Microsoft

Research Faculty Summit 2012

ADVANCING THE STATE OF THE ART

Reconsidering Strongly Typed Programming for the Information-Rich World

16 July, 2012

Don Syme
Principal Researcher, Microsoft Research

F# and Open Source

F# 2.0 compiler+library open source

http://github.com/fsharp/fsharp

Apache 2.0 license

Runs on Mac, Linux, Windows, Browsers

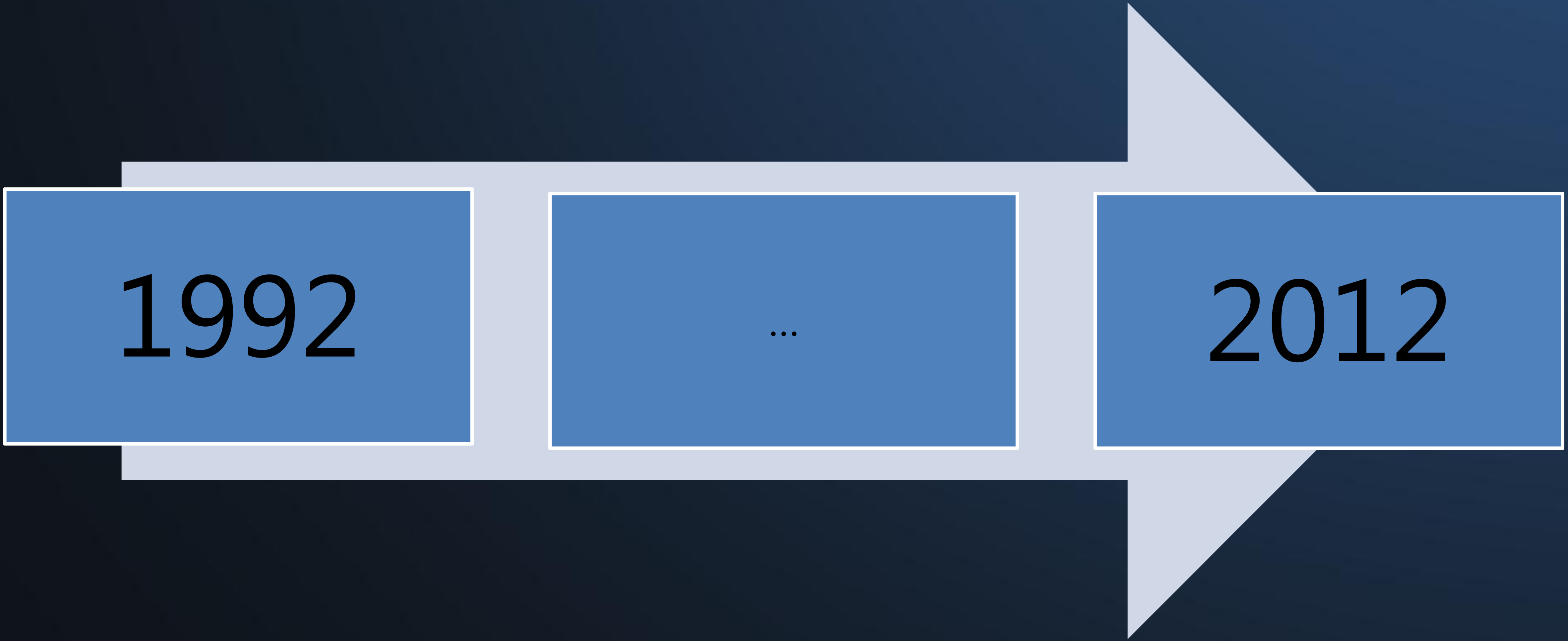
Free Open-Source IDE Tooling with MonoDevelop

