RAILWAY ORIENTED TYPESCRIPT

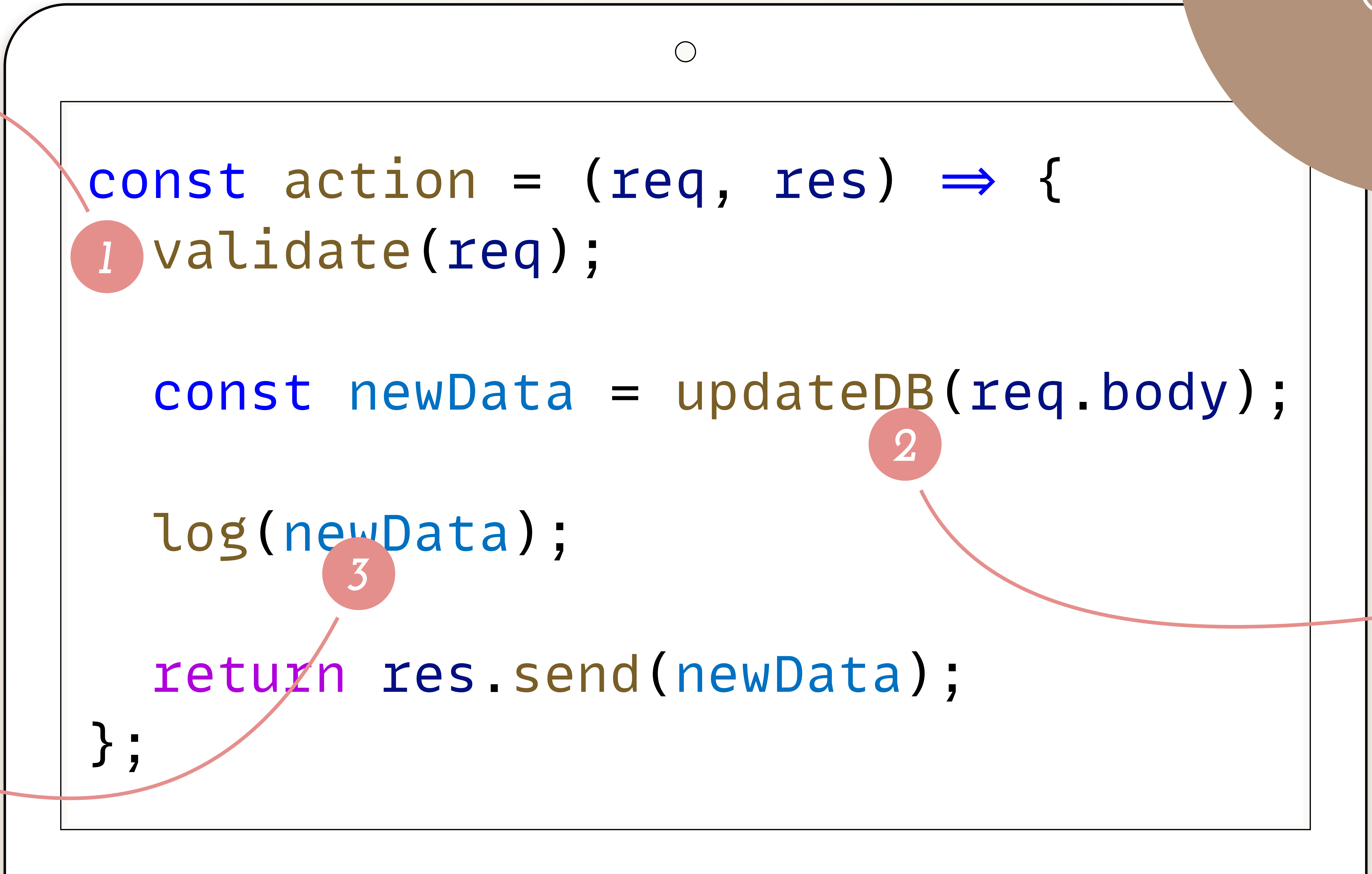
Robin Pokorny

Klarna.

EXAMPLE

A Simple Response To A Request

Happy-Path Code



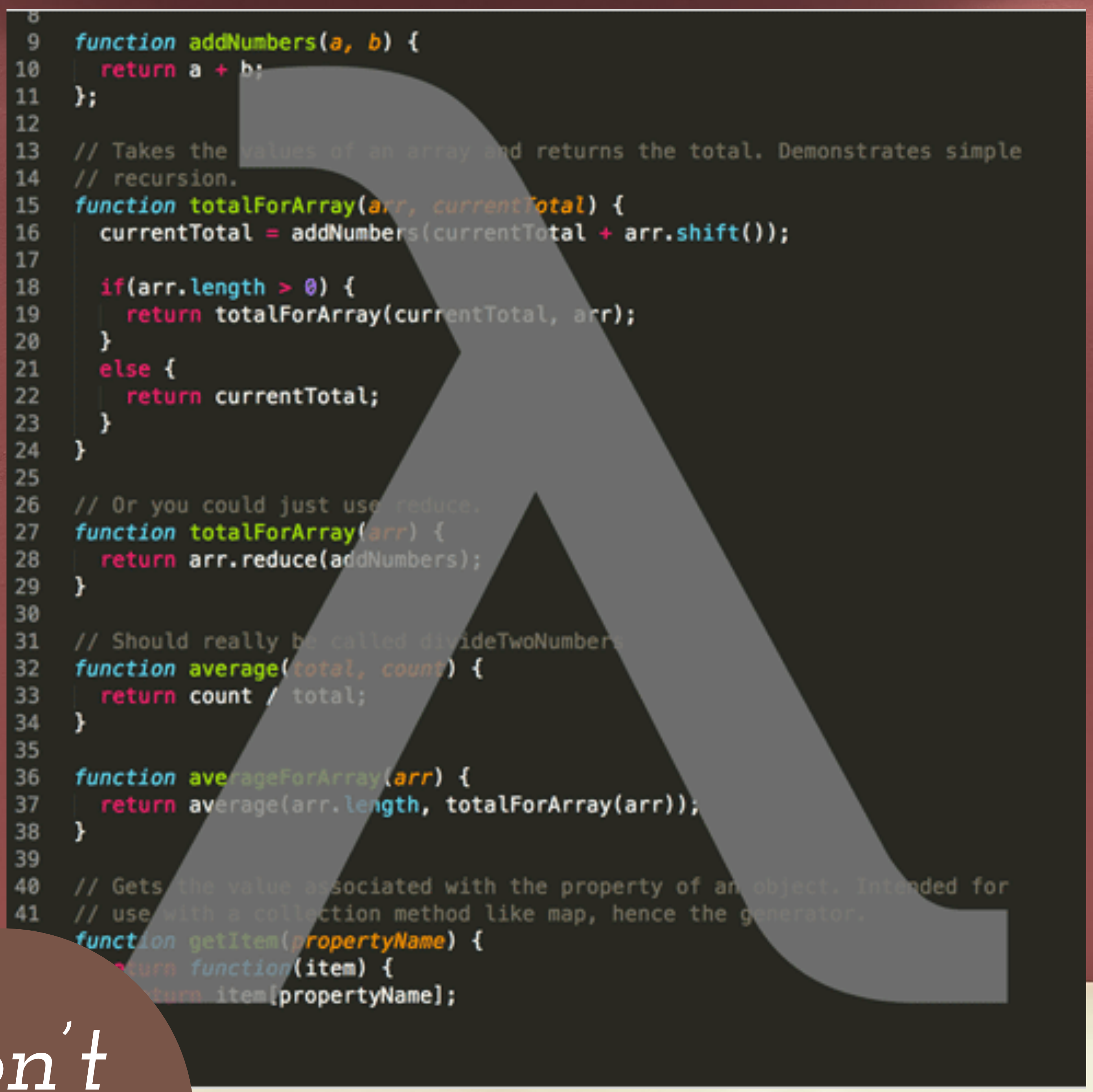
```
const action = (req, res) => {
  validate(req);

  const newData = updateDB(req.body);

  log(newData);

  return res.send(newData);
};
```

What if data is not valid?

What if the DB is unreachable?

What if log fails?

FUNCTIONAL PROGRAMMING



```
function addNumbers(a, b) {
  return a + b;
};

// Takes the values of an array and returns the total. Demonstrates simple
// recursion.
function totalForArray(arr, currentTotal) {
  currentTotal = addNumbers(currentTotal + arr.shift());

  if(arr.length > 0) {
    return totalForArray(currentTotal, arr);
  }
  else {
    return currentTotal;
  }
}

// Or you could just use reduce.
function totalForArray(arr) {
  return arr.reduce(addNumbers);
}

// Should really be called divideTwoNumbers
function average(total, count) {
  return count / total;
}

function averageForArray(arr) {
  return average(arr.length, totalForArray(arr));
}

// Gets the value associated with the property of an object. Intended for
// use with a collection method like map, hence the generator.
function getItem(propertyName) {
  return function(item) {
    return item[propertyName];
```

Don't leave!

