

RAILWAY ORIENTED PROGRAMMING

KOMPLEXE ORCHESTRIERUNG WARTBAR MACHEN

Patrick Drechsler

#dwx2018



25.06.2018



@drechsler



github.com/draptik

Patrick Drechsler

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

DISCLAIMER

Ich werde nicht erklären, was eine Monade ist

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

(Monaden-Paradoxon)

Video:

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

(2min)

How do I work with errors
in a functional way?

Begriffe wie Functor, Monoid und Monade
brauchen wir nicht

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 (z.B. RegistrationService)
 - 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

Unser Ziel:

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));
```

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

    return new CustomerCreatedViewModel(customer);
}
```

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

...NO ERROR HANDLING...

**WHAT COULD POSSIBLY
GO WRONG?**

...potentielle Fallstricke...

C#

```
// can fail
var customer = Validate(createCustomerViewModel);

// 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 = 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);
```

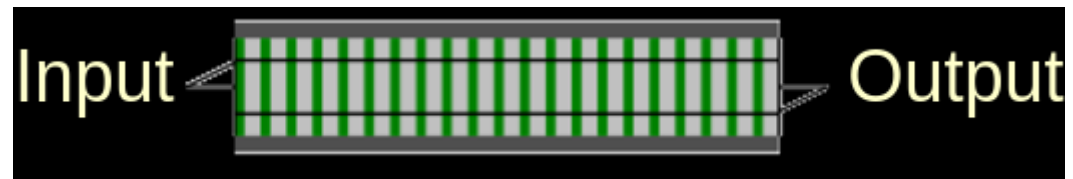
- Fehlerbehandlung macht einen Großteil des Codes aus
- Ergebnis einer Aktion ist oft Grundlage für weitere Aktion
- Exceptions: **throw** ist schlimmer als **goto**!

FUNKTIONALE PROGRAMMIERUNG

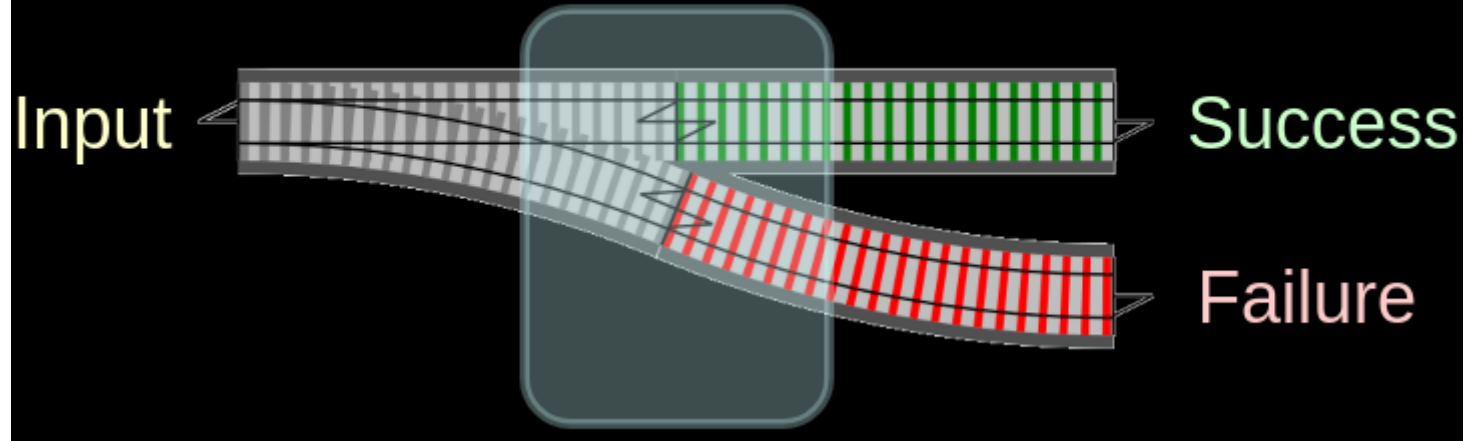
- Pure Functions
 - gleiche Eingabe gibt immer gleiches Ergebnis zurück
 - keine Seiteneffekte
- Higher Order Functions
 - Funktionen können als Eingabe- und Rückgabewert verwendet werden

