Introduction to F#

Scalable, Type-safe, Succinct, Interoperable, Mathematically-oriented Programming for .NET

Don Syme
MSR Cambridge

http://research.microsoft.com/fsharp

Today…

→ F# overview

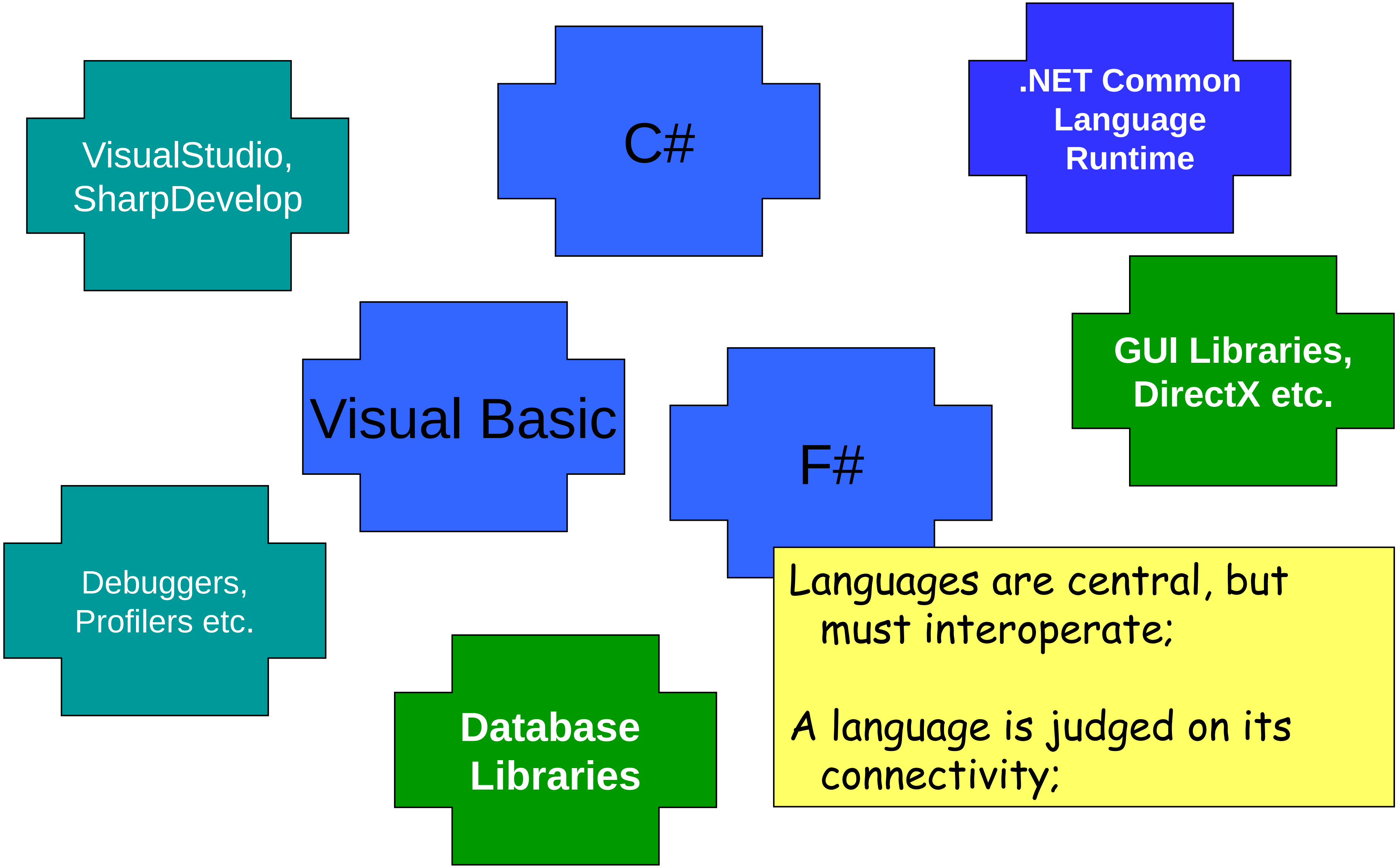
→ Some introductory F# programming

→ Two case studies

Which functional language:

- Connects with all Microsoft and most Open Source foundation technologies?
- Has 100s of Microsoft and open source developers working on its runtime systems, JIT compilers and libraries?
- Has concurrent GC and SMP support?
- Has CPU profilers, memory profilers, debuggers, test, doc tools?
- Lets you publish types and code accessible by 100,000s of developers?
- Consists of only ~25K LOC

F# = ML in the world of .NET



VisualStudio, SharpDevelop

C#

.NET Common Language Runtime

Visual Basic

F#

GUI Libraries, DirectX etc.

Debuggers, Profilers etc.

Database Libraries

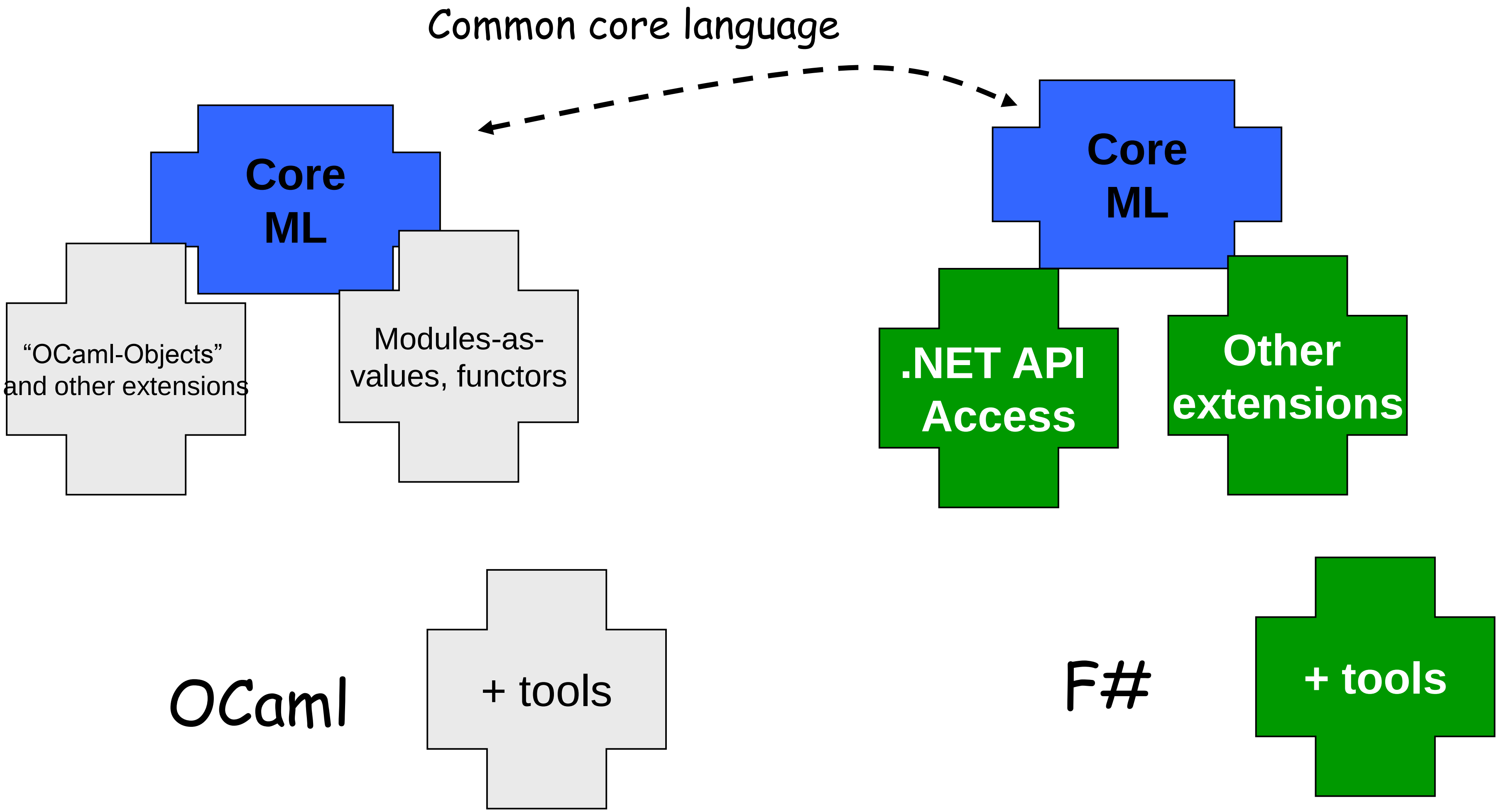
Languages are central, but must interoperate;

A language is judged on its connectivity;

Introducing F#...

