Taking Over the World with F# Agents

Tomas Petricek

tomas@tomasp.net | @tomaspetricek

Conspirator behind http://fsharp.org

software stacks

trainings teaching F# user groups snippets

mac and linux cross-platform books and tutorials

F# Software Foundation

F# community open-source MonoDevelop

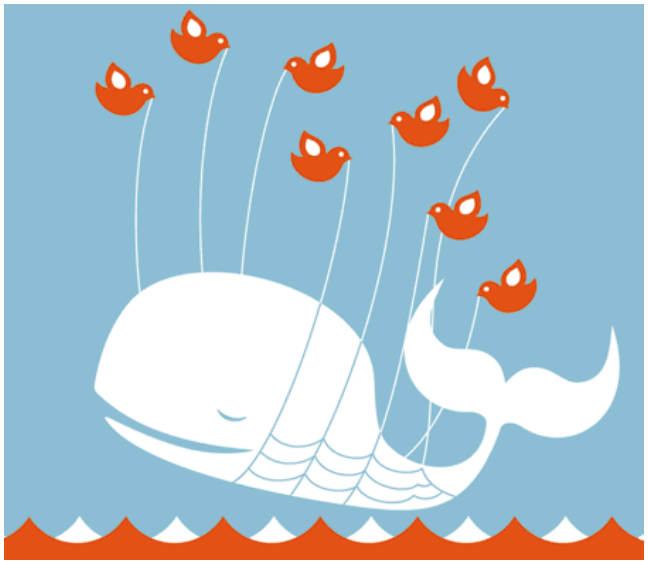
http://www.fsharp.org

contributions research support

consultancy mailing list

Asynchronous programming

On the **server side**



On the **client side**

Async GUI programming

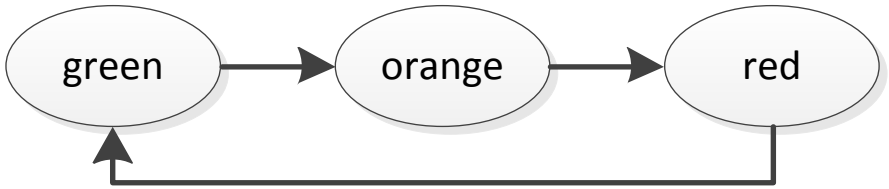
Controlling traffic light

Using **int** or **enum** to keep current state?

But – what does the state represent?

Better using **asynchronous waiting**

Loop that *asynchronously waits* for transitions



F# to JavaScript

Before it was cool.

F# to JavaScript

TypeScript type provider

- Import types for JS libraries
- Somebody else writes them!

Libraries & frameworks

- **Open source:** **FunScript** and Pit
- **Commercial:** IntelliFactory **WebSharper**

DEMO: Traffic lights

Writing loops using workflows

Using standard language constructs

```
let semaphoreStates() = async {
  while true do
    for current in [ green; orange; red ] do
      let! md = Async.AwaitEvent(this.MouseDown)
      display(current) }
```

Infinite loop!

Wait for click

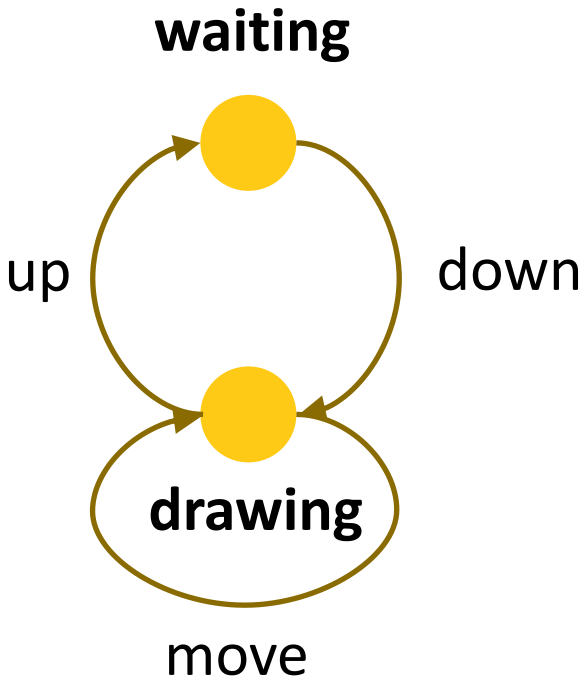
Key idea – asynchronous waiting

C# – events are not first-class values

F# – can use functional style (recursion)

Drawing rectangles

Describes how **drag & drop** works



DEMO: Drawing rectangles

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

Synchronous

Asynchronous

Synchronous code

```
var wc = new WebClient();
var html = wc.DownloadData(url);
outputStream.WriteAsync(html)
```

Easy to write

Can use **loops** and **exception handling**

Blocks thread and **does not scale!**

Event-based code

```
var wc = new WebClient();
wc.DownloadDataCompleted += (s, e) =>
  outputStream.BeginWrite
    ( e.Result, 0, e.Result.Length,
      res => outputStream.EndRead(res), null);
wc.DownloadDataAsync(url);
```

Makes code more **scalable**

Impossible to **read & write** by mere mortals

Can be surprisingly popular (**Node.js**)

(A)synchronous code

```
var wc = new WebClient();
var html = await wc.DownloadStringTaskAsync(url);
await outputStream.WriteAsync()
```

