Narratives in The Early History of F#

Don Syme, Researcher and Community Contributor, Microsoft

A meta-talk about a meta-language

Not a recap of the paper

From 1970s to 2015

From Milner, Newey, Morris to Microsoft and Open Source

History is always written
selectively

I’ll be explicit about the narratives
I’ve chosen

Narrative #1

A personal journey

Narrative #2

A community's journey
(and origin mythology)

Narrative #3

Things don’t just happen. Things happen to people, who act, react and change in very human ways

creation, rejection, denial, stubbornness, cooperation, subversion, opportunism, …

Narrative #4

What happens when computing traditions with strong belief systems collide, both in industry and academia?

Narrative #5

The dynamic interplay between academic research and industry concerns, between programming and programmability

Narrative #6

The golden thread – what holds true in programming language design, what stands the test of time?

The
World as
It Was

(~1997)

ACM SIGPLAN Functional Programming

Why no one uses functional languages[1]

Editor: *Philip Wadler, Bell Laboratories, Lucent Technologies; wadler@research.bell-labs.com*

Philip Wadler

To say that no one uses functional languages is an exaggeration. Phone calls in the European Parliament are routed by programs written in Ericsson's functional language Erlang. Virtual CDs are distributed on Cornell's network via the Ensemble system written in INRIA's CAML, and real CDs are shipped by Polygram in Europe using Software AG's Natural Expert. Functional languages are the language of choice for writing theorem provers, including the HOL system which helped debug the design of the HP 9000 line of multiprocessors. These applications and others are described in a previous column [1].

Still … I work at Bell Labs, where C and C++ were invented. Compared to users of C, "no one" is a tolerably accurate count of the users of functional languages.

Advocates of functional languages claim they produce an order of magnitude improvement in productivity. Experiments don't always verify that figure — sometimes they show an improvement of only a factor of four. Still, code that's four times as short, four times as quick to write, or four times easier to maintain is not to be sniffed at. So why aren't functional languages more widely used?

rather than built from scratch. Many of these components are written in C or C++, so a foreign function interface to C is essential, and interfaces to other languages can be useful.

The isolationist nature of functional languages is beginning to give way to a spirit of open interchange. Serious implementations now routinely provide interfaces to C, and sometimes other languages. Interworking with the imperative world is straightforward for strict languages like ML or Erlang, but trickier for lazy languages like Haskell or Clean, since laziness makes the order of evaluation difficult to predict. However, through a pleasing interplay of theory and practice, recent research has shown how abstract concepts such as monads or linear logic can be applied to smoothly interface lazy functional languages to the real world [2, 3].

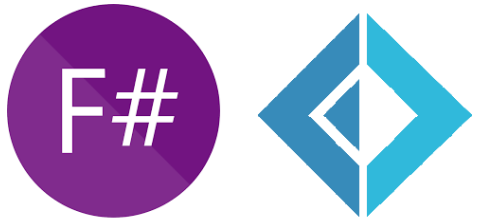
Conquering isolationism is a task for everyone, not just functional programmers. The computing industry is now beginning to deploy standards, such as CORBA and COM, that support the construction of software from reusable components. Recent work allows any Haskell program to be packaged as a COM component, and any

The World Today

Practical Strongly Typed Functional Programming Techniques are Used Everywhere

The World Today

Practical Strongly Typed Functional Programming Languages are Usable Everywhere



TypeScript

Scala

Swift

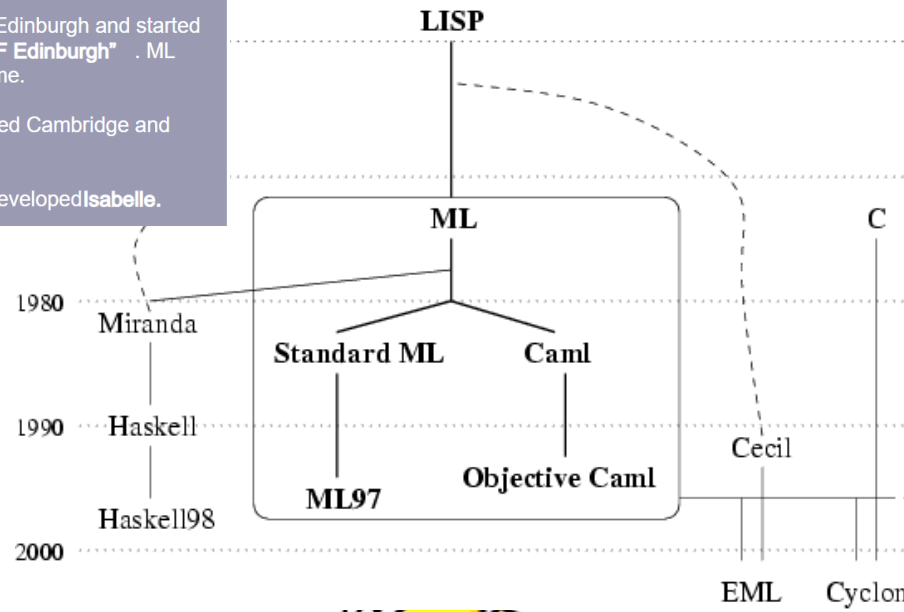
Elm

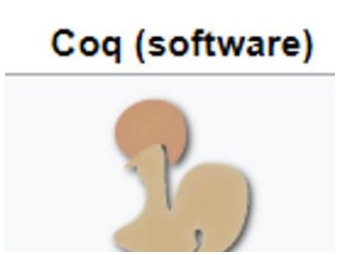
Kotlin

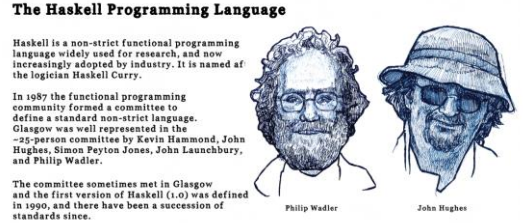
Rollback to 80s-90s

Strongly typed FP was small but active tribe.

Rooted in theorem proving, theory, Standard ML, Haskell, OCaml, experiments







The Object-Oriented Tidal Wave, 1986-1998

	Oct	*The C++ Programming Language* [Stroustrup,1986]
1986	Aug	The ''whatis paper'' [Stroustrup,1986b]
	Sep	1st OOPSLA conference (start of OO hype centered on Smalltalk)
	Nov	1st commercial Cfront PC port (Cfront 1.1, Glockenspiel)
1987	Feb	Cfront Release 1.2