- A .NET language
- Aims to combine much of Lisp, ML, Scheme and Haskell in the context of .NET
- Functional, math-oriented, scalable
- Aimed particularly at the "Symbolic Scripting and Programming" niche at Microsoft
  - e.g. Static Driver Verifier, Terminator, Machine Learning, Vision and more

F# as a Language



Common core language

Core ML

“OCaml-Objects” and other extensions

Modules-as-values, functors

OCaml

+ tools

Core ML

.NET API Access

Other extensions

F#

+ tools

Some Simple F#

NOTE: type inferred

→ `let data = (1,2,3)`

```
val data: int * int * int
```

→ `let sqr x = x * x`

```
val sqr: int -> int
```

pattern matching

→ `let f (x,y,z) = (sqr x, sqr y, sqr z)`

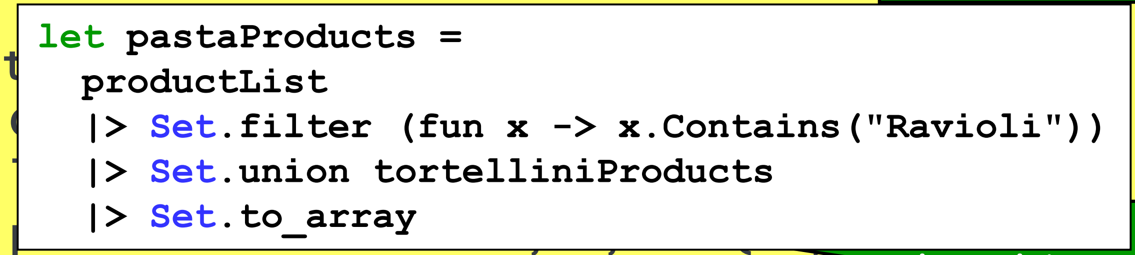
parentheses optional on application

→ `let sx,sy,sz = f (10,20,30)`

→ `printf "hello world"; 1+2`

sequencing

→
```
let
   (
   l
   |
   sqr
```



```
let pastaProducts =
  productList
  |> Set.filter (fun x -> x.Contains("Ravioli"))
  |> Set.union tortelliniProducts
  |> Set.to_array
```

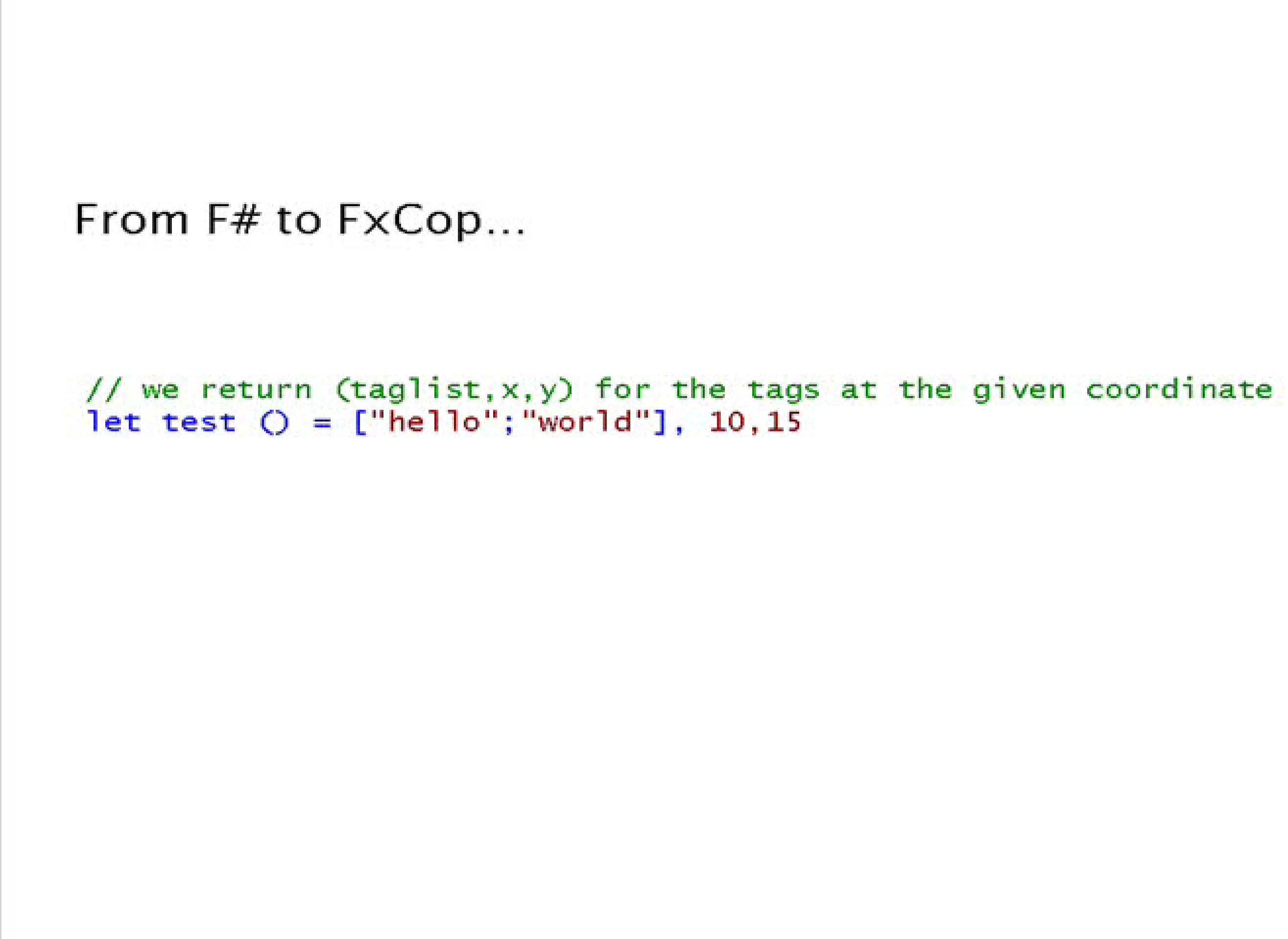
local binding, sequencing

→ `let (|>) x f = f x`

pipelining operator

Some Sample F# Programming

Video...



From F# to FxCop...

```
// we return (taglist,x,y) for the tags at the given coordinate
let test () = ["hello";"world"], 10,15
```

Orthogonal & Unified Constructs

100s of "delegate" types in .NET platform effectively unified away

- Functions: unified and simple

```
(fun x -> x + 1)

let f x y = x * y
let g x y = x + y

let p = (f,g)
```

```
predicate = 'a -> bool
```

```
send = 'a -> unit
```

```
threadStart = unit -> unit
```

```
comparer = 'a -> 'a -> int
```

```
hasher = 'a -> int
```

```
equality = 'a -> 'a -> bool
```

Effective abstractions

- Type parameters

```
Map<‘a,’b>
List<‘a>
Set<‘a>
```

- Discriminated unions

```
type expr =
   | Sum of expr * expr
   | Prod of expr * expr
….
```

