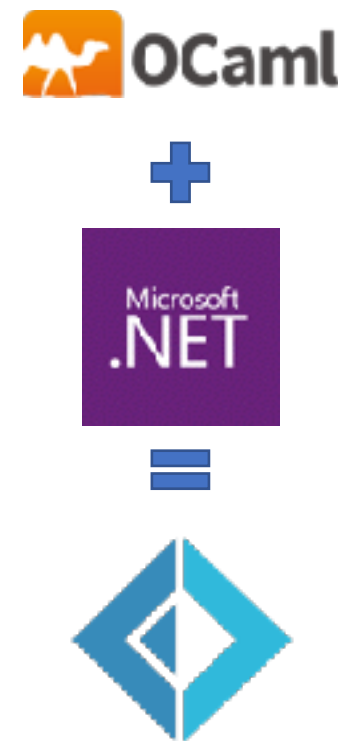
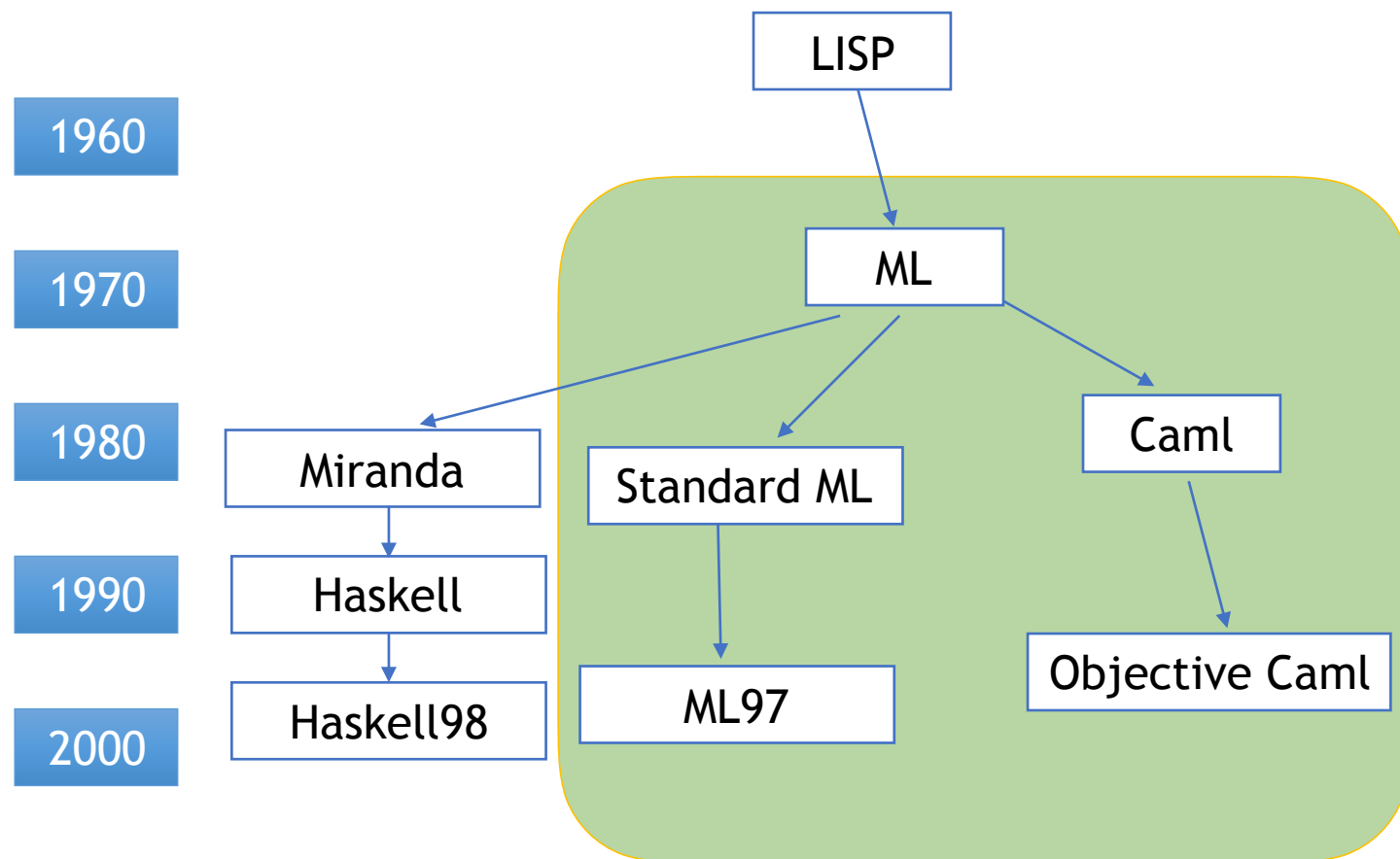


F#

Олег Заимкин, ДатаВоркс, Новосибирск

Что такое F#?



