F#

Succinct, Expressive, Efficient Functional Programming for .NET

The F# Team

Microsoft Developer Division, Redmond

Microsoft Research, Cambridge

Topics

- What is F# about?
- Some Simple F# Programming
- A Taste of Parallel/Reactive with F#

What is F# about?

Or: *Why is Microsoft investing in functional programming anyway?*

Simplicity

Economics

Fun! Fun! Fun!

Simplicity

Code!

```
//F#
open System
let a = 2
Console.WriteLine a
```

```
//C#
using System;

namespace ConsoleApplication1
{
  class Program
  {
    static int a()
    {
        return 2;
    }
    static void Main(string[] args)
    {
        Console.WriteLine(a);
    }
  }
}
```

Code!

```
//F#
open System
let a = 2
Console.WriteLine a
```

```
//C#
using System;

namespace ConsoleApplication1
{
  class Program
  {
    static int a()
    {
        return 2;
    }
    static void Main(string[] args)
    {
        Console.WriteLine(a);
    }
  }
}
```

More Noise Than Signal!

Pleasure

```
type Command = Command of (Rover -> unit)
let BreakCommand     = Command(fun rover -> rover.Accelerate(-1.0))
let TurnLeftCommand  = Command(fun rover -> rover.Rotate(-5.0<degs>))
```

Pain

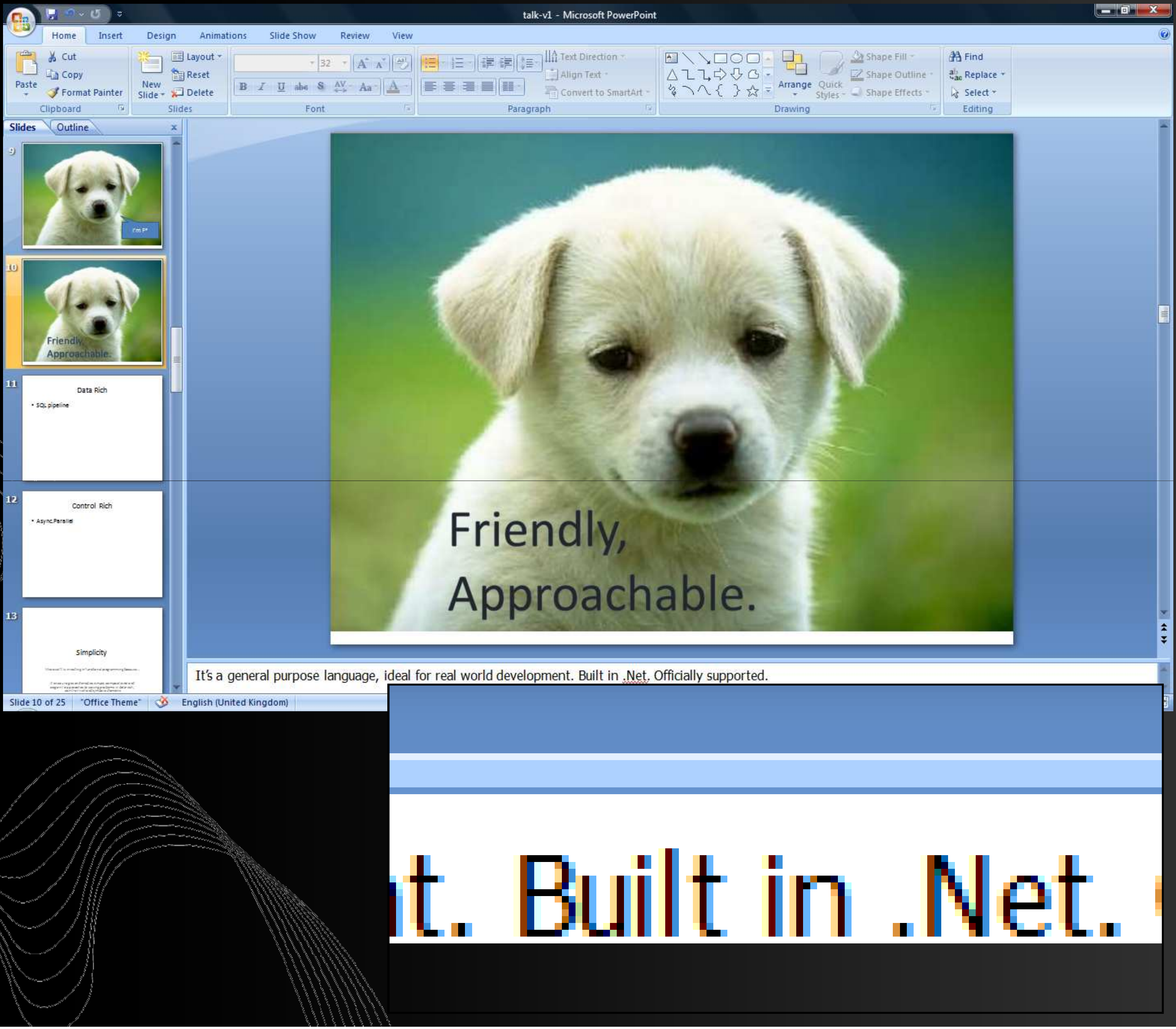
```
  abstract class Command
   {
       public virtual void Execute();
   }
   abstract class MarsRoverCommand : Command
   {
       protected MarsRover Rover { get; private
     set; }

       public MarsRoverCommand(MarsRover rover)
       {
           this.Rover = rover;
       }
   }
   class BreakCommand : MarsRoverCommand
   {
       public BreakCommand(MarsRover rover)
           : base(rover)
       {
       }
       public override void Execute()
       {
           Rover.Rotate(-5.0);
       }
   }
class TurnLeftCommand : MarsRoverCommand
   {
       public TurnLeftCommand(MarsRover rover)
           : base(rover)
       {
       }
       public override void Execute()
       {
           Rover.Rotate(-5.0);
       }
   }
```

Pleasure

```
let rotate(x,y,z) = (z,x,y)

let reduce f (x,y,z) = f x + f y + f z
```

Pain

```
Tuple<V,T,U> Rotate(Tuple<T,U,V> t)
{
    return new Tuple<V,T,U>(t.Item3,t.Item1,t.Item2);
}

int Reduce(Func<T,int> f,Tuple<T,T,T> t)
{
    return f(t.Item1) + f(t.Item2) + f (t.Item3);
}
```



Friendly, Approachable.

