

# Design patterns in functional programming

It's not about monads.

# Me me me

- Currently founding a Behavioural Analytics firm: Logary Analytics
- It will model user segments and user behaviours with causative analytics;
  - Highlighting what you can do more of to improve your key metrics
  - Highlighting what you should do less of to improve your key metrics



Expecto

An advanced testing library for F#

build passing build passing nuget v9.0.0 368.0k f# 99.0% License Apache 2.0 Sponsor

Unit-, fuzz-stress and generative, parallel, async testing



Logging, metrics, tracing – used at Tradera in production in hot paths



FuncProgSTHLM founder

# tradera

On-prem to Google Cloud migration

# Qvitoo\_

Founder, CEO, architect, etc...

The image shows a screenshot of the AFA Forskning website with a banner about funding research for the workplace. Below it are two Twitter posts from the AFA Forskning account. The first post is from Karin Olson (@karinolson) about the product development team, and the second is from Henrik Feldt (@henrikfeldt) about a hackathon. Both posts include screenshots of the event.

AFA Forskning  
Om oss Sök finansiering Forskningsarkivet Aktuellt

Vi finansierar forskning som kommer till nytta i arbetslivet

Karin Olson - 1st Product management, AFA Försäkring  
Vi har runt 10 produktutvecklingsteam på AFA Försäkring. De bemannas till stor del av kompetenser från IT, men även produktlägare och verksamhetsutvecklare från andra delar av verksamheten.

För att uppmuntra till lärande mellan teamen var det däg premiär för vår egna Tech Meetup, öppen för alla som var intresserade. Eft för oss nytt sätt att sprida kunskap och inspireras av varandra; ett uppskattat initiativ från vår IT-erhet!

För ut var @henrikfeldt som delade med sig av de arkitektur- och teknikal de gjort i hans produktteam.

AFA Försäkring 6,427 followers 2mo • Edited

Bygg och lansera en hemsida utan att utmaningen för vårt FoU-team som pre har varit igång i ett år men den senaste skulle falla på plats, från design och inr lösningar.

- Det har varit en otroligt spänande ut samtidigt som vi jobbat på distans. Vi h timmer om dagen, för att prata lösningar, vana att sitta tätt ihop på jobbet och af direkt. Men det har g tt otroligt bra och hur vi i teamet arbetat tillsammans och forskningskommunikatör och Content!

AFA Försäkring satsar 150 Mkr per år ti arbetsmiljö och hlska som ska komma t sajen st rker FoU-teamet kommunikat s kta finansiering för sin forskning. <https://lnkd.in/gmtrykr>

Eldsjälarna bakom hemsidan: Anna-Mia Olofsson, Alex Rästen, Alexander Kolbratt, Henrik Feldt, Rami Daghawi, Cecilia Carlsson, Robin van Wijk och Fredrik E.

Helena Jahncke - 1st Chef FoU och Rehabstöd på AFA Försäkring  
Vi är s nöjda med sidan, bra jobbat! 😊

Teamet bakom nya sajten AFA Försäkring FoU [afaforsakring.se](https://afaforsakring.se) • 1 min read

AFA Forskning;  
Tech lead/architect

F# Web server, co-author

Phillip Carter #BlackLivesMatter @\_cartermp Jun 11  
Shout out to fine folks like @alfonsogcnuez, @zaid\_ajaj, @granicz, and @Tarmil\_ for building awesome #fsharp stuff for the web. I was privileged to brag about your work to an eager NDC Oslo audience today!