О языке

- FP-first
- Типизация
- АТД + ПП
- Pattern matching
- Null безопасность
- Вывод типов

```
/// Implementation of "dry run"
let dryRun ctx options (groups: string list list) =
    let getDeps = getChangeReasons ctx ▷ memoizeRec

    // getPlainDeps getDeps (getExecTime ctx)
    do ctx.Logger.Log Command "Running (dry) targets %A" groups
    let doneTargets = System.Collections.Hashtable()

    let print f = ctx.Logger.Log Info f
    let indent i = String.replicate i " "

    let rec showDepStatus ii reasons =
        reasons ▷ function
        | ChangeReason.Other reason →
            print "%sReason: %s" (indent ii) reason
        | ChangeReason.Depends t →
            print "%sDepends '%s' - changed target" (indent ii) t.ShortName
        | ChangeReason.DependsMissingTarget t →
            print "%sDepends on '%s' - missing target" (indent ii) t.ShortName
        | ChangeReason.FilesChanged (file :: rest) →
            print "%sFile is changed '%s' %s" (indent ii) file (if List.isEmpty
List.length rest)
            reasons →
                do print "%sSome reason %A" (indent ii) reasons
        ()

    let rec displayNestedDeps ii =
        function
        | ChangeReason.DependsMissingTarget t
        | ChangeReason.Depends t →
            showTargetStatus ii t
        | _ → ()
```



Алгебраические типы данных (ADT)

```
// simple types
type Nothing = unit
type ChannelId = int
type Message = string

type MyFun = int → string

// Product types
type Timestamp = int * DateTime
let ts = (0, DateTime.Now)
```

Алгебраические типы данных (ADT)

```
type UserInfo = {ident: int; name: string}
```

```
type Channel = {  
  Messages: Message list  
  Info: string  
  Users: UserInfo list  
  PostText: string  
}
```

```
// Экземпляр типа
```

```
let chan = {Messages = []; Info = "None"; Users = []; PostText = ""}
```

ADT: ТИПЫ-СУММЫ

```
type Shape =  
  | Circle of float  
  | Square of float  
  | Rectangle of float * float
```

```
let shape = Square 42.0
```

FP: Pattern matching

```
type Shape =  
  | Circle of float  
  | Square of float  
  | Rectangle of float * float  
  
let shape = Square 42.0  
  
let area shape =  
  match shape with  
  | Circle r → Math.PI * r * r  
  | Square x → x * x  
  | Rectangle (w,h) → w * h
```


Рекурсия и ПП

```
type List<'T> =  
  | Nil  
  | Cons of 'T * List<'T>  
  
let list = Cons(1, Cons(2, Cons (3, Nil)))  
  
let foo = 1 :: 2 :: 3 :: List.Empty  
let listToo = [1; 2; 3]  
  
let rec pairProduct ls =  
  match ls with  
  | Cons (a, Cons (b, tail)) → Cons(a * b, pairProduct tail)  
  | other → other
```

FP: Active patterns

```
let (|IsChannelId|_|) s =  
    match Int32.TryParse s with  
    | true, value → Some value  
    | _ → None
```

```
let chan = function  
    | IsChannelId chanId → printf "%i" chanId  
    | any → printf "not a channel identifier"
```

FR: иммутабельность

```
let update (msg: ChannelMessage) (channel: Channel) : Channel =  
  
  match msg with  
  | Init (info, userlist) →  
    { Info = info; Messages = []; PostText = ""; Users = userlist}  
  
  | Update info →  
    { channel with Info = info }  
  
  | AppendMessage message →  
    { channel with Messages = state.Messages @ [message] }
```

FP: чистые функции

Чистая функция — это функция, которая при одинаковых аргументах всегда возвращает одни и те же значения и не имеет видимых побочных эффектов.