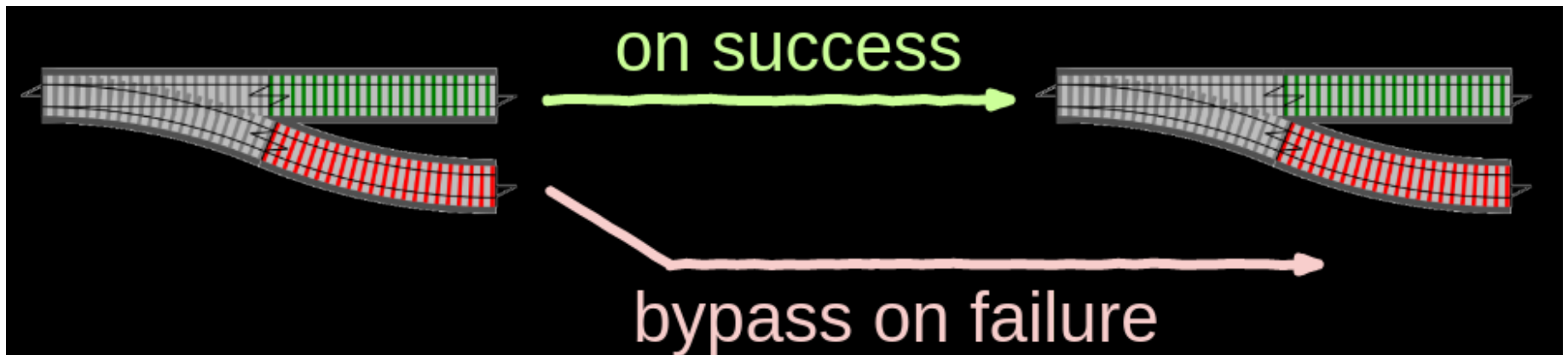
A still from the TV show The Big Bang Theory featuring Sheldon Cooper. He is sitting at his desk, looking off to the side with a frustrated expression. He is wearing a black t-shirt with a blue and yellow graphic of a car. The background shows his cluttered desk with a laptop, papers, and a mouse.

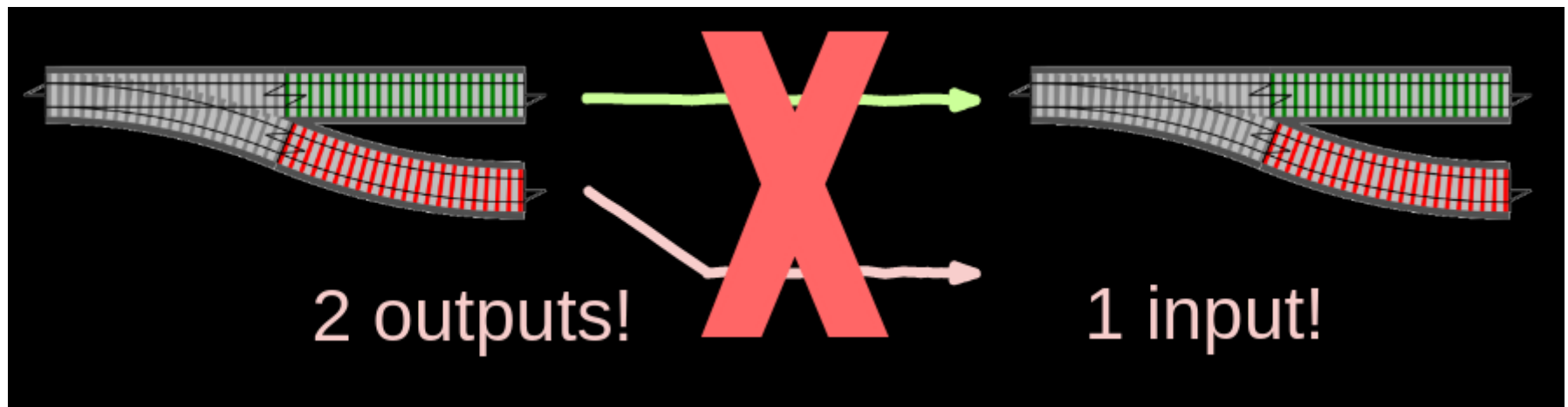
WHERE ARE THE RAILWAYS?
YOU PROMISED TRAINS!