F# 3.0 open source in preparation

Part 0

A quick retrospective

Time Warp…

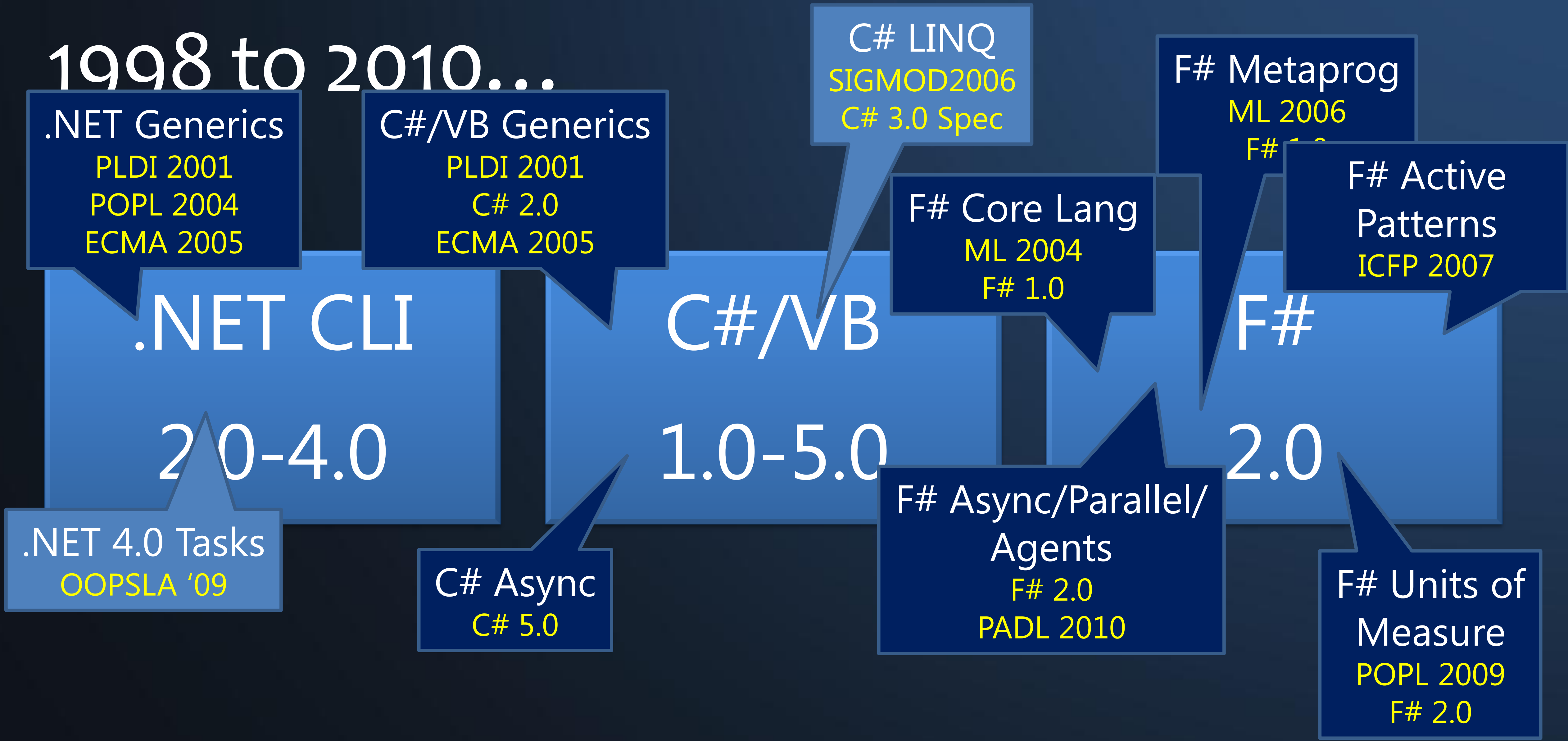


1992

…

2012

1998 to 2010…



.NET CLI

2.0-4.0

C#/VB

1.0-5.0

F#

2.0

.NET Generics
PLDI 2001
POPL 2004
ECMA 2005

C#/VB Generics
PLDI 2001
C# 2.0
ECMA 2005

C# LINQ
SIGMOD2006
C# 3.0 Spec

F# Metaprog
ML 2006
F# 1.0

F# Core Lang
ML 2004
F# 1.0

F# Active Patterns
ICFP 2007

.NET 4.0 Tasks
OOPSLA '09

C# Async
C# 5.0

F# Async/Parallel/Agents
F# 2.0
PADL 2010

F# Units of Measure
POPL 2009
F# 2.0

1998 to 2010…

The dark days of object fundamentalism are past us ☺

Mixed OO/FP languages are now industry standard

(C#, C++, VB, Java, Javascript, Scala, F#, …)

