties: {
        remote?: {
            levelResolver?: (levelName: string) => boolean
        }
    }
) {
  const shouldPrintLog = properties.remote?.levelResolver?.("warn") === true;
  // ...
}




// generated JS
function getLogger(properties) {
    var _a, _b, _c;
    var shouldPrintLog = ((_c = (_a = properties.remote) === null || _a === void 0 ? void 0
        : (_b = _a).levelResolver) === null || _c === void 0 ? void 0 : _c.call(_b, "warn")) === true;
}
```

Optional chaining of functions

Fight verbosity with…

## Nullish Coalescing (TS 3.7)



```
enum LogLevel {
    ERROR = 0,
    WARN = 1,
    INFO = 2,
}

function log(msg: string, config: { level?: number } = {}) {
    const level = config.level || LogLevel.INFO;

    console.log(`[${LogLevel[level]}] ${msg}`);
}

log("Critical failure", { level: LogLevel.ERROR });
```

Can you spot the mistake?

```typescript
enum LogLevel {
    ERROR = 0,
    WARN = 1,
    INFO = 2,
}

function log(msg: string, config: { level?: number } = {}) {
    const level = config.level || LogLevel.INFO;

    console.log(`[${LogLevel[level]}] ${msg}`);
}

log("Critical failure", { level: LogLevel.ERROR });
```

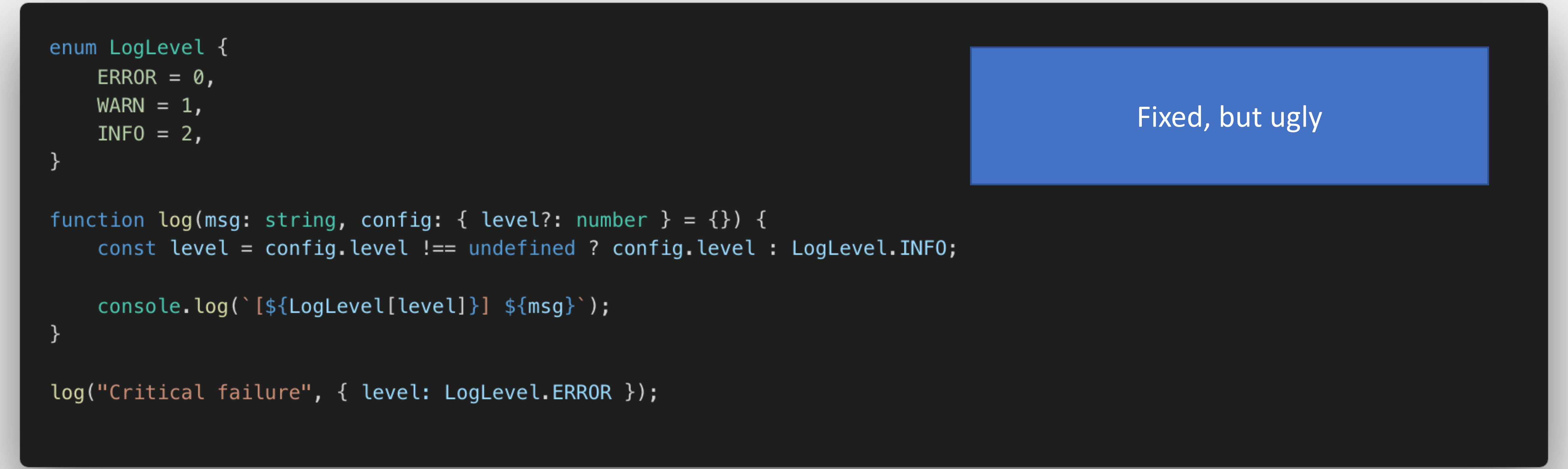
Can you spot the mistake?

Prints ‘[INFO] …’

```
enum LogLevel {
    ERROR = 0,
    WARN = 1,
    INFO = 2,
}

function log(msg: string, config: { level?: number } = {}) {
    const level = config.level || LogLevel.INFO;

    console.log(`[${LogLevel[level]}] ${msg}`);
}

log("Critical failure", { level: LogLevel.ERROR });
```

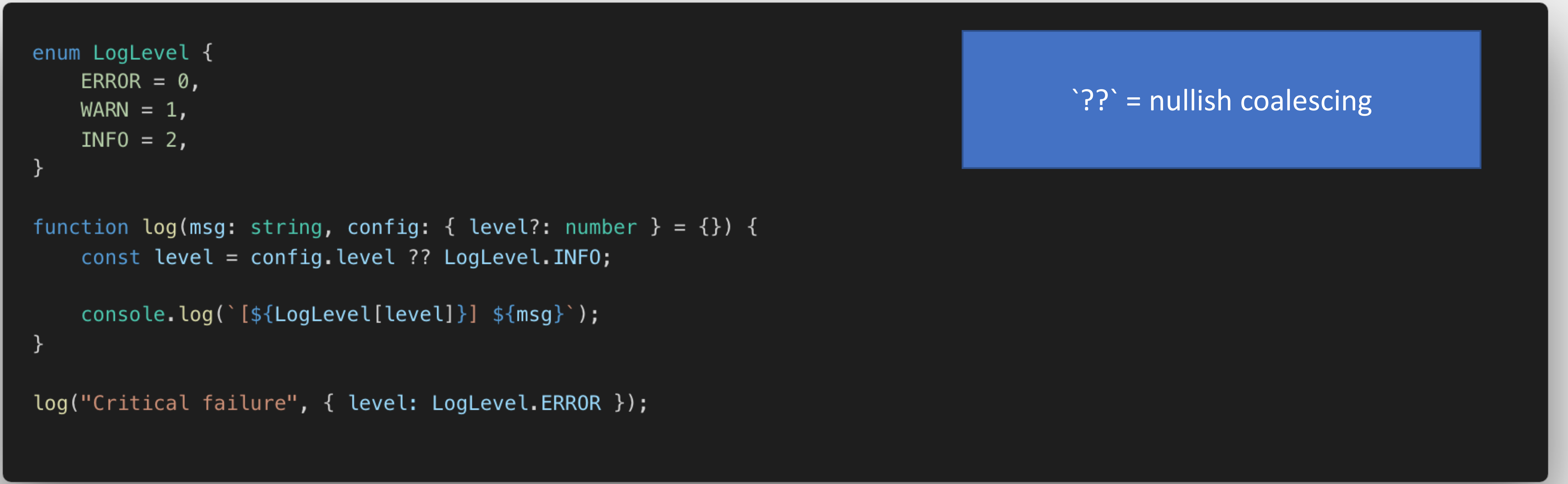
Can you spot the mistake?

ERROR = 0
0 = Falsy



```
enum LogLevel {
    ERROR = 0,
    WARN = 1,
    INFO = 2,
}

function log(msg: string, config: { level?: number } = {}) {
    const level = config.level !== undefined ? config.level : LogLevel.INFO;

    console.log(`[${LogLevel[level]}] ${msg}`);
}

log("Critical failure", { level: LogLevel.ERROR });
```

Fixed, but ugly



```
enum LogLevel {
    ERROR = 0,
    WARN = 1,
    INFO = 2,
}

function log(msg: string, config: { level?: number } = {}) {
    const level = config.level ?? LogLevel.INFO;

    console.log(`[${LogLevel[level]}] ${msg}`);
}

log("Critical failure", { level: LogLevel.ERROR });
```

`??` = nullish coalescing

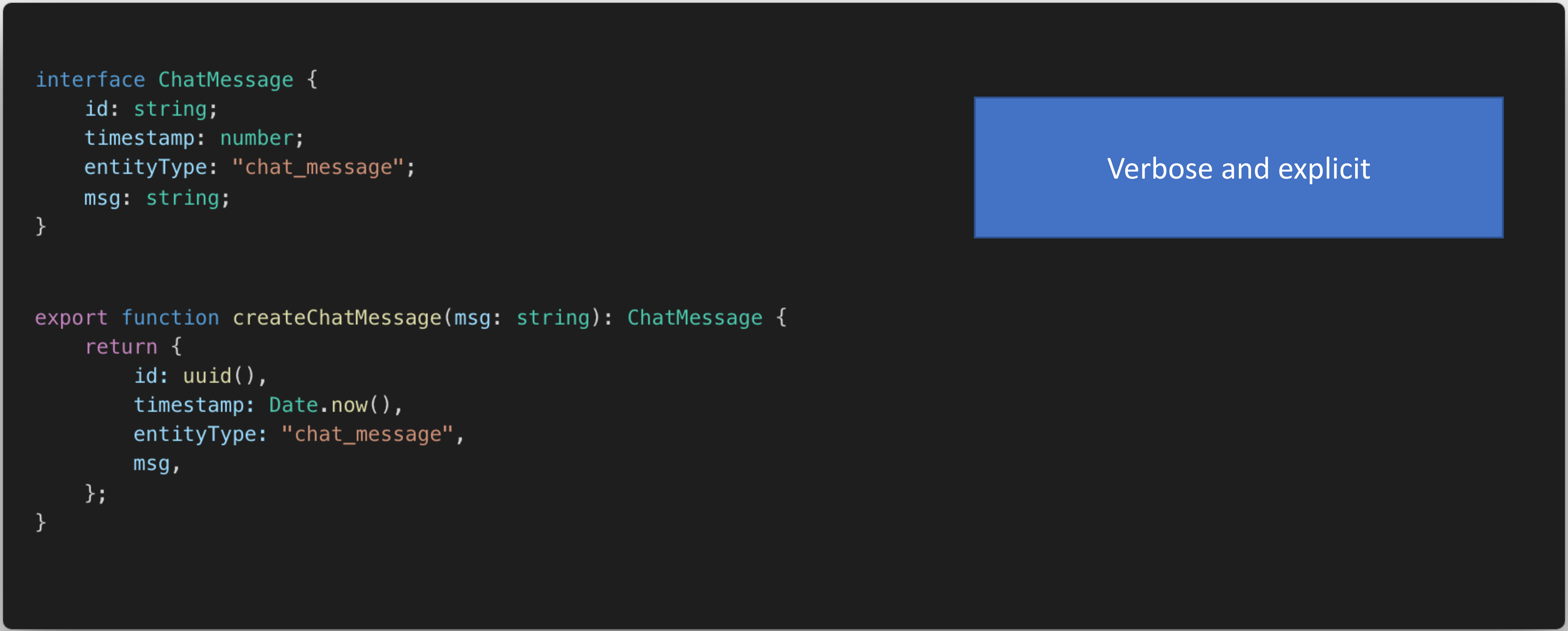
Fight…

# Repetitions

Stay DRY

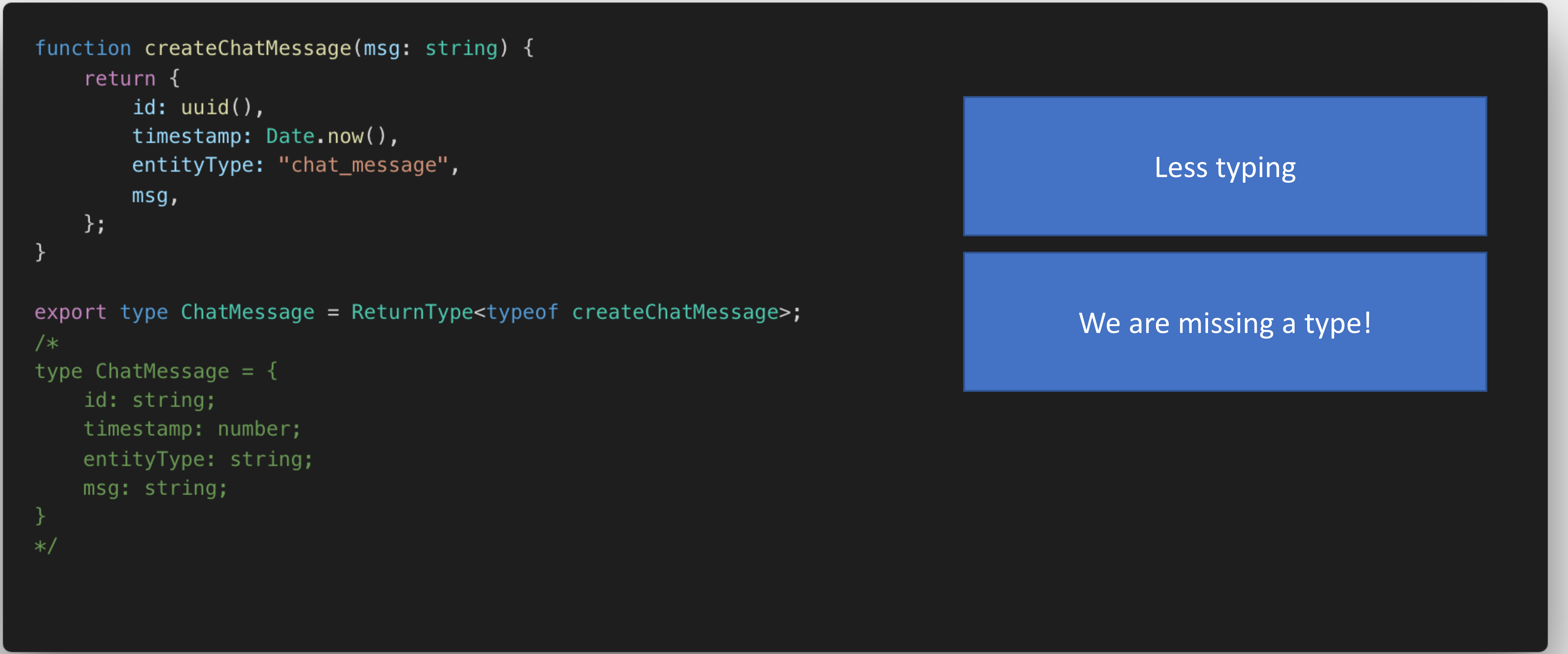
Fight repetitions with…

## Extracting types



```
interface ChatMessage {
    id: string;
    timestamp: number;
    entityType: "chat_message";
    msg: string;
}


export function createChatMessage(msg: string): ChatMessage {
    return {
        id: uuid(),
        timestamp: Date.now(),
        entityType: "chat_message",
        msg,
    };
}
```

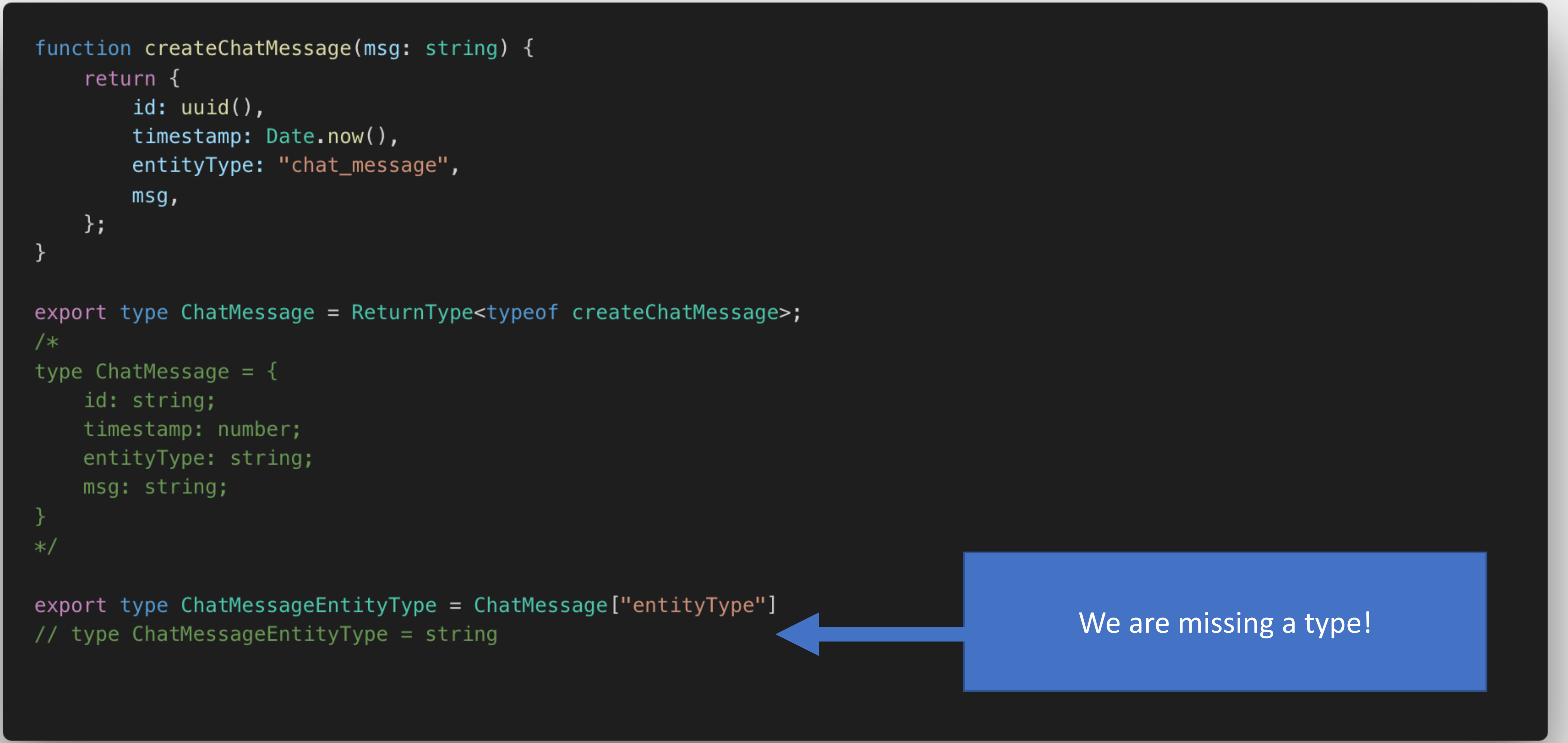
Verbose and explicit