2 11 50

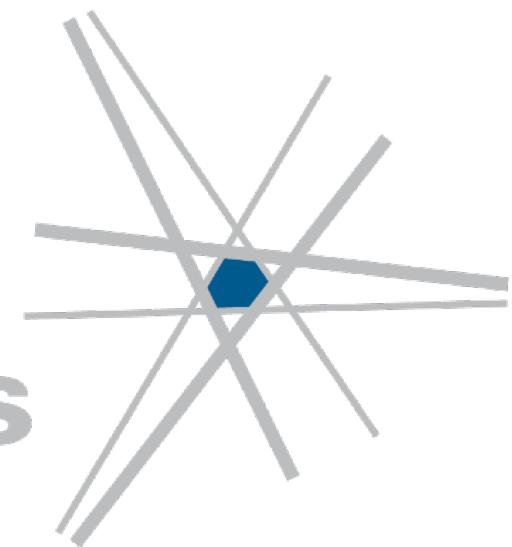
Phillip Carter #BlackLivesMatter @\_cartermp

Also @dustinmoris, @k\_cieslak, @ad3mar, @henrikfeldt, and @pim\_brouwers for their work on #fsharp for the server. I'm surely missing more amazing folks, and I wish I had a big thank you slide at the end of the talk.

11:33 PM · Jun 11, 2020 · Twitter for iPhone

6 Retweets 30 Likes

# RaySearch Laboratories



Built an HPC cluster for cancer treatment

# voi.

Bootstrap engineering dept

# What's a design pattern?

A construction that repeats itself because it's useful and apt for the situation

— me



# Categories of patterns

- Parsers and validation ("Parse don't validate")
- State mutation, handling ("Moore", "Mealy" and the monoids)
- Control flow ("await", "async" and the monads)
- Higher order functions

# Some interesting patterns

- Delegator / hole-in-the-middle
- Simple state machines with mutually recursive functions
- Currying
- Dynamic configuration via closure
- Continuation passing style
- Scans and reductions / folds
- Cold observables / Alt/Job for retries

# Hole in the middle

When you want to reuse a  
**complex function  
implementation**



# Delegator / hole-in-the-middle

```
4
5 // currying and hole-in-the-middle
6 function colourise(configure: (config: RequestInit) => RequestInit):
7 // returning a configured function
8 (item: Item) => Promise<ColourisedItem> {
9
10 // async-await (aka monadic control flow)
11 return async (item: Item) => {
12     const res = await fetch('/api/colours', configure({
13         method: 'GET',
14     }))
15
16     const json = await res.json()
17     return {
18         ...item,
19         colour: json.colour as string
20     }
21 }
22 }
```

# Delegator / hole-in-the-middle usage

**localToPage is some state only the caller knows**

```
47 // currying usage
48 const localToPage = '0123456789ABCDEF'
49 const getColour = colourise(init => {
50   ...init,
51   headers: {
52     ...init.headers,
53     'x-vanity-header': localToPage
54   }})
55
```

Varying the usage of the complex implementation is easy now.