Microsoft and MSR have been instrumental in this transformation

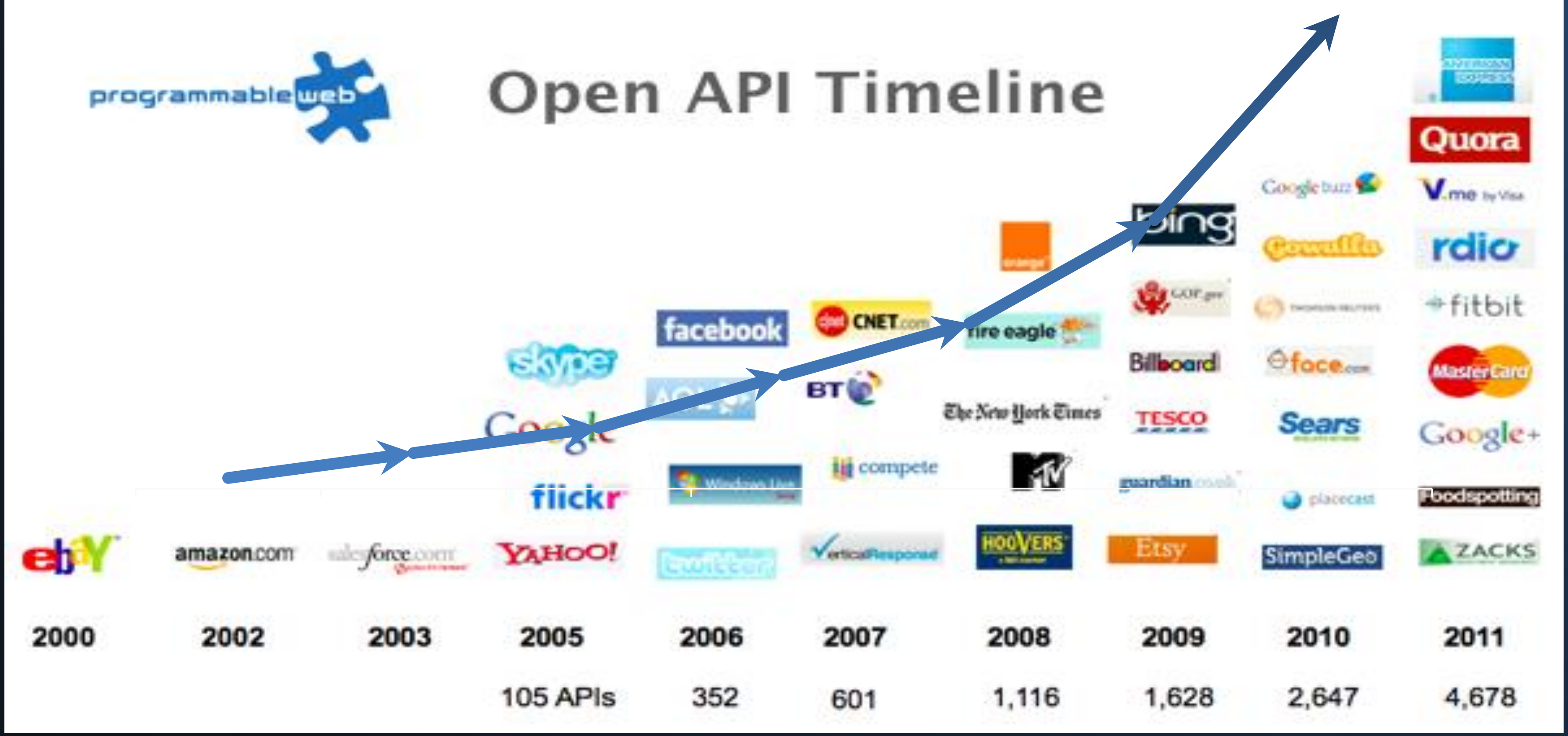
Part 1

Introducing F# and F# 3.0

Today’s talk is very simple

Proposition 1
The world is information-rich

The Information Revolution



Proposition 2
Our languages are information-sparse

Proposition 3
This is a big problem

(especially for strongly typed languages)