```
function createChatMessage(msg: string) {
    return {
        id: uuid(),
        timestamp: Date.now(),
        entityType: "chat_message",
        msg,
    };
}

export type ChatMessage = ReturnType<typeof createChatMessage>;
/*
type ChatMessage = {
    id: string;
    timestamp: number;
    entityType: string;
    msg: string;
}
*/
```

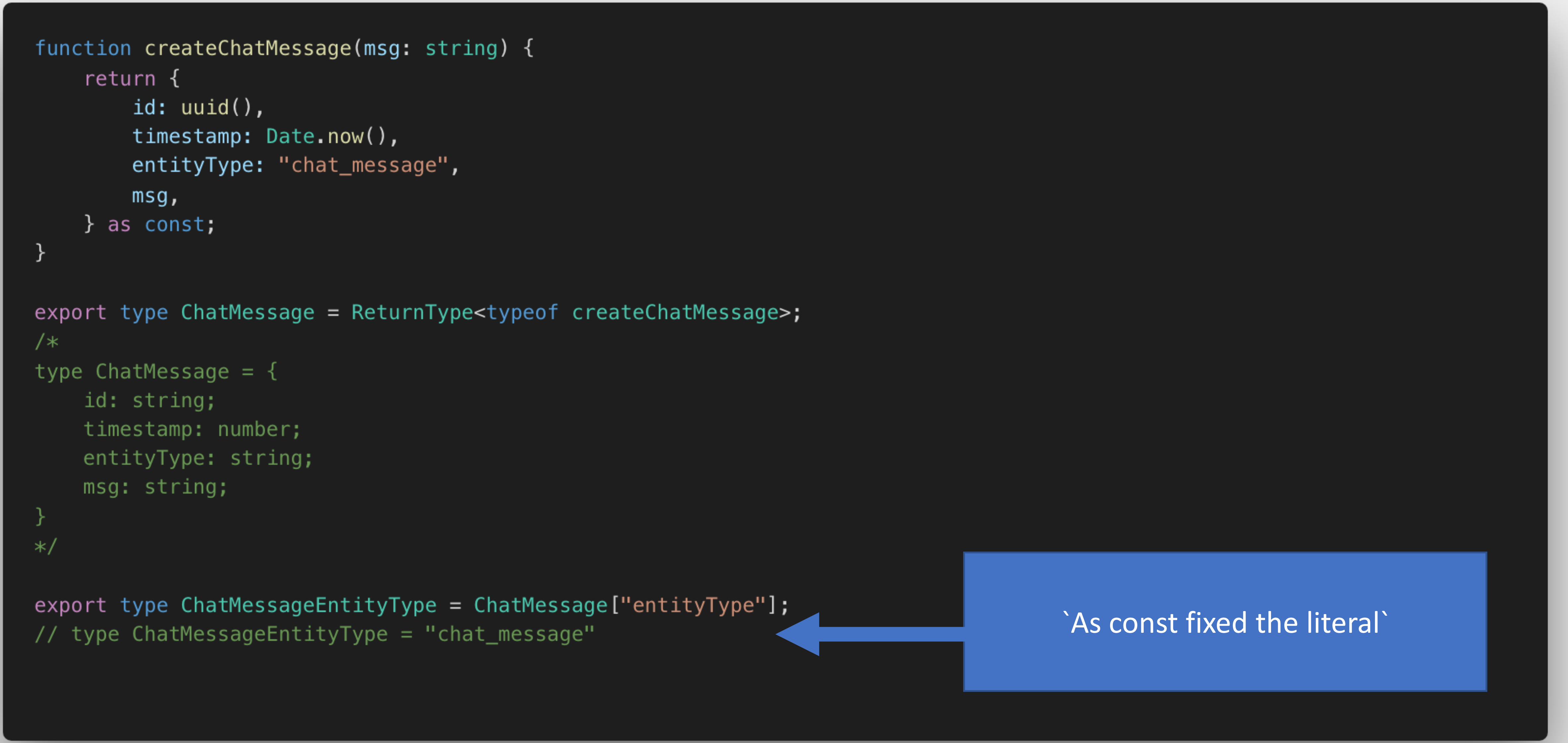
Less typing

We are missing a type!



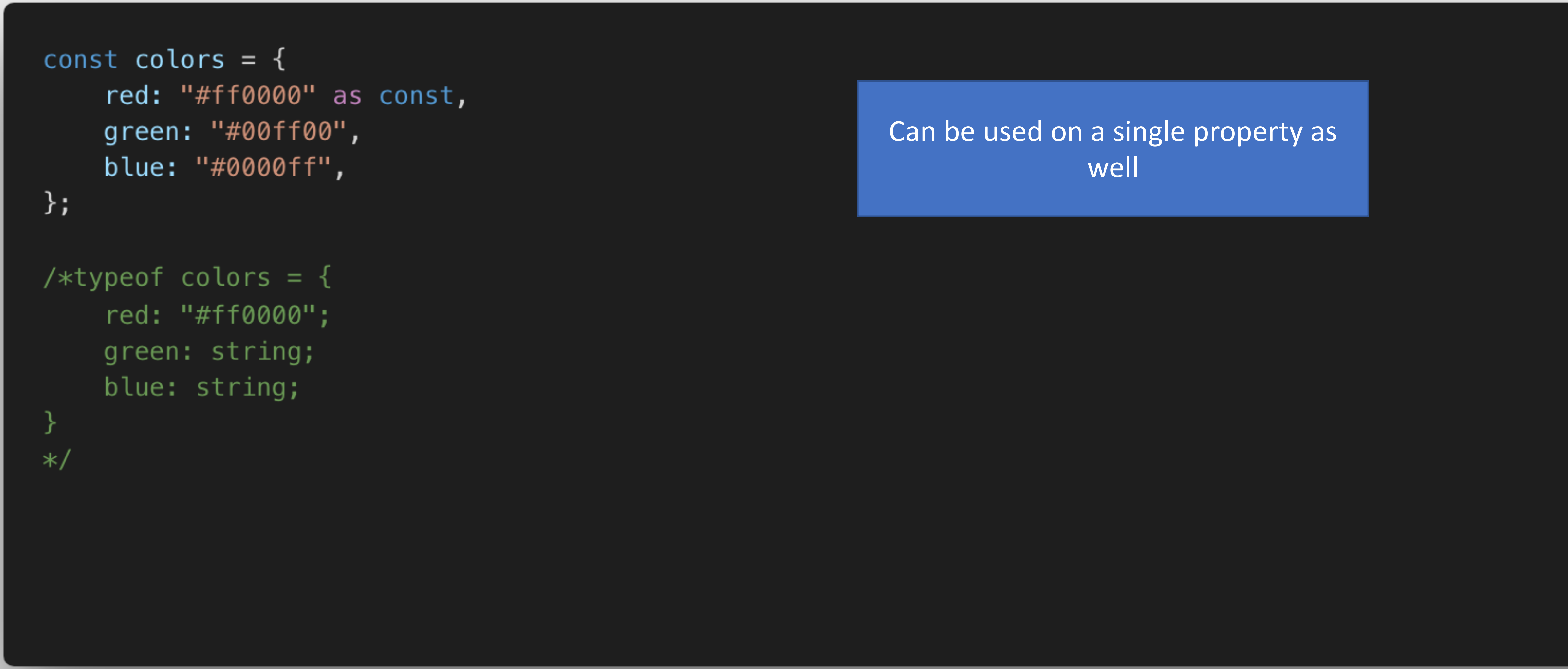
```
function createChatMessage(msg: string) {
    return {
        id: uuid(),
        timestamp: Date.now(),
        entityType: "chat_message",
        msg,
    };
}

export type ChatMessage = ReturnType<typeof createChatMessage>;
/*
type ChatMessage = {
    id: string;
    timestamp: number;
    entityType: string;
    msg: string;
}
*/

export type ChatMessageEntityType = ChatMessage["entityType"]
// type ChatMessageEntityType = string
```

We are missing a type!



```typescript
function createChatMessage(msg: string) {
    return {
        id: uuid(),
        timestamp: Date.now(),
        entityType: "chat_message",
        msg,
    } as const;
}

export type ChatMessage = ReturnType<typeof createChatMessage>;
/*
type ChatMessage = {
    id: string;
    timestamp: number;
    entityType: string;
    msg: string;
}
*/

export type ChatMessageEntityType = ChatMessage["entityType"];
// type ChatMessageEntityType = "chat_message"
```

`As const fixed the literal`



```
const colors = {
    red: "#ff0000" as const,
    green: "#00ff00",
    blue: "#0000ff",
};

/*typeof colors = {
    red: "#ff0000";
    green: string;
    blue: string;
}
*/
```

Can be used on a single property as well

Fight repetitions

## What if an external library doesn’t provide you with the type you want?

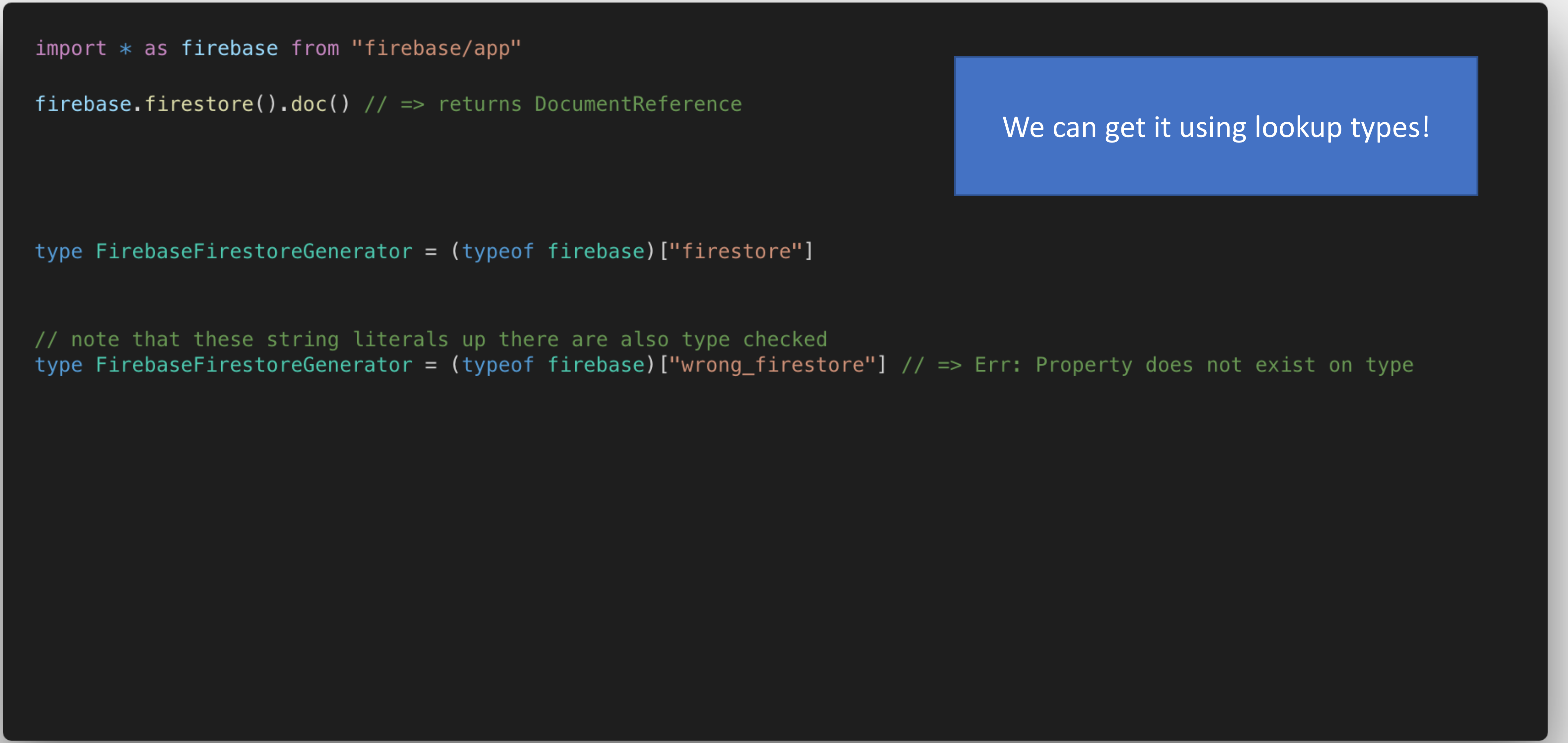
```
import * as firebase from "firebase/app"

firebase.firestore().doc() // => returns DocumentReference

type FirebaseDocumentReference = ?
```

We want it bad…

```
import * as firebase from "firebase/app"

firebase.firestore().doc() // => returns DocumentReference

type FirebaseDocumentReference = ?
```

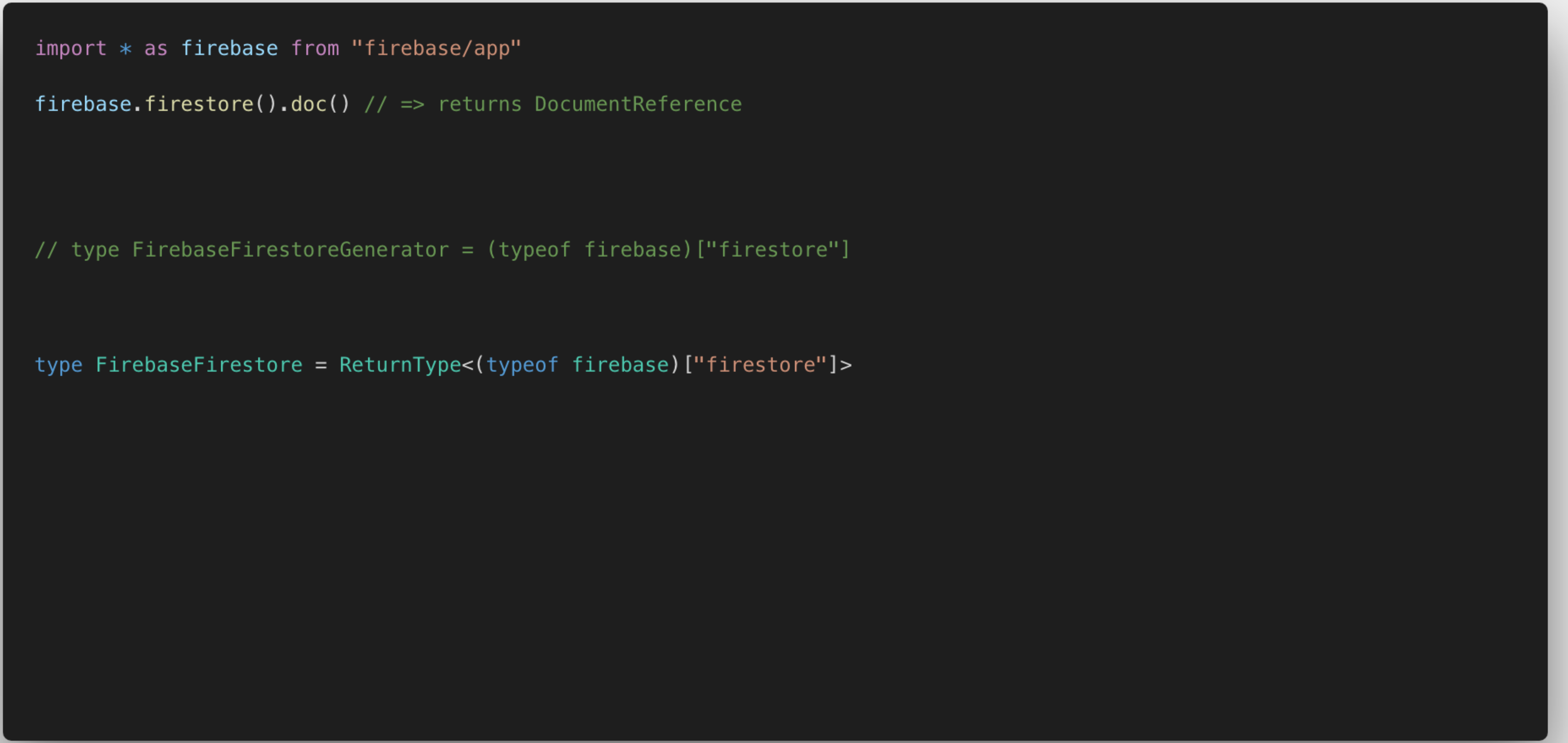
We want it bad…

We can get it using lookup types!

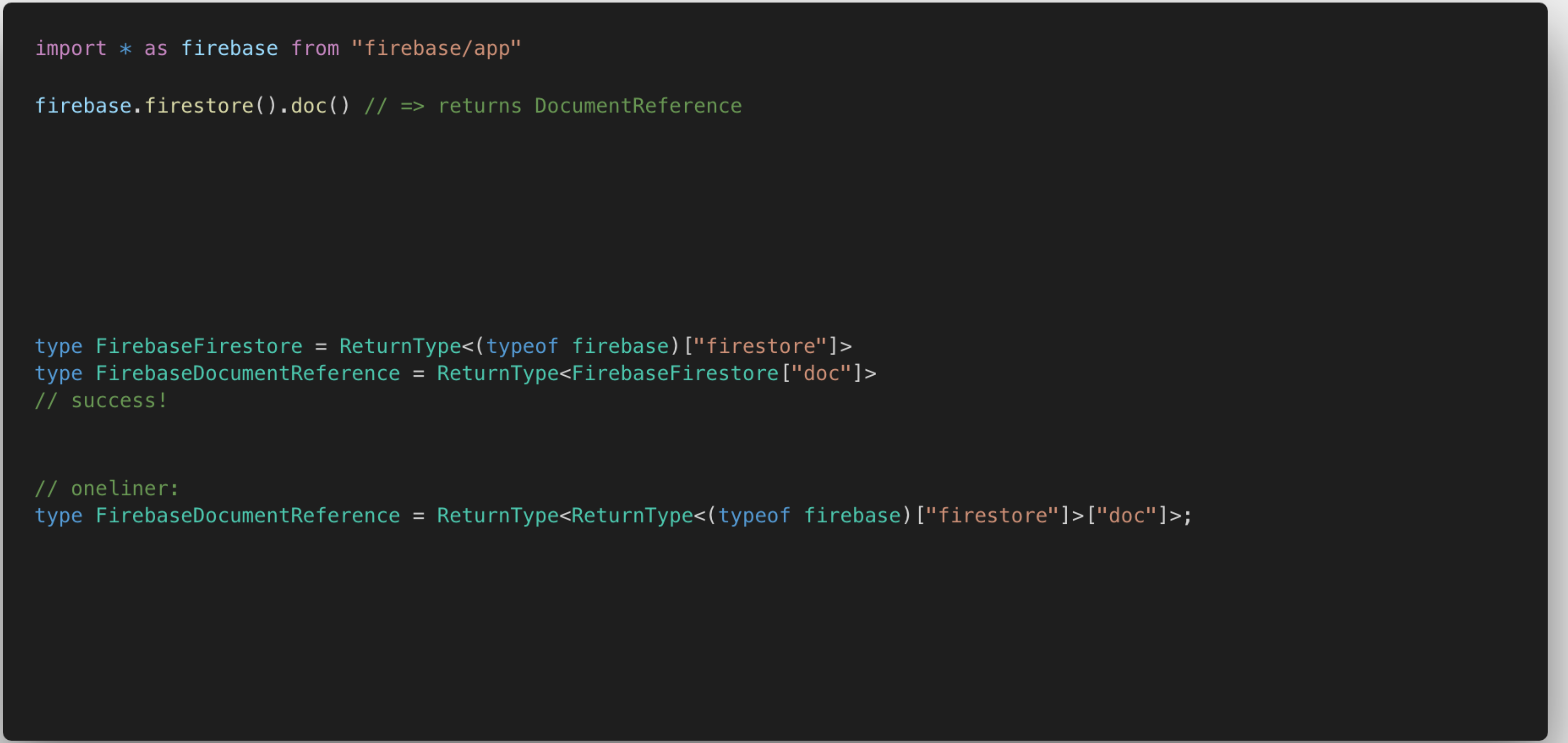


```
import * as firebase from "firebase/app"

firebase.firestore().doc() // => returns DocumentReference



type FirebaseFirestoreGenerator = (typeof firebase)["firestore"]


// note that these string literals up there are also type checked
type FirebaseFirestoreGenerator = (typeof firebase)["wrong_firestore"] // => Err: Property does not exist on type
```

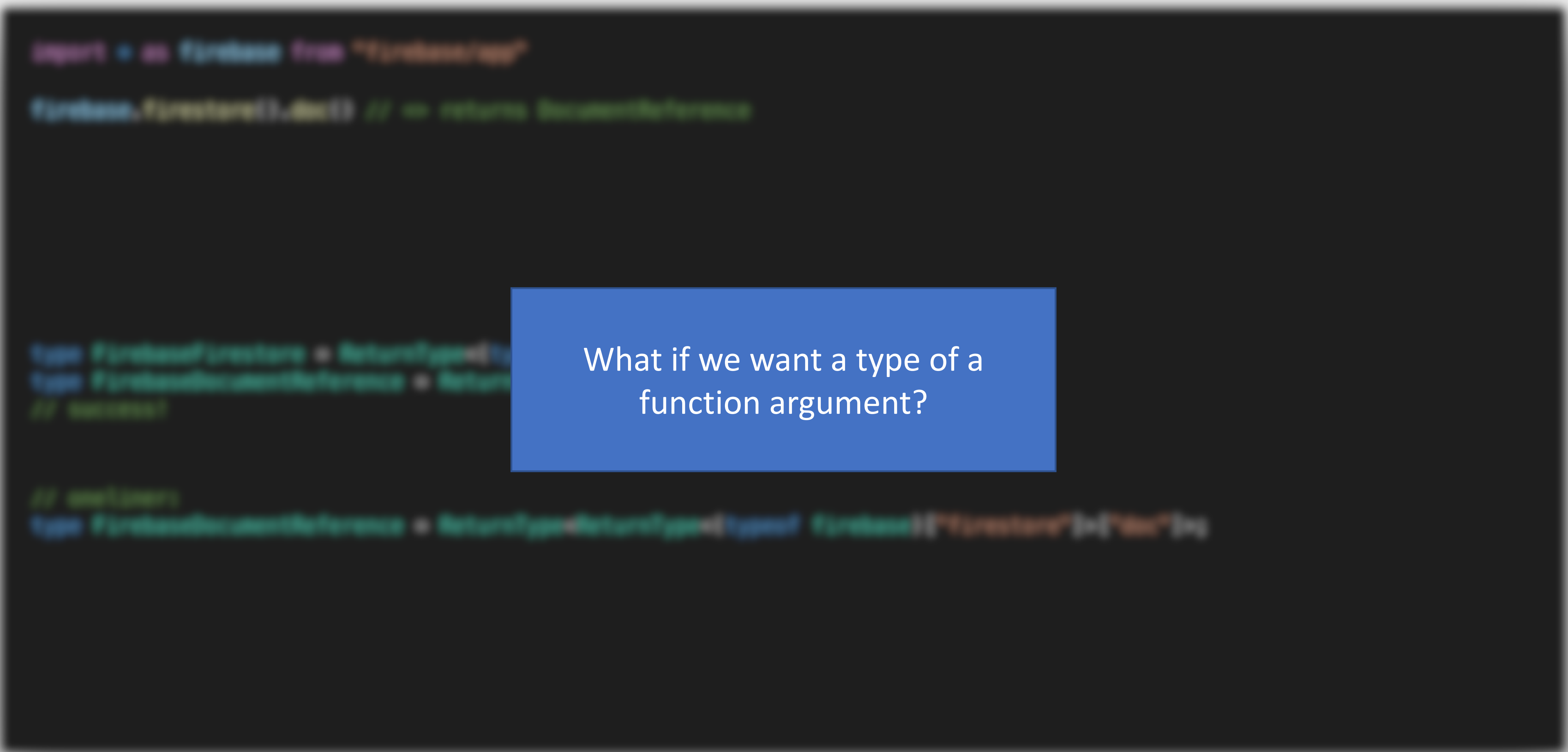
We can get it using lookup types!



