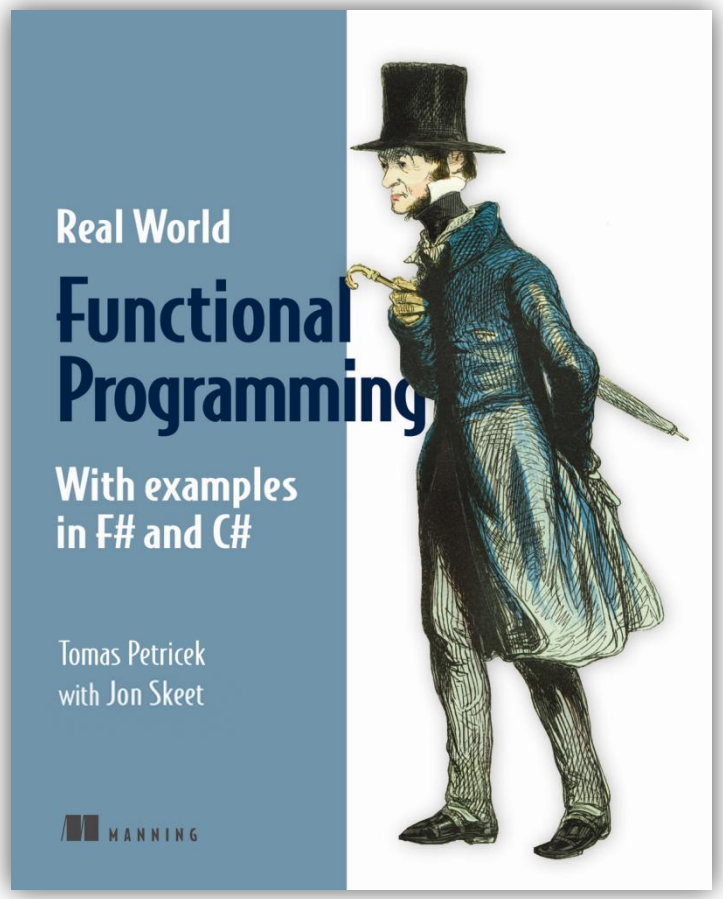
Asynchronous programming on the **server** and the **client** in **F#**

Tomas Petricek
@tomaspetricek

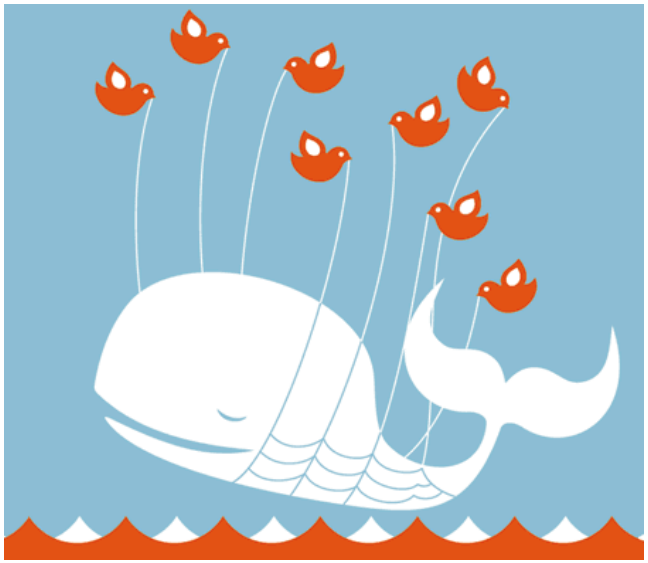
QCon


Real World
Functional Programming
With examples in F# and C#
Tomas Petricek
with Jon Skeet
MANNING