Proposition 4
F# 3.0 starts to fix this

Today

(1) Demonstrate F# 3.0

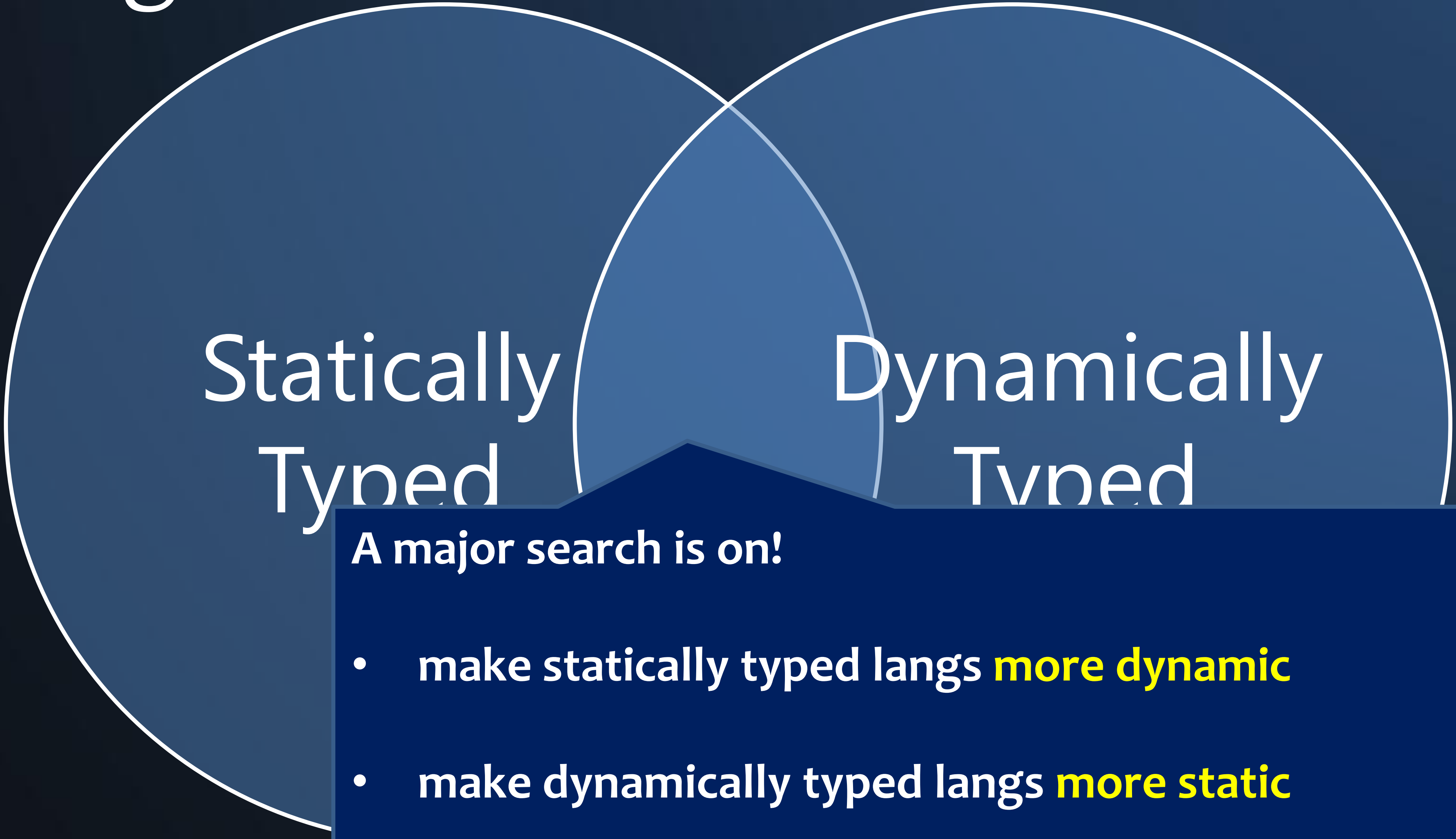
(2) Themes in Information Rich Programming

Paradigm Locator

Statically Typed

Dynamically Typed

Paradigm Locator



A major search is on!

- make statically typed langs **more dynamic**
- make dynamically typed langs **more static**
- **moderate static typing** in limited ways

But first, F# 1.0/2.0…

(Visual Studio 2008 & 2010)

F# is…

…a **productive, supported, interoperable, functional language** that allows you to write **simple code** to solve **complex problems.**

F#

```fsharp
let swap (x, y) = (y, x)



let rotations (x, y, z) =
    [ (x, y, z);
      (z, x, y);
      (y, z, x) ]




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

C#

```csharp
Tuple<U,T> Swap<T,U>(Tuple<T,U> t)
{
    return new Tuple<U,T>(t.Item2, t.Item1)
}

ReadOnlyCollection<Tuple<T,T,T>> Rotations<T>(Tuple<T,T,T>
{
  new ReadOnlyCollection<int>
   (new Tuple<T,T,T>[]
    {new Tuple<T,T,T>(t.Item1,t.Item2,t.Item3);
     new Tuple<T,T,T>(t.Item3,t.Item1,t.Item2);
     new Tuple<T,T,T>(t.Item2,t.Item3,t.Item1); });
}

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

Example (power company)

I have written **an application to balance the national power generation schedule** for a portfolio of power stations to a trading position 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 within enterprise software, namely **algorithmically complex analysis of large data sets**.

Simon Cousins (power company)

language taster

Fundamentals - Whitespace Matters

```
let computeDerivative f x =
    let p1 = f (x - 0.05)

  let p2 = f (x + 0.05)

       (p2 - p1) / 0.1
```

Offside (bad indentation)

Fundamentals - Whitespace Matters

```
let computeDerivative f x =
    let p1 = f (x - 0.05)

    let p2 = f (x + 0.05)

    (p2 – p1) / 0.1
```

F# - Objects + Functional

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

    let d2 = dx*dx+dy*dy

    member v.DX = dx

    member v.DY = dy

    member v.Length = sqrt d2

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

Inputs to object construction

Object internals

Exported properties

Exported method

Fundamentals - Queries

```
let avatarTitles =
    query { for t in netflix.Titles do
            where  (t.Name.Contains "Avatar")
            select  t }
```

LINQ Queries

Now, F# 3.0…

(now available from www.fsharp.net!)