```typescript
import * as firebase from "firebase/app"

firebase.firestore().doc() // => returns DocumentReference



// type FirebaseFirestoreGenerator = (typeof firebase)["firestore"]



type FirebaseFirestore = ReturnType<(typeof firebase)["firestore"]>
```

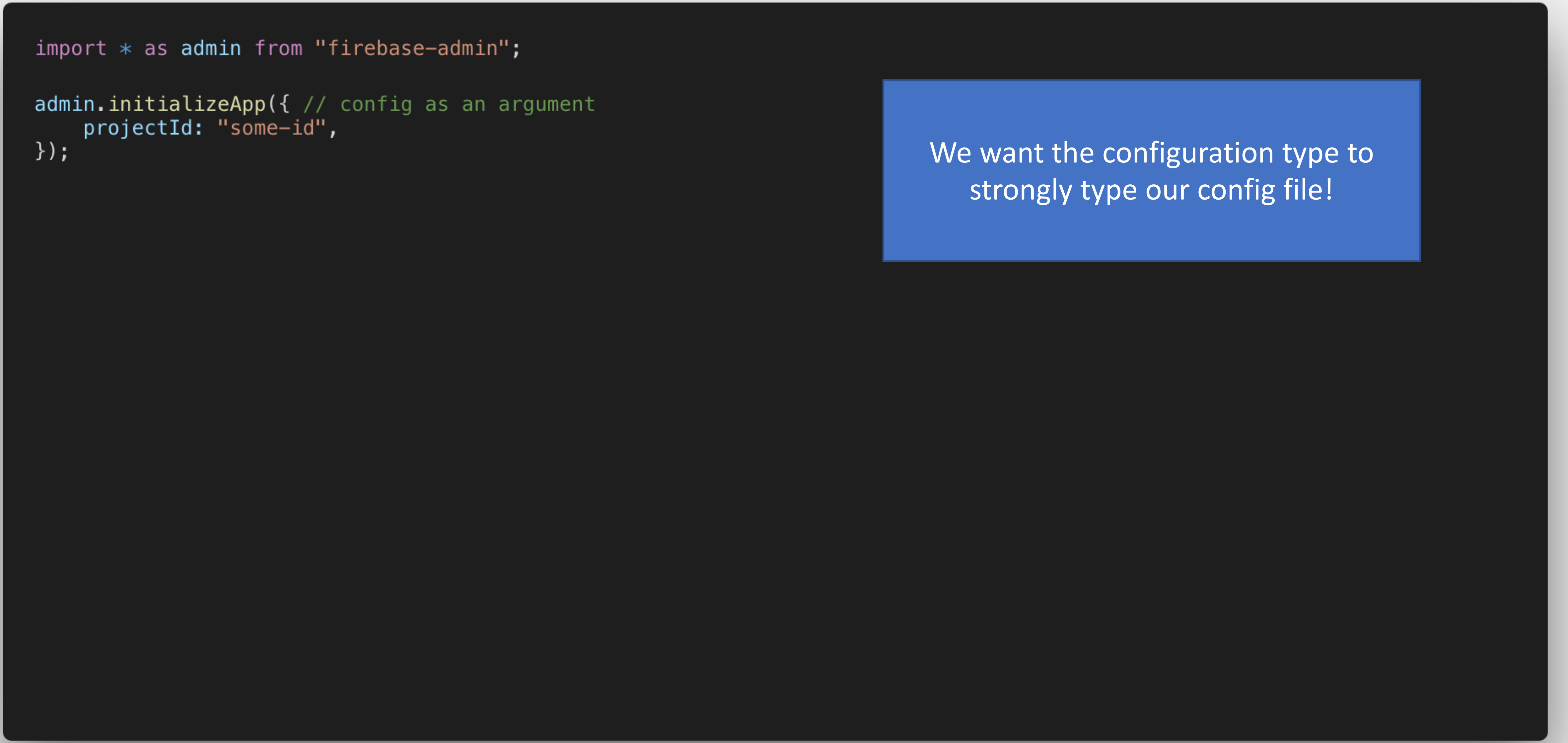


```
import * as firebase from "firebase/app"

firebase.firestore().doc() // => returns DocumentReference







type FirebaseFirestore = ReturnType<(typeof firebase)["firestore"]>
type FirebaseDocumentReference = ReturnType<FirebaseFirestore["doc"]>
// success!


// oneliner:
type FirebaseDocumentReference = ReturnType<ReturnType<(typeof firebase)["firestore"]>["doc"]>;
```



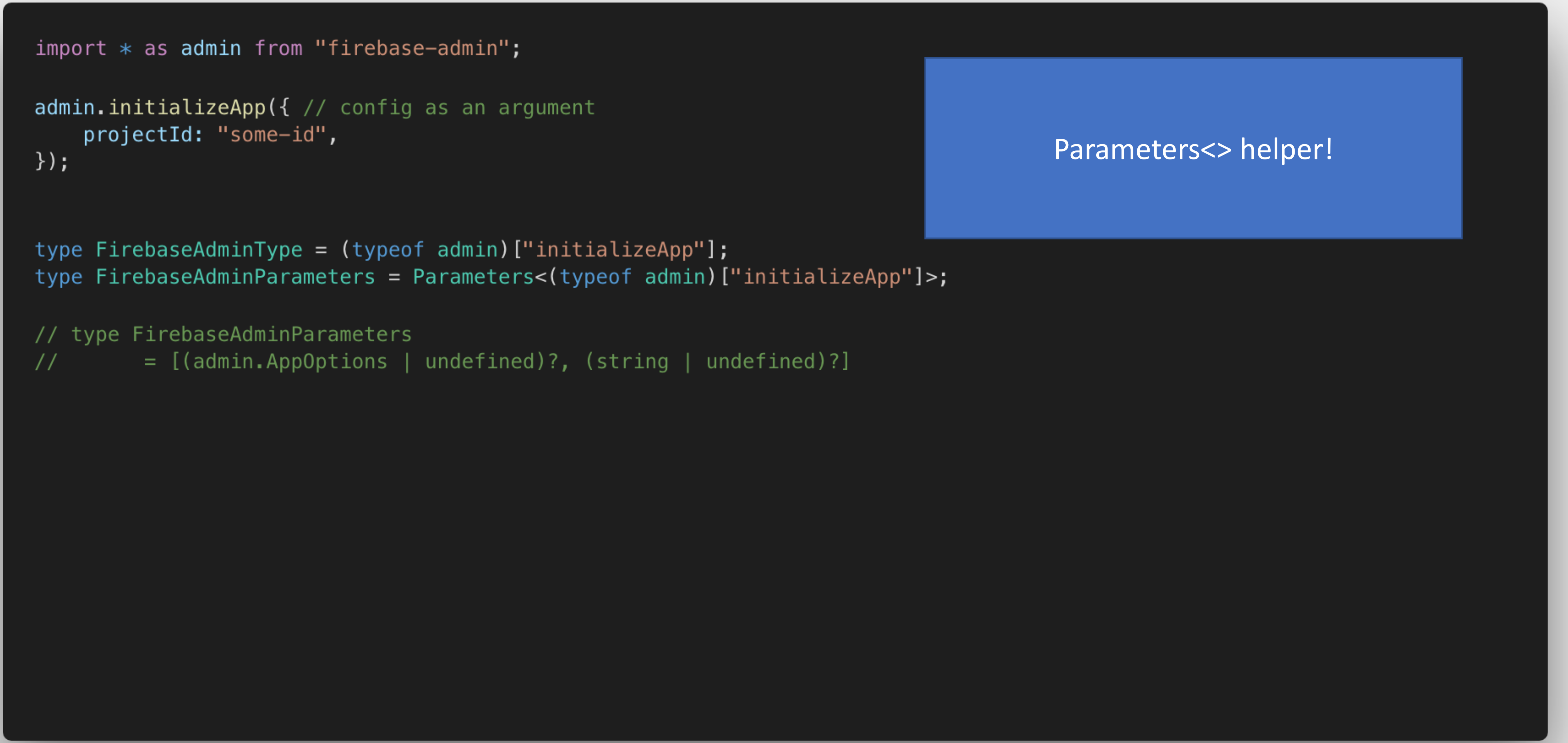
```
import * as firebase from "firebase/app"

firebase.firestore().doc() // => returns DocumentReference





type FirebaseFirestore = ReturnType<(ty
type FirebaseDocumentReference = Return
// success!


// oneliner:
type FirebaseDocumentReference = ReturnType<ReturnType<(typeof firebase)["firestore"]>["doc"]>;
```

What if we want a type of a function argument?



```
import * as admin from "firebase-admin";

admin.initializeApp({ // config as an argument
    projectId: "some-id",
});
```

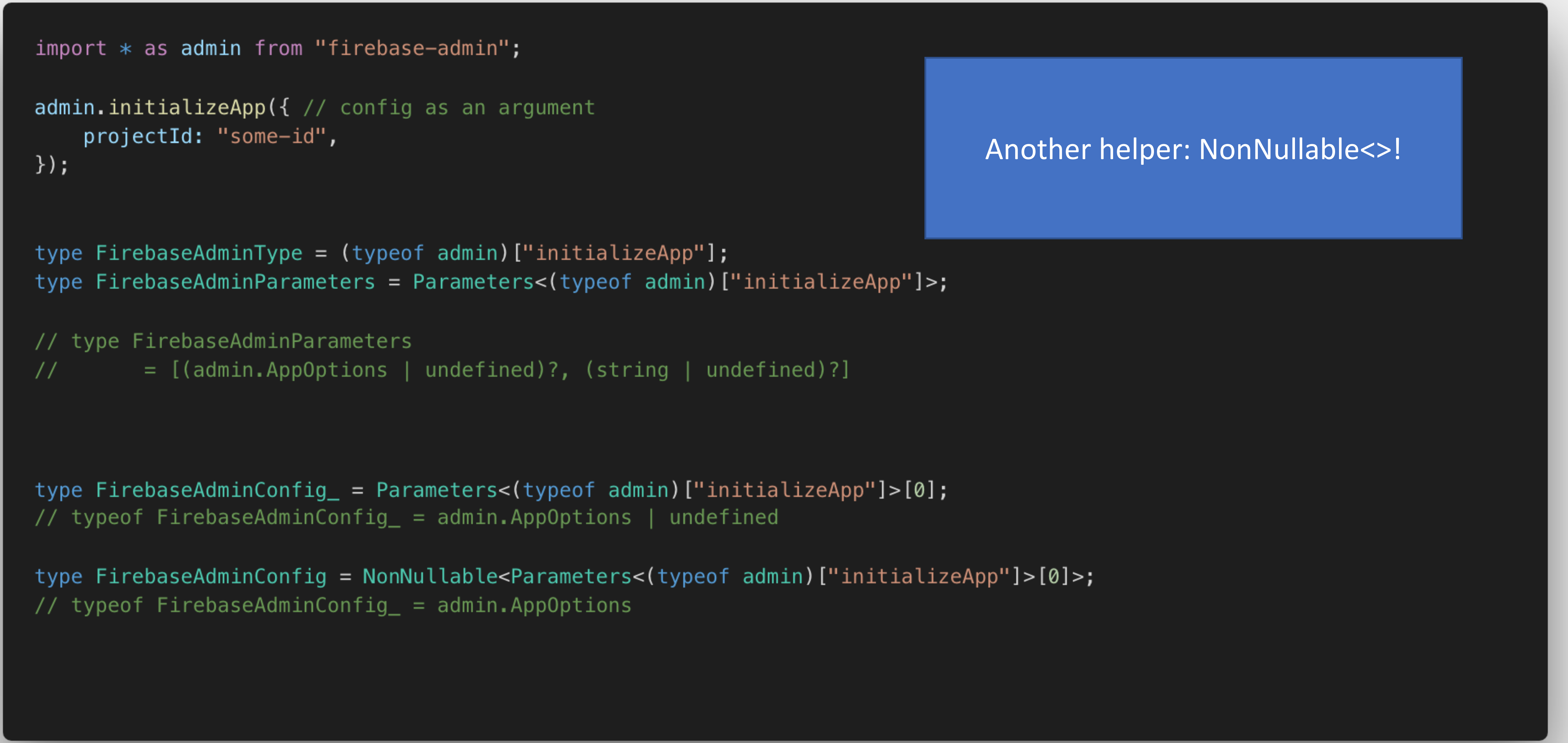
We want the configuration type to strongly type our config file!



```
import * as admin from "firebase-admin";

admin.initializeApp({ // config as an argument
    projectId: "some-id",
});


type FirebaseAdminType = (typeof admin)["initializeApp"];
type FirebaseAdminParameters = Parameters<(typeof admin)["initializeApp"]>;

// type FirebaseAdminParameters
//        = [(admin.AppOptions | undefined)?, (string | undefined)?]
```

Parameters<> helper!

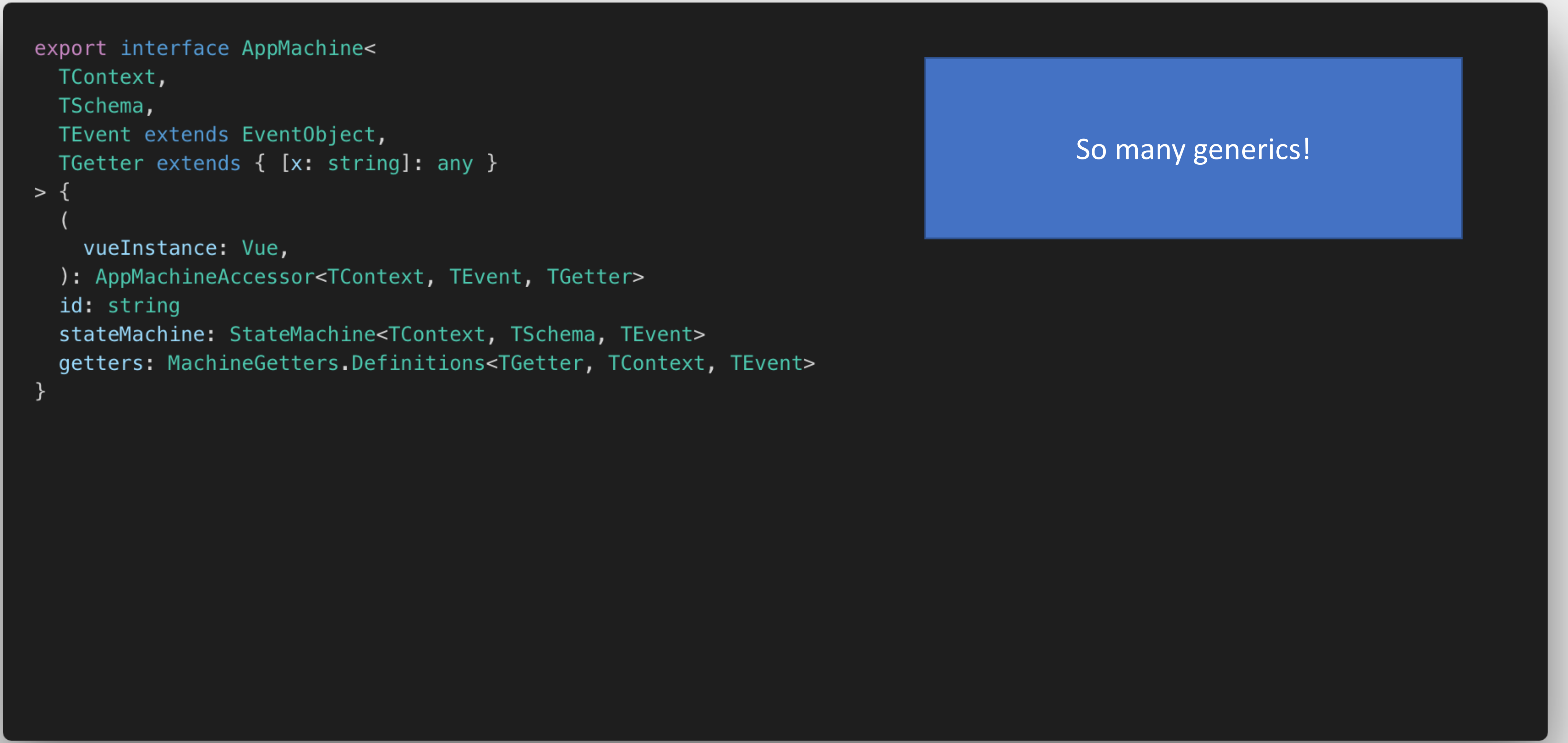
Another helper: NonNullable<>!