It's a general purpose language, ideal for real world development. Built in .Net. Officially supported.

t. Built in .Net.

Economics

Fun! Fun! Fun!

F#: Combining Paradigms

Functional

- Strong Typing
- Type Inference
- Data Types and Patterns
- 1st Class Functions
- Meta-Programming
- Workflows and Agents

Objects

- .NET OO Model
- Interoperable
- Compact type-inferred classes

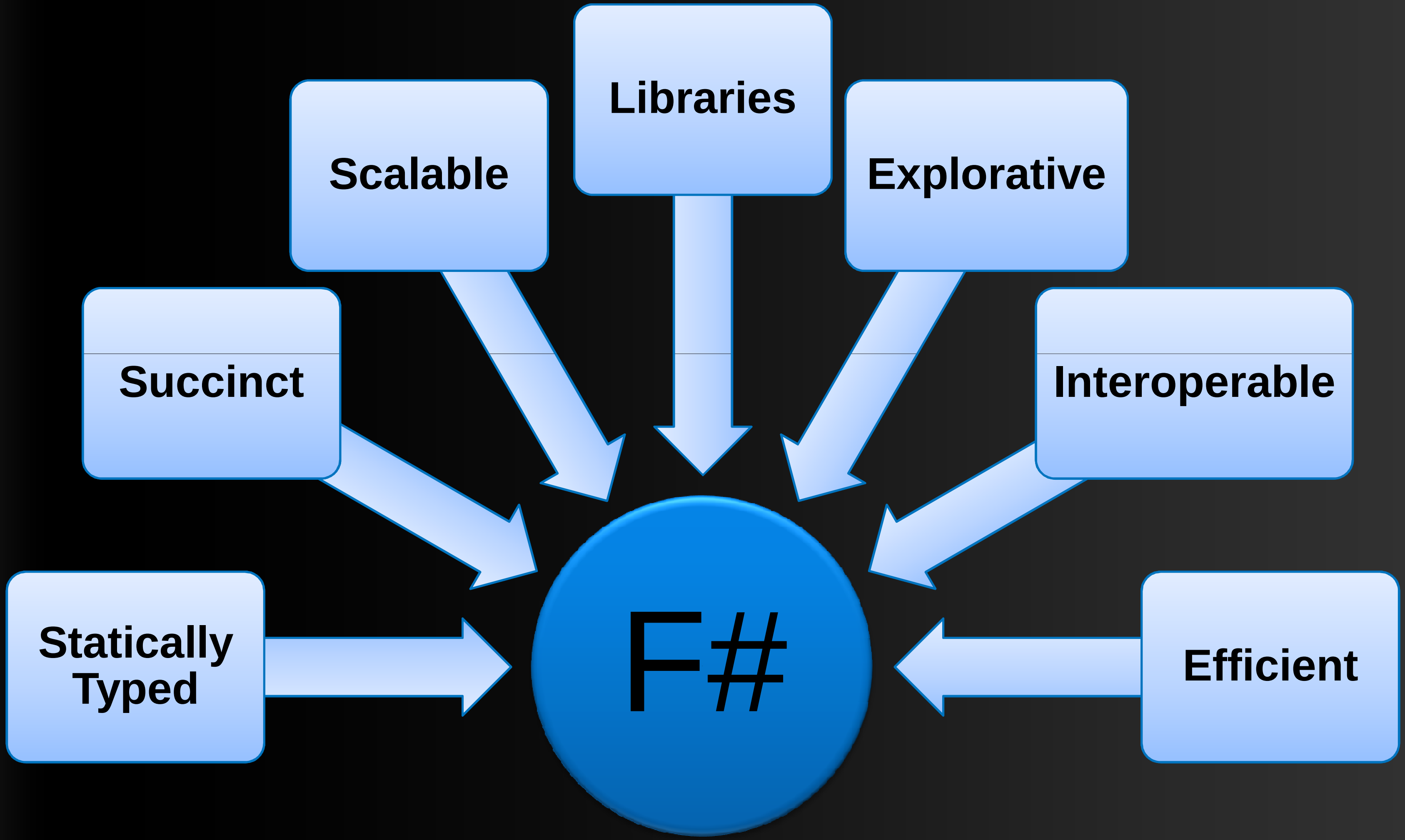
.NET

- Visual Studio
- Libraries
- Tools
- Concurrency
- LINQ

Tools

- F# Compiler
- F# Interactive
- Visual Studio Integration
- Debug
- Lex and Yacc

F#: The Combination Counts!