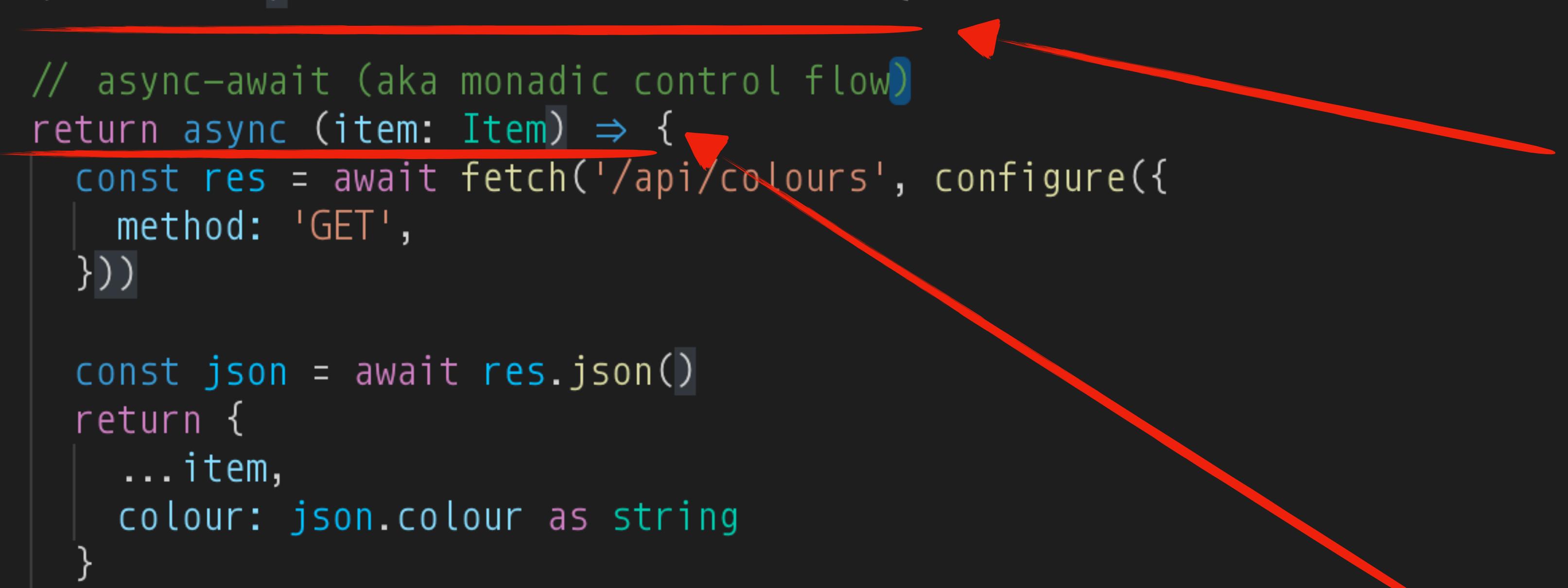
# Currying

When you want to reuse a  
complex **setup**



# Currying

```
4
5 // currying and hole-in-the-middle
6 function colourise(configure: (config: RequestInit) => RequestInit):
7     // returning a configured function
8     (item: Item) => Promise<ColourisedItem> {
9
10    // async-await (aka monadic control flow)
11    return async (item: Item) => {
12        const res = await fetch('/api/colours', configure({
13            method: 'GET',
14        }))
15
16        const json = await res.json()
17        return {
18            ...item,
19            colour: json.colour as string
20        }
21    }
22}
23}
```



Colourise is pre-configured (can pre-compute CPU-intensive stuff or fetch from the network before the secondary analytics)

# Curring usage

**getColour is a configured function**

```
47 // currying usage
48 const localToPage = '0123456789ABCDEF'
49 const getColour = colourise(init => ({
50   ...init,
51   headers: {
52     ...init.headers,
53     'x-vanity-header': localToPage
54   }})
55
```

So part of the cost of calling `colourise` is taken when you configure it, and then it's quick for subsequent callers.

# Dynamic reconfiguration

E.g. your set of processors/servers is changing continuously as your nodes go up and down



# Dynamic reconfiguration via composition

## Config

```
18     /// Null object pattern; will only return loggers that don't log.
19     T <span>Henrik Feldt</span>
20     let defaultConfig =
21         { getLogger = fun _ -> NullLogger.instance
22          getTimestamp = MonotonicClock.getTimestamp
23          consoleLock = Lock()
24          metrics = MetricRegistry() }
25
26     /// This is the "Global Variable" containing the last configured Logary
27     /// instance. If you configure more than one logary instance this will be
28     /// replaced.
29     DVar<T> <span>Henrik Feldt</span>
30     let internal configD = DVar.create defaultConfig
31
32     DVar<IClock> <span>Henrik Feldt</span>
33     let internal clockD =
34         <span>Henrik Feldt</span>
35         let createClock config = { new IClock with member x.GetCurrentInstant() = Instant.ofEpoch (config.getTimestamp()) }
36         configD |> DVar.map createClock
```

