



Delegate:
Showcase 2/3 years old F# Codebase
@ Prosa 2016-02-23

F#unctional Copenhageners Meetup Group
(MF#K)





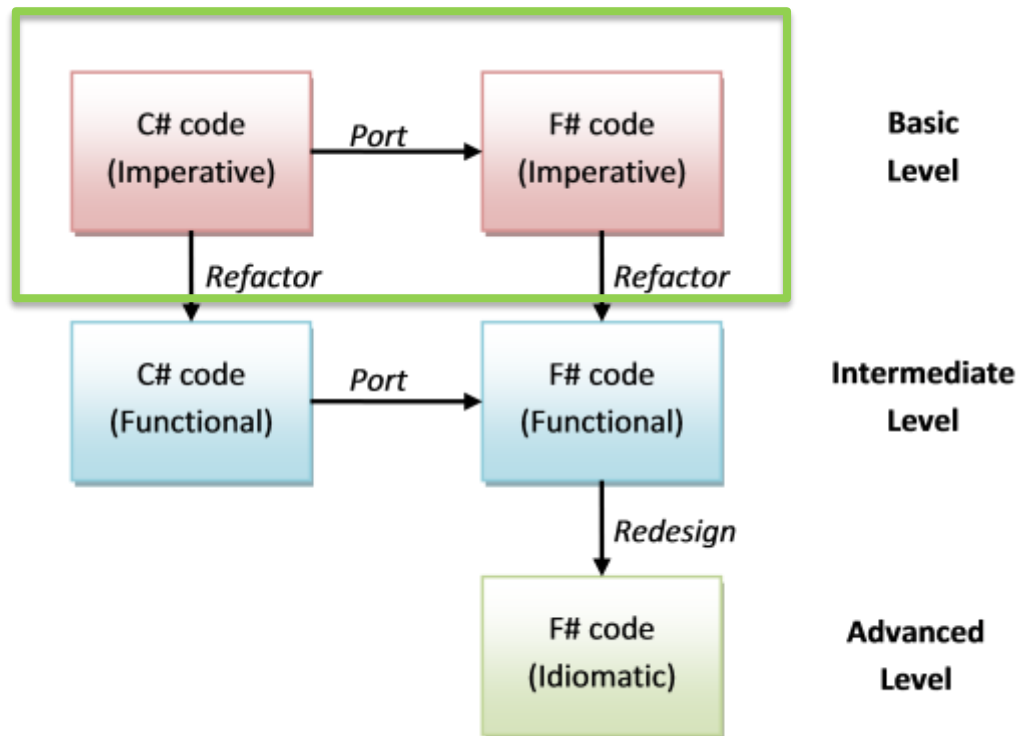
- About me
- F#unctional Copenhageners Meetup Group (MF#K)
- How did we start ...
- ... where are we now ...
- ... where are we going to
- Showcase Codebase
- Missing syntactic sugar in F#
- Q & A



- Ramón Soto Mathiesen
- MSc. Computer Science DIKU and minors in Mathematics HCØ
- Managing Specialist / CTO of CRM Department @ Delegate A/S
 - ER-modeling, WSDL, OData (REST API)
- F# / C# / JavaScript / C++
- Blog: <http://blog.stermon.com/>