Easy to **change**, easy to **write**

Can use **loops** and **exception handling**

Scalable – no blocking of threads

Async workflows in F# and C#

C# – Return `Task` and add `async` and `await`

```
async Task<int> PageLength(Uri url) {
  var wc = new WebClient();
  var html = await wc.DownloadStringTaskAsync(url);
  return html.Length;
}
```

F# – Wrap in `async` and use `let!` keyword

```
let pageLength (url:Uri) = async {
  let wc = new WebClient();
  let! html = wc.AsyncDownloadString(url)
  return html.Length }
```

DEMO: Asynchronous proxy

Agent-based programming

Program consists of agents

- **Lightweight** – can create lots of them
- Written using **asynchronous** workflows

Agents communicate via messages

- Communication is **thread-safe**
- Messages are **queued**

Enables parallelism

- **Different agents** vs. **multiple instances**

Simple agent in F#

Send “Hello” to the caller

```
let echo = Agent.Start(fun agent -> async {
  while true do
    let! name, rchan = agent.Receive()
    rchan.Reply("Hello " + name) })
```

Waiting for message is asynchronous

Can perform long-running I/O before replying

Calling agent asynchronously

```
let! str = echo.PostAndAsyncReply(fun ch -> "Tomas", ch)
```

DEMO: Proxy with caching

Multi-state agents

Accepting all messages

- Asynchronously `Receive` and use pattern matching

Waiting for a specific message

- Other messages stay in the queue

```
Agent.Start(fun agent ->
  let rec blocked = agent.Scan (function
    | Resume -> Some (async {
        printfn "Resumed!"
        return! running })
    | _ -> None)
  and running = (* ... *) )
```

Run workflow when Resume arrives

No response specified for other messages

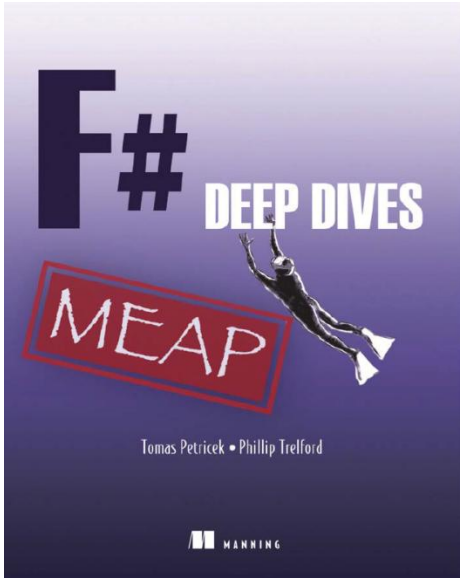
DEMO: Pausing Twitter

DEMO: Batch processing

New books & trainings

F# **Trainings** & Progressive **Tutorials**

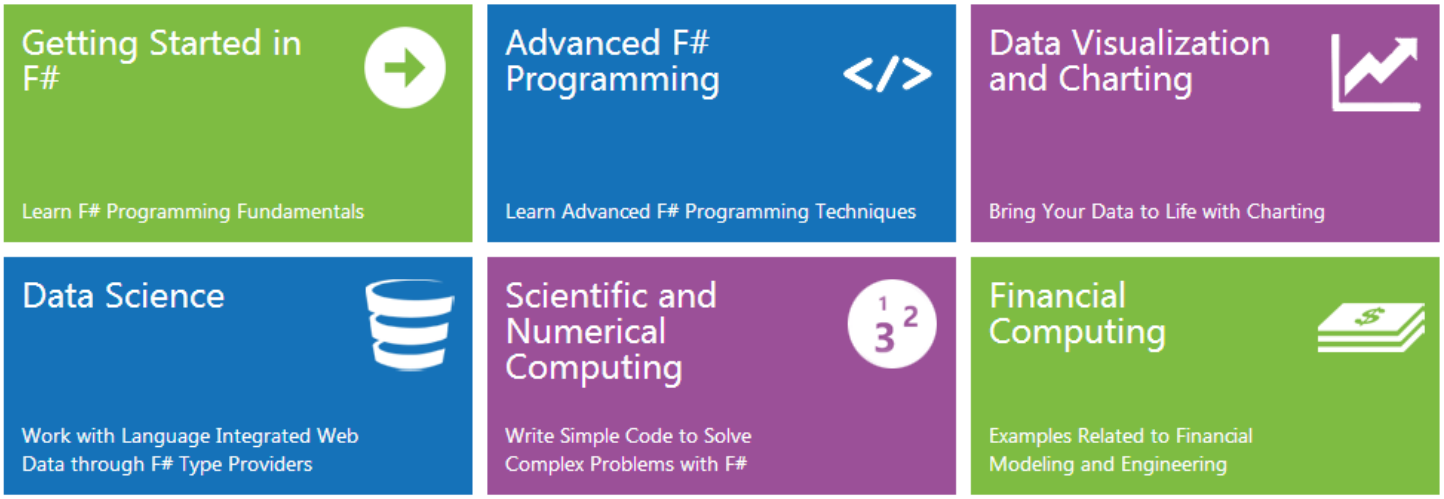
In London and **New York**
Get in touch: tomas@tomasp.net



F# Deep Dives book

Concurrency (Gaming, finance)
Business logic (Insurance, …)
Data (Research, Machine learning)
Testing (Finance, …)
Early access at: http://manning.com/petricek2

Online resources



www.fsharp.org — Information & community

www.tryfsharp.org — Interactive F# tutorials

Summary

- First-class async support
  - Available in F# now (and in C# 5.0)
- Allows interesting patterns
  - Both server-side and client-side
- F# has a few more things (agents, …)
  - Easy interoperability from C#