A story in itself, colossal impact

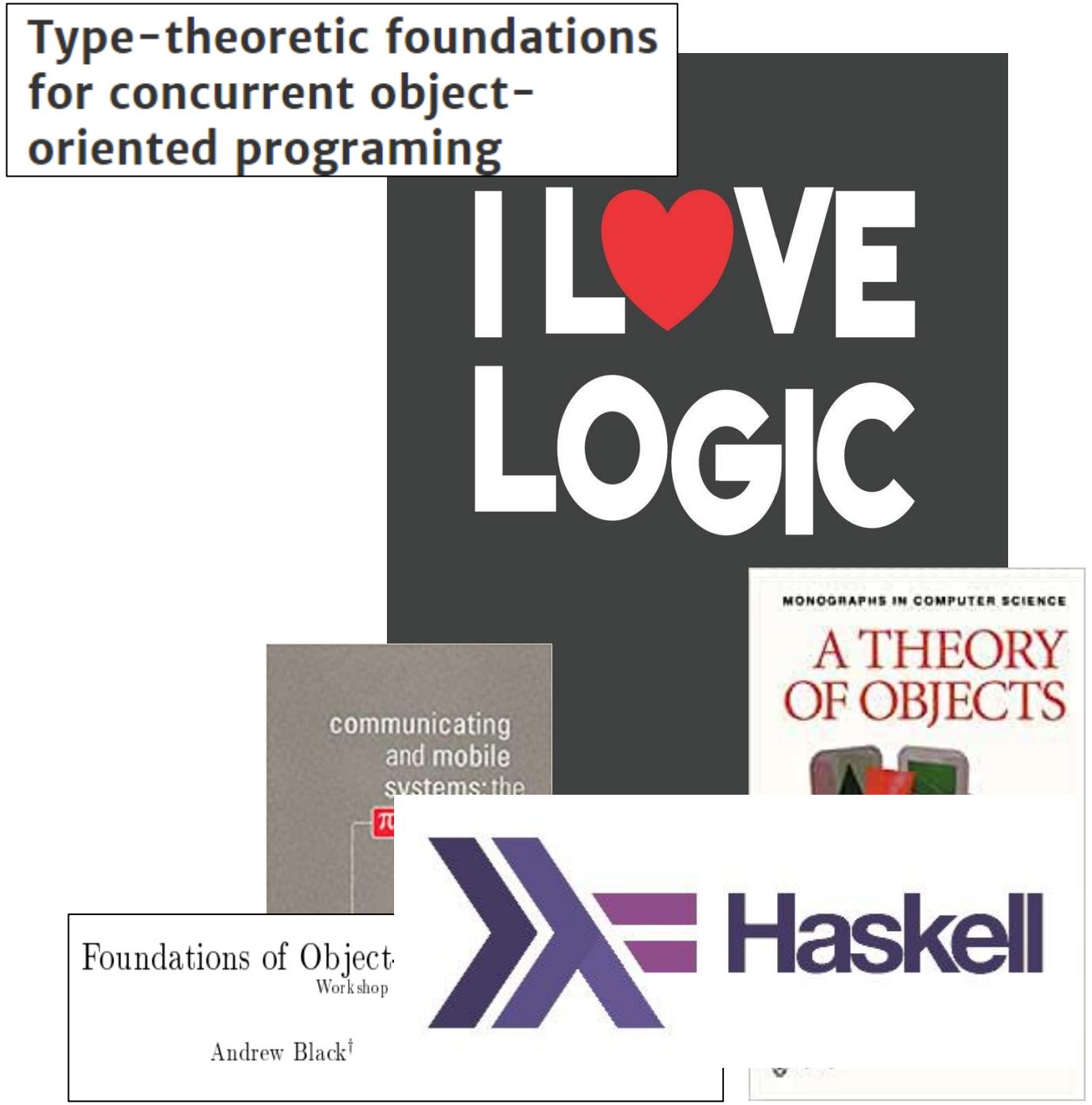
On Platform Companies (Obj-C, C++, Java)
On Microsoft (C++, Java)
On CS Academia (Java)
On Every Programming Language

Response 1

Find something else to believe in

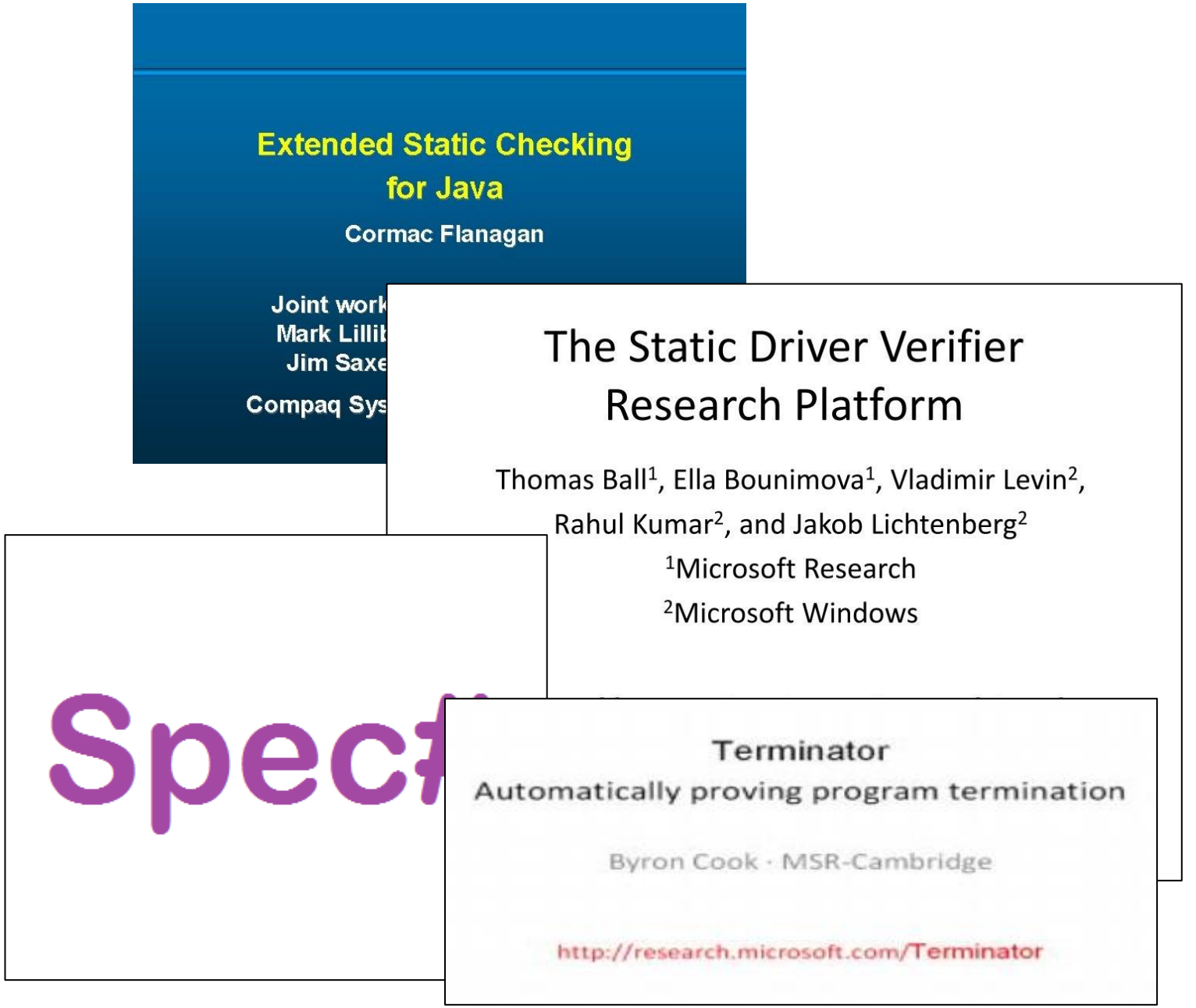
Response 2

Retreat to the fort! Focus on Foundations! Focus on Pure Functional!



Response 3

Verify the stuff using functional/theory tools!