Libraries

Scalable

Explorative

Succinct

Interoperable

Statically Typed

Efficient

F#

F#: Combining Paradigms

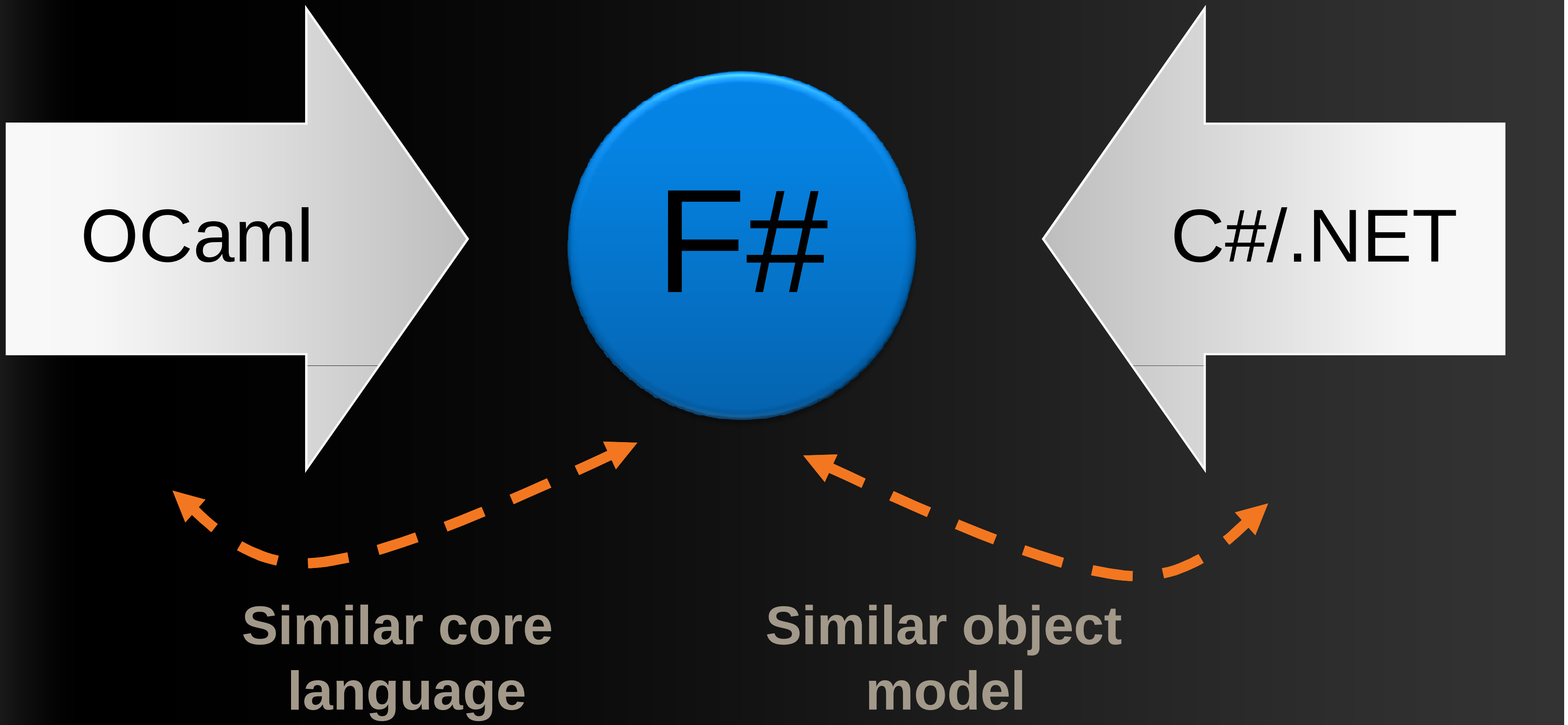
I've been coding in F# lately, for a production task.

F# allows you to ***move smoothly*** *in your programming style... I start with pure functional code, shift slightly towards an object-oriented style, and in production code, I sometimes have to do some imperative programming.*

I can ***start with a pure idea****, and still* ***finish my project with realistic code****. You're never disappointed in any phase of the project!*

Julien Laugel, Chief Software Architect, www.eurostocks.com

F#: Influences



The Path to Mastering F#

Topic	Covered Today
Scoping and “let”	✓
Tuples	✓
Pattern Matching	✓
Working with Functions	✓
Sequences, Lists, Options	✓
Records and Unions	✓
Basic Imperative Programming	✓
Basic Objects and Types	✓
The F# Libraries	✗
Advanced Functional/Imperative	✗
Advanced Functional/OO	✗
Language Oriented Programming	✓ **(later)**
Parallel and Asynchronous	✓ **(later)**

Quick Tour

Comments

```
// comment

(* comment *)

/// XML doc comment
let x = 1
```

Quick Tour

Overloaded Arithmetic

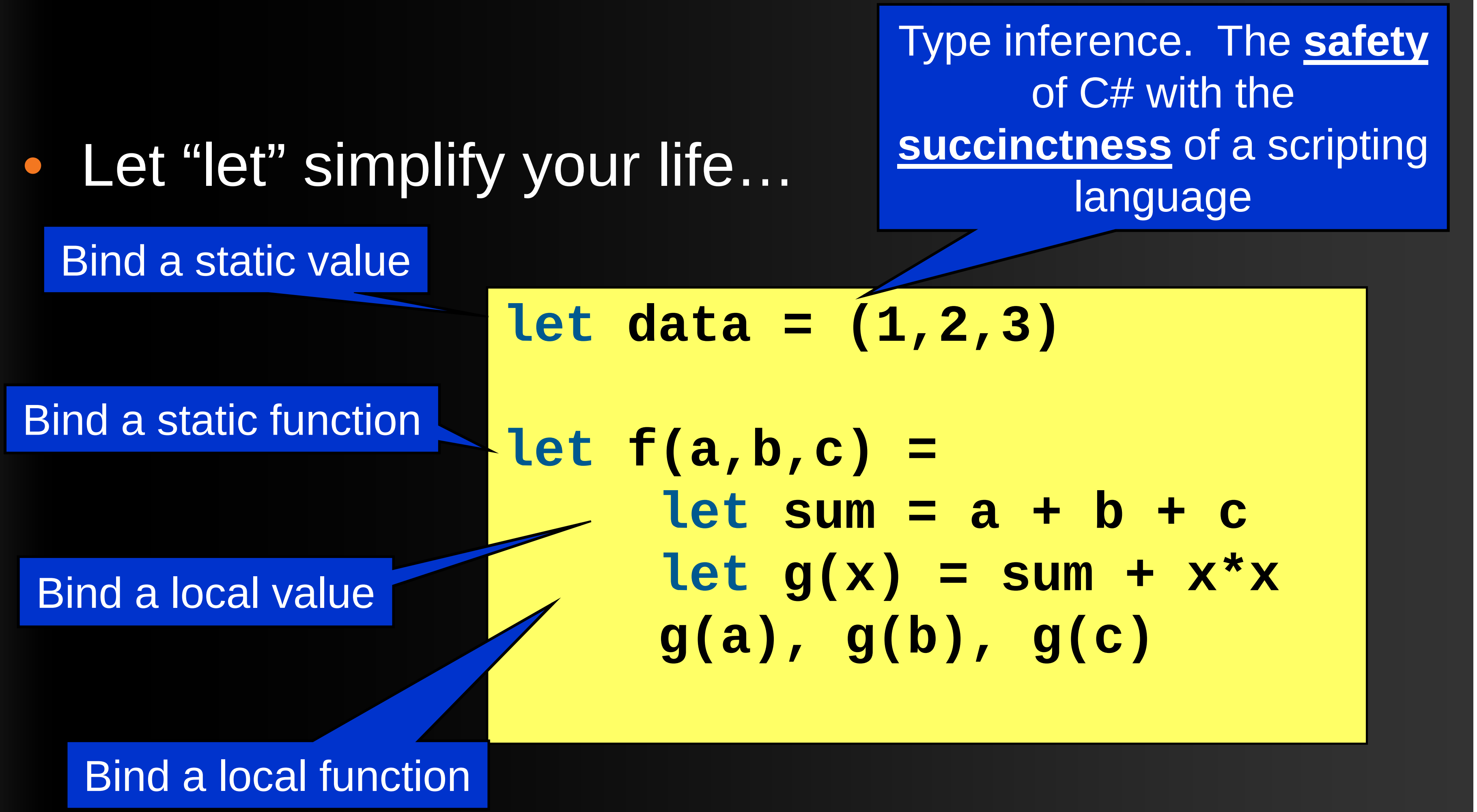
`x + y`	**Addition**
`x - y`	**Subtraction**
`x * y`	**Multiplication**
`x / y`	**Division**
`x % y`	**Remainder/modulus**
`-x`	**Unary negation**

