Scalable Integration of Heterogeneous Financial Data and Functional Programming

A world of financial data, at your fingertips

Dr. Don Syme

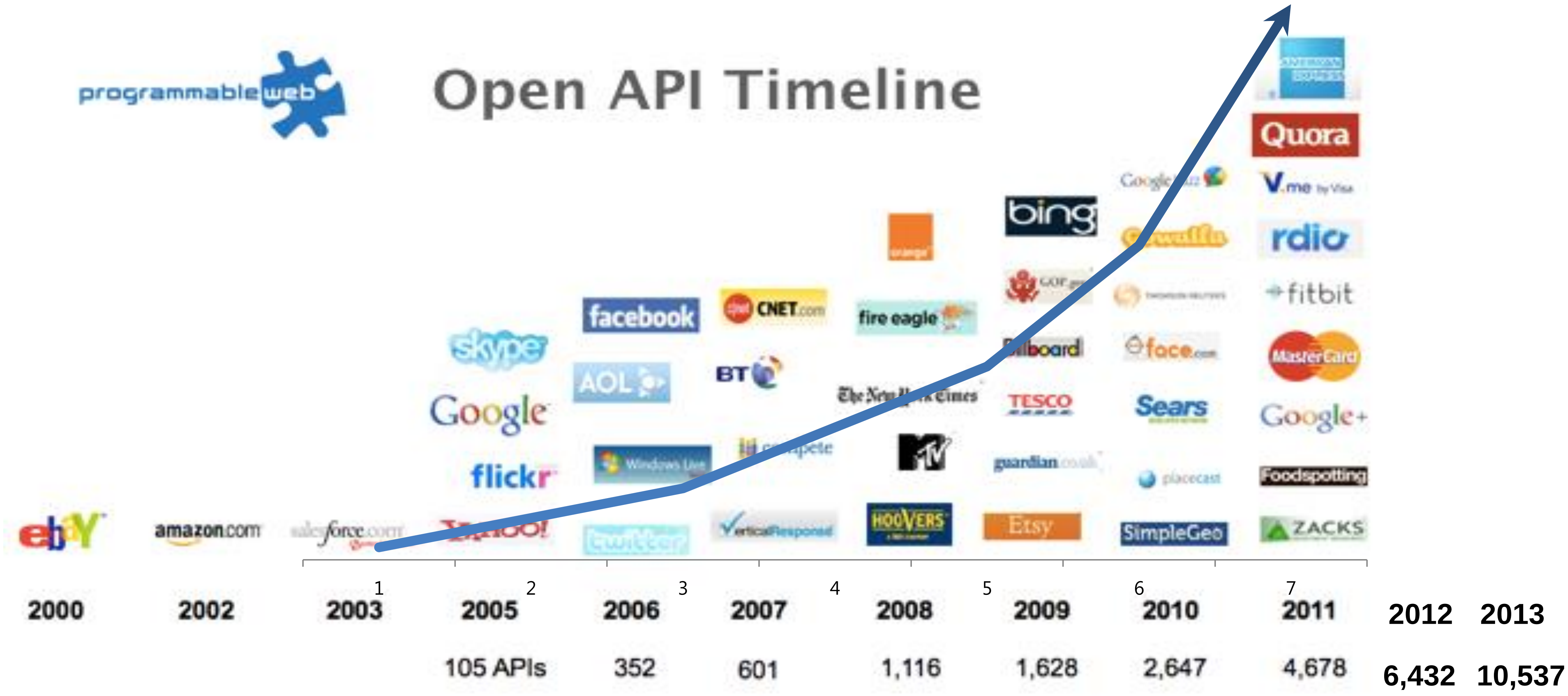
Principal Researcher, Microsoft Research

Agenda

- The “Information Rich Programming” Problem
  - broadly conceived
  - and for financial data specifically
- A bit about F#
- Applying F# scalable data/metadata integration techniques to financial data integration

Proposition 1
We are living in an Information Revolution

The Information Revolution



Proposition 2
Modern programming is intensely information-rich

Proposition 3
Our programming languages are information-sparse

Financial data is like water…

Financial data is like water…

Everyone needs it. Everyone knows where to get it. Simple.

But…

… nobody is sure where it really came from, or goes to.

… nobody really knows its true cost, or true value.

… nobody likes to pay for it, or to share it.

… nobody knows how much is wasted

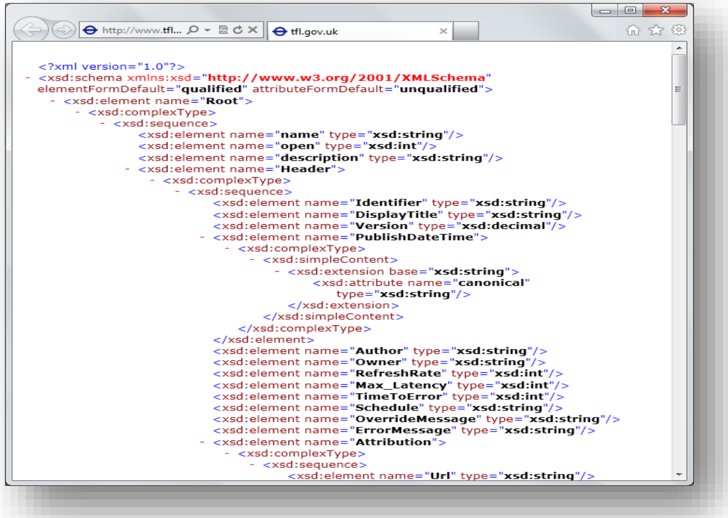
… nobody knows a good plumber

… nobody knows how bad it is until after you have drunk it

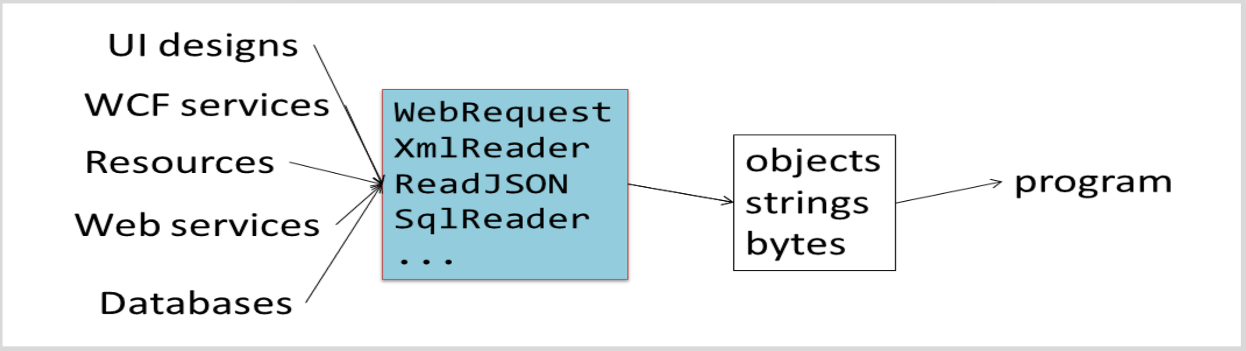
Actually these days it’s more like a flood…

The financial data plumber's perspective

- Languages do not integrate information
    - Weakly typed
    - Not at scale
    - Non-intuitive
    - Not simple
    - Disorganised
    - Static
    - High friction



```
   <?xml version="1.0"?>
 - <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified" attributeFormDefault="unqualified">
   - <xsd:element name="Root">
     - <xsd:complexType>
       - <xsd:sequence>
           <xsd:element name="name" type="xsd:string"/>
           <xsd:element name="open" type="xsd:int"/>
           <xsd:element name="description" type="xsd:string"/>
         - <xsd:element name="Header">
           - <xsd:complexType>
             - <xsd:sequence>
                 <xsd:element name="Identifier" type="xsd:string"/>
                 <xsd:element name="DisplayTitle" type="xsd:string"/>
                 <xsd:element name="Version" type="xsd:decimal"/>
               - <xsd:element name="PublishDateTime">
                 - <xsd:complexType>
                   - <xsd:simpleContent>
                     - <xsd:extension base="xsd:string">
                          <xsd:attribute name="canonical"
                            type="xsd:string"/>
                       </xsd:extension>
                     </xsd:simpleContent>
                   </xsd:complexType>
                 </xsd:element>
                 <xsd:element name="Author" type="xsd:string"/>
                 <xsd:element name="Owner" type="xsd:string"/>
                 <xsd:element name="RefreshRate" type="xsd:int"/>
                 <xsd:element name="Max_Latency" type="xsd:int"/>
                 <xsd:element name="TimeToError" type="xsd:int"/>
                 <xsd:element name="Schedule" type="xsd:string"/>
                 <xsd:element name="OverrideMessage" type="xsd:string"/>
                 <xsd:element name="ErrorMessage" type="xsd:string"/>
               - <xsd:element name="Attribution">
                 - <xsd:complexType>
                   - <xsd:sequence>
                       <xsd:element name="Url" type="xsd:string"/>
```



UI designs

WCF services

Resources

Web services

Databases

WebRequest
XmlReader
ReadJSON
SqlReader
. . .

objects
strings
bytes

program

We need to bring information **into** our “data rich” languages…

At **market-scale, in-cloud or on-premise, strongly tooled, strongly typed**

But before we get into that…

This R&D is done in the context of the open language F#

Lots of good reasons for that, see our tech report
"Strongly Typed Language Support for Internet-Scale Information Spaces"

A bit about F#….

F# is free, open source, cross platform

fsharp.org

F# is a programming language

F# is a functional-first programming language

F# is a succinct, interoperable, efficient, strongly-typed programming language

F# is a programming language with scalable, seamless data interoperability

"Functional-first programming is a general-purpose programming technique particularly suited to tasks where Time-to-deployment, Efficiency, Correctness and Taming Complexity dominate."

"Examples include executable financial models, market simulators, ETL pipelines, general data-manipulation, calculation engines, service implementation, programmatic Uis, machine-learning and data science.

While these problems can be solved using other programming paradigms, they are particularly amenable to functional-first programming."

Functional-first programming uses functional programming as the initial paradigm for most purposes, but employs other techniques such as objects and state as necessary.

F# has an intelligent, fun, contributing community

fsharp.org
meetup.com/FSharpLondon
fsharpforfunandprofit.com

F# helps address real business problems

around **Time to Market**, **Efficiency**, **Correctness** and **Tackling Complexity**

See: "Succeeding with Functional-first Programming in Finance", Don Syme

F# is used to make lots of money

fsharp.org/testimonials

Example #1: Energy trading simulation

I have written an application to balance the national power generation schedule … for an energy company.

