

# Introduction à la Programmation Fonctionnelle en Haskell

Yann Esposito

<2018-03-15 Thu>

# Courte Introduction

# Prelude

Initialiser l'env de dev:

```
curl -sSL https://get.haskellstack.org/ | sh
stack new ipfh https://git.io/vbpej && \
cd ipfh && \
stack setup && \
stack build && \
stack test && \
stack bench
```

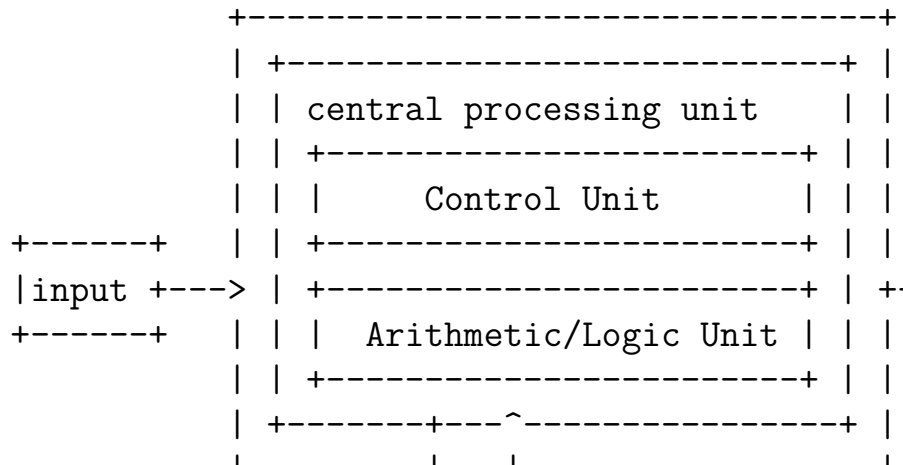
# Parcours jusqu'à Haskell

## Parcours Pro

- ▶ Doctorat (machine learning, hidden markov models) 2004
- ▶ Post doc (écriture d'un UI pour des biologistes en Java). 2006
- ▶ Dev Airfrance, (Perl, scripts shell, awk, HTML, CSS, JS, XML...) 2006 → 2013
- ▶ Dev (ruby, C, ML) pour GridPocket. (dev) 2009 → 2011, (impliqué) 2009 →
- ▶ Clojure dev & Machine Learning pour Vigiglobe. 2013 → 2016
- ▶ Senior Clojure développeur chez Cisco. 2016 →

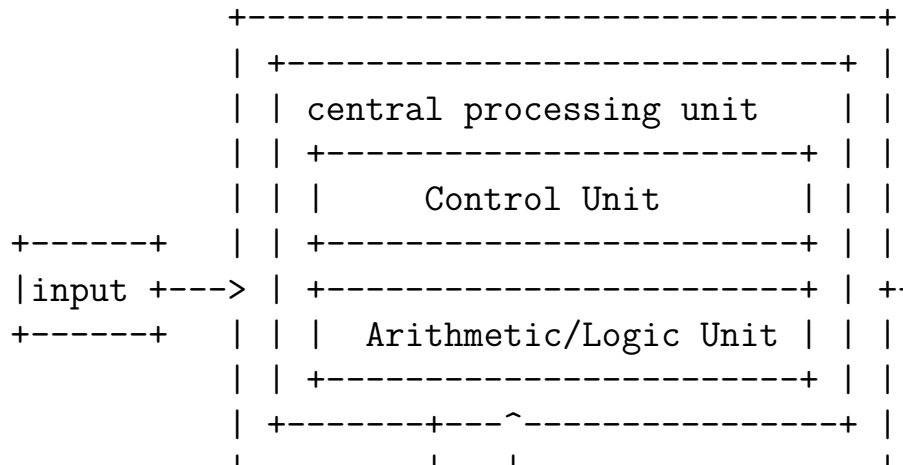
# Qu'est-ce que la programmation fonctionnelle?

## Von Neumann Architecture



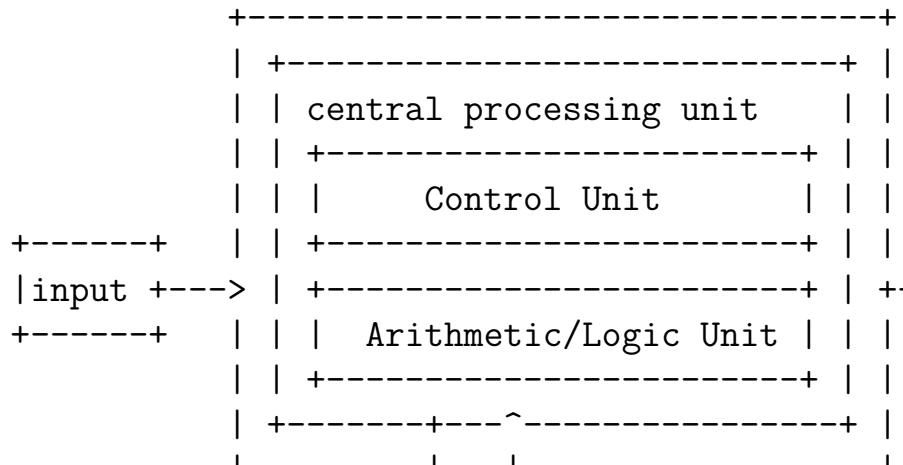
# Qu'est-ce que la programmation fonctionnelle?

