Why is Microsoft investing in Functional Programming?

Don Syme

With thanks to Leon Bambrick, Chris Smith and the puppies

All opinions are those of the author and not necessarily those of Microsoft

Simplicity

Economics

Fun

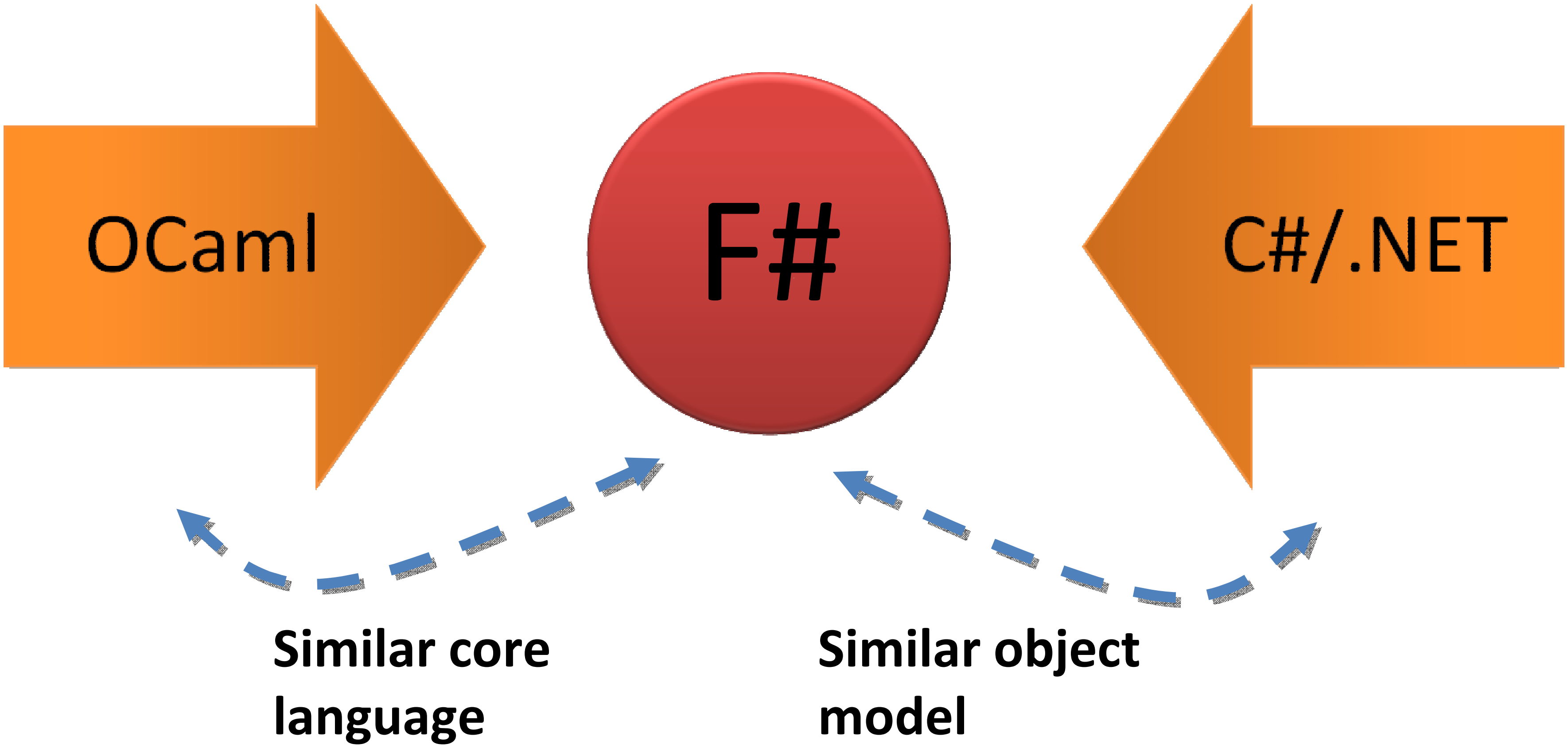
What Investments?

- C#
  - C# 2.0 (generics)
  - C# 3.0 (Language Integrated Queries - LINQ)
  - These represent a major industry shift towards functional programming
- F#
  - Bringing F# to product quality
- Haskell
  - Strongly supporting Haskell research
- VB, Python, Ruby
  - These incorporate many functional features and overlap with the functional programming ethos

Who?

- Microsoft Research (“MSR”)
  - F#
  - Haskell
- Microsoft Developer Division (“DevDiv”), Visual Studio Languages Group
  - C#
  - Visual Basic
  - F#
  - Python
  - Ruby

F#: Influences



OCaml

F#

C#/.NET

Similar core language

Similar object model

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);
    }
  }
}
```

More 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

```
// Use first-order functions as commands
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

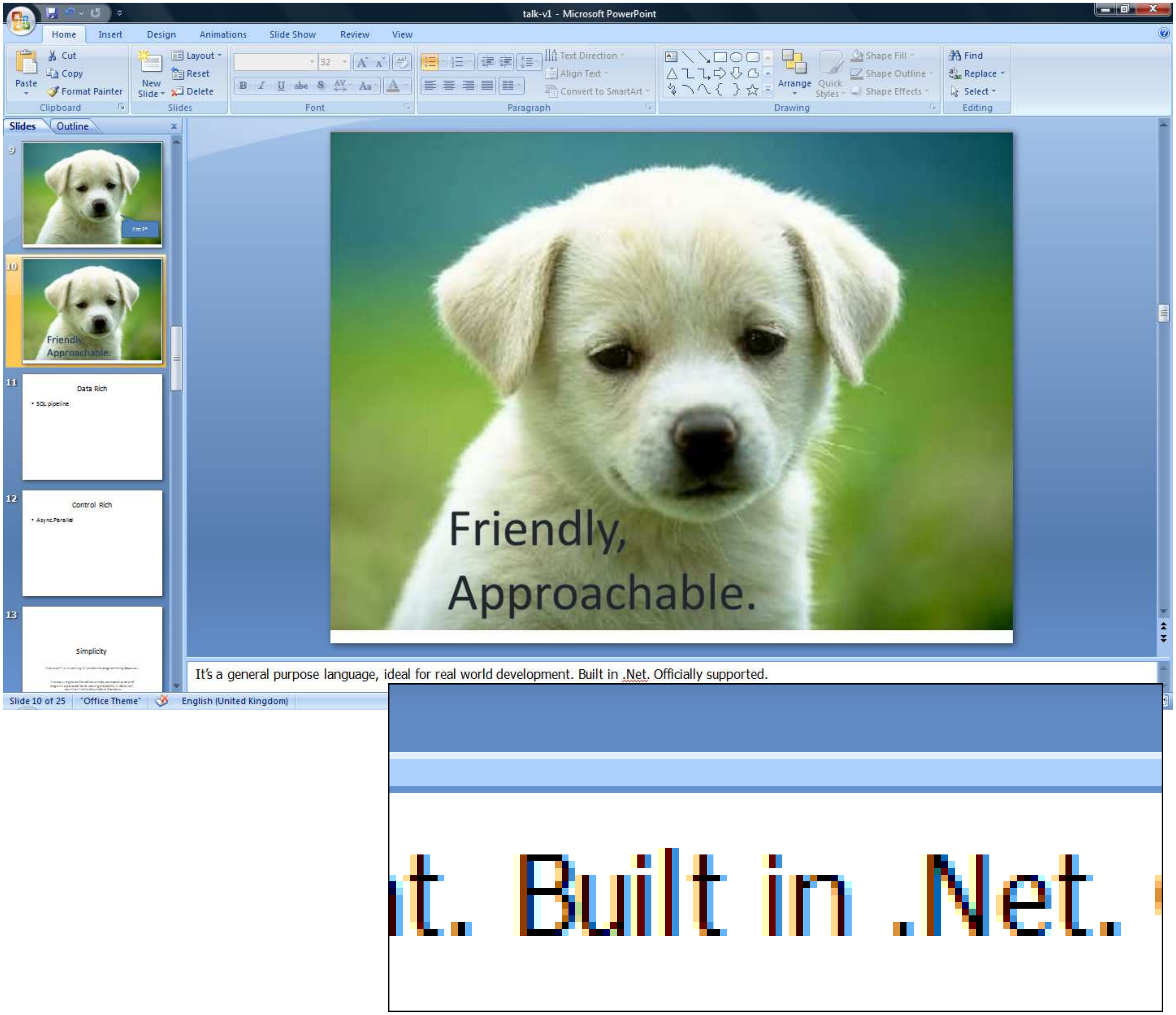
```
abstract class Command
 {
     public virtual void Execute();
 }
 abstract class MarsRoverCommand : Command
 {
     protected MarsRover Rover { get; private
  set; }

     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);
}
```