Booleans

`not` *`expr`*	**Boolean negation**
`expr` `&&` *`expr`*	**Boolean “and”**
`expr` `||` *`expr`*	**Boolean “or”**

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 a 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)
```

Demo: Let’s WebCrawl…

Orthogonal & Unified Constructs

- Functions: like delegates + unified and simple

```
(fun x -> x + 1

let f(x) = x + 1

(f,f)

val f : int -> int
```

One simple mechanism, many uses

ymous
tion v

Declare a
functio

A pair
of functio

A function type

```
predicate = 'a -> bool

send = 'a -> unit

threadStart = unit -> unit

comparer = 'a -> 'a -> int

hasher = 'a -> int

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

F# - Functional

```
let f x = x+1

let pair x = (x,x)

let fst (x,y) = x

let data = (Some [1;2;3], Some [4;5;6])

match data with
| Some(nums1), Some(nums2) -> nums1 @ nums2
| None, Some(nums)   -> nums
| Some(nums), None   -> nums
| None, None         -> failwith "missing!"
```

F# - Functional

```
List.map        Seq.fold

Array.filter    Lazy           Set.union

Map    LazyList    Events    Async...

[ 0..1000 ]
[ for x in 0..10 -> (x, x * x) ]
[| for x in 0..10 -> (x, x * x) |]
seq { for x in 0..10 -> (x, x * x) }
```

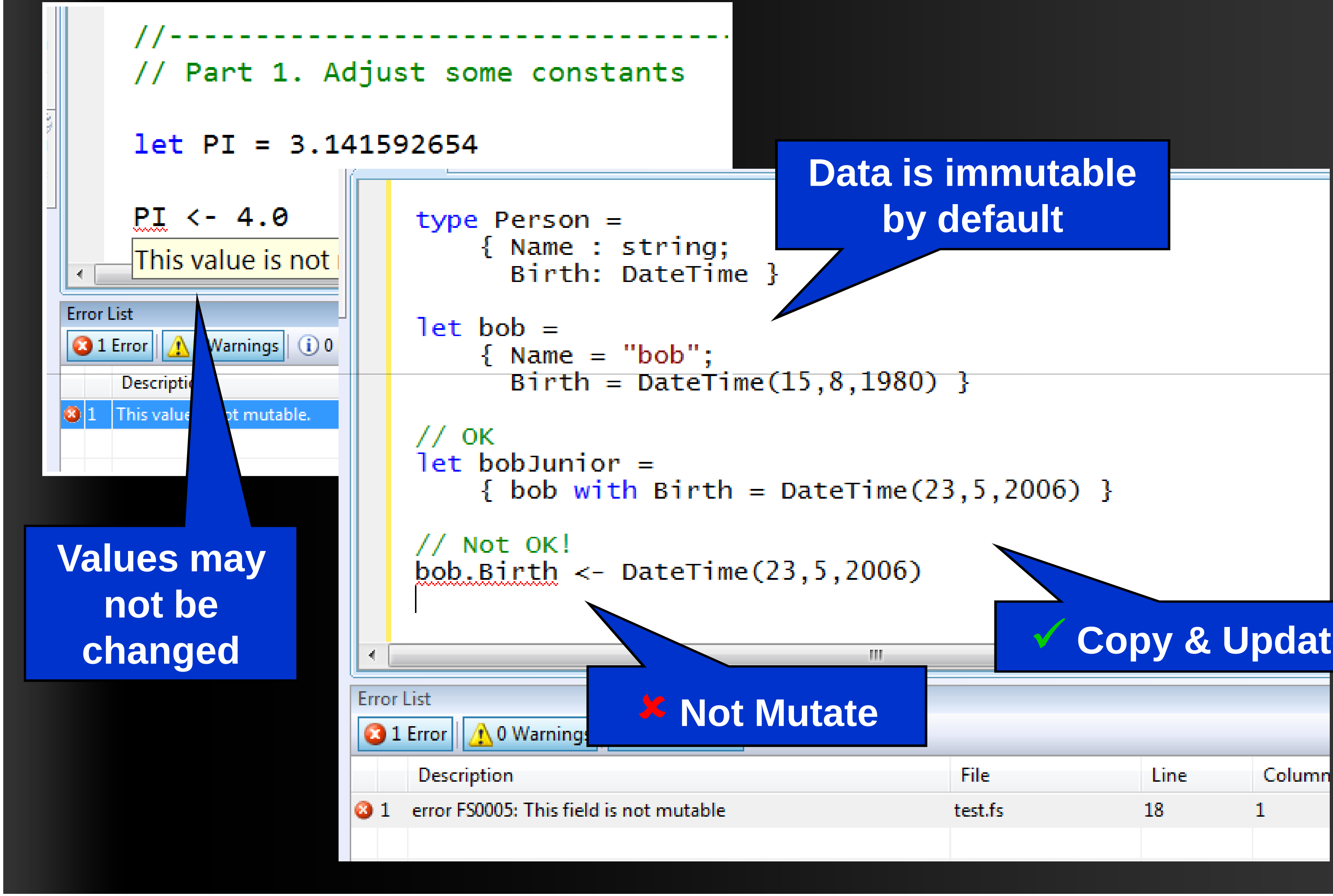
Range Expressions

List via query

Array via query

IEnumerable via query

Immutability the norm…



```
//-----------------------------------
// Part 1. Adjust some constants

let PI = 3.141592654

PI <- 4.0
```