…the calculation engine was written in F#.

The use of F# to address the complexity at the heart of this application clearly demonstrates a sweet spot for the language … algorithmic analysis of large data sets.

Simon Cousins

Example #1: Energy trading simulation

Time to Market

Interoperation … Seamless. The C# programmer need never know.

Correctness

Units of measure … a huge time saver...it eradicates a whole class of errors.

Time to Market

Exploratory programming …Working with F# Interactive allowed me to explore the solution space more effectively.

Correctness

Unit testing …a joy to test. There are no complex time-dependent interactions to screw things up….

Efficiency

Parallelism …The functional purity ... makes it ripe for exploiting the inherent parallelism in processing vectors of data.

Time to Market

Code reduction… … ve matrices…higher order functions eat these for breakfast with minimal fuss, minimal code. Beautiful.

Correctness

Lack of bugs… Function feel strange. .. once the type checker is satisfied that’s often it, it works.

Example #2: F# for Quant Consulting

Complexity

- Our bids for tendered contracts in quantitative finance are regularly half the price of competitors because of the increased productivity we get from F#.

Efficiency

- We are regularly able to deliver correct, robust, performant solutions on-time, which is what our customers value most.

Time to Market

Correctness

Daniel Egloff, QuantAlea Consulting, Zurich

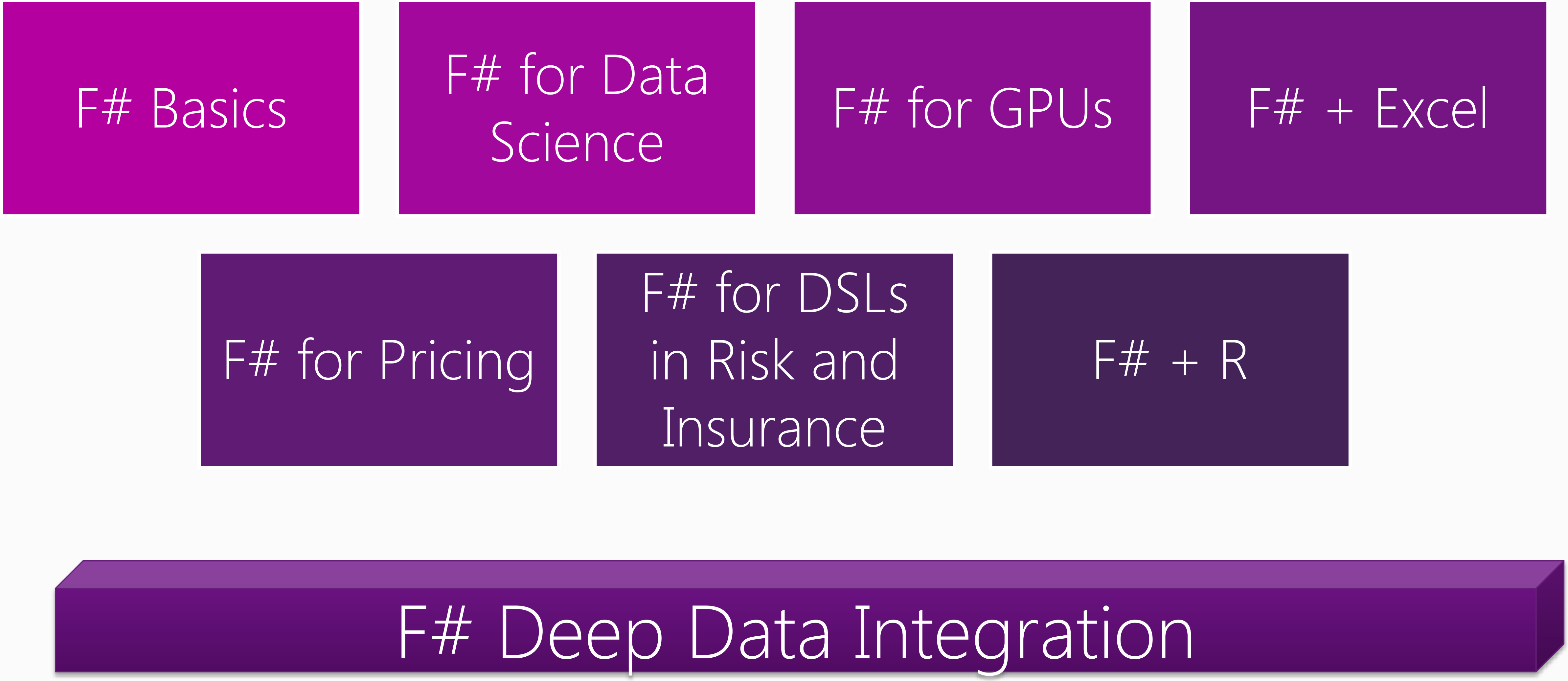
http://fsharp.org/testimonials

Microsoft recommend
F# and the Visual F# tools
for your functional-first programming needs

(note: works seamlessly with C#, can interoperate with C/C++/R/Python/…)

Back to the main topic…

Typical F# Topics



F# Basics

F# for Data Science

F# for GPUs

F# + Excel

F# for Pricing

F# for DSLs in Risk and Insurance

F# + R

F# Deep Data Integration

The Problem We’re Addressing

Our data plumbing tools are data-sparse. We need to bring financial information **into** the language…

At market-scale, strongly tooled, strongly typed

Which data?

Data like this enables entire cloud-based industries in financial analytics

Xenomorph Timescape is a major Microsoft partner for **Financial Data** in the **Azure Cloud Platform**

The specific questions:

Can we use F#'s unique "type provider" capabilities to integrate Xenomorph data at fine granularity?

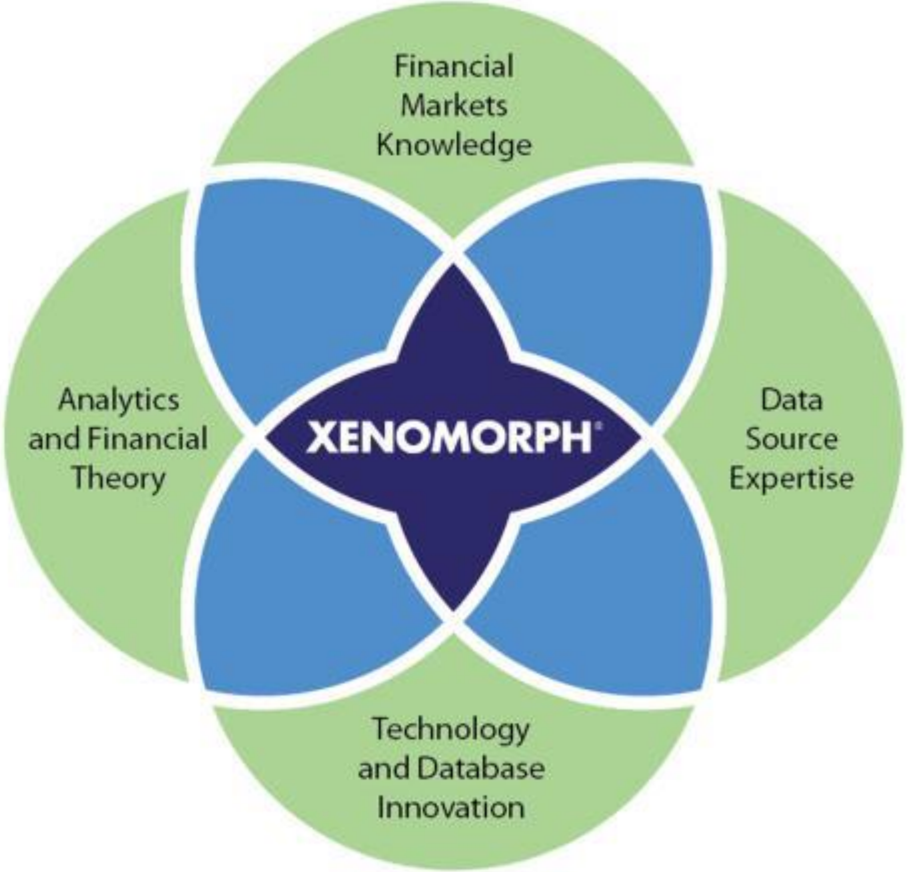
Can we specifically tackle the **quantity** of instruments and the **sparseness** of the data in a **programming language integration**.

XENOMORPH®

TimeScape - Leading the way in Analytics and Data Management

About Xenomorph

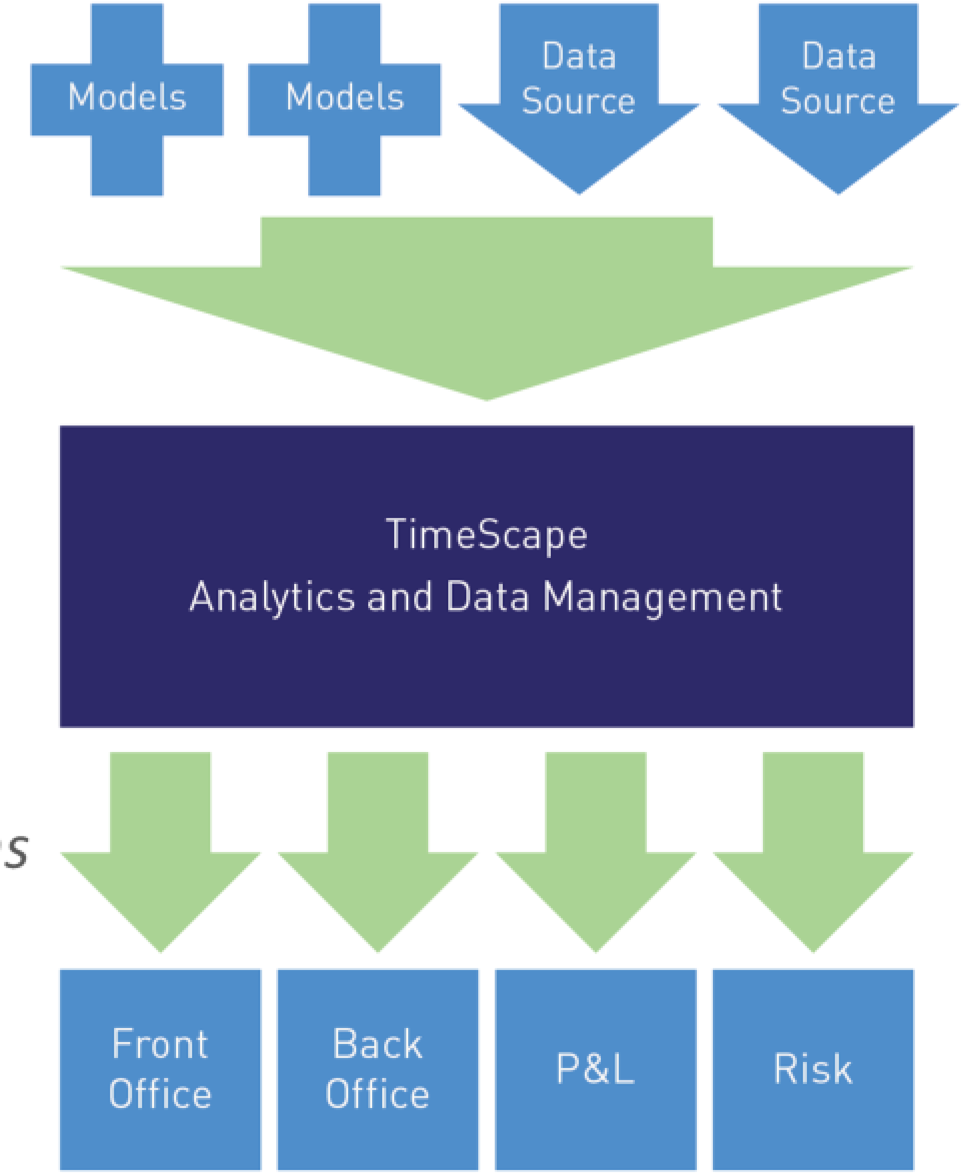
- Founded in 1995
- Serving a global client base
- Excellence in service and support
- Expert in rapid project delivery