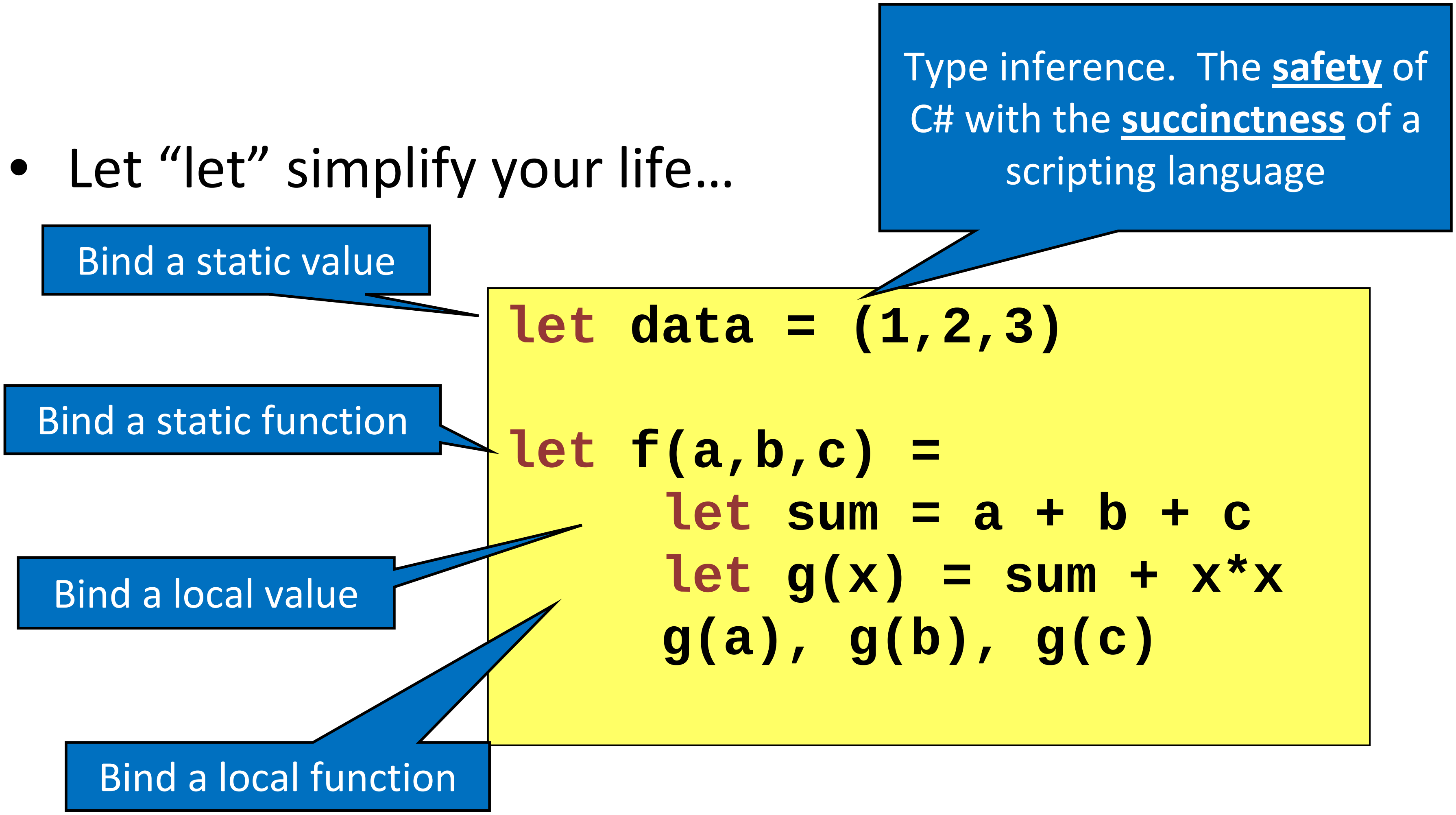
Friendly,
Approachable.

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

t. Built in .Net.

