Choosing a Data Structure

Kit Eason
www.kiteason.com
@kitlovesfsharp

What Really Matters

Productivity

The ability to produce correctly working features over time

Selecting Data Structures for Productivity

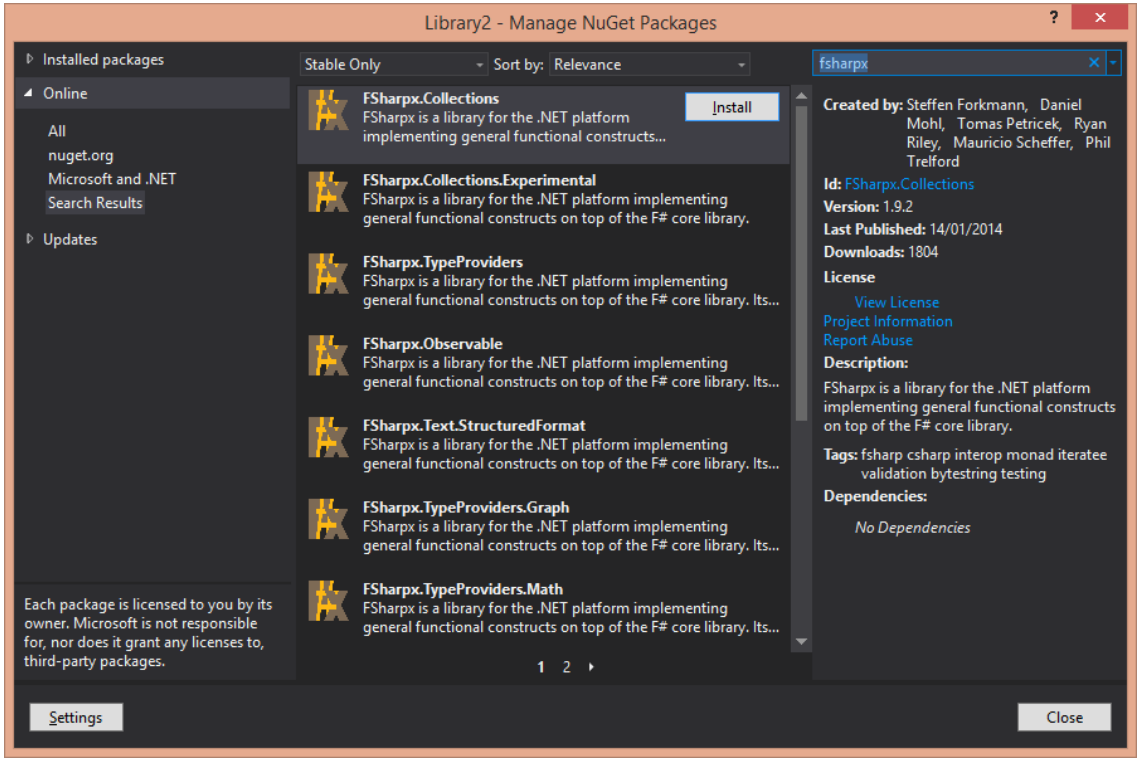
- Avoid ‘rolling your own’
- Find an F# structure that will do the job well
- …or a .NET structure
- …or an open source structure
- …or a commercially available structure
- For external data sources consider a Type Provider

Costs of Rolling Your Own

- Coding
- Testing
- Debugging
- Documentation
- Maintenance
- Accidental complexity

Beyond the Built-in Data Structures

- Fsharpx.Collections
  - http://tinyurl.com/FSharpXCollections
- ExtCore
  - http://tinyurl.com/FSharpExtCore



Type Providers

- OO-style access to external data sources
- …without any coding!
- XML
- CSV
- SQL Server
- R (statistics language)

Reliability

The ability of a system to work correctly, and to continue working correctly

Reliability

- **Again, avoid rolling your own**
- **Use unit tests**
  - NUnit
  - FSUnit
  - NCrunch (commercial)
- **Make Illegal States Unrepresentable**

"Make illegal states unrepresentable"