```typescript
import * as admin from "firebase-admin";

admin.initializeApp({ // config as an argument
    projectId: "some-id",
});


type FirebaseAdminType = (typeof admin)["initializeApp"];
type FirebaseAdminParameters = Parameters<(typeof admin)["initializeApp"]>;

// type FirebaseAdminParameters
//       = [(admin.AppOptions | undefined)?, (string | undefined)?]



type FirebaseAdminConfig_ = Parameters<(typeof admin)["initializeApp"]>[0];
// typeof FirebaseAdminConfig_ = admin.AppOptions | undefined

type FirebaseAdminConfig = NonNullable<Parameters<(typeof admin)["initializeApp"]>[0]>;
// typeof FirebaseAdminConfig_ = admin.AppOptions
```

Fight repetitions with…

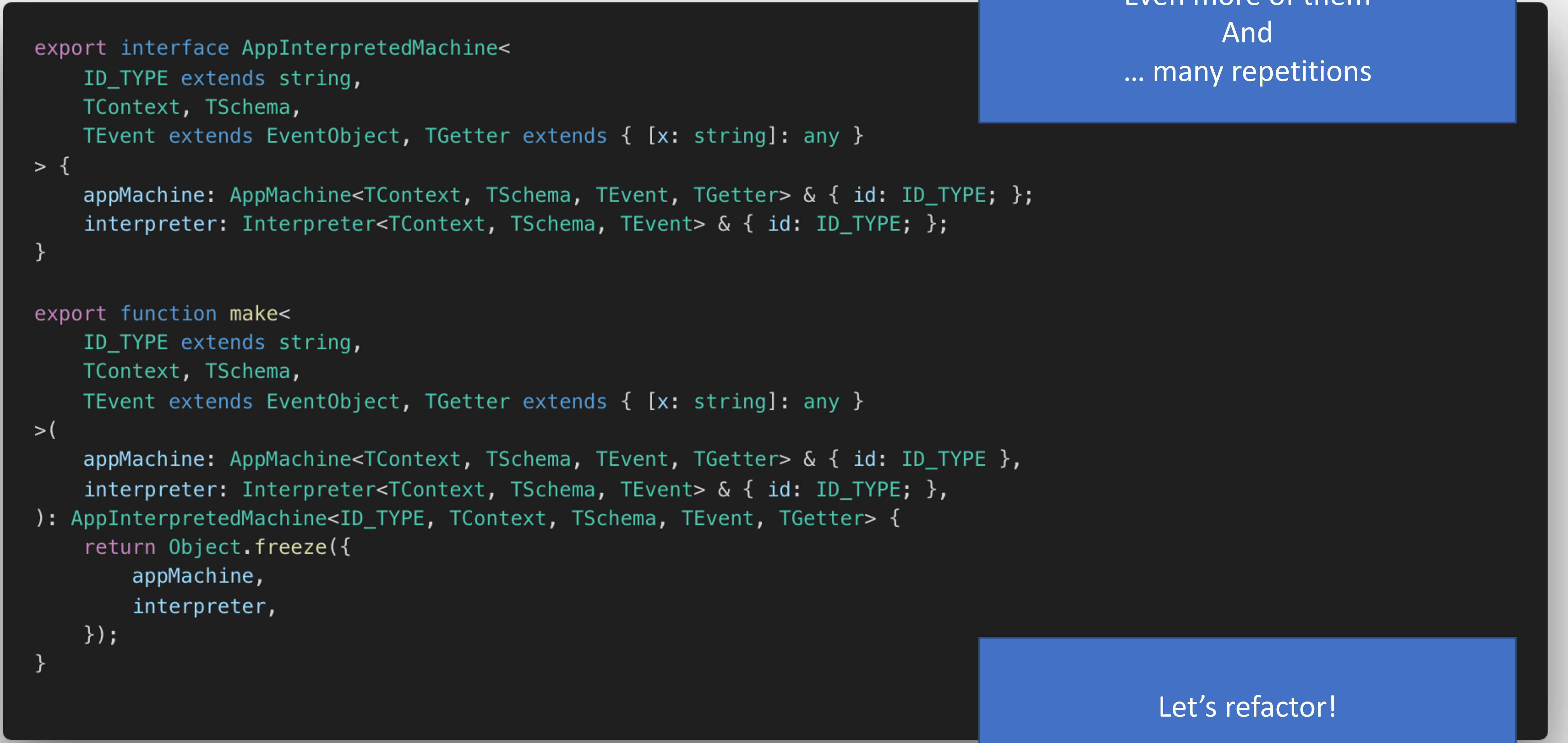
## Shipping type containers using conditional-infer



```
export interface AppMachine<
  TContext,
  TSchema,
  TEvent extends EventObject,
  TGetter extends { [x: string]: any }
> {
  (
    vueInstance: Vue,
  ): AppMachineAccessor<TContext, TEvent, TGetter>
  id: string
  stateMachine: StateMachine<TContext, TSchema, TEvent>
  getters: MachineGetters.Definitions<TGetter, TContext, TEvent>
}
```

So many generics!

Even more of them
And
… many repetitions



```
export interface AppInterpretedMachine<
    ID_TYPE extends string,
    TContext, TSchema,
    TEvent extends EventObject, TGetter extends { [x: string]: any }
> {
    appMachine: AppMachine<TContext, TSchema, TEvent, TGetter> & { id: ID_TYPE; };
    interpreter: Interpreter<TContext, TSchema, TEvent> & { id: ID_TYPE; };
}

export function make<
    ID_TYPE extends string,
    TContext, TSchema,
    TEvent extends EventObject, TGetter extends { [x: string]: any }
>(
    appMachine: AppMachine<TContext, TSchema, TEvent, TGetter> & { id: ID_TYPE },
    interpreter: Interpreter<TContext, TSchema, TEvent> & { id: ID_TYPE; },
): AppInterpretedMachine<ID_TYPE, TContext, TSchema, TEvent, TGetter> {
    return Object.freeze({
        appMachine,
        interpreter,
    });
}
```

Let’s refactor!

```typescript
export interface AppMachine<
  TContext,
  TSchema,
  TEvent extends EventObject,
  TGetter extends { [x: string]: any }
> {
  (
    vueInstance: Vue,
  ): AppMachineAccessor<TContext, TEvent, TGetter>
  id: string
  stateMachine: StateMachine<TContext, TSchema, TEvent>
  getters: MachineGetters.Definitions<TGetter, TContext, TEvent>
}

export type AppMachineInfer<T> = T extends AppMachine<
  infer TContext,
  infer TSchema,
  infer TEvent,
  infer TGetter
>
  ? {
      context: TContext
      schema: TSchema
      event: TEvent
      getter: TGetter
    }
  : never
```

**Type container**

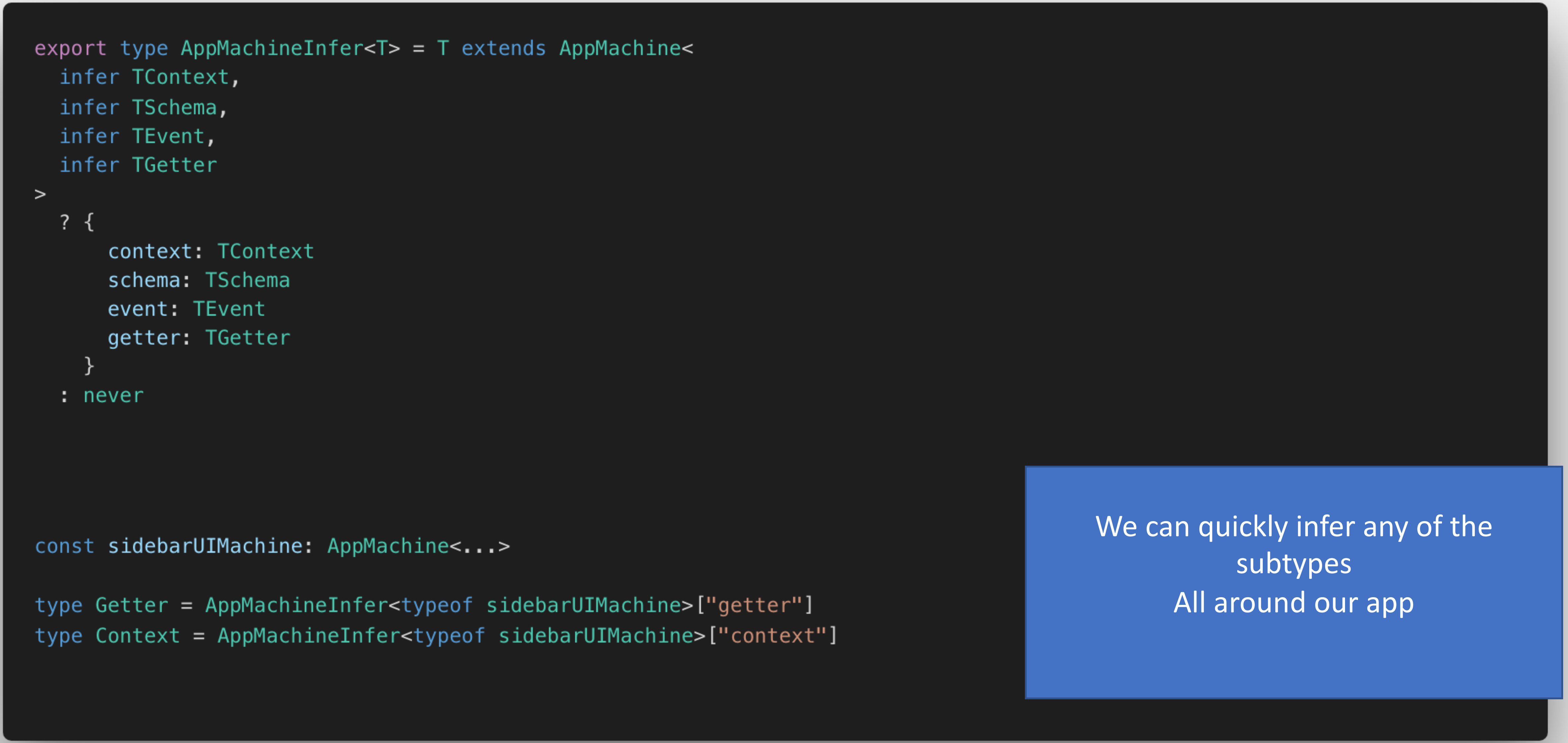
For given AppMachine returns a type

Of an object

That will never exist

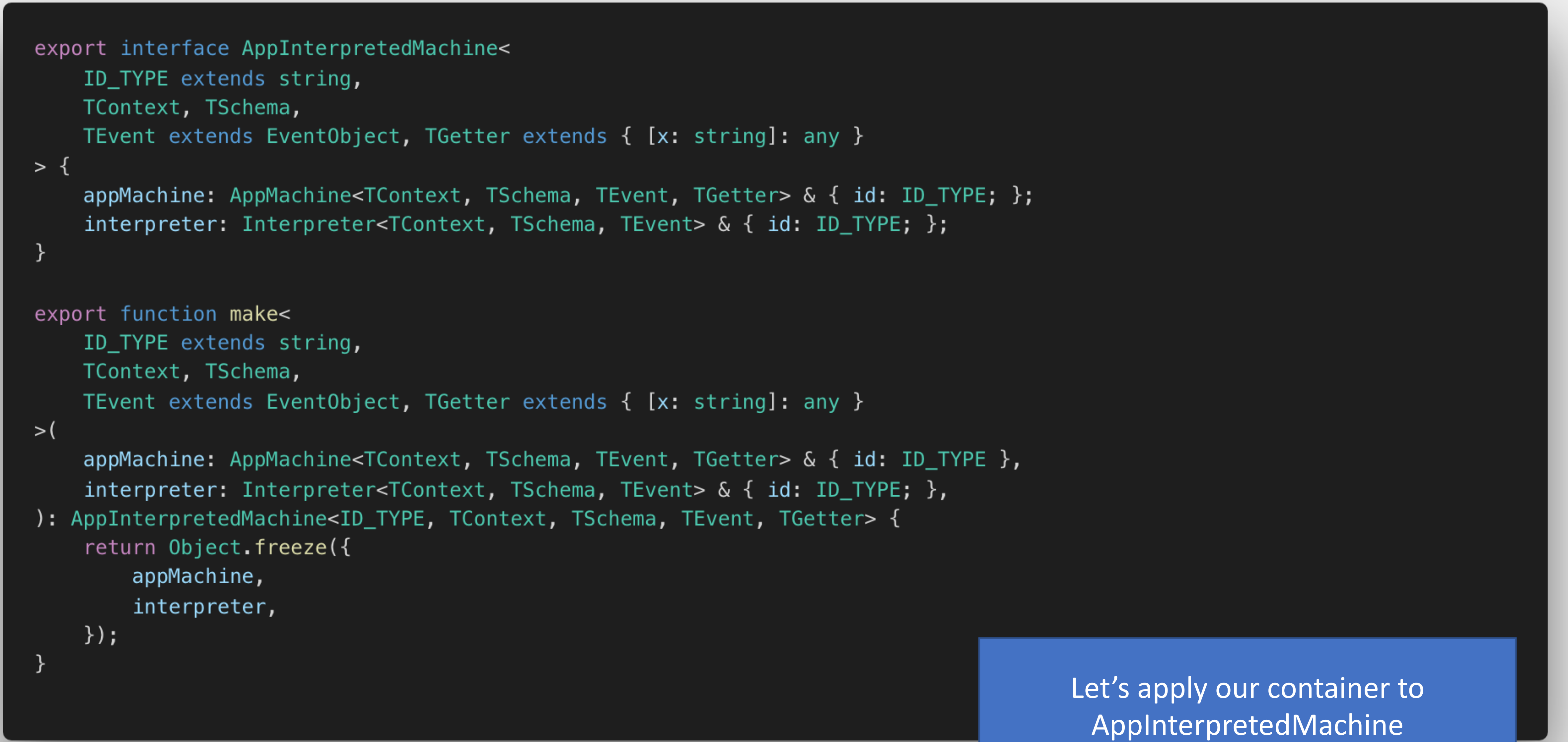
… but if it existed
It would hold a types for all generics

(easily accessible via lookups)



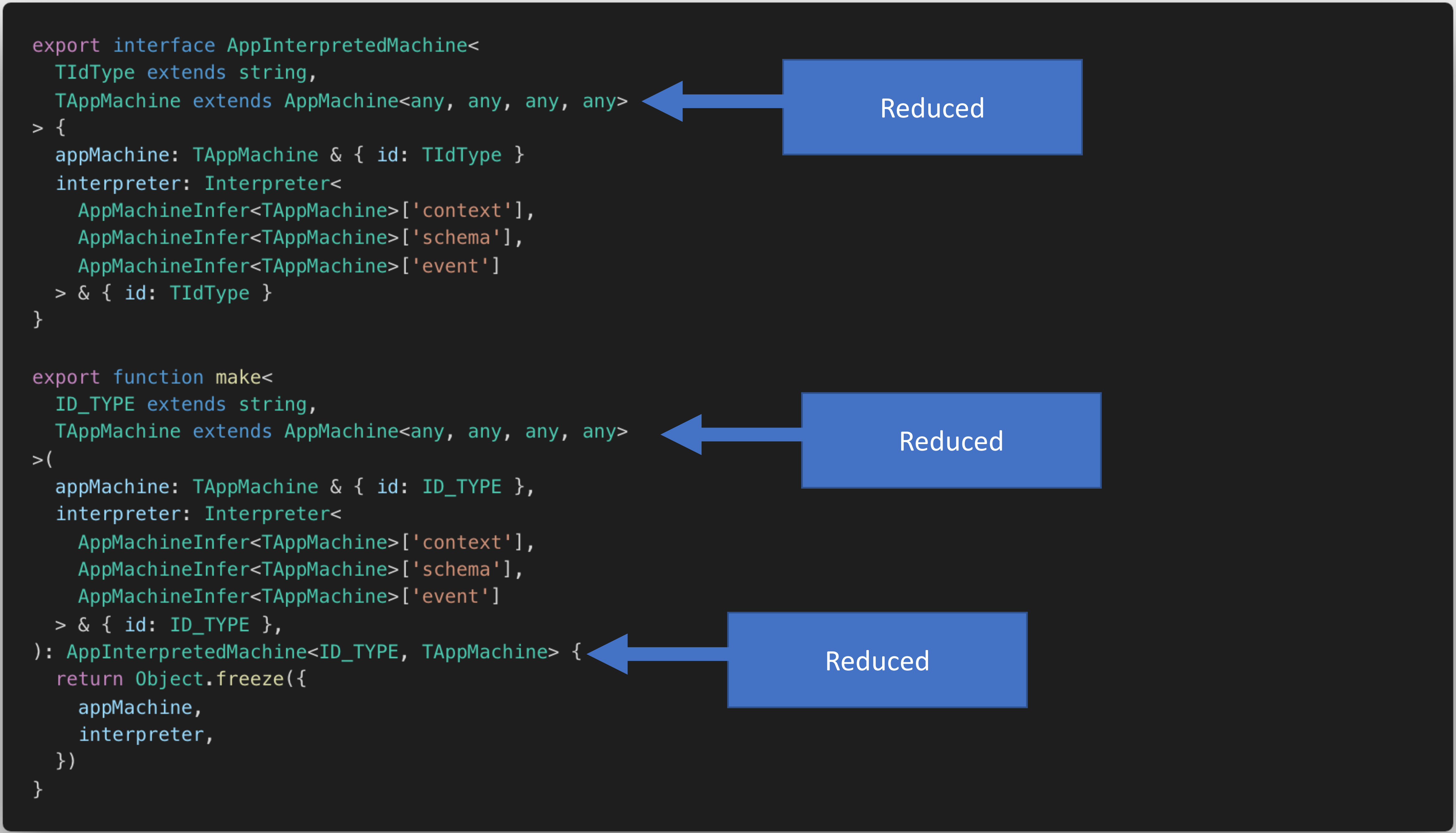
```
export type AppMachineInfer<T> = T extends AppMachine<
  infer TContext,
  infer TSchema,
  infer TEvent,
  infer TGetter
>
  ? {
      context: TContext
      schema: TSchema
      event: TEvent
      getter: TGetter
    }
  : never



