

RAILWAY ORIENTED TYPESCRIPT

Robin Pokorny

Klarna.

EXAMPLE

A Simple Response To A Request

*Happy-Path
Code*

*What if data
is not valid?*

```
const action = (req, res) => {  
  1 validate(req);  
  
  const newData = updateDB(req.body);  
  
  log(newData);  
  2  
  return res.send(newData);  
};  
  3
```

*What if the DB
is unreachable?*

*What if
log fails?*

FUNCTIONAL PROGRAMMING

```
8
9 function addNumbers(a, b) {
10   return a + b;
11 };
12
13 // Takes the values of an array and returns the total. Demonstrates simple
14 // recursion.
15 function totalForArray(arr, currentTotal) {
16   currentTotal = addNumbers(currentTotal + arr.shift());
17
18   if(arr.length > 0) {
19     return totalForArray(currentTotal, arr);
20   }
21   else {
22     return currentTotal;
23   }
24 }
25
26 // Or you could just use reduce.
27 function totalForArray(arr) {
28   return arr.reduce(addNumbers);
29 }
30
31 // Should really be called divideTwoNumbers
32 function average(total, count) {
33   return count / total;
34 }
35
36 function averageForArray(arr) {
37   return average(arr.length, totalForArray(arr));
38 }
39
40 // Gets the value associated with the property of an object. Intended for
41 // use with a collection method like map, hence the generator.
42 function getItem(propertyName) {
43   return function(item) {
44     return item[propertyName];
45   };
46 }
47
```

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?

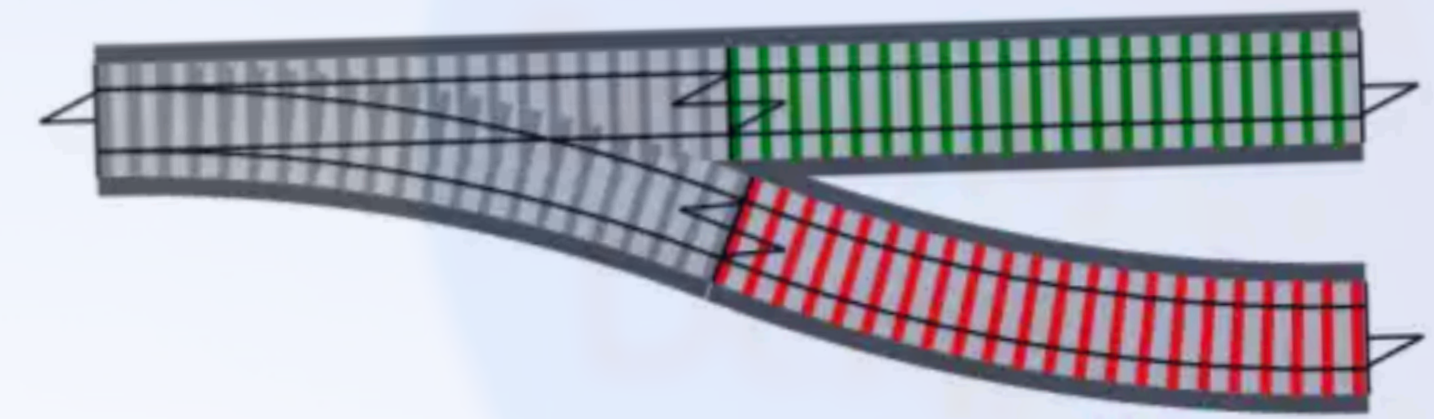


3

Code
Yes, really

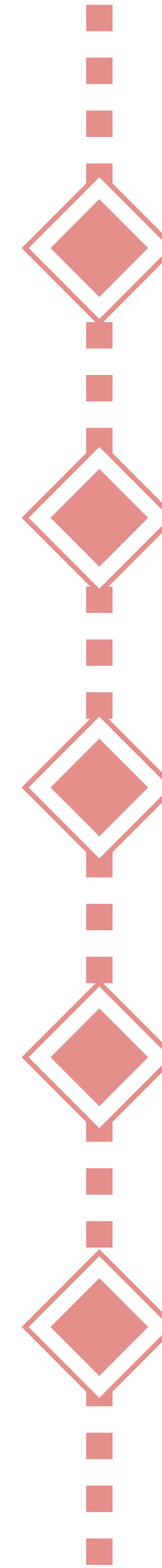
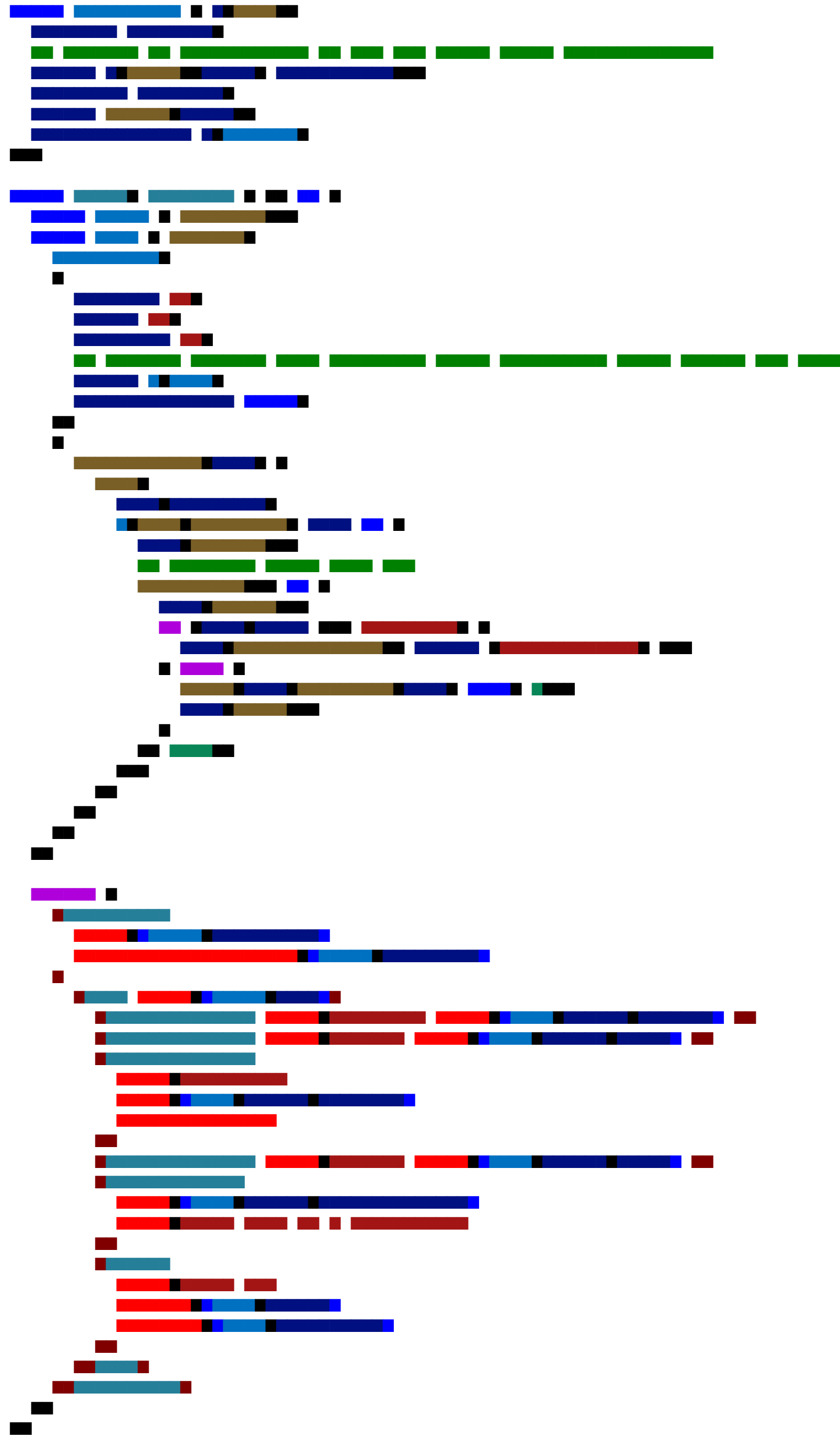
Railway Oriented Programming