MISU

— Yaron Minsky,
ocaml.janestreet.com

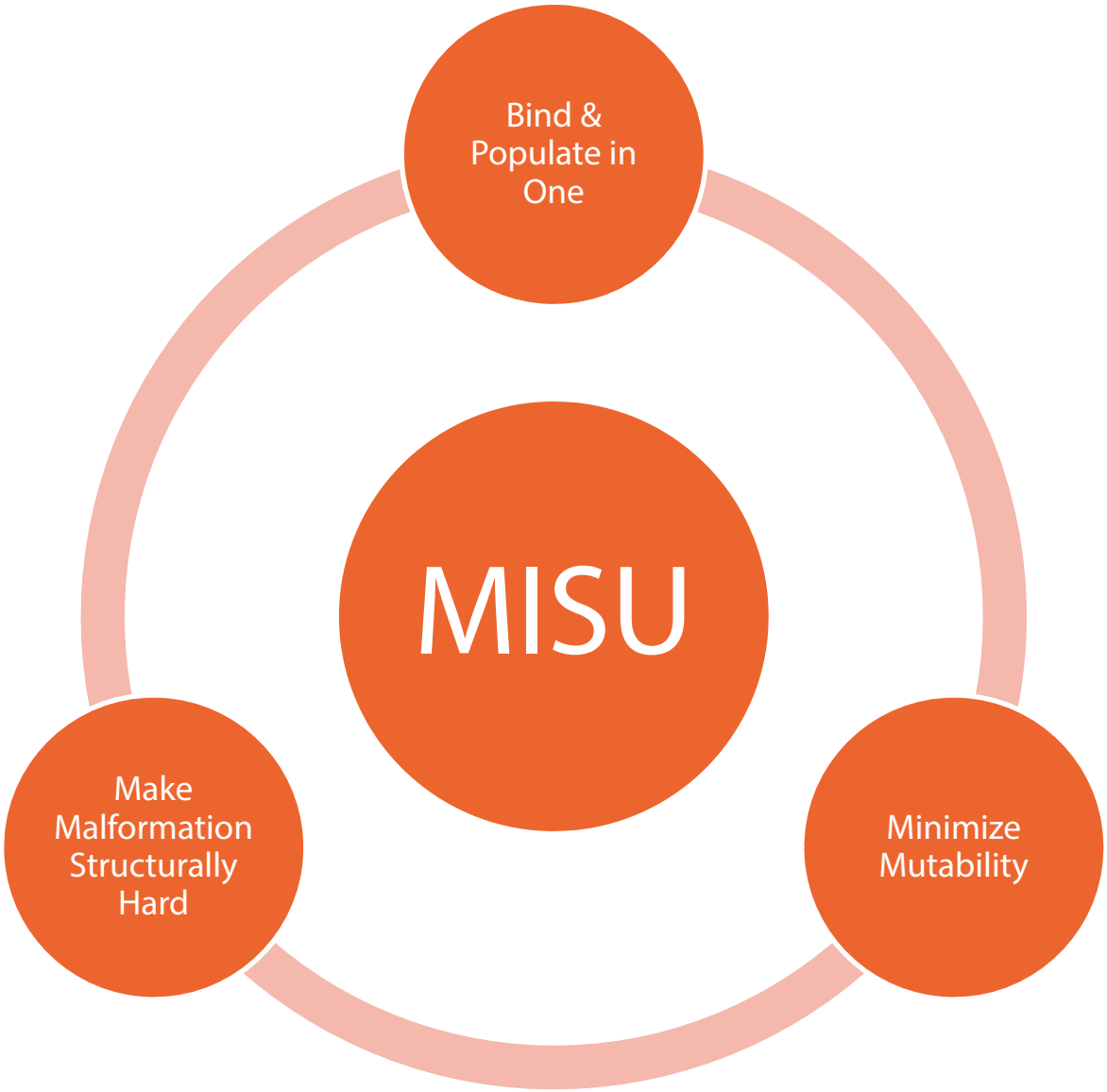
Make Illegal States Unrepresentable (MISU)

Design your data model so it can't be put into a state that would cause a crash or invalid output.