const sidebarUIMachine: AppMachine<...>

type Getter = AppMachineInfer<typeof sidebarUIMachine>["getter"]
type Context = AppMachineInfer<typeof sidebarUIMachine>["context"]
```

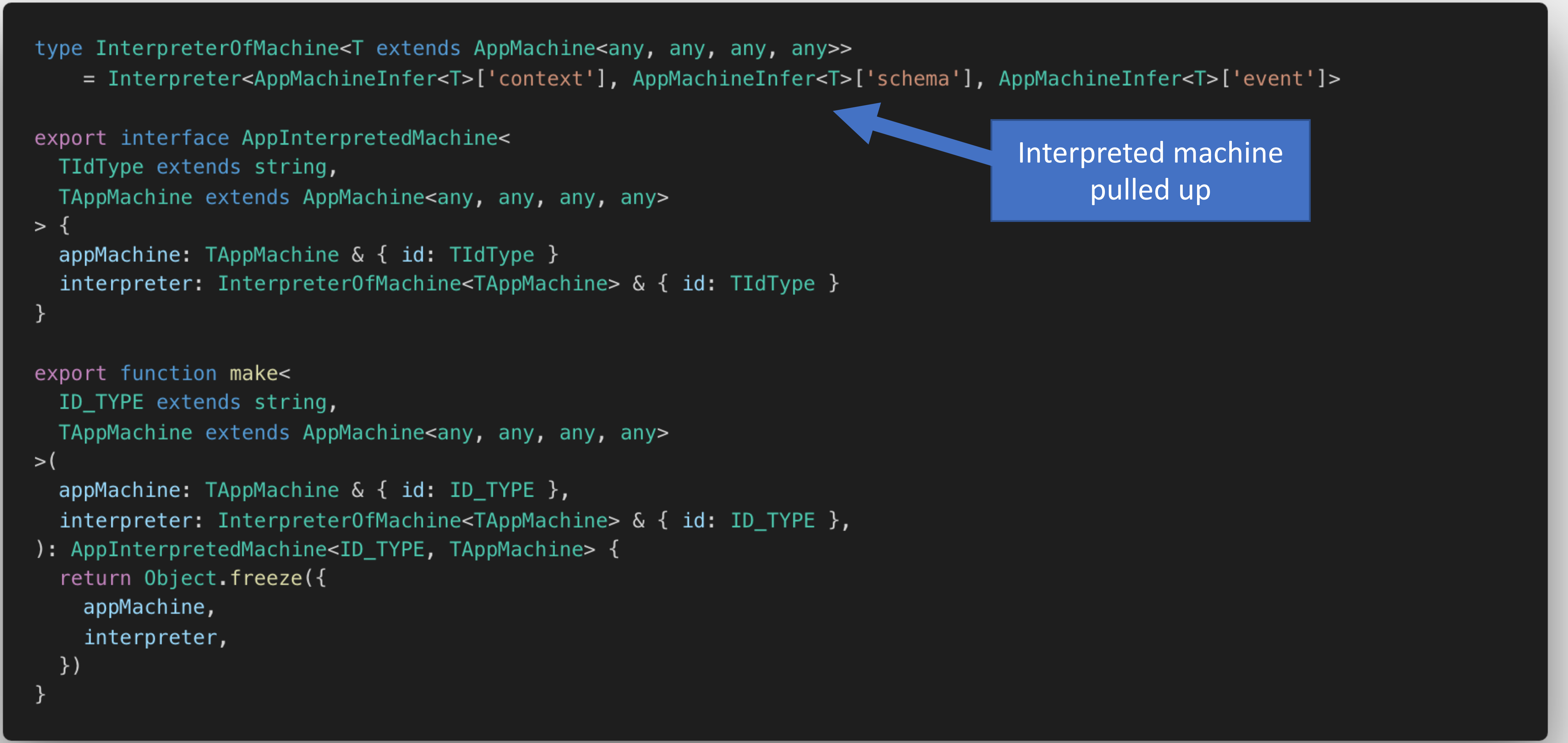
We can quickly infer any of the subtypes
All around our app



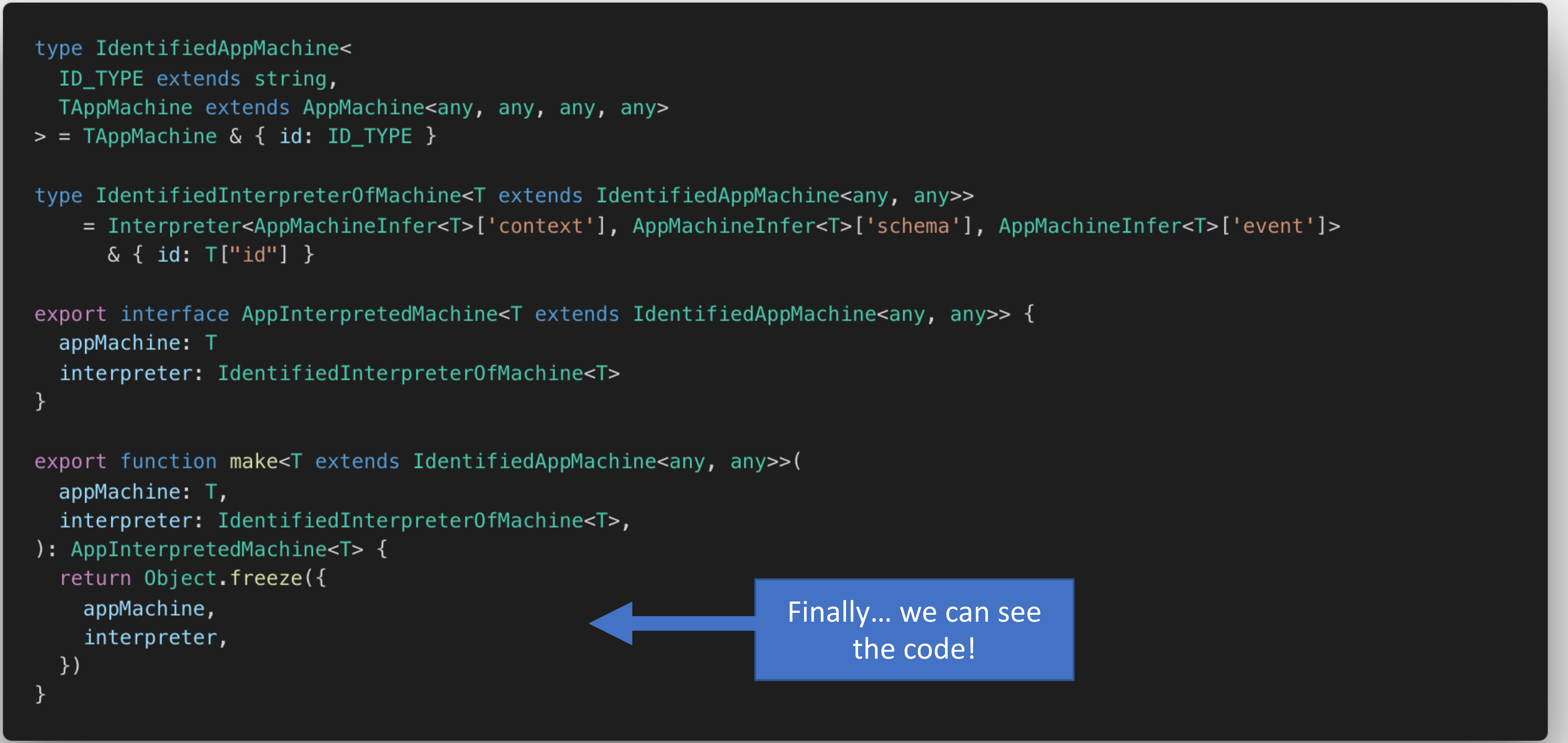
```typescript
export interface AppInterpretedMachine<
    ID_TYPE extends string,
    TContext, TSchema,
    TEvent extends EventObject, TGetter extends { [x: string]: any }
> {
    appMachine: AppMachine<TContext, TSchema, TEvent, TGetter> & { id: ID_TYPE; };
    interpreter: Interpreter<TContext, TSchema, TEvent> & { id: ID_TYPE; };
}

export function make<
    ID_TYPE extends string,
    TContext, TSchema,
    TEvent extends EventObject, TGetter extends { [x: string]: any }
>(
    appMachine: AppMachine<TContext, TSchema, TEvent, TGetter> & { id: ID_TYPE },
    interpreter: Interpreter<TContext, TSchema, TEvent> & { id: ID_TYPE; },
): AppInterpretedMachine<ID_TYPE, TContext, TSchema, TEvent, TGetter> {
    return Object.freeze({
        appMachine,
        interpreter,
    });
}
```

Let’s apply our container to AppInterpretedMachine


```
export interface AppInterpretedMachine<
  TIdType extends string,
  TAppMachine extends AppMachine<any, any, any, any>
> {
  appMachine: TAppMachine & { id: TIdType }
  interpreter: Interpreter<
    AppMachineInfer<TAppMachine>['context'],
    AppMachineInfer<TAppMachine>['schema'],
    AppMachineInfer<TAppMachine>['event']
  > & { id: TIdType }
}

export function make<
  ID_TYPE extends string,
  TAppMachine extends AppMachine<any, any, any, any>
>(
  appMachine: TAppMachine & { id: ID_TYPE },
  interpreter: Interpreter<
    AppMachineInfer<TAppMachine>['context'],
    AppMachineInfer<TAppMachine>['schema'],
    AppMachineInfer<TAppMachine>['event']
  > & { id: ID_TYPE },
): AppInterpretedMachine<ID_TYPE, TAppMachine> {
  return Object.freeze({
    appMachine,
    interpreter,
  })
}
```



```
type InterpreterOfMachine<T extends AppMachine<any, any, any, any>>
    = Interpreter<AppMachineInfer<T>['context'], AppMachineInfer<T>['schema'], AppMachineInfer<T>['event']>

export interface AppInterpretedMachine<
  TIdType extends string,
  TAppMachine extends AppMachine<any, any, any, any>
> {
  appMachine: TAppMachine & { id: TIdType }
  interpreter: InterpreterOfMachine<TAppMachine> & { id: TIdType }
}

export function make<
  ID_TYPE extends string,
  TAppMachine extends AppMachine<any, any, any, any>
>(
  appMachine: TAppMachine & { id: ID_TYPE },
  interpreter: InterpreterOfMachine<TAppMachine> & { id: ID_TYPE },
): AppInterpretedMachine<ID_TYPE, TAppMachine> {
  return Object.freeze({
    appMachine,
    interpreter,
  })
}
```



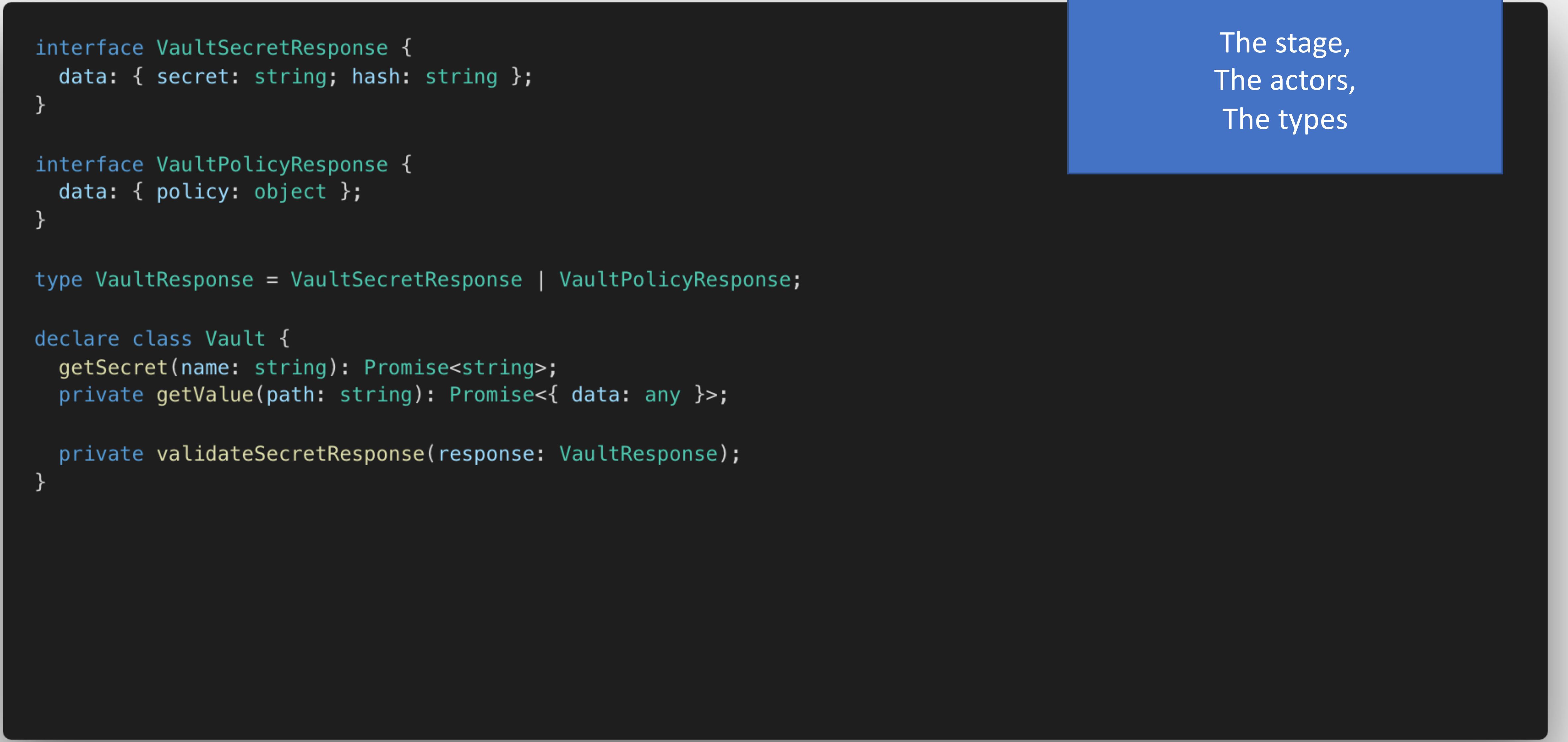
```
type IdentifiedAppMachine<
  ID_TYPE extends string,
  TAppMachine extends AppMachine<any, any, any, any>
> = TAppMachine & { id: ID_TYPE }

type IdentifiedInterpreterOfMachine<T extends IdentifiedAppMachine<any, any>>
    = Interpreter<AppMachineInfer<T>['context'], AppMachineInfer<T>['schema'], AppMachineInfer<T>['event']>
      & { id: T["id"] }

export interface AppInterpretedMachine<T extends IdentifiedAppMachine<any, any>> {
  appMachine: T
  interpreter: IdentifiedInterpreterOfMachine<T>
}

export function make<T extends IdentifiedAppMachine<any, any>>(
  appMachine: T,
  interpreter: IdentifiedInterpreterOfMachine<T>,
): AppInterpretedMachine<T> {
  return Object.freeze({
    appMachine,
    interpreter,
  })
}
```

Finally… we can see the code!

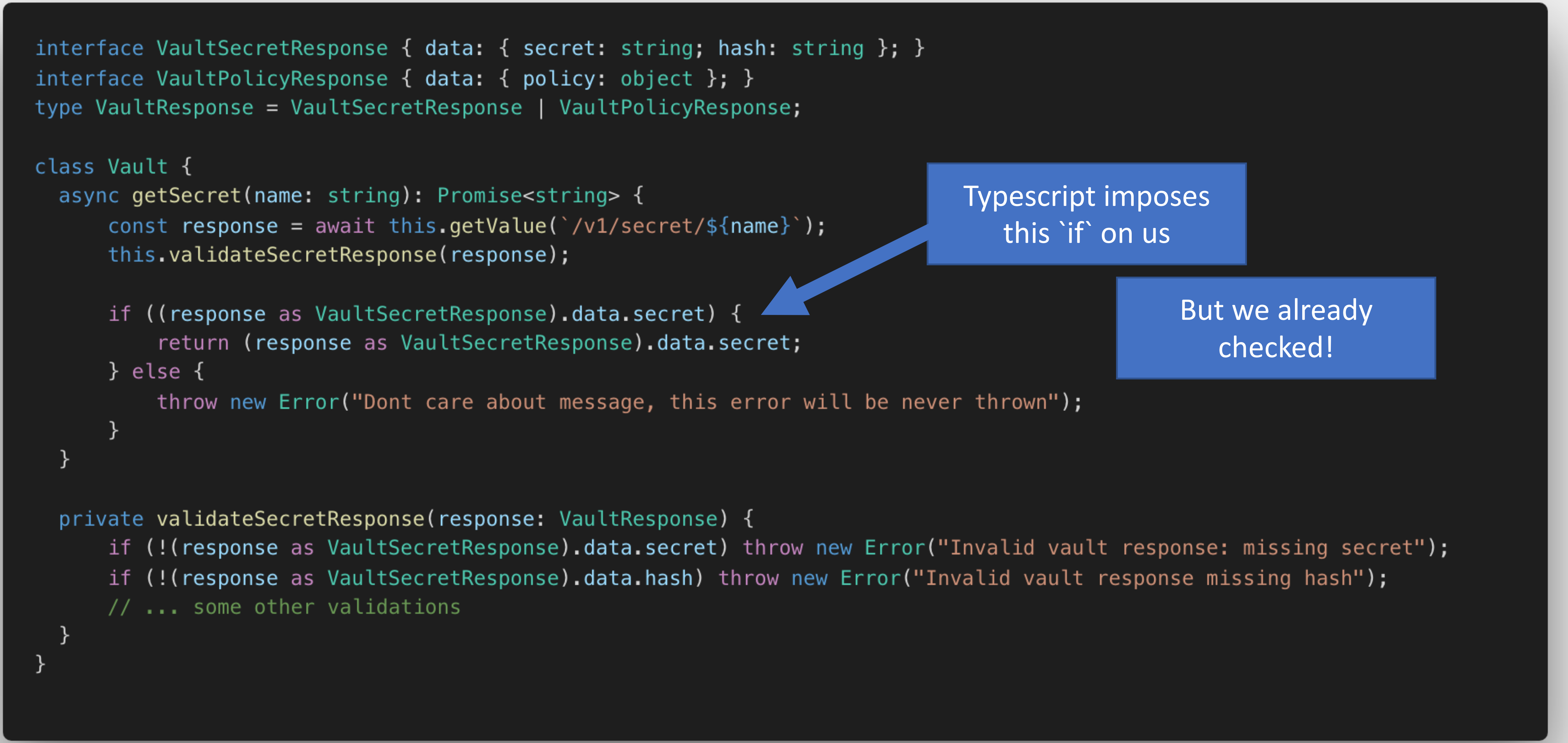
Fight repetitions with…

## Assertion functions (new in TS 3.7)

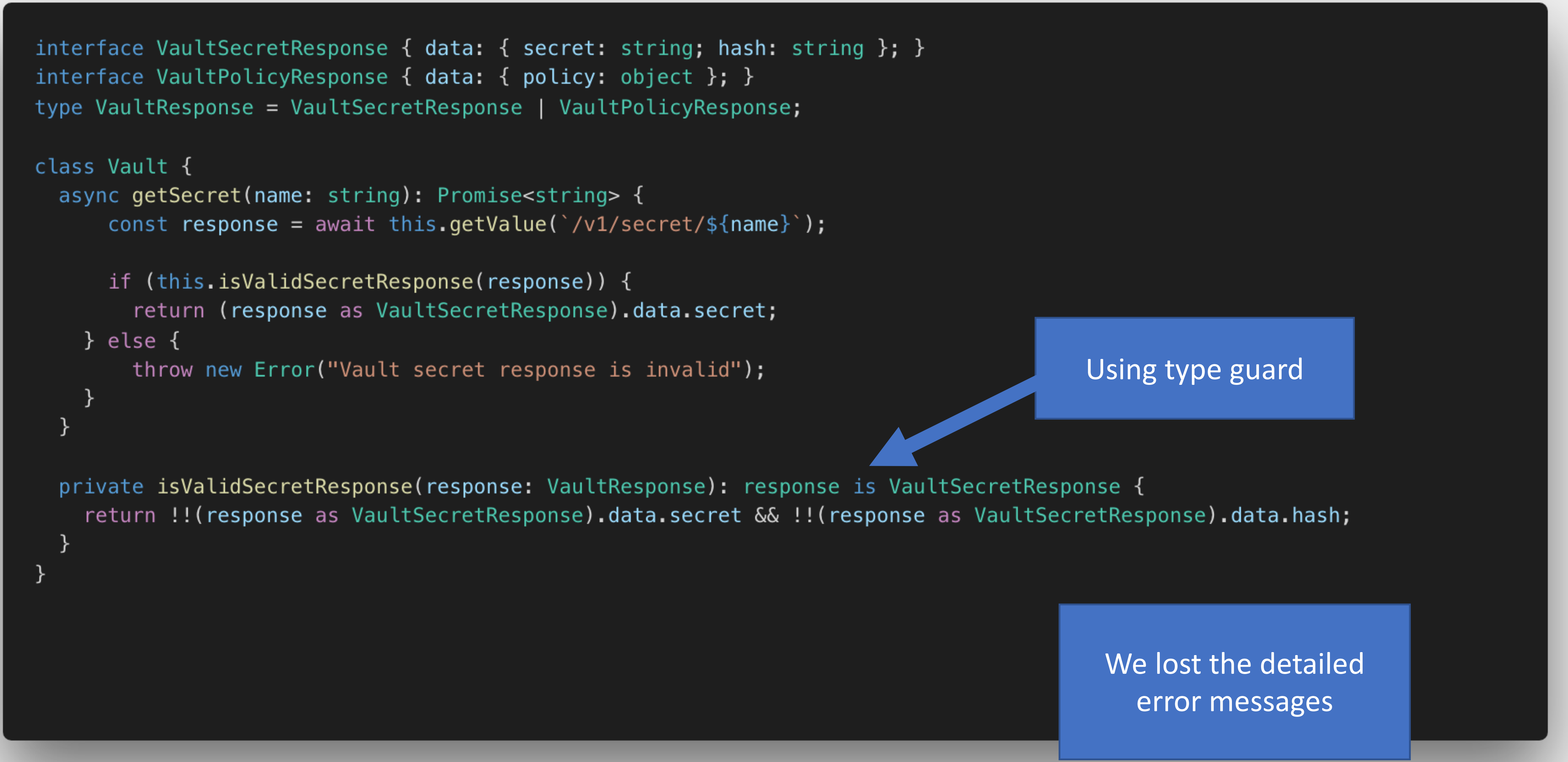
The stage,
The actors,
The types



