Functional Programming and Distributed Computing

Fanyu Ran • Yang Zhou • Yuhang Gao • Yuhao Lu

Overview

- Distributed Computing
- Problem with OOP
- Functional Programming
- What can FP offer
- FP and Distributed Computing

Distributed Computing

Purpose

- Parallel, High-Performance Applications

  (Big Data, HPC, etc.)

- Fault-tolerant Applications

  (Telecom Infrastructure, Web, etc.)

Basic Issue

- Utilization and cooperation of **multiple** processors.
- The potential for partial failure.

Object Oriented Programming
vs
Functional Programming

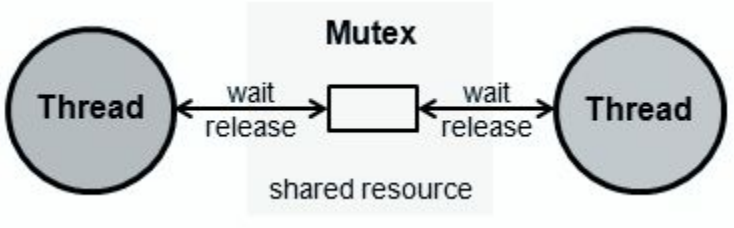
Problem with OOP

Mutable State

➔ The Object's methods is supposed to mutate its internal state (variables).

Problem with OOP

➔ When state is shared:



➔ Problem solved ?

Bottleneck, Deadlock, Complexity...

Image from:
https://www.keil.com/pack/doc/cmsis/RTOS/html/group__CMSIS__RTOS__MutexMgmt.html

Problem with OOP

Deadlock

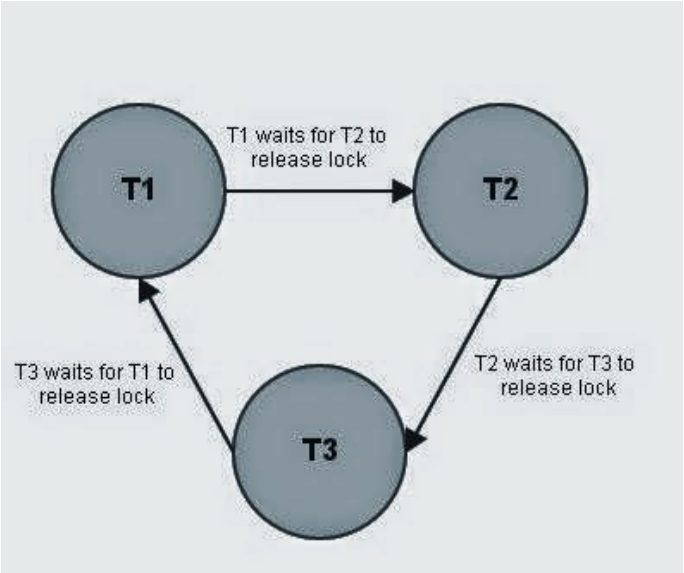


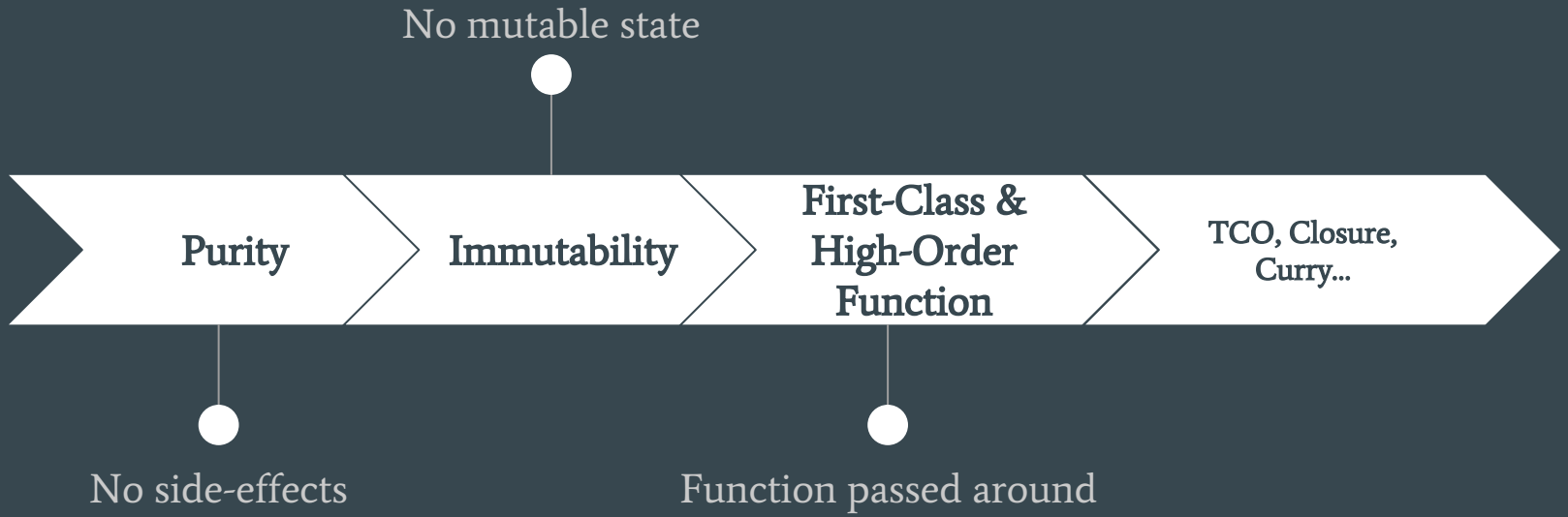
Image from: http://www.exploredatabase.com/2014/04/what-is-deadlock-in-database.html