# Dynamic reconfiguration via composition

## Impl

```
1  namespace Logary.Internals
2
3  open System
4  open System.Threading
5
6  type DVar<'a> = private { mutable cell : 'a; event : Event<'a> }
7
8  [<CompilationRepresentation\(CompilationRepresentationFlags.ModuleSuffix\)>]
9  module DVar =
10
11    'a -> DVar<'a>
12    let create (a:'a) : DVar<'a> =
13      { cell = a ; event = Event<'a>() }
14
15    DVar<'a> -> 'a
16    let get (d:DVar<'a>) : 'a =
17      d.cell
18
19    'a -> DVar<'a> -> unit
20    let put (a:'a) (d:DVar<'a>) : unit =
21      Interlocked.Exchange (&d.cell, a) |> ignore
22      d.event.Trigger a
23
24    DVar<'a> -> 'a -> unit
25    let inline set d a =
26      put a d
```

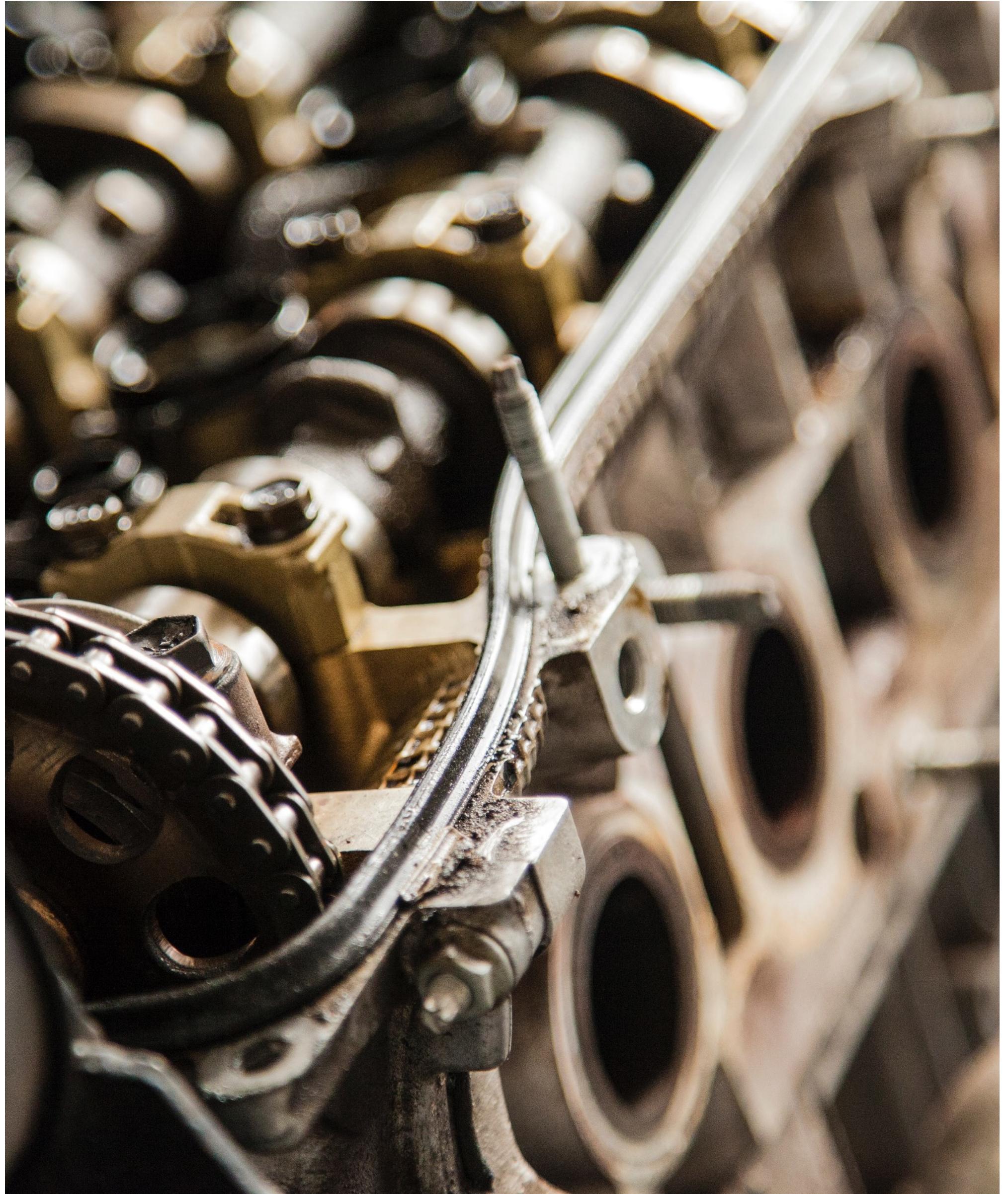
# Dynamic reconfiguration via composition