- Формализм
- Декларативность
- Кеширование
- Параллельное выполнение

View function

```
module Channel.View

let view dispatch (model: ChannelData) =
  [ chatInfo dispatch model
    messageList model.Messages
    div
      [ ClassName "fs-message-input" ]
      [ input
          [ Type «text»; Placeholder "Type the message here ..."
            valueOrDefault model.PostText ]
        button
          [ ClassName "btn" ]
          [ i [ ClassName "mdi mdi-send mdi-24px"
             OnClick (fun _ → dispatch PostText) ] [] ]
      ]
    ]
```

FR: статическая типизация и вывод типов

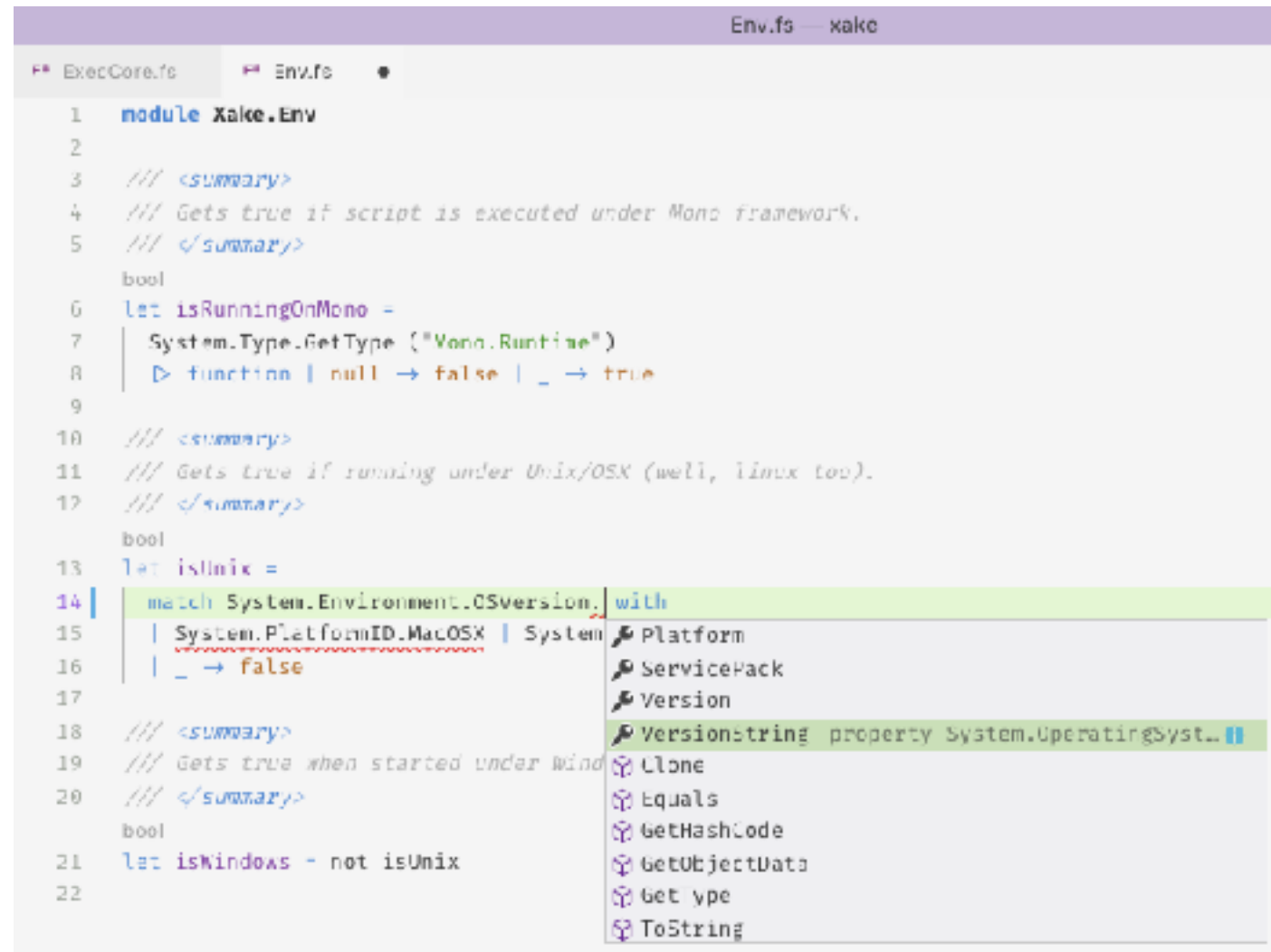
```
let f = (1 = 2)      // bool
```

```
let add a b = a + b  // int → int → int
```

```
let addFloat (a: float) b : float = a + (b : float)  
// float → float → float
```