What is Functional Programming ?

What is Functional Programming ?

- FP is a type of programming paradigm which has several features.
- Most of traditional languages (Javascript, Python, Java, etc.) can be written in functional style.
- FP language is language designed with FP in mind.
  - Lisp
  - Haskell
  - OCaml
  - Erlang
  - Scala (?)
  - ...

Features of FP



Purity

- Function reads all inputs from its input arguments.
- Function exports all outputs to its return values.

```
a = 1
b = 1
sum = None
def impure_function():
    sum = a + b

def pure_function(a, b):
    return a + b
```

Purity

- The function always evaluates the same result value given the same argument value(s).
- Evaluation of the result does not cause any semantically observable side effect or output, such as mutation of mutable objects or output to I/O devices.

```
a = 1
b = 1
sum = None
def impure_function():
    sum = a + b

def pure_function(a, b):
    return a + b
```

https://en.wikipedia.org/wiki/Pure_function

Immutability

State of objects cannot be modified after it is created.

How can we program without modifying state ?

Immutability

Loop as an example:

```
arr = [1, 2, 3]

# imperative
def sum(arr):
    res = 0
    for i in range(0, len(arr)):
        res = res + arr[i]


# naive recursive
def sum(arr):
    if len(arr) == 1:
        return arr[0]
    else:
        return arr[0] + sum(arr[1:])
# sum([1, 2, 3])
# 1 + sum([1, 2, 3])
# 1 + 2 + sum([3])
# 1 + 2 + 3
# 6
```

First-Class Function

Capability of programming language to:

- pass functions as arguments to other functions
- return functions as the values from other functions
- assign functions to variables
- store functions in data structures

To be concise, function is just like all other values like integer, float, double, etc..

https://en.wikipedia.org/wiki/First-class_function

High-Order Function

Function that does at least one of the following

- takes one or more functions as arguments
- returns a function as its result

```
arr = [1, 2, 3]
# high-order function
def add(a, b):
    return a + b

def sum(arr):
    reduce(add, arr)
# 0, [1, 2, 3]  -> add(0, 1) = 1
# 1, [2, 3]     -> add(1, 2) = 3
# 3, [3]        -> add(3, 3) = 6
# 6, []         -> 6

# more concise
def sum(arr):
    reduce(lambda a,b: a + b, arr)
```

What can FP offer to distributed computing ?

No side-effects and mutable variables

FP facilitates code distribution over several CPU and eases concurrent programming.

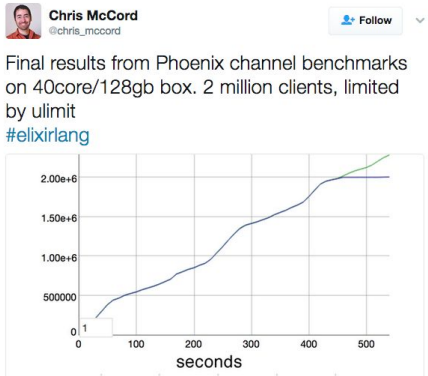
Functions as building blocks

Functions can be **combined**, **sent remotely** and **applied locally** on distributed data sets.

FP in the real world Erlang/Elixir

What is Elixir?

- Elixir is a dynamic, functional language designed for building scalable and maintainable applications
- Elixir leverages the Erlang VM, known for running low-latency, distributed and fault-tolerant systems
- being successfully used in web development and the embedded software domain(e.g.2 Million Websocket Connections in single machine).



FP in the real world Erlang/Elixir

NINE nines(99.9999999% reliability)?