```
interface VaultSecretResponse {
  data: { secret: string; hash: string };
}

interface VaultPolicyResponse {
  data: { policy: object };
}

type VaultResponse = VaultSecretResponse | VaultPolicyResponse;

declare class Vault {
  getSecret(name: string): Promise<string>;
  private getValue(path: string): Promise<{ data: any }>;

  private validateSecretResponse(response: VaultResponse);
}
```



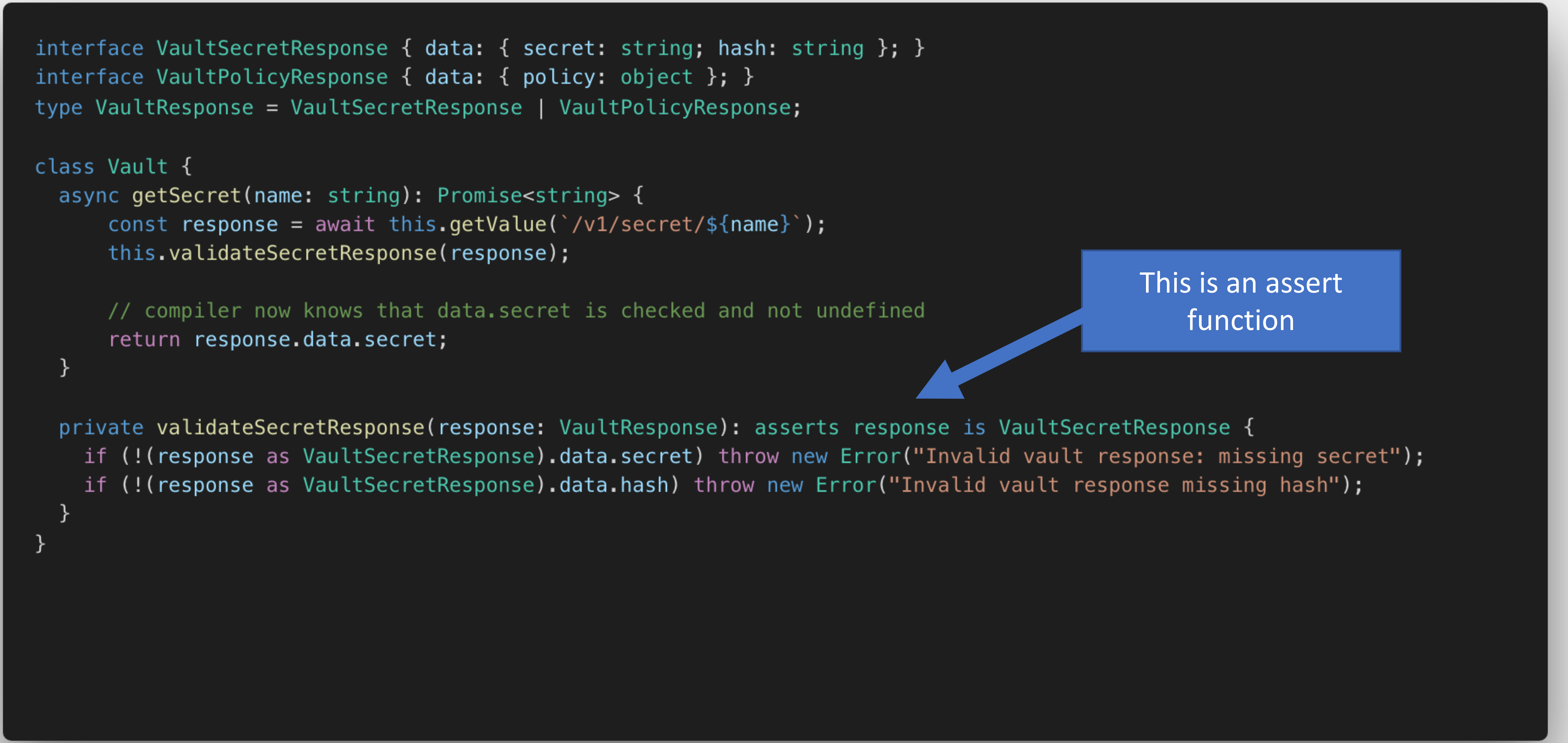
```
interface VaultSecretResponse { data: { secret: string; hash: string }; }
interface VaultPolicyResponse { data: { policy: object }; }
type VaultResponse = VaultSecretResponse | VaultPolicyResponse;

class Vault {
  async getSecret(name: string): Promise<string> {
      const response = await this.getValue(`/v1/secret/${name}`);
      this.validateSecretResponse(response);

      if ((response as VaultSecretResponse).data.secret) {
          return (response as VaultSecretResponse).data.secret;
      } else {
          throw new Error("Dont care about message, this error will be never thrown");
      }
  }

  private validateSecretResponse(response: VaultResponse) {
      if (!(response as VaultSecretResponse).data.secret) throw new Error("Invalid vault response: missing secret");
      if (!(response as VaultSecretResponse).data.hash) throw new Error("Invalid vault response missing hash");
      // ... some other validations
  }
}
```



```
interface VaultSecretResponse { data: { secret: string; hash: string }; }
interface VaultPolicyResponse { data: { policy: object }; }
type VaultResponse = VaultSecretResponse | VaultPolicyResponse;

class Vault {
  async getSecret(name: string): Promise<string> {
      const response = await this.getValue(`/v1/secret/${name}`);

      if (this.isValidSecretResponse(response)) {
        return (response as VaultSecretResponse).data.secret;
    } else {
        throw new Error("Vault secret response is invalid");
    }
  }

  private isValidSecretResponse(response: VaultResponse): response is VaultSecretResponse {
    return !!(response as VaultSecretResponse).data.secret && !!(response as VaultSecretResponse).data.hash;
  }
}
```

Using type guard

We lost the detailed error messages



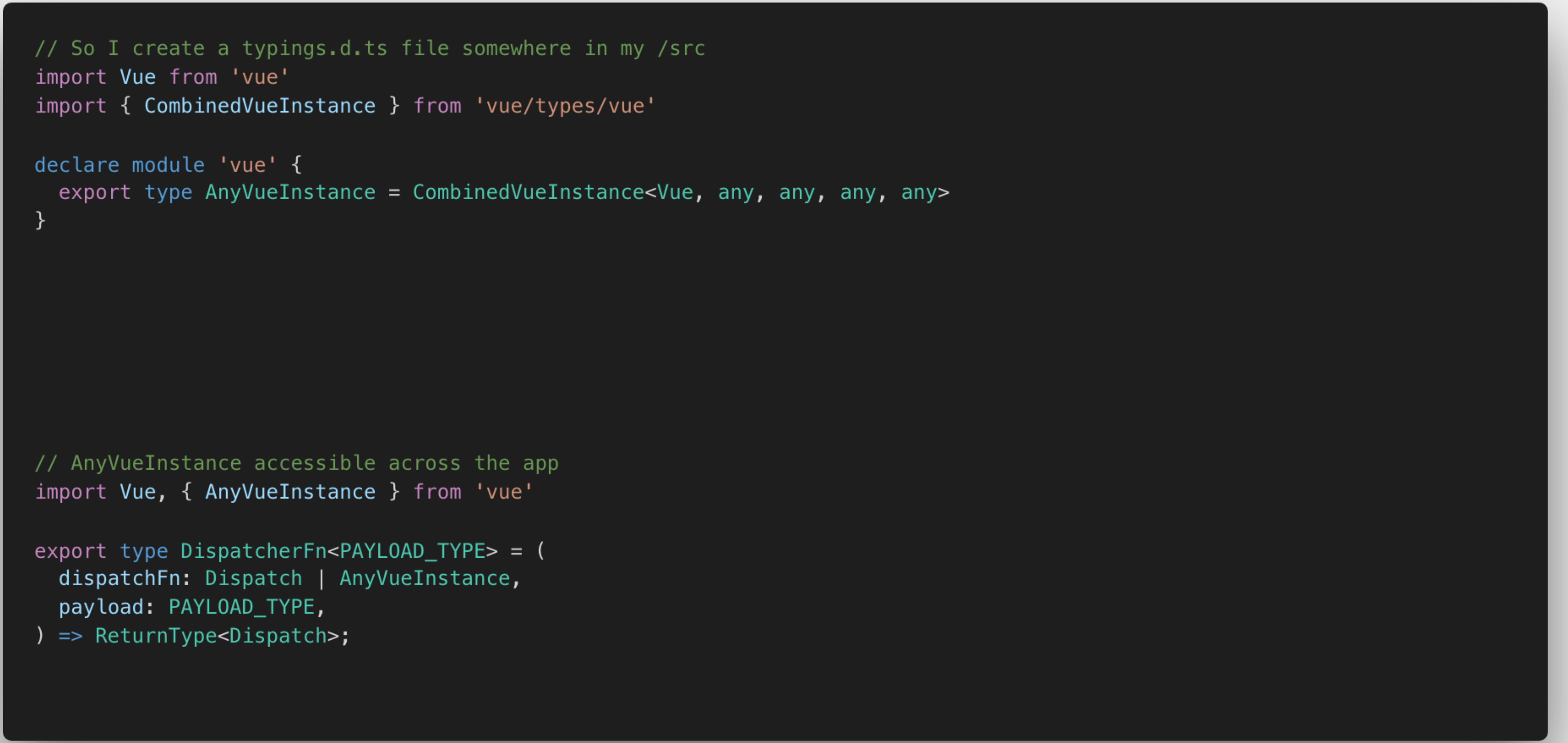
```
interface VaultSecretResponse { data: { secret: string; hash: string }; }
interface VaultPolicyResponse { data: { policy: object }; }
type VaultResponse = VaultSecretResponse | VaultPolicyResponse;

class Vault {
  async getSecret(name: string): Promise<string> {
      const response = await this.getValue(`/v1/secret/${name}`);
      this.validateSecretResponse(response);

      // compiler now knows that data.secret is checked and not undefined
      return response.data.secret;
  }

  private validateSecretResponse(response: VaultResponse): asserts response is VaultSecretResponse {
    if (!(response as VaultSecretResponse).data.secret) throw new Error("Invalid vault response: missing secret");
    if (!(response as VaultSecretResponse).data.hash) throw new Error("Invalid vault response missing hash");
  }
}
```

This is an assert function

Fight repetitions with…

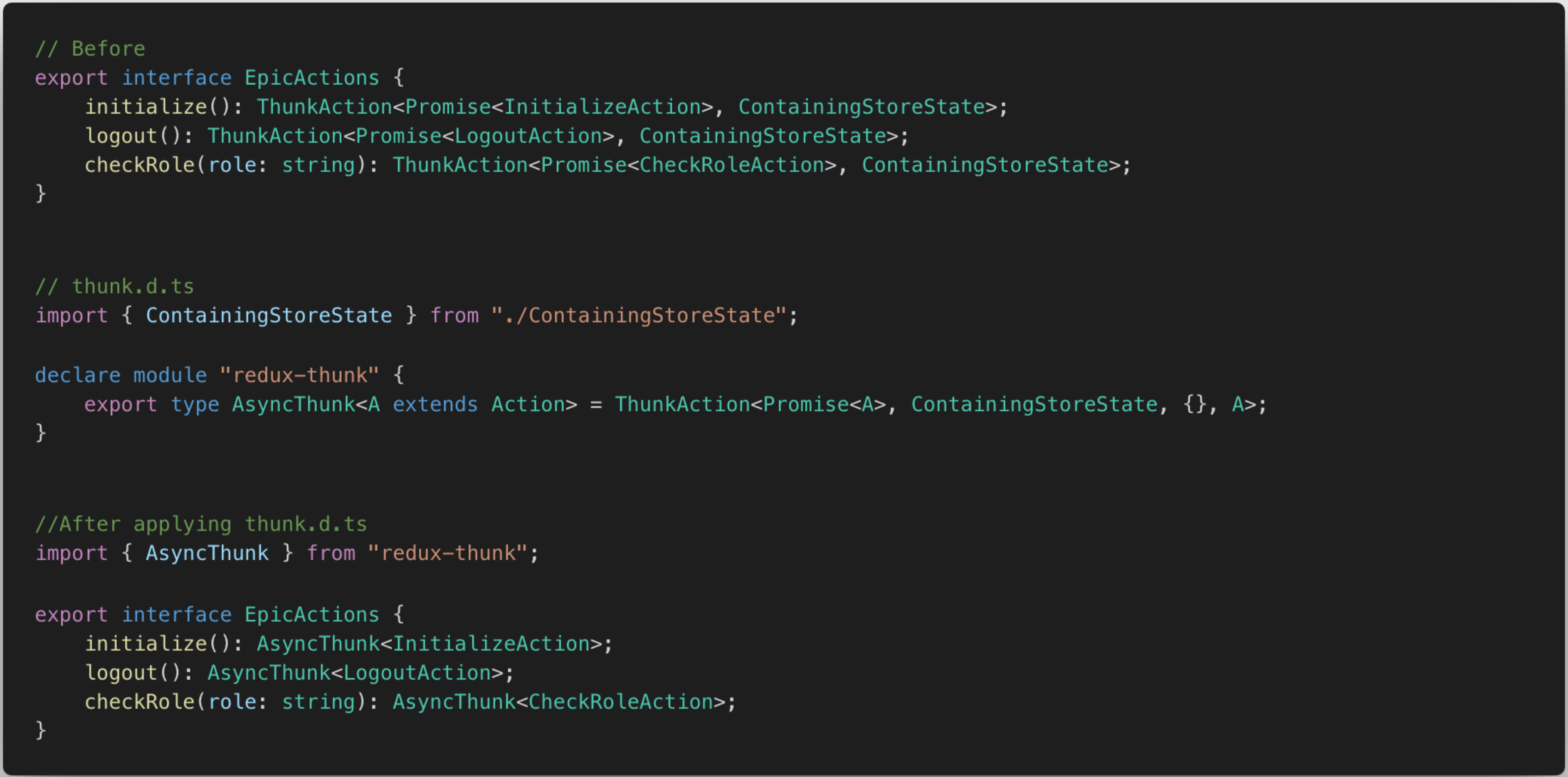
## Appending types to external libraries

```
// I am sometimes using it a lot
CombinedVueInstance<any, any, any, any, any>
```

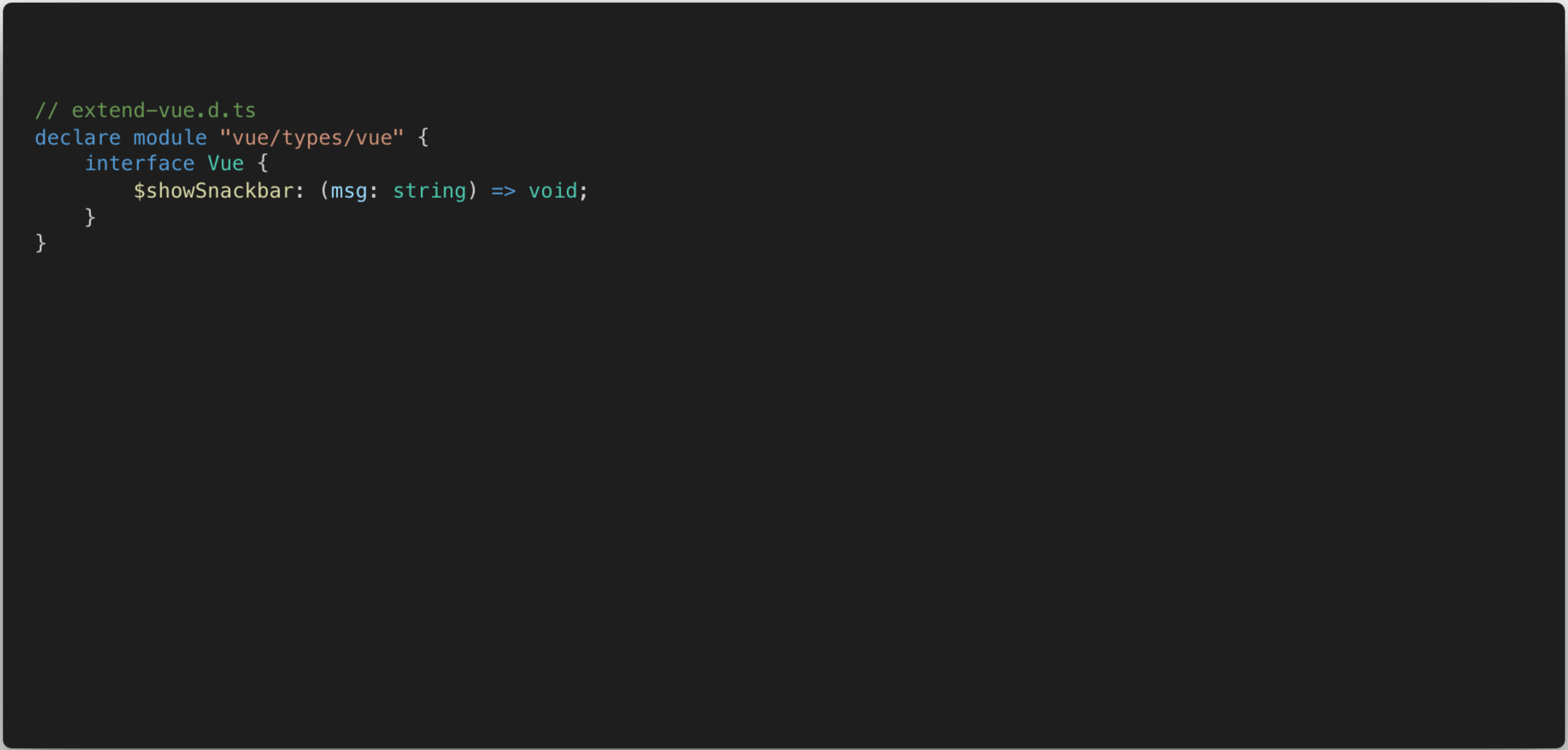


