

RAILWAY ORIENTED PROGRAMMING

KOMPLEXE ORCHESTRIERUNG WARTBAR MACHEN

Patrick Drechsler

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)

3min monads fun

Scott Wlaschin

start at: 14.30 end at: 17.30

Alle Beispiele sind in **C#**
(geht aber auch alles mit Java)

Umfrage:

- Java?
- C#?
- Javascript/Typescript?
- Haskell (oder Lisp, OCaml, etc)?

ORCHESTRIERUNG

Stelle im Code, die

- selbst wenig interne Logik hat
- viele andere Module/Klassen benötigt
- z.B. den Ablauf einer User Story beschreibt
- oft in "Service" Klasse (z.B. RegistrationService)

Einfaches Beispiel

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

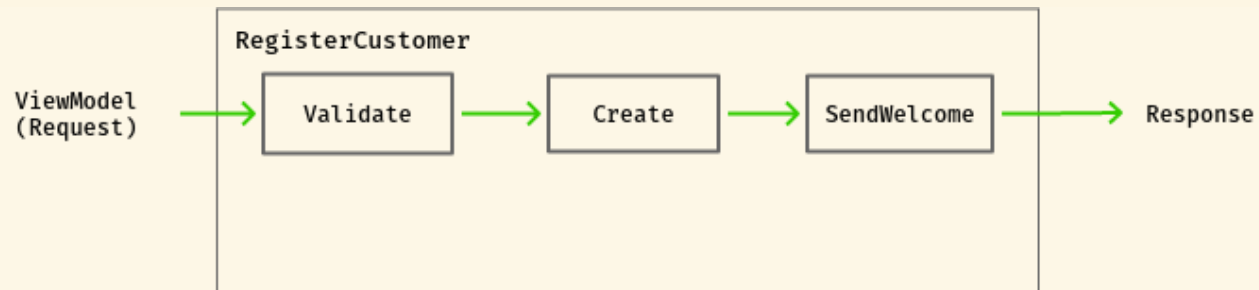
Realistischeres Beispiel

```
var result = Validate(createCustomerViewModel)
    .OnSuccess(c => _customerRepository.Create(c))
    .OnSuccess(c => createCustomerViewModel.WantsPremiumSupport
        ? _creditCardGateway.Charge(c.CreditCardNumber)
            .OnSuccess(gateway => gateway.ChargeWasBooked
                ? _customerRepository.UpgradeToPremium(c)
                : _creditCardGateway.RollbackLastTransaction(c))
            : Result.Ok(c))
    .OnSuccess(c => _mailConfirmer.SendWelcome(c))
    .OnBoth(cResultAtEnd => cResultAtEnd.IsSuccess
        ? new CustomerCreatedViewModel(cResultAtEnd.Value.Id)
        : CreateErrorResponse(cResultAtEnd.Error));
```

