iOS Mobile App Development with Xamarin and F#

Rachel Reese

@rachelreese

rachelree.se, github: rachelreese

F# basics - Immutability

```
let x = 1
```


x is a value, not a variable,

because F# is immutable by default.

F# basics - Immutability

```
let x = 1
```

Reasonable code 

```
x = x + 1
```

F# basics - Immutability

```
let x = 1
```

Reasonable code 

```
x = x + 1
```

F# REPL output

```
> val it : bool = false
```

F# basics - Immutability

```
let x = 1
```

Reasonable code → `x = x + 1` ← Awful math!

```
> val it : bool = false
```

F# basics - Pipelining, higher-order functions

```
[1..10]
    |> List.filter (fun x -> x % 2 = 0)
    |> List.map (fun x -> x + 3)
    |> List.sum
```

iOS in F# Basics, significant whitespace

```
[<Register ("AppDelegate")>]
type AppDelegate () =
    inherit UIApplicationDelegate ()

    override val Window = null with get, set

    // This method is invoked when the application is ready to run.
    override this.FinishedLaunching (app, options) =
        this.Window <- new UIWindow (UIScreen.MainScreen.Bounds)
        this.Window.RootViewController <- new MinesweeperViewController ()
        this.Window.MakeKeyAndVisible ()
        true

module Main =
    [<EntryPoint>]
    let main args =
        UIApplication.Main (args, null, "AppDelegate")
        0
```

Why F#?

“> @davetchepak What can C# do that F# cannot? NullReferenceException :-)”

Tomas Petricek

Founding member of F# Software foundation; F# MVP

Why F#?: Object hierarchies in C#

```
public abstract class Transport{ }

public abstract class Car : Transport {
  public string Make { get; private set; }
  public string Model { get; private set; }
  public Car (string make, string model) {
    this.Make = make;
    this.Model = model;
  }
}

public abstract class Bus : Transport {
  public int Route { get; private set; }
  public Bus (int route) {
    this.Route = route;
  }
}

public class Bicycle: Transport { } ...
```

Why F#?: Discriminated unions

```
type Transport =
    | Car of Make:string * Model:string
    | Bus of Route:int
    | Bicycle
```

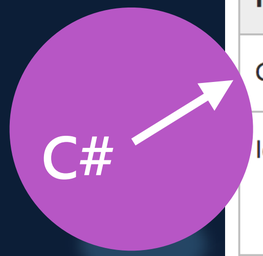
Same information as C#!

Why F#?: Discriminated unions

```
type Transport =
    | Car of Make:string * Model:string
    | Bus of Route:int
    | Bicycle
```

Same information as C#!

Trivial to pattern match on!



Name	Description	Example
Constant pattern	Any numeric, character, or string literal, an enumeration constant, or a defined literal identifier	`1.0`, `"test"`, `30`, `Color.Red`
Identifier pattern	A case value of a discriminated union, an exception label, or an active pattern case	`Some(x)` `Failure(msg)`
Variable pattern	*identifier*	`a`
as pattern	*pattern* as *identifier*	`(a, b) as tuple1`
OR pattern	*pattern1* | *pattern2*	`([h] | [h; _])`
AND pattern	*pattern1* & *pattern2*	`(a, b) & (_, "test")`
Cons pattern	*identifier* :: *list-identifier*	`h :: t`
List pattern	[*pattern_1*; ... ; *pattern_n*]	`[ a; b; c ]`
Array pattern	[| *pattern_1*; ..; *pattern_n*]	`[| a; b; c |]`
Parenthesized pattern	(*pattern*)	`( a )`
Tuple pattern	(*pattern_1*, ... , *pattern_n*)	`( a, b )`
Record pattern	{ *identifier1* = *pattern_1*; ... ; *identifier_n* = *pattern_n* }	`{ Name = name; }`
Wildcard pattern	_	`_`
Pattern together with type annotation	*pattern* : *type*	`a : int`
Type test pattern	:? *type* [as *identifier*]	`:? System.DateTime as dt`
Null pattern	null	`null`

Why F#?: Pattern matching

```
type Transport =
    | Car of Make:string * Model:string
    | Bus of Route:int
    | Bicycle
```

```
let getThereVia (transport:Transport) =
    match transport with
       | Car (make,model) -> ()
       | Bus route -> ()
```