MONAD

SEMIGROUP

CATEGORY

SUM TYPE

FUNCTOR

FOLDABLE

ERLANG

HASKELL

CLOSURE

NEXT 40 MINUTES

1

Show

Don't tell

2

TypeScript

But, why?

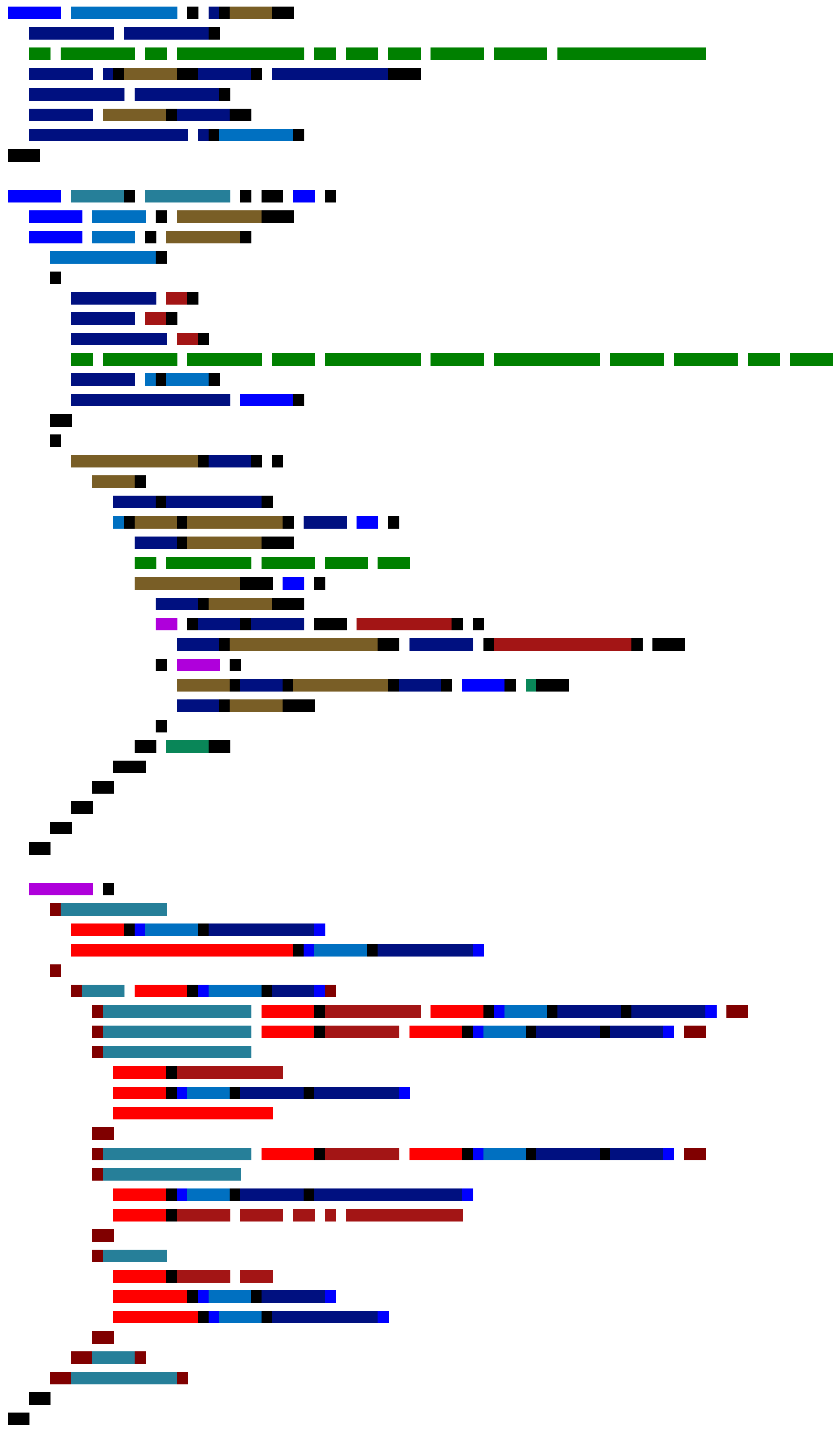
3

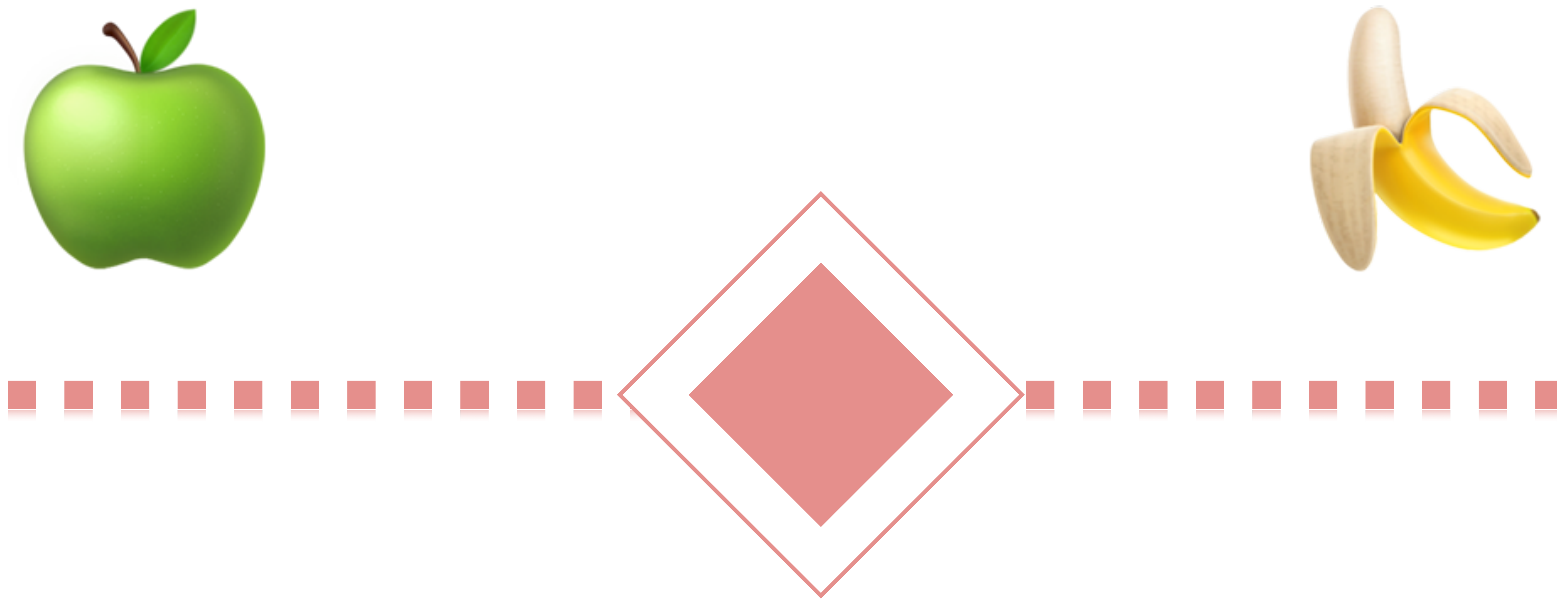
Code

Yes, really

Railway Oriented Programmin
A functional approach to error handling
What do
have to
programm

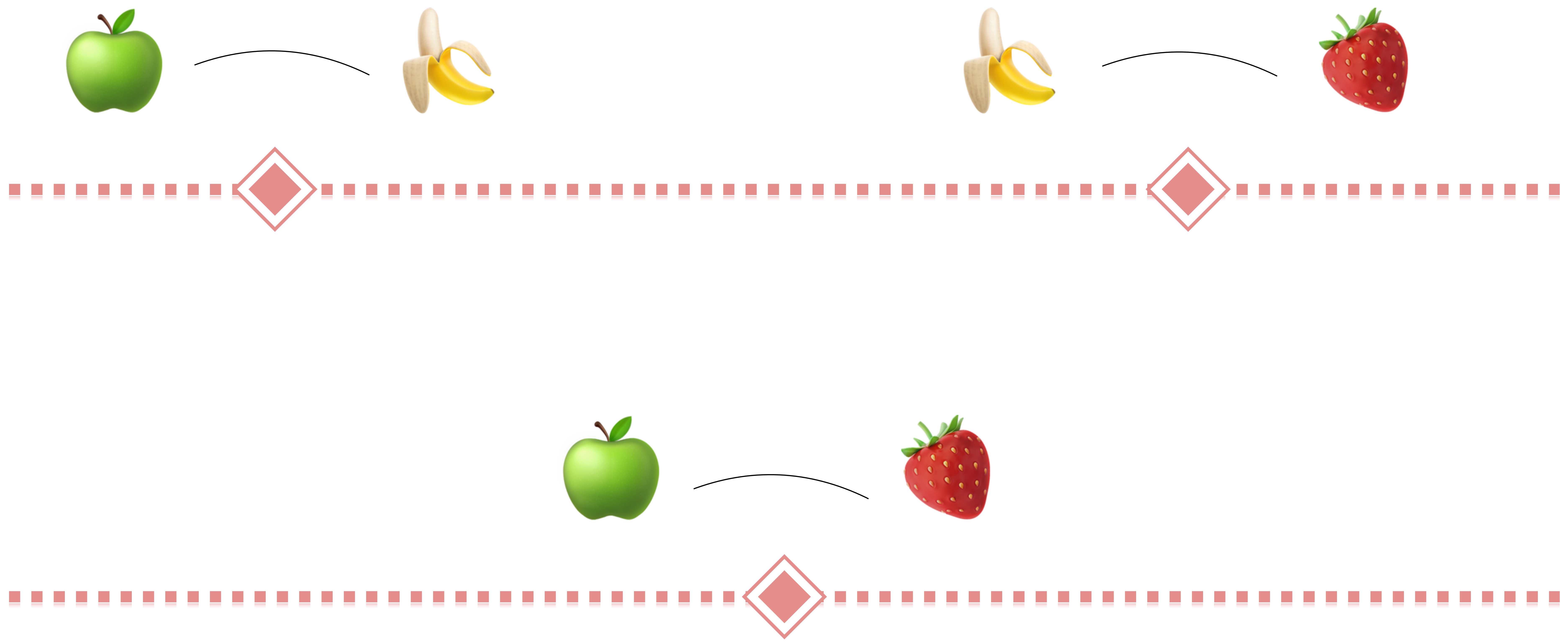
SCOTT
WLASCHIN
RAILWAY ORIENTED
PROGRAMMING