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

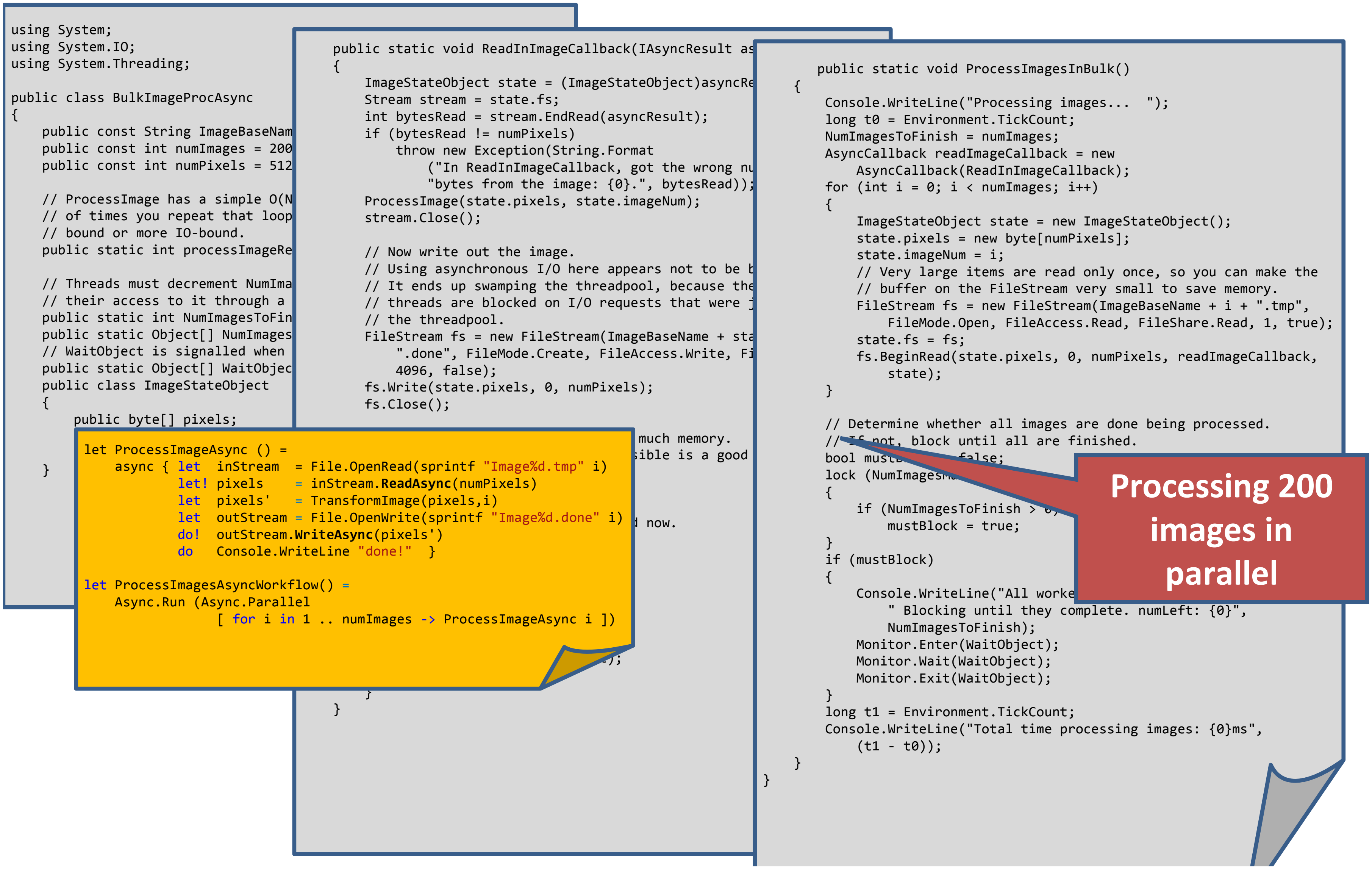
Bind a static function

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)
```

Simplicity



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

public class BulkImageProcAsync
{
    public const String ImageBaseNam
    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 NumIma
    // their access to it through a
    public static int NumImagesToFin
    public static Object[] NumImages
    // WaitObject is signalled when
    public static Object[] WaitObjec
    public class ImageStateObject
    {
        public byte[] pixels;

    }
```

```
    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.
                                               sible is a good


                                                d now.






                                                  );

        }
    }
```

```
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 ])
```

```
    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 not, block until all are finished.
    bool must      false;
    lock (NumImagesT
    {
        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));
  }
}
```

Processing 200 images in parallel

Simplicity

Microsoft is investing in functional programming because....

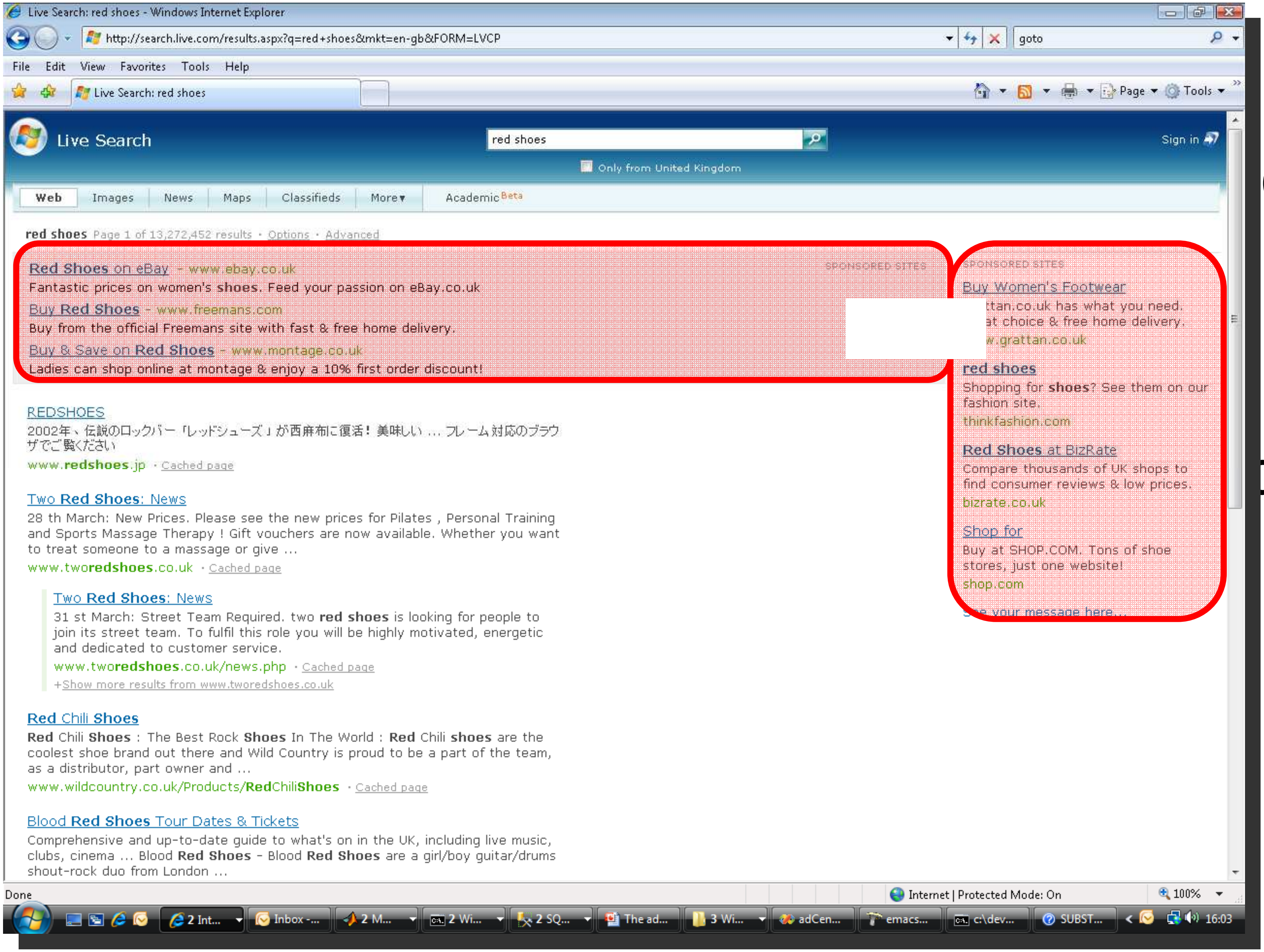
It enables simple, compositional and elegant problem solving in data-rich, control-rich and symbolic domains

Case Study

Ad Ranking,

MSR Cambridge Online Services and Advertising Group

The adCenter Problem



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!

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

e your message here...

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.two**redshoes**.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.two**redshoes**.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/**Red**Chili**Shoes** · 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 …

d

d

OSA Machine Learning

- Internal Competition
- Use F# for major adCenter and Xbox Live projects
  - 4 week project, 4 machine learning experts
  - 100million probabilistic variables
  - Processes 6TB of training data
  - Real time processing

“F# was absolutely integral to our success”

“We delivered a robust, high-performance solution on-time.”

“We couldn’t have achieved this with any other tool given the constraints of the task”

“F# programming is fun – I feel like I learn more about programming every day”

OSA Machine Le

Observations

– Quick Coding

– Agile Coding

– Scripting

– Performance

– Memory-Faithful

– Succinct

– Symbolic

– .NET Integration

F#’s type inference means less typing,

Int

or

Immediate scaling to massive data sets

Live in the **domain**.