- Pattern matching

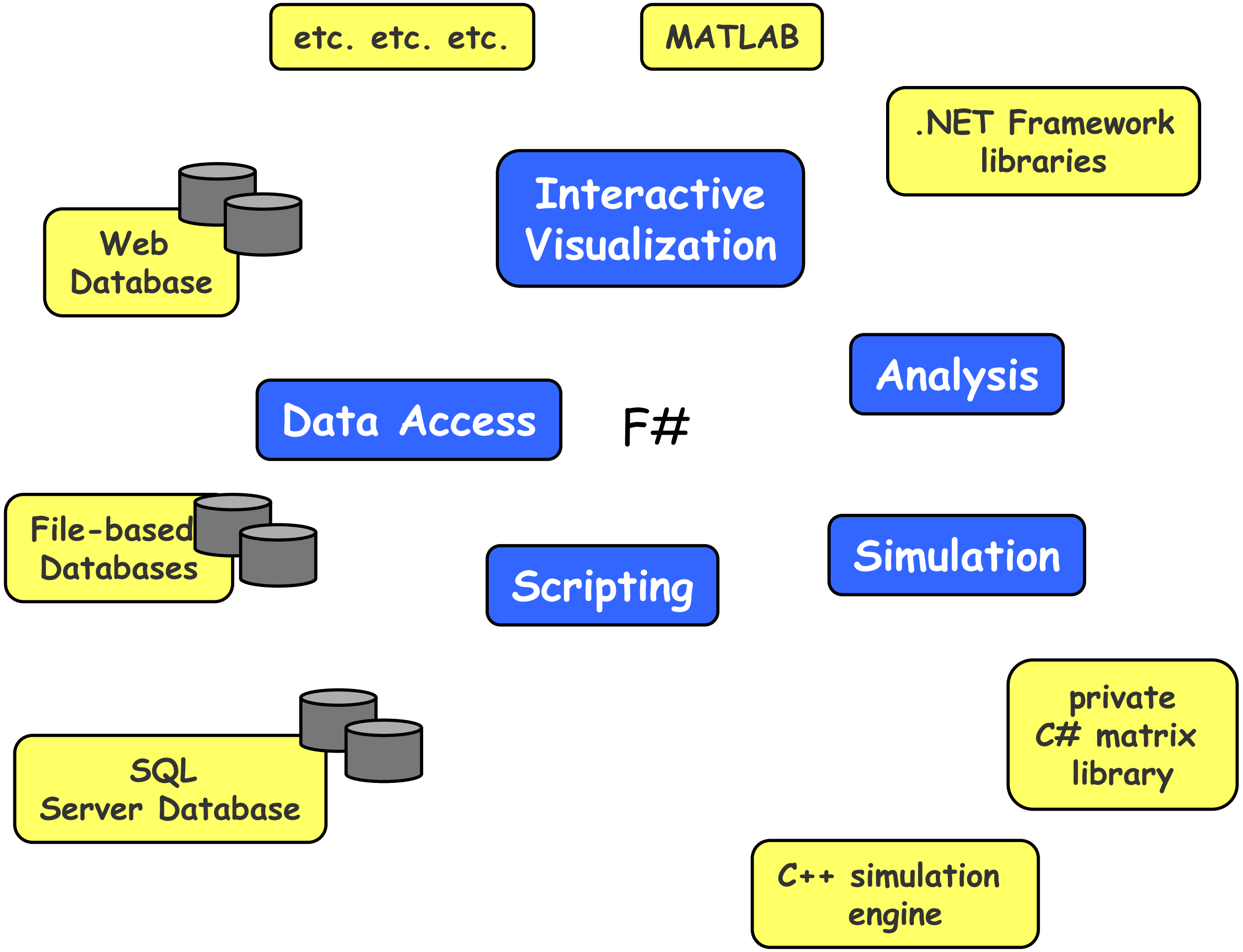
- Type inference

```
match expr with
   | Sum(a,b) -> ...
   | Prod(a,b) -> ... ….
```

- Recursion (Mutually-referential objects)

```
let rec map = ...
```

Typical F# Project Architecture



Base Tools

✓ Windows (any edition)

✓ .NET Framework 2.0

✓ F# 1.1.11

Readily Accessible Extras

✓ Visual Studio 2005

✓ SQL Server 2005

✓ Alchemi (.NET distribution framework)

✓ also many, many others e.g. Visual C++, DirectX, dnAnalytics, MKL, LAPACK, MATLAB, AJAX libraries etc. etc. etc.

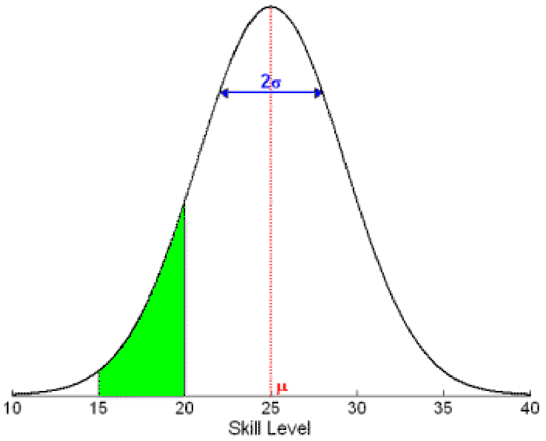
Case Study: TrueSkill

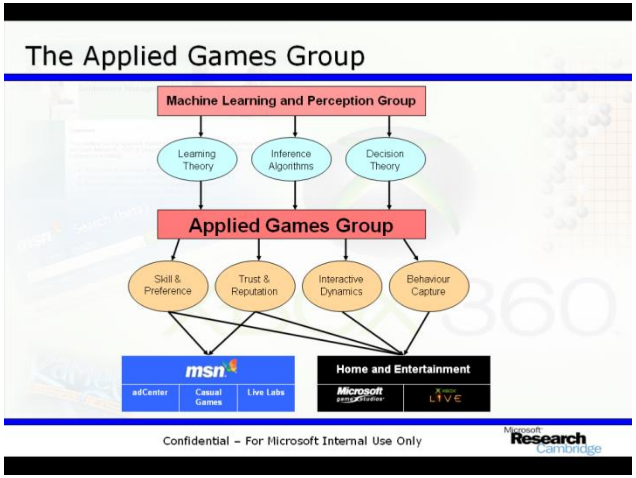
Live game ranking algorithms in F#

TrueSkill ™

→ Skill based ranking for Xbox Live! from MSR.

- □ Skill is a normal distribution
- □ Mean is the "expected skill"
- □ Variance is the "uncertainty"





The Applied Games Group

Machine Learning and Perception Group

Learning Theory

Inference Algorithms

Decision Theory

Applied Games Group

Skill & Preference

Trust & Reputation

Interactive Dynamics

Behaviour Capture

msn

adCenter

Casual Games

Live Labs

Home and Entertainment

Microsoft game studios

Xbox LIVE

Confidential – For Microsoft Internal Use Only

Microsoft Research Cambridge

F# as a Scripting Language

- **Problem:**
  - Parsing 110 GB of Xbox matchmaking log data (12 days).
  - Data spreads over 11,000 text files in over 300 directories.
- **Task:**
  - Importing data in structural form into a SQL database.
- **Code:**
  - 90 lines long!
- **Development time (code):**
  - 1 – 2 hours.
- **Performance:**
  - In under 18 hours = 10,000 log lines processed per second!

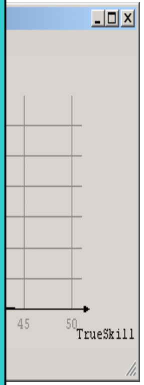
F# for Large Scale Data Analysis

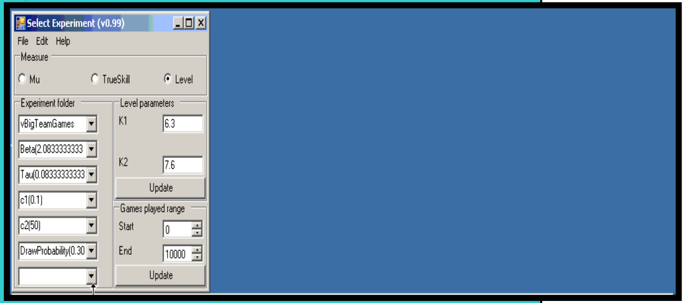
- **Problem:**
  - Analysis of 4.2 million Xbox user feedbacks (4 months worth of data).
  - Data is already in a SQL database.
- **Task:**
  - Adopt TrueSkill™ model to the user feedback problem for integration into the Xbox service.
- **Code:**
  - 100 lines long!
- **Development time (code):**
  - 3 – 4 hours.
- **Performance:**
  - 10 minutes runtime for the whole dataset!

F# for Complex Scientific Modelling

Why F#?

- Deep .Net Integration.
  - System.IO
  - System.Data.SqlClient
  - Custom TrueSkill™ C# libraries.
  - Custom C# Matrix library.
- Interactive development.
- Full Visual Studio integration.
- Built-in type inference.
- Anonymous functions.
- Pattern matching.



