Functional Programming

Venkat Subramaniam
venkats@agiledeveloper.com
@venkat_s

Why Functional Programming?

Programming has become complex

But why?

- Domain is only part of the reason
- We've gone too far with OO programming and mutable state

Mutable State

What's wrong with mutable state?

After all, we've used it for such a long time

Perils of Mutable State

Mutable state

- leads to more bugs in code
- makes concurrency quite difficult

What's Old is New Again

Functional Programming was introduced a long time ago!

It was way ahead of its time

Current developments in the hardware area have rekindled interest in this paradigm

What's Functional Programming?

Assignment-less programming

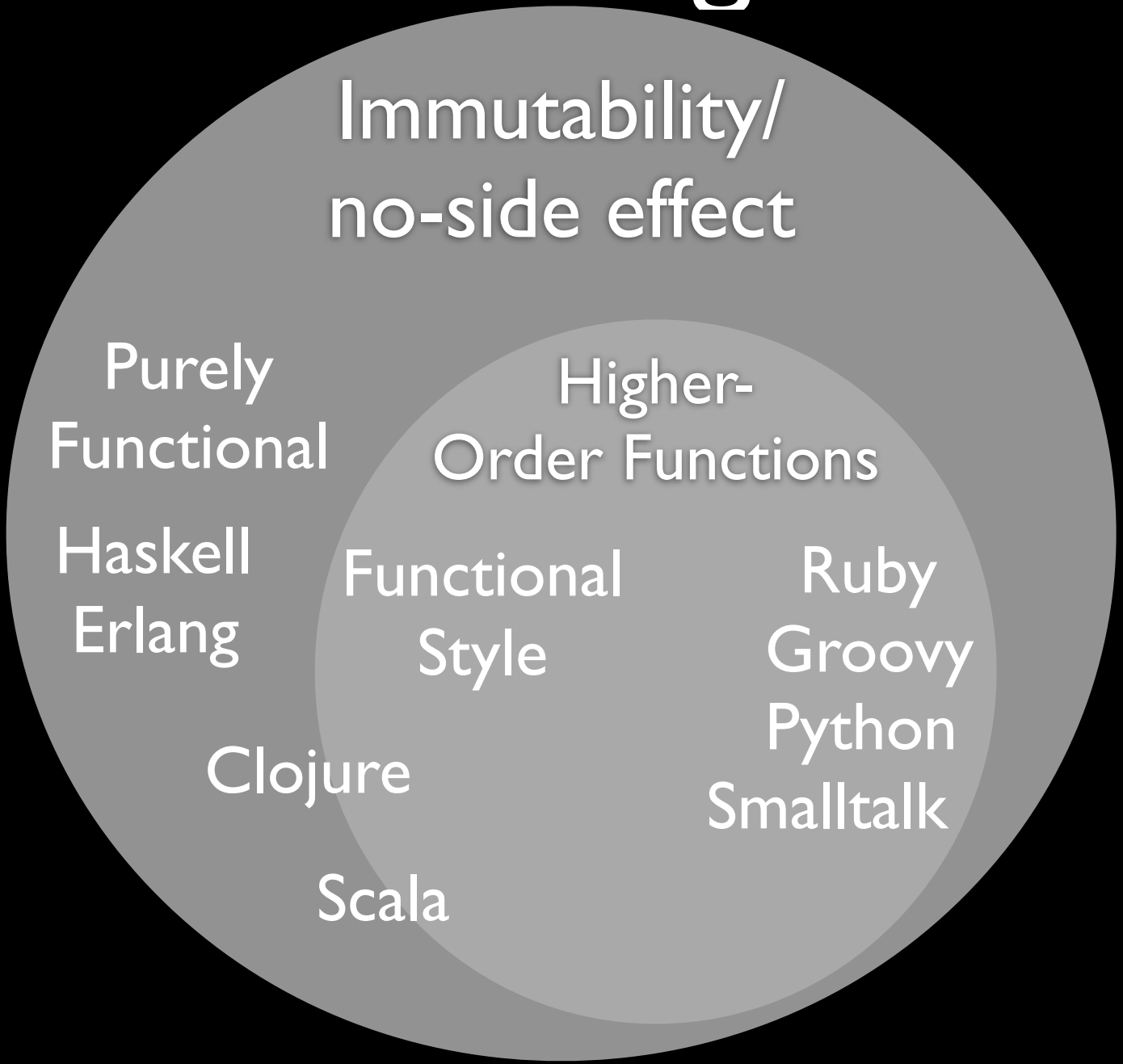
Immutable State

Functions as first-class citizens

Higher-order functions

Functions with no side-effects

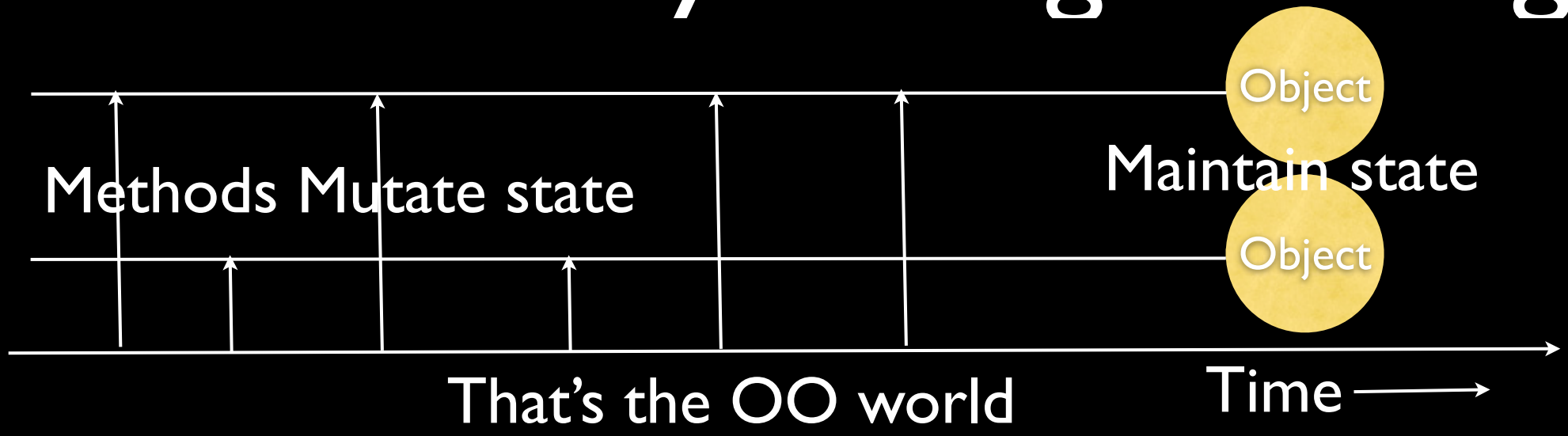
Functional Programming



Languages

	non-functional	Hybrid	Functional
Statically typed	Java C++ C#	Scala F#	Haskell
Dynamically typed	Ruby Groovy	Clojure	LISP Erlang

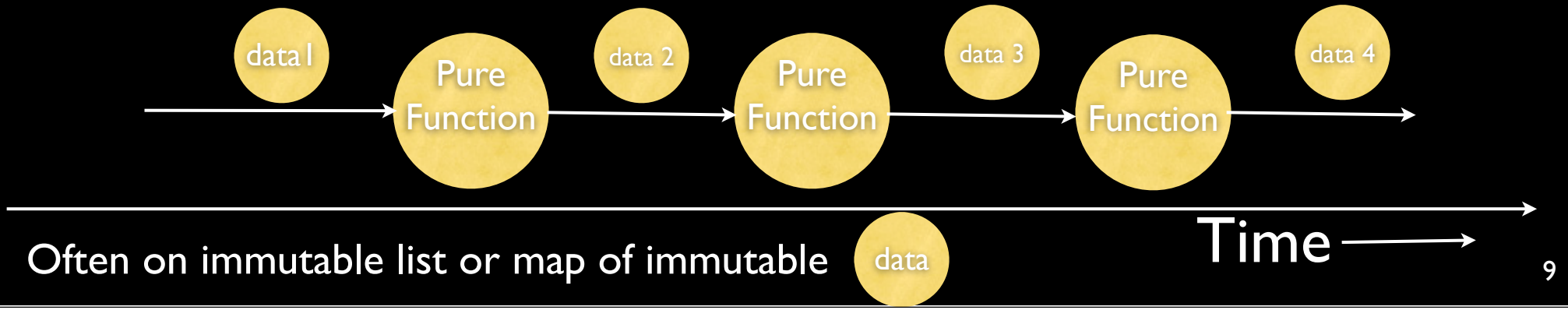
Functional Style Programming



Functional World

Functional composition

Series of transformations



Imperative vs. Functional

Imperative	Functional
You specify each step	More directive in style
How to do stuff	What you want to get
Mutates at will	Immutable data
Often has side-effect	Has no side-effect
Accepts data/objects	Accepts functions also
Hard to compose	Functional composition
Data mutated	Data transformed

Double: Imperative Style

Double values in a list

Double: Functional Style

Double values in a list

Exercise 1

Given a list of names, produce a list of tuples with names and size of each name. For example, given “John”, “Jack”, “Jill”, “Sam”, “William”, the result should be the list (“John”, 4), (“Jack, 4), (“Jill”, 4), (“Sam”, 3), (“William, 7).

Total: Imperative Style

