RAILWAY ORIENTED PROGRAMMING

KOMPLEXE ORCHESTRIERUNG WARTBAR MACHEN

Patrick Drechsler

.NET User Group Ulm

15.10.2018 @drechsler draptik

Patrick Drechsler

- C# Entwickler
- Schwerpunkte: DDD, FP
- Softwerkskammer

AGENDA

- Problemstellung
- Live Coding (ohne ROP)
- Bilder
- Live Coding (mit ROP)
- FP Basics
- Recap

WIR SIND FAUL

WARTBAREN CODE SCHREIBEN !

(dein zukünftiges Ich wirds dir danken)

WAS IST "ORCHESTRIERUNG"?

- Code...
  - mit wenig interner Logik
  - bei dem viel "zusammenläuft":
    - viele Abhängigkeiten
    - oft in "Service" Klassen
    - beschreibt oft den Ablauf einer User Story...

USER STORY: ANMELDUNG ALS NEUER BENUTZER

Wenn ein neuer Benutzer sich anmeldet,

- werden seine Eingaben validiert
- wird er im System gespeichert
- erhält er eine Bestätigungsmail

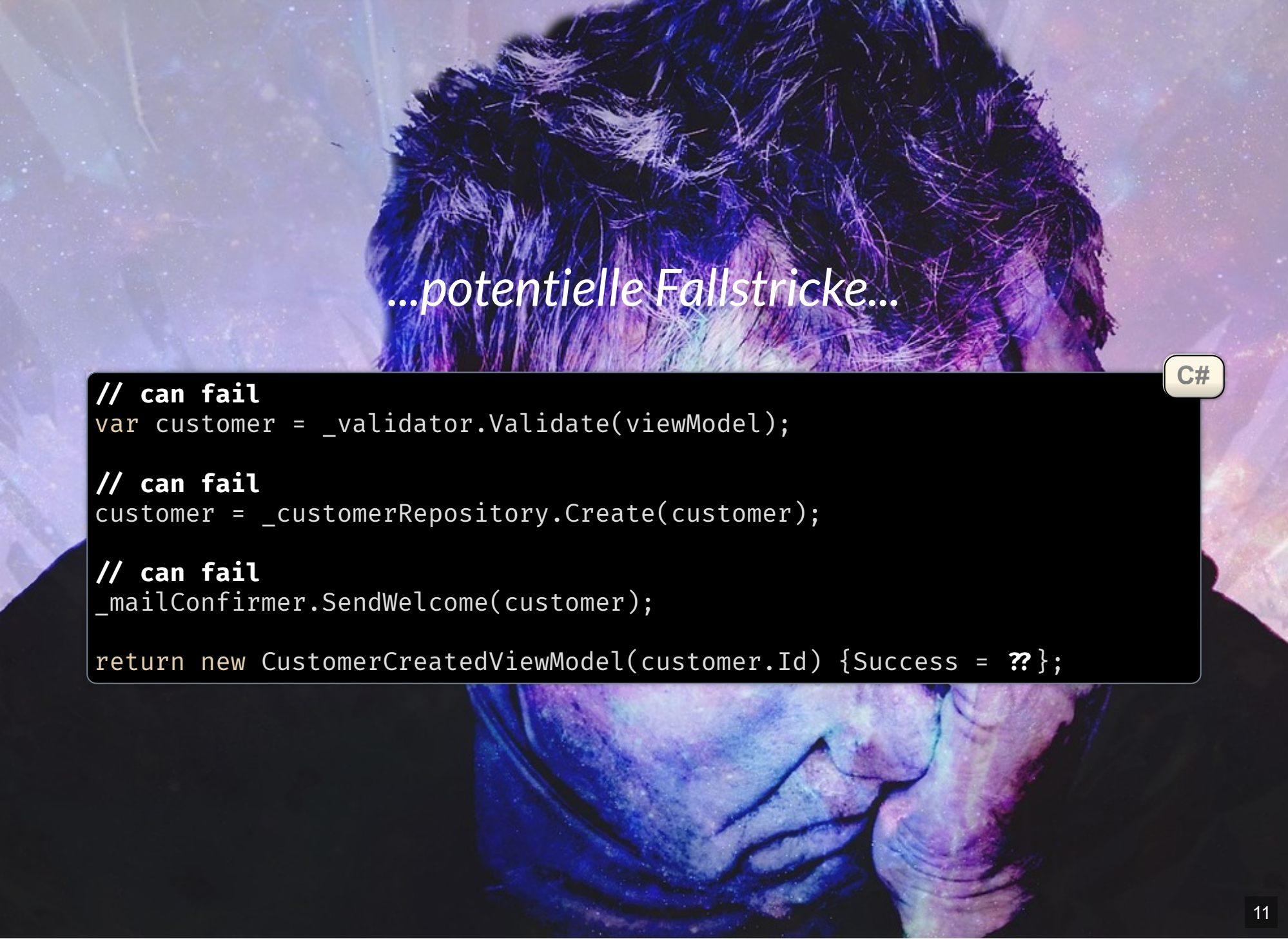
```
public CustomerCreatedViewModel RegisterCustomer(SomeVM viewModel)
{
    var customer = _validator.Validate(viewModel);
    customer = _customerRepository.Create(customer);
    _mailConfirmer.SendWelcome(customer);

    return new CustomerCreatedViewModel(customer);
}
```

C#

- Cool, wir sind fertig!
- let's go live...