F# Compiler Services



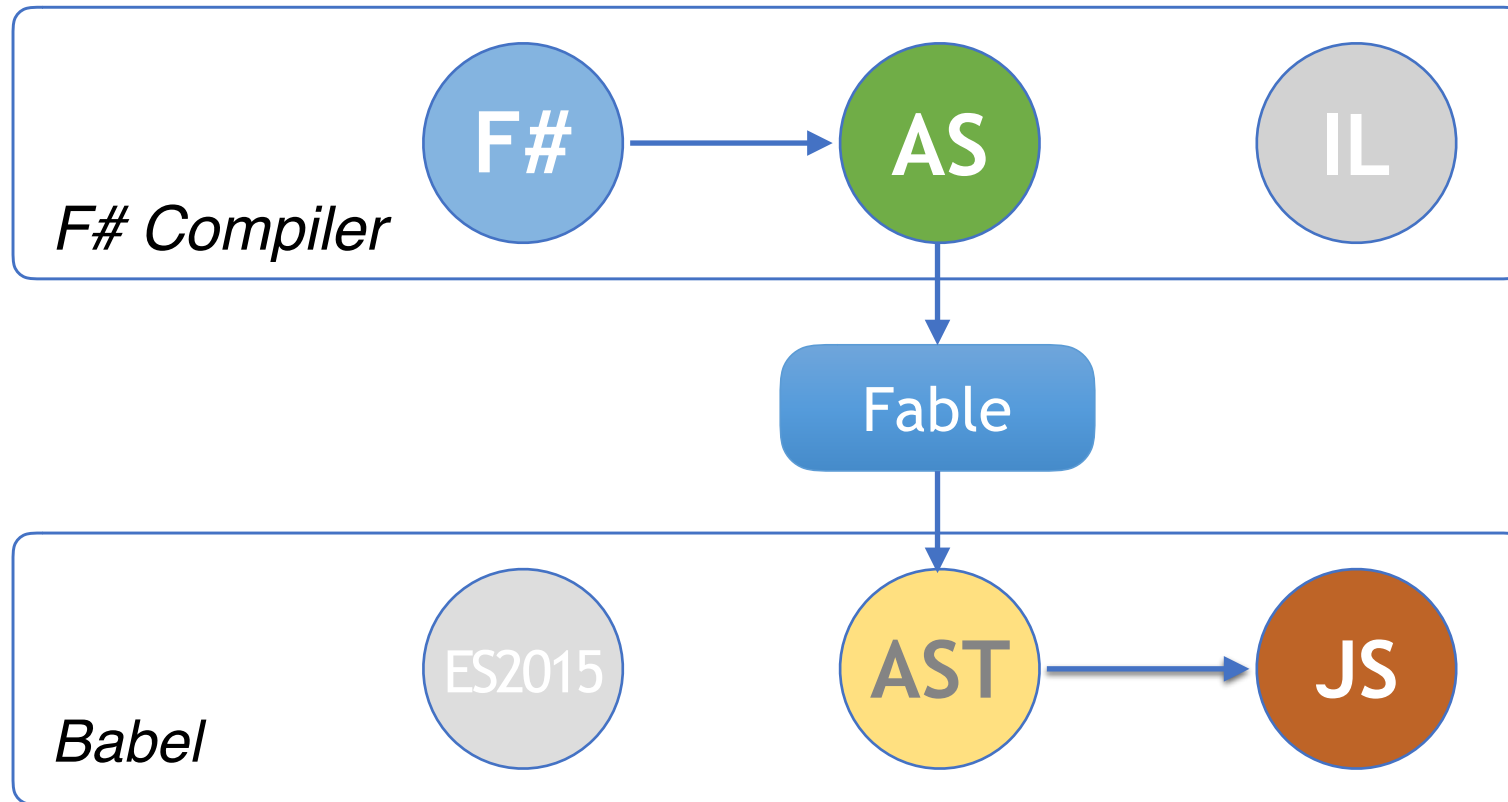
The screenshot shows an F# IDE window titled 'Env.fs — xake'. The code defines a module 'Xake.Env' with two boolean properties: 'isRunningOnMono' and 'isUnix'. The 'isUnix' property is defined using a match expression on 'System.Environment.OSVersion.Platform'. A dropdown menu is open for the 'with' keyword, showing various options like 'Platform', 'ServicePack', 'Version', 'VersionString', 'Clone', 'Equals', 'GetHashCode', 'GetObjectData', 'GetType', and 'ToString'. The 'VersionString' option is highlighted.

```
1 module Xake.Env
2
3 /// <summary>
4 /// Gets true if script is executed under Mono framework.
5 /// </summary>
6 bool
7 let isRunningOnMono =
8     System.Type.GetType("Mono.Runtime")
9     > function | null -> false | _ -> true
10
11 /// <summary>
12 /// Gets true if running under Unix/OSX (well, linux too).
13 /// </summary>
14 bool
15 let isUnix =
16     match System.Environment.OSVersion.Platform with
17     | System.PlatformID.MacOSX | System.PlatformID.Unix -> true
18     | _ -> false
19
20 /// <summary>
21 /// Gets true when started under Windows.
22 /// </summary>
23 bool
24 let isWindows = not isUnix
```