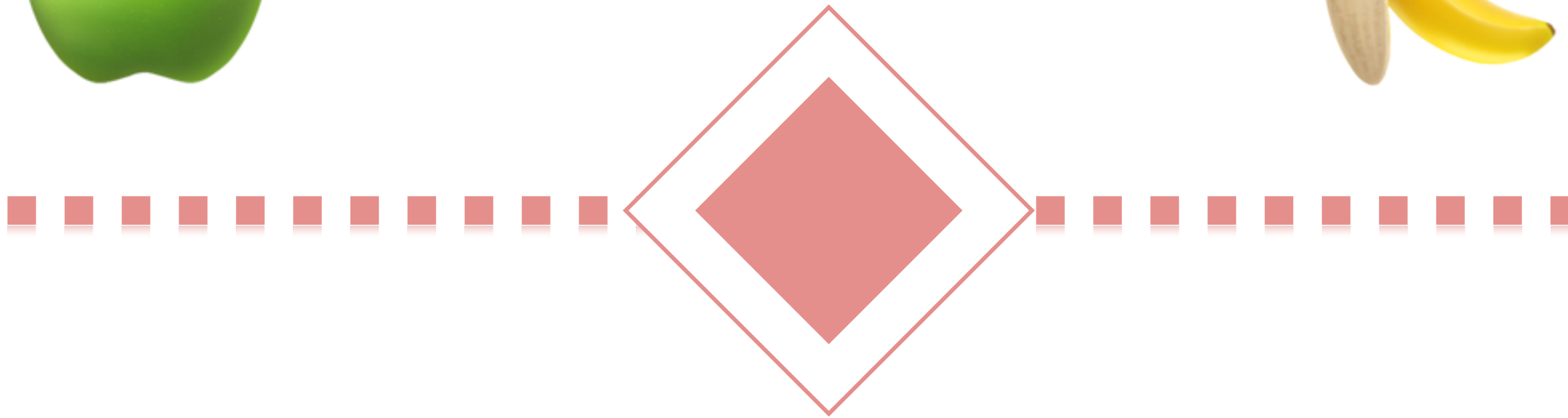
A functional approach to error handling



What do
have to
programm

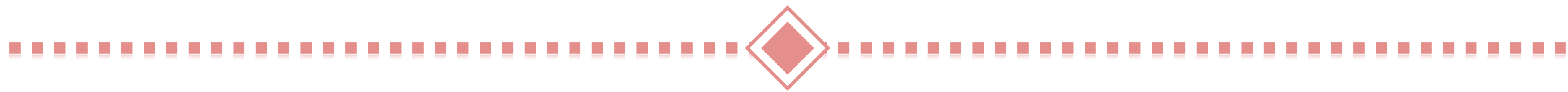
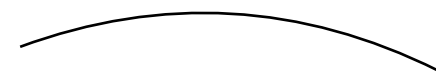
**SCOTT
WLASCHIN
RAILWAY ORIENTED
PROGRAMMING**





FUNCTION

Apple -> Banana



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

  const newData = updateDB(req.body);

  log(newData);