Response 4

Oh yeah! Get my language running on those object VMs!

The MLj Compiler

MLj is a complete system for SML to Java bytecode compilation. Its features include:

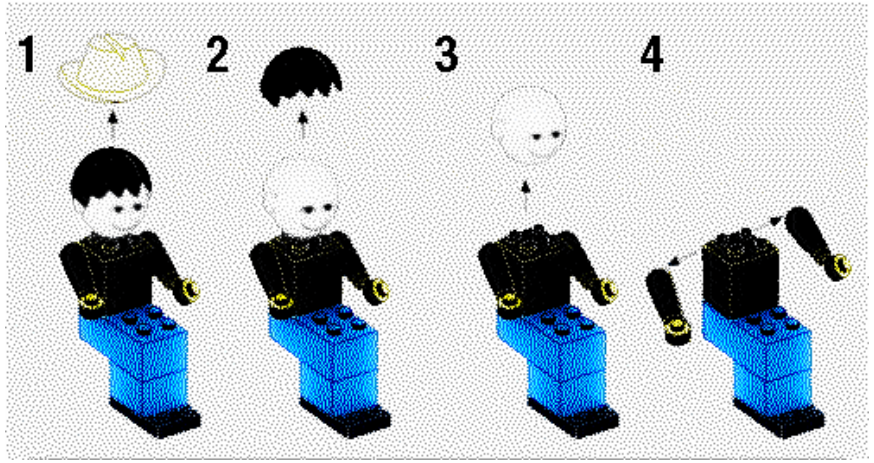
- conformance to a subset of SML '97, appro
- implementation of a large subset of the SM
- typed-checked interlanguage working exte
- automatic recompilation management;
- whole-program optimisation to produce co

I've heard that there was a project where Microsoft started to integrate Haskell on .NET and then it was replaced with the F# project. Is that true? If it's true, why?

That's a small part of the sequence. The visional design of the .NET platform was very much expected to be a multilanguage platform from the start. Right back in 1998, just in fact as our research group in programming languages started at

Response 5

Rationally deconstruct Functional and Object Programming



20+ features of OO

1. dot notation (x.Length)
2. instance members
3. type-directed name resolution
4. implicit constructors
5. static members
6. indexer notation arr.[x]
7. named arguments
8. optional arguments
9. interface types
10. mutable data
11. defining events
12. defining operators on types
13. auto properties
14. IDisposable, IEnumerable
15. type extensions
16. structs
17. delegates
18. enums
19. implementation inheritance
20. nulls and Unchecked.defaultof<_>
21. method overloading
22. curried method overloads
23. protected members
24. self types
25. wildcard types
26. aspect oriented programming ...
27. ...

???

Response 6a

Seek a synthesis!

Make Functional Languages more Object-Oriented!

Dylan

O'Haskell

O'Haskell is Haskell conservatively extended with static subtyping and monadic objects

- O'Haskell, as archived by the Wayback Machine

Response 6b

Seek a synthesis!

Make Object-Oriented Languages more Functional!

A slice of Pizza:
A quick introduction to Pizza, a dialect of Java

Martin Odersky
University of Southern Australia - Philip Wadler
Lucent Technologies

10 October 1997

Pizza is a superset of Java that incorporates four additional features.

- *Parametric polymorphism* : the ability to parameterize classes
- *Higher-order functions* : the ability to treat functions as values
- *Algebraic data types* : a convenient way of representing trees v
- *Tail calls* : don't grow the stack when the last operation in one

+ many other more recent examples

Meanwhile…. Over at Microsoft….

Microsoft®
Research

Microsoft®
.net

The worlds of Robin Milner and Bill Gates collide!

All the tribes of computer science descend on Microsoft and Microsoft Research!

Much early work centred on .NET

.NET Generics (1998-2004)

Breaking through the nominal, class-based Object-Oriented wall

Built on GJ, Pizza by Wadler/Odersky et al

"Puts .NET 20 years ahead"

MSR White Paper: Proposed Extensions to COM+ VOS (Draft)

Don Syme, Nick Benton, Simon Peyton-Jones, Cedric Fournet

1.1 Getting Serious about Language Innovation

1.2.3 Function Types, Closures and Thunks (= Generalized Delegates) 6
1.2.4 Parametric Polymorphism 6
1.3 Halfway is Not Good, but may have to do 7
2 Compound Types 8
2.1 Very Simple and Very Useful 8
2.2 An Example 9
2.3 Compound Types are needed anyway 10
2.4 Implementation Path A – Full Support 11
2.5 Implementation Path B – Support by Agreement 11
3 Parametric Polymorphism 12
3.1 Introduction 12
3.1.1 PP, Source Languages and the CLS 12
3.1.2 Overview 13
3.2 Parametric Types 13
3.3 Examples of Implementing Parametric Types 14
3.3.1 Collection Classes 14
3.3.2 Polymorphic Methods 18
3.3.3 Comparison and Sorting 19
3.3.4 Printable Lists via Bounded Parameters 20
3.3.5 Cloneable Lists 20
3.3.6 Closures and Map 21
3.4 Further Details 22
3.4.1 More on Instantiations 22

"Puts .NET 20 years ahead"

By 2002 .NET Generics was safely landed

However, C# 2.0 was still, from the strongly-typed FP perspective, verbose, unusable, untrustworthy

Sample after sample showed 3x-5x code size difference for like-for-like, full of null checking etc.

A later example

350,000

lines of C# OO
by offshore team

The C# project took five years and peaked at ~8 devs. It never fully implemented all of the contracts.

The F# project took less than a year and peaked at three devs (only one had prior experience with F#). All of the contracts were fully implemented.

30,000

lines of robust F#, with parallel +more features

An application to evaluate the revenue due from Balancing Services contracts in the UK energy industry

http://simontcousins.azurewebsites.net/does-the-language-you-use-make-a-difference-revisited/

Implementation	C#	F#
Braces	56,929	643
Blanks	29,080	3,630
Null Checks	3,011	15
Comments	53,270	487
Useful Code	163,276	16,667
App Code	305,566	21,442
Test Code	42,864	9,359
Total Code	348,430	30,801

F# 1.0 was born from an obsessive compulsion - “we must do this or else what else are we here for?”

A desire to ensure strongly-typed FP was a real option in the 2000s

Orthogonal & Unified Constructs

→ Let "let" simplify your life…

Type inference. The safety of C# with the succinctness of a scripting language

Bind a static value

Bind a static function

Bind a local value

Bind an local function

```
let data = (1,2,3)

let f a b c =
    let sum = a + b + c
    let g x = sum + x*x
    g a , g b, g c
```

OCaml

.NET Runtime

(This is a truck engine)