Financial Markets Knowledge
Analytics and Financial Theory
XENOMORPH®
Data Source Expertise
Technology and Database Innovation

XENOMORPH®

TimeScape by Xenomorph

An integrated solution to large and complex data problems



Who uses TimeScape?

Front to back office

- Trading
- Quants
- Research
- Product control
- Risk
- IT and operations

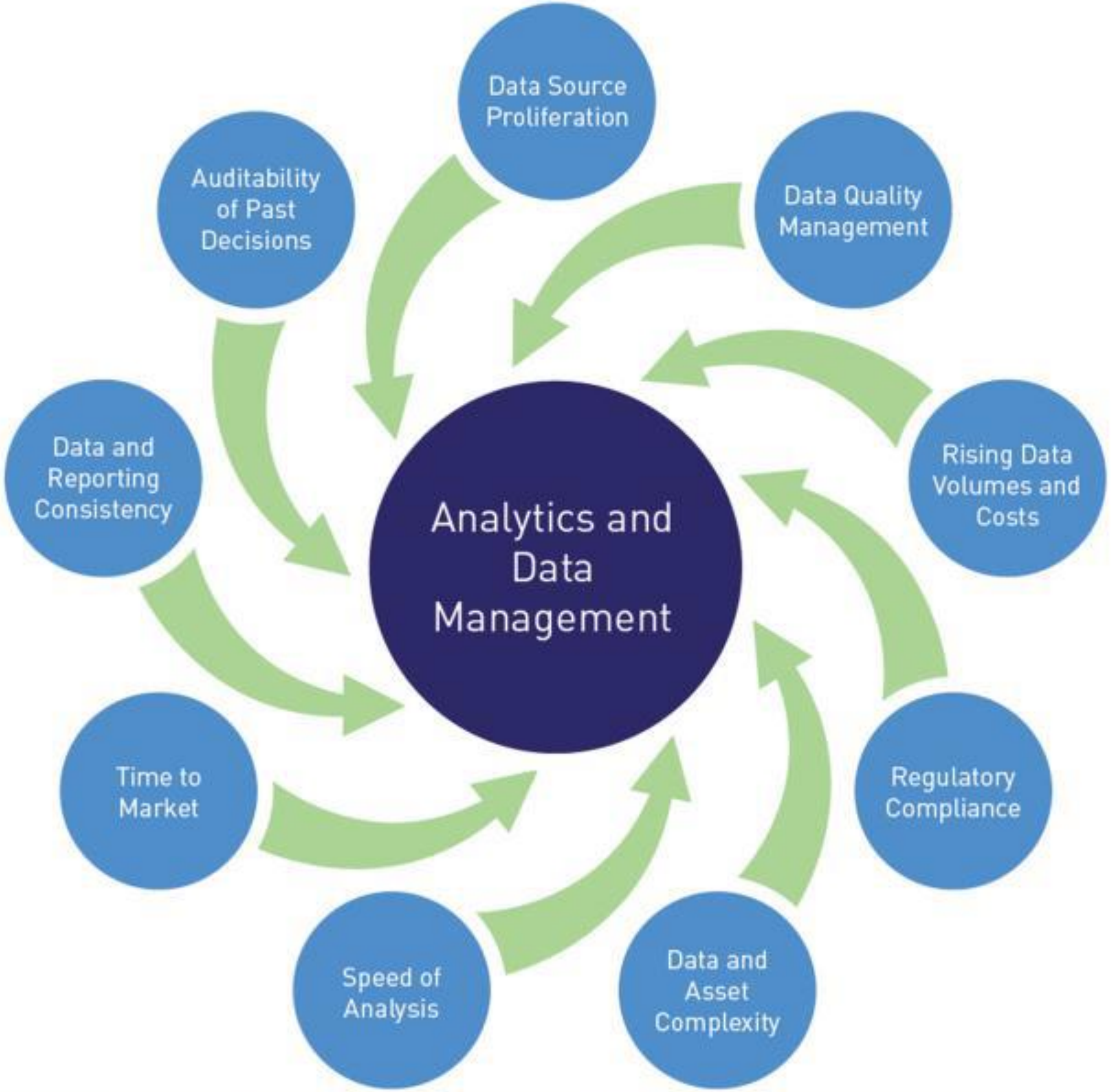
Sell-side to buy-side

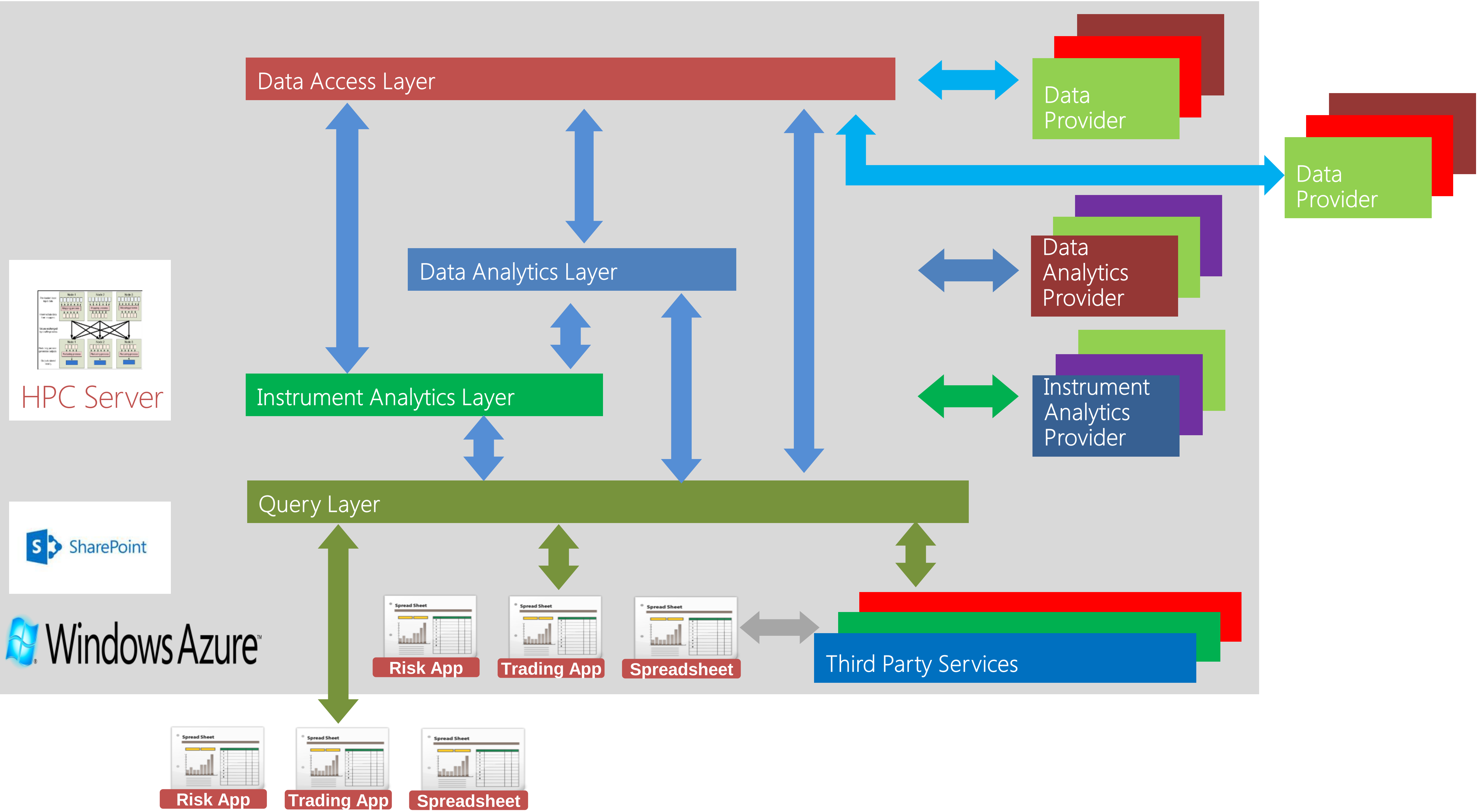
- Investment banks
- Brokers
- Energy traders
- Hedge funds
- Asset managers
- Insurers

XENOMORPH

Industry-wide issues

- *More data*
- *More complexity*
- *More analysis*


Analytics and Data Management
Data Source Proliferation
Data Quality Management
Rising Data Volumes and Costs
Regulatory Compliance
Data and Asset Complexity
Speed of Analysis
Time to Market
Data and Reporting Consistency
Auditability of Past Decisions



Data Access Layer

Data Provider

Data Provider

Data Analytics Layer

Data Analytics Provider

HPC Server

Instrument Analytics Layer

Instrument Analytics Provider

Query Layer

SharePoint

Windows Azure

Spread Sheet

Risk App

Spread Sheet

Trading App

Spread Sheet

Spreadsheet

Third Party Services

Spread Sheet

Risk App

Spread Sheet

Trading App

Spread Sheet

Spreadsheet

The problem is this:

the data is still weakly integrated into our programming languages

The Technique: “F# Type Providers”

Allow scalable, robust, deep integration

Similar Problem: Integrate all of freebase.com

“as if it were a library”

40M entities, 1Billion facts, 24,000 types, 65,000 properties

Similar Problem: Integrate all of worldbank.org/data

“as if it were a library”

10,000+ time series for hundreds of countries/regions

Demo

F# + Freebase

An F# type provider for deep, robust integration of web data

A Type Provider is….