A Challenge!

Task #1: A Chemistry Elements Class Library

Task #2: A Biology Class Library

Task #3: Repeat for all fields of human knowledge and endeavour…

Language Integrated Web Data

demo

Exploring Your World with Language Integrated World Bank Data

demo

F# Data Scripting for Hadoop and Hive

demo

A Type Provider is….

"Just like a library"

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

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

"Staged, on-demand type macros…"

Note: Language still contains no data

Open architecture

You can write your own type provider

Part 1 – Summary

At MSR our job is to take languages forward

F# 1.0-2.0 focused on FP+OO+Parallel+Units

F# 3.0 focuses on information integration

Part 2

Themes in Information Rich Programming

Recap: the problem…

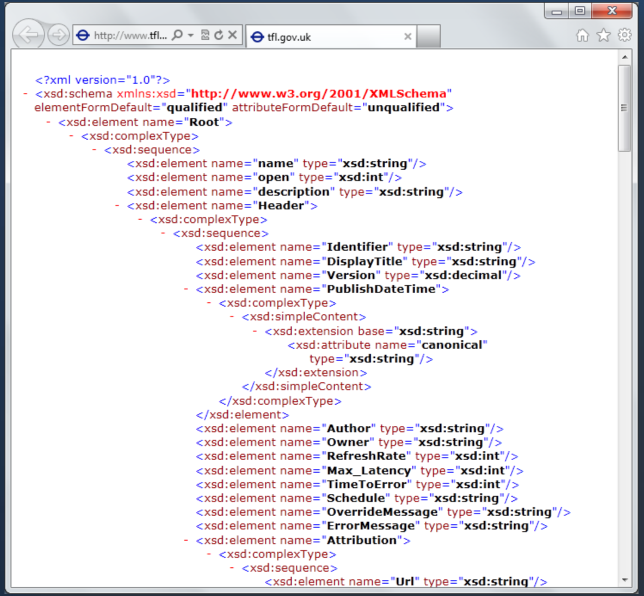
Languages do not integrate information

- Non-intuitive
- Not simple
- Disorganised
- Static
- High friction

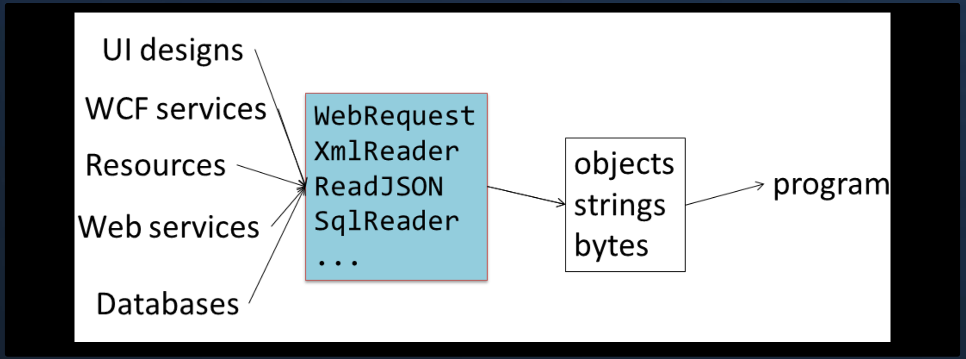
Recap: the problem…

Existing techniques have major problems

- Over-reliance on code-generation
- Do not scale
- No tooling
- No strong types
- Lowest-common-denominator



```
  <?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

Time for a paradigm shift?

The neglected child?

$$\Gamma \vdash \boldsymbol{e} : \tau$$

The great lie!!!

$$\Gamma_0 = 0$$

Definition #1

Information-rich programming is where external schematized information sources are integral to the operation of the programs being constructed.