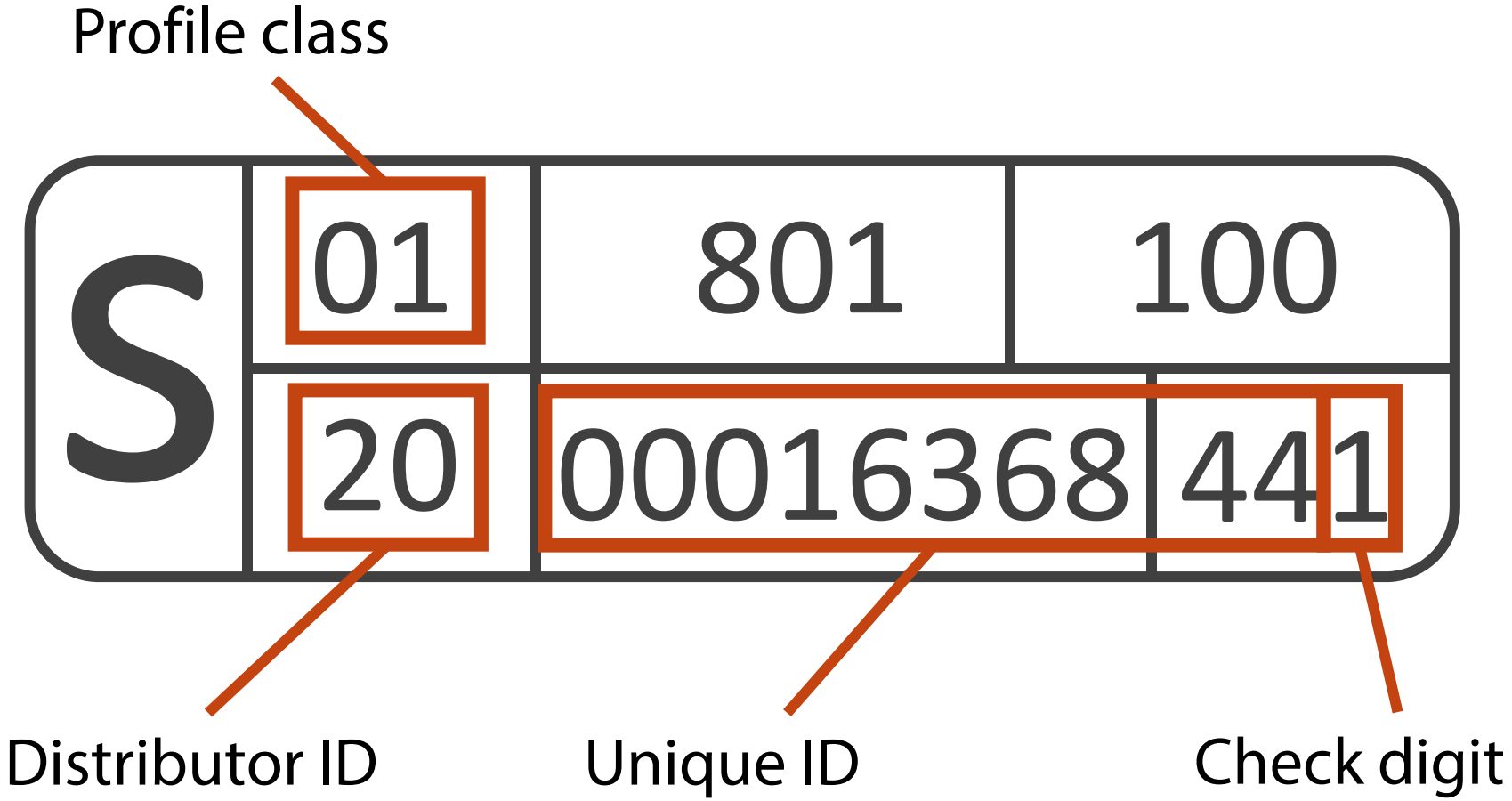
> I call it my billion-dollar mistake. It was the invention of the null reference in 1965.
>
> — Tony Hoare

- **Idiomatic F# code is much less prone to nulls.**

Getting to MISU



Meter Point Administration Number



MISU Code Smells

- **Lots of exception handling logic**
  - Except maybe at boundary
- **Lots of unit tests around invalid state**
  - How are you even writing those?
- **Traditional OO style**
  - Writing C# in F# syntax
- **Non-local mutability**
  - If you need mutability you **must** be declaring and populating separately

Maintainability

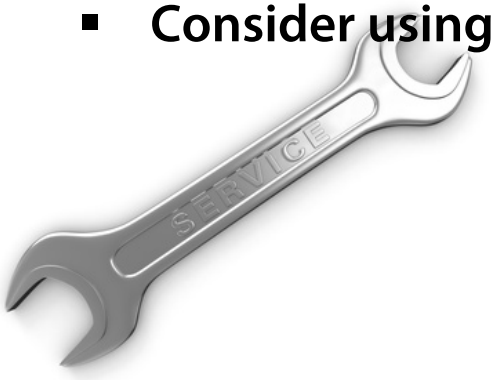
The ease with which code can be changed to fix bugs and make enhancements

Maintainability

- **Naming**
  - Keep names descriptive in context
  - Short names are good in small contexts
  - Very long names are a code smell
    - It means your functions are probably too long
- **Avoid long tuples**
  - >~ 3 elements
  - Small context
  - Consider declaring a record

```
type LatLong =
   { Lat : float
     Long : float }
```

- **Consider using type abbreviations**



Type Abbreviations

- **An alias for an existing type**
  - `type NewType = ExistingType`

```
type Transaction =
    { Id : string
      Net : decimal
      Tax : decimal }
```

```
type Id = string
type Money = decimal

type Transaction =
    { Id : Id
      Net : Money
      Tax : Money }
```

Single Case Discriminated Unions

- A kind of stricter type abbreviation
  - `type TransactionId = Id of string`

```
type TransactionId = Id of string
type Money = Amount of decimal

type Transaction =
   { Id : TransactionId
     Net : Money
     Tax : Money }
```

```
let myTx =
   { Id = Id "ABC123"
     Net = Amount 1099.0M
     Tax = Amount 1099.0M * Amount 0.2M }
```