Total values in a list

Total: Functional Style

Total values in a list

Exercise 2

Given a list of names, find the total number of characters. For example, given “John”, “Jack”, “Jill”, “Sam”, “William”, the result should be 22.

Exercise 3

Given a number, determine if the number is a perfect number or not. A perfect number is a number for which the sum of its factors equals twice the number. For example, factors of 6 are 1, 2, 3, and 6, total of which is 12, which is equal to 6 * 2. The number 7 is not perfect since 1 + 7 == 8 != 14.

Recursion

One way to gain immutability is through recursion

Recursion, however, posses a challenge

- May result in Stack Overflow exception

Procedure vs. Process

Structure and Interpretation of Computer Programs—Sussman, et. al. (SICP—a great book)

Procedure—code we write

Process—code that runs

Iterative Procedure often transforms to iterative process

Recursion often transforms to recursive process

What if we can take a recursive procedure and transform it to iterative process

Tail Call Optimization (TCO)

Functional Programming languages often provide TCO

Scala does it through compiler optimization

Trail Recursion

Tail recursion and stack usage

Exercise 4

Given a number, determine the factorial of that number using tail recursion

Purity Facilitates Memoization

Dynamic programming exploits two things

- Recursive nature of the problem
- High redundancy in the solution space

Introduces excessive redundancy, but memoizes (caches) the solution to avoid repeated computations

Exercise 5

Using memoization, compute the Fibonacci number where Fib(n) = Fib(n-1) + Fib(n-2), and Fib(0) = 0, and Fib(1) = 1. Measure the time for non-memoized recursive version and the memoized version.

Summary

Discussions

Usage of Functional Style of Programming

Thank you

http://www.agiledeveloper.com

Venkat Subramaniam
venkats@agiledeveloper.com
twitter: @venkat_s