...NO ERROR HANDLING...

WHAT COULD POSSIBLY GO WRONG?



...potentielle Fallstricke...

C#

```
// can fail
var customer = _validator.Validate(viewModel);

// can fail
customer = _customerRepository.Create(customer);

// can fail
_mailConfirmer.SendWelcome(customer);

return new CustomerCreatedViewModel(customer.Id) {Success = ??};
```

PRO-TIPP

GEWÜNSCHTES FEHLERVERHALTEN ABKLÄREN

- Nicht einfach drauflos programmieren:
  - Zuerst mit Kunde/Domain-Experten klären!
  - Dann die User Story aktualisieren (oder neue User Story für Fehlerfälle erstellen)

FEHLERBEHANDLUNG

C#

```
Customer customer;
try { customer = _validator.Validate(createCustomerViewModel); }
catch (Exception e) { return CreateErrorResponse(e); }

try { customer = _customerRepository.Create(customer); }
catch (Exception e) { return CreateErrorResponse(e); }

try { _mailConfirmer.SendWelcome(customer); }
catch (Exception e)
{
    // don't fail, but maybe: logging, retry-policy
}

return new CustomerCreatedViewModel(customer.Id);
```

PRO TIPP: AVOID EXCEPTIONS

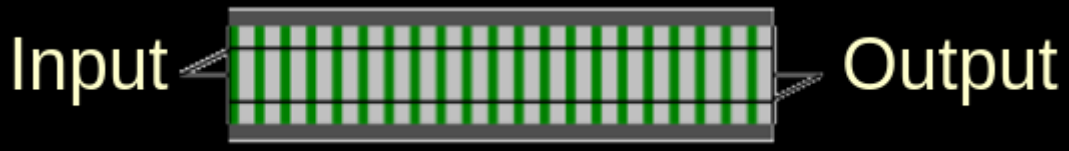
- **throw** is worse than **goto**
- Exceptions are **new user stories**

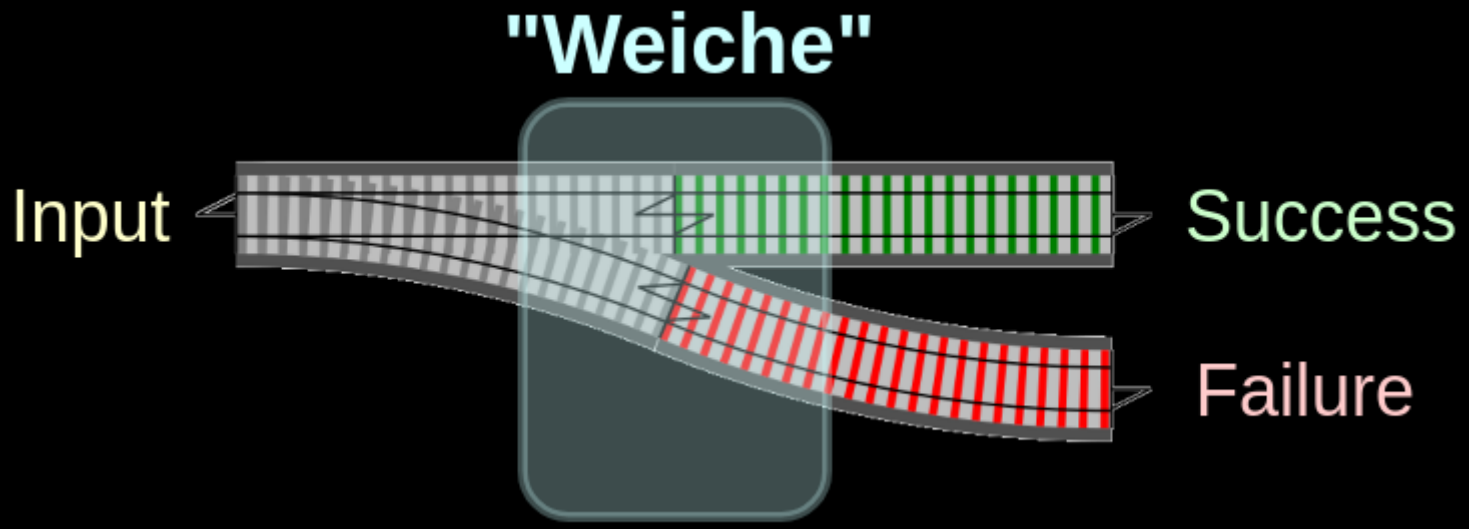
Fehlerbehandlung macht (oft) Großteil des Codes aus