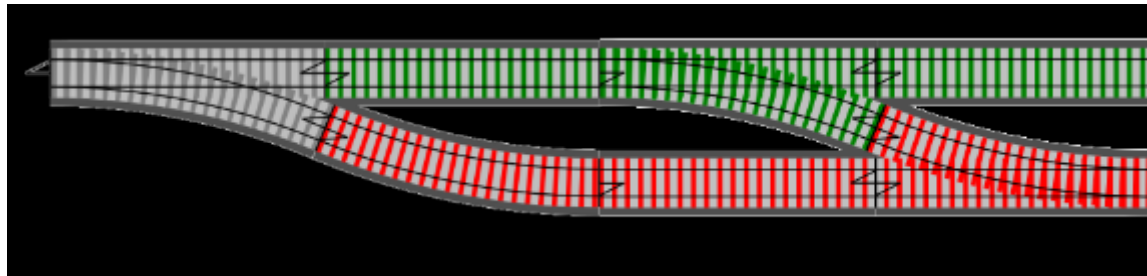
"Weiche"



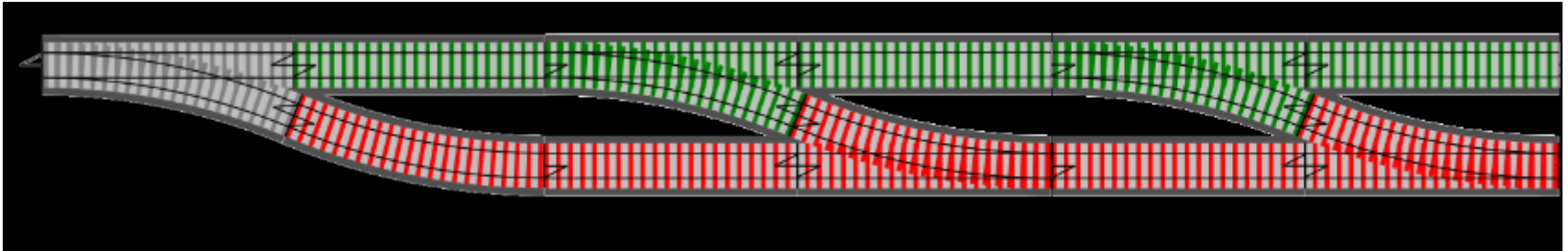




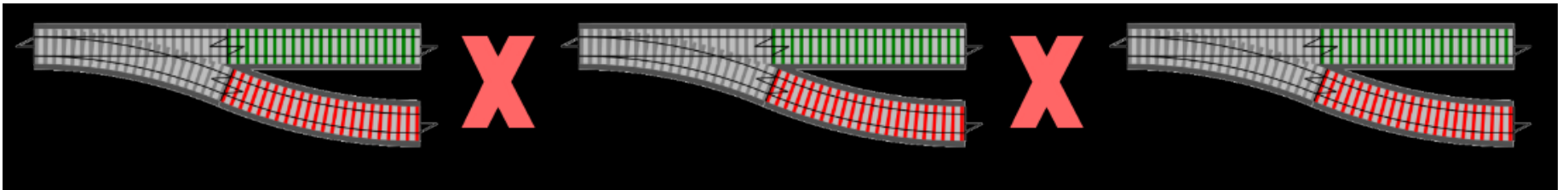
einfach: F2 kann Failure empfangen:



dann kann man weiterarbeiten:



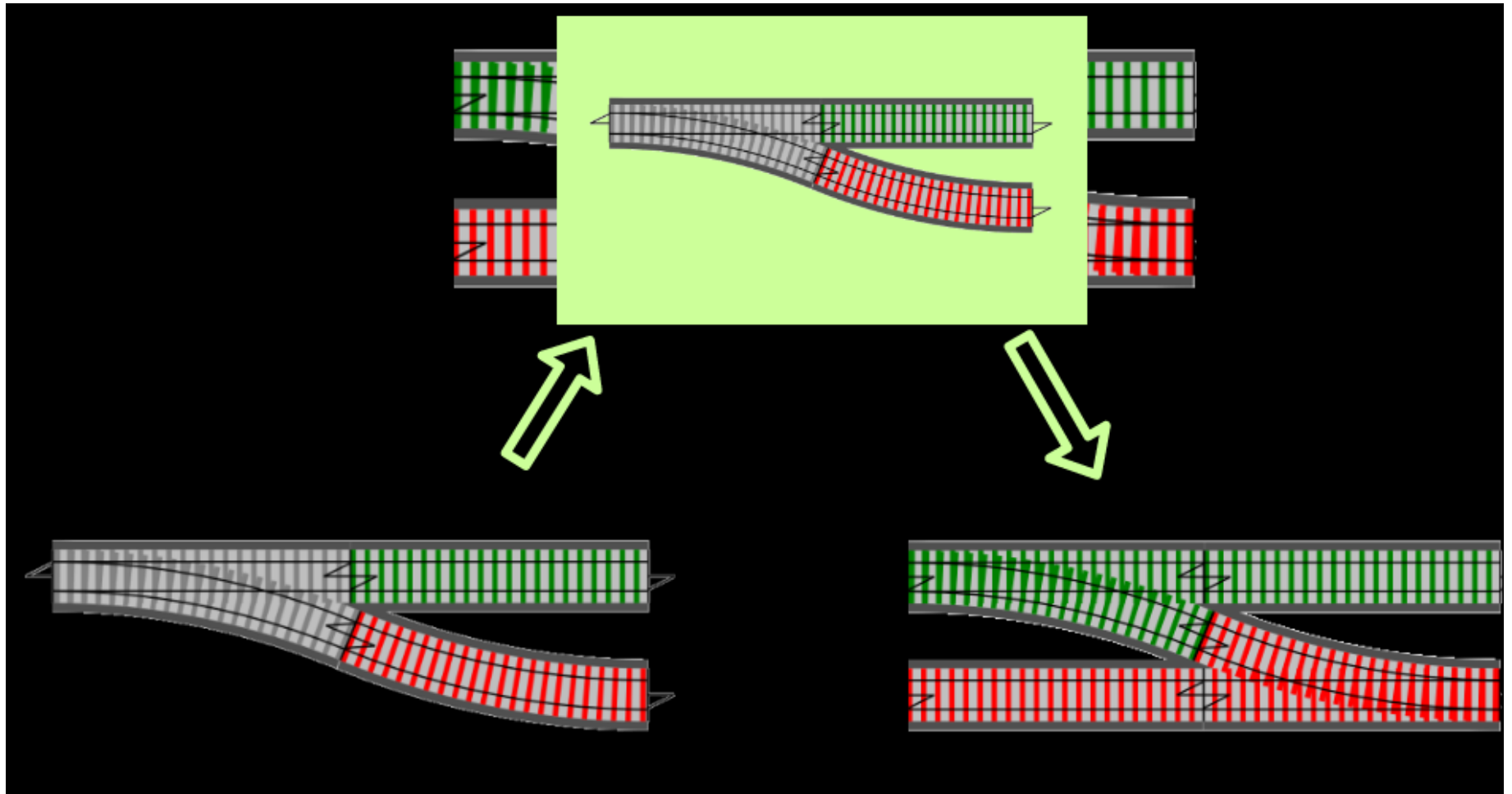
Ups: F2, F3 können keinen Fehler entgegennehmen:

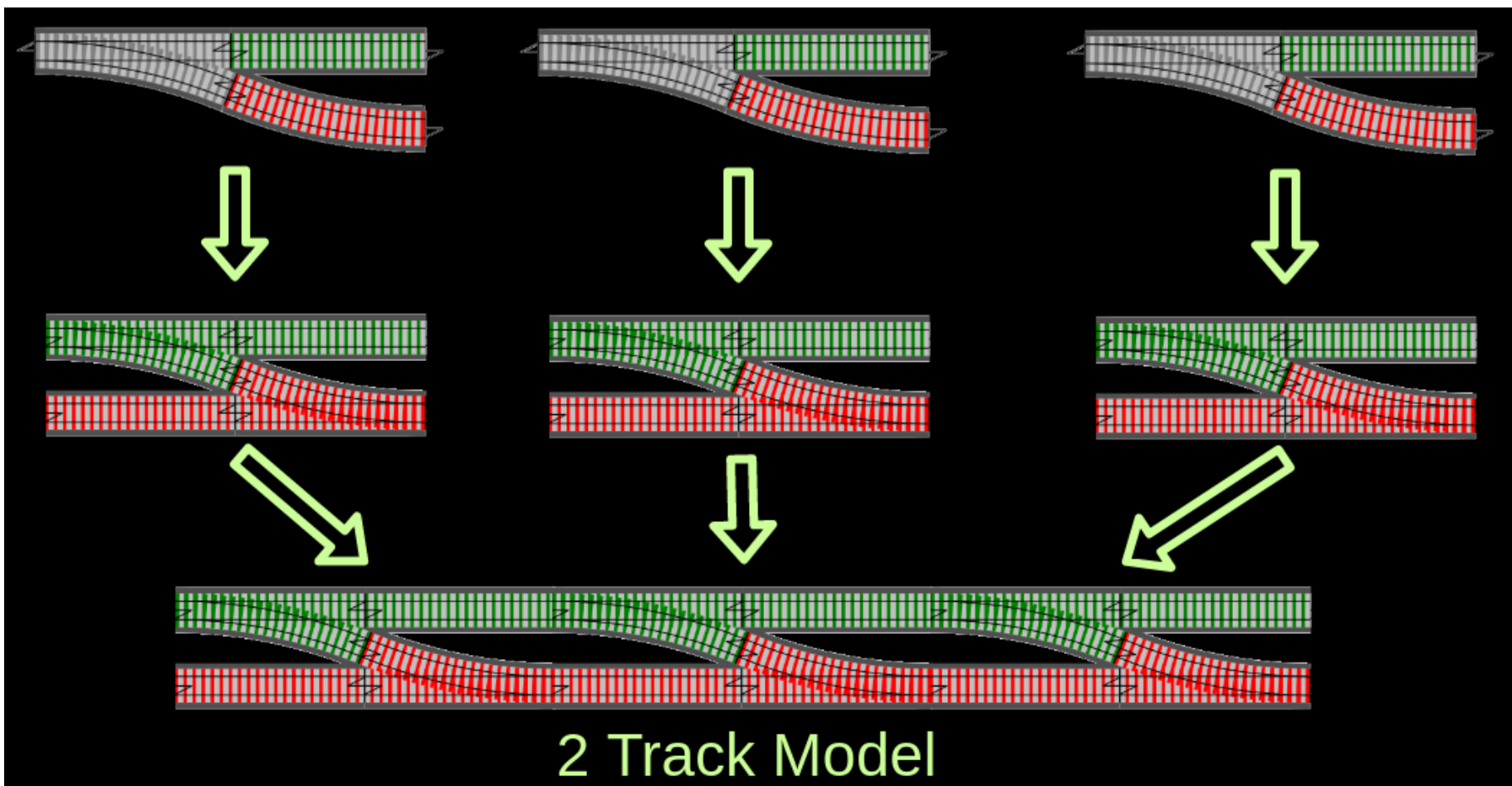


wir brauchen eine Funktion, die Fehler entgegennimmt:

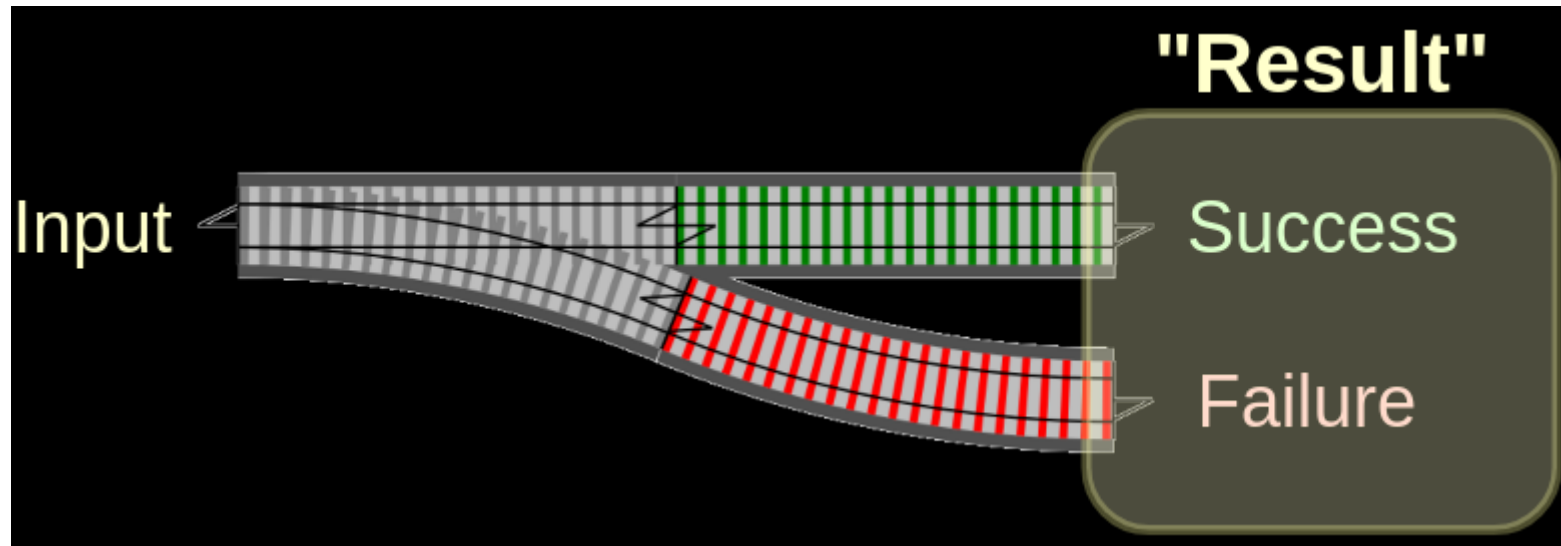


Umwandeln von 1-Track Input in 2-Track Input mit einem "Adapter Block"





"Result" kapselt Success und Failure



- **"Result"** ist kein Sprachfeature von C# / Java



- C#
 - CSharpFunctionalExtensions*
 - LaYumba.Functional
 - language-ext
- Java: auch möglich (Link im Abspann)
- F#, Rust: Sprachfeature
- JS: Promises



Basteln wir uns ein **Result**...

"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;  
    }  
}
```

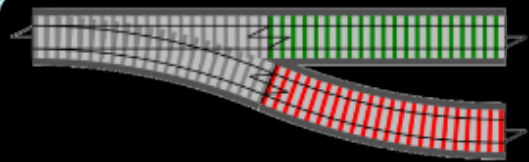

ERSTELLEN VON RESULT

C#

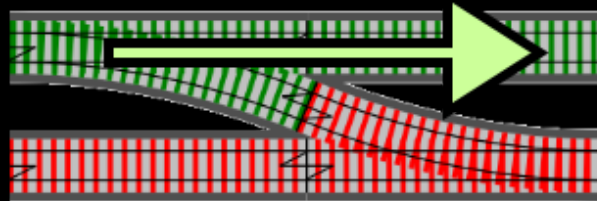
```
public Result<Customer> Validate(Customer customer) {  
    return IsValid(customer)  
        ? Result.Ok(customer) // ← static ctor for success  
        : Result.Fail("invalid") // ← static ctor for failure  
}
```

LIVE CODING

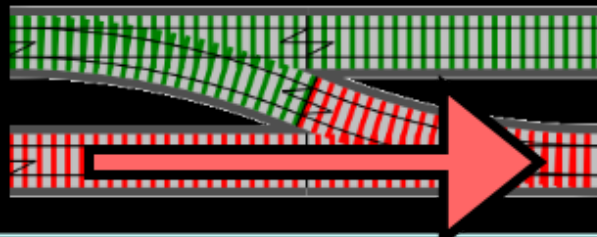
(Result Klasse zu Fuß)



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

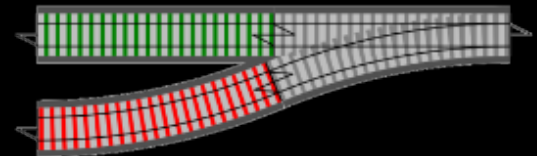


"OnSuccess"



"OnFailure"

Convert Result<T> back to T
"OnBoth"



KOMBINATION VON RESULTS

(via Extension Methods)

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

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

VERKETTEN VON RESULT

```
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) { /* ... */ }
```

C#

LIVE CODING

(Beispielcode)

Ausblick: F#...

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

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

F#


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

bind kombiniert zwei 2-Track Funktionen ...

(entspricht OnSuccess, 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
```

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));
```

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/>
- Stefan Macke "ROP für Java"
<https://www.heise.de/developer/artikel/Railway-Oriented-Programming-in-Java-3598438.html>
- 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