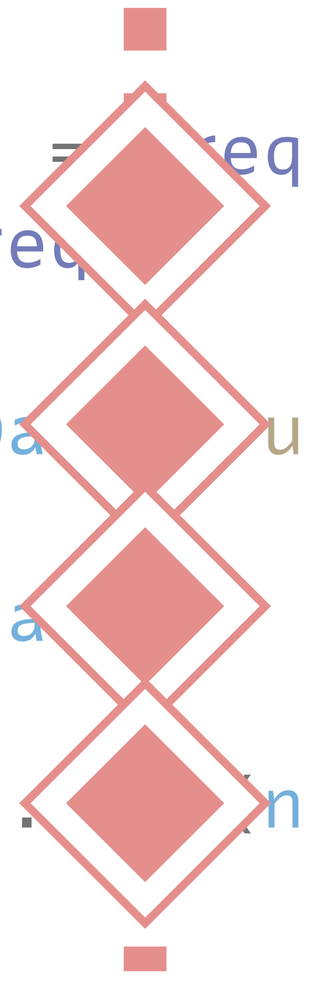




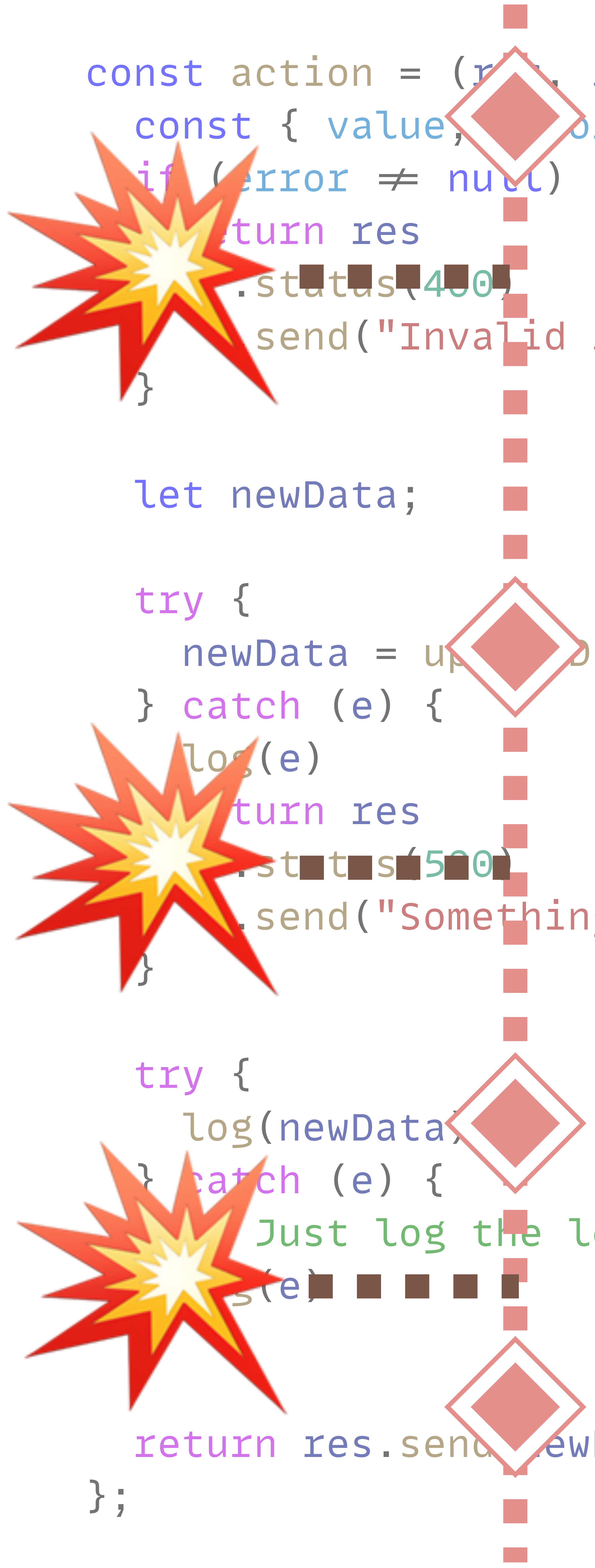
FUNCTION

Apple -> Banana

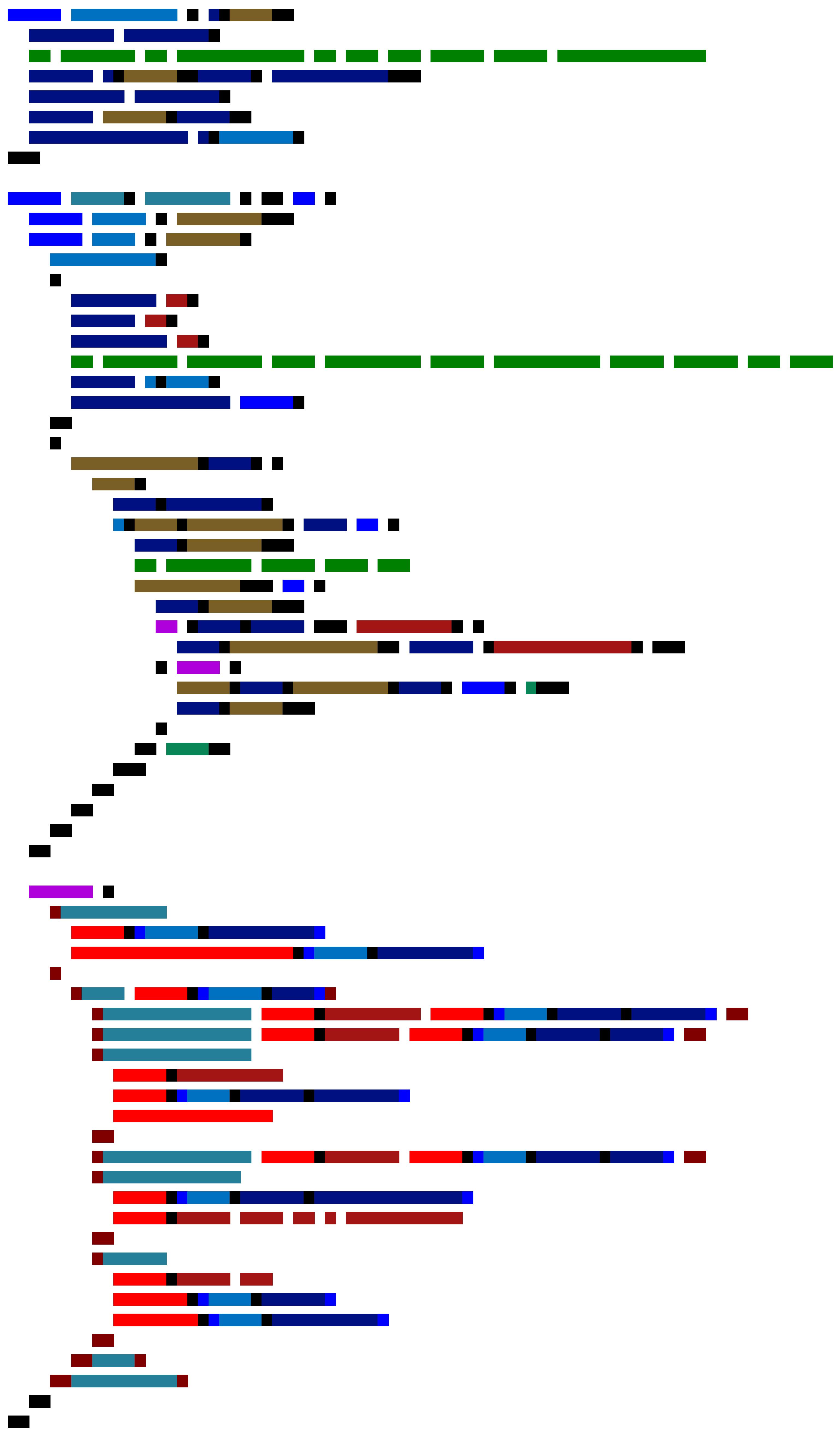


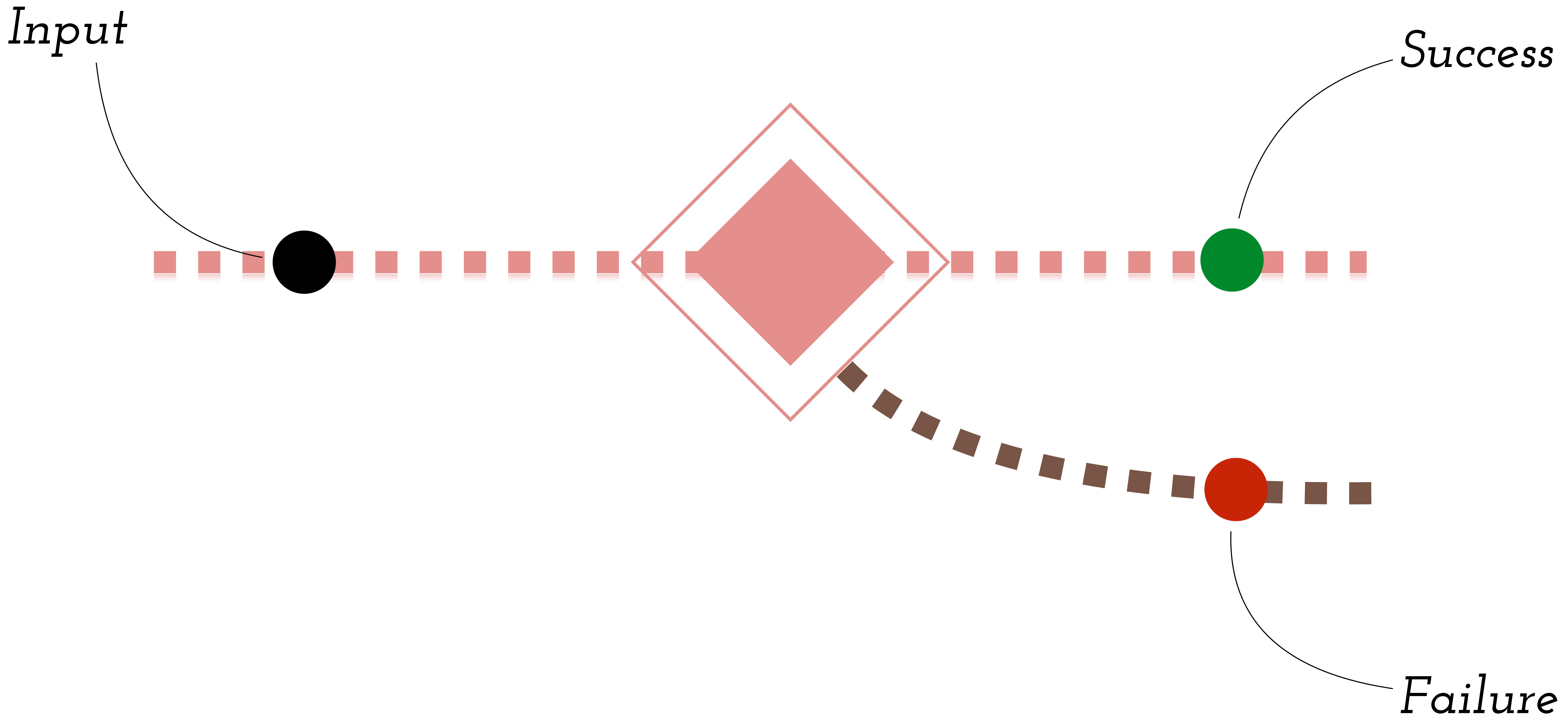


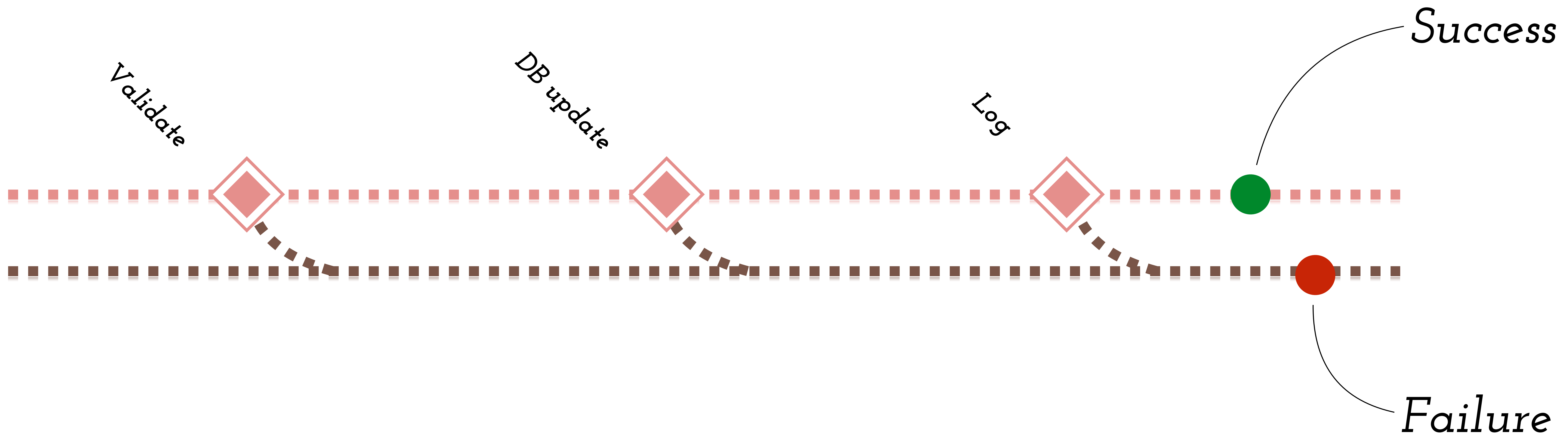
```
const action = (req, res) => {
  validate(req);

  const newData = updateDB(req.body);

  log(newData);

  return res.send(newData);
};
```



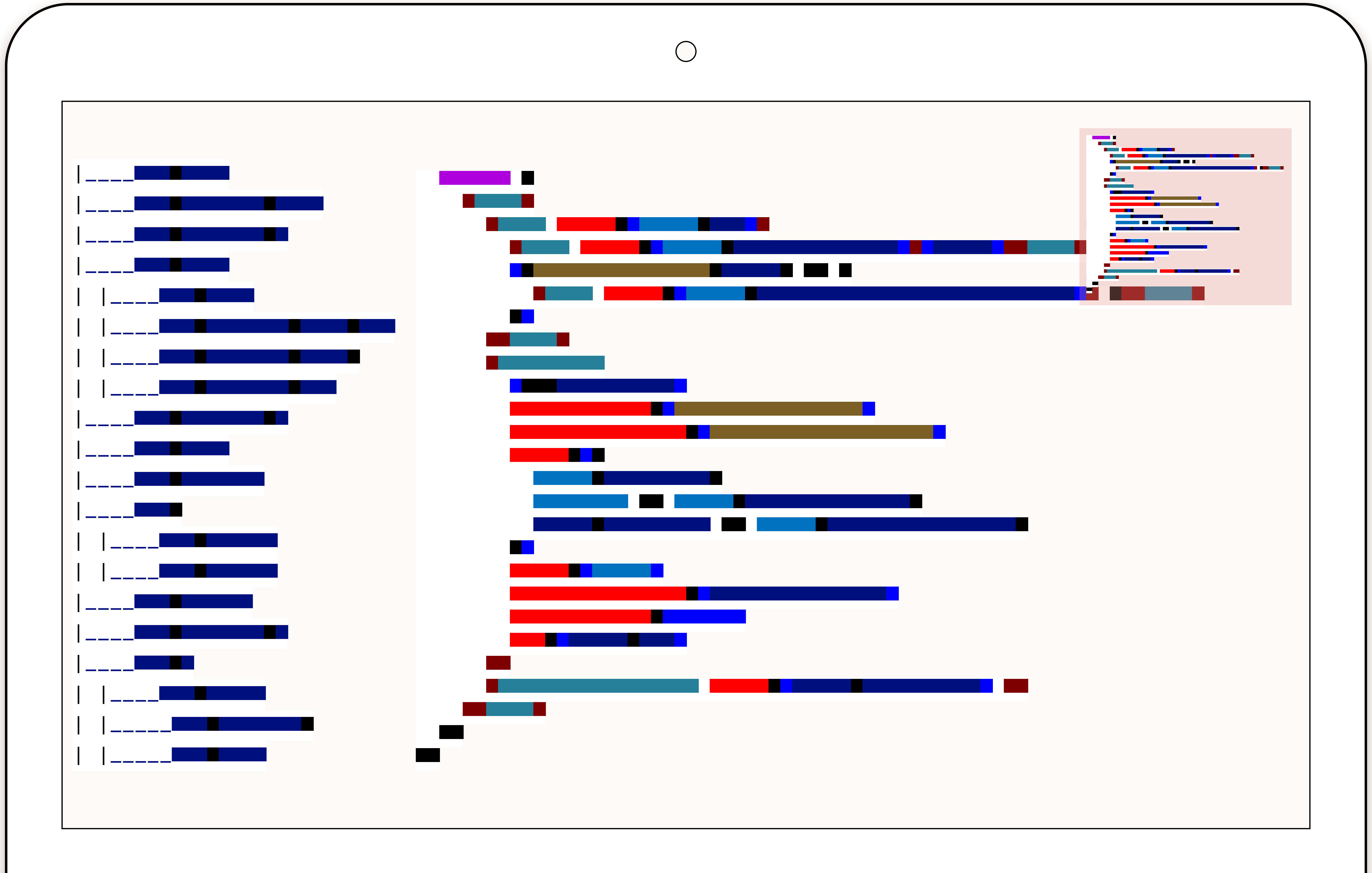
```
const action = (req, res) => {
  const { value, error } = validate(req);
  if (error != null) {
    return res
      .status(400)
      .send("Invalid request");
  }

  let newData;

  try {
    newData = updateDB(req.body);
  } catch (e) {
    log(e)
    return res
      .status(500)
      .send("Something went wrong 🤷‍♂️");
  }

  try {
    log(newData);
  } catch (e) {
    // Just log the logging error
    log(e)
  }

  return res.send(newData);
};
```