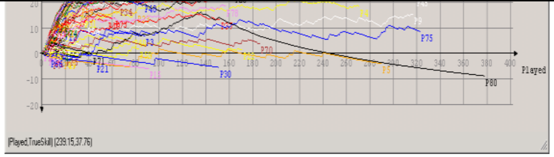


Measure: ○ Mu ○ TrueSkill ● Level

Experiment folder:
vBigTeamGames
Beta(2.0833333333
Tau(0.08333333333
c1(0.1)
c2(50)
DrawProbability(0.30

Level parameters:
K1 6.3
K2 7.6

Games played range:
Start 0
End 10000



What they say…

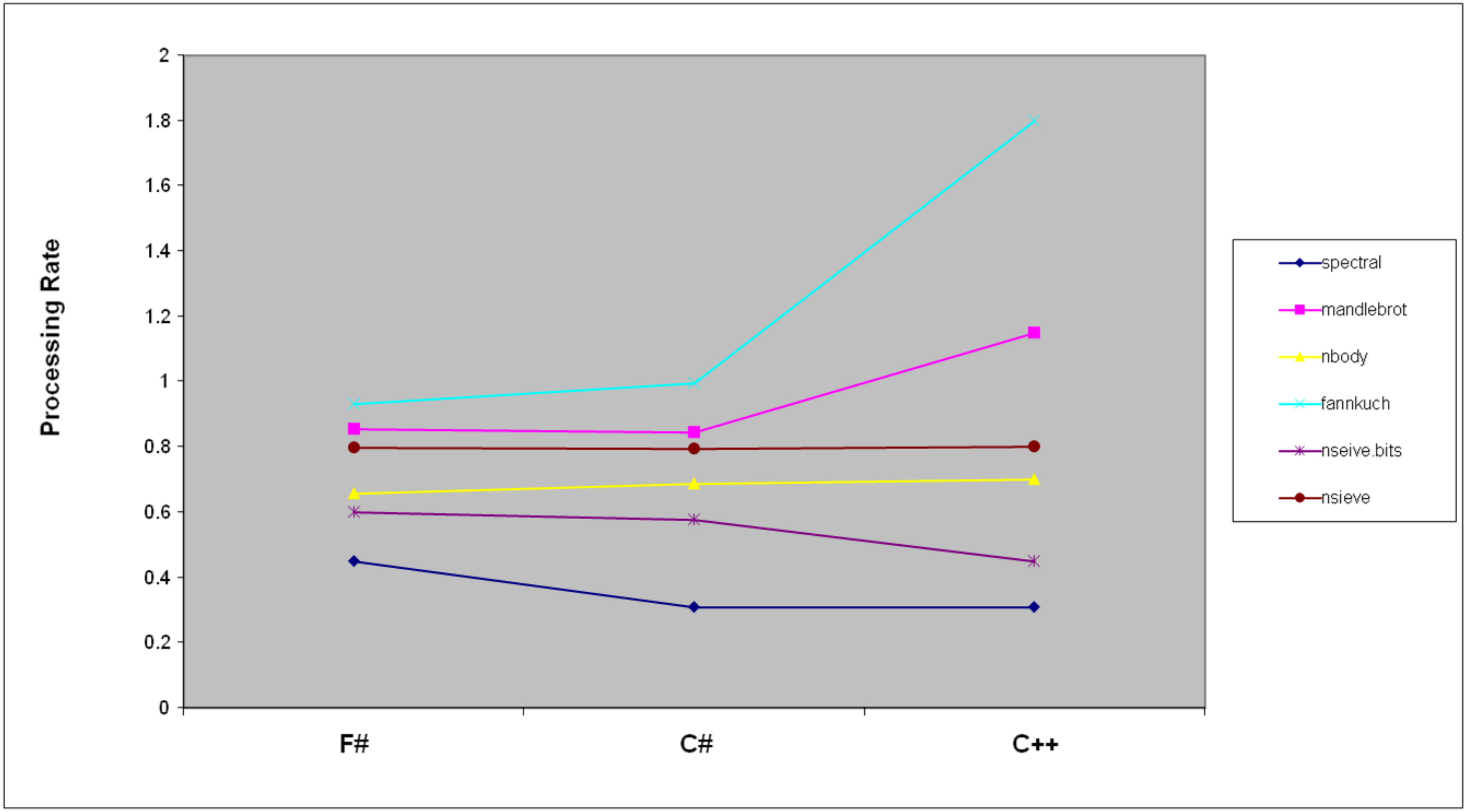
- New F# user (experienced OO programmer)

"We crunched **400Gb of data**, scripting over smaller sets then finally running 3 days **with 15 computers**. The UI code streams a **100Mb CSV file from disk in around 5 seconds**, while **simultaneously calculating** and **drawing** a histogram."

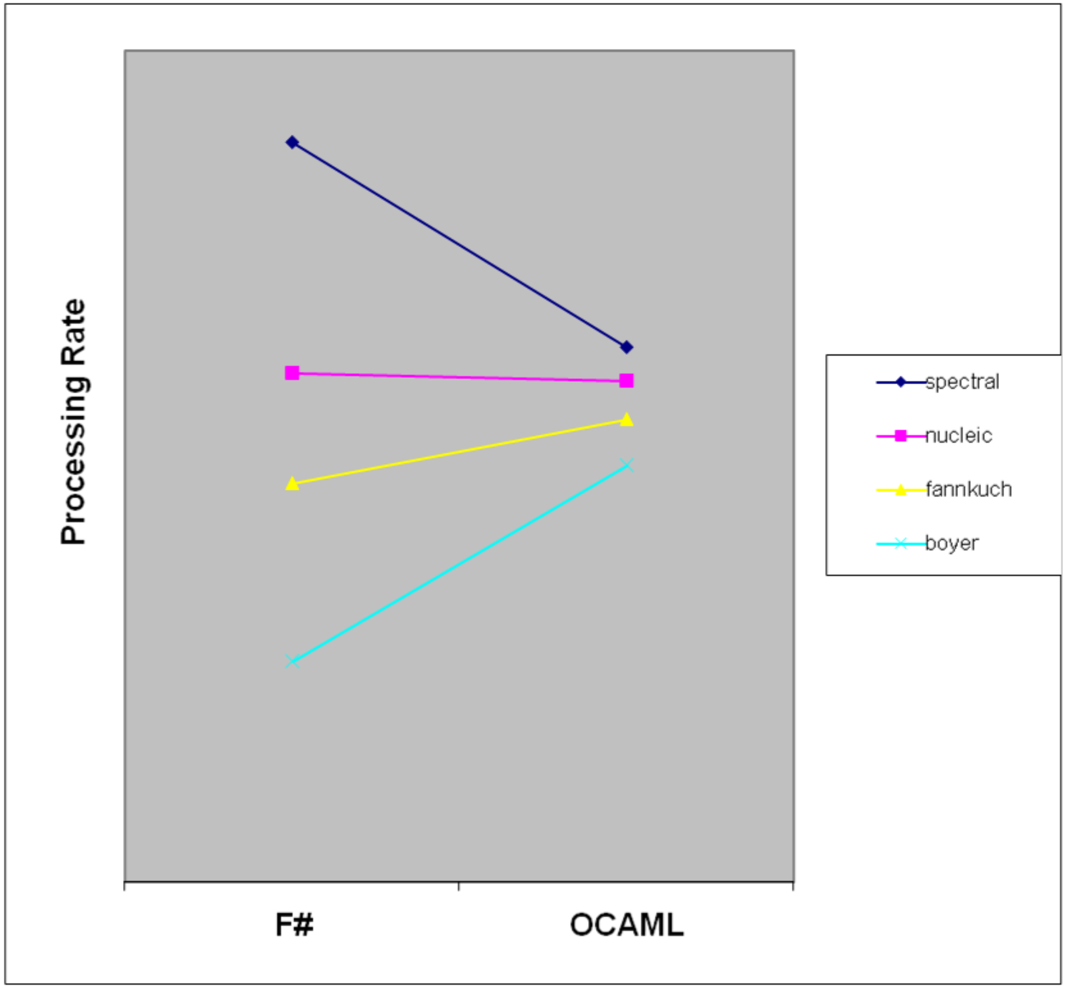
- "The F# code has **excellent performance**."
- "F# is fun!"
- "I really enjoyed the brevity of the resulting code. **It helped me focus on the ends, not the means.**"
- "The F# code was **easy to maintain and evolve**"