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

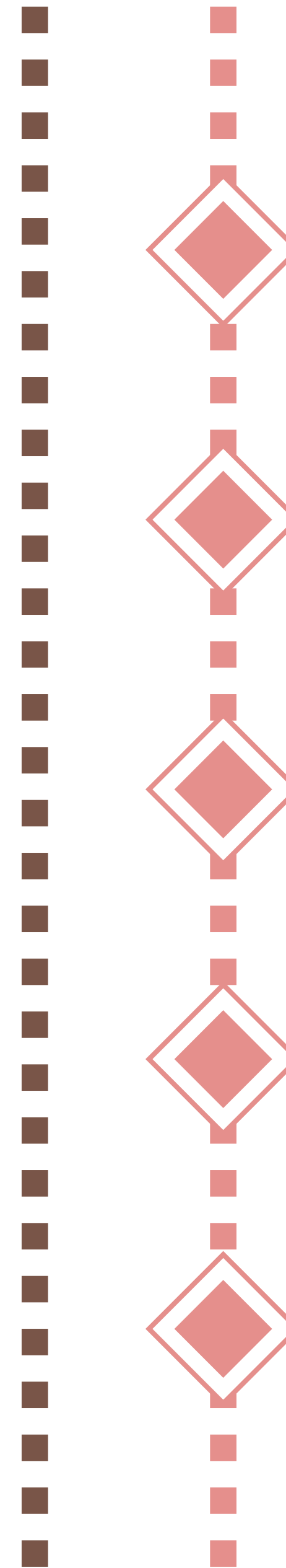
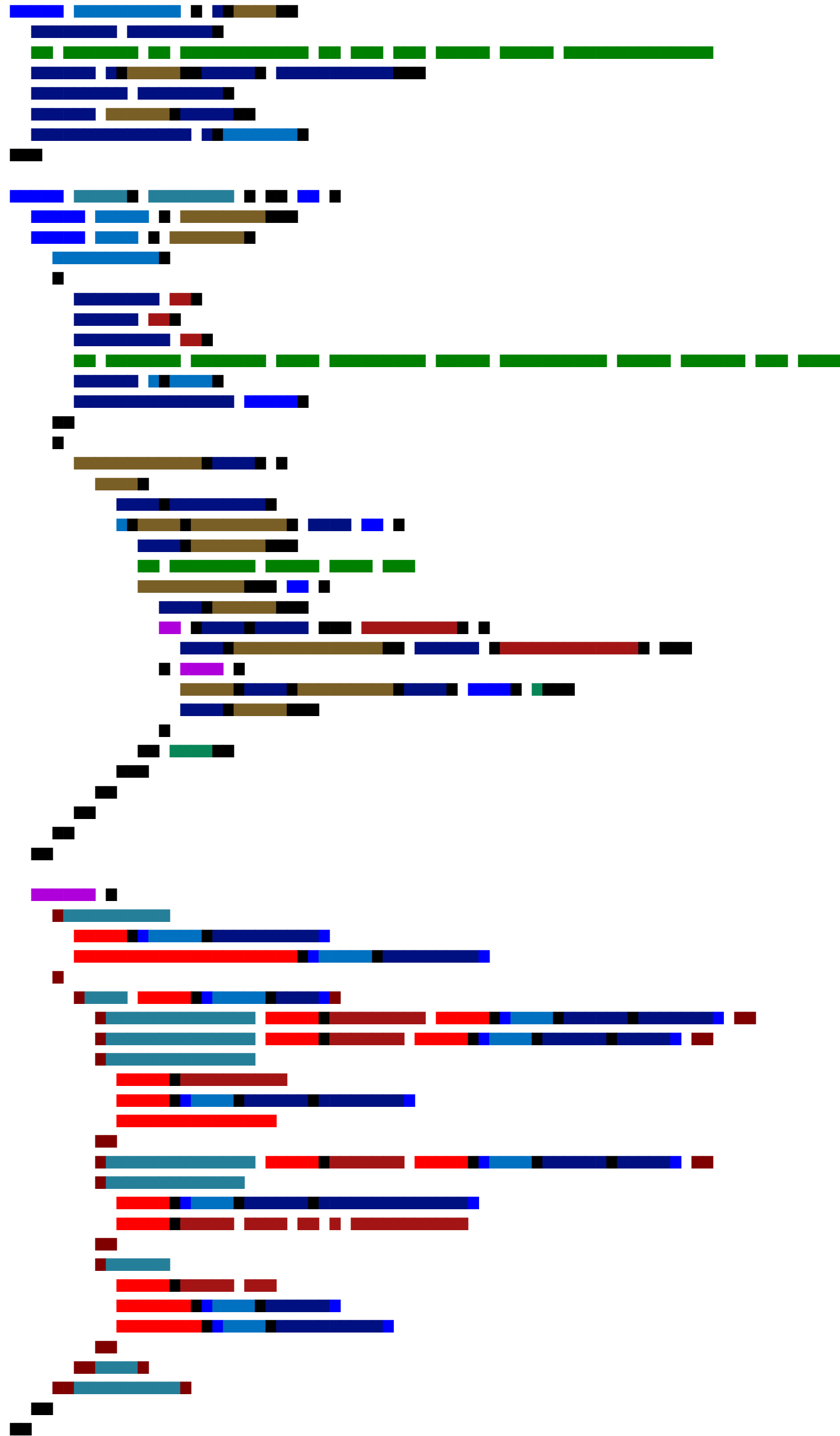
```
const action = (req, res) => {
  const { value, error } = validate(req);
  if (error !== null) {
    return res
      .status(400)
      .send("Invalid request");
  }

  let newData;

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

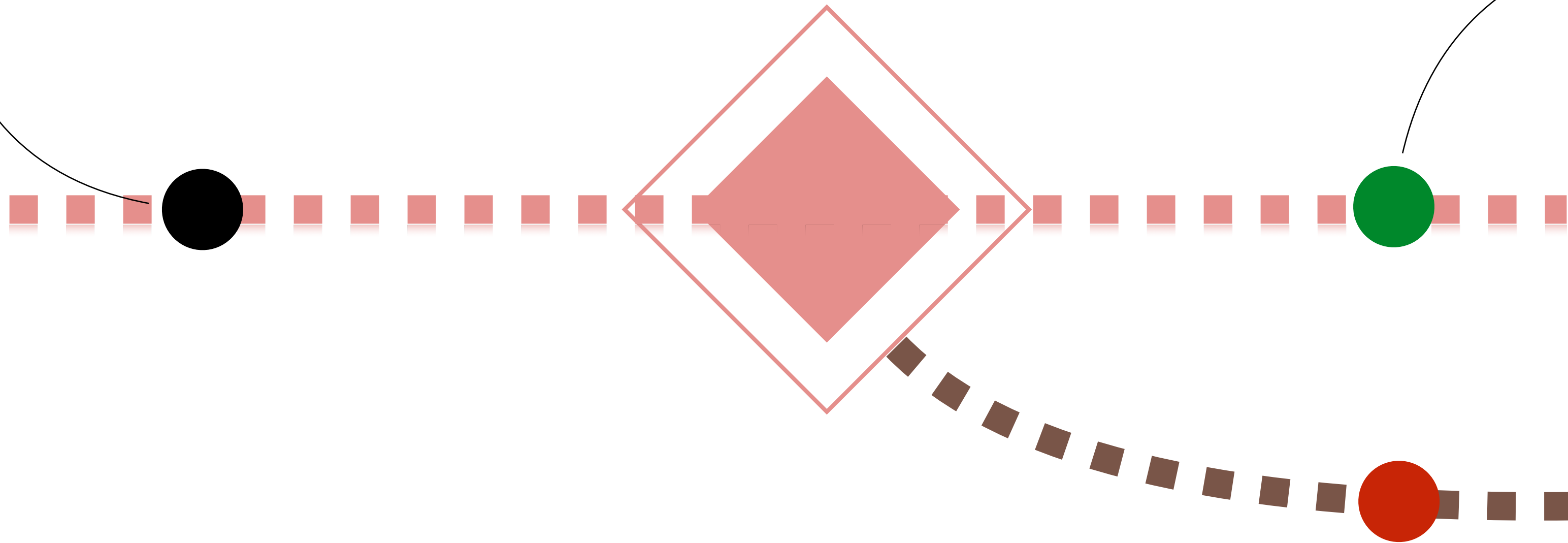
  try {
    log(newData);
  } catch (e) {
    // Just log the logging error
    console.log(e);
  }

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



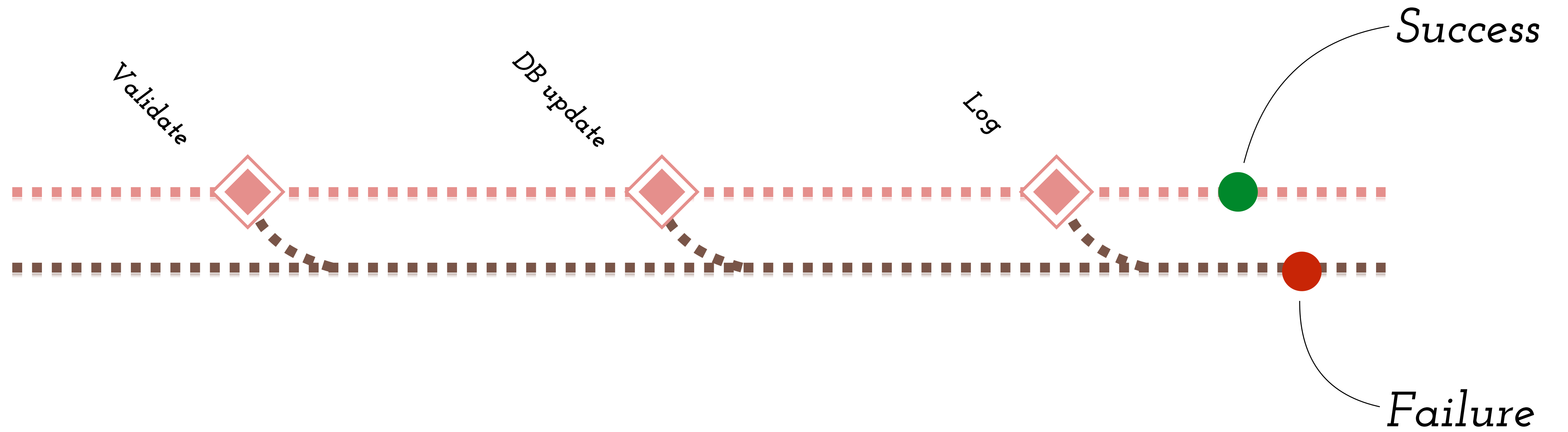


Input

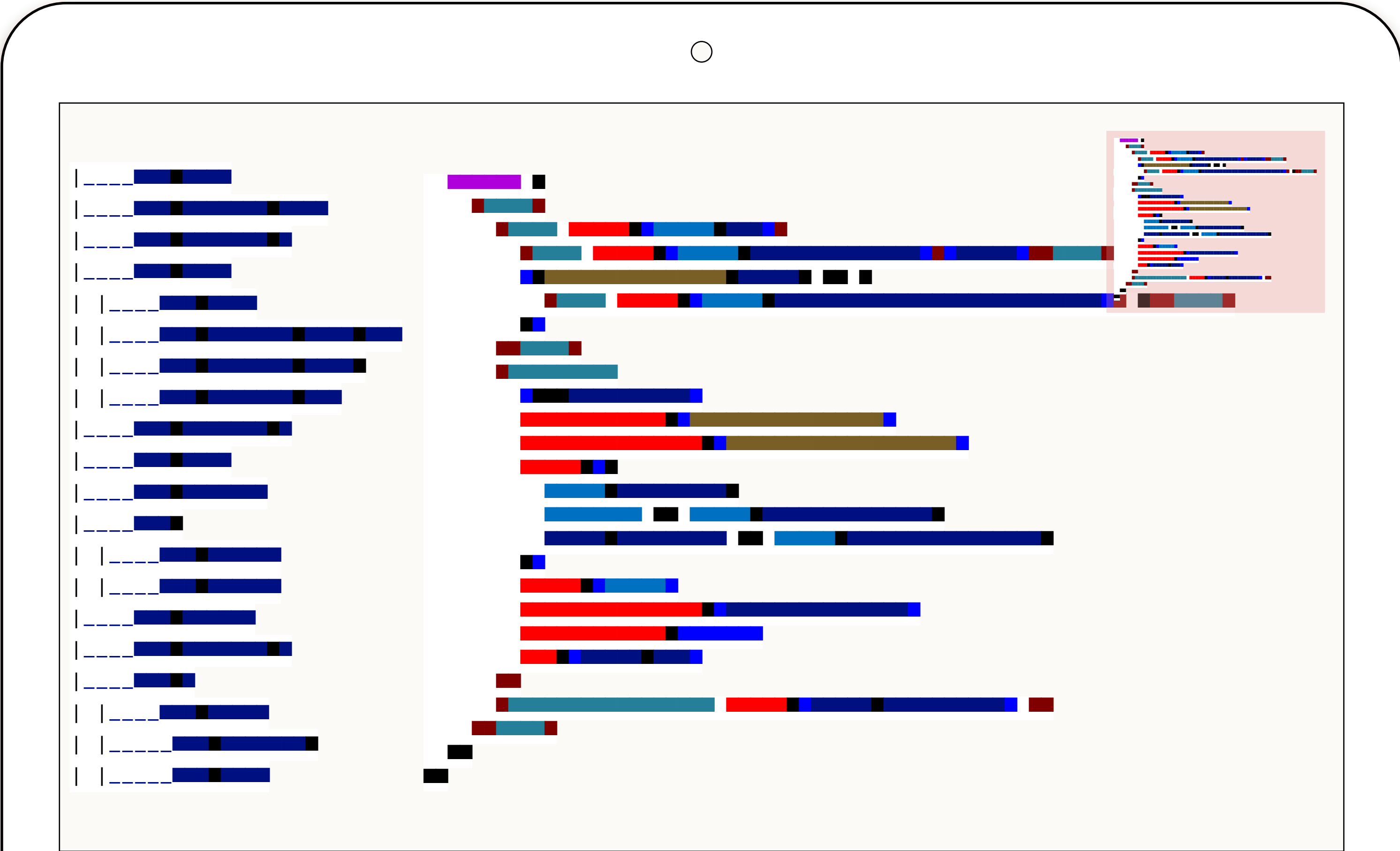


Success

Failure



DEMO

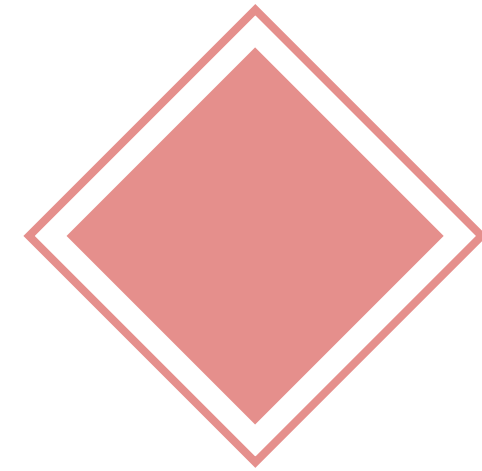