These may be as simple as textual DSLs embedded as strings in the program itself, as familiar as SQL databases, as massive as a service exposing Wikipedia data, or the world-wide-web of HTML documents itself.

Definition #2

A *(strongly-typed) information-rich programming language* integrates external information sources, where the schema and content of these sources are presented in a (strongly-typed) idiomatic form

Theme #1

Big Data → Big Metadata

Strongly typed languages and tooling *must* have a role here, surely!

Theme #2

Huge Information Spaces can and should be viewed as Software Components

example:
schema change <-> component versioning <->
source compatibility <-> binary compatibility

Theme #3

Multiple Data Standards with One Simple Mechanism

Rich type systems, importing rich metadata where it exists

Theme #4

Measure languages by their effectiveness at working with rich external information spaces

Example: queries, JSON, XML, type providers, async…

Theme #5

Programming Type Systems v. Information Space Metadata Synergy or Conflict?

Examples: Types, Schema, Constraints, Units of Measure, Security Information, Documentation, Definition Locations, Help , Provenance, Privacy, Ratings, Rankings, Search…

Some Sample Research Questions

Can we design type systems which incorporate schema change policies?

Can we automatically provide all the data in the enterprise?

Can we provide and verify richer constraints?

Can security, privacy and provenance annotations be provided?

Can provided probabilistic metadata be useful for tooling?