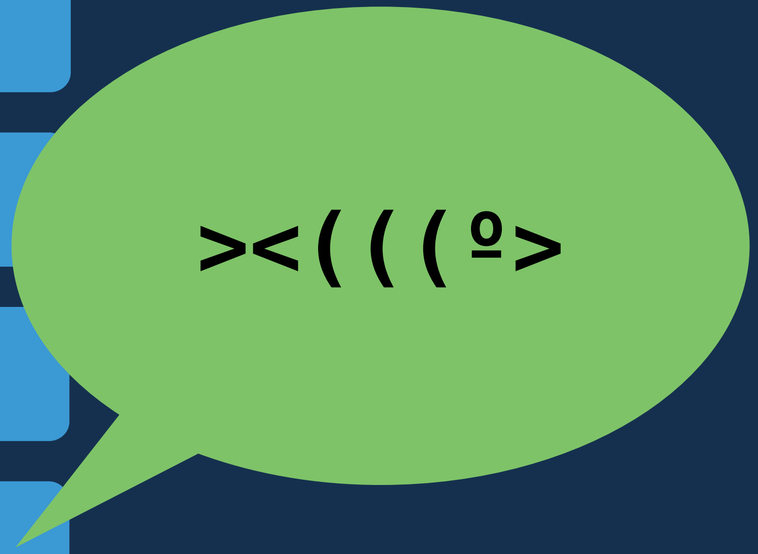
```
Warning FS0025: Incomplete pattern matches on this expression. For example,
the value 'Bicycle' may indicate a case not covered by the pattern(s).
```

Several reasons you should learn F#

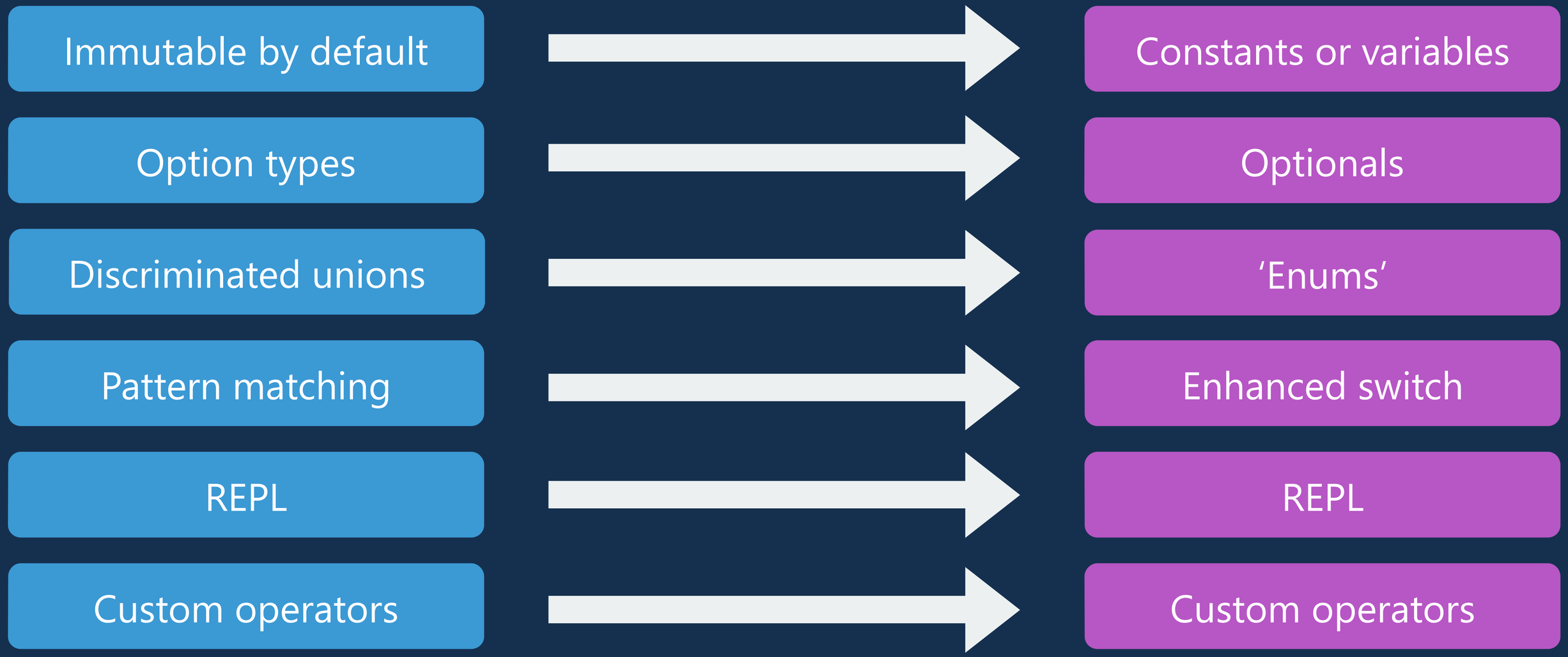
- Immutable by default
- Option types
- Discriminated unions
- Pattern matching
- REPL
- Custom operators