Input
Success
Failure


Validate
DB update
Log
Success
Failure

DEMO



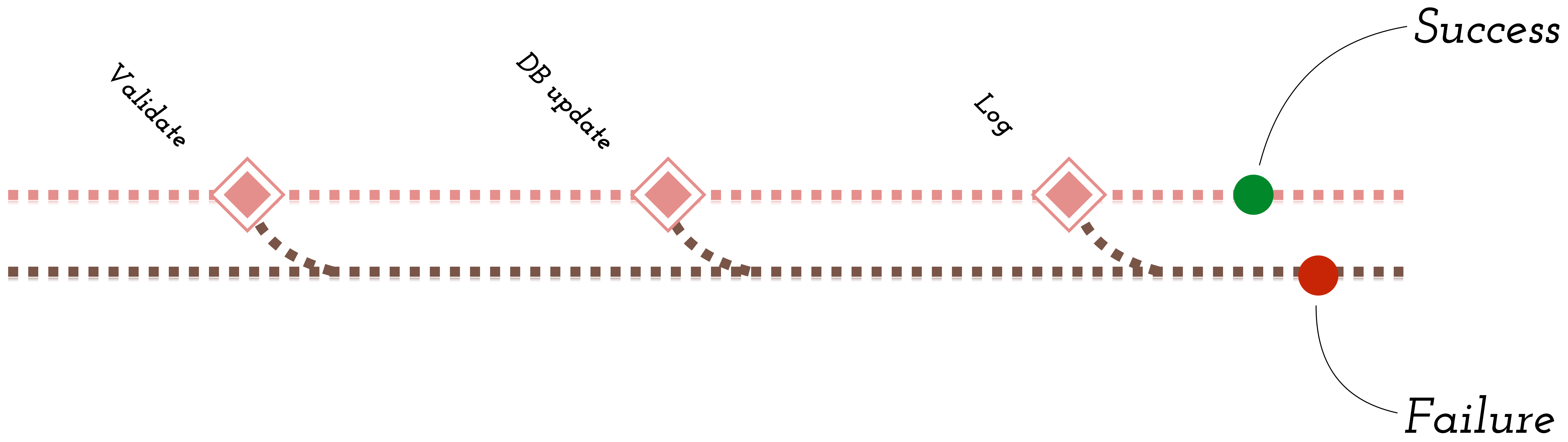
TS

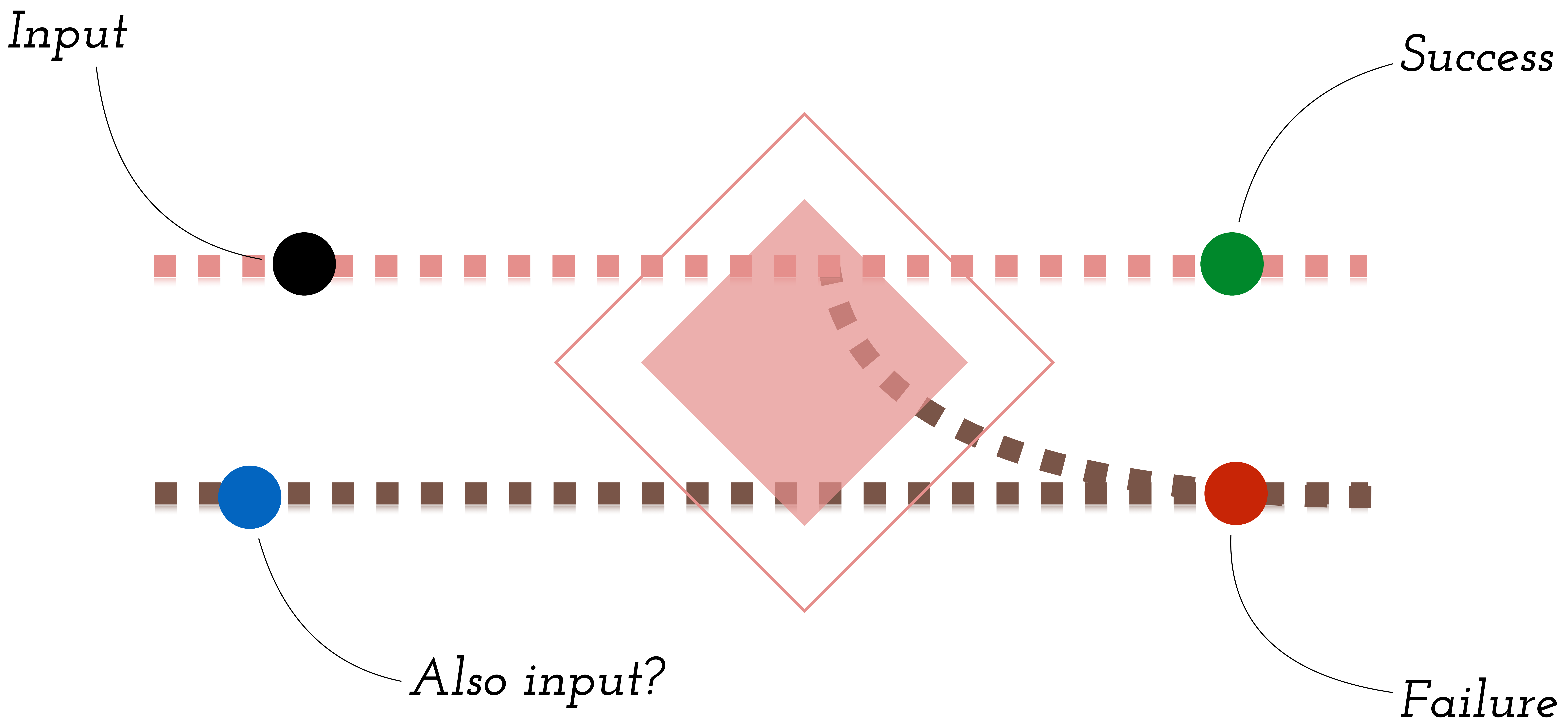
YAGNI…?

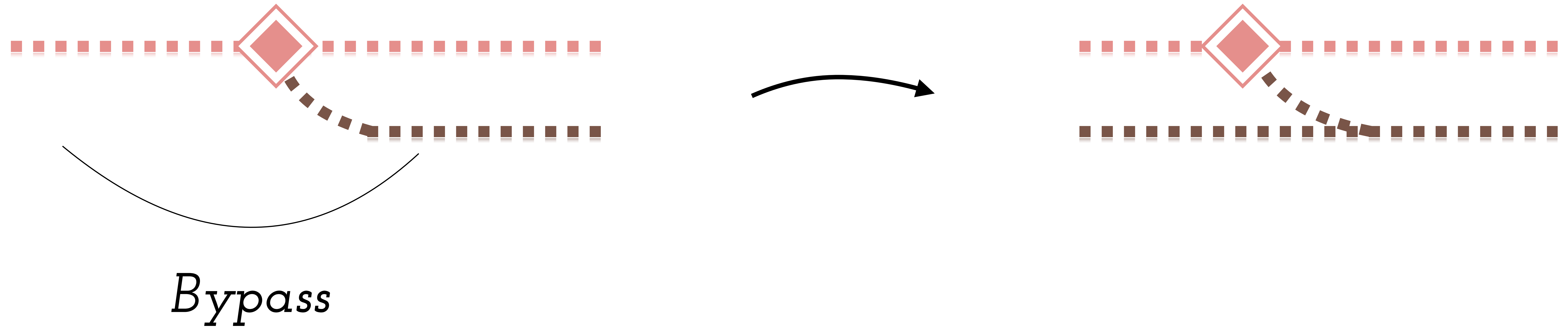
You Ain't Gonna Need It… ?