The F# approach

Start with Caml Core

Stay true to core FP principles

Interoperate

Reuse industrial VM+ecosystem

Deconstruct OO, FP, find synthesis

Include IDE tooling, REPL

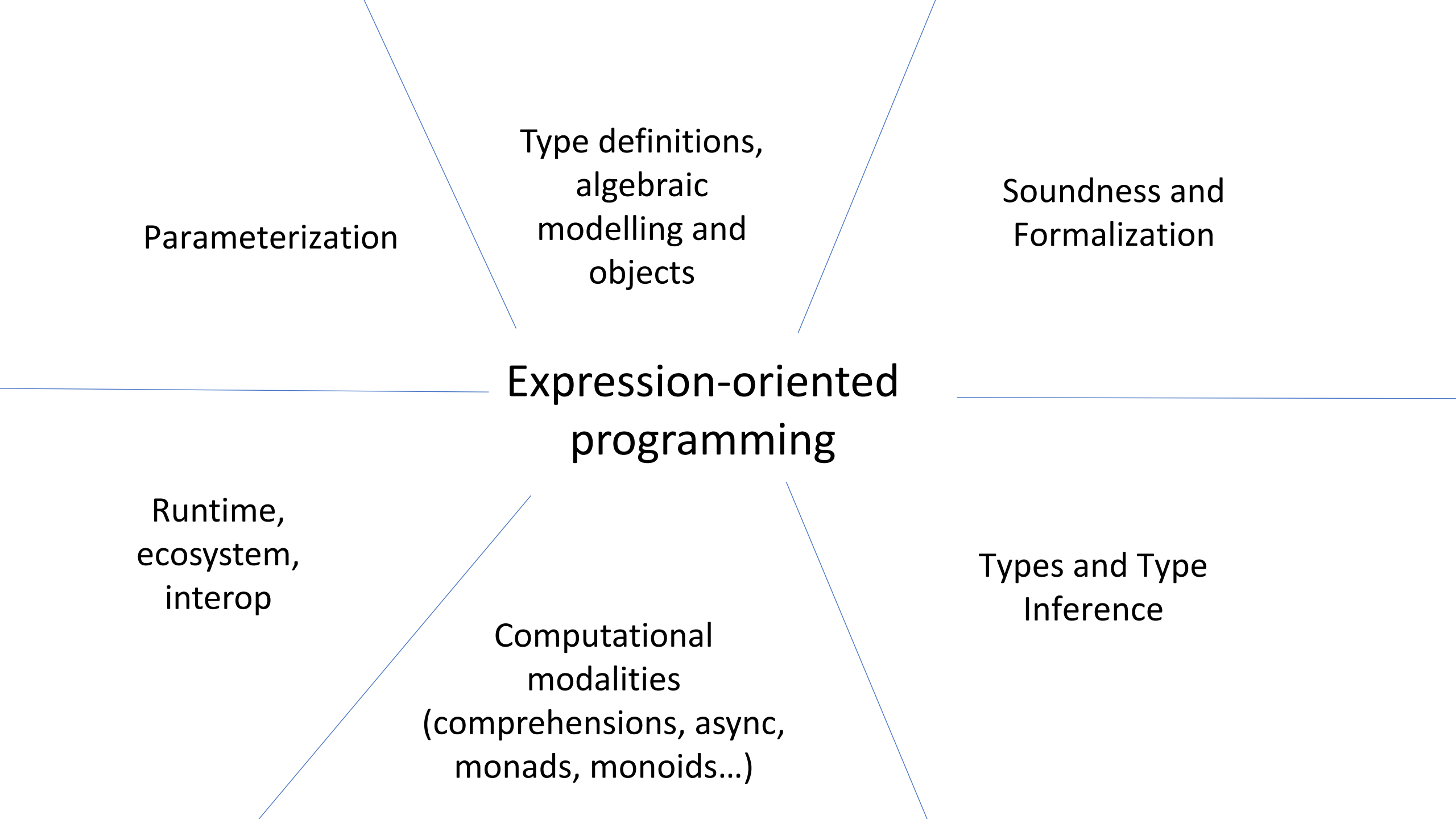
Deliver and apply in industry

Long-term home at Microsoft (Research)

Focused iterative improvements

Breaking the Rules of the Strongly Typed Functional Tribe

Beliefs held, Beliefs lost, Beliefs gained


Parameterization
Type definitions, algebraic modelling and objects
Soundness and Formalization
Expression-oriented programming
Runtime, ecosystem, interop
Computational modalities (comprehensions, async, monads, monoids…)
Types and Type Inference

Expression-oriented programming

Functors (parameterized large components)

Parameterization

Functional abstraction and parameterization

Nominal objects (classes, interfaces)

Structural objects

Type definitions

Functional objects

Immutable by default, algebraic data

Languages must have a formal specification!

Soundness supreme

Soundness

Initialization soundness, no nulls

“F# types are .NET types, .NET types are F# types”

Interop

Type Providers

List comprehensions

Computational modalities

Monadic notation (comprehensions, async, monads, monoids…)

Rich, compositional computation notation

Types for modelling, perf, correctness, interop

Hindley-Milner for typed succinctness

Types and Type Inference

Hindley-Milner modified for objects+more

Fast generic unboxed arith + inline

Type-classes and type-level computation

Types for logic and proof

2008-2014 – “F# Parachutes Into the Dark Heart of the Industry”

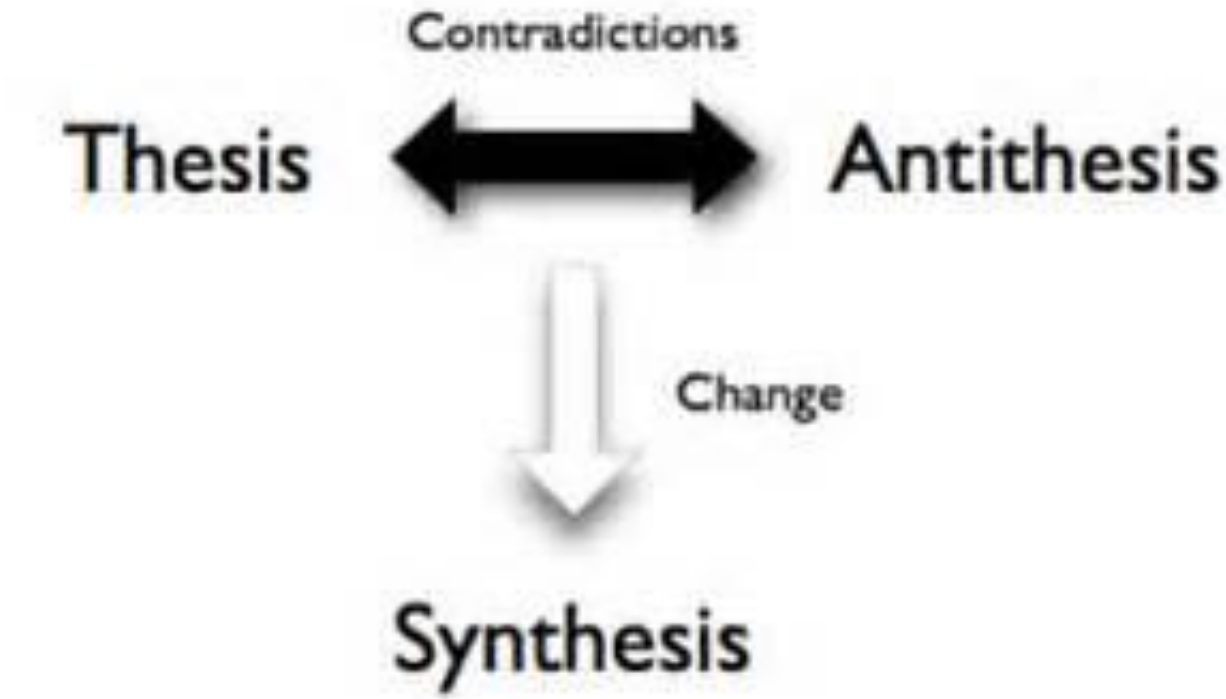
An Introduction to F# - Luca Bolognese 2008