Schema compilation and efficient “Schedule” representations key

Especially Excel, SQL Server

The Team’s Summary

- “F# was absolutely integral to our success”
- “We delivered a robust, high-performance solution on-time.”
- “We couldn’t have achieved this with any other tool given the constraints of the task”
- “F# programming is fun – I feel like I learn more about programming every day”

Some Code Highlights

- Type-safe Schema Bulk Import

```
BulkImporter<'Schema>:
   database:string * prefix:string -> BulkImport<'Schema>
```

  - Written as part of the team’s toolchain
  - Schema in F# types
  - Compiled using F# “schema compilation” techniques
  - 800 lines

  - Enabled team to clean and insert entire data set over 3 day period

Some Code Highlights

The essence of their data import line

```
/// Create the SQL schema
let schema = BulkImporter<PageView> ("cpidssdm18", "Cambridge", “June10")

/// Try to open the CSV file and read it pageview by pageview
File.OpenTextReader “HourlyRelevanceFeed.csv"
|> Seq.map (fun s -> s.Split [|','|])
|> Seq.chunkBy (fun xs -> xs.[0])
|> Seq.iteri (fun i (rguid,xss) ->
    /// Write the current in-memory bulk to the Sql database
    if i % 10000 = 0 then
        schema.Flush ()

    /// Get the strongly typed object from the list of CSV file lines
    let pageView = PageView.Parse xss

    /// Insert it
    pageView |> schema.Insert
)
/// One final flush
schema.Flush ()
```

Some Code Highlights

```
/// A schedule of computation in a factor graph
/// (i.e., a series of update functions
/// and variables that should be updated)
type Schedule =
    | ScheduleStep of (IFactor * int)
    | ScheduleSeq of Schedule[]
    | ScheduleLoop of Schedule * float

    /// Runs a schedule
    member schedule.Run() =
        match schedule with
        | ScheduleStep (fac,idx)    ->
            fac.UpdateMessage idx
        | ScheduleSeq sseq          ->
            sseq |> Seq.maxOn (fun s -> s.Run())
        | ScheduleLoop (s,maxDelta) ->
            let delta = s.Run()
            if delta > maxDelta then schedule.Run() else delta
```

Expressing and evaluating “Approximation Schedules” was crucial to this work.

Functional programming made this beautiful

(Aside: Units Of Measure)

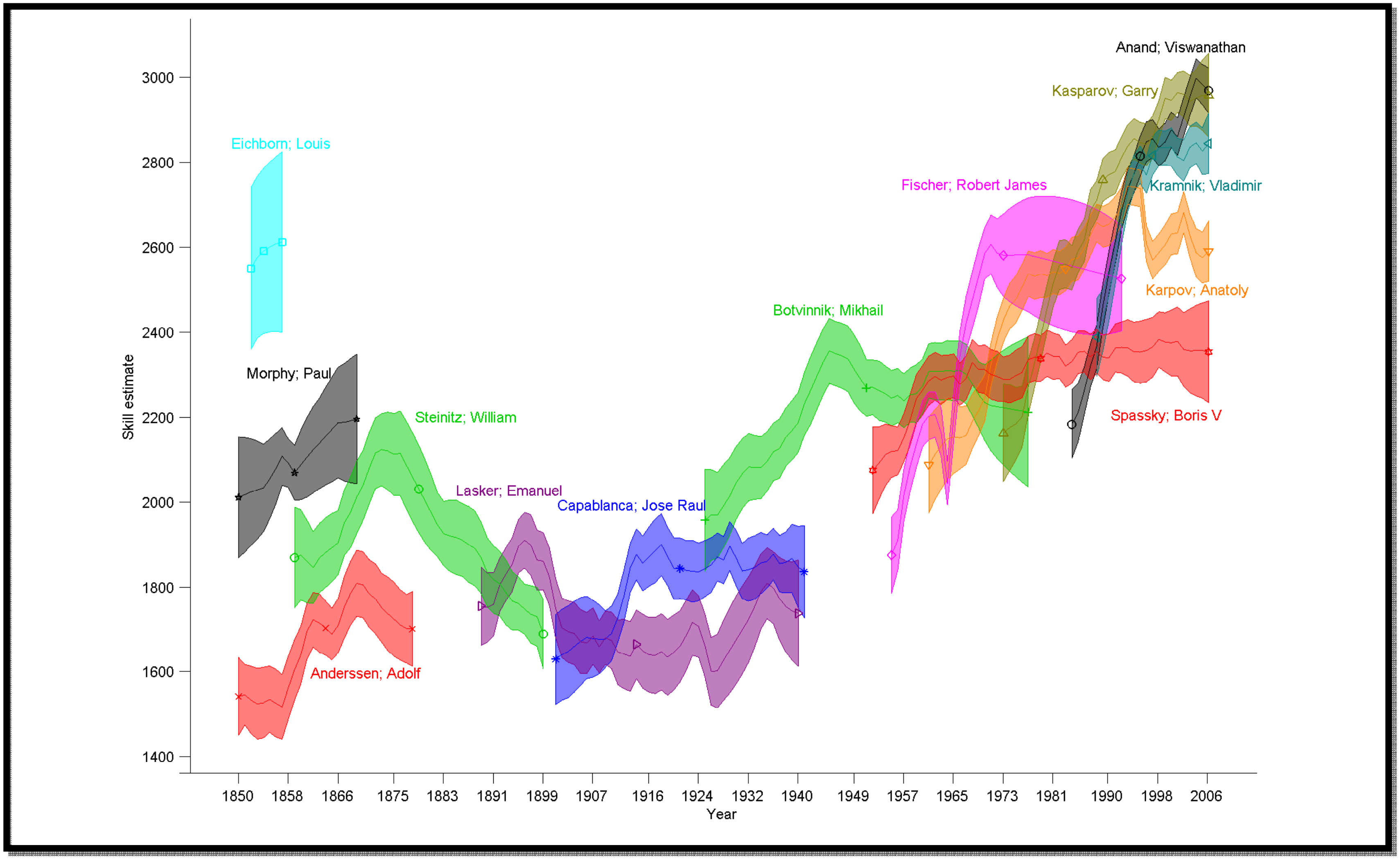
```
type acceleration = float<m / s^2>

let fast = 3.0e6<m/s>
let gravity = -9.81<m/s^2)
```

The F# September CTP includes “units of measure” Inference and checking

```
/// Computes the absolute difference between two Gaussians
let AbsoluteDifference (a:Gaussian<‘u>,b:Gaussian<‘u>) =
    max (abs(a.PrecisionMean - b.PrecisionMean))
        (sqrt(abs(a.Precision - b.Precision)))
```