TYPESCRIPT

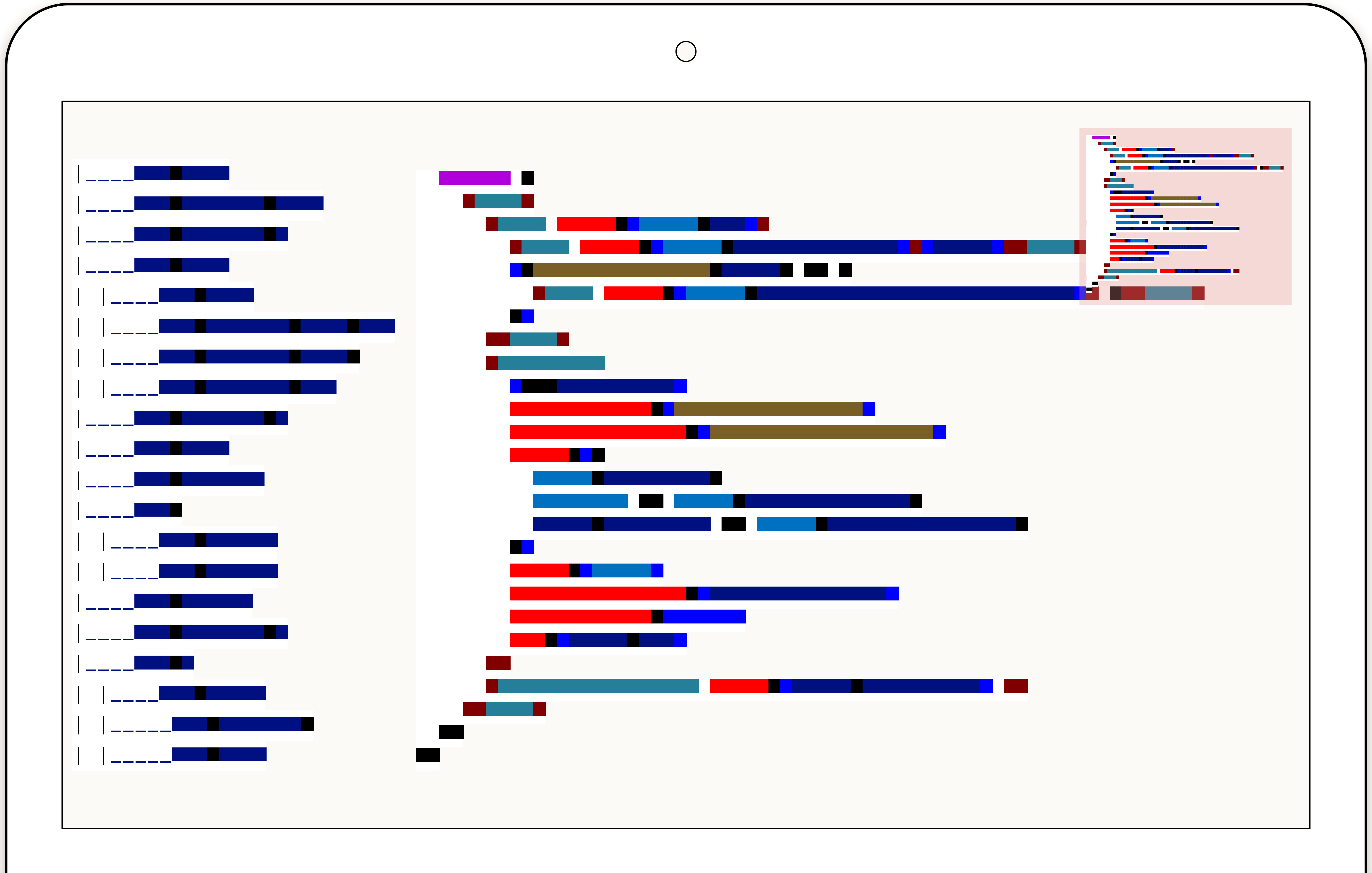
- *Self-Documenting Code*
- *Exhaustive Failure Checking*
- *Composing Can Be Tricky*


Validate
DB update
Log
Success
Failure


Input
Success
Also input?
Failure


Bypass

DEMO



```
Promise.resolve()
```

YAGNI...?

You Ain't Gonna Need It... ?