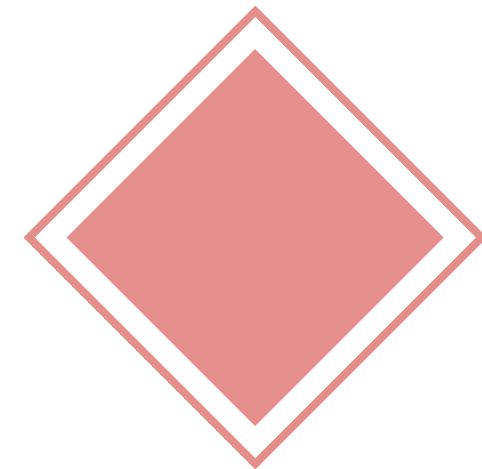
YAGNI...?

You Ain't Gonna Need It... ?

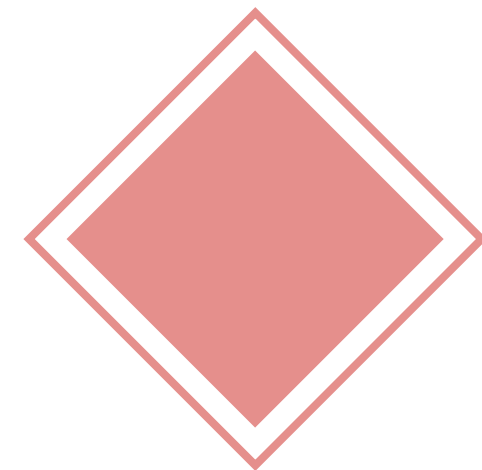
TYPESCRIPT



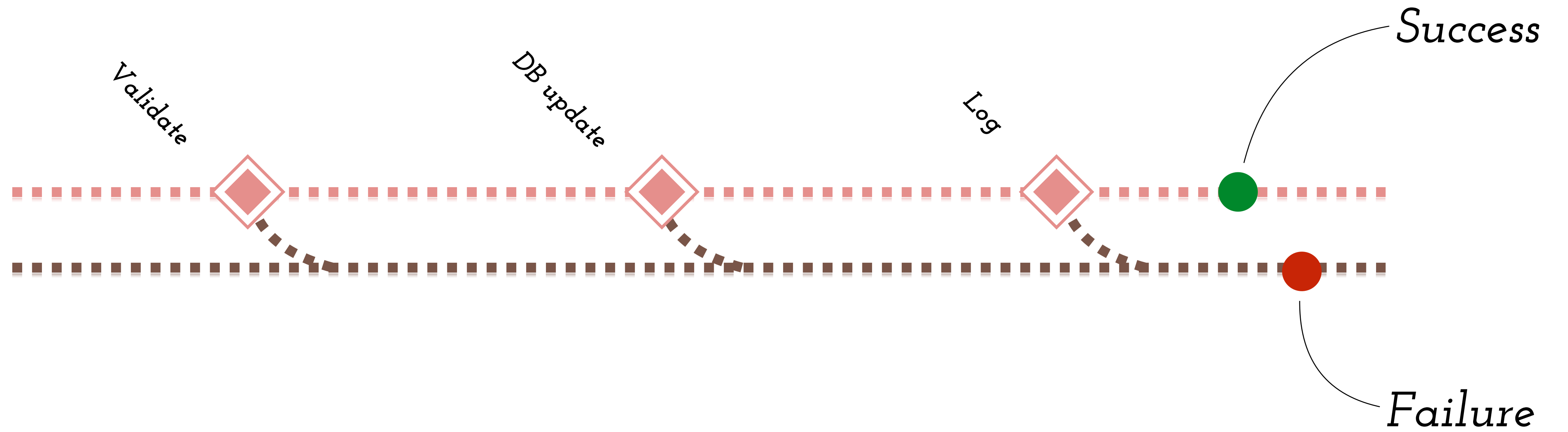
Self-Documenting Code



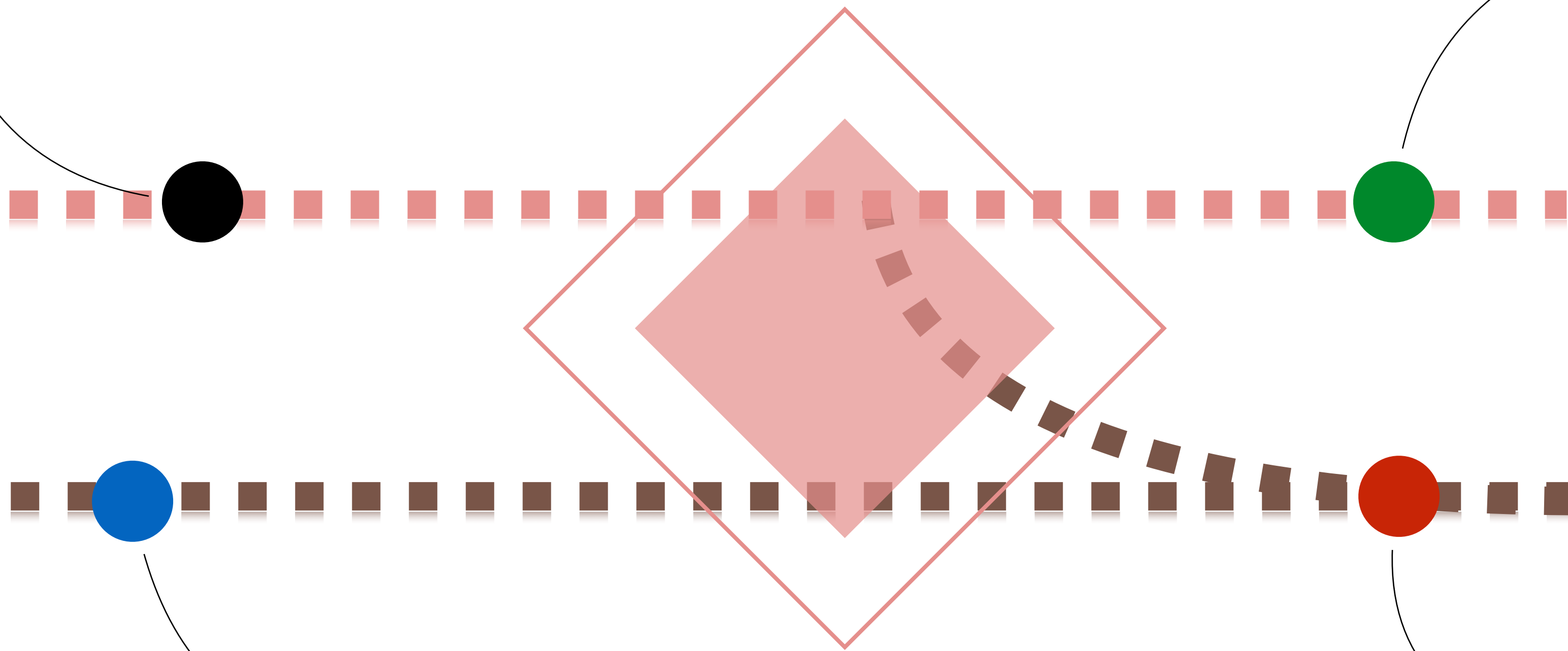
Exhaustive Failure Checking



Composing Can Be Tricky



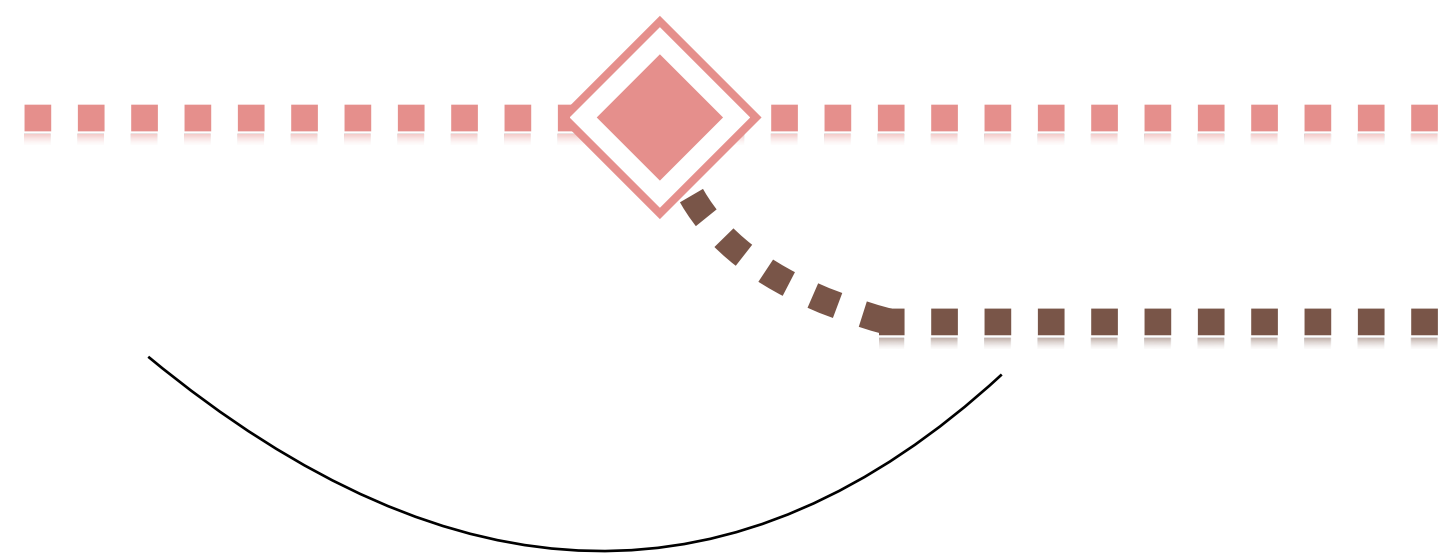
Input



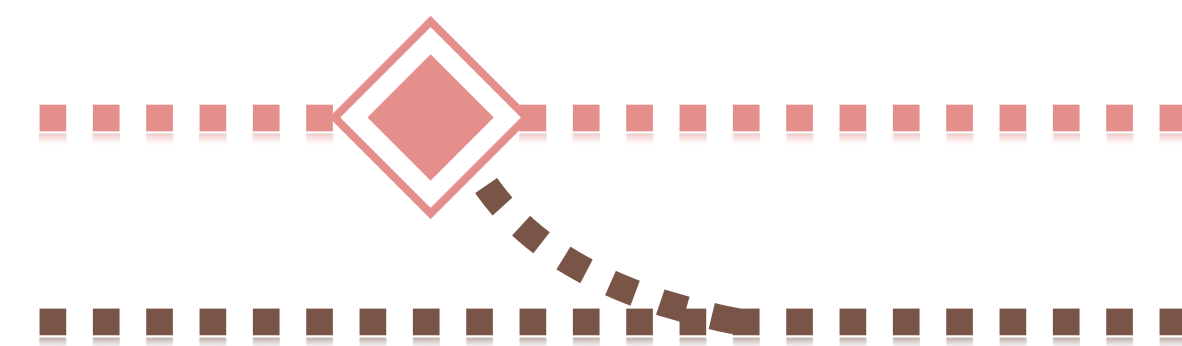
Success

Also input?

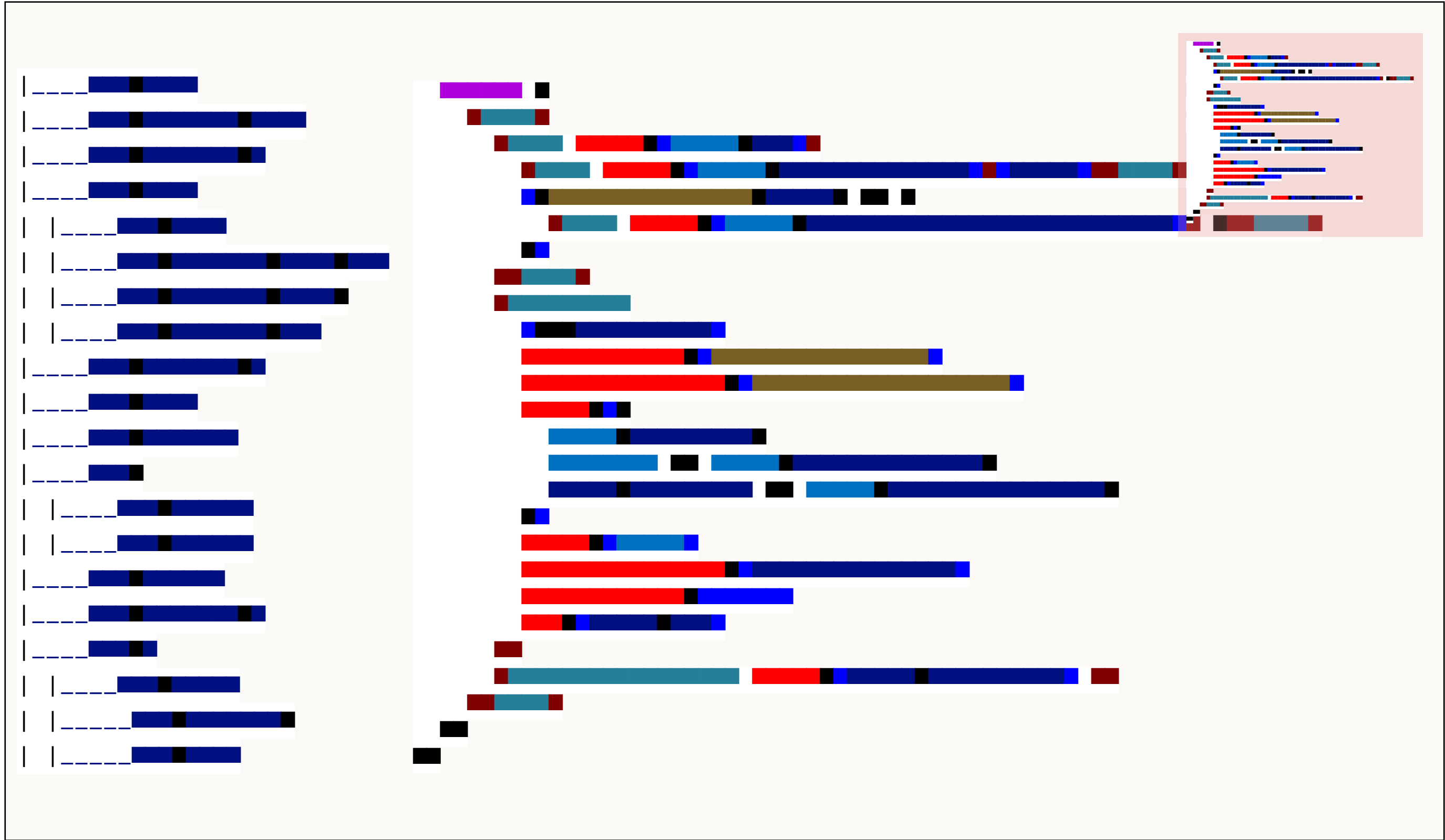
Failure



Bypass



DEMO





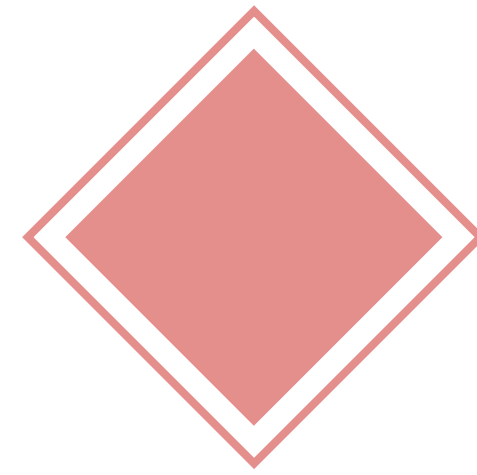
Promise.resolve()

YAGNI...?

You Ain't Gonna Need It... ?



PROMISES



Limited API

PROMISES API

Methods

Promise#then

Promise#catch

Promise#finally

Combinators

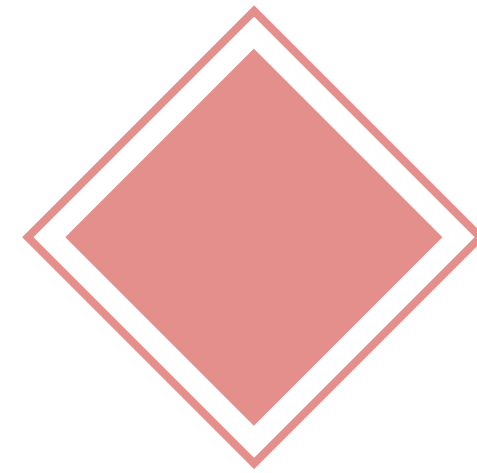
Promise.race