Several reasons you should learn F#

- Immutable by default
- Option types
- Discriminated unions
- Pattern matching
- REPL
- Custom operators



F# vs. Swift


Immutable by default
Option types
Discriminated unions
Pattern matching
REPL
Custom operators
Constants or variables
Optionals
‘Enums’
Enhanced switch
REPL
Custom operators

But Swift is still missing out!

No type providers	No units of measure	Lacks full type inference	Requires “return”
Still void, no unit	No parallelism and async!	Single platform	Young language
No easy cloning syntax for structures (e.g. records)	Literal data structures are only list and dict	Simpler garbage collection process	Active patterns

Xamarin.Forms

YES

Can you use Xamarin.Forms with F#?

Demo

Xamarin.Forms Hello World

Type providers

What are type providers?

- A mechanism to provide typed info to the compiler from an external data source

Why do we need them?

- Current data consumption methods
- Always in sync with the source
- Intellisense, tooltips, and other IDE tooling available
- No code generation
- Scalable to millions of types

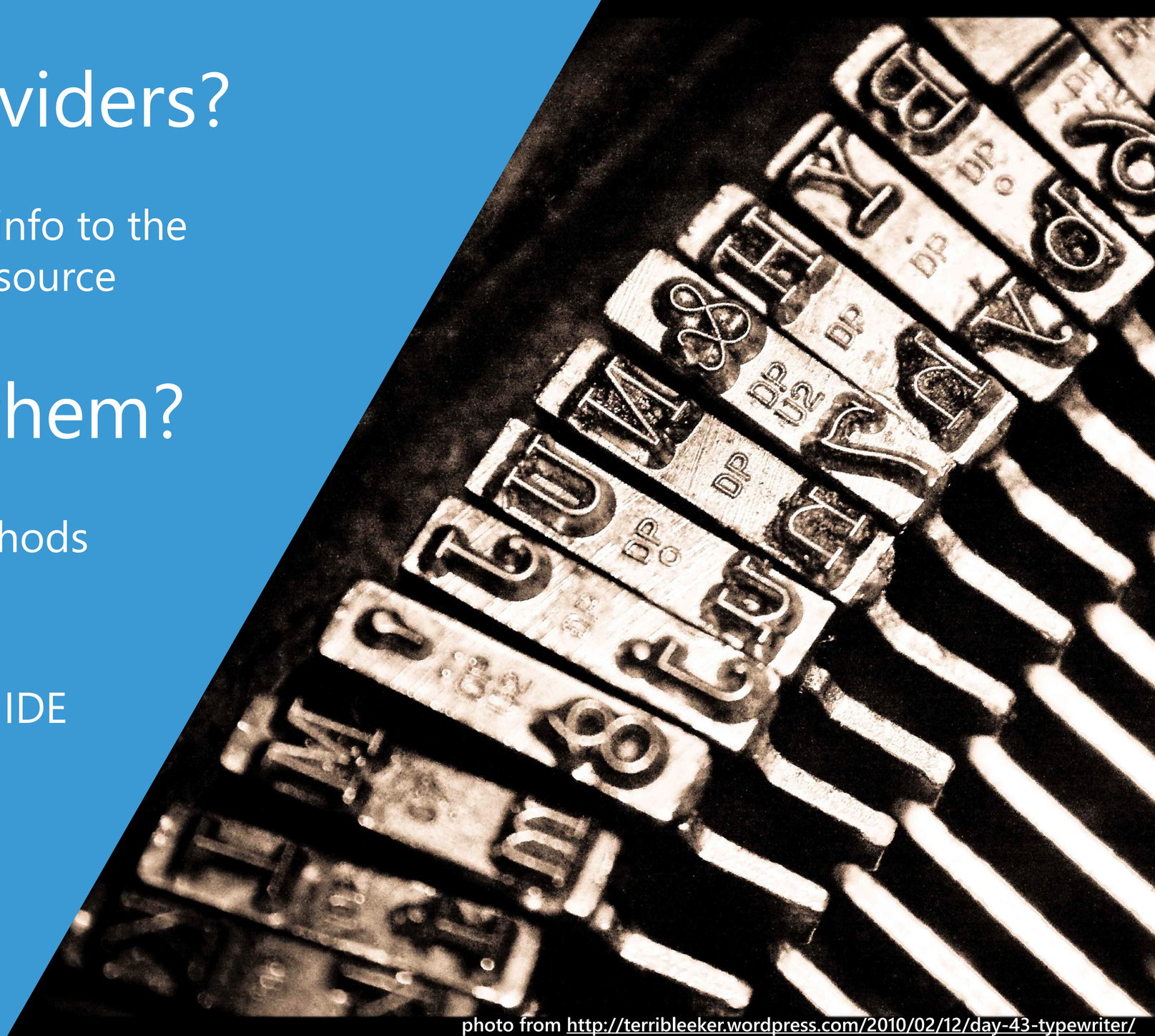


photo from http://terribleeker.wordpress.com/2010/02/12/day-43-typewriter/

Demo

Type Providers!

A handful of existing type providers

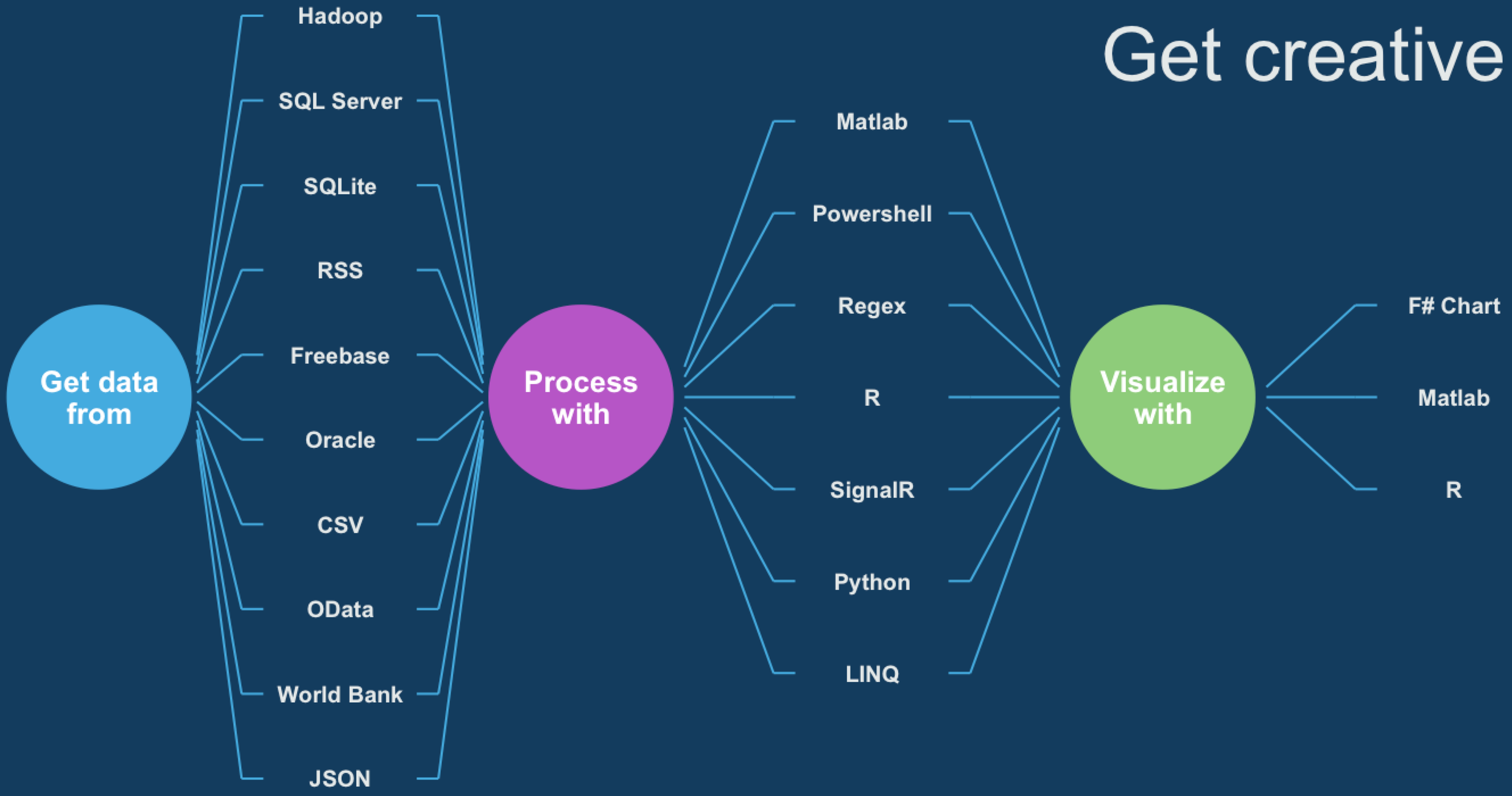
Minesweeper
Choose your own adventure
Azure
MVVMCross
Matlab
Rock, Paper, Scissors
Oracle
FunScript
File System
SignalR
Python
World Bank
Squirrels
SQLite!
XML
LINQ
MS Dynamics CRM
RSS
SQL Server
Dates
SQL Server with EF
XAML
Freebase
Hadoop
OData
Apiary
COM
JSON
Powershell
R
Fizzbuzz
Amazon S3
IKVM
Regex
WSDL
Don Syme
Facebook
CSV