This value is not

Error List

		Descripti
●	1	This value ot mutable.

```
type Person =
    { Name : string;
      Birth: DateTime }

let bob =
    { Name = "bob";
      Birth = DateTime(15,8,1980) }

// OK
let bobJunior =
    { bob with Birth = DateTime(23,5,2006) }

// Not OK!
bob.Birth <- DateTime(23,5,2006)
```

Error List

		Description	File	Line	Column
●	1	error FS0005: This field is not mutable	test.fs	18	1

Data is immutable by default

Values may not be changed

✓ Copy & Update

✗ Not Mutate

In Praise of Immutability

- Immutable objects can be relied upon
- Immutable objects can transfer between threads
- Immutable objects can be aliased safely
- Immutable objects lead to (different) optimization opportunities

F# - Imperative + Functional

Using .NET collections

```
open System.Collections.Generi

let dict = new Dictionary<int,string>(1000)

dict.[17] <- "Seventeen"
dict.[1000] <- "One Grand"

for (KeyValue(k,v)) in dict do
    printfn "key = %d, value = %s" k v
```

F# - Imperative + Functional

“use” =
C# “using”

```
open System.IO
open System.Collections.Generic

let readAllLines(file) =
    use inp = File.OpenText file
    let res = new List<_>()
    while not(inp.EndOfStream) do
        res.Add(inp.ReadLine())
    res.ToArray()
```

F# - Sequences

Sequence Expressions and On -demand I/O

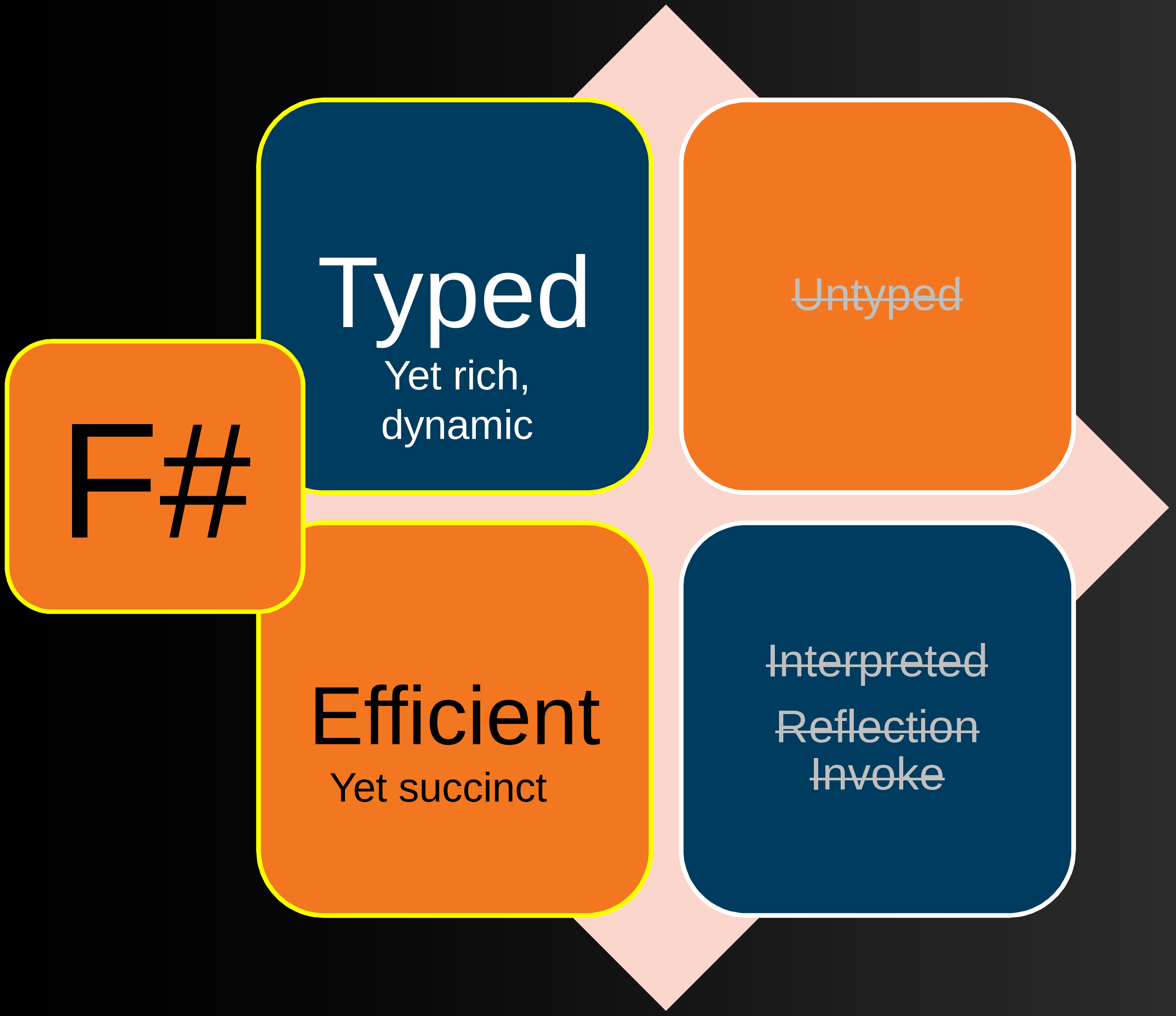
```
open System.IO
let rec allFiles(dir) =
  seq
   { for file in Directory.GetFiles(dir) do
        yield file
     for sub in Directory.GetDirectories(dir) do
        yield! allFiles(sub) }

allFiles(@"C:\WINDOWS") |> Seq.take 100 |> show
```

Weakly Typed? Slow?

```
//F#
#light
open System
let a = 2
Console.WriteLine(a)
```

```
//C#
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static int a()
        {
            return 2;
        }

           tic void Main(string[] args)

            Console.WriteLine(a);
```

Looks Weakly typed?
Maybe Dynamic?