Promise.allSettled

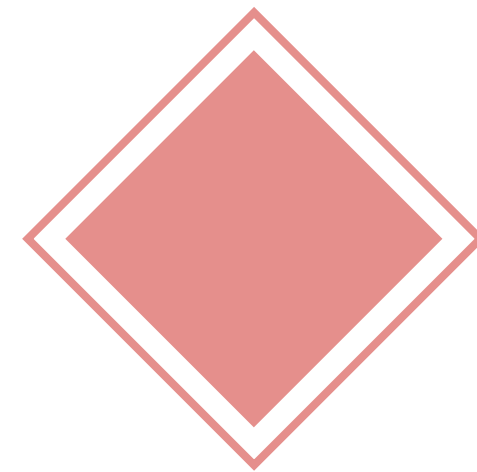
Promise.all

Promise.any

PROMISES



Limited API



Domain × Panic

DOMAIN

ERRORS

PANIC



NO EXHAUSTIVE CHECKS

INVALID INPUT DATA

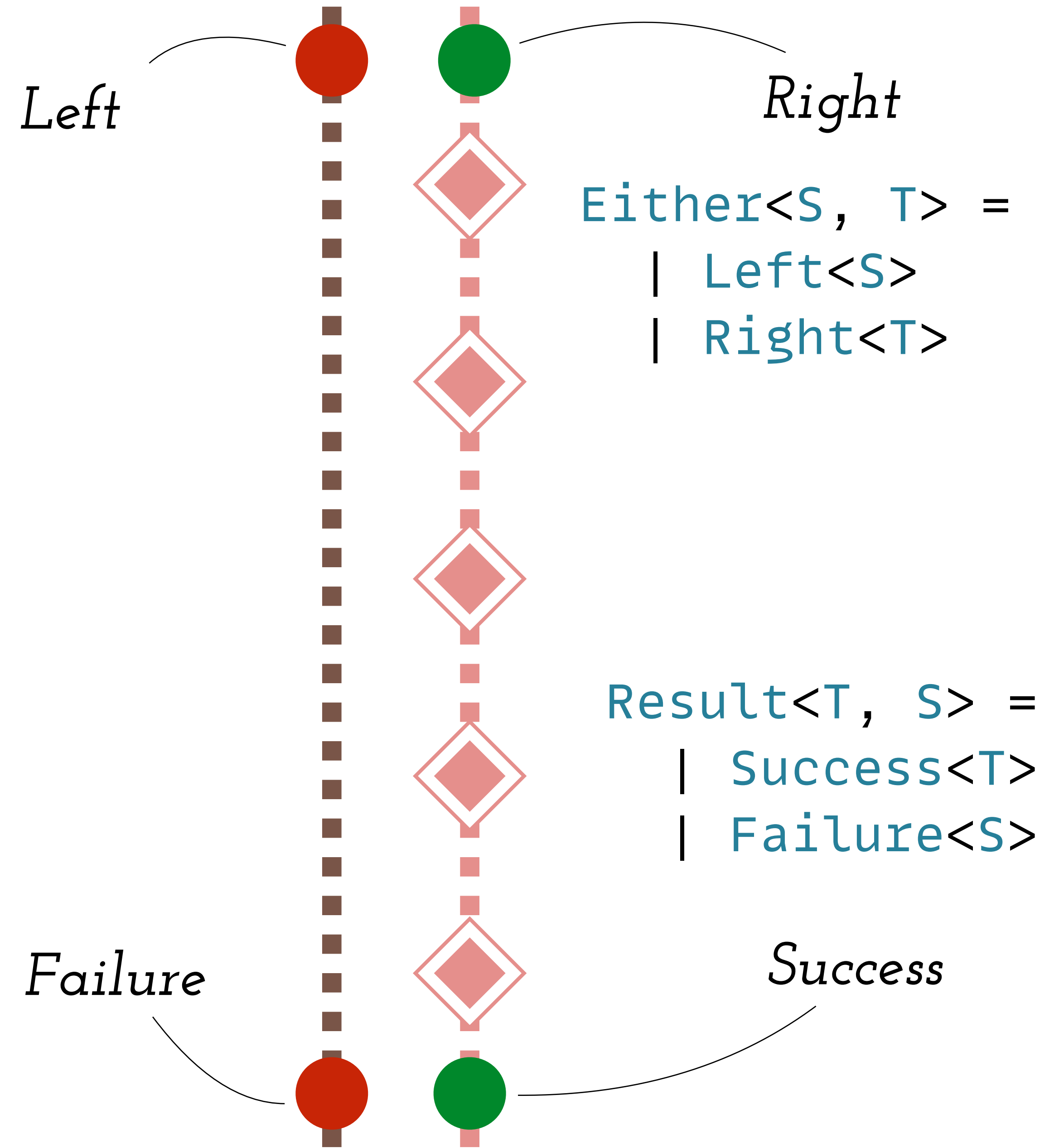
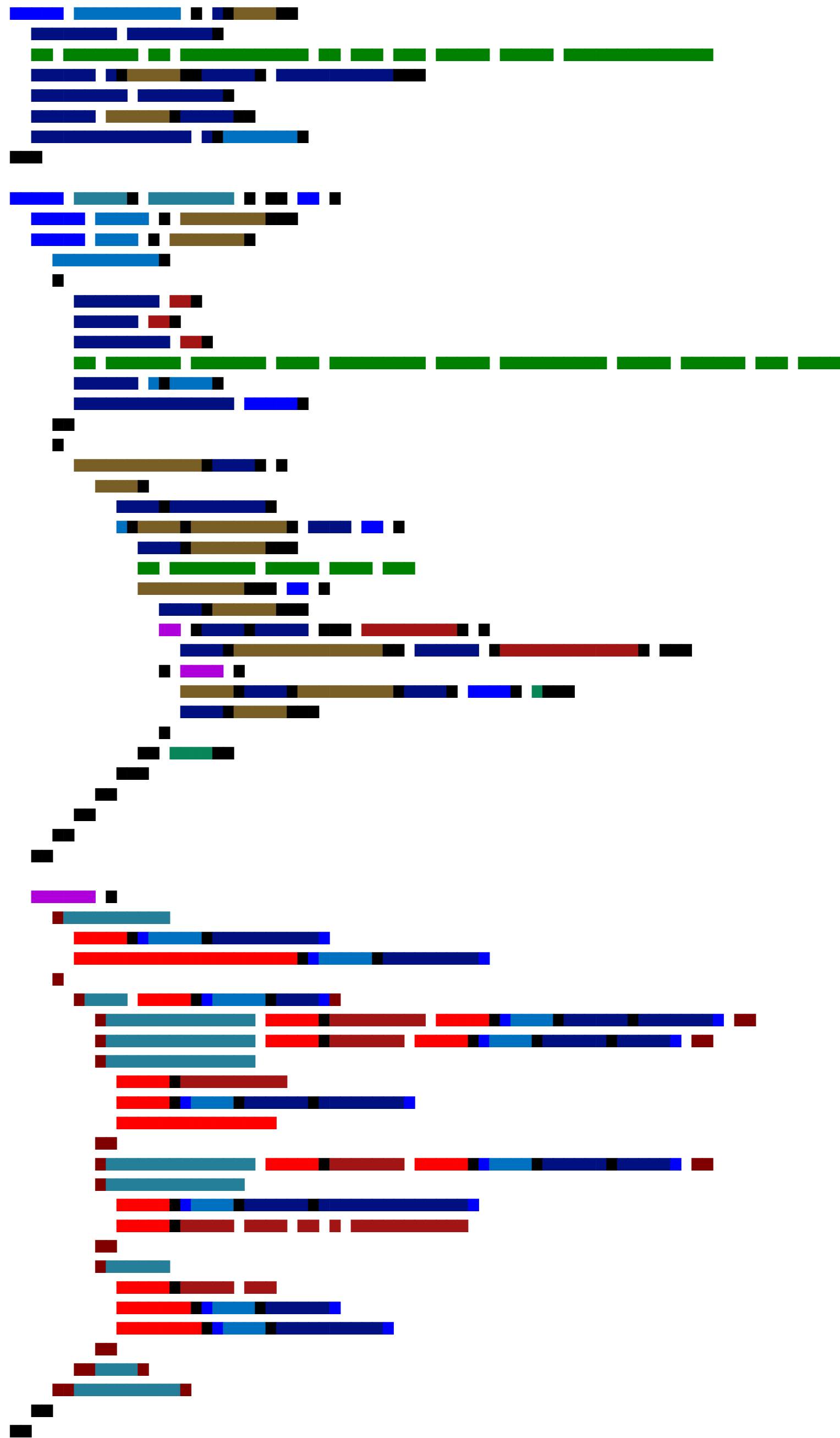
EXTERNAL SYSTEM UNREACHABLE

OUT OF MEMORY

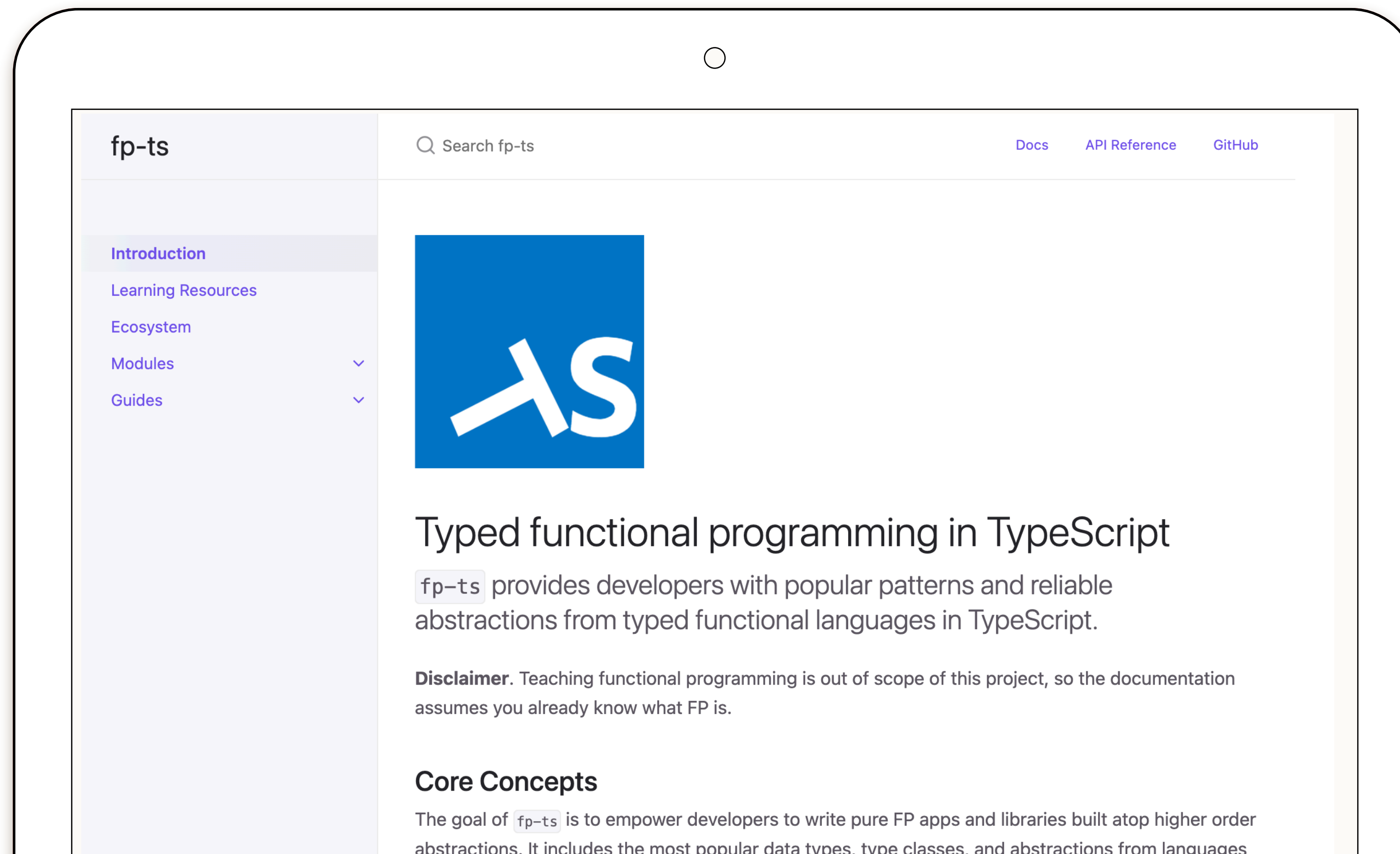
DIVIDE BY ZERO



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



DEMO



The screenshot displays the fp-ts website. At the top left is the 'fp-ts' logo. A search bar with the placeholder 'Search fp-ts' is located at the top center. On the top right, there are links for 'Docs', 'API Reference', and 'GitHub'. A left-hand navigation menu lists 'Introduction', 'Learning