Performance and related issues

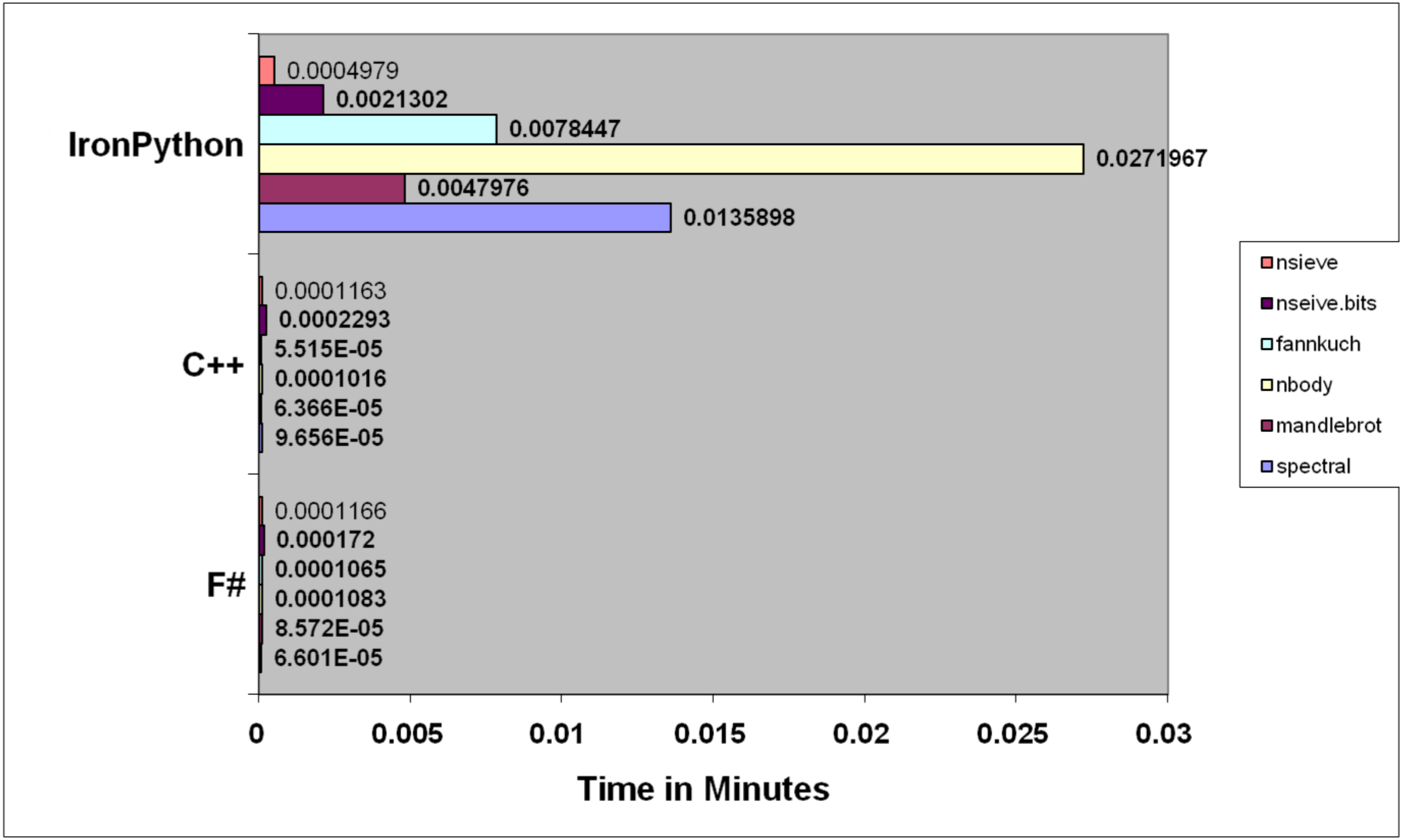
Benchmark Performance by Language



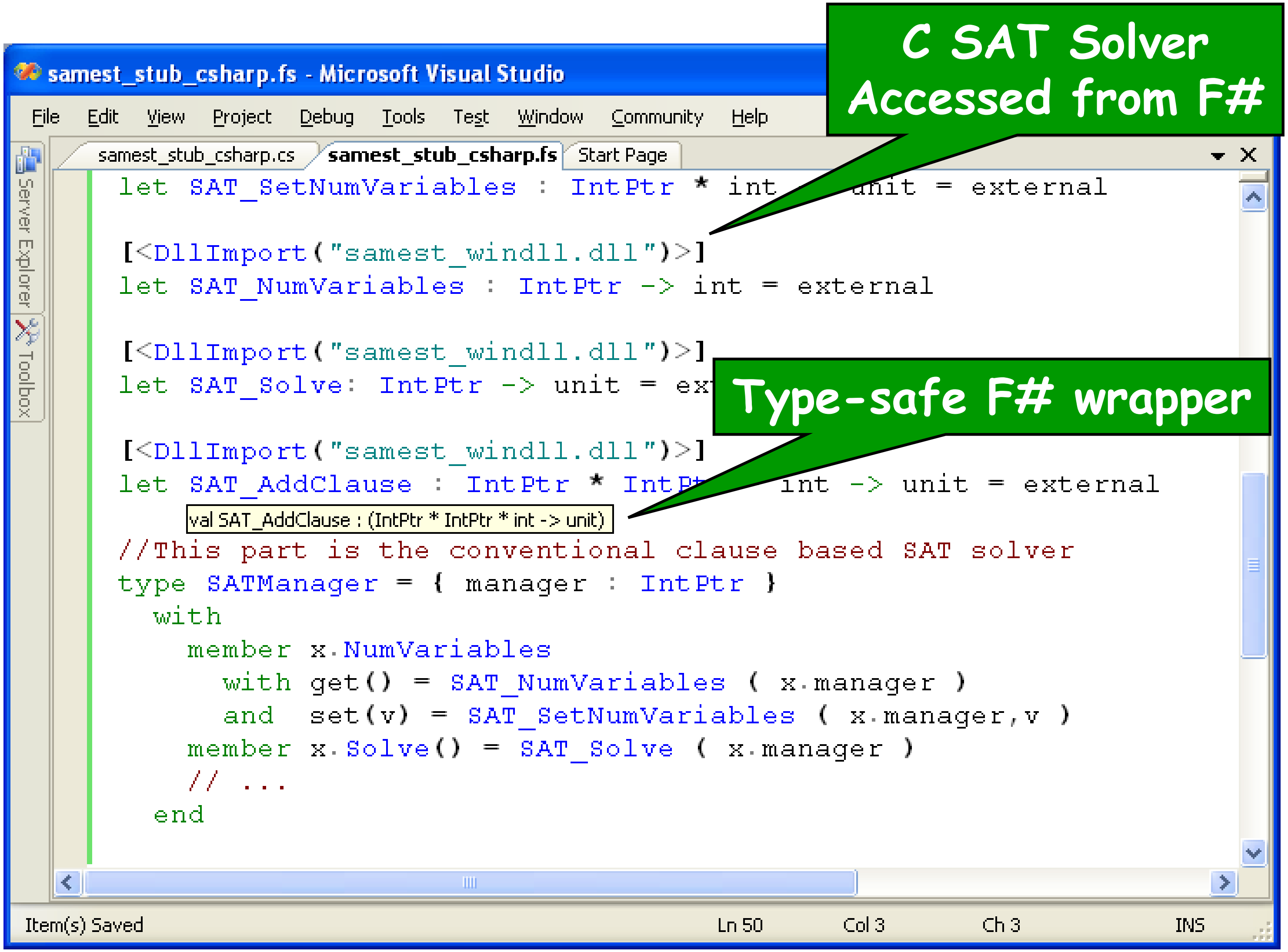
F# and OCAML benchmarks



Running times for benchmarks



Calling C/C++



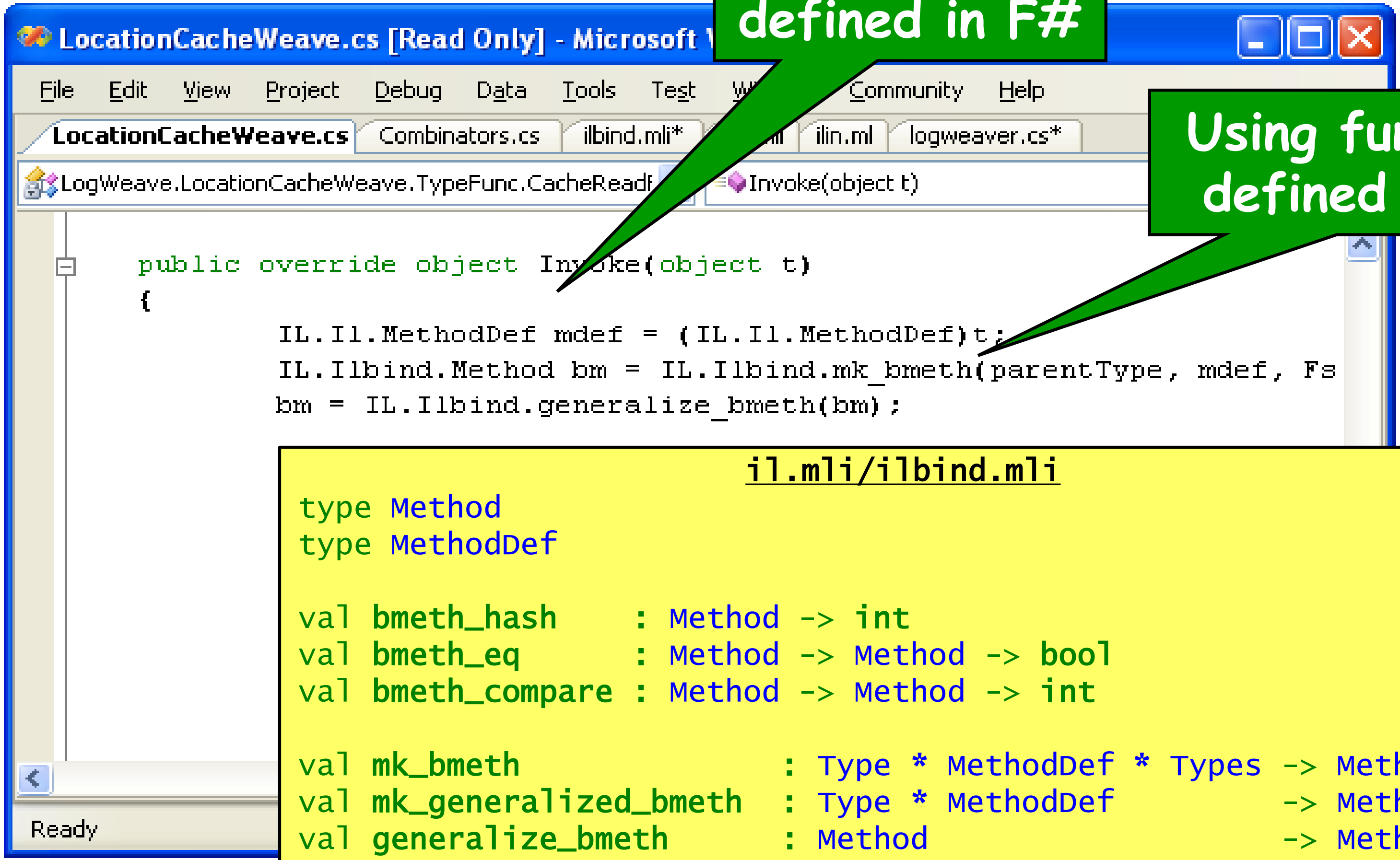
```
let SAT_SetNumVariables : IntPtr * int        unit = external

[<DllImport("samest_windll.dll")>]
let SAT_NumVariables : IntPtr -> int = external

[<DllImport("samest_windll.dll")>]
let SAT_Solve: IntPtr -> unit = ex

[<DllImport("samest_windll.dll")>]
let SAT_AddClause : IntPtr * IntPt       int -> unit = external
      val SAT_AddClause : (IntPtr * IntPtr * int -> unit)
//This part is the conventional clause based SAT solver
type SATManager = { manager : IntPtr }
  with
    member x.NumVariables
      with get() = SAT_NumVariables ( x.manager )
      and  set(v) = SAT_SetNumVariables ( x.manager,v )
    member x.Solve() = SAT_Solve ( x.manager )
    // ...
  end
```

C SAT Solver Accessed from F#

Type-safe F# wrapper

#2: Calling F# from C#