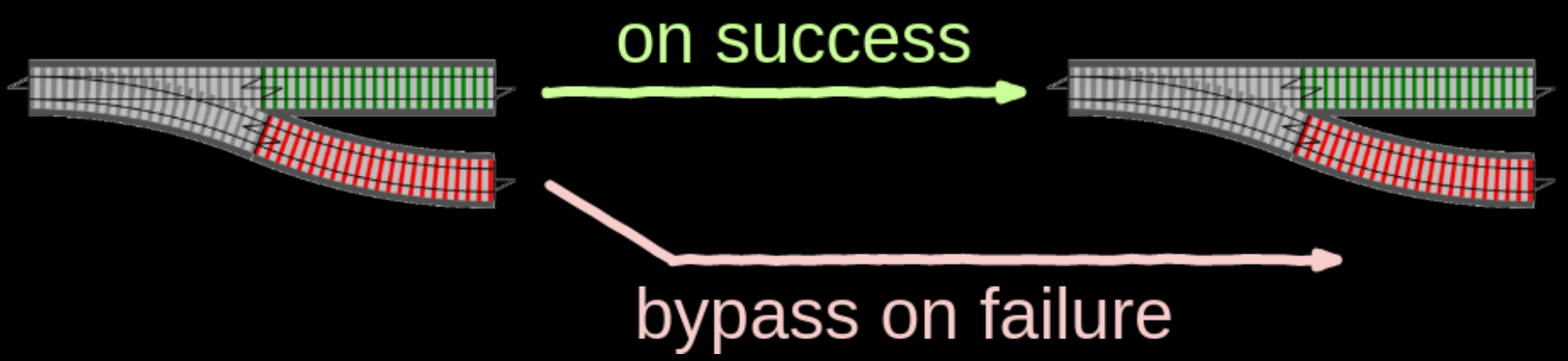
LIVE CODING

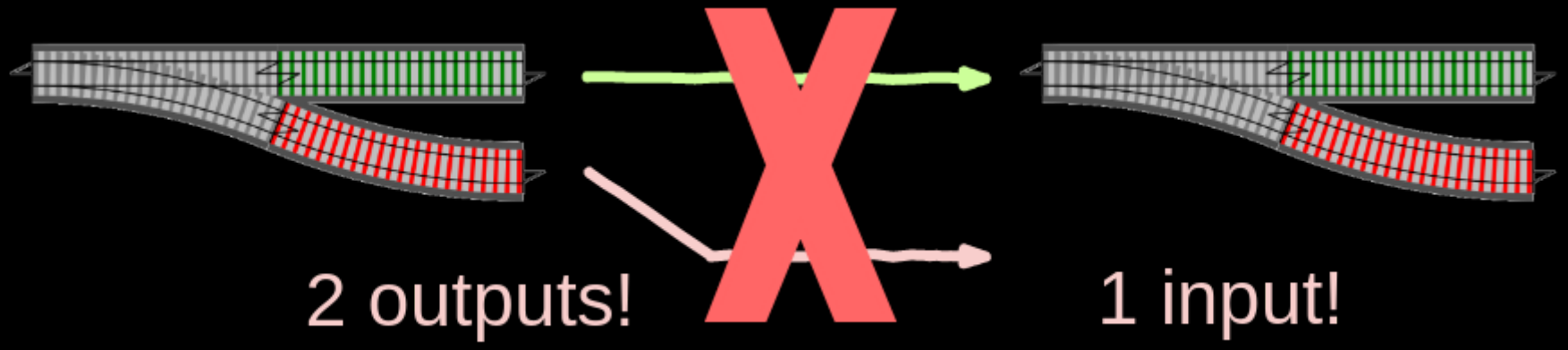
(ohne ROP)


WHERE ARE THE RAILWAYS?
YOU PROMISED TRAINS!