“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 F# type system…”

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

But what about financial data?

Demo

F# + Xenomorph TimeScape

An F# type provider for deep, robust integration of financial data

Theme #1

On-Demand Types = Internet Scalable Magic

On-Demand Type Provision

```
let data = RiskLab.GetDataContext()
```

1. Compiler/IDE requests metadata for symbol **`GetDataContext`**
   - ✓ Provider reports return type of **`RiskLabDataContext`**

```
data.Categories
```

2. Compiler/IDE requests contents of **`RiskLabDataContext`** and property **`Categories`**
   - ✓ Provider asks Freebase metadata service for top-level domains
   - ✓ Provider reports top-level domains of Freebase as properties of the type

```
data.Categories.``GB Equities``
```

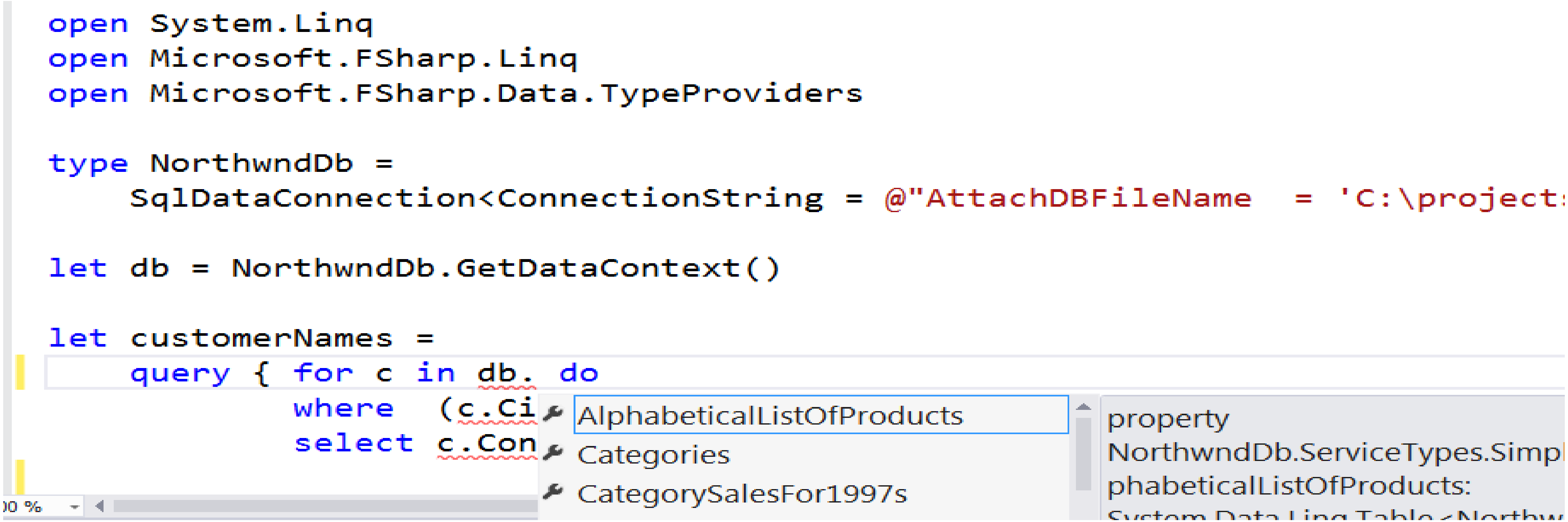
3. Compiler/IDE requests metadata for symbol **``` ``GB Equities`` ```**

…

Theme #2

Many Data Sources, One Mechanism

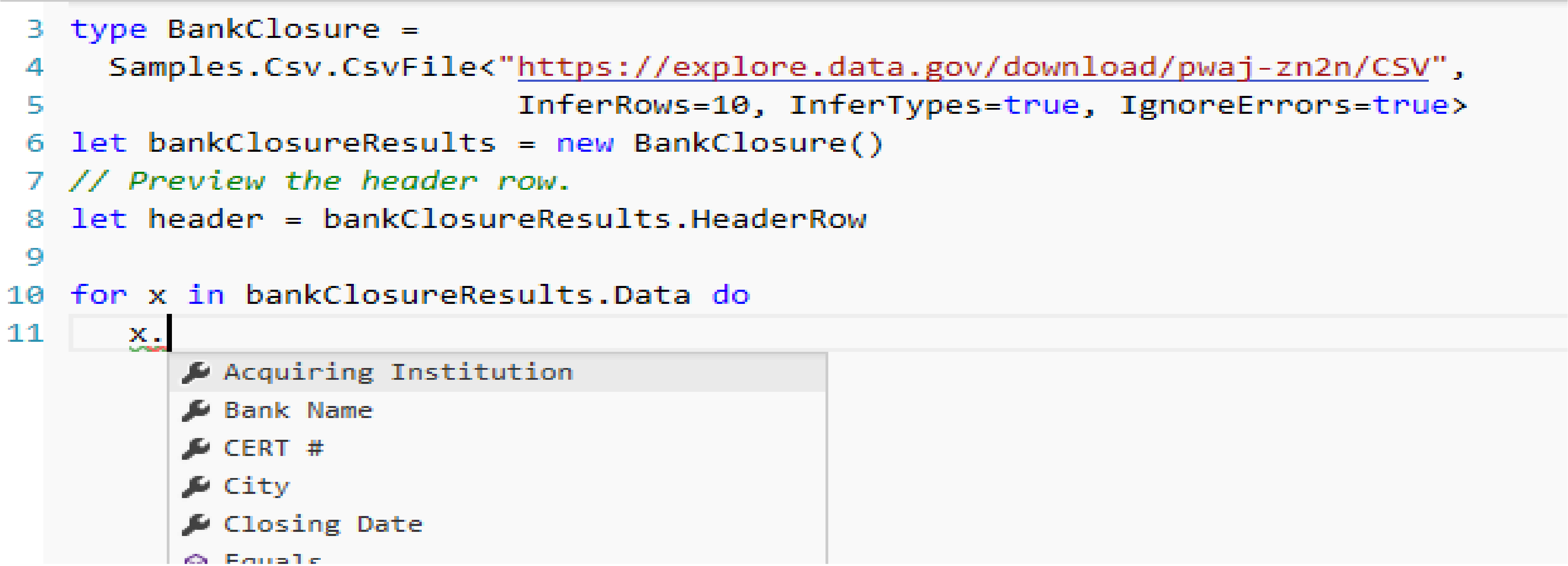
SQL



```
open System.Linq
open Microsoft.FSharp.Linq
open Microsoft.FSharp.Data.TypeProviders

type NorthwndDb =
    SqlDataConnection<ConnectionString = @"AttachDBFileName  = 'C:\project

let db = NorthwndDb.GetDataContext()

let customerNames =
    query { for c in db. do
            where  (c.Ci
            select c.Con
```

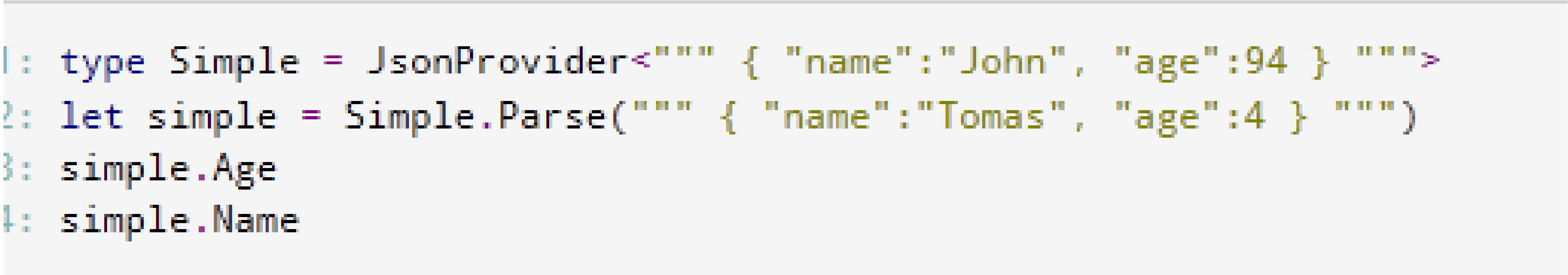
AlphabeticalListOfProducts
Categories
CategorySalesFor1997s
property
NorthwndDb.ServiceTypes.Simpl
phabeticalListOfProducts:
System.Data.Linq.Table<Northw
00 %

CSV



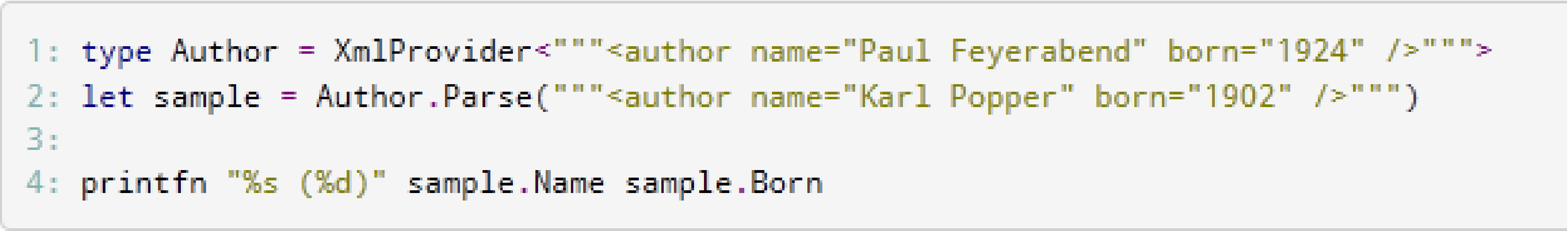
```
type BankClosure =
  Samples.Csv.CsvFile<"https://explore.data.gov/download/pwaj-zn2n/CSV",
                      InferRows=10, InferTypes=true, IgnoreErrors=true>
let bankClosureResults = new BankClosure()
// Preview the header row.
let header = bankClosureResults.HeaderRow

for x in bankClosureResults.Data do
    x.
```