F#

Typed

Yet rich,
dynamic

~~Untyped~~

Efficient

Yet succinct

~~Interpreted~~

~~Reflection~~
~~Invoke~~

F# - Imperative + Functional

Read lines on demand

```
open System.IO
let allLines =
  seq { use inp = File.OpenText "test.txt"
        while not(inp.EndOfStream) do
            yield (inp.ReadLine()) }
```

Pipelines

```
allLines
   |> Seq.truncate 1000
   |> Seq.map (fun s -> uppercase s,s)
   |> Seq.to_array
```

Why Object-Oriented Programming

- Builds reusable components
- Scalable design
- Easier maintenance
- Language of .NET/Java

Objects

Class Types

```
type ObjectType(args) =

    let internalValue = expr
    let internalFunction args = expr
    let mutable internalState = expr

    member x.Prop1 = expr
    member x.Meth2 args = expr
```

Constructing Objects

```
new FileInfo(@"c:\misc\test.fs")
```

Interface Types

```
type IObject =
  interface ISimpleObject
  abstract Prop1 : type
  abstract Meth2 : type -> type
```

F# - Objects + Functional

```
type Vector2D(dx:double,dy:double) =
   member v.DX = dx
   member v.DY = dy
   member v.Length = sqrt(dx*dx+dy*dy)
   member v.Scale(k) = Vector2D(dx*k,dy*k)
```

Inputs to object construction

Exported properties

Exported method

F# - Objects + Functional

```
type Vector2D(dx:double,dy:double) =
    let norm2 = dx*dx+dy*dy
    member v.DX = dx
    member v.DY = dy
    member v.Length = sqrt(norm2)
    member v.Norm2 = norm2
```

Internal (pre-computed) values and functions

F# - Objects + Functional

Immutable inputs

```
type HuffmanEncoding(freq:seq<char*int>) =

    ...
    < 50 lines of beautiful functional code>
    ...

     member x.Encode(input: seq<char>) =
          encode(input)

     member x.Decode(input: seq<char>) =
          decode(input)
```

Internal tables

Publish access

F# - Objects + Functional

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

   let mutable currDX = dx

   let mutable currDX = dy

   member v.DX = currDX

   member v.DY = currDY

   member v.Move(x,y) =
        currDX <- currDX+x
        currDY <- currDY+y
```

Internal state

Publish internal state

Mutate internal state

F# - Language Oriented

Embedded Language

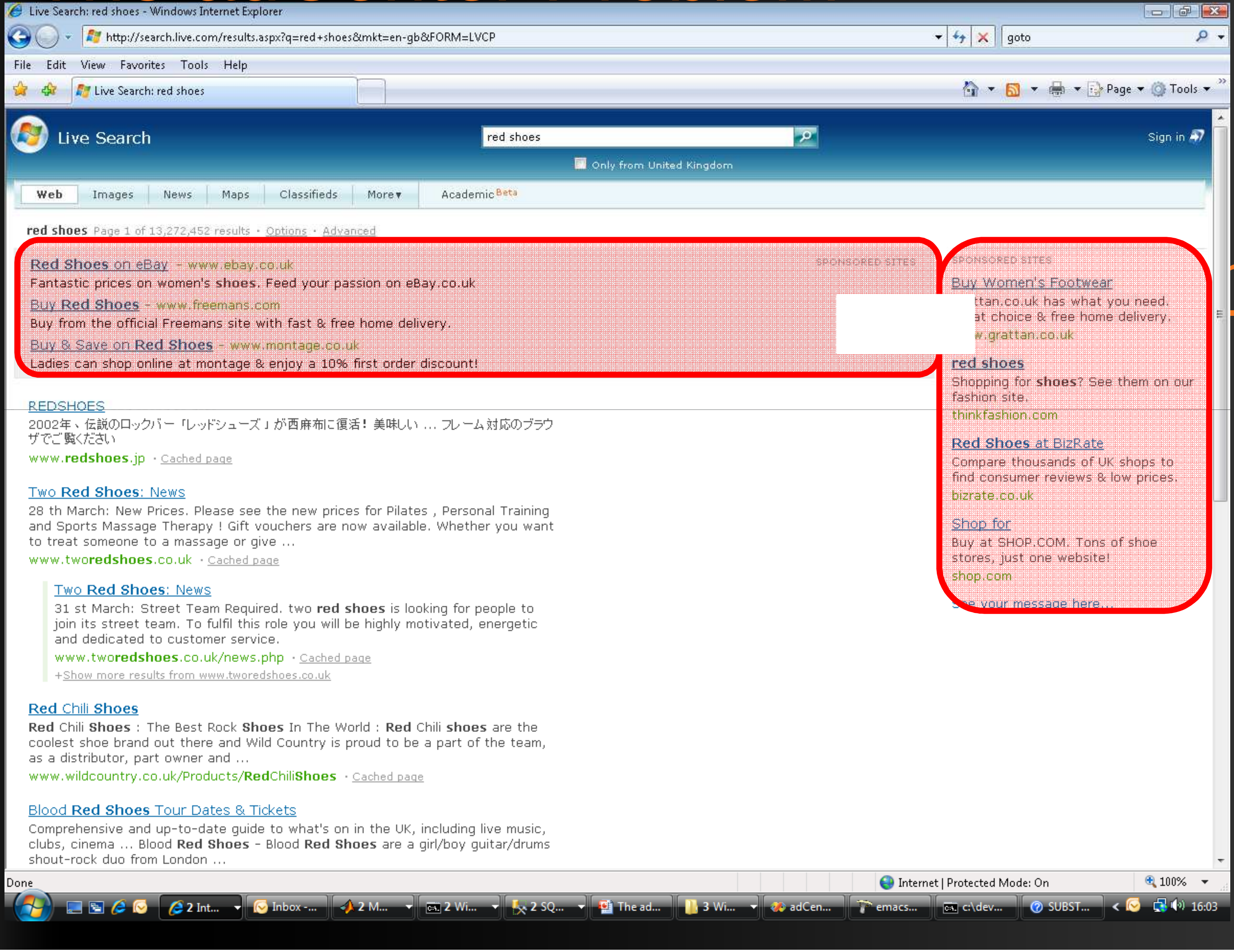
```
type PropLogic =
    | And of PropLogic * PropLogic
    | Not of PropLogic
    | True