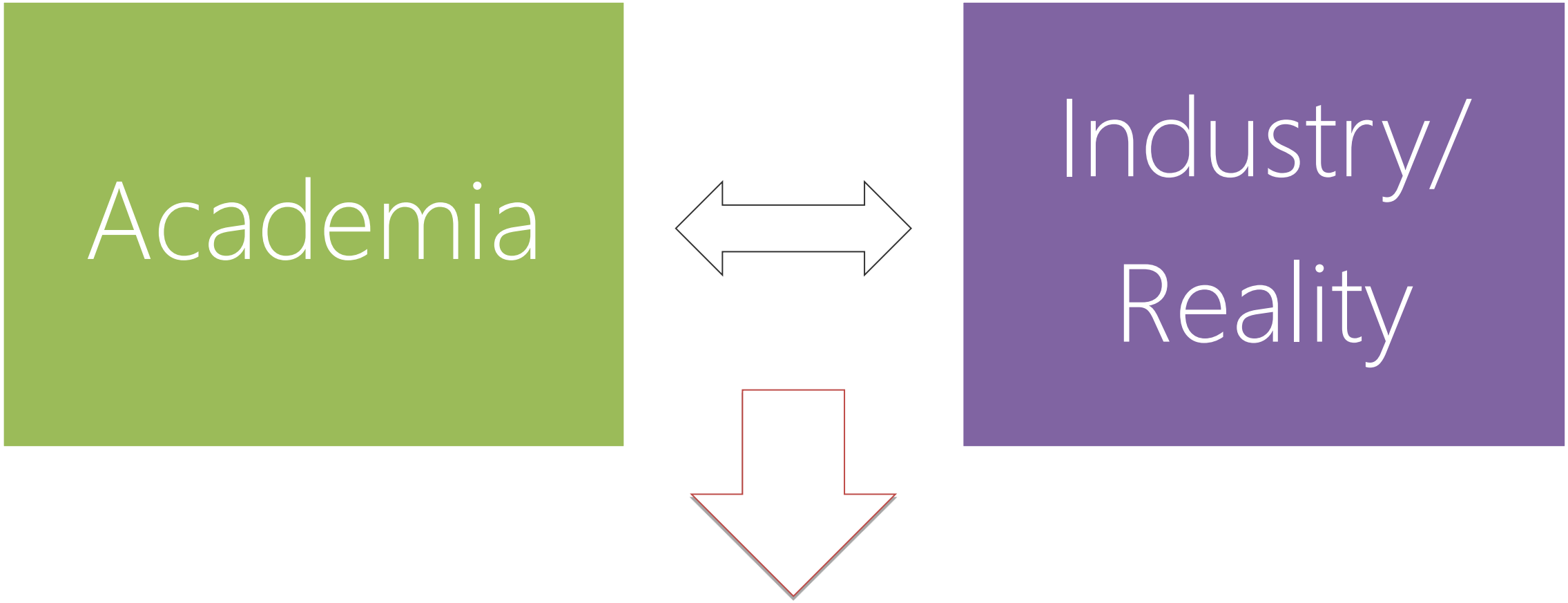
F# Eye for the C# Guy – Phil Trelford

F# Eye for the C# Guy – Leon Bambrick

F# 1.0 – 5.0

Dialectics in Action

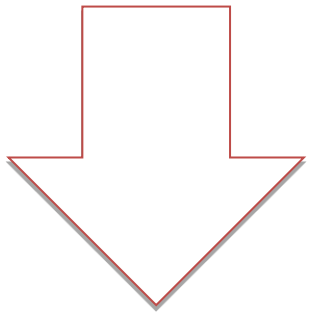

Contradictions
Thesis
Antithesis
Change
Synthesis



Microsoft Research

Functional

Interop

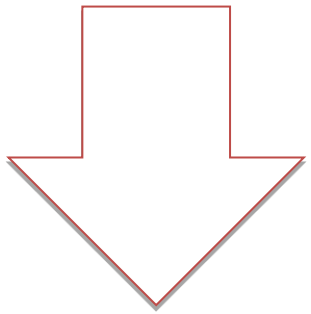


2004: F# 1.0 on .NET

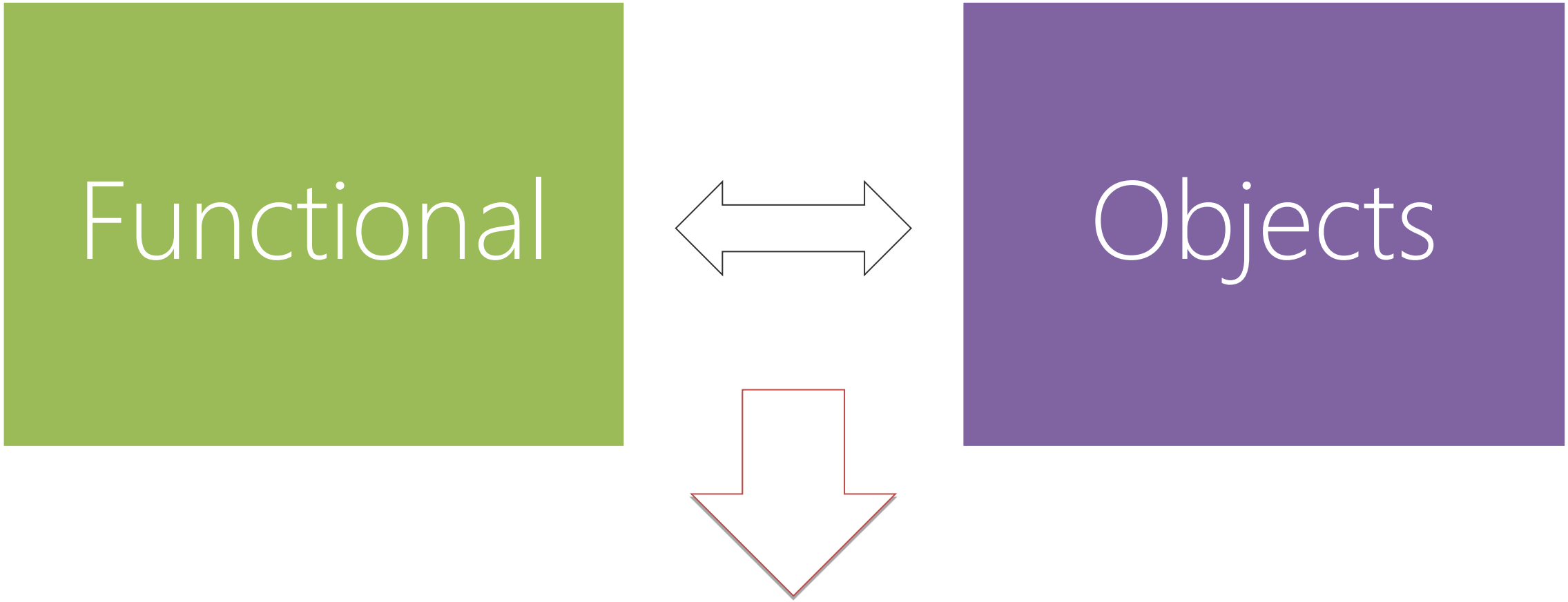
2016: Fable on Javascript

Strong Typing

Dynamic, Explorative, REPL, Data



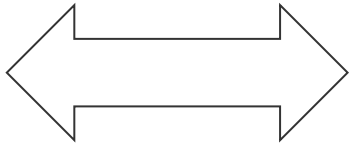
2005: F# Interactive REPL on .NET



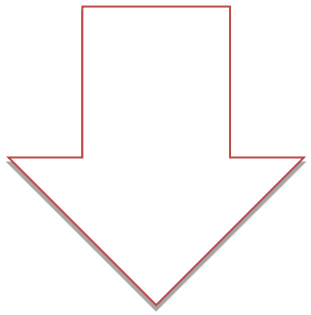
2006: F# Object Programming

(modified Hindley-Milner, type-directed name resolution, nominal, subtype-friendly, delegation-oriented, expression-friendly)