JSON



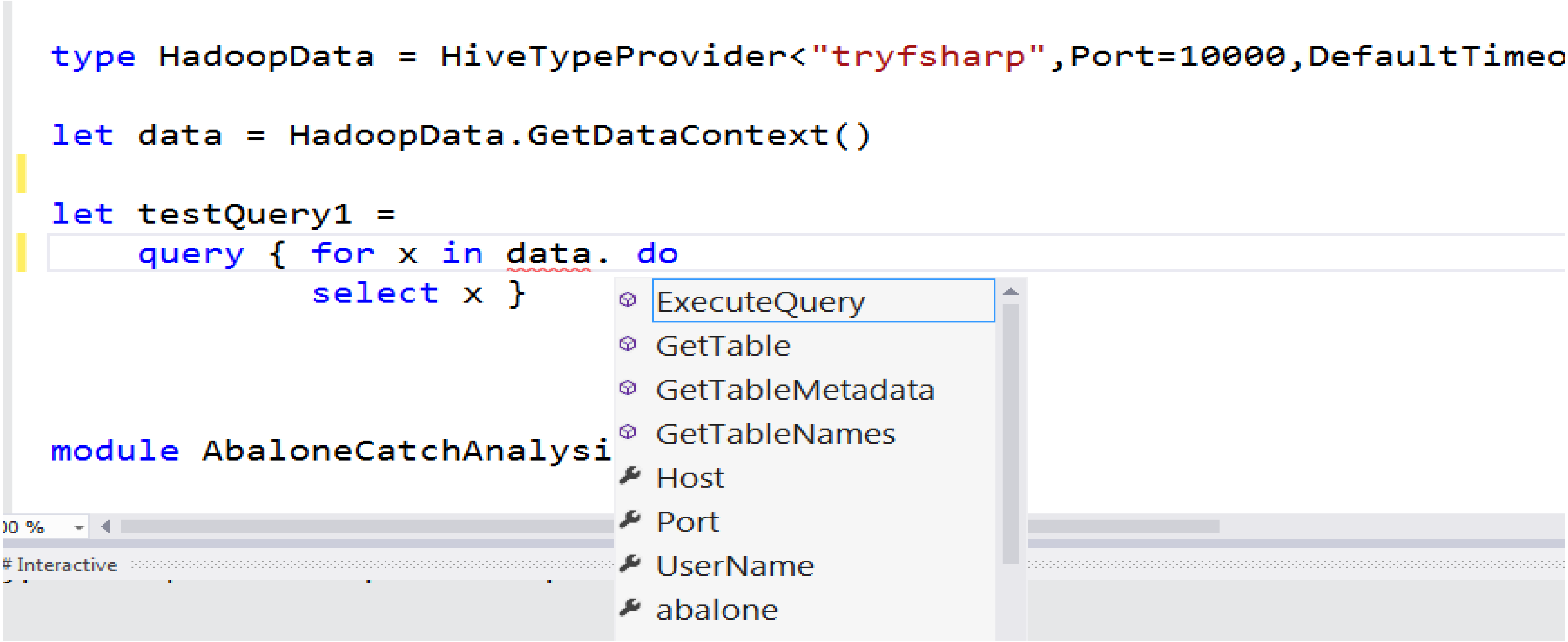
```
type Simple = JsonProvider<""" { "name":"John", "age":94 } """>
let simple = Simple.Parse(""" { "name":"Tomas", "age":4 } """)
simple.Age
simple.Name
```

XML



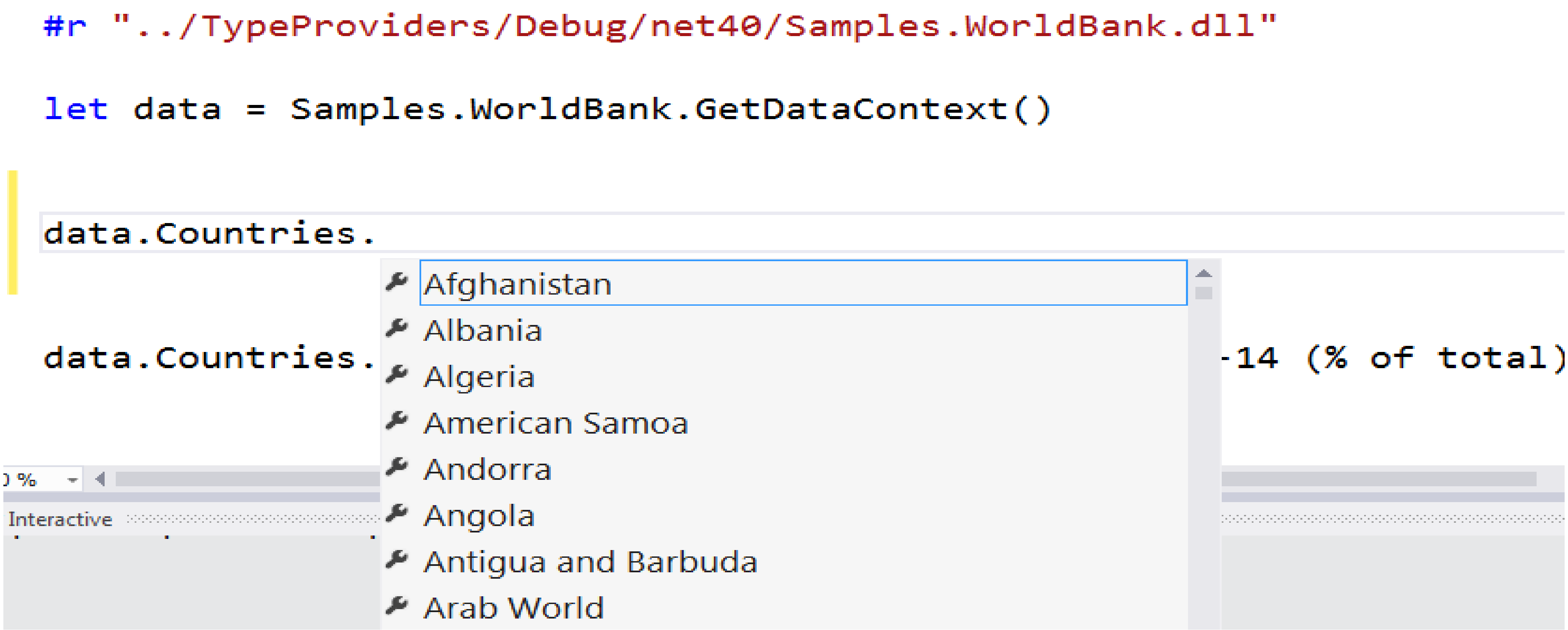
```
type Author = XmlProvider<"""<author name="Paul Feyerabend" born="1924" />""">
let sample = Author.Parse("""<author name="Karl Popper" born="1902" />""")

printfn "%s (%d)" sample.Name sample.Born
```

Hadoop/Hive



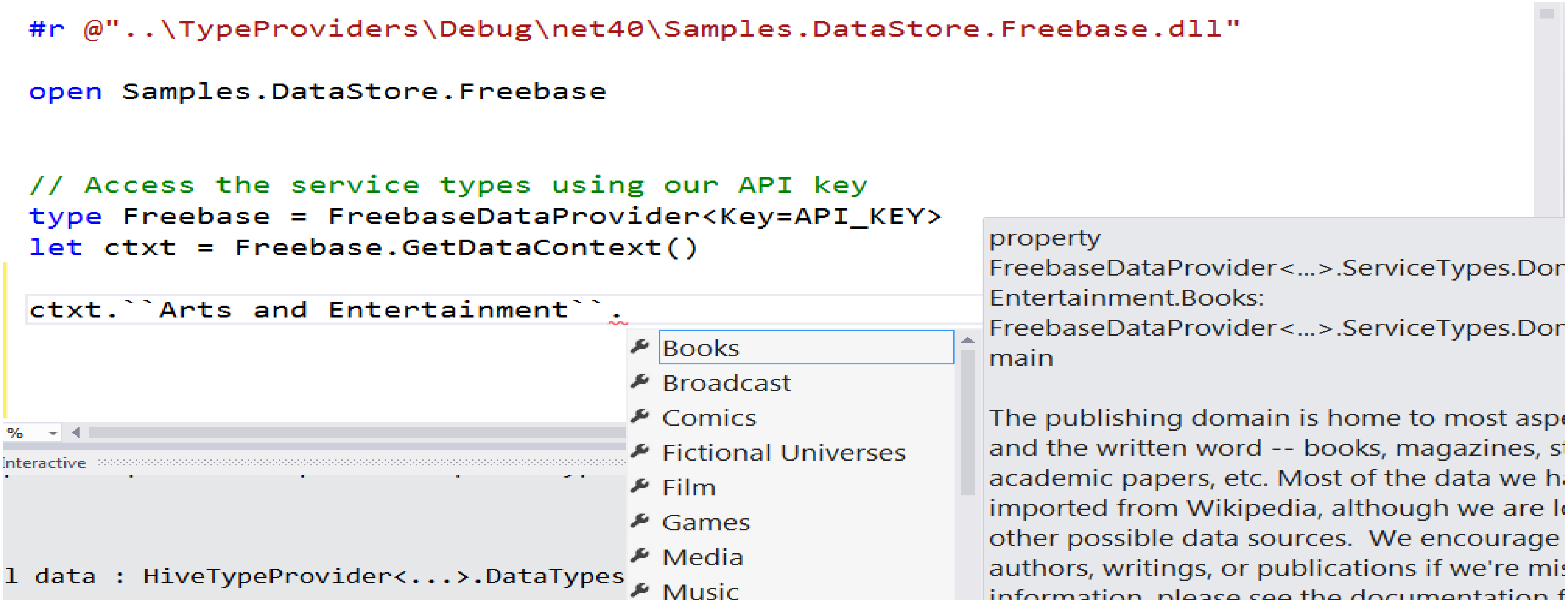
```
type HadoopData = HiveTypeProvider<"tryfsharp",Port=10000,DefaultTimeo

let data = HadoopData.GetDataContext()

let testQuery1 =
    query { for x in data. do
                select x }


module AbaloneCatchAnalysi
```