“"There's a type provider for that" is the #fsharp equivalent of "There's an app for that" :-)”

Yan Cui

Lead Server Engineer, Gamesys

Get creative.



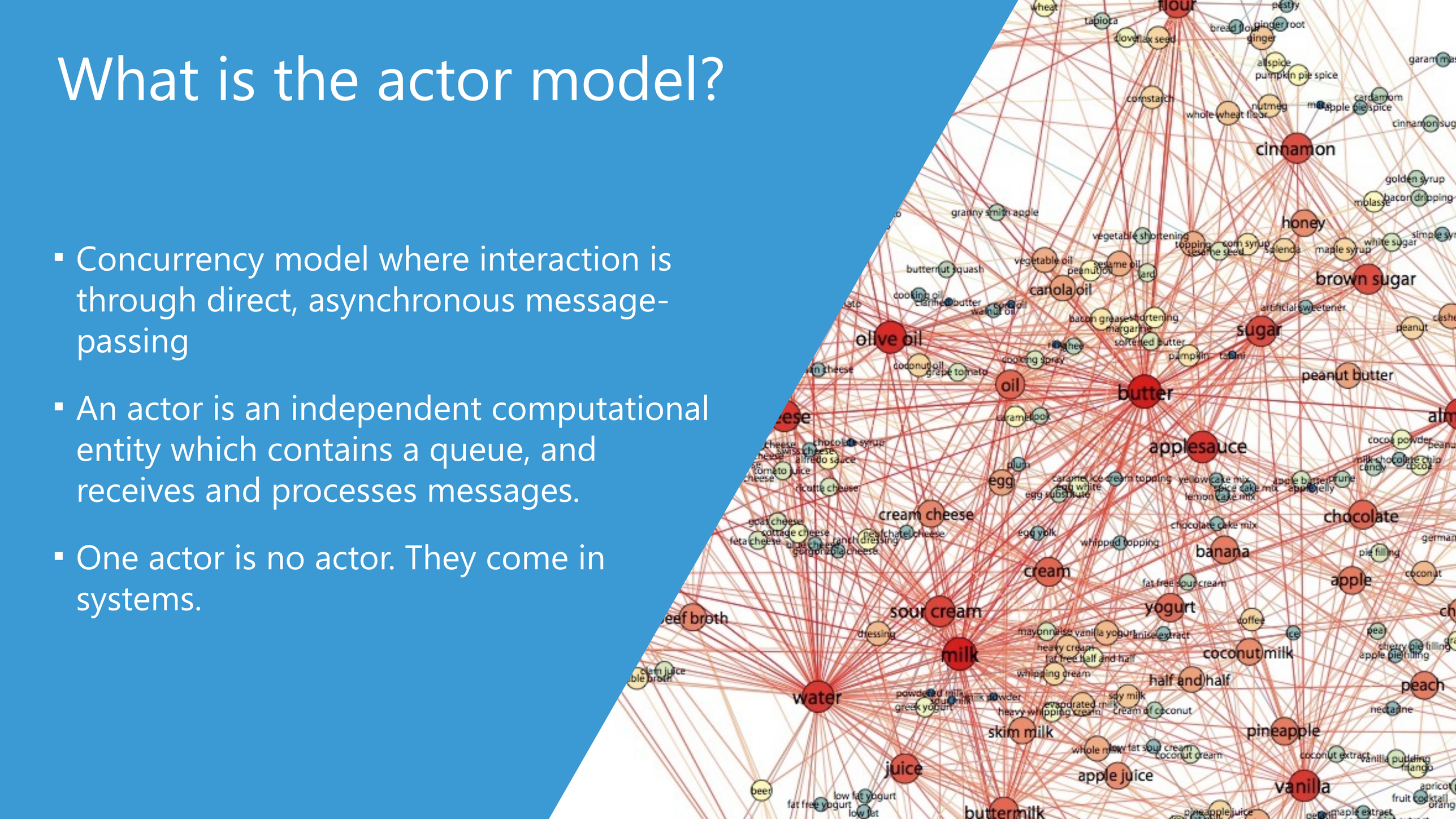
Demo

Tasky w/ SQLite type provider

MailboxProcessors

What is the actor model?

- Concurrency model where interaction is through direct, asynchronous message-passing
- An actor is an independent computational entity which contains a queue, and receives and processes messages.
- One actor is no actor. They come in systems.