Pattern Matching



Abstraction



(3) ML does not adopt the clausal form of function definition, which is found so convenient by users of HOPE and PROLOG. How can we get a semantically rigorous form of this clausal definition, in which the constructor-patterns in formal parameters can involve not only primitive constructors, but also the constructors of user-defined abstract types? The problem is to know that these constructors are constructors, in the sense of being uniquely decomposable (or else to admit non-

https://www.pure.ed.ac.uk/ws/portalfiles/portal/17084823/Milner_R_1982_How_ML_Evolved.pdf

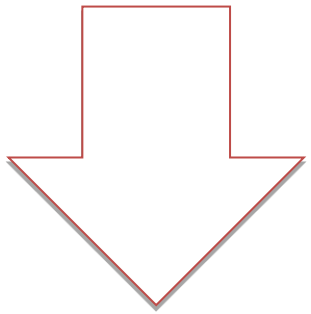
2006: F# Active Patterns

(Views made practical and simple)

Plain code

Computational modalities

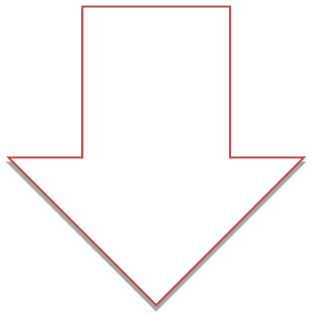
(co-routines, tasks, multi-threading, monads, monoids, comprehensions, queries, DSLs)



2007: F# Async and Computation Expressions

Typed Code

A Vast World of Connected Data

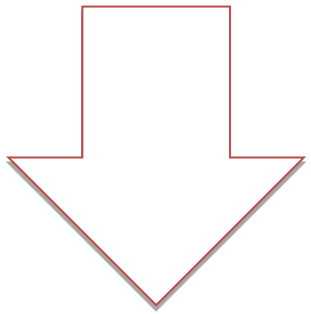


2010: F# Type Providers

(compile-time meta-programming to compute type spaces based on external information sources)

Enterprise

Openness



2010-15: C#, F#, Microsoft embrace openness and cross-platform normality

Computational modalities in C#, F# (computation expressions)

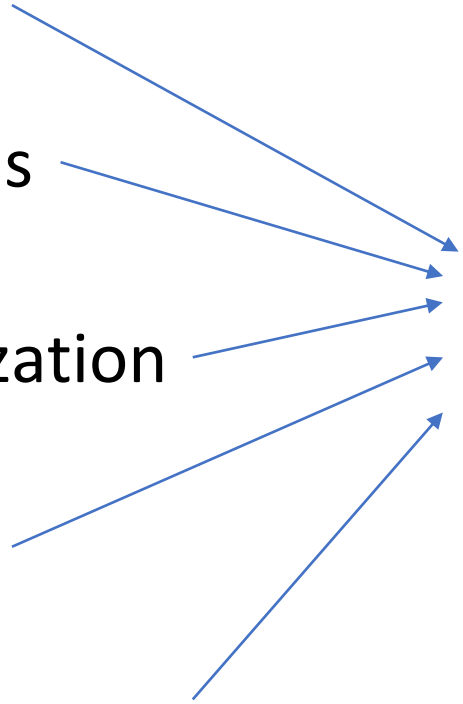
C# Iterator methods

C# Query expressions

C# Collection initialization

C# Async methods

C# Async iterator methods



modality { code }

2007 – monads, monoids, async, comprehensions
2012 – queries, extended comprehensions
2020 – applicatives

Computational modalities in C#, F# (computation expressions)

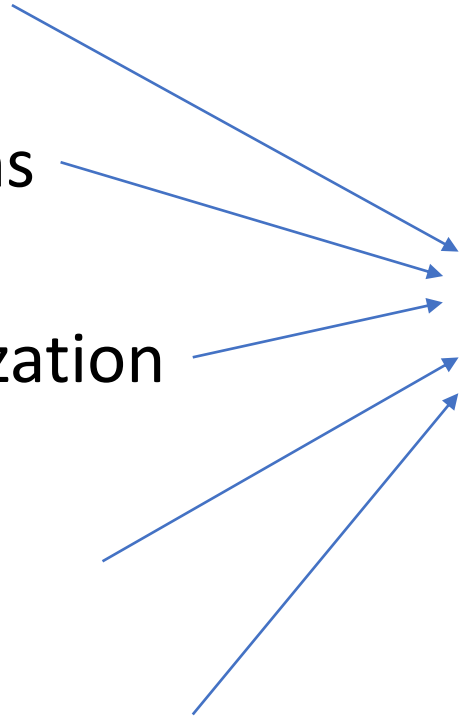
C# Iterator methods

C# Query expressions

C# Collection initialization

C# Async methods

C# Async iterator methods



seq { code + yield }

query { code + yield + custom }

list { code + yield }

async { code + bind }

asyncSeq { code + bind + yield }

coroutine { code + yield() }

F# : Objects + Functional = ❤️

```
type Vector2D(dx:double, dy:double) =

    let d2 = dx*dx+dy*dy

    member _.DX = dx

    member _.DY = dy

    member _.Length = sqrt d2

    member _.Scale(k) = Vector2D (dx*k, dy*k)
```

Inputs to object construction

Object internals

Exported properties

Exported method

Functional

Objects

	Expression-oriented	No null	Multiple Args = Tuples	Closure and Capture	First-class Values	Currying	HM Type Inference
void (unit)	✓	✓	✓	✓	✓	✓	✓
Object Types	✓	✓	✓	✓	✓		
Subtyping	✓	✓	✓	✓	✓	✓	
Dot-notation	✓	✓	✓	✓	✓	✓	¾
Inheritance	✓	✓				✓	
Method Overloding	✓	✓	✓	✓	¾	✗	¾

Relatively Graceful Degradation (some type annotations needed)

Relatively Graceful Degradation (some type annotations needed)

Some combinations outlawed for sanity

F# Today

- So much I could say about this
- F# has a strong position as the “functional language for .NET”
  - Delivery in the .NET SDK, F# exists wherever C# exists
  - Also Fable, a Javascript compiler for F#
- Strong stories for web-client, mobile, server-side, cloud
- Strong methodologies for practical, applied functional programming in practice

The influence of F#?