World Bank



```
#r "../TypeProviders/Debug/net40/Samples.WorldBank.dll"

let data = Samples.WorldBank.GetDataContext()

data.Countries.

data.Countries.                                             -14 (% of total)
```

Freebase



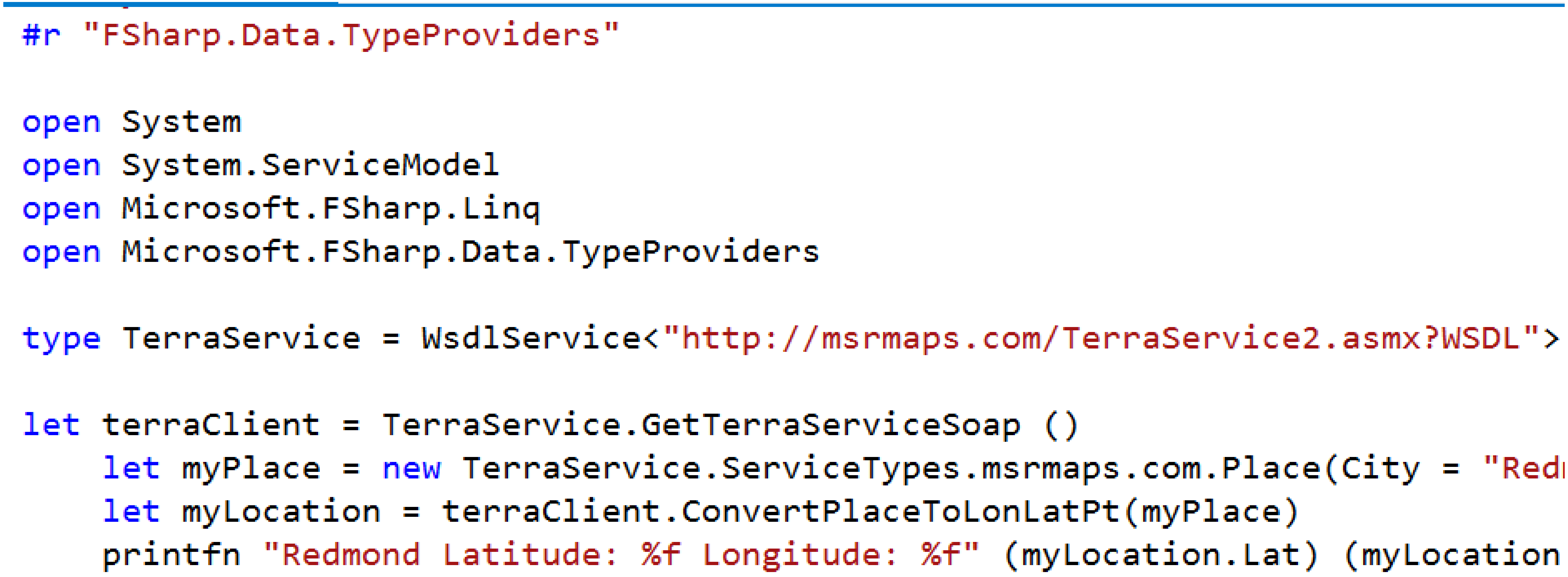
```
#r @"..\TypeProviders\Debug\net40\Samples.DataStore.Freebase.dll"

open Samples.DataStore.Freebase


// Access the service types using our API key
type Freebase = FreebaseDataProvider<Key=API_KEY>
let ctxt = Freebase.GetDataContext()

ctxt.``Arts and Entertainment``.
```

property
FreebaseDataProvider<...>.ServiceTypes.Dor
Entertainment.Books:
FreebaseDataProvider<...>.ServiceTypes.Dor
main

The publishing domain is home to most aspe
and the written word -- books, magazines, s
academic papers, etc. Most of the data we h
imported from Wikipedia, although we are l
other possible data sources. We encourage
authors, writings, or publications if we're mis
information, please see the documentation f

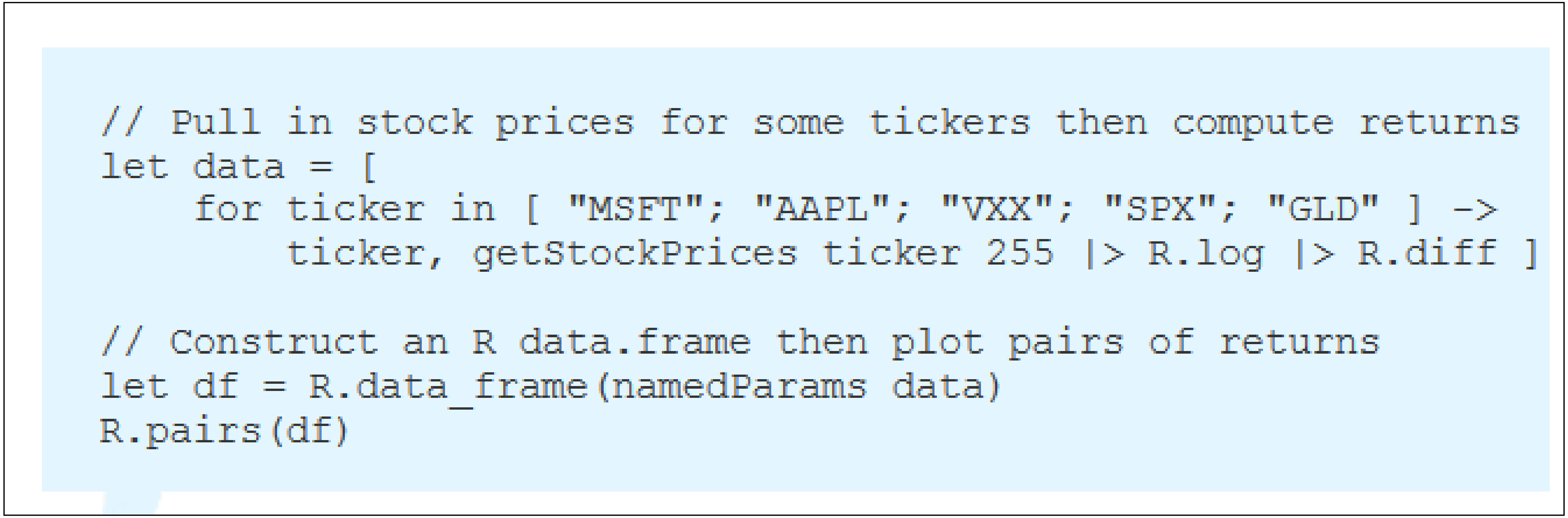
```
l data : HiveTypeProvider<...>.DataTypes
```

WSDL



```
#r "FSharp.Data.TypeProviders"

open System
open System.ServiceModel
open Microsoft.FSharp.Linq
open Microsoft.FSharp.Data.TypeProviders

type TerraService = WsdlService<"http://msrmaps.com/TerraService2.asmx?WSDL">

let terraClient = TerraService.GetTerraServiceSoap ()
    let myPlace = new TerraService.ServiceTypes.msrmaps.com.Place(City = "Red
    let myLocation = terraClient.ConvertPlaceToLonLatPt(myPlace)
    printfn "Redmond Latitude: %f Longitude: %f" (myLocation.Lat) (myLocation
```

R



```
// Pull in stock prices for some tickers then compute returns
let data = [
    for ticker in [ "MSFT"; "AAPL"; "VXX"; "SPX"; "GLD" ] ->
        ticker, getStockPrices ticker 255 |> R.log |> R.diff ]

// Construct an R data.frame then plot pairs of returns
let df = R.data_frame(namedParams data)
R.pairs(df)
```

Demo

F# + R

Theme #3

Data and Types at Multiple Scales

Data at Multiple Scales

From Everything to Individuals

```
data.AllEntites
```

```
data.Categories.``GB Equity``
```

```
data.Categories.``GB Equity``.``SAINSBURY(J) (SBRY.L, Reuters)``
```

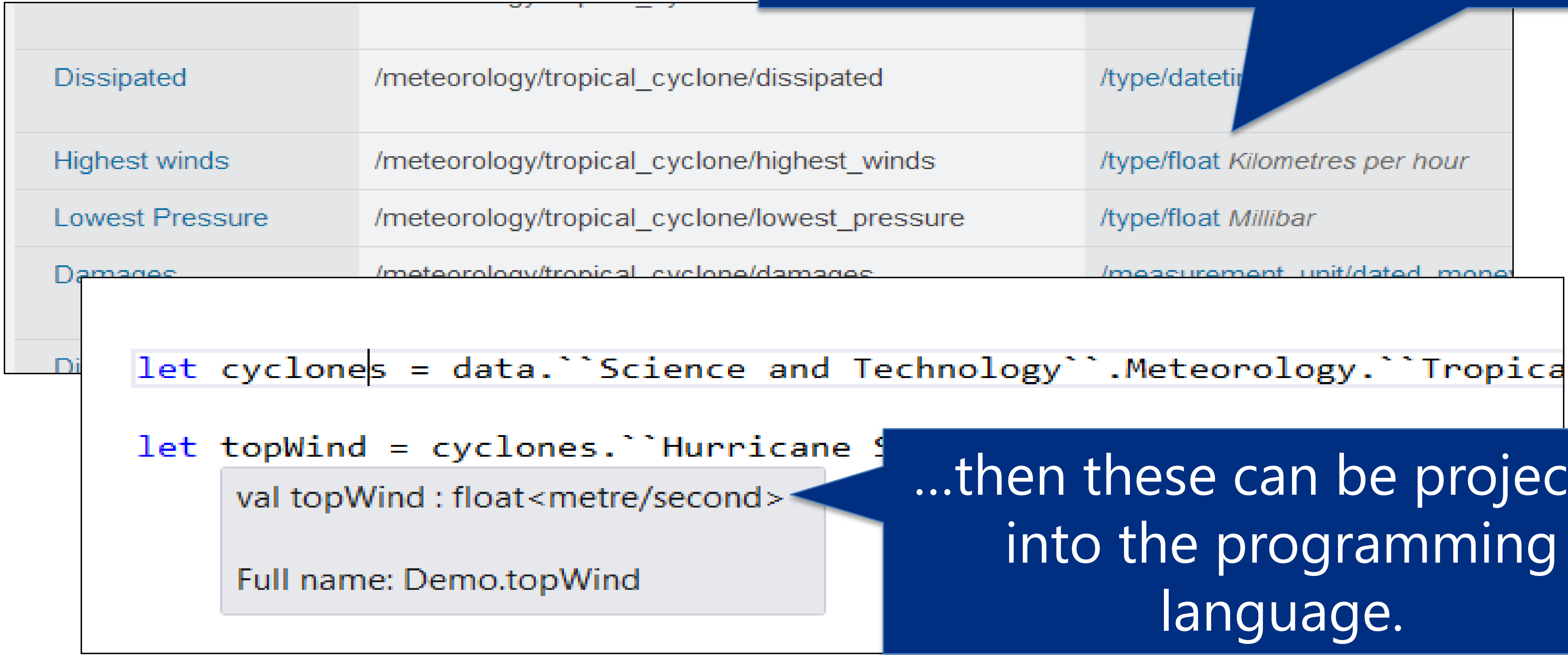
Data Scripters need to work with different granularities of schematization