USER STORY: ANMELDUNG ALS NEUER BENUTZER

Wenn ein neuer Benutzer sich anmeldet,

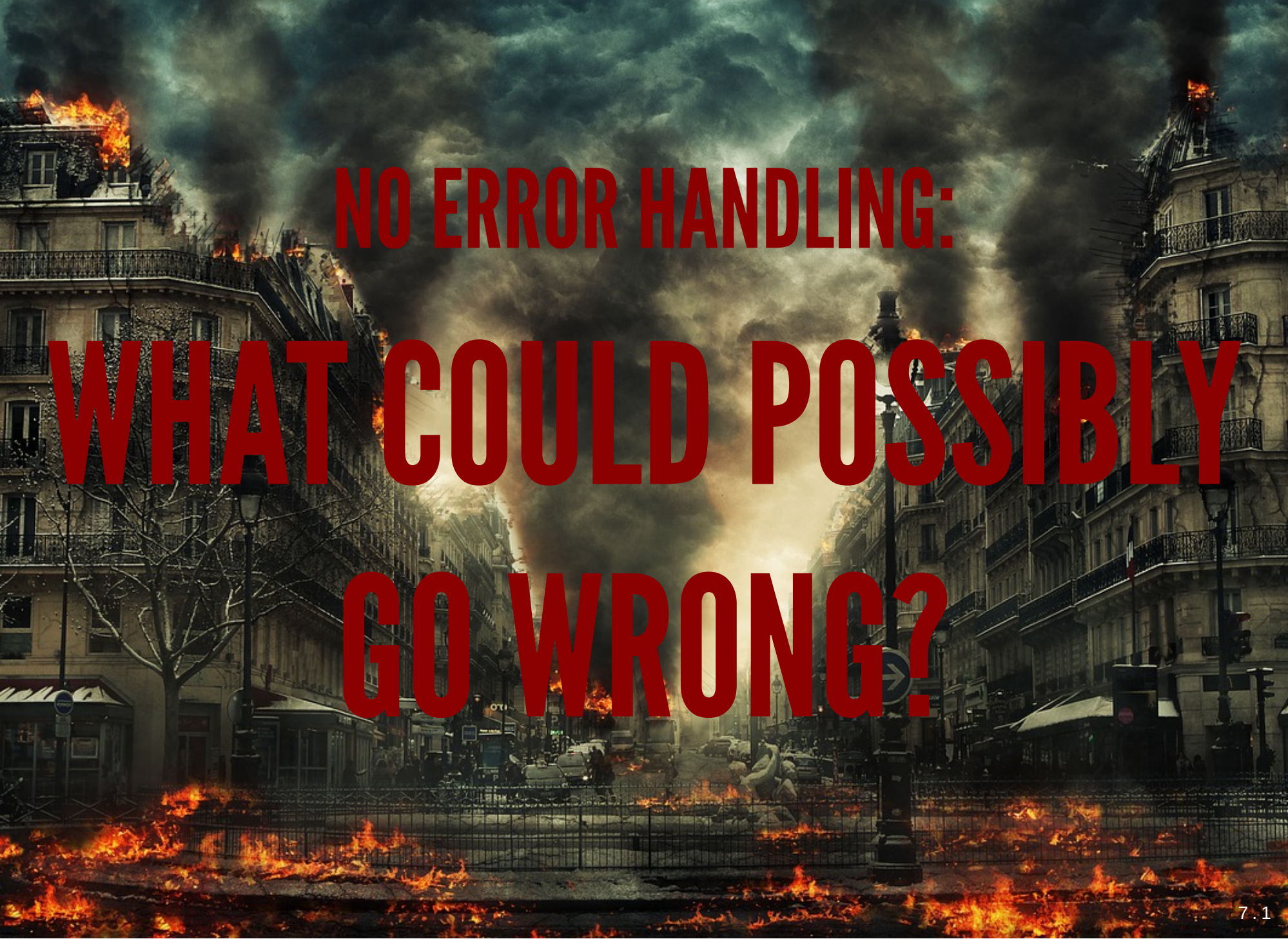
- werden seine Eingaben validiert
- wird er im System gespeichert
- erhält er eine Bestätigungsmail



```
public CustomerCreatedViewModel RegisterCustomer(SomeVM viewModel)
{
    var customer = 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?**

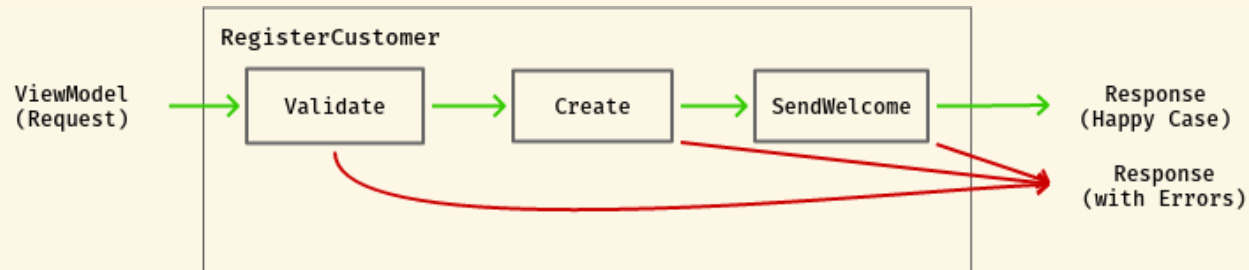
...potentielle Fallstricke...

```
// can fail  
var customer = Validate(createCustomerViewModel);  
  
// can fail  
customer = _customerRepository.Create(customer);  
  
// can fail  
_mailConfirmer.SendWelcome(customer);  
  