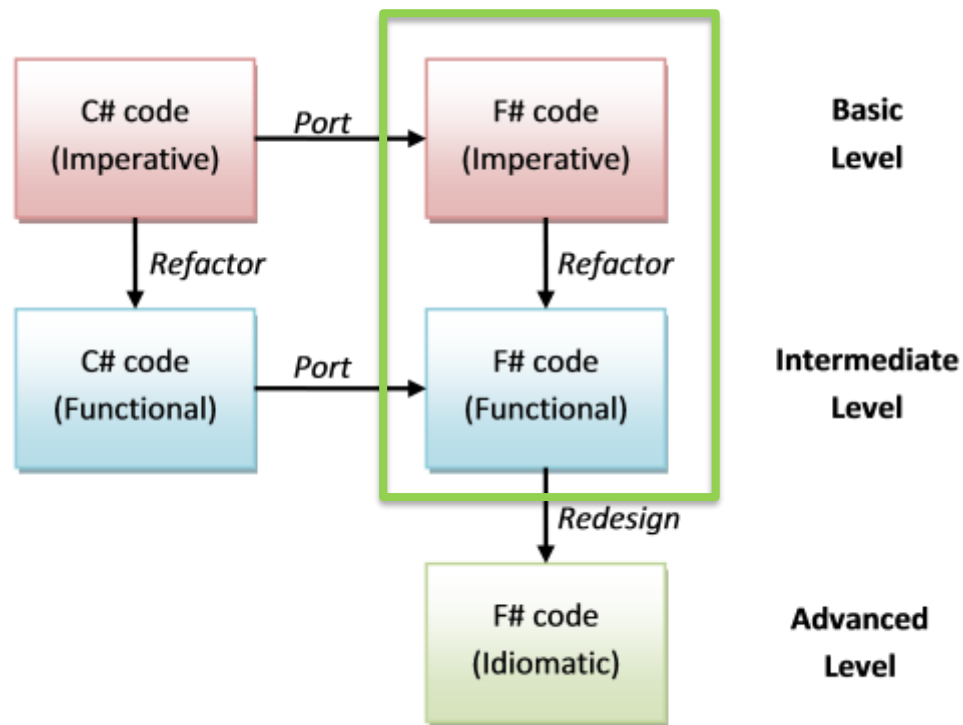


- F#unctional Copenhageners Meetup Group will try to get more and more software projects to be based on functional programming languages.
- We mainly focus on F# and Haskell, but other functional programming languages like Scala, Lisp, Erlang, Clojure, etc. are more than welcome.
- We expect to meet at least twelve times a year (last Tuesday every month), if not more, to share experiences with regards of the use of functional programming languages in software projects that are in / or heading to production.



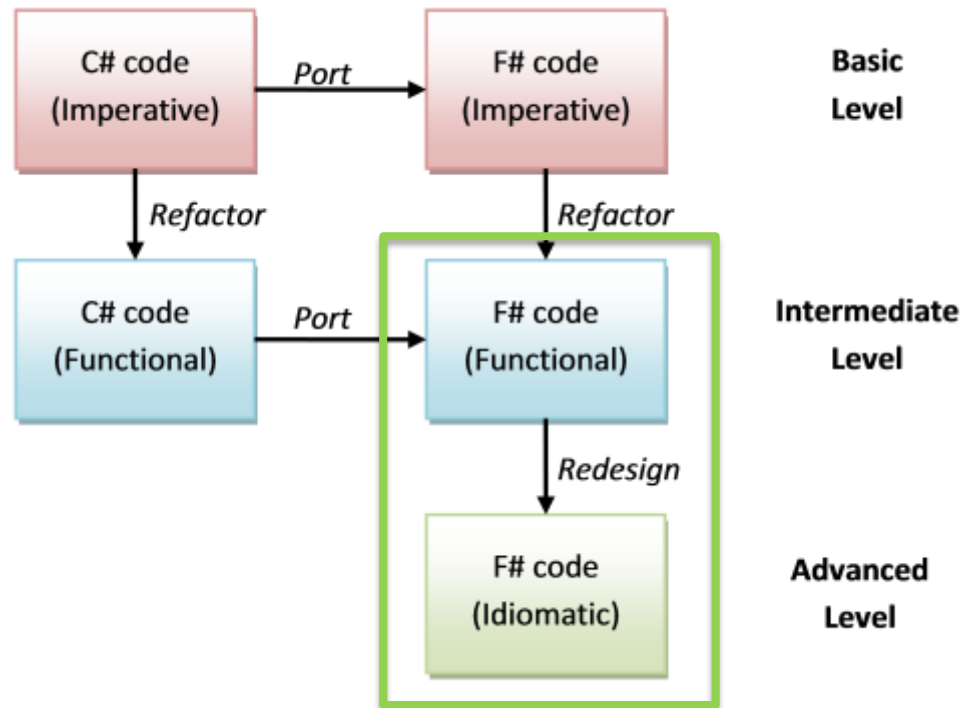
- Porting from C# to F#:

- <http://fsharpforfunandprofit.com/posts/porting-to-csharp-intro/>



- Porting from C# to F#:

- <http://fsharpforfunandprofit.com/posts/porting-to-csharp-intro/>



- Porting from C# to F#:

- <http://fsharpforfunandprofit.com/posts/porting-to-csharp-intro/>



- Daxif: <https://github.com/delegateas/Delegate.Daxif>



A look at Microsoft Orleans through Erlang-tinted glasses

Some time ago, Microsoft announced Orleans, an implementation of the actor model in .Net which is designed for the cloud environment where instances are ephemeral.