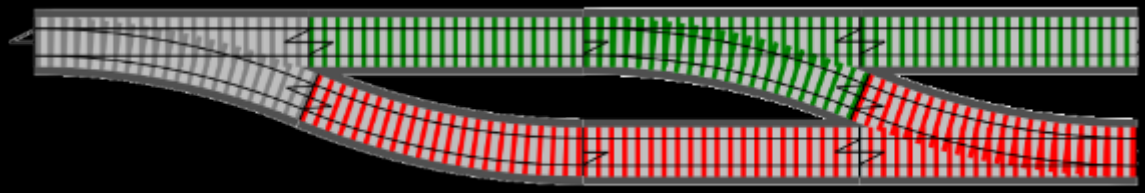

Input
Output


"Weiche"
Input
Success
Failure

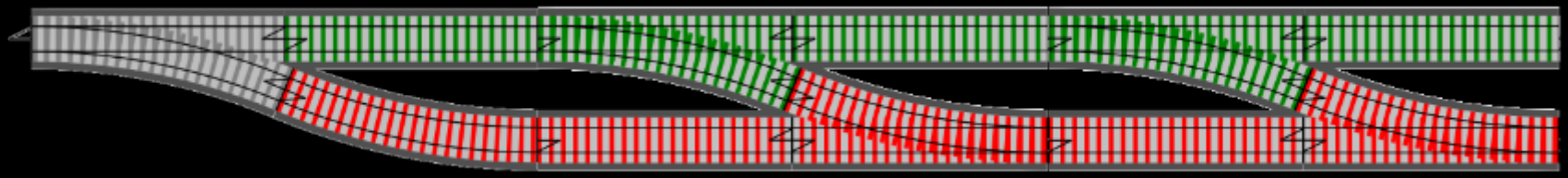

on success
bypass on failure


2 outputs!
1 input!

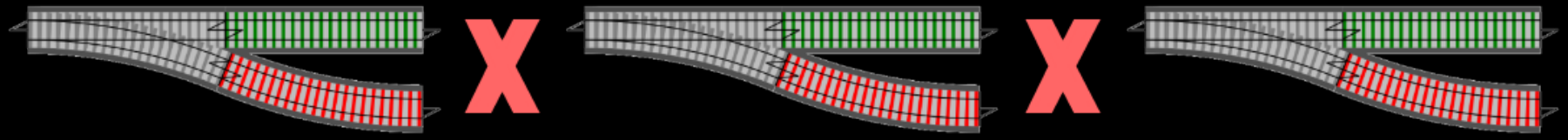
einfach: F2 kann Failure empfangen:



dann kann man weiterarbeiten:

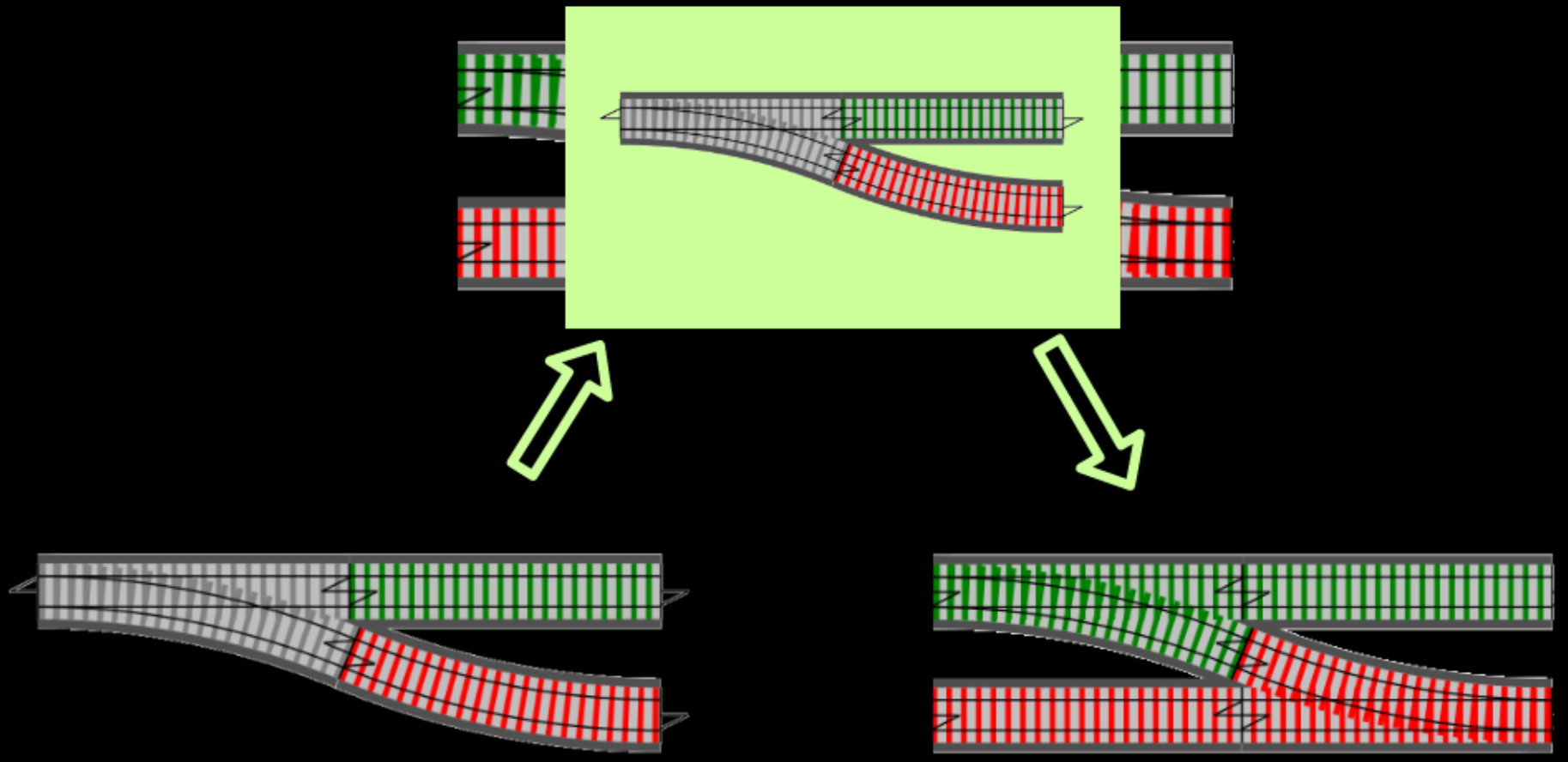


Ups: F2, F3 können keinen Fehler entgegennehmen:



wir brauchen eine Funktion, die Fehler entgegennimmt:

Umwandeln von 1-Track Input in 2-Track Input mit einem **"Adapter Block"**

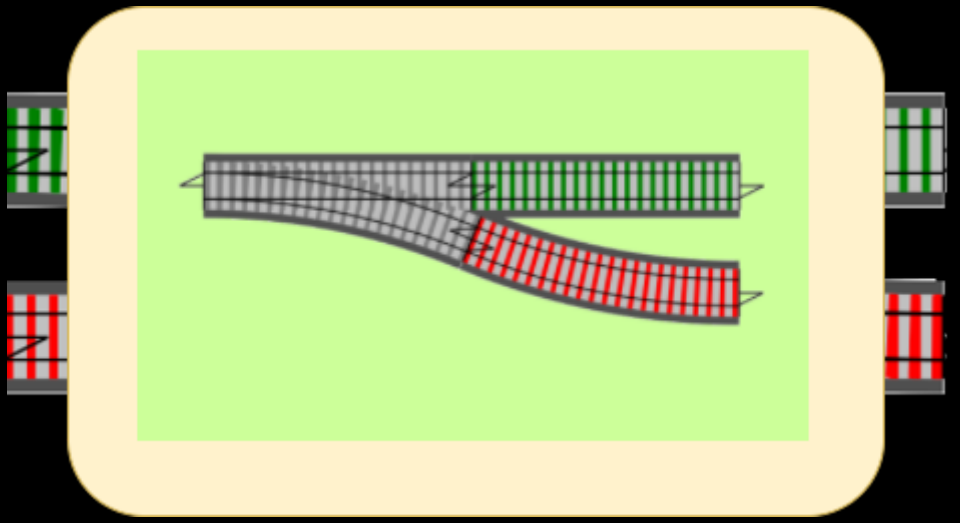


Adapter Block ist ein Wrapper fuer eine Methode, die

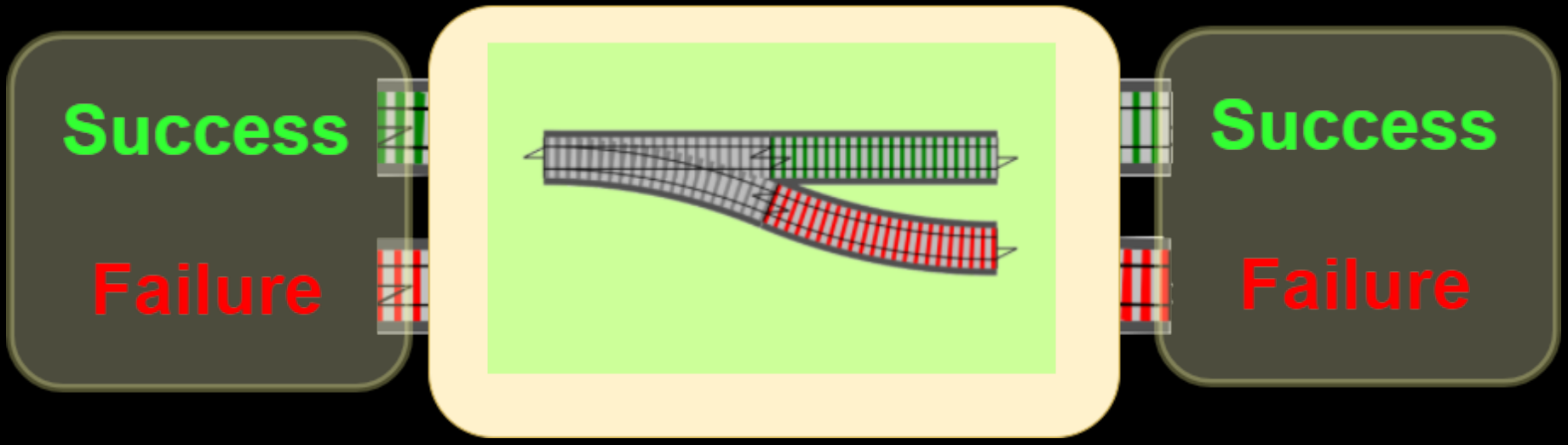
- **1 Eingabe** und
- 2 Ausgaben

hat

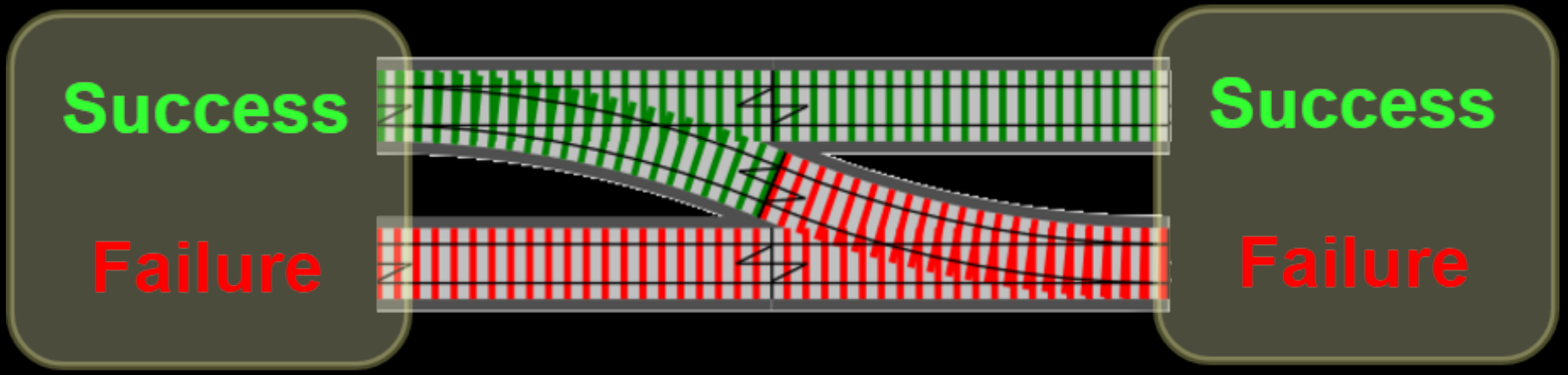
Ein Adapter kann **2 Eingaben** entgegennehmen