return new CustomerCreatedViewModel(customer.Id) {Success = ??};
```

JUST MY 2 CENTS

Nicht einfach drauflos programmieren: Zuerst mit
Kunde/Domain-Experten klären!

Dann die User Story aktualisieren (oder neue User
Story erstellen)

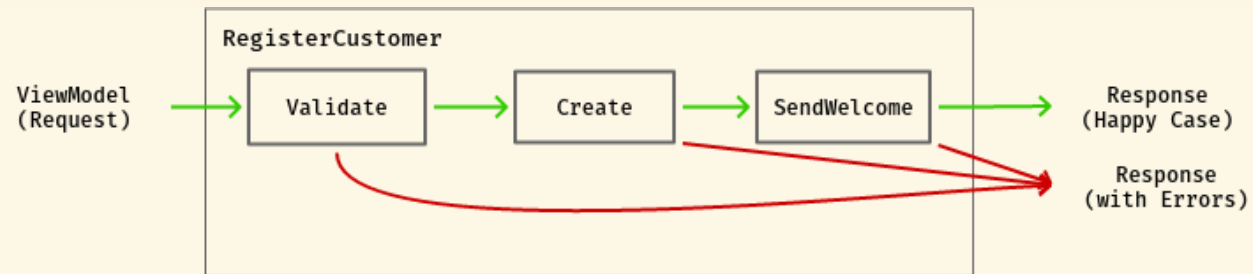


```
Customer customer;
try { customer = Validate(createCustomerViewModel); }
catch (Exception e) { return CreateErrorResponse(e); }

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

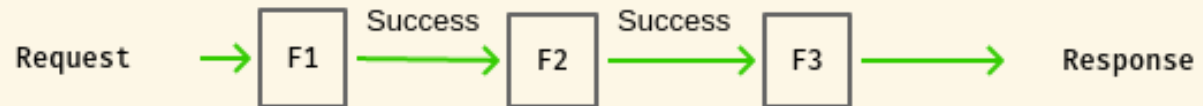
try { _mailConfirmer.SendWelcome(customer); }
catch (Exception e)
{
    // don't fail, but maybe: logging, retry-policy
}

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

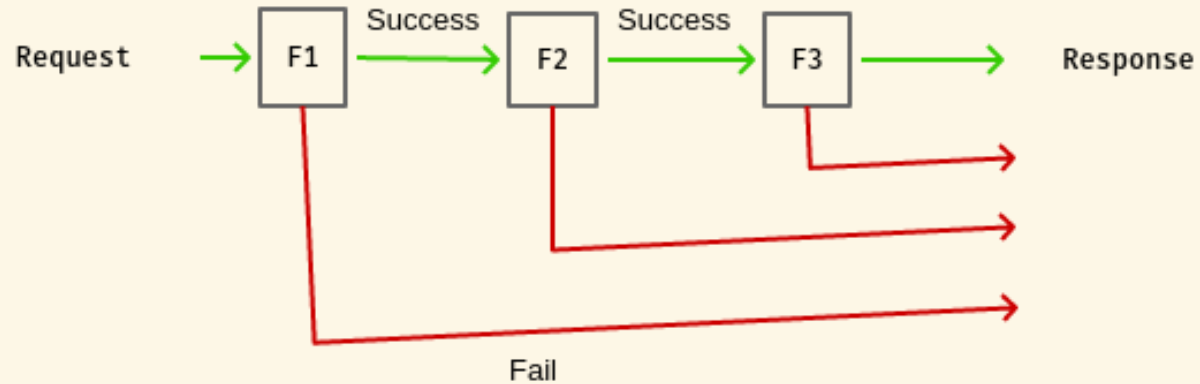


Fehlerbehandlung macht einen Großteil des Codes aus

...happy case...




...mit Fehlerbehandlung...



FUNKTIONALE PROGRAMMIERUNG

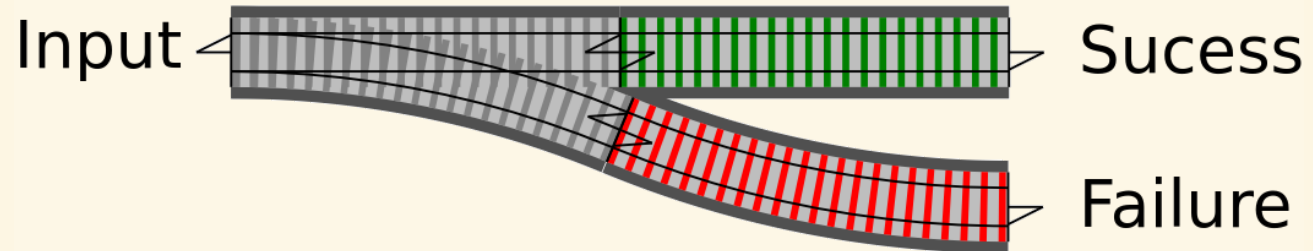
Wer möchte heute noch auf Lambdas verzichten?