PROMISES

◆ *Limited API*

PROMISES API

Methods

- Promise#then
- Promise#catch
- Promise#finally

Combinators

- Promise.race
- Promise.allSettled
- Promise.all
- Promise.any

PROMISES

- *Limited API*
- *Domain* ✖ *Panic*

ERRORS

DOMAIN

Expected

INVALID INPUT DATA

EXTERNAL SYSTEM UNREACHABLE

PANIC

Unh

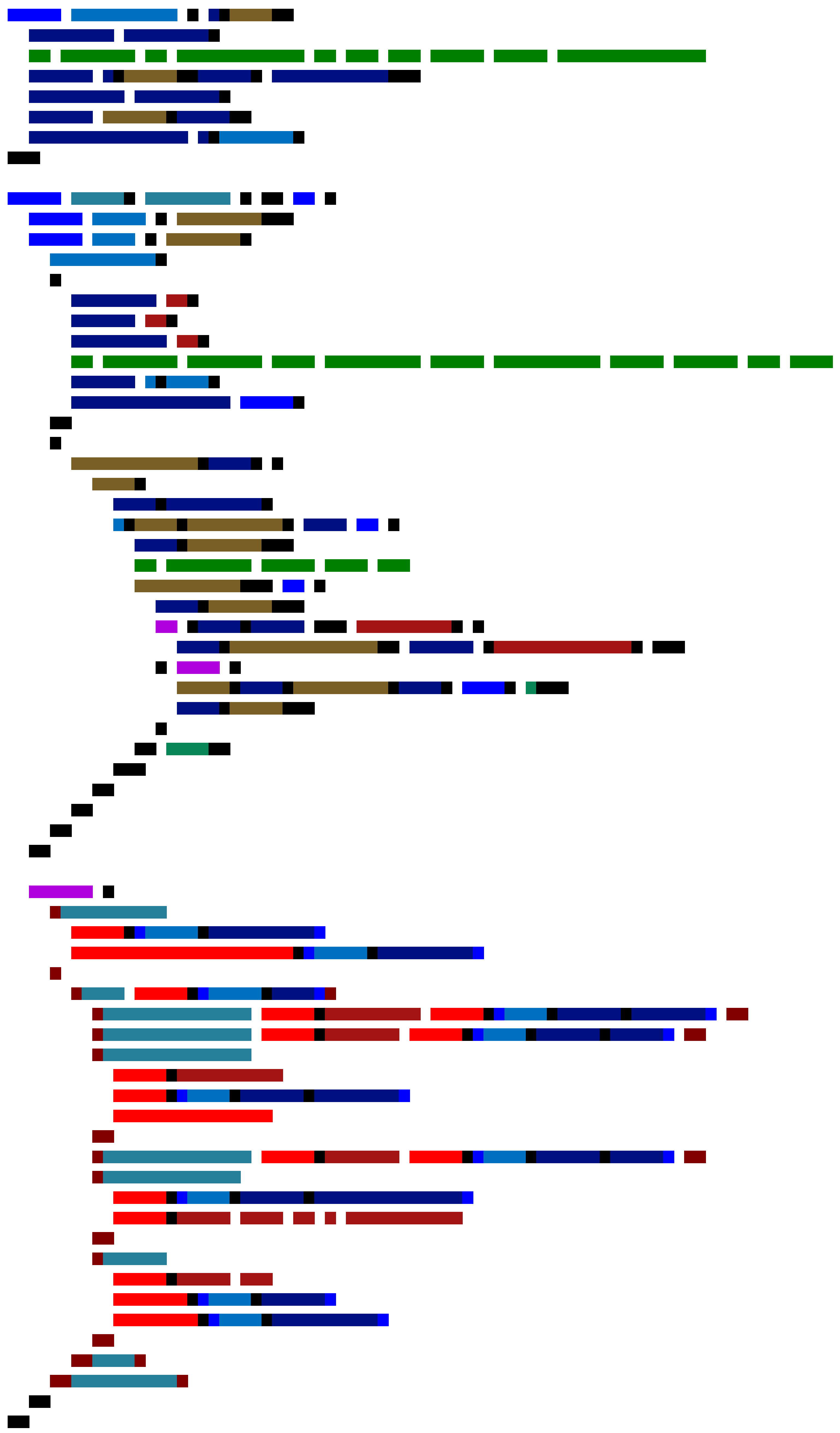
OUT OF MEMORY

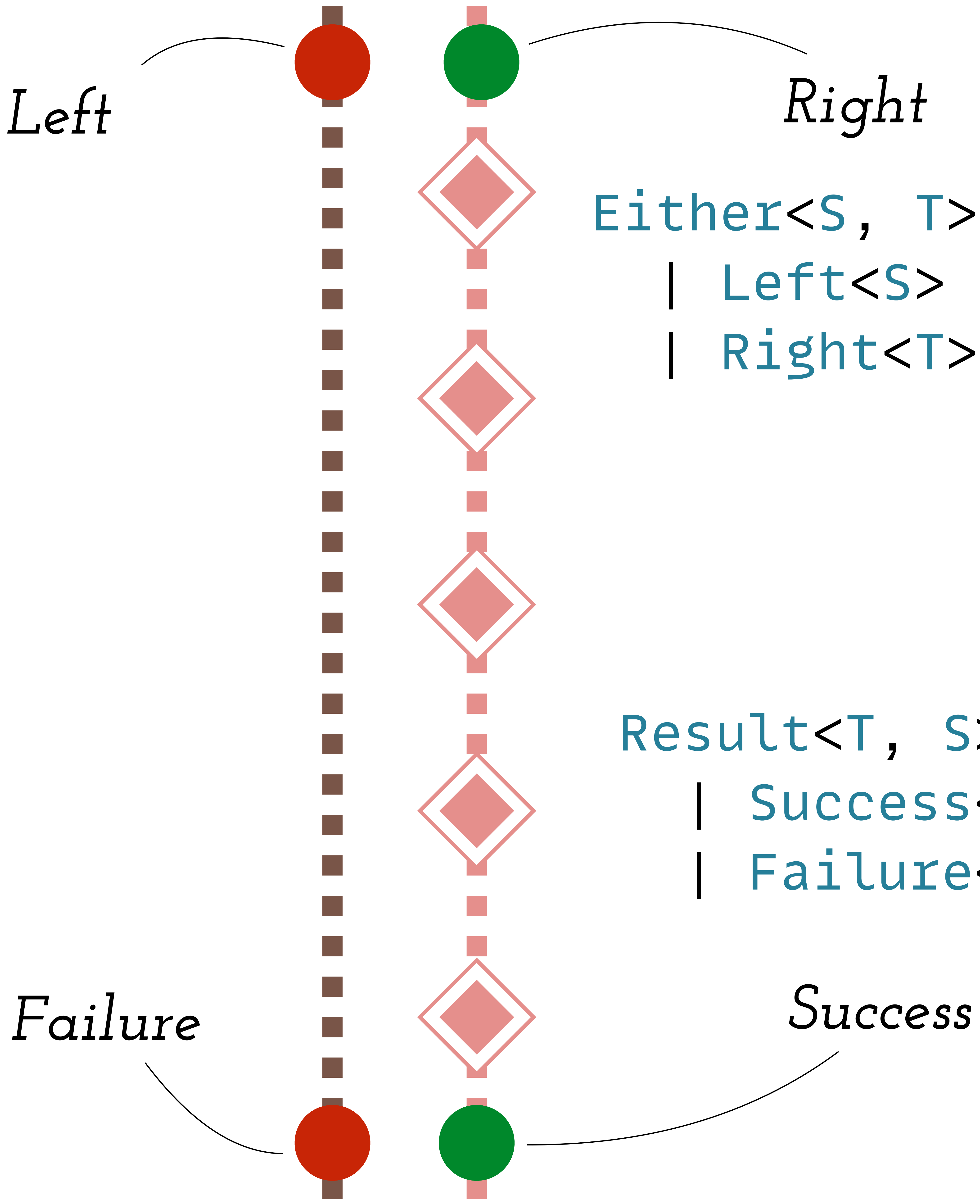
DIVIDE BY ZERO

NO EXHAUSTIVE CHECKS

FP-TS

https://github.com/gcanti/fp-ts

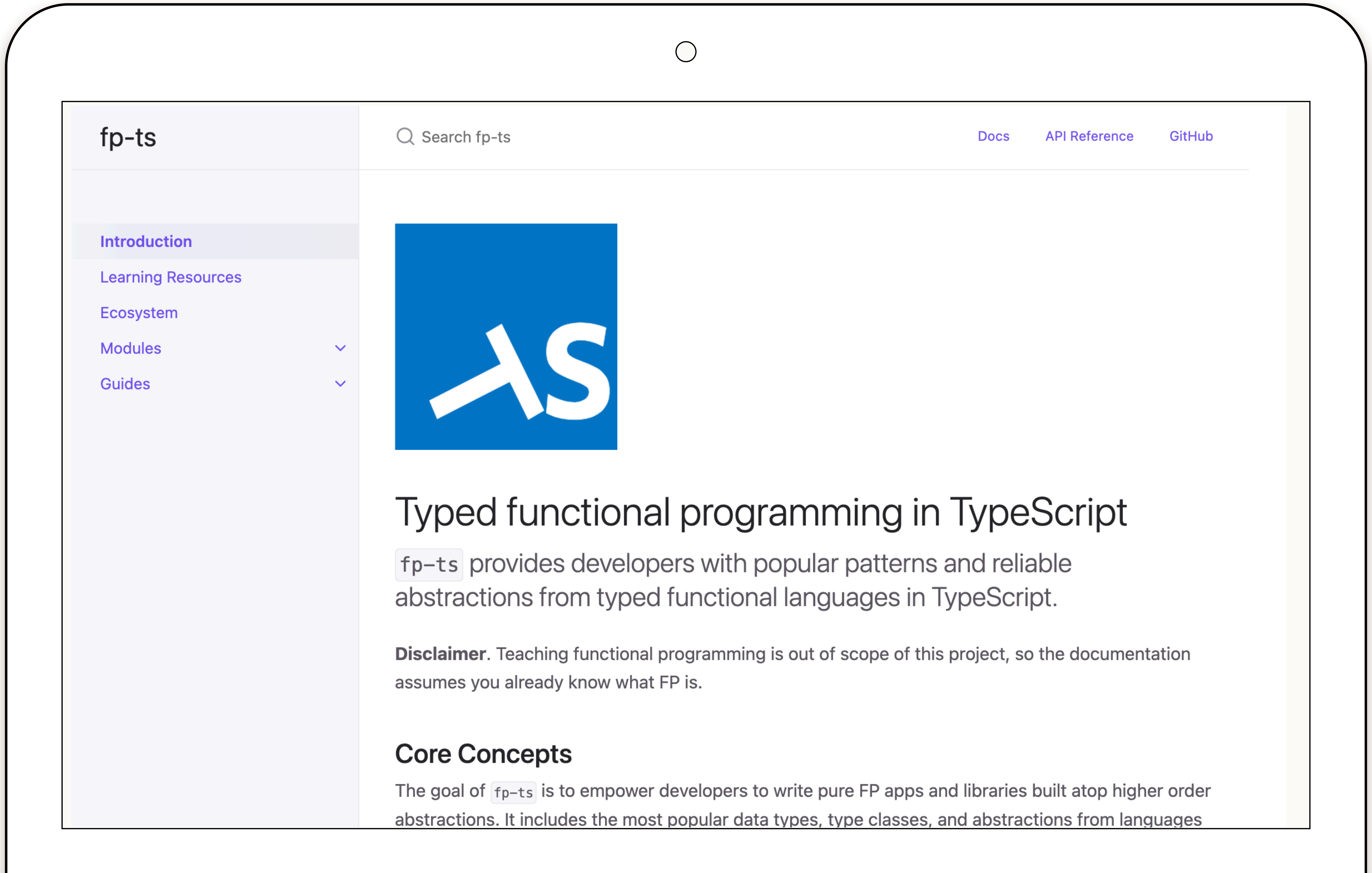




```
Either<S, T> =
    | Left<S>
    | Right<T>
```

```
Result<T, S> =
    | Success<T>
    | Failure<S>
```