In **Visual Studio** since 2010

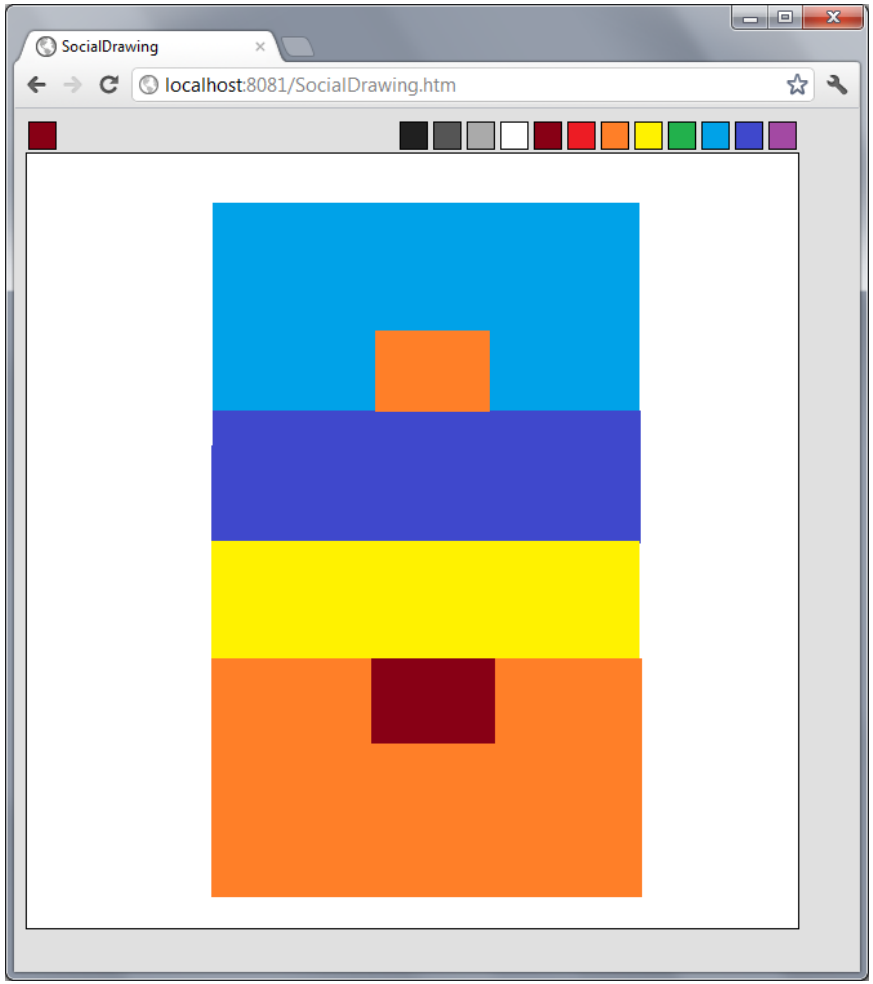
Asynchronous programming

On the **server side**

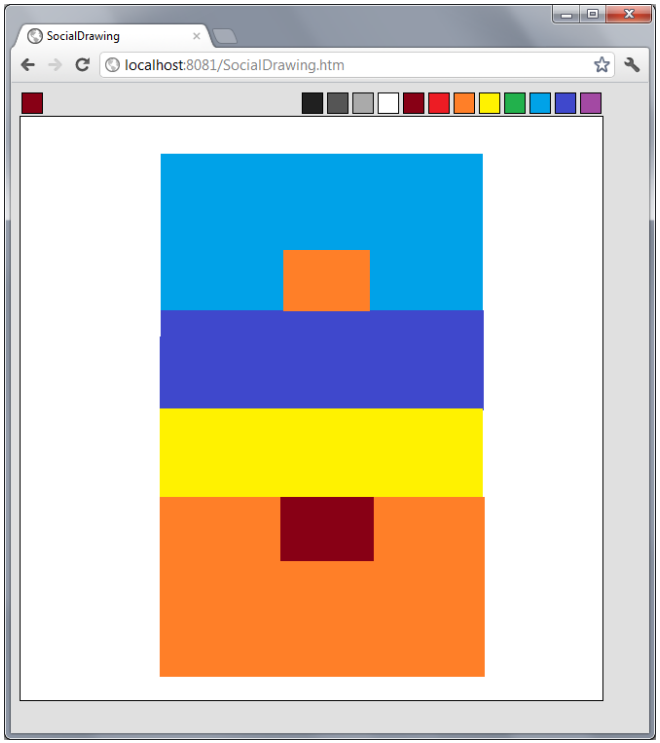


On the **client side**

Demo: Social drawing app



Demo: Social drawing app



Server → Agents → Browser → Client

Async on the Server

Reactive programming without the **inversion of control**

Async on the Server

Reactive model is important

- **Node.js** and **C# 5.0**

F# asynchronous workflows

- Keep standard **programming model**
- Standard **exception handling** and **loops**
- **Sequential** and **parallel** composition