let rec Eval(prop) =
    match prop with
    | And(a,b) -> Eval(a) && Eval(b)
    | Not(a) -> not (Eval(a))
    | True -> true
```

Crisp Semantics

Case Study

The adPredict Competition

The adCenter Problem


Live Search: red shoes - Windows Internet Explorer
http://search.live.com/results.aspx?q=red+shoes&mkt=en-gb&FORM=LVCP
goto
File Edit View Favorites Tools Help
Live Search: red shoes
Page Tools
Live Search
red shoes
Sign in
Only from United Kingdom
Web Images News Maps Classifieds More▾ Academic Beta
Done
Internet | Protected Mode: On
100%
2 Int... Inbox -... 2 M... 2 Wi... 2 SQ... The ad... 3 Wi... adCen... emacs... c:\dev... SUBST...
16:03

red shoes Page 1 of 13,272,452 results · Options · Advanced

SPONSORED SITES

Red Shoes on eBay - www.ebay.co.uk
Fantastic prices on women's shoes. Feed your passion on eBay.co.uk

Buy Red Shoes - www.freemans.com
Buy from the official Freemans site with fast & free home delivery.

Buy & Save on Red Shoes - www.montage.co.uk
Ladies can shop online at montage & enjoy a 10% first order discount!

REDSHOES
2002年、伝説のロックバー「レッドシューズ」が西麻布に復活！ 美味しい ... フレーム対応のブラウザでご覧ください
www.redshoes.jp · Cached page

Two Red Shoes: News
28 th March: New Prices. Please see the new prices for Pilates , Personal Training and Sports Massage Therapy ! Gift vouchers are now available. Whether you want to treat someone to a massage or give ...
www.tworedshoes.co.uk · Cached page

Two Red Shoes: News
31 st March: Street Team Required. two red shoes is looking for people to join its street team. To fulfil this role you will be highly motivated, energetic and dedicated to customer service.
www.tworedshoes.co.uk/news.php · Cached page
+Show more results from www.tworedshoes.co.uk

Red Chili Shoes
Red Chili Shoes : The Best Rock Shoes In The World : Red Chili shoes are the coolest shoe brand out there and Wild Country is proud to be a part of the team, as a distributor, part owner and ...
www.wildcountry.co.uk/Products/RedChiliShoes · Cached page

Blood Red Shoes Tour Dates & Tickets
Comprehensive and up-to-date guide to what's on in the UK, including live music, clubs, cinema ... Blood Red Shoes - Blood Red Shoes are a girl/boy guitar/drums shout-rock duo from London ...

SPONSORED SITES

Buy Women's Footwear
ttan.co.uk has what you need.
at choice & free home delivery.
w.grattan.co.uk

red shoes
Shopping for shoes? See them on our fashion site.
thinkfashion.com

Red Shoes at BizRate
Compare thousands of UK shops to find consumer reviews & low prices.
bizrate.co.uk

Shop for
Buy at SHOP.COM. Tons of shoe stores, just one website!
shop.com

See your message here...

F# and adCenter

- 4 week project, 4 machine learning experts
- 100million probabilistic variables
- Processes 6TB of training data
- Real time processing

AdPredict: What We O

- Quick Coding
- Agile Coding
- Scripting
- Performance
- Memory-Faithful
- Succinct
- Symbolic
- .NET Integration

F#’s powerful type inference means less typing, more thinking

Type-inferred code is easily refactored

“Hands-on” exploration.

Immediate scaling to massive data sets

mega-data structures, 16GB machines

Live in the **domain**, not the language

Schema compilation and “Schedules”

Especially Excel, SQL Server

F# Async/Parallel

Concurrent programming with shared state…

…can be hard

F# - Concurrent/Reactive/Parallel

- **Concurrent**: *Multiple threads* of execution
- **Parallel**: These execute *simultaneously*
- **Asynchronous**: Computations that complete "*later*"
- **Reactive**: *Waiting* and *responding* is normal

Why is it so hard?

- To get 50 web pages in parallel?
- To get from thread to thread?
- To create a worker thread that reads messages?
- To handle failure on worker threads?

Why isn’t it this easy?

```
let ProcessImages() =
    Async.Run
      (Async.Parallel
         [ for i in 1 .. numImages -> ProcessImage(i) ])
```

Why isn’t it this easy?

```
let task =
    async { ...
            do! SwitchToNewThread()
            ...
            do! SwitchToThreadPool()
            ...
            do! SwitchToGuiThread()
            .... }
```

Simple Examples

Compute 22 and 7 in parallel

```
Async.Parallel [ async { -> 2*2 + 3*6 };
                 async { -> 3 + 5 - 1 } ]
```

Get these three web pages and wait until all have come back

```
Async.Parallel [WebRequest.Async "http://www.live.com
                WebRequest.Async "http://www.yahoo.com ,
                WebRequest.Async "http://www.google.com" ]
```

```
let parArrMap f (arr: _[]) =
   Async.Run (Async.Parallel [| for x in arr -> async { -> f x } |])
```

Naive Parallel Array Map

Taming Asynchronous I/O

```
using System;
using System.IO;
using System.Threading;

public class BulkImageProcAsync
{
    public const String ImageBaseName
    public const int numImages = 200
    public const int numPixels = 512

    // ProcessImage has a simple O(N
    // of times you repeat that loop
    // bound or more IO-bound.
    public static int processImageRe

    // Threads must decrement NumImag
    // their access to it through a
    public static int NumImagesToFin
    public static Object[] NumImagesM
    // WaitObject is signalled when
    public static Object[] WaitObjec
    public class ImageStateObject
    {
        public byte[] pixels;
        public int imageNum;
        p
    }
```

```
public static void ReadInImageCallback(IAsyncResult as
{
    ImageStateObject state = (ImageStateObject)asyncRe
    Stream stream = state.fs;
    int bytesRead = stream.EndRead(asyncResult);
    if (bytesRead != numPixels)
        throw new Exception(String.Format
            ("In ReadInImageCallback, got the wrong nu
            "bytes from the image: {0}.", bytesRead));
    ProcessImage(state.pixels, state.imageNum);
    stream.Close();

    // Now write out the image.
    // Using asynchronous I/O here appears not to be b
    // It ends up swamping the threadpool, because the
    // threads are blocked on I/O requests that were j
    // the threadpool.
    FileStream fs = new FileStream(ImageBaseName + sta
        ".done", FileMode.Create, FileAccess.Write, Fi
        4096, false);
    fs.Write(state.pixels, 0, numPixels);
    fs.Close();

                                        much memory.
                                        ible is a good

                                         now.