Resources', 'Ecosystem', 'Modules', and 'Guides'. The main content area features a blue square with the white TypeScript 'TS' logo. Below the logo is the title 'Typed functional programming in TypeScript'. The introductory text states that fp-ts provides developers with popular patterns and reliable abstractions from typed functional languages in TypeScript. A disclaimer follows, noting that teaching functional programming is out of scope. The 'Core Concepts' section begins by stating the goal of fp-ts is to empower developers to write pure FP apps and libraries built atop higher order abstractions, including popular data types, type classes, and abstractions from languages.

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 array and returns the total. Demonstrates simple
// recursion.
function totalForArray(arr, total) {
  currentTotal = addNumbers(currentTotal, arr.shift());
}

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

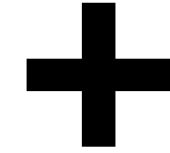
// Or you could just use
function totalForArray(arr) {
  return arr.reduce(addNumbers, 0);
}

function reallyTotal(arr) {
  return arr.reduce(addNumbers, 0);
}

function average(arr) {
  return count / total;
}

function average(arr) {
  return arr.reduce(addNumbers, 0) / arr.length;
}

// Gets the property associated with the property of an object. Used for
// functions that return objects like map, hence the name.
function getProperty(obj, propName) {
  return obj[propName];
}
```