```typescript
// So I create a typings.d.ts file somewhere in my /src
import Vue from 'vue'
import { CombinedVueInstance } from 'vue/types/vue'

declare module 'vue' {
  export type AnyVueInstance = CombinedVueInstance<Vue, any, any, any, any>
}
```

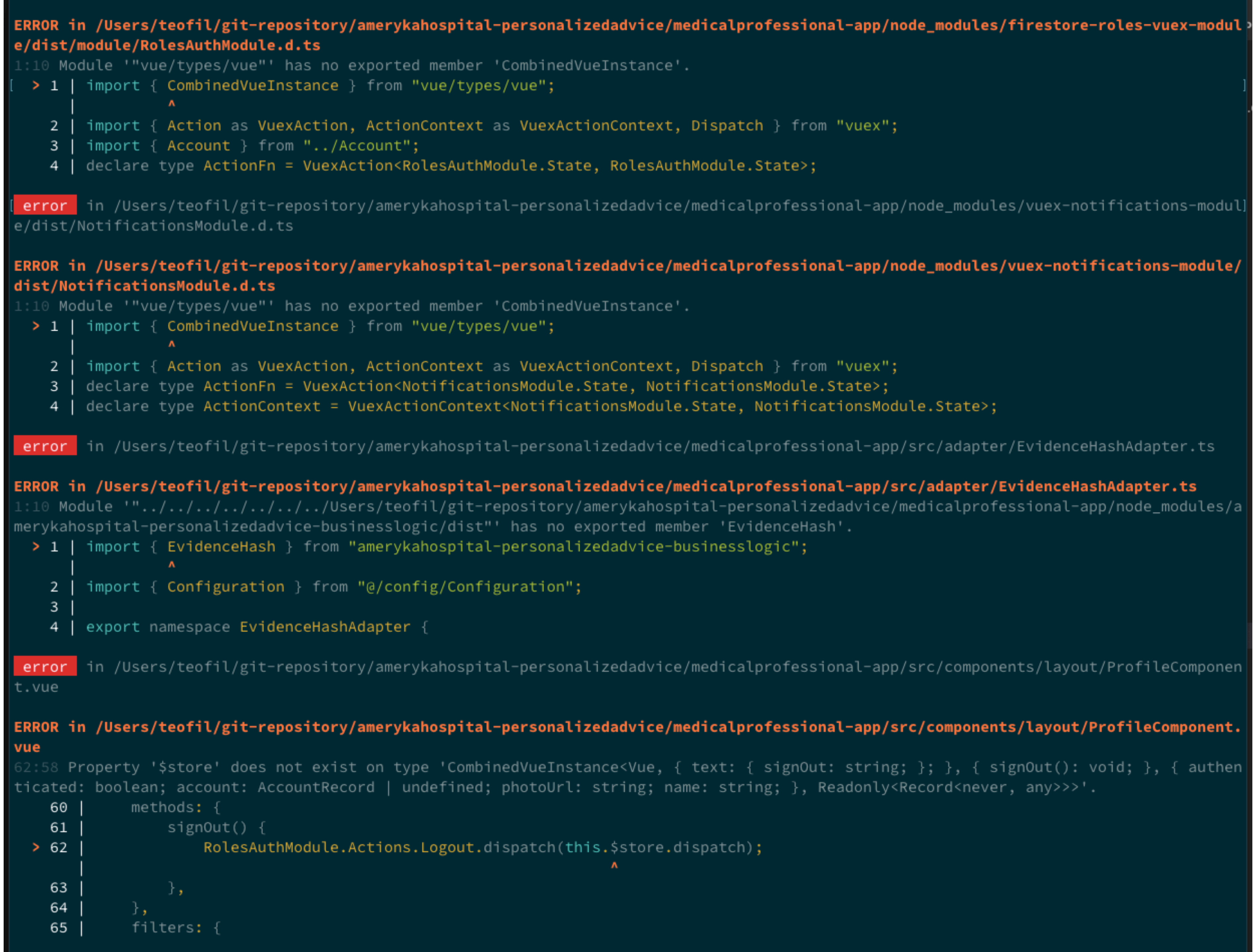
```typescript
// AnyVueInstance accessible across the app
import Vue, { AnyVueInstance } from 'vue'

export type DispatcherFn<PAYLOAD_TYPE> = (
  dispatchFn: Dispatch | AnyVueInstance,
  payload: PAYLOAD_TYPE,
) => ReturnType<Dispatch>;
```



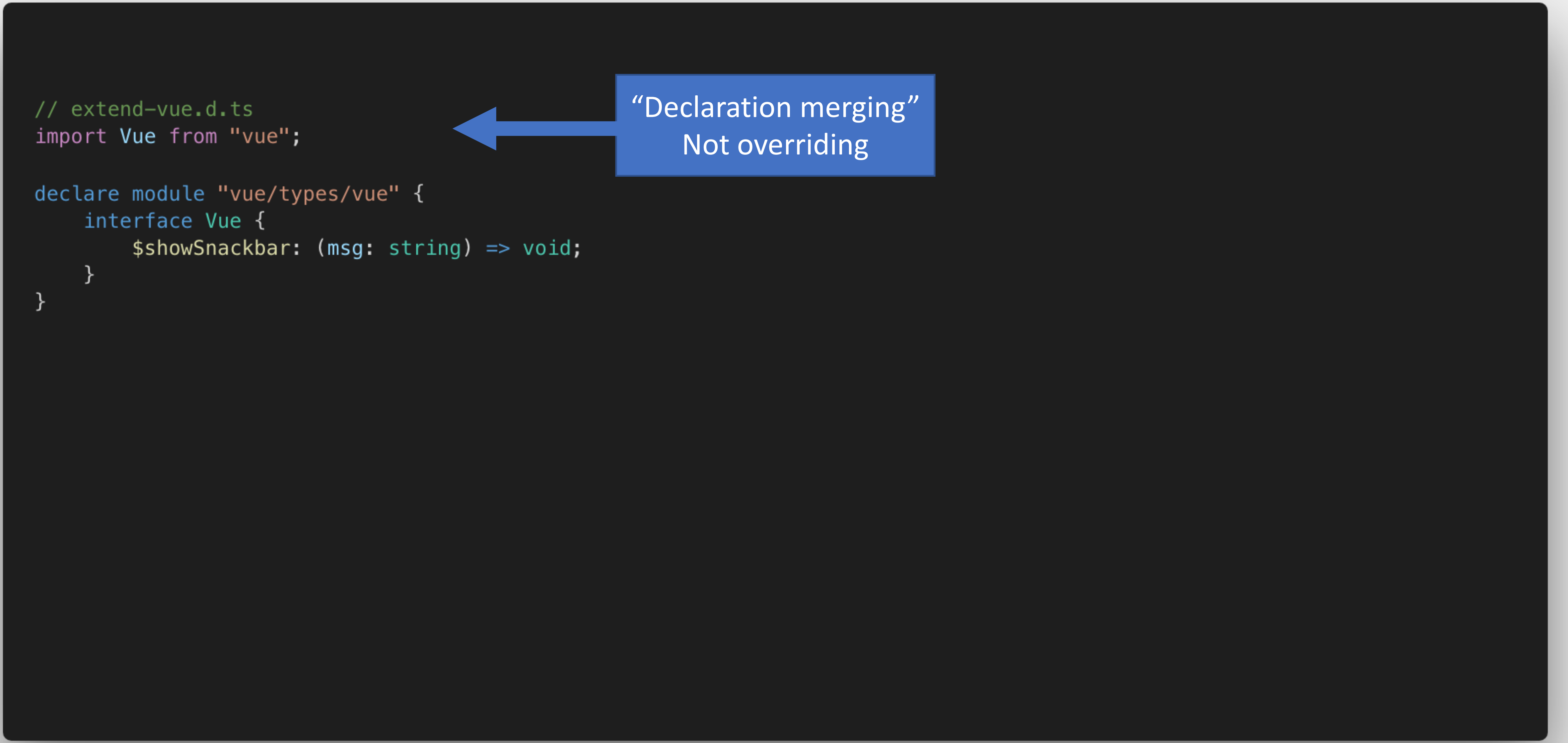
```typescript
// Before
export interface EpicActions {
    initialize(): ThunkAction<Promise<InitializeAction>, ContainingStoreState>;
    logout(): ThunkAction<Promise<LogoutAction>, ContainingStoreState>;
    checkRole(role: string): ThunkAction<Promise<CheckRoleAction>, ContainingStoreState>;
}


// thunk.d.ts
import { ContainingStoreState } from "./ContainingStoreState";

declare module "redux-thunk" {
    export type AsyncThunk<A extends Action> = ThunkAction<Promise<A>, ContainingStoreState, {}, A>;
}


//After applying thunk.d.ts
import { AsyncThunk } from "redux-thunk";

export interface EpicActions {
    initialize(): AsyncThunk<InitializeAction>;
    logout(): AsyncThunk<LogoutAction>;
    checkRole(role: string): AsyncThunk<CheckRoleAction>;
}
```



```
// extend-vue.d.ts
declare module "vue/types/vue" {
    interface Vue {
        $showSnackbar: (msg: string) => void;
    }
}
```



```
ERROR in /Users/teofil/git-repository/amerykahospital-personalizedadvice/medicalprofessional-app/node_modules/firestore-roles-vuex-modul
e/dist/module/RolesAuthModule.d.ts
1:10 Module '"vue/types/vue"' has no exported member 'CombinedVueInstance'.
  > 1 | import { CombinedVueInstance } from "vue/types/vue";
      |          ^
    2 | import { Action as VuexAction, ActionContext as VuexActionContext, Dispatch } from "vuex";
    3 | import { Account } from "../Account";
    4 | declare type ActionFn = VuexAction<RolesAuthModule.State, RolesAuthModule.State>;

 error  in /Users/teofil/git-repository/amerykahospital-personalizedadvice/medicalprofessional-app/node_modules/vuex-notifications-modul
e/dist/NotificationsModule.d.ts

ERROR in /Users/teofil/git-repository/amerykahospital-personalizedadvice/medicalprofessional-app/node_modules/vuex-notifications-module/
dist/NotificationsModule.d.ts
1:10 Module '"vue/types/vue"' has no exported member 'CombinedVueInstance'.
  > 1 | import { CombinedVueInstance } from "vue/types/vue";
      |          ^
    2 | import { Action as VuexAction, ActionContext as VuexActionContext, Dispatch } from "vuex";
    3 | declare type ActionFn = VuexAction<NotificationsModule.State, NotificationsModule.State>;
    4 | declare type ActionContext = VuexActionContext<NotificationsModule.State, NotificationsModule.State>;

 error  in /Users/teofil/git-repository/amerykahospital-personalizedadvice/medicalprofessional-app/src/adapter/EvidenceHashAdapter.ts

ERROR in /Users/teofil/git-repository/amerykahospital-personalizedadvice/medicalprofessional-app/src/adapter/EvidenceHashAdapter.ts
1:10 Module '"../../../../../../../Users/teofil/git-repository/amerykahospital-personalizedadvice/medicalprofessional-app/node_modules/a
merykahospital-personalizedadvice-businesslogic/dist"' has no exported member 'EvidenceHash'.
  > 1 | import { EvidenceHash } from "amerykahospital-personalizedadvice-businesslogic";
      |          ^
    2 | import { Configuration } from "@/config/Configuration";
    3 |
    4 | export namespace EvidenceHashAdapter {

 error  in /Users/teofil/git-repository/amerykahospital-personalizedadvice/medicalprofessional-app/src/components/layout/ProfileComponen
t.vue

ERROR in /Users/teofil/git-repository/amerykahospital-personalizedadvice/medicalprofessional-app/src/components/layout/ProfileComponent.
vue
62:58 Property '$store' does not exist on type 'CombinedVueInstance<Vue, { text: { signOut: string; }; }, { signOut(): void; }, { authen
ticated: boolean; account: AccountRecord | undefined; photoUrl: string; name: string; }, Readonly<Record<never, any>>>'.
    60 |     methods: {
    61 |         signOut() {
  > 62 |             RolesAuthModule.Actions.Logout.dispatch(this.$store.dispatch);
       |                                                          ^
    63 |         },
    64 |     },
    65 |     filters: {
```



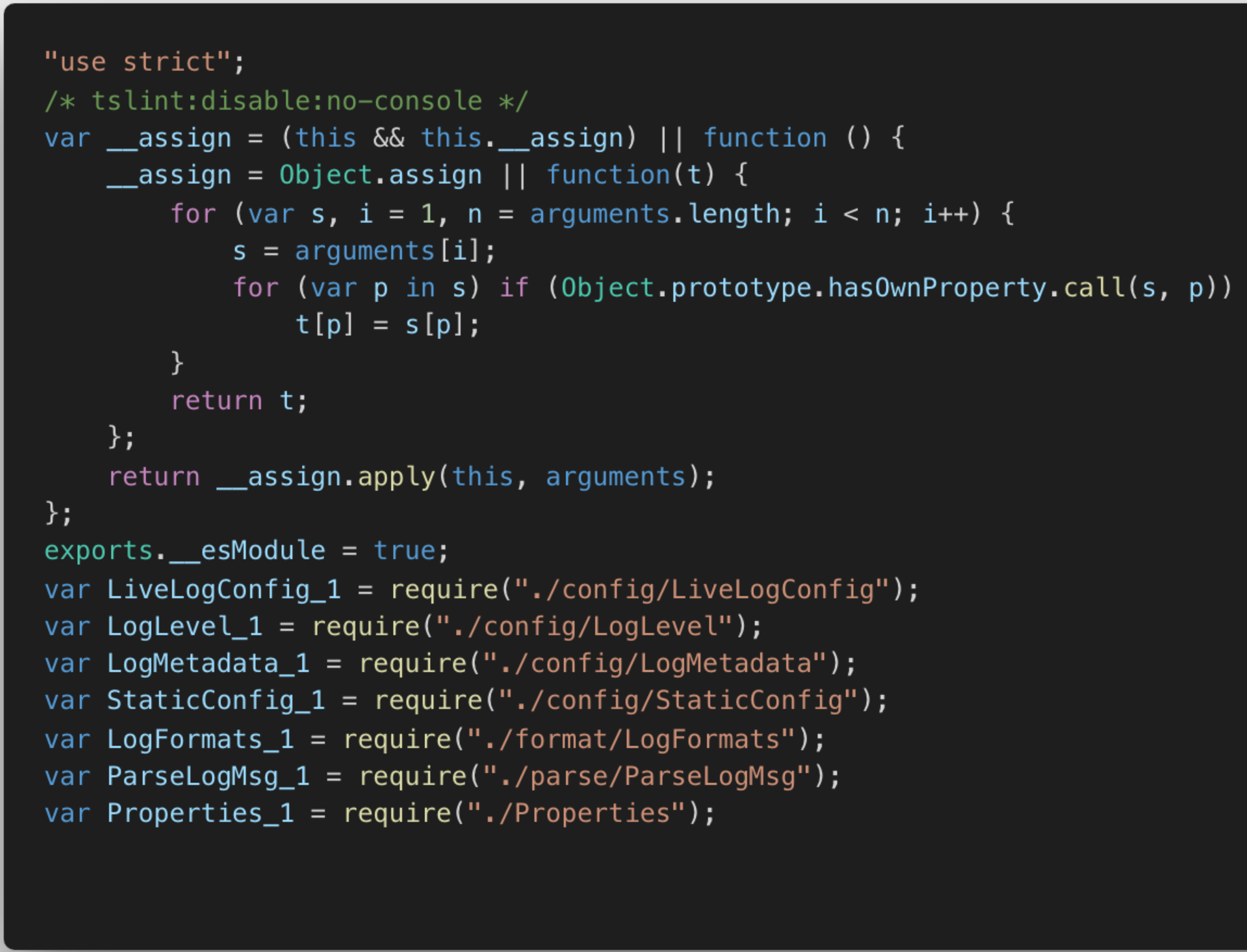
```
// extend-vue.d.ts
import Vue from "vue";

declare module "vue/types/vue" {
    interface Vue {
        $showSnackbar: (msg: string) => void;
    }
}
```

# Optimizing bundle size

In Typescript apps

Optimizing bundle size

## importHelpers

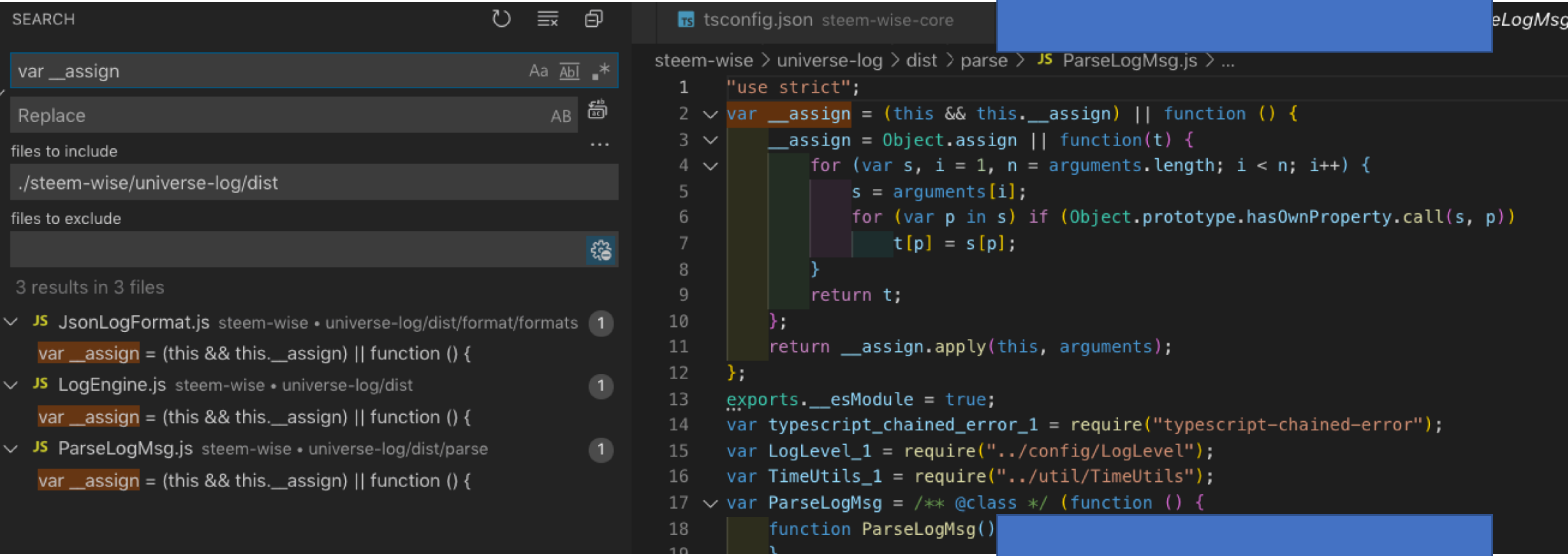


```
"use strict";
/* tslint:disable:no-console */
var __assign = (this && this.__assign) || function () {
    __assign = Object.assign || function(t) {
        for (var s, i = 1, n = arguments.length; i < n; i++) {
            s = arguments[i];
            for (var p in s) if (Object.prototype.hasOwnProperty.call(s, p))
                t[p] = s[p];
        }
        return t;
    };
    return __assign.apply(this, arguments);
};
exports.__esModule = true;
var LiveLogConfig_1 = require("./config/LiveLogConfig");
var LogLevel_1 = require("./config/LogLevel");
var LogMetadata_1 = require("./config/LogMetadata");
var StaticConfig_1 = require("./config/StaticConfig");
var LogFormats_1 = require("./format/LogFormats");
var ParseLogMsg_1 = require("./parse/ParseLogMsg");
var Properties_1 = require("./Properties");
```

__assign is one of the import helpers

Tsc appends them to the output

This library has only 19 files



SEARCH

var __assign

Replace

files to include

./steem-wise/universe-log/dist

files to exclude

3 results in 3 files

JsonLogFormat.js steem-wise • universe-log/dist/format/formats 1

var __assign = (this && this.__assign) || function () {

LogEngine.js steem-wise • universe-log/dist 1

var __assign = (this && this.__assign) || function () {

ParseLogMsg.js steem-wise • universe-log/dist/parse 1

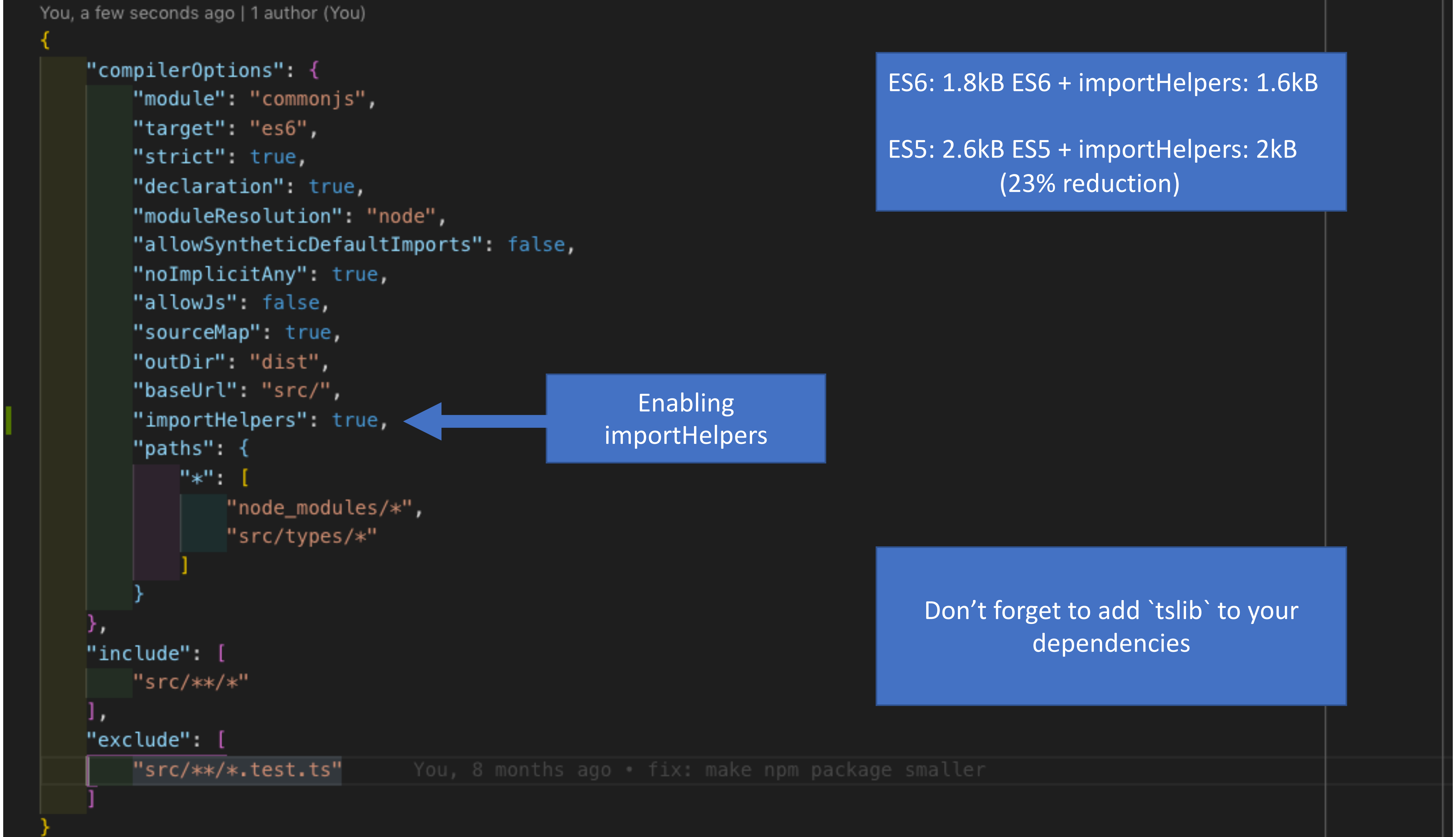
var __assign = (this && this.__assign) || function () {

tsconfig.json steem-wise-core

steem-wise > universe-log > dist > parse > ParseLogMsg.js > ...