- Was unterscheidet Funktionen von Methoden?
 - entspricht eher einer math. Funktion (und nicht einer Eigenschaft einer Klasse)
 - gleiche Eingabe gibt immer gleiches Ergebnis zurück
 - (idealerweise ohne Seiteneffekte)
- Funktionen können als Eingabe- und Rückgabewert verwendet werden ("higher order functions")

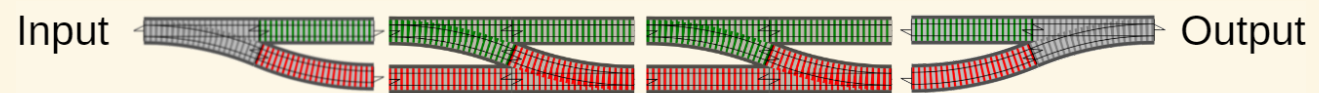
A still from the TV show 'The Big Bang Theory' featuring Sheldon Cooper. He is sitting at his desk in his apartment, looking off-camera with a frustrated expression. He is wearing a black t-shirt with a blue and yellow graphic of a car. The background shows his desk with a laptop, papers, and a shelf with various items.

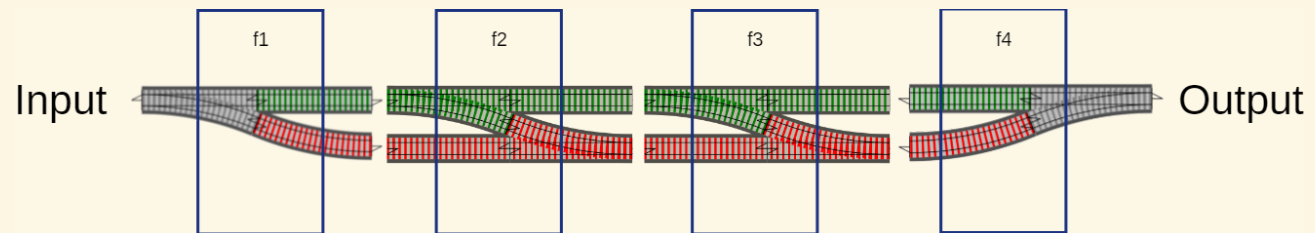
**WHERE ARE THE TRAINS?
YOU PROMISED RAILWAYS!**

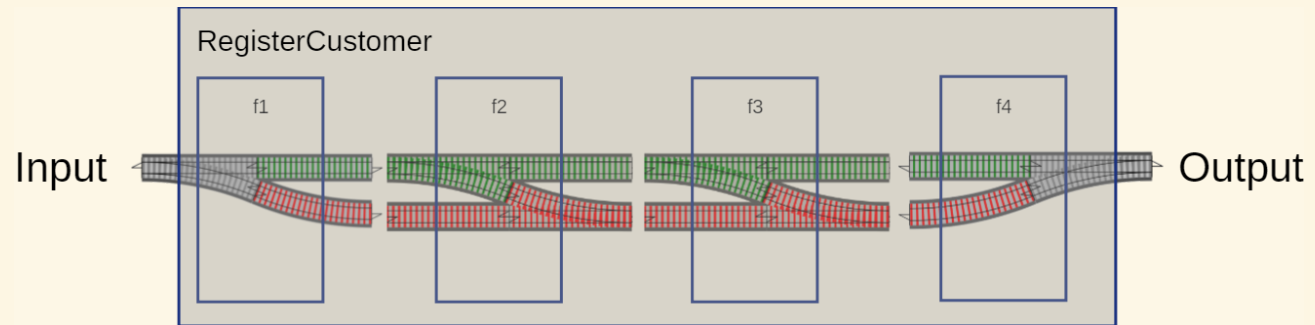
"Weiche"



(...endlich die Railway Metapher...)

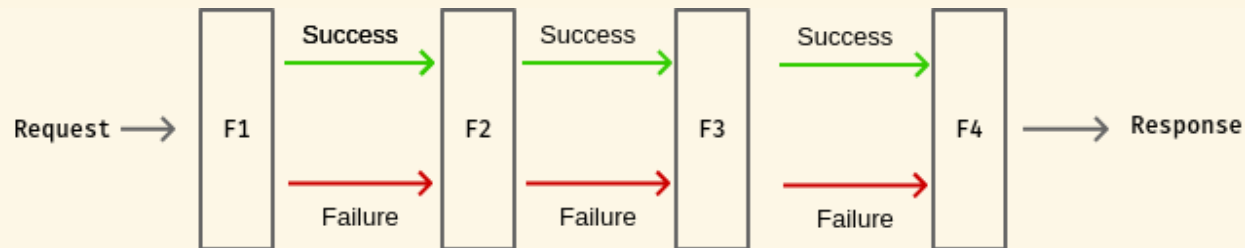






- Wir brauchen ein Objekt, dass "beide Schienen im Bauch hat" (Success/Failure)
- Dieses Objekt nennen wir **Result**