Culturally, F# core team existed “right next door” to C# core team, leading to many direct influences both ways

C# 3.0	• Auto-implemented properties • Anonymous types • Query expressions • Lambda expressions • Expression trees • Extension methods • Implicitly typed local variables • Partial methods • Object and collection initializers
C# 4.0	• Dynamic binding • Named/optional arguments • Generic covariant and contravariant • Embedded interop types
C# 5.0	• Asynchronous members • Caller info attributes
C# 6.0	• Static imports • Exception filters • Auto-property initializers • Expression bodied members • Null propagator • String interpolation • nameof operator • Index initializers • Await in catch/finally blocks • Default values for getter-only properties
C# 7.0	• Out variables • Tuples and deconstruction • Pattern matching • Local functions • Expanded expression bodied members • Ref locals and returns • Discards • Binary Literals and Digit Separators • Throw expressions
C# 7.1	• async Main method • default literal expressions • Inferred tuple element names • Pattern matching on generic type parameters
C# 7.2	• Techniques for writing safe efficient code • Non-trailing named arguments • Leading underscores in numeric literals • private protected access modifier • Conditional ref expressions
C# 7.3	• You can access fixed fields without pinning. • You can reassign ref local variables. • You can use initializers on stackalloc arrays. • You can use fixed statements with any type that supports a pattern. • You can use additional generic constraints.
C# 8.0	• Readonly members • Default interface methods • Pattern matching enhancements • Switch expressions • Property patterns • Tuple patterns • Positional patterns • Using declarations • Static local functions • Disposable ref structs • Nullable reference types • Asynchronous streams • Indices and ranges • Null-coalescing assignment • Unmanaged constructed types • Stackalloc in nested expressions • Enhancement of interpolated verbatim strings

+ C# 9.0 records, top-level statements

Other Influences of F#...

C# (via Hejlsberg, Meijer, Hoban, Parsons, Torgersen + others),

Scala (via Odersky + others – extractors + more)

TypeScript, Swift, Elm, Kotlin (F# was a reference point)

Python (via C# - async)

Java, C++ … (via C# - async)

Concluding Remarks

The golden thread of strongly-typed FP is the simplicity of the core experience, e.g.

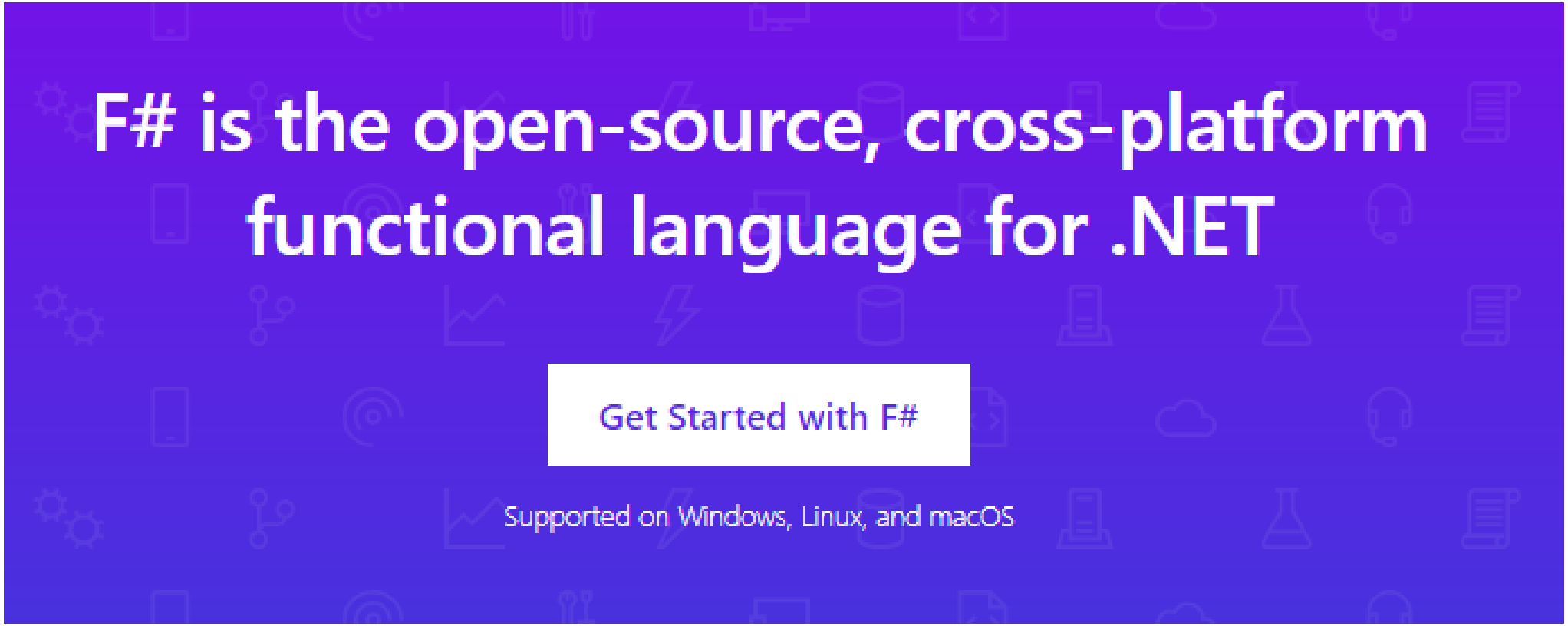
```
let f x = (x, x)
type term = T of string * term list
...
```

Milner, Newey, Morris saw this in 1970, I personally experienced it in 1990, and it's stayed true.