```
"use strict";
var __assign = (this && this.__assign) || function () {
    __assign = Object.assign || function(t) {
        for (var s, i = 1, n = arguments.length; i < n; i++) {
            s = arguments[i];
            for (var p in s) if (Object.prototype.hasOwnProperty.call(s, p))
                t[p] = s[p];
        }
        return t;
    };
    return __assign.apply(this, arguments);
};
exports.__esModule = true;
var typescript_chained_error_1 = require("typescript-chained-error");
var LogLevel_1 = require("../config/LogLevel");
var TimeUtils_1 = require("../util/TimeUtils");
var ParseLogMsg = /** @class */ (function () {
    function ParseLogMsg()
```

__assign helper outputted 3 times!



You, a few seconds ago | 1 author (You)

```
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es6",
    "strict": true,
    "declaration": true,
    "moduleResolution": "node",
    "allowSyntheticDefaultImports": false,
    "noImplicitAny": true,
    "allowJs": false,
    "sourceMap": true,
    "outDir": "dist",
    "baseUrl": "src/",
    "importHelpers": true,
    "paths": {
      "*": [
        "node_modules/*",
        "src/types/*"
      ]
    }
  },
  "include": [
    "src/**/*"
  ],
  "exclude": [
    "src/**/*.test.ts"      You, 8 months ago • fix: make npm package smaller
  ]
}
```

ES6: 1.8kB ES6 + importHelpers: 1.6kB

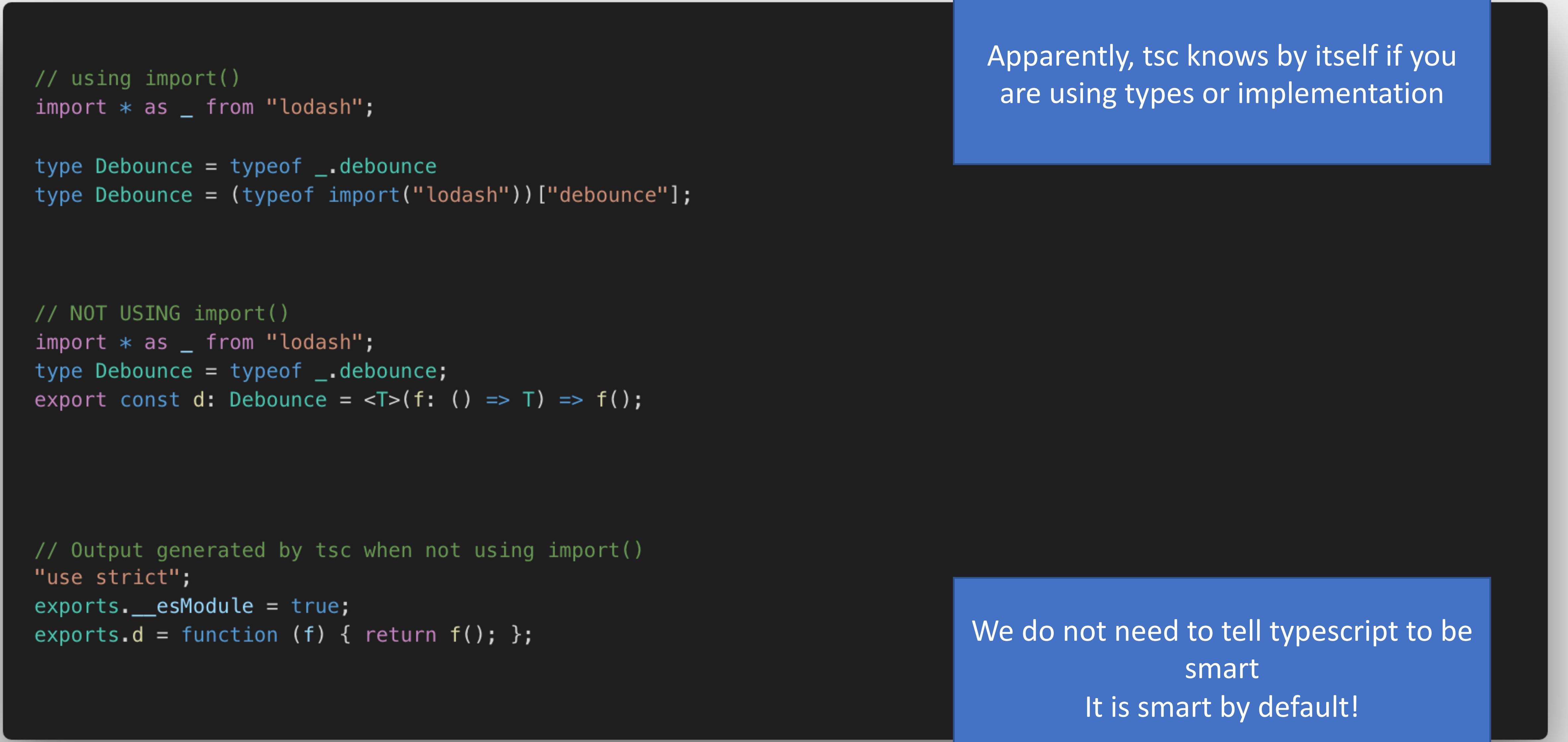
ES5: 2.6kB ES5 + importHelpers: 2kB (23% reduction)

Enabling importHelpers

Don’t forget to add `tslib` to your dependencies

Optimizing bundle size

## importing() only types



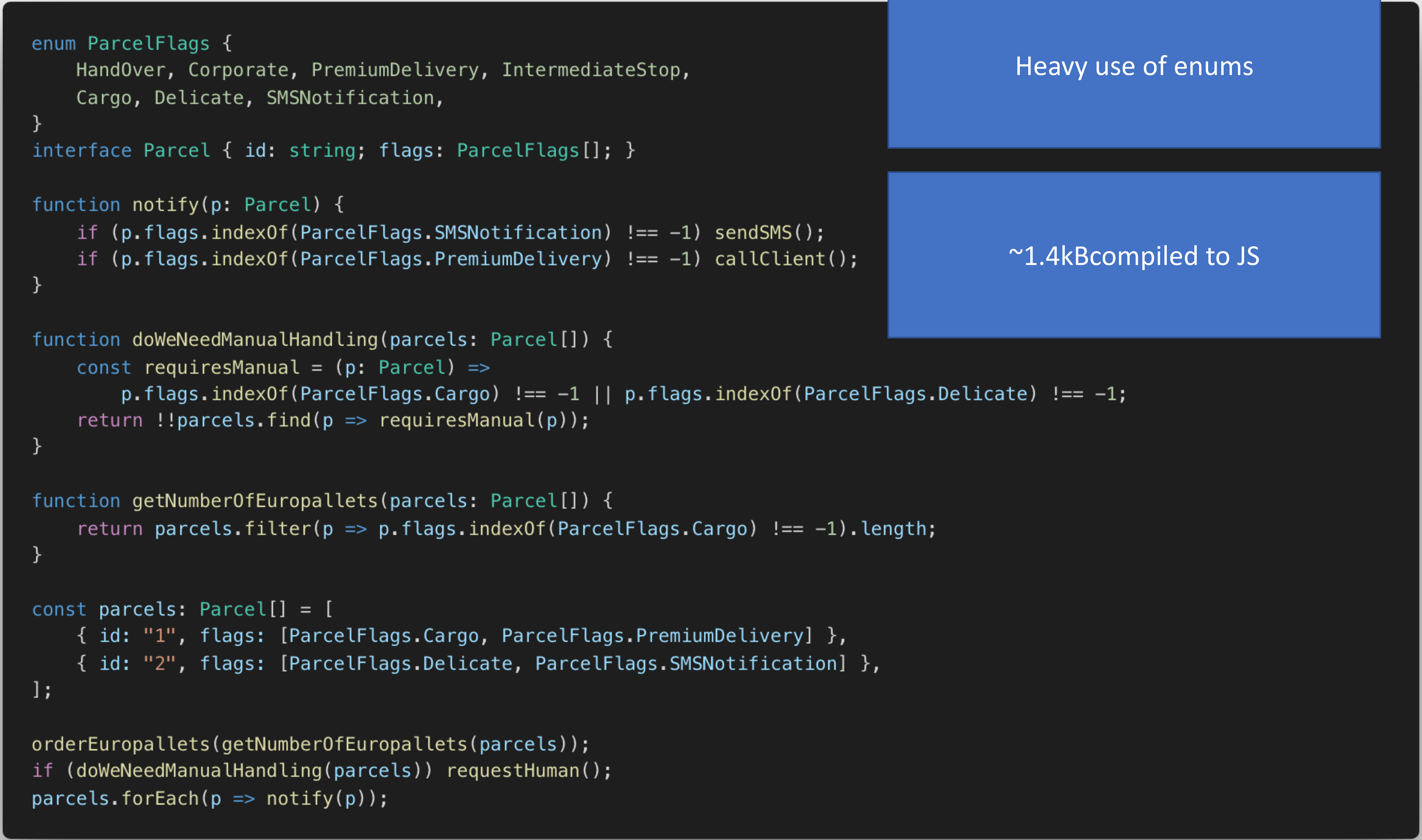
```
// using import()
import * as _ from "lodash";

type Debounce = typeof _.debounce
type Debounce = (typeof import("lodash"))["debounce"];



// NOT USING import()
import * as _ from "lodash";
type Debounce = typeof _.debounce;
export const d: Debounce = <T>(f: () => T) => f();



// Output generated by tsc when not using import()
"use strict";
exports.__esModule = true;
exports.d = function (f) { return f(); };
```

Apparently, tsc knows by itself if you are using types or implementation

We do not need to tell typescript to be smart
It is smart by default!

Optimizing bundle size

## Using `const enums`



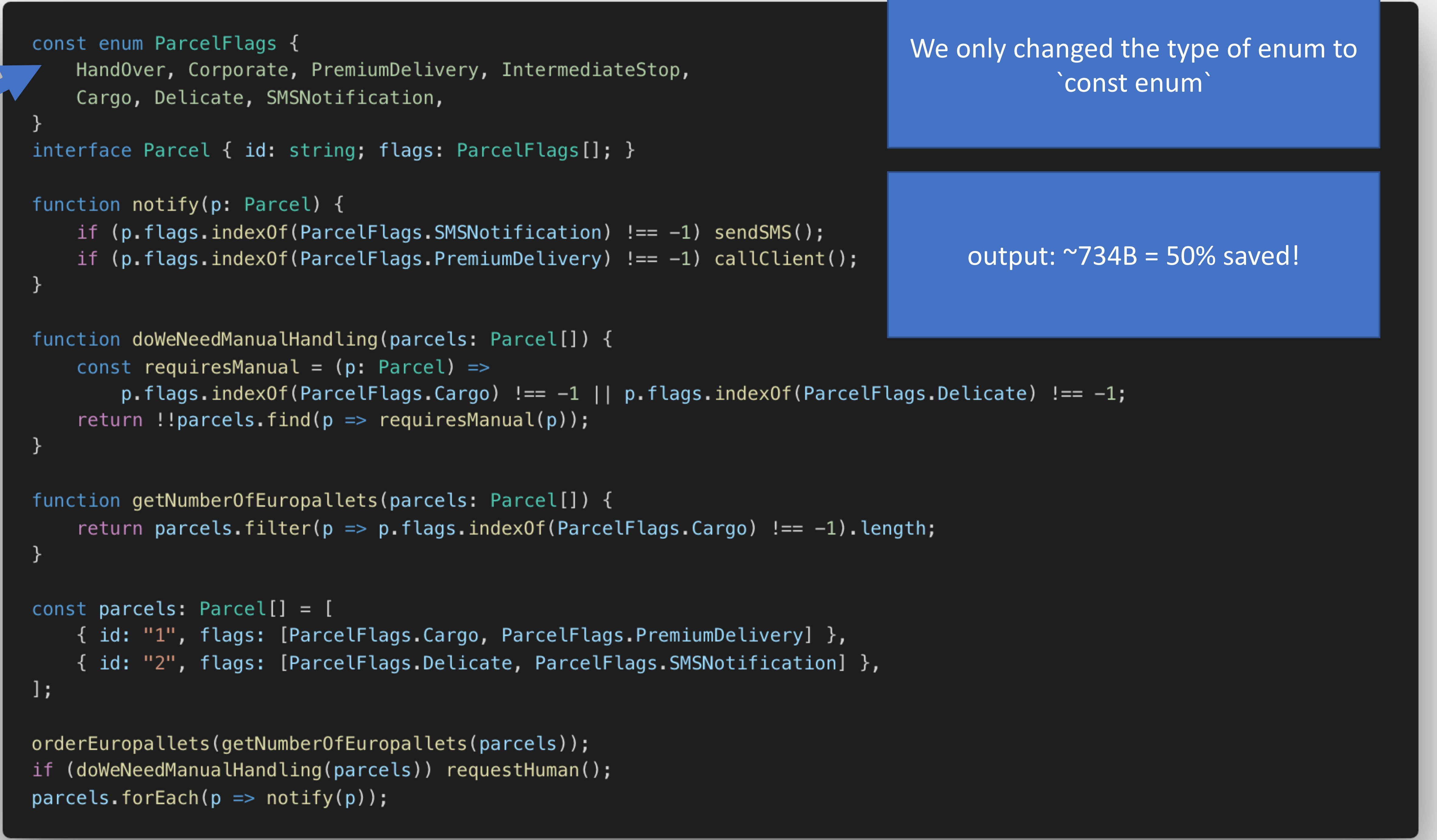
```
enum ParcelFlags {
    HandOver, Corporate, PremiumDelivery, IntermediateStop,
    Cargo, Delicate, SMSNotification,
}
interface Parcel { id: string; flags: ParcelFlags[]; }

function notify(p: Parcel) {
    if (p.flags.indexOf(ParcelFlags.SMSNotification) !== -1) sendSMS();
    if (p.flags.indexOf(ParcelFlags.PremiumDelivery) !== -1) callClient();
}

function doWeNeedManualHandling(parcels: Parcel[]) {
    const requiresManual = (p: Parcel) =>
        p.flags.indexOf(ParcelFlags.Cargo) !== -1 || p.flags.indexOf(ParcelFlags.Delicate) !== -1;
    return !!parcels.find(p => requiresManual(p));
}

function getNumberOfEuropallets(parcels: Parcel[]) {
    return parcels.filter(p => p.flags.indexOf(ParcelFlags.Cargo) !== -1).length;
}

const parcels: Parcel[] = [
    { id: "1", flags: [ParcelFlags.Cargo, ParcelFlags.PremiumDelivery] },
    { id: "2", flags: [ParcelFlags.Delicate, ParcelFlags.SMSNotification] },
];

orderEuropallets(getNumberOfEuropallets(parcels));
if (doWeNeedManualHandling(parcels)) requestHuman();
parcels.forEach(p => notify(p));
```

Heavy use of enums

~1.4kBcompiled to JS

We only changed the type of enum to `const enum`

output: ~734B = 50% saved!



```
const enum ParcelFlags {
    HandOver, Corporate, PremiumDelivery, IntermediateStop,
    Cargo, Delicate, SMSNotification,
}
interface Parcel { id: string; flags: ParcelFlags[]; }

function notify(p: Parcel) {
    if (p.flags.indexOf(ParcelFlags.SMSNotification) !== -1) sendSMS();
    if (p.flags.indexOf(ParcelFlags.PremiumDelivery) !== -1) callClient();
}

function doWeNeedManualHandling(parcels: Parcel[]) {
    const requiresManual = (p: Parcel) =>
        p.flags.indexOf(ParcelFlags.Cargo) !== -1 || p.flags.indexOf(ParcelFlags.Delicate) !== -1;
    return !!parcels.find(p => requiresManual(p));
}

function getNumberOfEuropallets(parcels: Parcel[]) {
    return parcels.filter(p => p.flags.indexOf(ParcelFlags.Cargo) !== -1).length;
}

const parcels: Parcel[] = [
    { id: "1", flags: [ParcelFlags.Cargo, ParcelFlags.PremiumDelivery] },
    { id: "2", flags: [ParcelFlags.Delicate, ParcelFlags.SMSNotification] },
];

orderEuropallets(getNumberOfEuropallets(parcels));
if (doWeNeedManualHandling(parcels)) requestHuman();
parcels.forEach(p => notify(p));
```

```
function notify(p) {
    if (p.flags.indexOf(6) !== -1)
        sendSMS();
    if (p.flags.indexOf(2) !== -1)
        callClient();
}
function doWeNeedManualHandling(parcels) {
    var requiresManual = function (p) {
        return p.flags.indexOf(4) !== -1 || p.flags.indexOf(5) !== -1;
    };
    return !!parcels.find(function (p) { return requiresManual(p); });
}
// ... and so on
```

All enums replaced with numbers

Beware! Do not export const enums in public api of a library cicha obecność

wise team

AmeRykA

ASK SOLI DEO

Keynote available at

**//jblew.pl**

JĘDRZEJ LEWANDOWSKI