- LogWeave (Office XAF Optimization Tool)
- 4000 lines C#, using brary


LocationCacheWeave.cs [Read Only] - Microsoft V
File Edit View Project Debug Data Tools Test W Community Help
LocationCacheWeave.cs Combinators.cs ilbind.mli* ml ilin.ml logweaver.cs*
LogWeave.LocationCacheWeave.TypeFunc.CacheReadt
Invoke(object t)
Ready

```
public override object Invoke(object t)
{
        IL.Il.MethodDef mdef = (IL.Il.MethodDef)t;
        IL.Ilbind.Method bm = IL.Ilbind.mk_bmeth(parentType, mdef, Fs
        bm = IL.Ilbind.generalize_bmeth(bm);
```

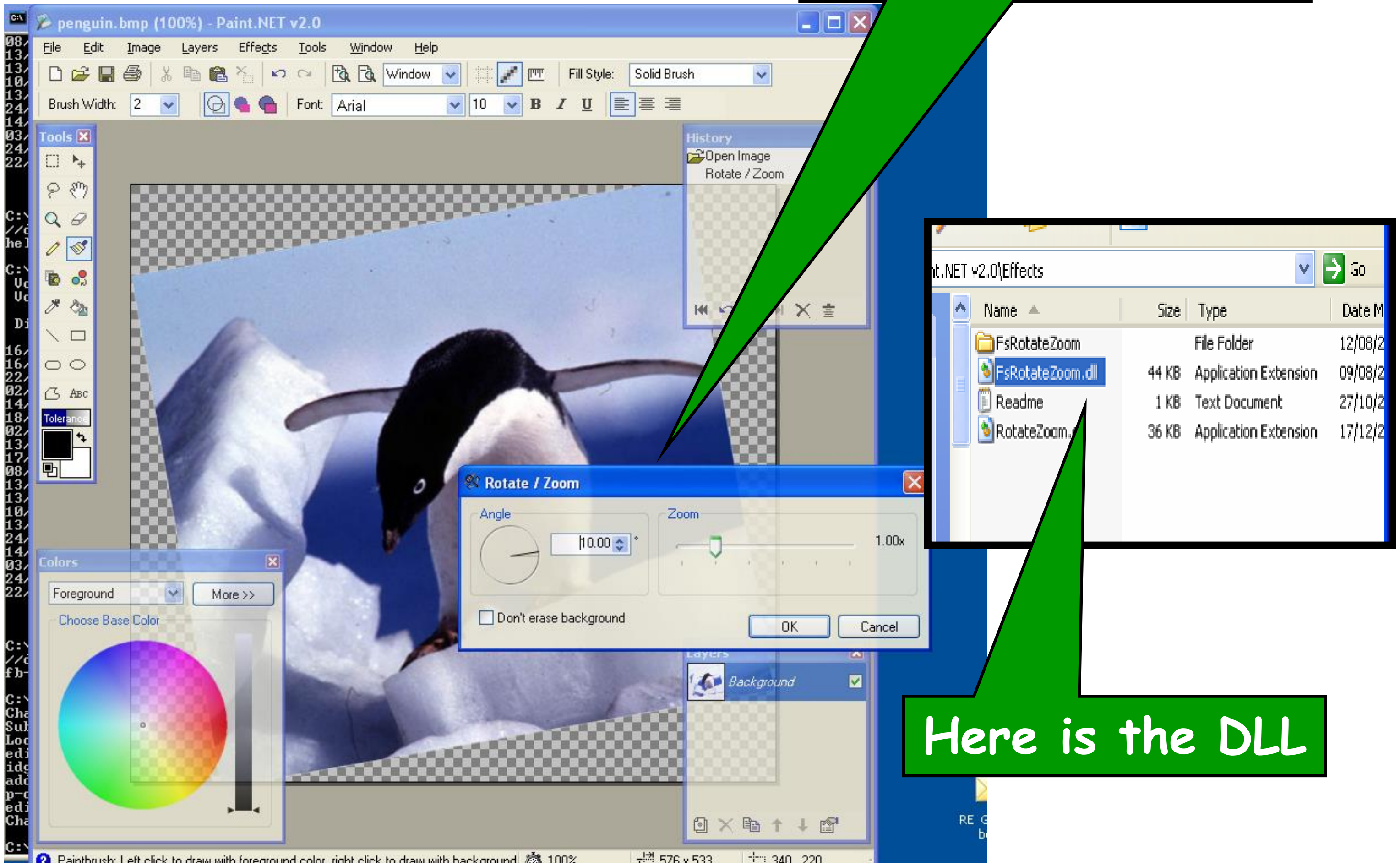
Using types defined in F#

Using functions defined in F#

il.mli/ilbind.mli

```
type Method
type MethodDef

val bmeth_hash    : Method -> int
val bmeth_eq      : Method -> Method -> bool
val bmeth_compare : Method -> Method -> int

val mk_bmeth             : Type * MethodDef * Types -> Method
val mk_generalized_bmeth : Type * MethodDef         -> Method
val generalize_bmeth     : Method                   -> Method
```