## Usage

```
type Flyweight(name: PointName) =  
    DVar<Logger> & Henrik Feldt  
    let loggerD = configD |> DVar.map (fun cfg -> cfg.getLogger name)  
    interface Logger with // flyweight  
        PointName & Henrik Feldt  
        member x.name = name  
        LogLevel & Henrik Feldt  
        member x.level =  
            let logger = DVar.get loggerD in logger.level  
            bool * LogaryMessage -> LogResult & Henrik Feldt  
        member x.logWithAck(putBufferTimeOut, message) =  
            let logger = DVar.get loggerD  
            logger.logWithAck(putBufferTimeOut, message)
```

# Mutually recursive functions as a state machine

When you want to separate  
runtime states from each other



# State machine w/ functions

```
FileConf -> Will<TargetMessage [] * uint16> -> TargetAPI -> Job<unit>Henrik Feldt
let loop (conf: FileConf) (will: Will<TargetMessage[] * uint16>) (api: TargetAPI) =
    let ri, rotateCh = api.runtime, Ch ()

    let shutdownState state = ...

    // In this state the File target opens its file stream and checks if it
    // progress to the recovering state.
    let rec init () = ...

    // In this state we try to write as many log messages as possible.
    and running (state: State): Job<unit> = ...

    // In this state we verify the state of the file.
    and checking (state: State) = ...

    // In this state we do a single rotation.
    and rotating (state: State) = ...

    and recovering (state: State) (lastBatch: TargetMessage[]) = function ...
        init ()
```

# State machine w/ functions

```
55 // In this state we try to write as many log messages as possible.
56 and running (state: State): Job<unit> =
57     Alt.choose [
58         api.shutdownCh ^=> fun ack ->
59             shutdownState state >>=
60             ack *<= () 
61
62         // TODO: handle scheduled rotation
63         rotateCh ^=> fun () -> rotating state
64
65         RingBuffer.takeBatch (conf.batchSize) api.requests ^=> fun reqs ->
66             writeRequestsSafe ri.logger conf state reqs >>=
67                 function
68                     | Choice1Of2 () ->
69                         checking state
70                     | Choice2Of2 err ->
71                         ri.logger.errorAck("IO Exception while writing to file. Batch size is {batchSize}.", fun m ->
72                             m.setField("batchSize", conf.batchSize)
73                             m.addExn err)
74                         >>=. Will.update will (reqs, 1us)
75                         >>=. Job.raises err
76
77     ] :> Job<_>
```

# Finding more patterns

- Functional Pearls – [haskell.org](http://haskell.org)
- Read my code-bases; [Expecto](#), [LogaryJS](#), [Logary](#), [HttpFs](#) – they all use these patterns
- Read [Parse don't validate](#) by Alexis King
- More interesting keywords:
  - Process algebra, pi-calculus, Kahn process networks, Mealy machines, Stateful functions (Flink), Stream processing, the Elm community

# Questions?



for listening