Two Pillars of resilience and reliability

- Message-passing between isolated processes
- Automatic recovery and monitoring

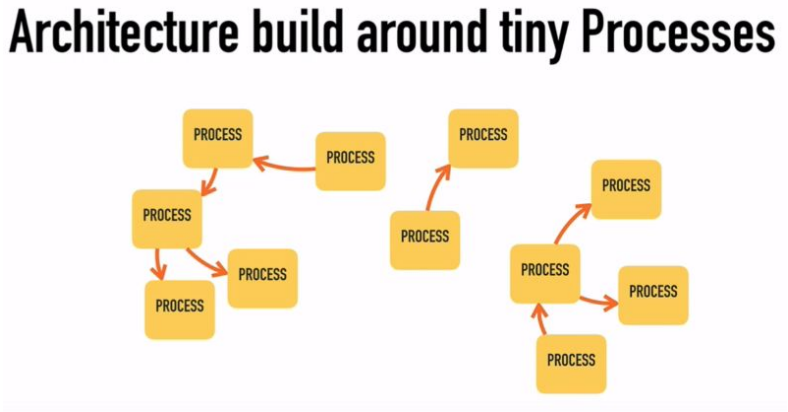


Image from https://www.youtube.com/watch?v=naNN_gJas2A

FP in the real world Erlang/Elixir

Elixir power tools

- Message with **GenServer** modules

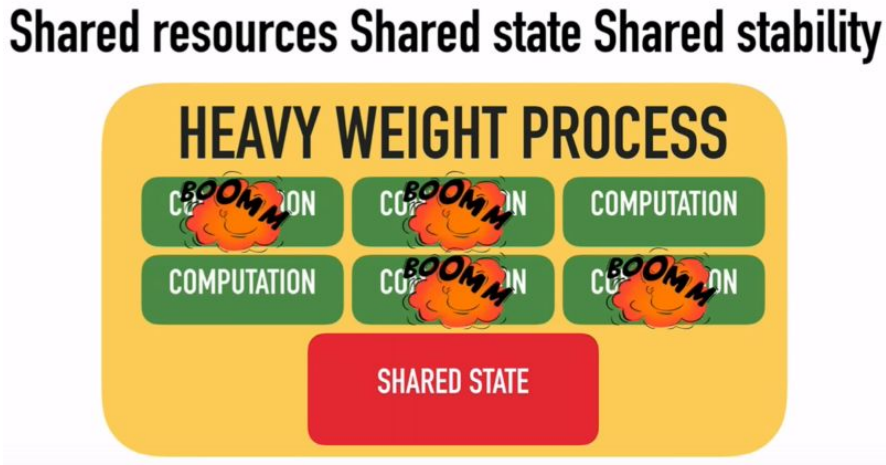


Image from https://www.youtube.com/watch?v=naNN_gJas2A

FP in the real world Erlang/Elixir

Elixir power tools

- Message with **GenServer** modules

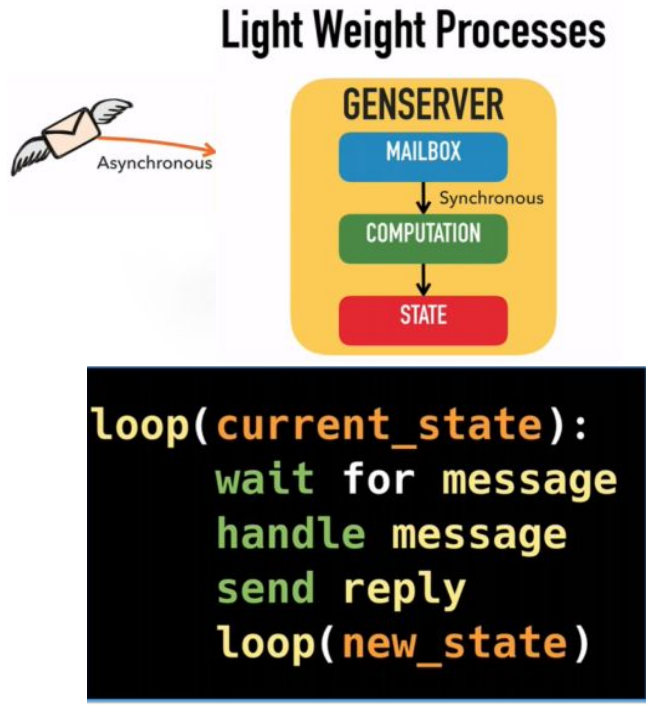


Image from https://www.youtube.com/watch?v=naNN_gJas2A

FP in the real world Erlang/Elixir

Elixir power tools

- **Supervisor** for transparent resilience

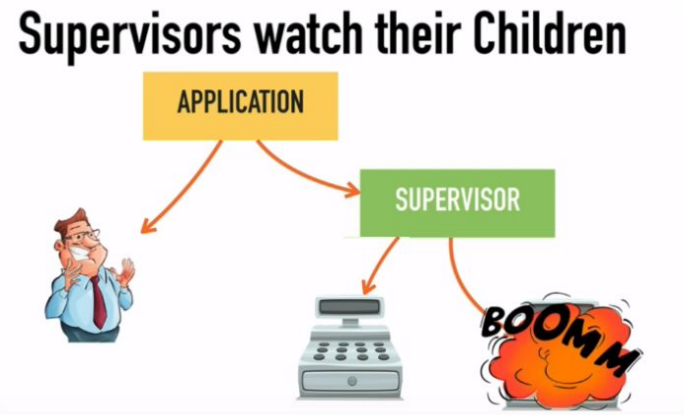


Image from https://www.youtube.com/watch?v=naNN_gJas2A

Thank You