#3: Paint.NET & Plugins



Fill Style: Solid Brush

Brush Width: 2

Font: Arial 10

Foreground

Rotate / Zoom

Angle: 10.00 °

Zoom: 1.00x

□ Don't erase background

Paintbrush: Left click to draw with foreground color, right click to draw with background | 100% | 576 x 533 | 340, 220

nt.NET v2.0\Effects

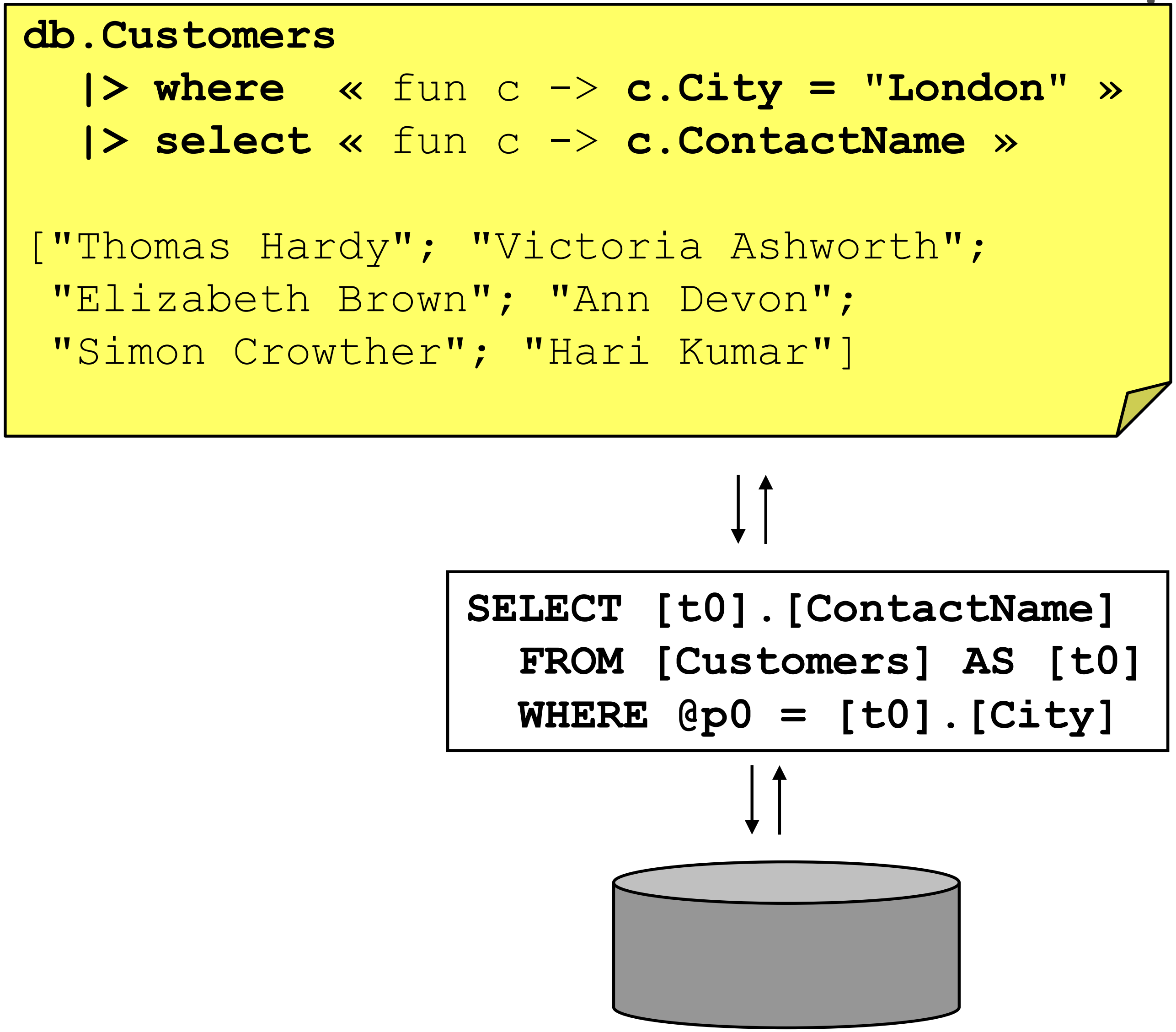
Name	Size	Type	Date M
FsRotateZoom		File Folder	12/08/2
FsRotateZoom.dll	44 KB	Application Extension	09/08/2
Readme	1 KB	Text Document	27/10/2
RotateZoom.	36 KB	Application Extension	17/12/2

Plugin written in F#

Here is the DLL

F# and LINQ

Language Integrated Queries with F#/LINQ



```
db.Customers
   |> where  « fun c -> c.City = "London" »
   |> select « fun c -> c.ContactName »

["Thomas Hardy"; "Victoria Ashworth";
 "Elizabeth Brown"; "Ann Devon";
 "Simon Crowther"; "Hari Kumar"]
```

```
SELECT [t0].[ContactName]
  FROM [Customers] AS [t0]
  WHERE @p0 = [t0].[City]
```

The Vision: Heterogeneous Execution

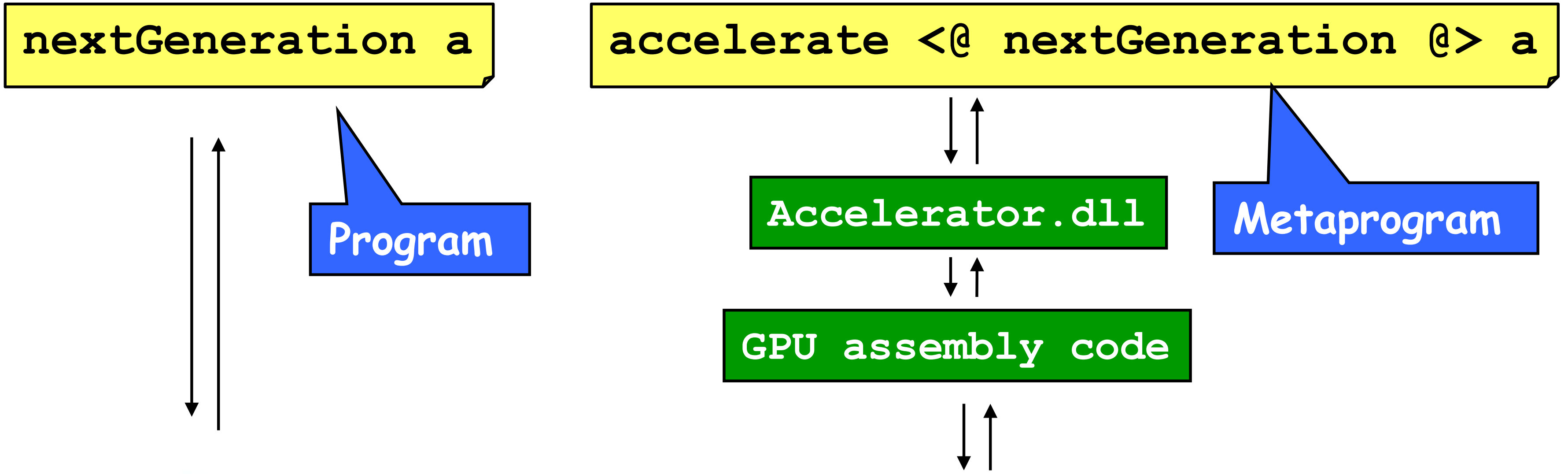
- Today languages use homogeneous execution:
  - The CPU