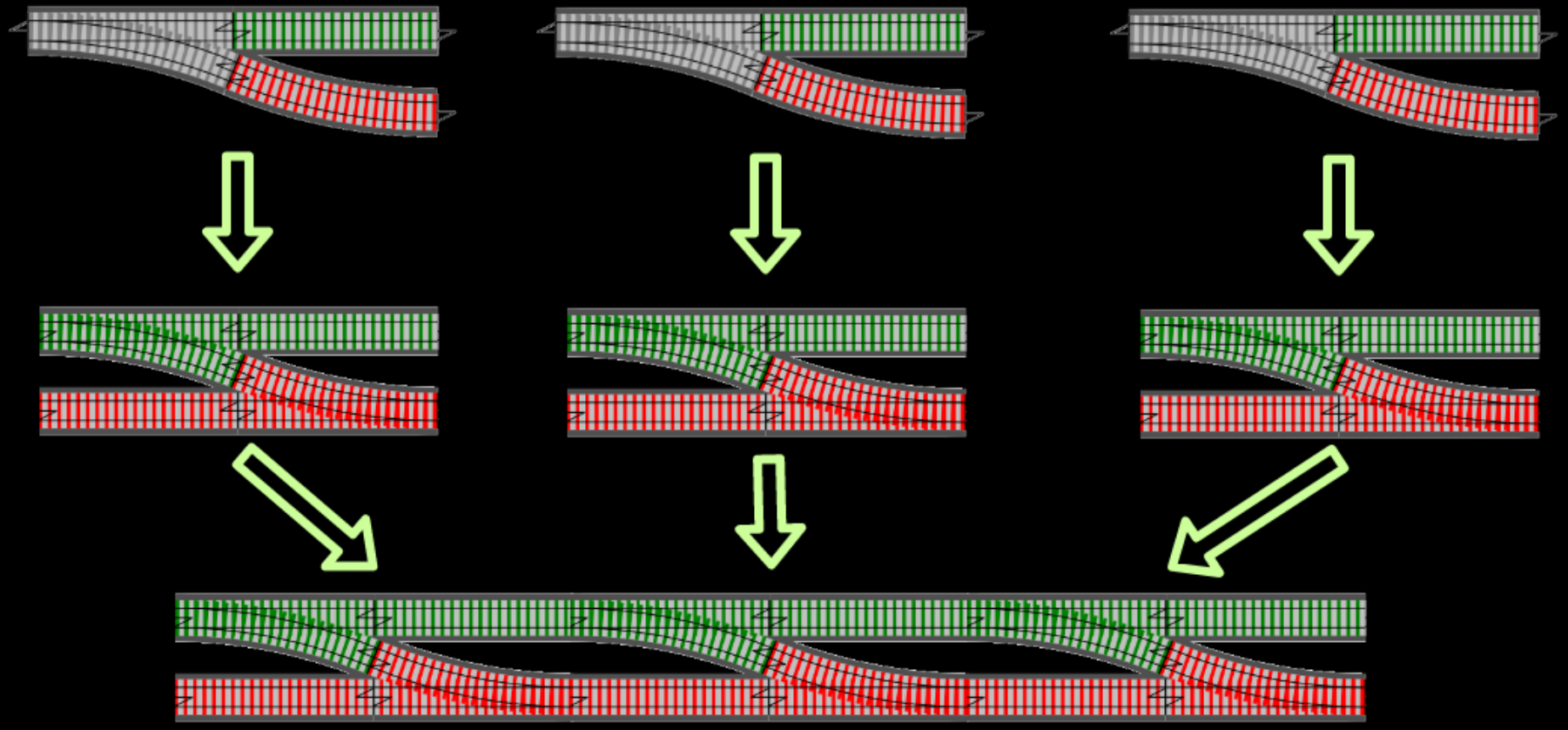
"Result"



"Result"



Ziel:



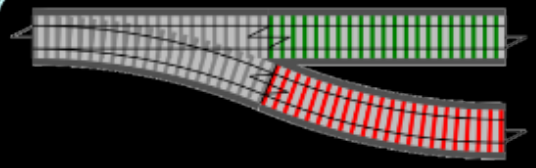
2 Track Model

ARBEITEN MIT RESULTS

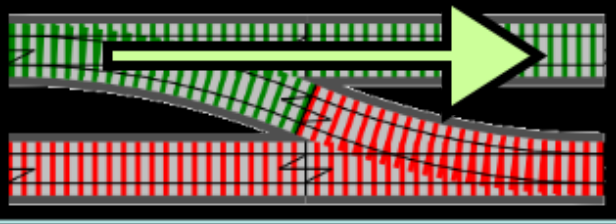
(via Extension Methods)

- **OnSuccess**
- **OnBoth**
- **OnFailure**

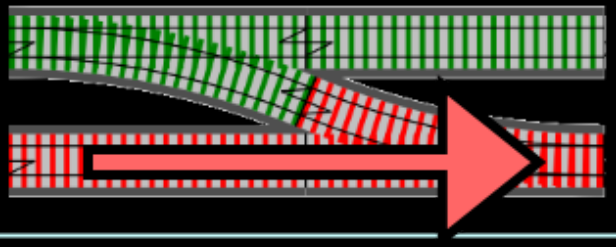
Hinweis: Extension Methods in C# sind wie "traits" (Scala) oder "mixins" (Ruby)



Creating a Result
(Customer -> Result<Customer>)

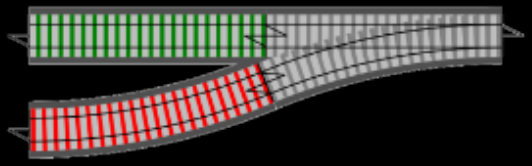


"OnSuccess"



"OnFailure"

Convert Result<T> back to T
"OnBoth"



VERKETTEN VON RESULT

```C#
static Result<U> OnSuccess(this R<T> result,
                           Func<T, U> func) { /* ... */ }
```

```C#
static Result<T> OnFailure<T>(this Result<T> result,
                           Action<string> action) { /* ... */ }
```

```C#
static K OnBoth<T, K>(this Result<T> result,
                           Func<Result<T>, K> func) { /* ... */ }
```

- **"Result"** ist kein Sprachfeature von C# / Java ☹
- C#
  - CSharpFunctionalExtensions *
  - LaYumba.Functional
  - language-ext
- Java: auch möglich (Link im Abspann)
- F#, Rust: Sprachfeature
- JS: Promises

ERSTELLEN VON RESULT

C#

```
public Result<Customer> Validate(Customer customer)
{
    return IsValid(customer)
        ? Result.Ok(customer)                  // ← static ctor for success
        : Result.Fail<Customer>("invalid") // ← static ctor for failure
}
```

LIVE CODING

(mit ROP)

FUNKTIONALE PROGRAMMIERUNG

- Pure Functions
  - gleiche Eingabe gibt immer gleiches Ergebnis zurück
  - keine Seiteneffekte
- Higher Order Functions
  - Funktionen können als Eingabe- und Rückgabewert verwendet werden

"RESULT ZU FUSS"...