with

- Platform
- ServicePack
- Version
- VersionString property System.OperatingSystem...
- Clone
- Equals
- GetHashCode
- GetObjectData
- GetType
- ToString

Fable



JS interop

```
type IJQuery = interface end
module JQuery =
  [<Emit(«window['$']($0)»)>]
  let select (selector: string) : IJQuery = jsNative
  ...

JQuery.ready (fun () →
  JQuery.select "#main"
  ▷ JQuery.css "background-color" "red"
  ▷ JQuery.addClass "fancy-class"
  ▷ ignore
)
```

Elmish

```
type Model = int
type Msg = | Inc | Dec

let init() : Model = 0

let view model dispatch =
    div []
        [ button [ OnClick
                    div [] [ model.To
                    button [ OnClick

let update msg (model: Model) =
    match msg with
    | Inc → model + 1
    | Dec → model - 1
```

```
open Elmish.React
...

// App
Program.mkProgram init update view
▷ Program.withDebugger
▷ Program.withHMR
▷ Program.withReact "elmish-app"
▷ Program.run
```





EXPLORER

OPEN EDITORS

F# State.fs src

ELMISH

Counter

Home

Info

Navbar

F# App.fs

elmish.fsproj

F# Global.fs

paket.references

F# State.fs

F# Types.fs

.editorconfig

.gitattributes

.gitignore

elmish.sln

NuGet.Config

package-lock.json

package.json

paket.dependencies

paket.lock

README.md

RELEASE_NOTES..

JS webpack.config.js

yarn.lock

F# State.fs

```
Option<Page> -> Model * Cmd<Msg>
25 let init result =
26     let (counter, counterCmd) = Counter.State.init()
27     let (home, homeCmd) = Home.State.init()
28     let (model, cmd) =
29         urlUpdate result
30         { currentPage = Home
31           counter = counter
32           home = home }
33     model, Cmd.batch [ cmd
34                       Cmd.map CounterMsg counterCmd
35                       Cmd.map HomeMsg homeCmd ]
```

```
Msg -> Model -> Model * Cmd<Msg>
37 let update msg model =
38     match msg with
39     | CounterMsg msg ->
40         let (counter, counterCmd) = Counter.State.update msg
41         { model with counter = counter }, Cmd.map CounterMsg
42     | HomeMsg msg ->
43         let (home, homeCmd) = Home.State.update msg model.home
44         { model with home = home }, Cmd.map HomeMsg homeCmd
```



F# goodies

```
let fetchEntity url =  
    promise {  
        let! fetched = fetch url []  
        let! response = fetched.text()  
        return response  
    }
```

```
type Simple = JsonProvider<""> { "name"  
let simple = Simple.Parse("{" "name"  
simple.Age  
simple.Name
```

safe
mean

ИТОГИ

- хороший базис
- зрелый язык и компилятор
- короткий learning curve
- более простой код
- работает везде
- хороший инструментарий
- full-stack



ССЫЛКИ И КОНТАКТЫ

- <https://www.youtube.com/watch?v=LBekZt8QB4w>
- <http://fsharp-lang.ru>
- <http://fsharpforfunandprofit.com>
- <https://github.com/kunjee17/awesome-fable>
- <https://github.com/OlegZee>
- @olegzee в Telegram



Конец