Agents and message-passing

Protected * (**Behaviour** + **State**)

F# and the Browser

F# and Silverlight

- Both **compiler** and **libraries**
- Interactive **Try F#**

F# and JavaScript

- Translating since 2006!
- Open-source **Pit**, commercial **WebSharper**

Event handling in F#

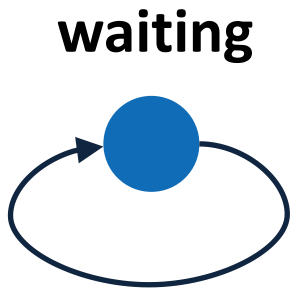
Data flow using combinators
and **control flow** using async

Asynchronous GUI

User interactions = **State machines**

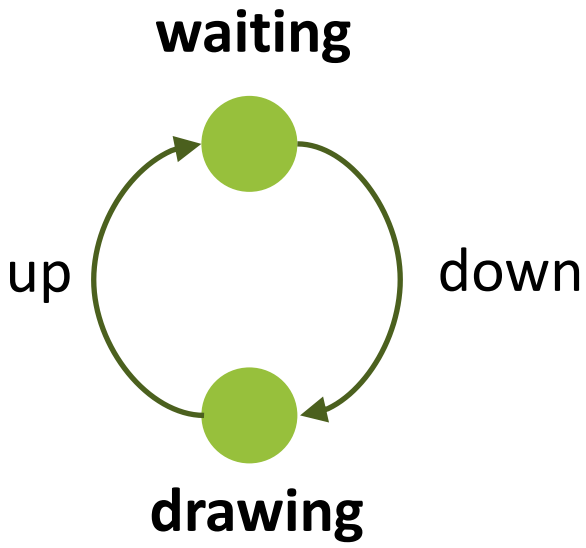
Asynchronous GUI

Updating rectangles



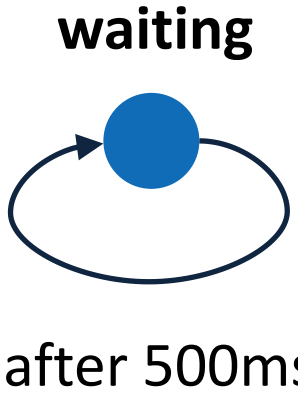
after 500ms

Drawing rectangles

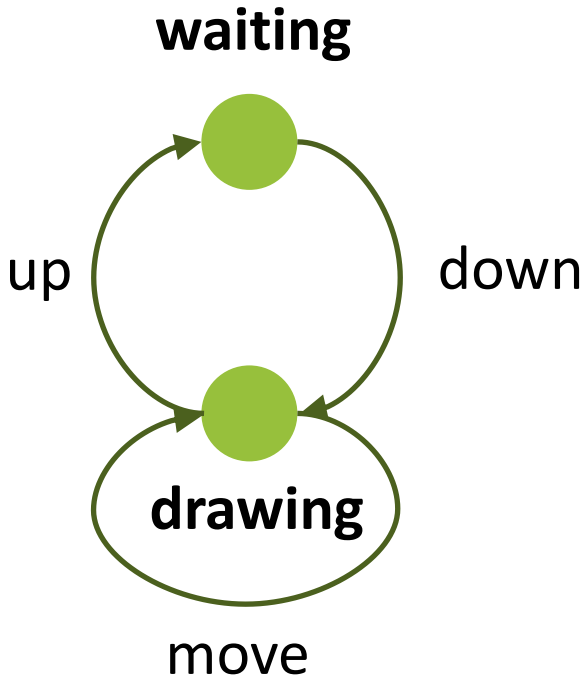


Asynchronous GUI

Updating rectangles



Drawing rectangles



What else is there?