Re-Ranking the History of Chess

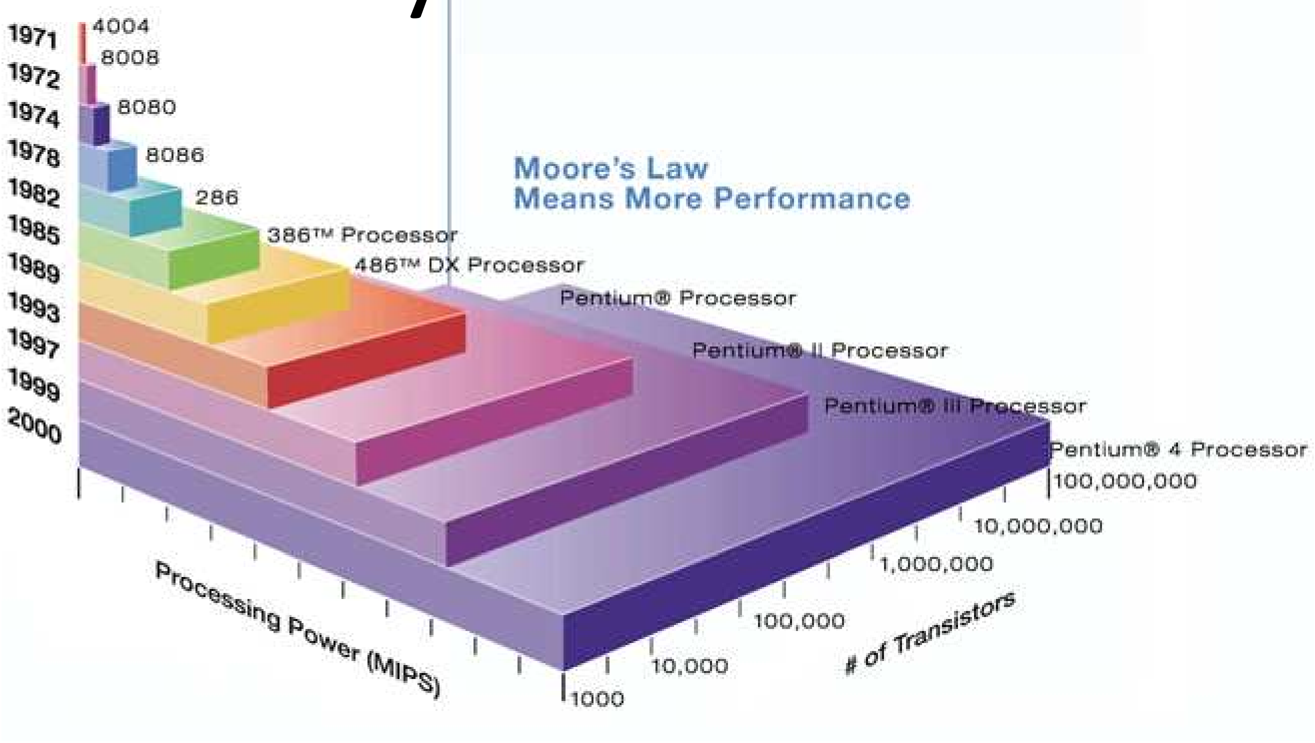
Search for "TrueSkill Through Time" (MSR Cambridge Online Services and Advertising Group)



Control Rich

```
Async.Run
    (Async.Parallel
        [ Async.GetHttp "www.google.com";
          Async.GetHttp "www.live.com";
          Async.GetHttp "www.yahoo.com"; ]
```

Why learn F#?



Moore’s Law, but no speed increase

Parallelism

- The Economics of the Hardware Industry are Changing

- Functional programming is a crucial tool in parallel and asynchronous programming
  - For architecture
  - For implementation

- Good synergies, e.g. with Parallel Extensions for .NET

Economics

Economies of Scale at Microsoft

- Have .NET
- Have .NET Libraries
- Have Visual Studio, Silverlight, .NET CF, ASP.NET, XNA GameStudio, RoboticsStudio
- Have Tools (profilers, debuggers, designers)

- Given this basis, the opportunities for low-cost, value-add investments are enormous:
  - Dynamic Languages
  - Functional Languages
  - Web programming (client, server, services)
  - Business programming
  - Parallel programming
  - Game programming
  - Data mining programming

- Cost: low, Value: high

Economics for Users

- Learn .NET
- Can use the tools right for the job
- Can reuse much knowledge from tool to tool

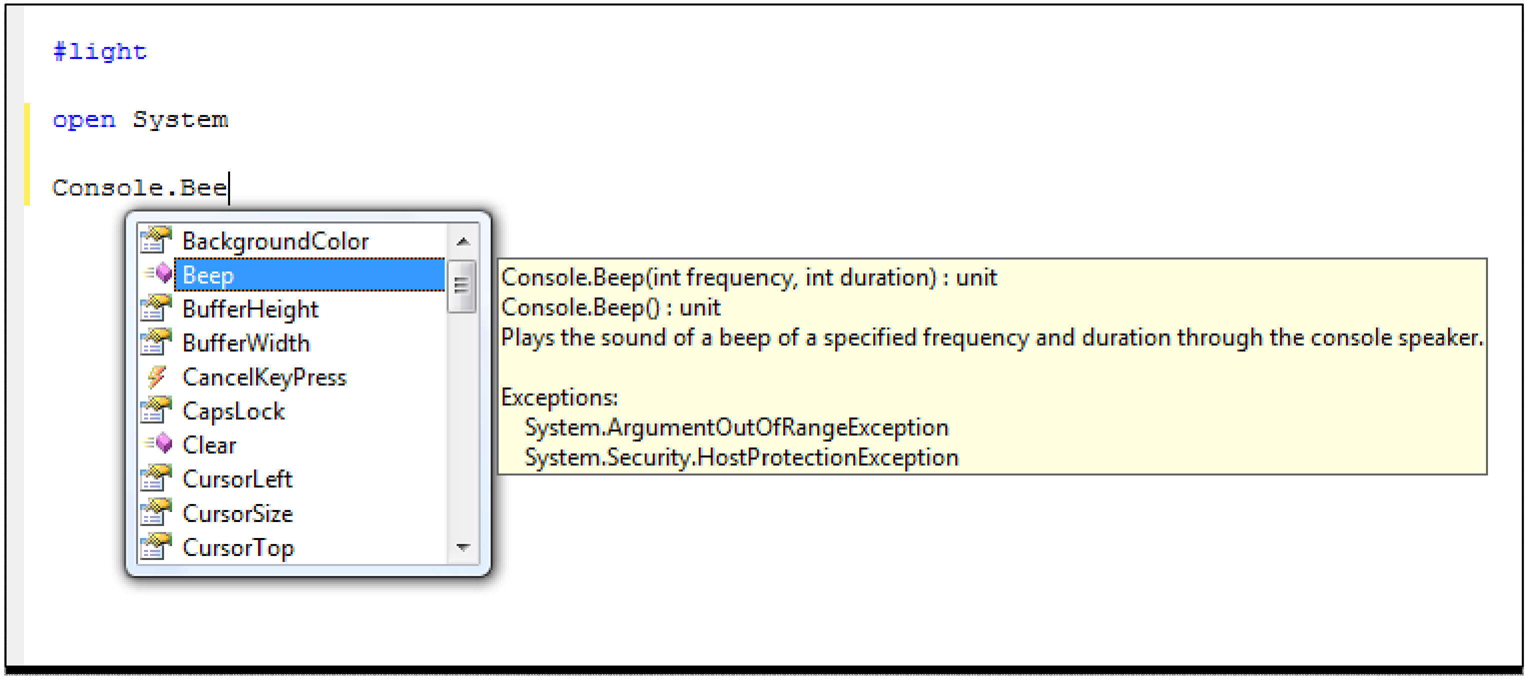
Economics

Microsoft is investing in functional programming because....