    }
}
```

```
    public static void ProcessImagesInBulk()
    {
        Console.WriteLine("Processing images...  ");
        long t0 = Environment.TickCount;
        NumImagesToFinish = numImages;
        AsyncCallback readImageCallback = new
            AsyncCallback(ReadInImageCallback);
        for (int i = 0; i < numImages; i++)
        {
            ImageStateObject state = new ImageStateObject();
            state.pixels = new byte[numPixels];
            state.imageNum = i;
            // Very large items are read only once, so you can make the
            // buffer on the FileStream very small to save memory.
            FileStream fs = new FileStream(ImageBaseName + i + ".tmp",
                FileMode.Open, FileAccess.Read, FileShare.Read, 1, true);
            state.fs = fs;
            fs.BeginRead(state.pixels, 0, numPixels, readImageCallback,
                state);
        }

        // Determine whether all images are done being processed.
        // If      block until all are finished.
        bool mustBloc
        lock (NumImagesMutex)
        {
            if (NumImagesToFinish > 0)
                mustBlock = true;
        }
        if (mustBlock)
        {
            Console.WriteLine("All worke
                " Blocking until they complete. numLeft: {0}",
                NumImagesToFinish);
            Monitor.Enter(WaitObject);
            Monitor.Wait(WaitObject);
            Monitor.Exit(WaitObject);
        }
        long t1 = Environment.TickCount;
        Console.WriteLine("Total time processing images: {0}ms",
            (t1 - t0));
    }
}
```

```
let ProcessImageAsync () =
    async { let  inStream  = File.OpenRead(sprintf "Image%d.tmp" i)
            let! pixels    = inStream.ReadAsync(numPixels)
            let  pixels'   = TransformImage(pixels,i)
            let  outStream = File.OpenWrite(sprintf "Image%d.done" i)
            do!  outStream.WriteAsync(pixels')
            do   Console.WriteLine "done!"  }

let ProcessImagesAsyncWorkflow() =
    Async.Run (Async.Parallel
                 [ for i in 1 .. numImages -> ProcessImageAsync i ])
```

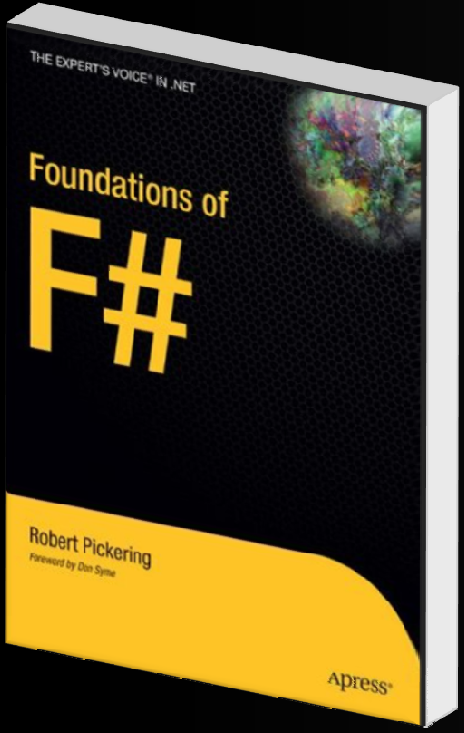
Processing 200 images in parallel

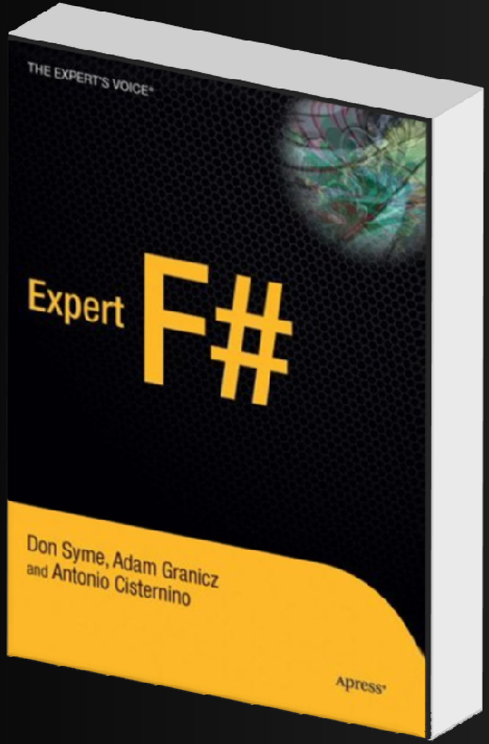
8 Ways to Learn

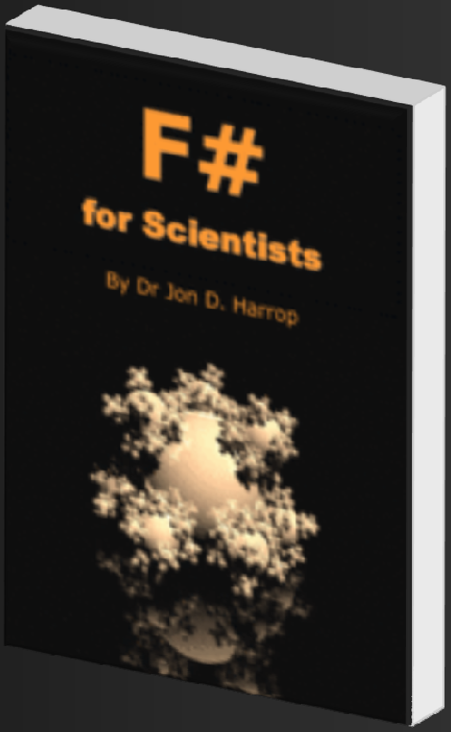
- FSI.exe
- Samples Included
- Go to definition
- Lutz’ Reflector

- http://cs.hubfs.net
- Codeplex Fsharp Samples
- Books
- ML

Books about F#







Visit **www.fsharp.net**

Getting F#

- September CTP released (1.9.6)
  - Focus on Simplicity, Regularity, Correctness

- Next stop “Visual Studio 2010”

Questions & Discussion

Microsoft®

This presentation is for informational purposes only.

MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.