DEMO



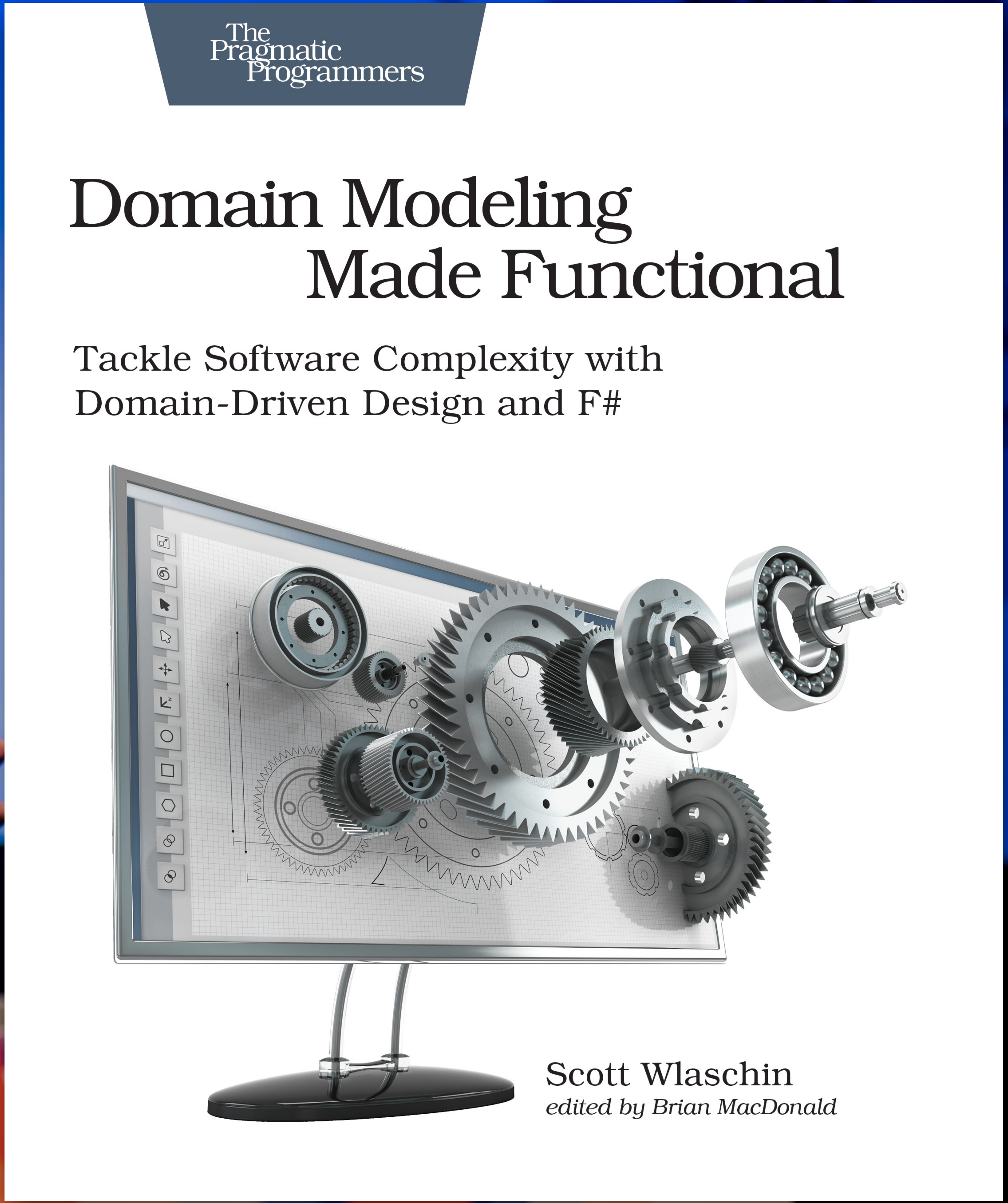
Typed functional programming in TypeScript

`fp-ts` provides developers with popular patterns and reliable abstractions from typed functional languages in TypeScript.

Disclaimer. Teaching functional programming is out of scope of this project, so the documentation assumes you already know what FP is.

Core Concepts

The goal of `fp-ts` is to empower developers to write pure FP apps and libraries built atop higher order abstractions. It includes the most popular data types, type classes, and abstractions from languages



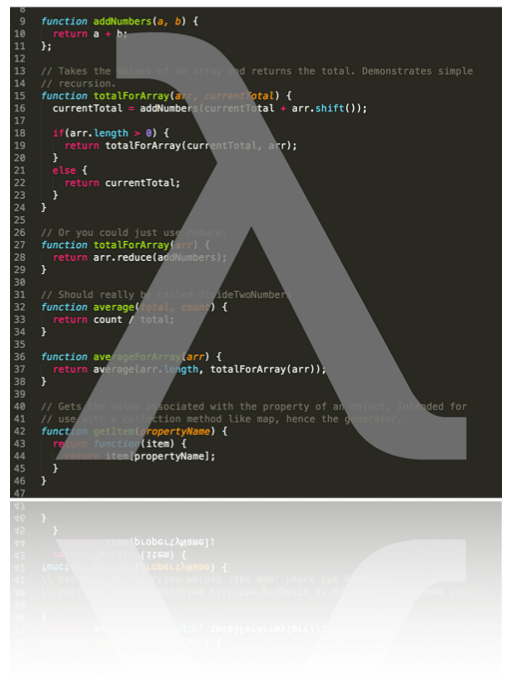
The Pragmatic Programmers

Domain Modeling Made Functional

Tackle Software Complexity with Domain-Driven Design and F#

Scott Wlaschin

edited by Brian MacDonald



```
function addNumbers(a, b) {
  return a + b;
};

// Takes the values of an array and returns the total. Demonstrates simple
// recursion.
function totalForArray(arr, currentTotal) {
  currentTotal = addNumbers(currentTotal + arr.shift());

  if(arr.length > 0) {
    return totalForArray(currentTotal, arr);
  }
  else {
    return currentTotal;
  }
}

// Or you could just use reduce.
function totalForArray(arr) {
  return arr.reduce(addNumbers);
}

// Should really be called divideTwoNumbers
function average(total, count) {
  return count / total;
}

function averageForArray(arr) {
  return average(arr.length, totalForArray(arr));
}

// Gets the value associated with the property of an object. Intended for
// use with a collection method like map, hence the generator.
function getItem(propertyName) {
  return function(item) {
    return item[propertyName];
  }
}
```

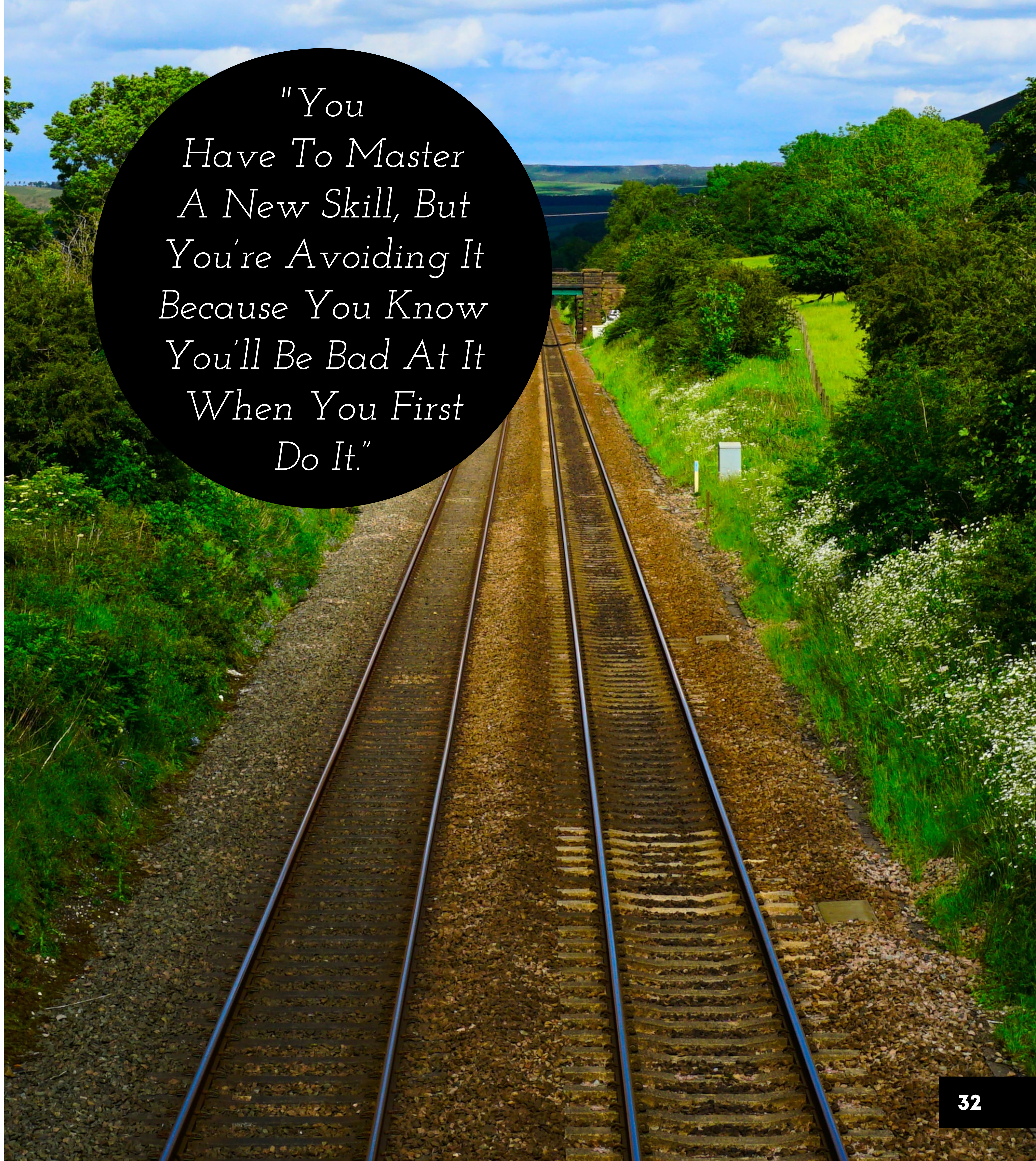
+ TS

INNOVATION

IS

COMBINATION

— Greg Satell



"You Have To Master A New Skill, But You're Avoiding It Because You Know You'll Be Bad At It When You First Do It."

RAILWAY ORIENTED TYPESCRIPT

@robinpokorny

Klarna.