It is a sensible, relatively low-cost investment that adds real value to Visual Studio and the .NET Framework

Fun

This is fun

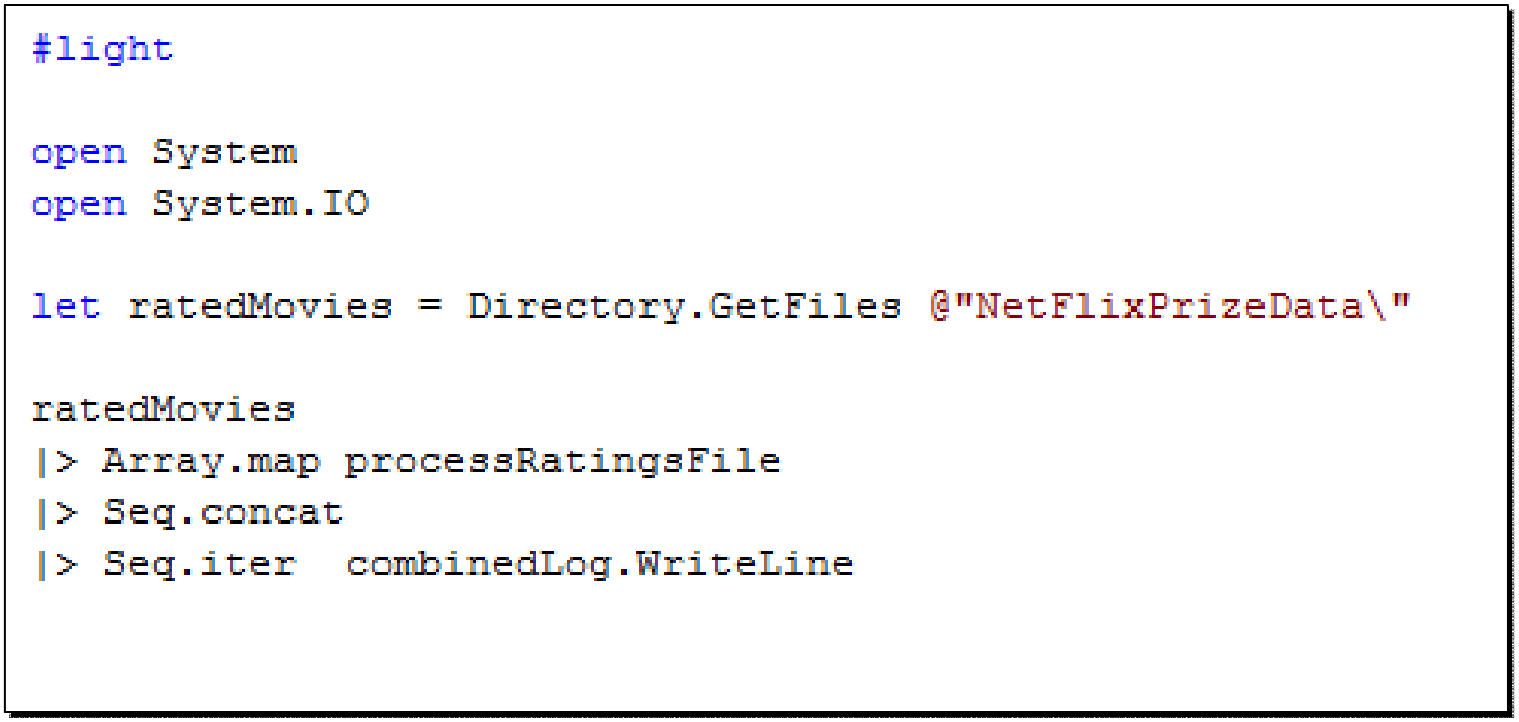


```
#light

open System

Console.Bee
```

Console.Beep(int frequency, int duration) : unit
Console.Beep() : unit
Plays the sound of a beep of a specified frequency and duration through the console speaker.

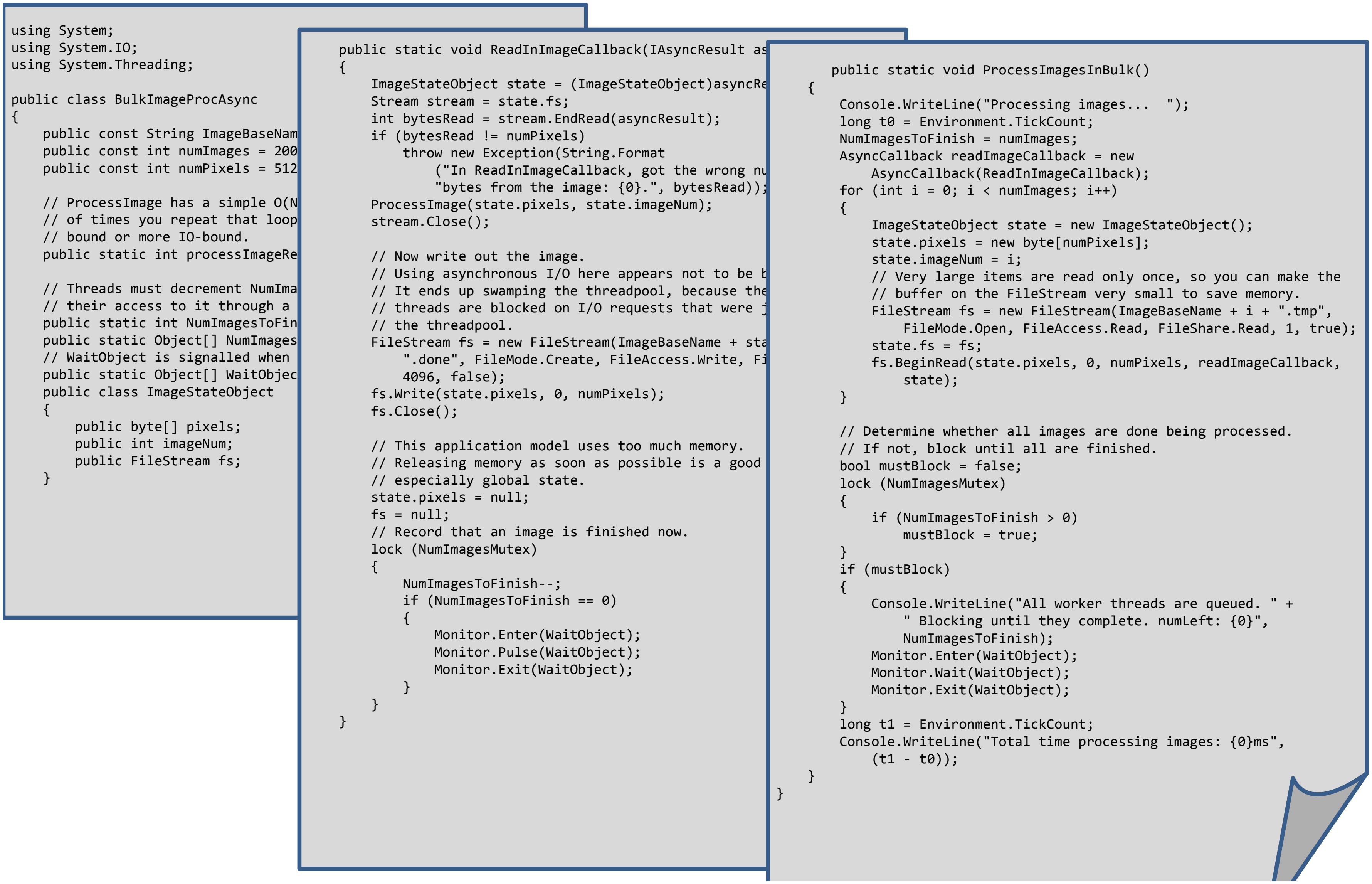
Exceptions:
System.ArgumentOutOfRangeException
System.Security.HostProtectionException