F# Interactive in your web browser

- **www.tryfsharp.org**

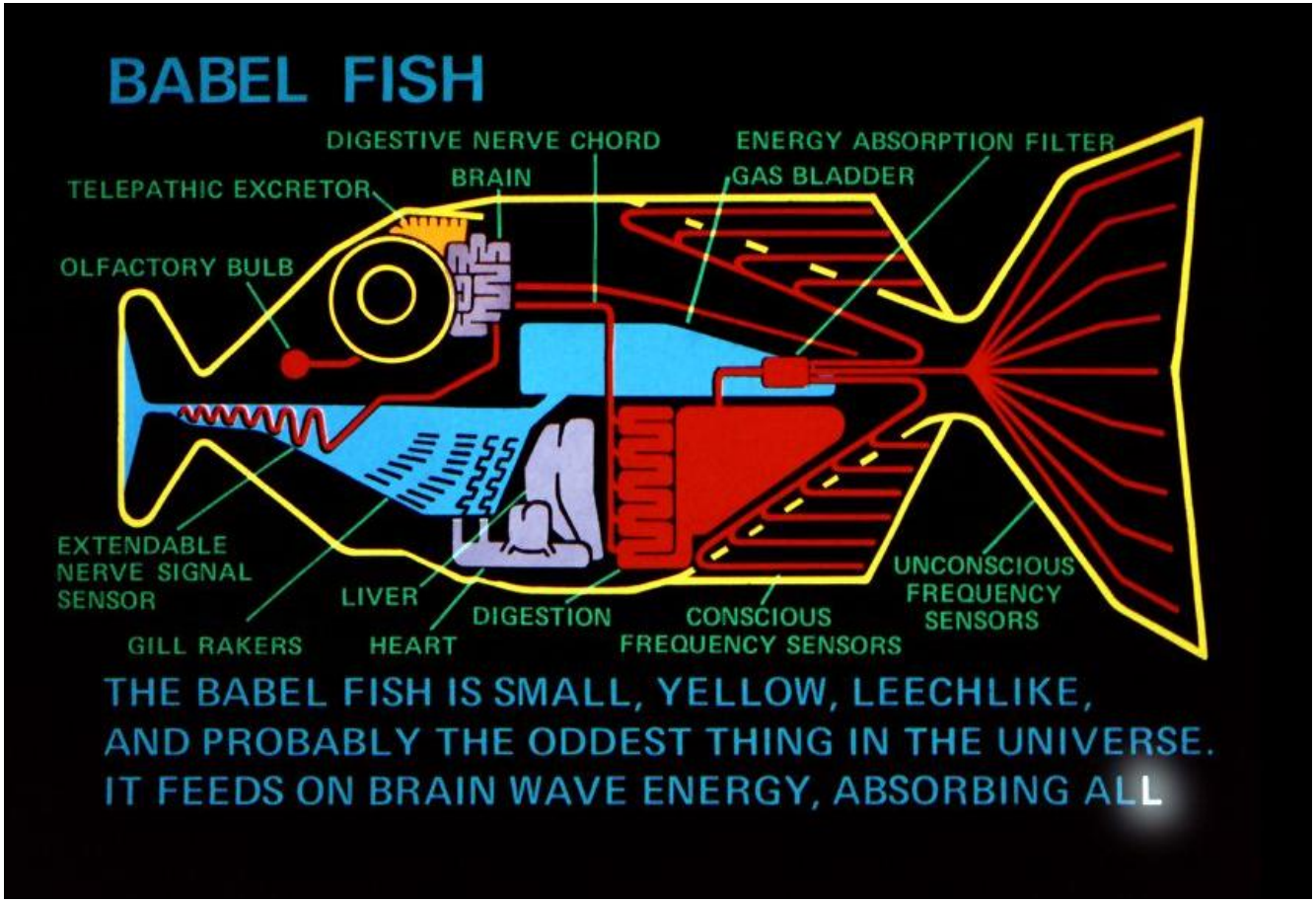
Type providers in F# 3.0

- Integrating **data** in the **language**

Bridges an important mismatch

- Data and services use **REST**, **XML**, …
- Languages use **types** and **objects**

Type providers



Where to learn more?

Functional and F# trainings

http://functional-programming.net

In London and New York

Functional Programming eXchange

http://skillsmatter.com

Next Friday (March 16th)

Summary

Asynchronous programming

- Writing **non-blocking** code
- **Without** the inversion of control

Application areas

- **Server-side** – reactive request processing
- **Client-side** – encoding state machines