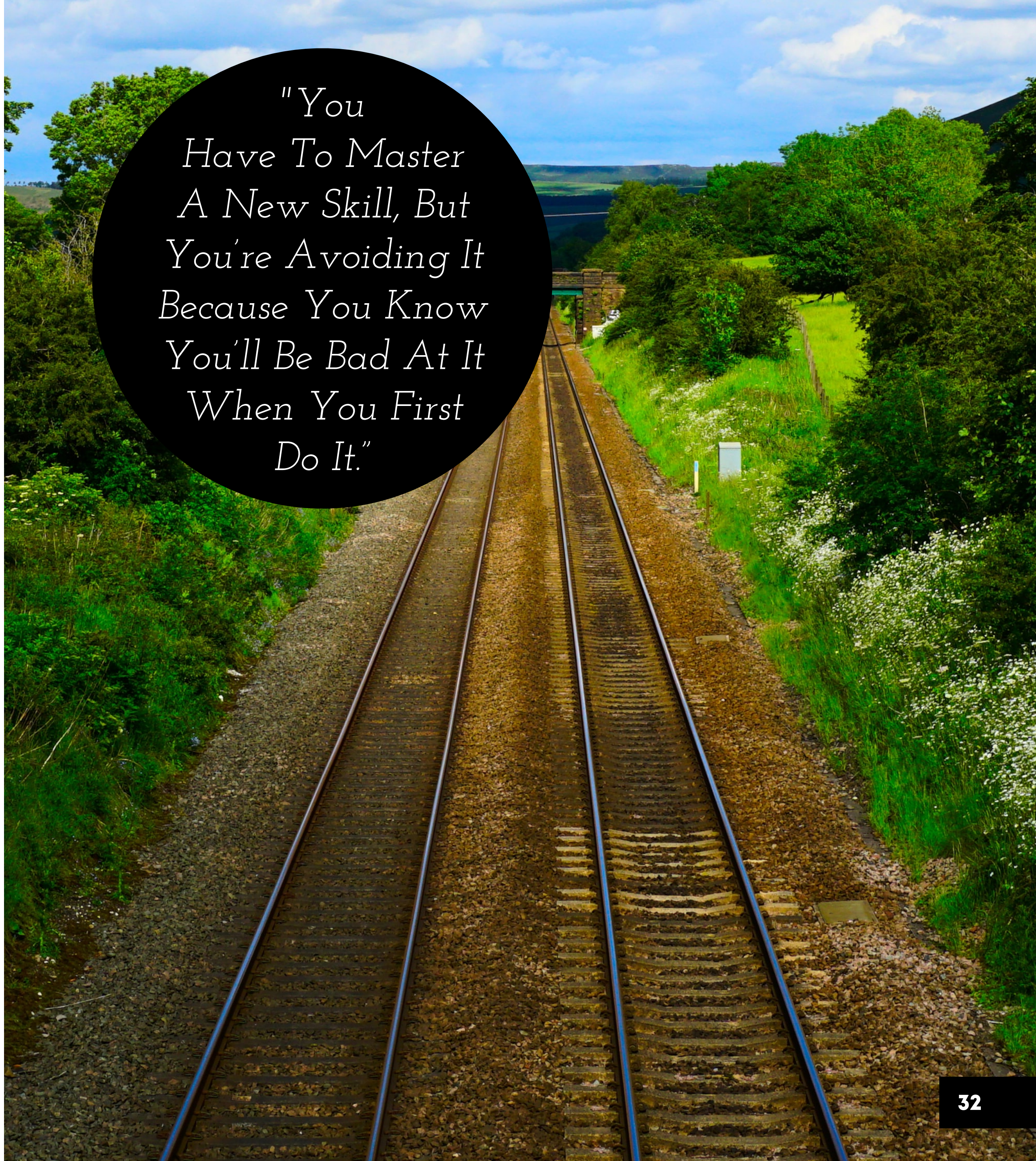
INNOVATION

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."



A steam locomotive is crossing a large, multi-arched stone viaduct in a mountainous landscape. The locomotive is emitting a large plume of white steam that fills the upper portion of the frame. The viaduct is constructed from dark stone and features several prominent arches. The background consists of rolling green hills and mountains under a hazy sky. The overall scene is captured in a cinematic, slightly desaturated style.

**RAILWAY ORIENTED
TYPESCRIPT**

@robinpokorny

Klarna.