This is fun



```
#light

open System
open System.IO

let ratedMovies = Directory.GetFiles @"NetFlixPrizeData\"

ratedMovies
|> Array.map processRatingsFile
|> Seq.concat
|> Seq.iter  combinedLog.WriteLine
```

This is not fun

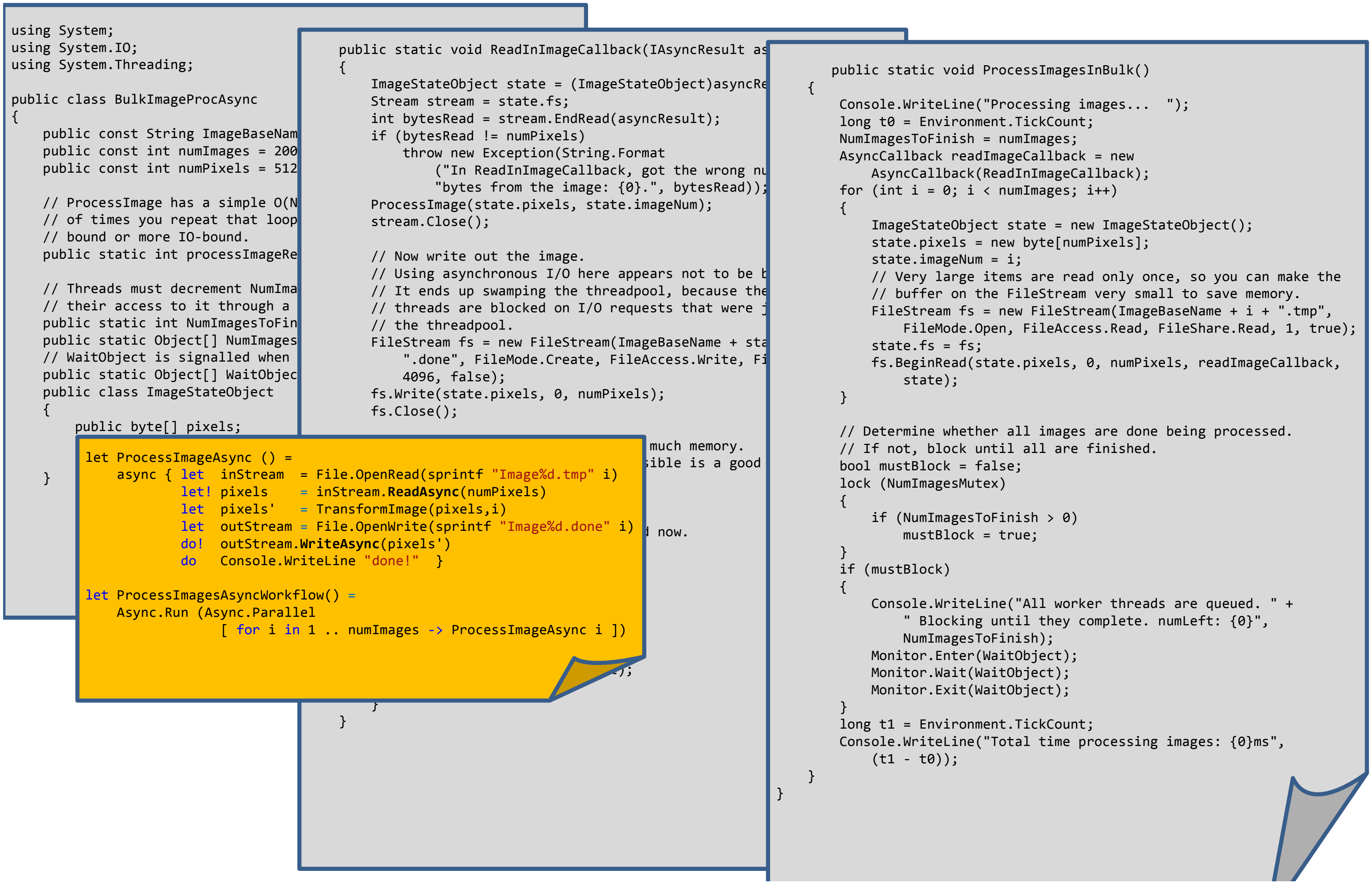


```
using System;
using System.IO;
using System.Threading;

public class BulkImageProcAsync
{
    public const String ImageBaseNam
    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 NumIma
    // their access to it through a
    public static int NumImagesToFin
    public static Object[] NumImages
    // WaitObject is signalled when
    public static Object[] WaitObjec
    public class ImageStateObject
    {
        public byte[] pixels;
        public int imageNum;
        public FileStream fs;
    }
```

```
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();

    // This application model uses too much memory.
    // Releasing memory as soon as possible is a good
    // especially global state.
    state.pixels = null;
    fs = null;
    // Record that an image is finished now.
    lock (NumImagesMutex)
    {
        NumImagesToFinish--;
        if (NumImagesToFinish == 0)
        {
            Monitor.Enter(WaitObject);
            Monitor.Pulse(WaitObject);
            Monitor.Exit(WaitObject);
        }
    }
}
```

```
        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 not, block until all are finished.
        bool mustBlock = false;
        lock (NumImagesMutex)
        {
            if (NumImagesToFinish > 0)
                mustBlock = true;
        }
        if (mustBlock)
        {
            Console.WriteLine("All worker threads are queued. " +
                " 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));
    }
}
```

This is fun



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

public class BulkImageProcAsync
{
    public const String ImageBaseNam
    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 NumIma
    // their access to it through a
    public static int NumImagesToFin
    public static Object[] NumImages
    // WaitObject is signalled when
    public static Object[] WaitObjec
    public class ImageStateObject
    {
        public byte[] pixels;


    }
```

```
    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.
                                                  sible is a good



                                                  d now.







                                                  );

        }
    }
```

```
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 ])
```

```
    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 not, block until all are finished.
        bool mustBlock = false;
        lock (NumImagesMutex)
        {
            if (NumImagesToFinish > 0)
                mustBlock = true;
        }
        if (mustBlock)
        {
            Console.WriteLine("All worker threads are queued. " +
                " 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));
    }
}
```

This is fun!

```
Async.Run
    (Async.Parallel
        [ GetWebPage "http://www.google.com";
          GetWebPage "http://www.live.com";
          GetWebPage "http://www.yahoo.com"; ]
```

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

This is fun too!