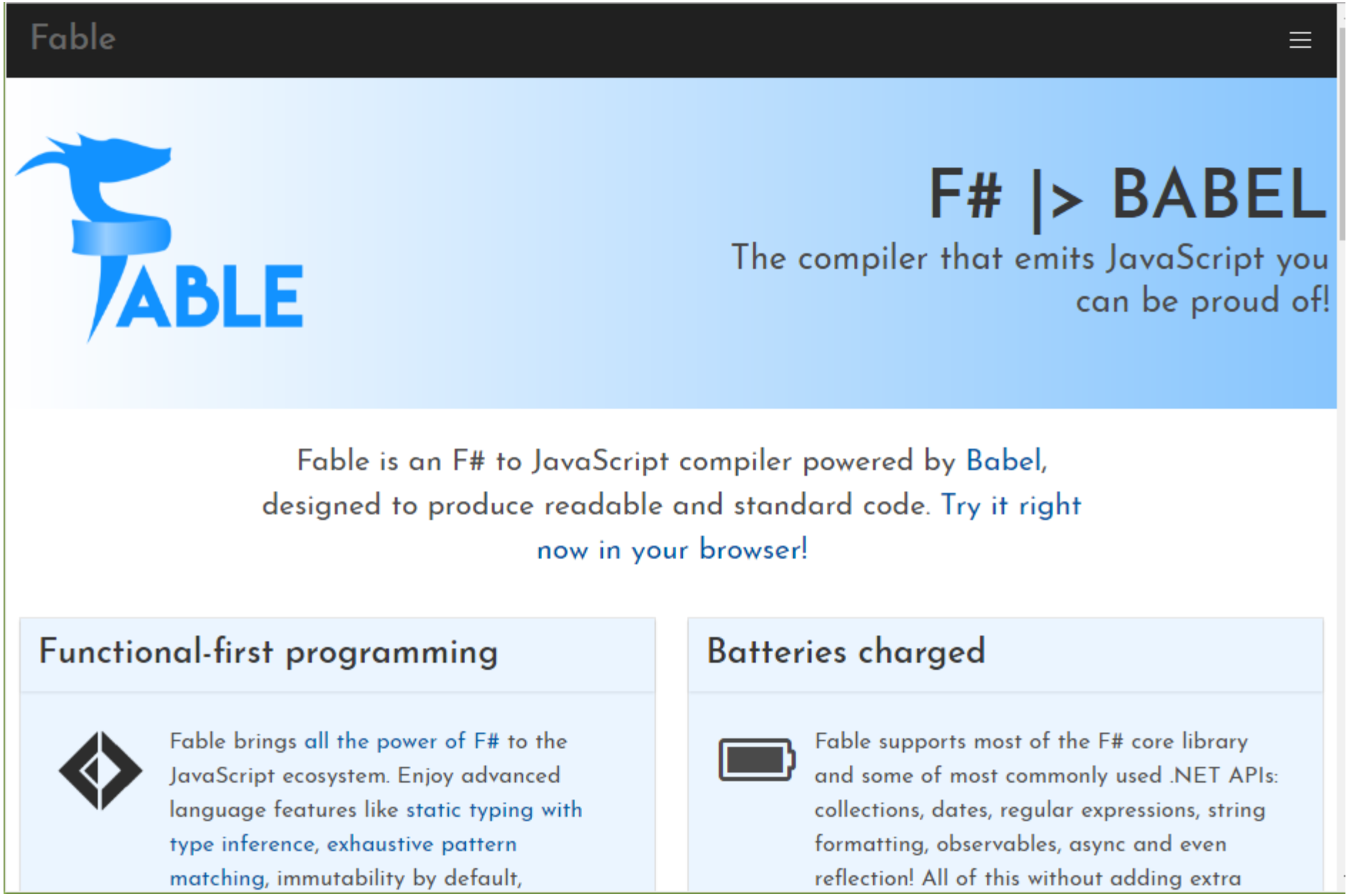
Concluding Notes

- The programming world has corrected the OO dominance of the late 90s
  - **Generics, lambdas**, **parameterization**, **immutability**, **algebraic types**, **various computational modalities**, **type inference** have now rightly occupied their place as practical core techniques in nearly all programming languages
- F# has been a part of this story since 2003
  - Part of the reaction of the functional tribe to the OO tidal wave
  - Stayed true to its core design goals
  - Many practical additions to the expression-oriented, statically typed paradigm
  - A strong user base and a secure future
- One way or another, the golden thread will appeal far into the future.

Backup slides



www.microsoft.com/net/


Fable
F# |> BABEL
The compiler that emits JavaScript you can be proud of!
Fable is an F# to JavaScript compiler powered by Babel, designed to produce readable and standard code. Try it right now in your browser!
Functional-first programming
Fable brings all the power of F# to the JavaScript ecosystem. Enjoy advanced language features like static typing with type inference, exhaustive pattern matching, immutability by default,
Batteries charged
Fable supports most of the F# core library and some of most commonly used .NET APIs: collections, dates, regular expressions, string formatting, observables, async and even reflection! All of this without adding extra

F# get started

```
dotnet new -lang F#

dotnet build
```

F# get started

```
dotnet new -lang F#

dotnet build
```

F# tools are part of the .NET SDK, available everywhere

F# for the backend

```
dotnet new -i "giraffe-template::*"

dotnet giraffe
```

GIRAFFE

A functional ASP.NET Core micro web framework for building rich web applications.

github.com/giraffe-fsharp/Giraffe

F# for the backend

```
dotnet new -i "giraffe-template::*"

dotnet giraffe
```

High perf, functional server-side programming

A functional ASP.NET Core micro web framework for building rich web applications.

github.com/giraffe-fsharp/Giraffe

F# for the frontend (JS)


```
dotnet new -i "Fable.Template::*"

dotnet new fable
npm install
npm start
```

F# for the frontend (JS)


```
dotnet new -i "Fable.Template::*"

dotnet new fable
npm install
npm start
```

You can use F# as a Javascript language

F# for the full stack

```
dotnet new -i SAFE.Template

dotnet new SAFE
dotnet tool restore
dotnet fake run
```

A $3B Unicorn, Built on F#

https://techcrunch.com/2016/08/07/walmart-buys-jet-com-for-3-billion/

News Startups Mobile Gadgets Trending Tesla Google Facebook

Advertising Tech | jet.com | Walmart Labs | Walmart

Walmart is buying Jet.com for $3 billion

Posted Aug 7, 2016 by Jonathan Shieber (@jshieber), John Mannes (@JohnMannes)

Next Story

Walmart Stores is buying Jet.com in a deal worth $3 billion dollars according to a source with direct knowledge of the deal, confirming reports that have been pouring in about the bid for Jet.com all week.

According to our source the signatures for the deal were dry on Friday and will be announced as early as Monday morning — echoing what was reported in both Bloomberg and Recode.

www.managedsolution.com/jet-built-its-entire-e-commerce-platform-including-development-and-del

MANAGED SOLUTION

Home About + Products & Services + Client Support + Contact Us Try Before You Buy!

Jet built its entire e-commerce platform, including development and delivery infrastructure, on Microsoft #Azure.

Jet.com – E-commerce challenger eyes the top spot, runs on the Microsoft cloud

Marc Lore is perhaps best known as the creator of the popular e-commerce site Diapers.com, which was eventually sold to Amazon. Now, the entrepreneur and his team are ready to compete head-on with the e-retailing giant through an innovative online marketplace called Jet.com. To get up and running quickly, Jet built its entire e-commerce platform, including development and delivery infrastructure, on Microsoft Azure, using both .NET and open-source technologies.

Business Challenge

In 2010, Marc Lore sold his company Quidsi (which ran e-retailing sites like Diapers.com and Soap.com) to Amazon for $550 million. Four years later, Marc is competing against Amazon directly—with the creation of a new online marketplace called Jet.com.

There are many reasons to think that Lore might just pull it off. For one, he plans to eliminate any margins from product sales. The company's only source of revenue will come from membership dues, eliminating the kind of mark-ups that Amazon charges and passing the savings on to the customer. In addition, an innovative pricing engine will work to reduce or eliminate costs in the e-commerce value chain, especially fulfillment costs and marketplace commissions.

"Our pricing engine will continually work out the most cost-effective way to fulfill an order from merchant locations closest to the consumer,' explains Lore, Co-Founder and CEO of Jet. "The engine will also figure out which merchants can fulfill most cheaply by putting multiple

F# 5.0! (2021)

- ✔ #r nuget packages in scripts “#r "nuget: Newtonsoft.Json"
- ✔ Jupyter and .NET Interactive notebooks!
- ✔ string interpolation
- ✔ nameof
- ✔ applicatives syntax in computation expressions
- ✔ improved .NET interop
- ✔ improved Map/Set performance + more

https://github.com/fsharp/fslang-design/tree/master/FSharp-5.0

Enter F# Type Providers….

"Just like a library"

"A design-time component that computes a space of types and methods on-demand…"

"An adaptor between data/services and the .NET type system…"

"On-demand, scalable compile-time provision of type/module definitions…"

https://fsprojects.github.io/FSharp.Data/images/json.gif