We've been using Orleans for a while now, and it's been a great experience. It's a win for the team, more people being able to look at our codebase, for instance.

As such I have been taking an interest in Orleans to see if it represents a good fit, and whether or not it fits the lofty promises around scalability, performance and reliability. Below is an account of my personal views having downloaded the SDK and looked through the samples and followed through Richard Arthur's Pinesight course.

How we read English

For starters, it's not partition tolerant towards partitions to the data store used for its SaaS management. Should a node be partitioned or suffer an outage, it'll result in a full outage of your service. These are not traits of a masterless system that is desirable when you have strict uptime requirements.

When everything is working, Orleans guarantees that there is only one instance of a virtual actor running in when a node is lost the cluster's knowledge of nodes will diverge and during this time the single-activation model is eventually consistent. However, you can provide stronger guarantees yourself (see SaaS Management section below).

Orleans uses at least once message delivery, which means it's possible for the same message to be sent to a receiving node is under load or simply fails to acknowledge the first message in a timely fashion. This again, is something that you can mitigate yourself (see Message Delivery Guarantees section below).

Finally, its task scheduling mechanism appears to be identical to that of a naive event loop and exhibits all the fallacies of an

1. left-to-right

How we read English

2. top-to-bottom

1. left-to-right

```
let doSomething x y =  
  monkey()  
  |> zoo  
  |> bar x  
  |> foo y
```

2. top-to-bottom

- How to read F# code?
 - Left-right and top-down (also for projects) but start with interfaces 😊



```
[<AutoOpen>]
module Either =
    type ('a,'b) Either = Choice<'a,'b>
    let Left x : Either<'a,'b> = Choice10f2 x
    let Right x : Either<'a,'b> = Choice20f2 x

    let (|Left|Right|) = function Choice10f2 x -> Left x | Choice20f2 x -> Right x
    let bind f = function | Right a -> f a | Left b -> Left b
    let inline (>>=) m f = bind f m

    let left = function | Left s -> Some s | Right _ -> None
    let right = function | Left _ -> None | Right f -> Some f

[<AutoOpen>]
module Result =
    let success x = Either.Right x
    let failure x = Either.Left x
    let succeeded x = Either.right x
    let failed x = Either.left x
```

- Based on Control.Monad.Either by re-using Choice Type:
 - <https://hackage.haskell.org/package/category-extras-0.52.0/docs/Control-Monad-Either.html>



- This code:

```
let foobar x y =  
    success x / y  
    failure ex -> ex
```

- Should be syntactic sugar for this:

```
let foobar x y =  
    try x / y |> success  
    with ex -> ex |> failure
```

- Based on Control.Monad.Either by re-using Choice Type:
 - <https://hackage.haskell.org/package/category-extras-0.52.0/docs/Control-Monad-Either.html>



```
List.init 10 (foobar)
|> List.map(fun dividedBy -> dividedBy 1)
|> List.choose(succeeded)

3 |> foobar 22 >>= foobar 33 >>= foobar 44 |> succeeded
```

```
val foobar : x:int -> y:int -> (exn,int) Either
>
val it : int list = [0; 1; 2; 3; 4; 5; 6; 7; 8; 9]
>
val it : int option = Some 11
```

- Based on Control.Monad.Either by re-using Choice Type:
 - <https://hackage.haskell.org/package/category-extras-0.52.0/docs/Control-Monad-Either.html>



Questions?



- Code is available @:
 - <https://github.com/delegateas/Delegate.Daxif>
- Slides will be available @ [MF#K \(Files\)](#)
- Sign up @ [MF#K](#) for:
 - More *fun*
 - Hands-on:
 - None so far ...
 - Talks:
 - In the pipeline talks about:
 - Upcoming: Nothing yet, you want to give a talk?
- MF#K would like to thank our sponsor(s):

Forbundet af It-professionelle

PROSA