Narita Airport
Airport Terminal 2
Jōban Line
Chiyoda Line
Kita-senju
Ayase
Minami-senju
Kanamachi
Thru to Imba-Nihon-idai
Keisei Kanamachi Line
Hokusō Line
Keisei Line
Hikifune
Tōbu Kameido Line
Tōbu Isesaki Line
Tsukuba Express
Aoto
Keisei-yawata
Keisei Oshiage Line
Keisei-takasago
Motoyawata
Oshiage
Asakusa
Honjo-azumabashi
Tawaramachi
Inarichō
Iriya
Minowa
Minowabashi
Nippori
Nishi-nippori
Uguisudani
Ueno
Keisei-ueno
Tabata
Ōji
Ōji-ekimae
Akabane
Akabane-iwabuchi
Nishigahara
Komagome
Hon-komagome
Tōdaimae
Sugamo
Nishi-sugamo
Shin-kōshinzuka
Itabashi
Shin-itabashi
Itabashihonchō
Itabashi kuyakushomae
Ōtsuka-ekimae
Ōtsuka
Sengoku
Hakusan
Myōgadani
Shin-ōtsuka
Ikebukuro
Tōbu Tōjō Line
Wakōshi
Chikatetsu-narimasu
Chikatetsu-akatsuka
Heiwadai
Hikawadai
Kotake-mukaihara
Senkawa
Kanamechō
Hikarigaoka
Nerima-kasugachō
Toshimaen
Nerima
Seibu Ikebukuro Line
Thru to Hannō
Shin-egota
Ochiai-minami-nagasaki
Nakai
Seibu Shinjuku Line
Yūrakuchō Line (New Line)
Yamanote Line
Marunouchi Line
Higashi-ikebukuro 4-chōme
Higashi-ikebukuro
Mejiro
Edogawabashi
Gokokuji
Waseda
Takadanobaba
Kōrakuen
Kasuga
Hongō-sanchōme
Namboku Line
Mita Line
Nezu
Sendagi
Ueno-okachimachi
Yushima
Ueno-hirokōji
Naka-okachimachi
Okachimachi
Shin-okachimachi
Kuramae
Kinshichō
Hanzōmon Line
Sōbu Line
Kameido
Shinozaki
Mizue
Ichinoe
Funabori
Higashi-ōjima
Ōjima
Nishi-ōjima
Nishi-funabashi
Baraki-nakayama
Myōden
Gyōtoku
Minami-gyōtoku
Urayasu
Kasai
Nishi-kasai
Tsudanuma
Chiba
Minami-sunamachi
Tōyōchō
Kiba
Shibayama-chiyoda
Higashi-narita
Shibayama Railway Line
Tōyō Rapid Line Thru to Tōyō-katsutadai
Sōbu Line / Narita Line
Ogikubo
Chūō Line
Thru to Mitaka
Minami-asagaya
Shin-kōenji
Higashi-kōenji
Shin-nakano
Nakano
Higashi-nakano
Ochiai
Tōzai Line
Waseda
Kagurazaka
Shin-ōkubo
Wakamatsu-kawada
Ushigome-yanagichō
Ushigome-kagurazaka
Ōedo Line
Iidabashi
Suidōbashi
Jimbōchō
Ochanomizu
Suehirochō
Akihabara
Asakusa-bashi
Ryōgoku
Morishita
Kikukawa
Sumiyoshi
Bakurochō
Bakuro-yokoyama
Iwamotochō
Higashi-nihombashi
Hamachō
Ningyōchō
Kodemmachō
Shin-nihombashi
Mitsukoshimae
Suitengūmae
Kiyosumi-shirakawa
Shinjuku Line
Kanda
Shin-ochanomizu
Ogawamachi
Awajichō
Hōnanchō
Nakano-fujimichō
Nakano-shimbashi