...Only a language with massively scalable metadata integration can operate at all these levels

Every stable entity can get a unique type

Providing Units of Measure

via F#'s Units of Measure



Dissipated	/meteorology/tropical_cyclone/dissipated	/type/dateti
Highest winds	/meteorology/tropical_cyclone/highest_winds	/type/float *Kilometres per hour*
Lowest Pressure	/meteorology/tropical_cyclone/lowest_pressure	/type/float *Millibar*
Damages	/meteorology/tropical_cyclone/damages	/measurement_unit/dated_mone
Di		

```
let cyclones = data.``Science and Technology``.Meteorology.``Tropica

let topWind = cyclones.``Hurricane S
```

```
val topWind : float<metre/second>

Full name: Demo.topWind
```

If the metadata contains units (including currencies)…

…then these can be projected into the programming language.

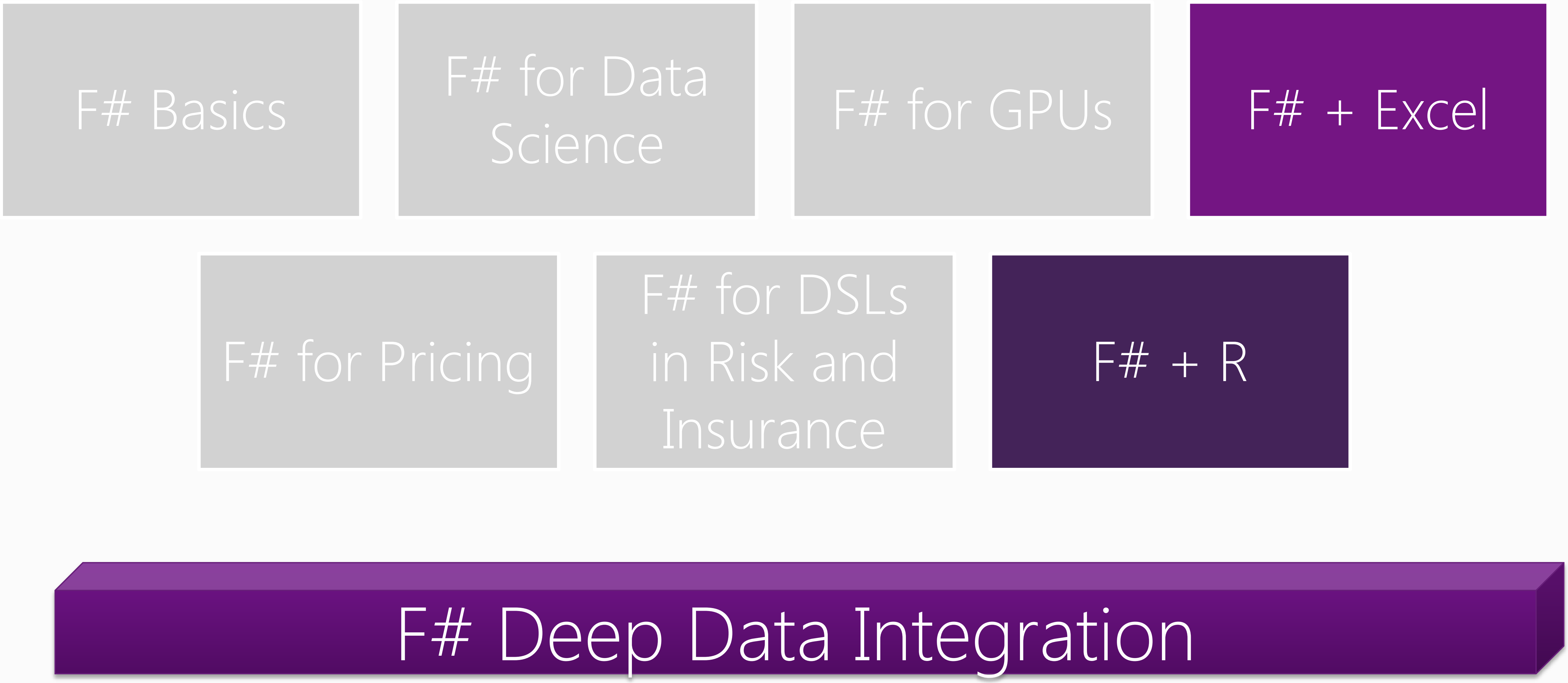
Theme #4

Reactive Streaming

Not covered today, but F# and .NET has excellent compositional primitives for reactive, streaming data, and provided data sources can easily make use of these

Integrating with Other Systems?