- The natural extension of the LINQ vision is heterogeneous execution, leveraging
  - The database
  - The server
  - The GPU
  - The web browser (ala AJAX)
  - Symbolic execution

- Write your code in one language, execute it in many completely different ways

Accelerate ahead with F# Quotations!

```
let nextGeneration(a) =
  let sum =  rot a (-1) (-1) .+ rot a (-1) 0 .+ rot a (-1) 1
          .+ rot a    0  (-1) .+                 rot a    0  1
          .+ rot a    1  (-1) .+ rot a    1  0 .+ rot a    1  1  in
  (sum .= three) .|| ((sum .= two) .&& a);;
```



`nextGeneration a`

Program

`accelerate <@ nextGeneration @> a`

Metaprogram

`Accelerator.dll`

`GPU assembly code`

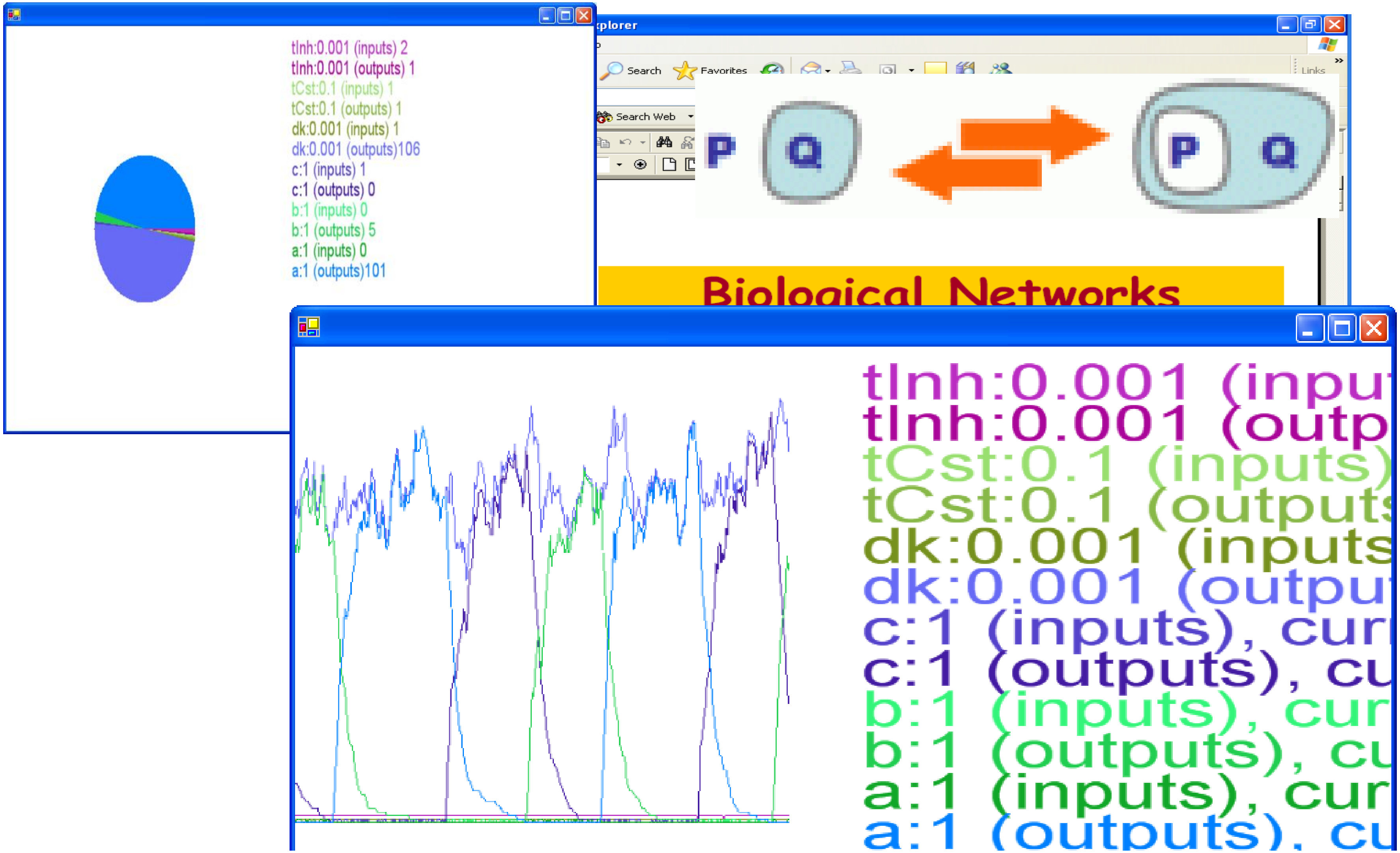
CPU

Graphics Card

Case Study: SPiM

Interactive Chemical/Biological Stochastic Simulation with F#

#3: SPiM: Biological Simulation and Visualization



tInh:0.001 (inputs) 2
tInh:0.001 (outputs) 1
tCst:0.1 (inputs) 1
tCst:0.1 (outputs) 1
dk:0.001 (inputs) 1
dk:0.001 (outputs)106
c:1 (inputs) 1
c:1 (outputs) 0
b:1 (inputs) 0
b:1 (outputs) 5
a:1 (inputs) 0
a:1 (outputs)101

Biological Networks

tInh:0.001 (inpu
tInh:0.001 (outp
tCst:0.1 (inputs)
tCst:0.1 (outputs
dk:0.001 (inputs
dk:0.001 (outpu
c:1 (inputs), curr
c:1 (outputs), cu
b:1 (inputs), cur
b:1 (outputs), cu
a:1 (inputs), cur
a:1 (outputs), cu

Summary

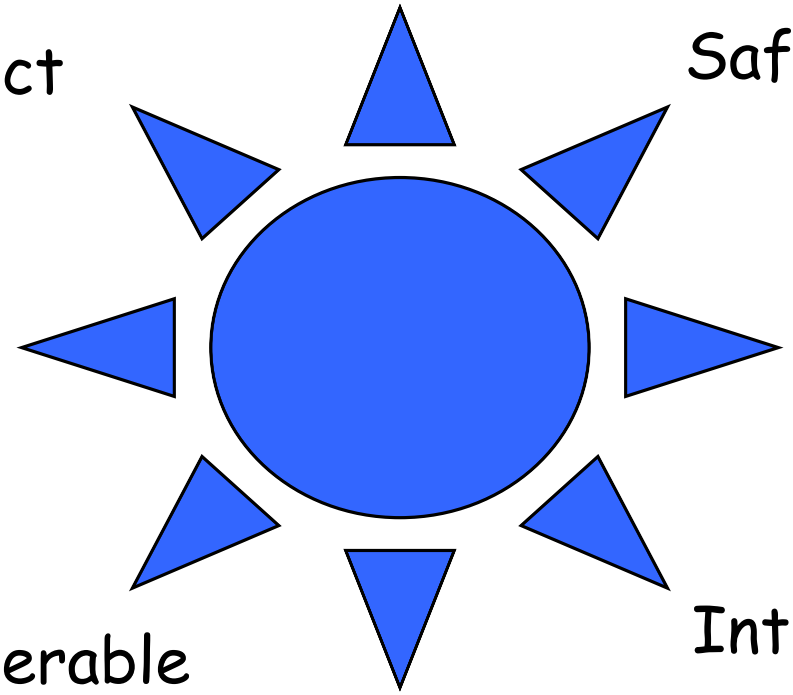
Challenges of modern language design

Statically typed, hence
Scalable (> 100K LOC)

Succinct

Safe

It's the combination that counts



Efficient

Libraries

Interoperable

Interactive

Good
Platform

Summary

- .NET 2.0 + F# 1.1
  - An excellent combination for practical scientific programming
  - .NET gives you a rich software eco-system of relevant, polished components
  - F# gives you **scripting**, **performance** and the **beauty** and **productivity** of **functional programming**
- Enjoy!

Questions?