Nakano-sakaue
Nishi-shinjuku
Shinjuku-nishiguchi
Higashi-shinjuku
Seibu-shinjuku
Akebonobashi
Nishi-shinjuku-gochōme
Tochōmae
Shinjuku
Shinjuku-gyoemmae
Shinjuku-sanchōme
Yotsuya
Yotsuya-sanchōme
Ichigaya
Kudanshita
Kōjimachi
Hanzōmon
Takebashi
Ōtemachi
Nihombashi
Kayabachō
Monzen-nakachō
Meidaimae
Thru to Hashimoto, Takaosanguchi
Keiō Line
Sasazuka
Keiō Inokashira Line
Sendagaya
Shinanomachi
Aoyama-itchōme
Yoyogi
Kokuritsu-kyōgijō
Nagatachō
Nijūbashimae
Tōkyō
Kyōbashi
Takarachō
Hatchōbori
Keiyō Line
Yoyogi-uehara
Thru to Karakida, Hon-atsugi
Odakyū Line
Shimo-kitazawa
Yoyogi-kōen
Harajuku
Meiji-jingūmae
Gaiemmae
Akasaka-mitsuke
Sakuradamon
Yūrakuchō
Ginza-itchōme
Shintomichō
Toyosu
Tatsumi
Shin-kiba
Maihama
Disney Resort Line
Thru to Chūō-rinkan
Futako-tamagawa
Tōkyū Den-en-toshi Line
Shibuya
Omote-sandō
Akasaka
Tameike-sannō
Kokkai-gijidōmae
Kasumigaseki
Hibiya
Tsukiji
Tsukishima
Kokusai-tenjijō
Ariake
Tōkyū Ōimachi Line
Nogizaka
Roppongi-itchōme
Ginza
Higashi-ginza
Kachidoki
Tōkyū Tōyoko Line
Naka-meguro
Ebisu
Hiro-o
Roppongi
Uchisaiwaichō
Kamiyachō
Toranomon
Tsukijishijō
Thru to Kikuna
Den-en-chōfu
Jiyūgaoka
Hibiya Line
Ginza Line
Asakusa Line
Tōkyū Meguro Line
Azabu-jūban
Onarimon
Shimbashi
Ōokayama
Thru to Musashi-kosugi
Hatanodai
Tōkyū Ikegami Line
Meguro
Shirokanedai
Shirokane-takanawa
Mita
Akabanebashi
Shibakōen
Daimon
Hamamatsuchō
Shiodome
Yurikamome Line
Line Color
Line Symbol
Station Number
Toei Line
Tokyo Metro Line
Asakusa Line
Ginza Line
Mita Line
Marunouchi Line

Agents basics

```
type Agent<'a> = MailboxProcessor<'a>

let agent =
        Agent.Start(fun inbox ->
            let rec loop state =
                async {
                        let! msg = inbox.Receive()
                        printfn "got message %s, with state %i" msg state
                        return! loop (state + 1)
                }
            loop 0)

agent.Post "hello!"
```

Finite state machine!

> got message hello!, with state 0

> got message hello!, with state 1

...

> got message hello!, with state 6000

Demo

StockTicker w/ MailboxProcessors

Thanks!
For more F# goodness:
http://fsharp.org/

Rachel Reese

@rachelreese

rachelree.se, github: rachelreese