Can we automatically find bugs in provider components?

Can we plug and play more language logic?

Can we usefully provide massive quantities of geo data + geo metadata?

Summary

The world is information rich, our programming needs to be information-rich too

Big Data → Big Metadata

Information-richness changes how we think about strongly-typed programming languages

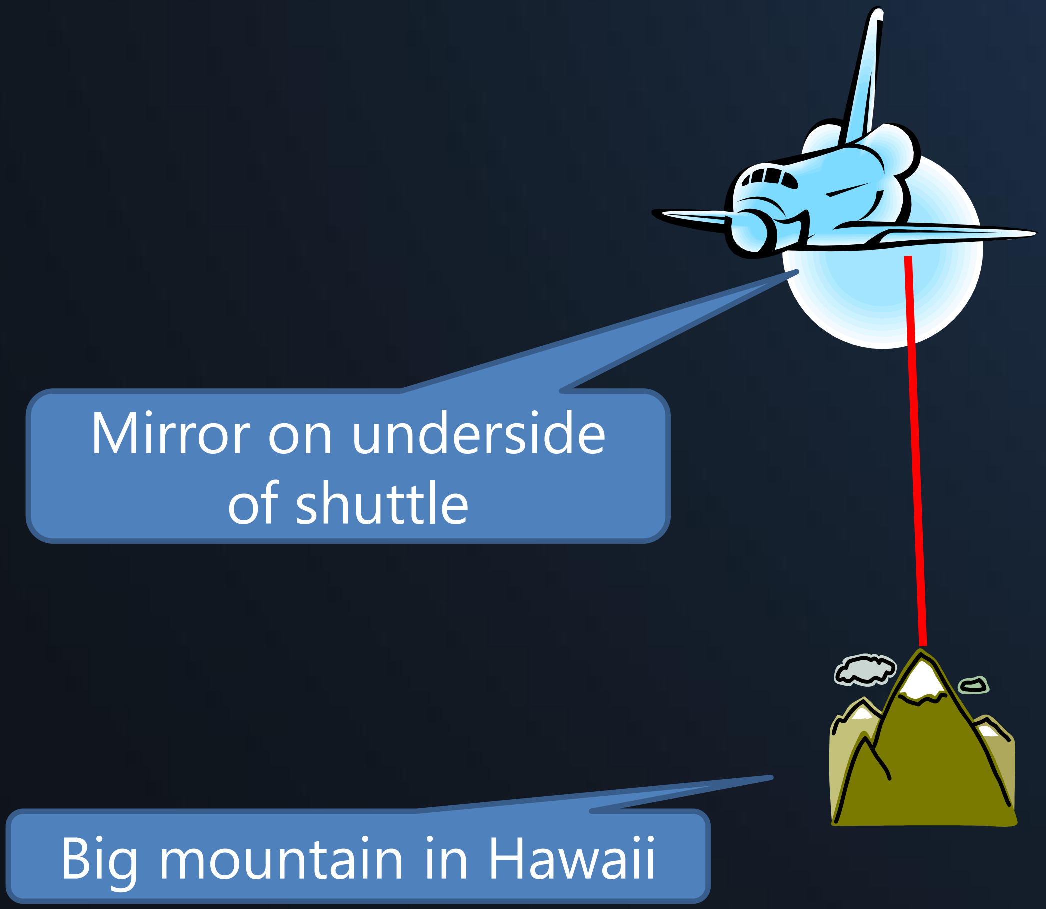
Thank you!

Questions?

dsyme@microsoft.com
@dsyme

www.fsharp.net
#fsharp

1985

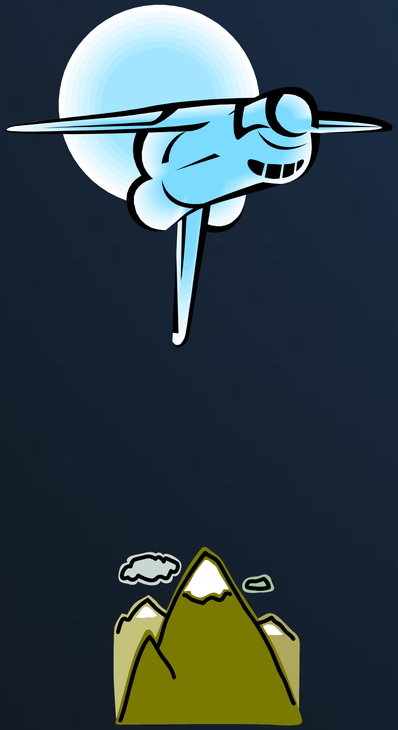


Mirror on underside of shuttle

Big mountain in Hawaii

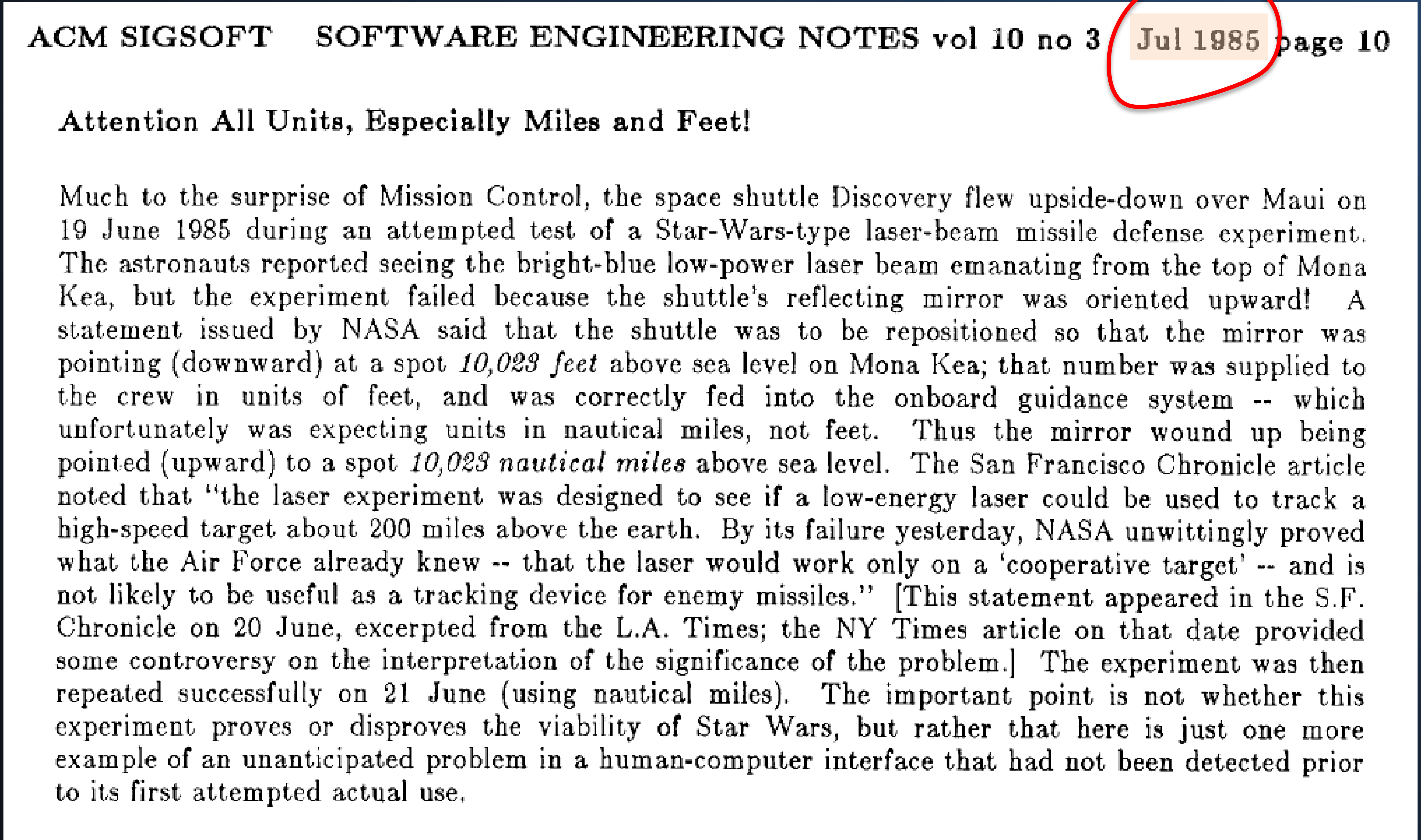
SDI experiment:
The plan

1985



SDI experiment:
The reality

1985



ACM SIGSOFT SOFTWARE ENGINEERING NOTES vol 10 no 3 Jul 1985 page 10

Attention All Units, Especially Miles and Feet!

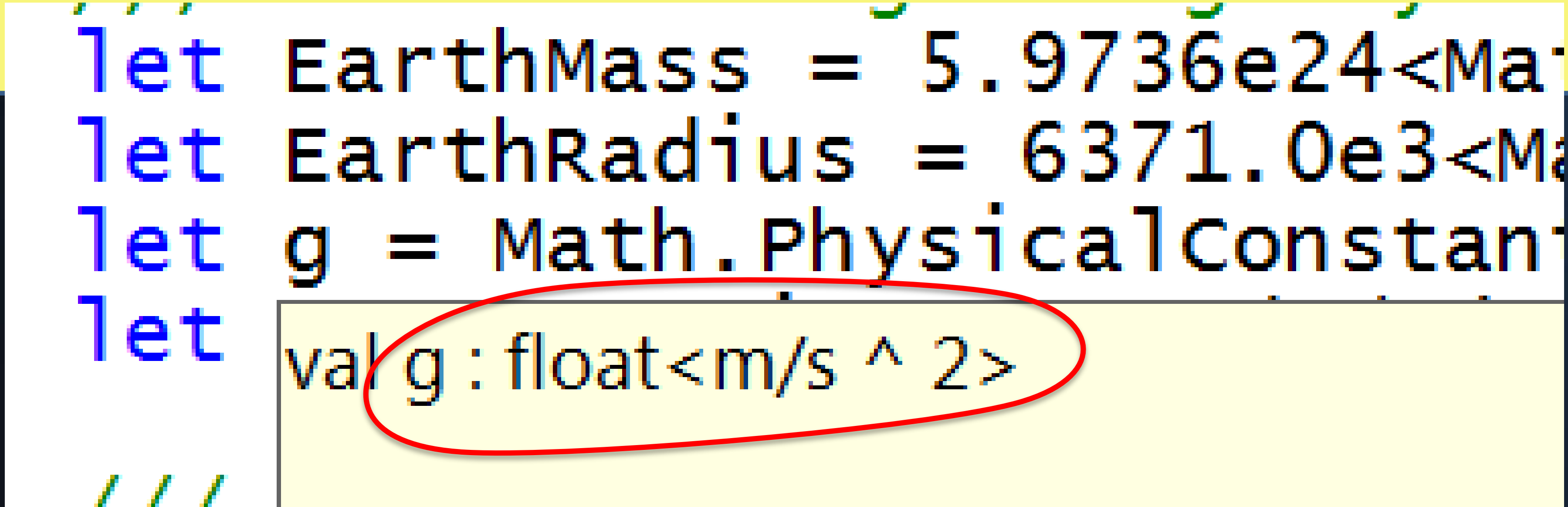
Much to the surprise of Mission Control, the space shuttle Discovery flew upside-down over Maui on 19 June 1985 during an attempted test of a Star-Wars-type laser-beam missile defense experiment. The astronauts reported seeing the bright-blue low-power laser beam emanating from the top of Mona Kea, but the experiment failed because the shuttle's reflecting mirror was oriented upward! A statement issued by NASA said that the shuttle was to be repositioned so that the mirror was pointing (downward) at a spot *10,023 feet* above sea level on Mona Kea; that number was supplied to the crew in units of feet, and was correctly fed into the onboard guidance