C#

```
public class Result {
    public bool Success { get; }
    public string Error { get; }

    protected Result(bool success, string error) { /* … */ }

    public static Result Fail(string message) { /* … */ }

    public static Result<T> Ok<T>(T value) {  /* … */ }
}
```

C#

```
public class Result<T> : Result {
    public T Value { get; }
    public bool IsFailure ⇒ !Success;

    protected internal Result(T value, bool success, string error)
        : base(success, error) {
        Value = value;
    }
}
```

Ausblick: F#...

Result ist mittlerweile ein Sprachfeature von F#, kann aber auch einfach selbst implementiert werden:

F#

```fsharp
// discriminated union
type Result<'TSuccess,'TFailure> =
    | Success of 'TSuccess
    | Failure of 'TFailure
```

"Discriminated Union" ist ein "Algebraic Data Type" (ADT)

```fsharp
let bind switchFunction twoTrackInput =
    // Pattern Matching
    match twoTrackInput with
    | Success s → switchFunction s
    | Failure f → Failure f
```

F#

bind kombiniert zwei 2-Track Funktionen ...

(entspricht OnSucces, OnFailure)

Anwendungsbeispiele

F#

```
type Request = {name:string; email:string} // ← Record type

let validate1 input =
   if input.name = "" then Failure "Name must not be blank"
   else Success input

let validate2 input =
   if input.name.Length > 50 then Failure "Name must not be longer ... "
   else Success input

let validate3 input =
   if input.email = "" then Failure "Email must not be blank"
   else Success input
```

F#

```
// Option 1
let combinedValidation =
    let validate2' = bind validate2
    let validate3' = bind validate3
    validate1 >> validate2' >> validate3'
```

F#

```
// Option 2
let combinedValidation =
    validate1
    >> bind validate2
    >> bind validate3
```

F#

```
// Option 3
let combinedValidation =
    validate1
    >=> validate2
    >=> validate3
```

https://fsharpforfunandprofit.com/posts/recipe-part2/

Haben wir unser Ziel erreicht?

C#

```
var customerResult = Validate(createCustomerViewModel);
var result = customerResult
    .OnSuccess(c ⇒ _customerRepository.Create(c))
    .OnSuccess(c ⇒ _mailConfirmer.SendWelcome(c))
    .OnBoth(resultAtEnd ⇒ resultAtEnd.IsSuccess
        ? new CustomerCreatedViewModel(resultAtEnd.Value.Id)
        : CreateErrorResponse(resultAtEnd.Error));
```

Scott Wlaschin
@ScottWlaschin

Following

Dunno why everyone thinks my railway-oriented post is trying to be a monad tutorial 🤔-- it ain't! It was written to help address a specific problem. To use a cooking analogy, it's all about the recipe, not the chemistry of the ingredients.

8:08 PM - 9 Oct 2018

Video:

Scott Wlaschin ("Mr. F#") on Monads

(2min)

014

How do I work with errors
in a functional way?

Wenn man verstanden hat, was eine Monade ist, verliert man die Fähigkeit zu erklären, was eine Monade ist.

(Monaden-Paradoxon)

Um **Railway Oriented Programming** zu praktizieren, muss man nichts über Funktoren, Monoiden oder Monaden wissen

FAZIT: RAILWAY ORIENTED PROGRAMMING

- lesbarer & wartbarer Code
- kompakte Fehlerbehandlung
- **Fehlerbehandlung wird Bestandteil der Domäne!**

...nebenbei haben wir Sinn und Zweck der "Either-Monade" verstanden... ☺

LINKS

- Scott Wlaschin "the original talk" http://fsharpforfunandprofit.com/rop/
- Vladimir Khorikov "Functional C#: Handling failures" http://enterprisecraftsmanship.com/2015/03/20/functional-c-handling-failures-input-errors/
- C# Bibliotheken
  - CSharpFunctionalExtensions https://github.com/vkhorikov/CSharpFunctionalExtensions
  - LaYumba.Functional https://github.com/la-yumba/functional-csharp-code
  - language-ext https://github.com/louthy/language-ext

DANKE!

- patrick.drechsler@redheads.de
- @drechsler
- draptik