Performance and Scalability

How fast a system runs, and how this changes as data volumes increase

Performance and Scalability

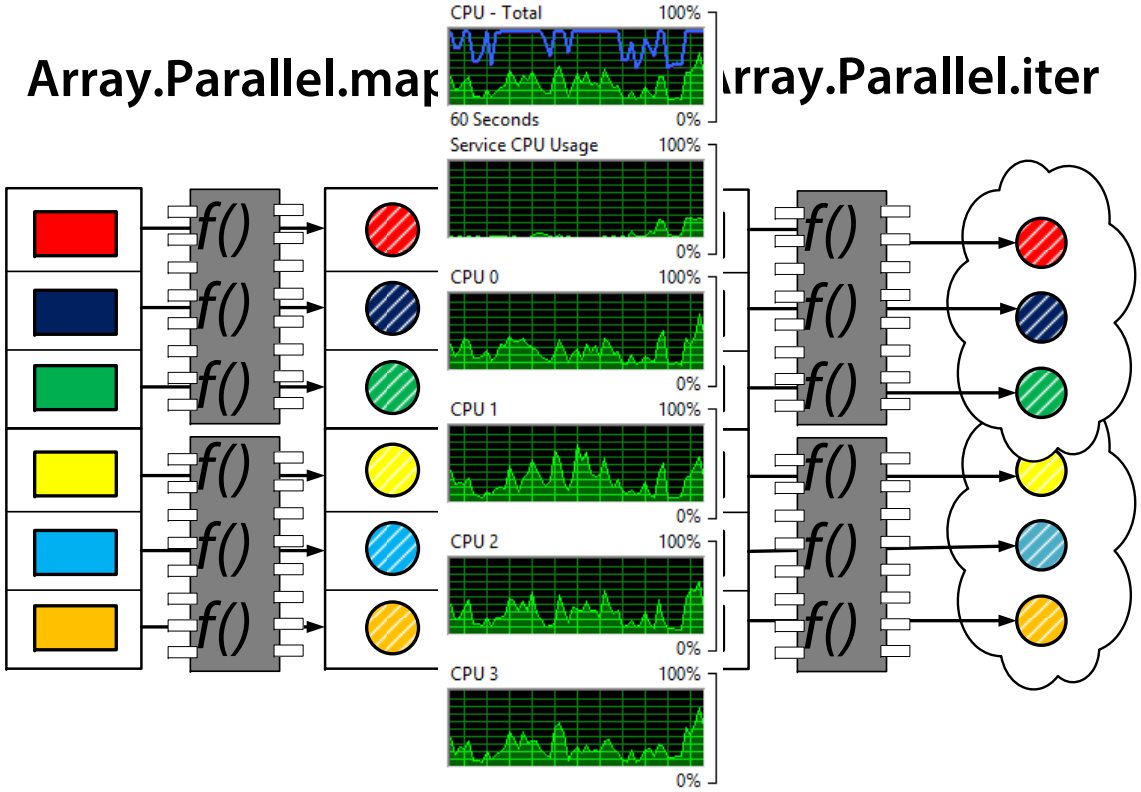
- **Arrays are great for performance**
  - Good locality of reference
  - O(1) access by index
- **Lists can be efficient for memory**
  - …when multiple lists share nodes
  - O(*n*) access by index
- **Sequences give delayed evaluation**
  - …but watch out for re-evaluation
  - Consider using Seq.cache

Performance and Scalability (2)

- **Dictionaries**
  - Remember the trade-offs between .NET Dictionaries, F# Maps and the dict operator
- **Immutable versus mutable collections**
  - Trade off safety and performance

The Array.Parallel Module

- Array.Parallel.map ... rray.Parallel.iter



Principles of Performance

- **Know the performance characteristics of collections**
- **Don't optimize prematurely**
  - Get a working system and demo it to verify requirements
  - Have a strong suite of unit tests
  - Refactor with confidence
- **Optimize scientifically**
  - Use a profiler
  - Note performance gains

Summary

- **Productivity**
  - Remember the wealth of structures already available to you
  - Type Providers

- **Reliability**
  - Unit Tests
  - MISU

- **Maintainability**
  - Naming
  - Type Abbreviations
  - Single-case DUs

- **Performance**
  - Know the performance of your data structures
  - Measure, optimize

Further Reading

- **F# For Fun and Profit**
  - http://fsharpforfunandprofit.com
- **The F# Software Foundation**
  - http://fsharp.org
- **Pluralsight!**
  - http://pluralsight.com
- **Expert F#**
  - ISBN: 978-1-4302-4650-3
- **Programming F# 3.0**
  - ISBN: 978-1-4493-2029-4