```
#r "Microsoft.ManagedDirectX.dll"
```

```
#r "System.Parallel.dll"
```

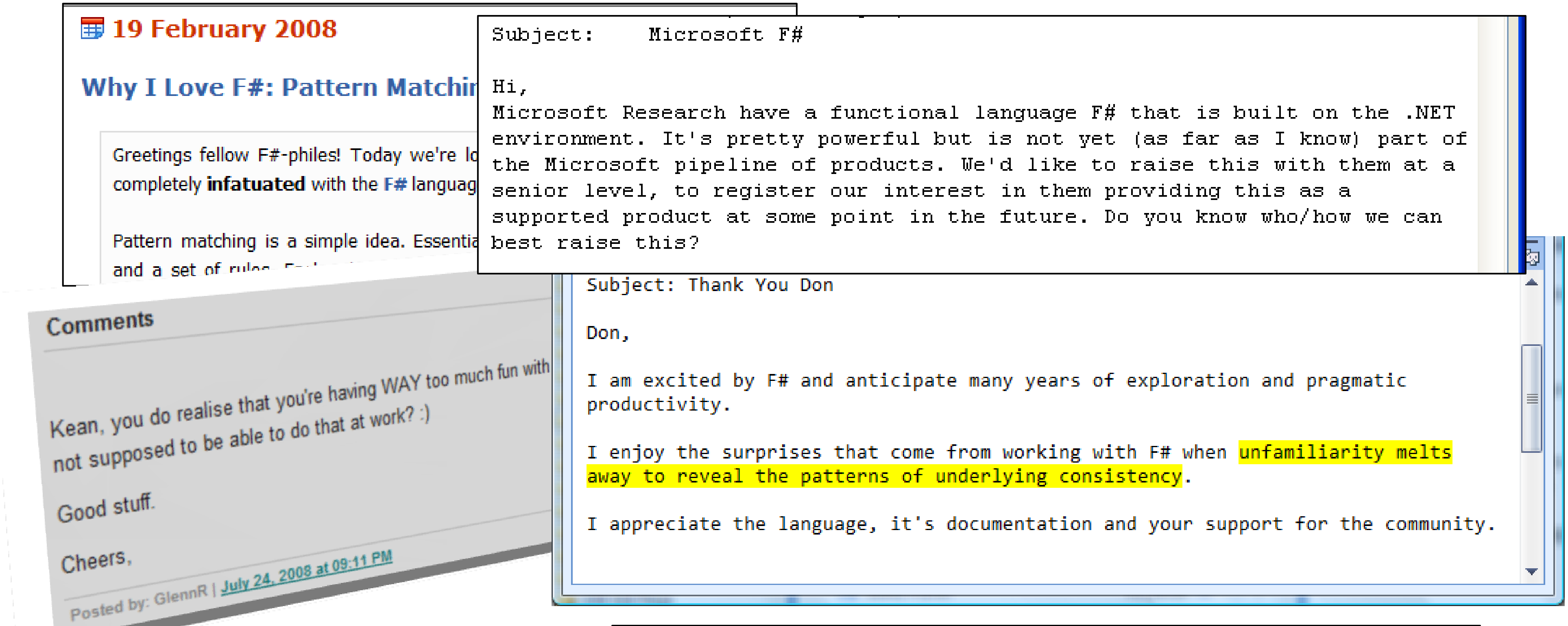
```
#r "System.Xml.dll"
```

```
#r "NUnit.Framework.dll"
```

```
#r "Xceed.Charting.dll"
```

```
#r "ExtremeOptimization.Math.dll"
```

Community fun



19 February 2008

Why I Love F#: Pattern Matchin

Greetings fellow F#-philes! Today we're lo
completely **infatuated** with the F# languag

Pattern matching is a simple idea. Essentia
and a set of rules

```
Subject:    Microsoft F#

Hi,
Microsoft Research have a functional language F# that is built on the .NET
environment. It's pretty powerful but is not yet (as far as I know) part of
the Microsoft pipeline of products. We'd like to raise this with them at a
senior level, to register our interest in them providing this as a
supported product at some point in the future. Do you know who/how we can
best raise this?
```

Comments

Kean, you do realise that you're having WAY too much fun with
not supposed to be able to do that at work? :)

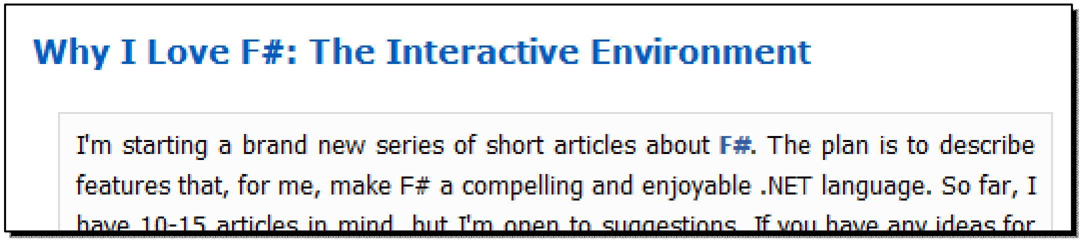
Good stuff.

Cheers,

Posted by: GlennR | July 24, 2008 at 09:11 PM

```
Subject: Thank You Don

Don,

I am excited by F# and anticipate many years of exploration and pragmatic
productivity.

I enjoy the surprises that come from working with F# when unfamiliarity melts
away to reveal the patterns of underlying consistency.

I appreciate the language, it's documentation and your support for the community.
```

It's the fastest genome assembly viewer I've ever seen and only 500 lines of F#. It's really an incredible language...

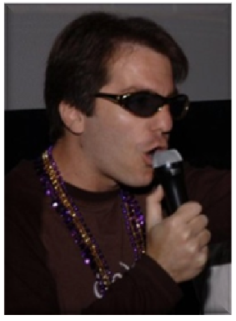


Why I Love F#: The Interactive Environment

I'm starting a brand new series of short articles about F#. The plan is to describe features that, for me, make F# a compelling and enjoyable .NET language. So far, I have 10-15 articles in mind, but I'm open to suggestions. If you have any ideas for

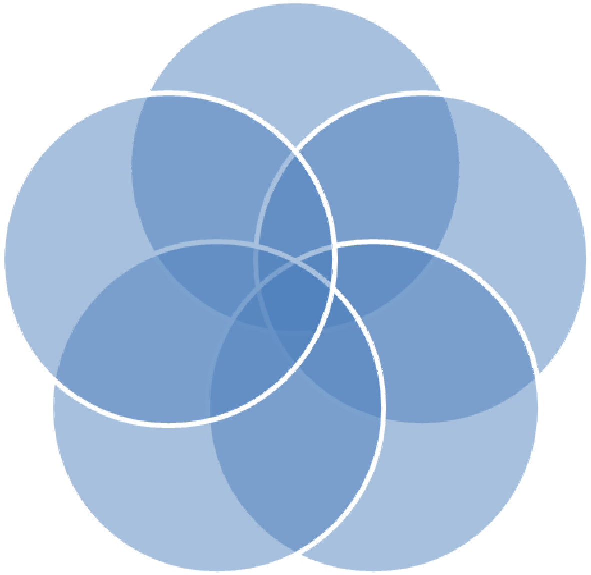
A Fantastic Team

- Developers
- QA
- Research/Architecture
- Program Managers
- Oversight



- +Joe,+Santosh,+James,+Baofa,+Sean,+Luca,+Tim,+Mike+Matteo
- The decision to bring F# to product quality was made and informed by a collective process involving:
  - Vice Presidents, Research leaders, Architects, Technical fellows, CTOs, Product Unit Managers, Developers, Testers, Researchers...

Team skills



Ocaml/F#

C/C++

C#/Linq

Haskell

OO

Fun

Microsoft is investing in functional programming because....

People want it
People like it
People are (in certain important domains) more productive with it

Summary

- Functional Programming **Brings Simplicity**
- Functional Programming with .NET makes **Business Sense**
- And it’s fun!