Typical F# Topics



F# Basics

F# for Data Science

F# for GPUs

F# + Excel

F# for Pricing

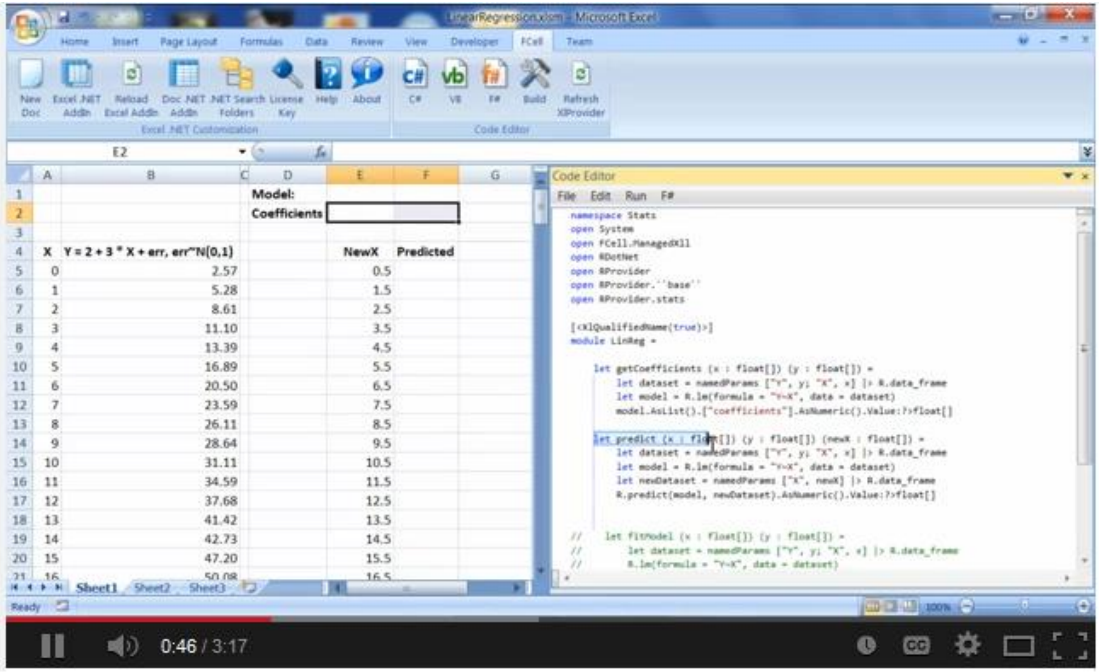
F# for DSLs in Risk and Insurance

F# + R

F# Deep Data Integration

Functional + R + Excel Integration

via fcell.io

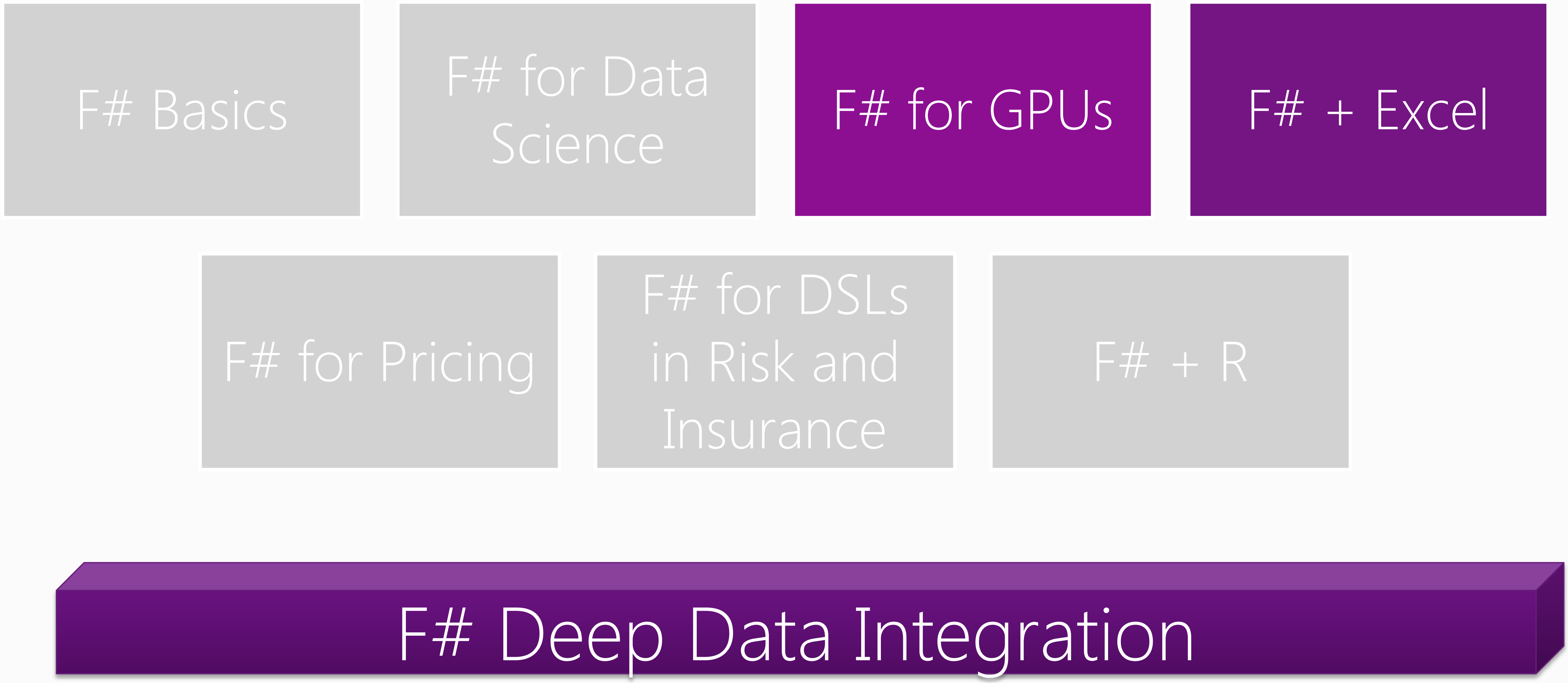


E2

	A	B	C	D	E	F	G
1				Model:			
2				Coefficients			
3							
4	X	Y = 2 + 3 * X + err, err~N(0,1)			NewX	Predicted	
5	0	2.57			0.5		
6	1	5.28			1.5		
7	2	8.61			2.5		
8	3	11.10			3.5		
9	4	13.39			4.5		
10	5	16.89			5.5		
11	6	20.50			6.5		
12	7	23.59			7.5		
13	8	26.11			8.5		
14	9	28.64			9.5		
15	10	31.11			10.5		
16	11	34.59			11.5		
17	12	37.68			12.5		
18	13	41.42			13.5		
19	14	42.73			14.5		
20	15	47.20			15.5		
21	16	50.08			16.5		

```
namespace Stats
open System
open FCell.ManagedXll
open RDotNet
open RProvider
open RProvider.``base``
open RProvider.stats

[<XlQualifiedName(true)>]
module LinReg =

    let getCoefficients (x : float[]) (y : float[]) =
        let dataset = namedParams ["Y", y; "X", x] |> R.data_frame
        let model = R.lm(formula = "Y~X", data = dataset)
        model.AsList().["coefficients"].AsNumeric().Value:?>float[]

    let predict (x : float[]) (y : float[]) (newX : float[]) =
        let dataset = namedParams ["Y", y; "X", x] |> R.data_frame
        let model = R.lm(formula = "Y~X", data = dataset)
        let newDataset = namedParams ["X", newX] |> R.data_frame
        R.predict(model, newDataset).AsNumeric().Value:?>float[]


//    let fitModel (x : float[]) (y : float[]) =
//        let dataset = namedParams ["Y", y; "X", x] |> R.data_frame
//        R.lm(formula = "Y~X", data = dataset)
```

Typical F# Topics



F# Basics

F# for Data Science

F# for GPUs

F# + Excel

F# for Pricing

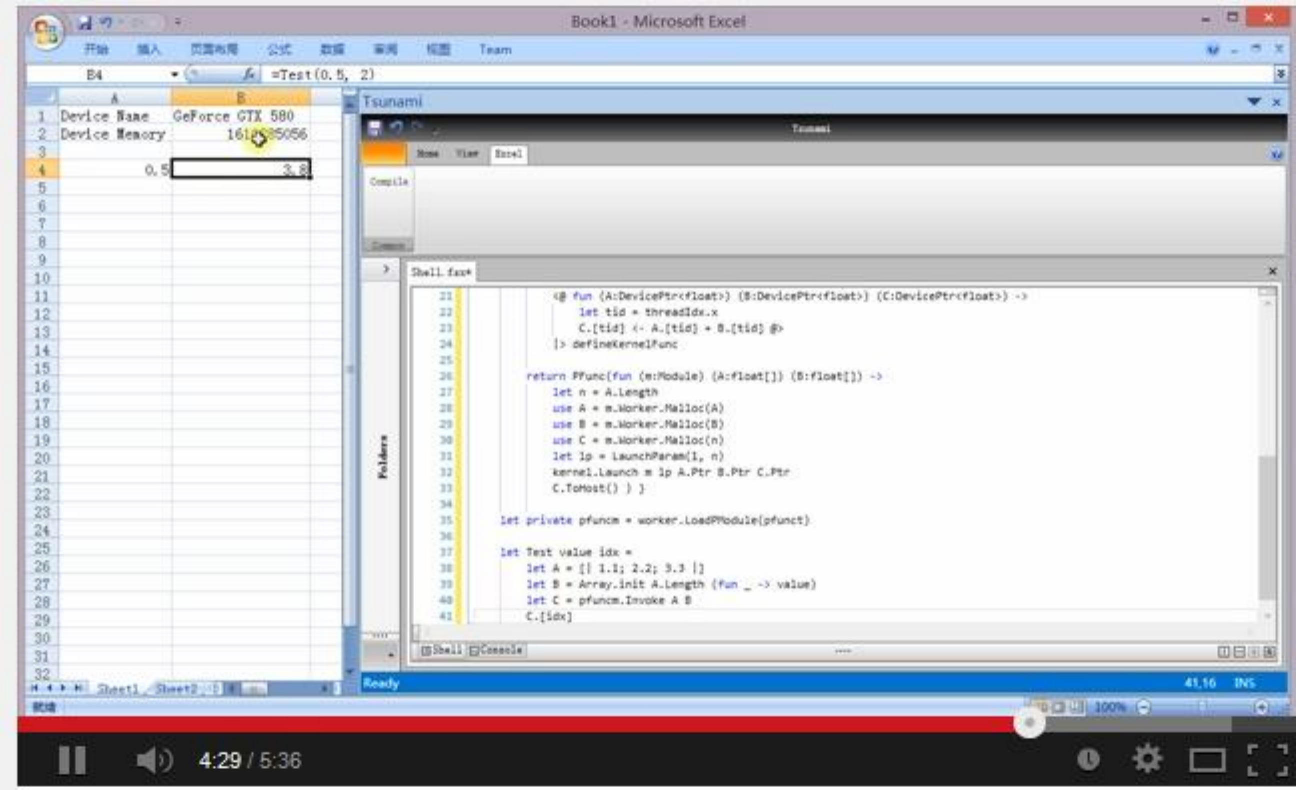
F# for DSLs in Risk and Insurance

F# + R

F# Deep Data Integration

Functional + GPGPU

F# + FCell + QuantAlea



B4 =Test(0.5, 2)

	A	B
1	Device Name	GeForce GTX 580
2	Device Memory	161[illegible]5056
3		
4	0.5	3.8

```
            <@ fun (A:DevicePtr<float>) (B:DevicePtr<float>) (C:DevicePtr<float>) ->
                let tid = threadIdx.x
                C.[tid] <- A.[tid] + B.[tid] @>
            |> defineKernelFunc

        return PFunc(fun (m:Module) (A:float[]) (B:float[]) ->
            let n = A.Length
            use A = m.Worker.Malloc(A)
            use B = m.Worker.Malloc(B)
            use C = m.Worker.Malloc(n)
            let lp = LaunchParam(1, n)
            kernel.Launch m lp A.Ptr B.Ptr C.Ptr
            C.ToHost() ) }

    let private pfuncm = worker.LoadPModule(pfunct)

    let Test value idx =
        let A = [| 1.1; 2.2; 3.3 |]
        let B = Array.init A.Length (fun _ -> value)
        let C = pfuncm.Invoke A B
        C.[idx]
```

Summary

Financial programming is ever more integrated with data

Integrating data+programming has many challenges

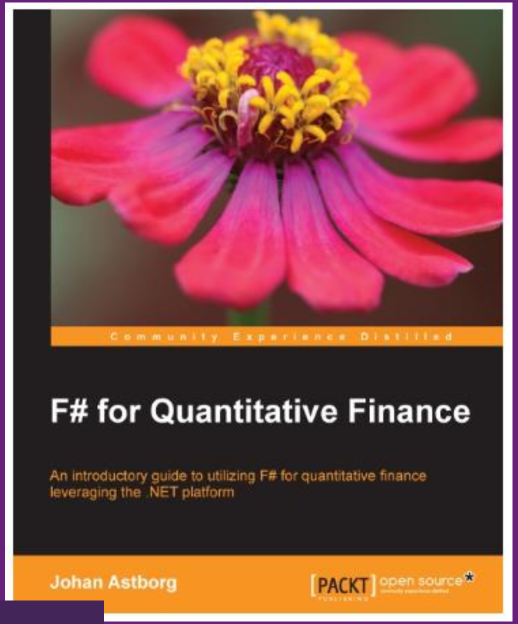
I presented techniques for scalable, intuitive integration of financial data+metadata

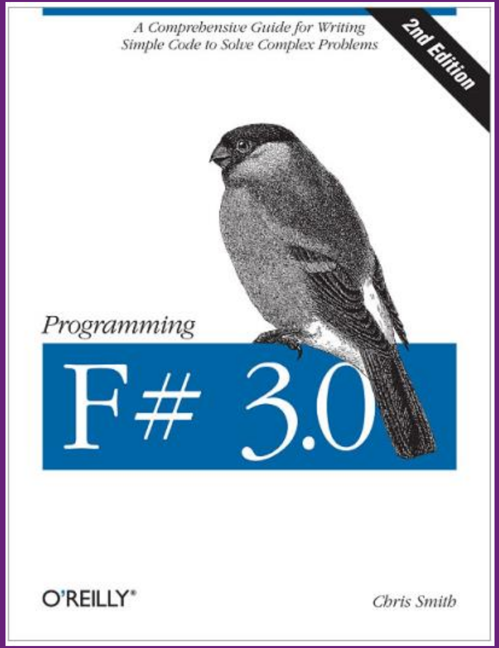
You can use these techniques for real and in production through F#

To find out more…

- Learn F# at tryfsharp.org (including financial)
- Lots of resources at fsharp.org
- Testimonials at fsharp.org/testimonials
- Over 100 videos at fsharp.org/videos

Questions?


Community Experience Distilled
F# for Quantitative Finance
An introductory guide to utilizing F# for quantitative finance leveraging the .NET platform
Johan Astborg
PACKT open source


A Comprehensive Guide for Writing Simple Code to Solve Complex Problems
2nd Edition
Programming
F# 3.0
O'REILLY
Chris Smith

tryfsharp.org

Microsoft