Machen wir unser Bildchen "funktionaler"



- F1: wie gehabt
- F2 und F3: können Fehler empfangen (cool)
- F2 und F3: austauschbar (sehr cool)

F1: wirklich wie gehabt? (1: Eingang, 2 Ausgänge)

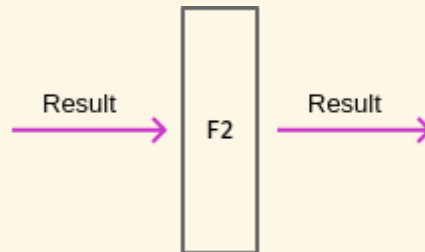
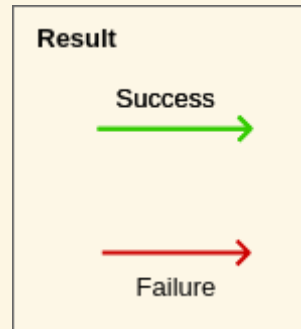
Nein!

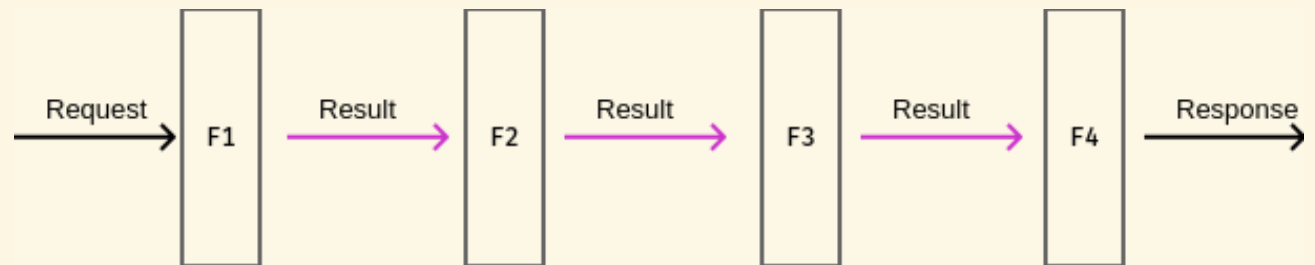
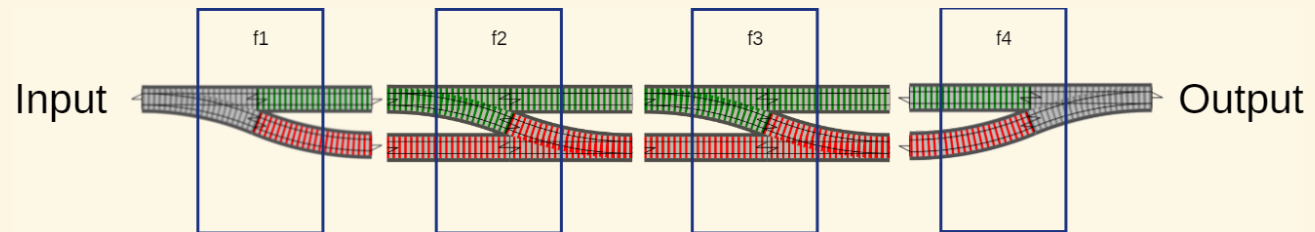
Eigentlich wollen wir das Ergebnis von F1 kapseln,
denn F2 erwartet



F2: muss auch Fehler empfangen können (2 Eingänge)

- wir müssen ein "Result" als Eingabe verarbeiten können
- **F1** muss als Ausgabe ein "Result" liefern





Was hat es mit dieser "Result" Klasse auf sich?

In C# und in Java gibt es aktuell keine "Result" Klasse



- C#: CSharpFunctionalExtensions (NuGet)
- Java: auch möglich (Link im Abspann)



C# API

NuGet: CSharpFunctionalExtensions

Result<T> als Rückgabewert

```
public Result<Customer> CreateCustomer(...)
{
    var customer = ...;
    if (IsValid(customer))
    {
        return Result.Ok(customer);
    }
    else
    {
        return Result.Fail<Customer>("Validation failed");
    }
}
```


Ist nix Neues: man "wrapped" das Ergebnis
Was neu ist: man kann die Ergebnisse verketten!

Finden wir zusammen raus, wie "Result" funktioniert

Mob-Session

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/>
- CSharpFunctionalExtensions
<https://github.com/vkhorikov/CSharpFunctionalExtensions>

FRAGEN?

Kontaktinfos:

-  @drechsler
-  socialcoding@pdrechsler.de