system -- which unfortunately was expecting units in nautical miles, not feet. Thus the mirror wound up being pointed (upward) to a spot *10,023 nautical miles* above sea level. The San Francisco Chronicle article noted that "the laser experiment was designed to see if a low-energy laser could be used to track a high-speed target about 200 miles above the earth. By its failure yesterday, NASA unwittingly proved what the Air Force already knew -- that the laser would work only on a 'cooperative target' -- and is not likely to be useful as a tracking device for enemy missiles." [This statement appeared in the S.F. Chronicle on 20 June, excerpted from the L.A. Times; the NY Times article on that date provided some controversy on the interpretation of the significance of the problem.] The experiment was then repeated successfully on 21 June (using nautical miles). The important point is not whether this experiment proves or disproves the viability of Star Wars, but rather that here is just one more example of an unanticipated problem in a human-computer interface that had not been detected prior to its first attempted actual use.

Units of Measure

```
let EarthMass = 5.9736e24<kg>

// Average between pole and equator radii
let EarthRadius = 6371.0e3<m>

// Gravitational acceleration on surface of Earth
let g = PhysicalConstants.G * EarthMass / (EarthRadius * EarthRadius)
```



```
let EarthMass = 5.9736e24<Ma
let EarthRadius = 6371.0e3<Ma
let g = Math.PhysicalConstan
let
```

```
val g : float<m/s ^ 2>
```

Microsoft