## Von Neumann Architecture



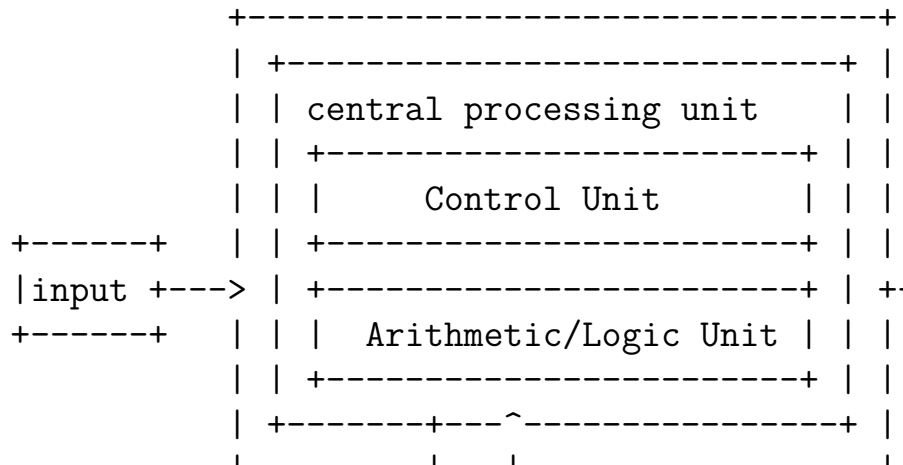
# Qu'est-ce que la programmation fonctionnelle?

## Von Neumann Architecture



# Qu'est-ce que la programmation fonctionnelle?

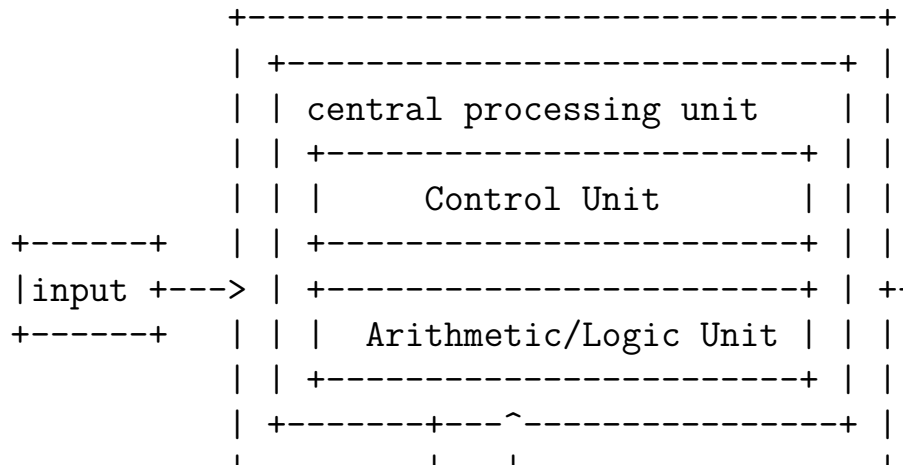
## Von Neumann Architecture





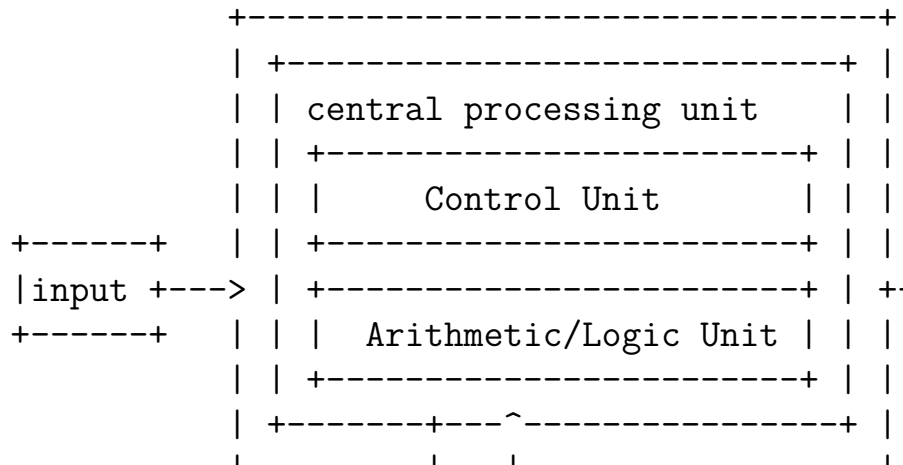
# Qu'est-ce que la programmation fonctionnelle?

## Von Neumann Architecture



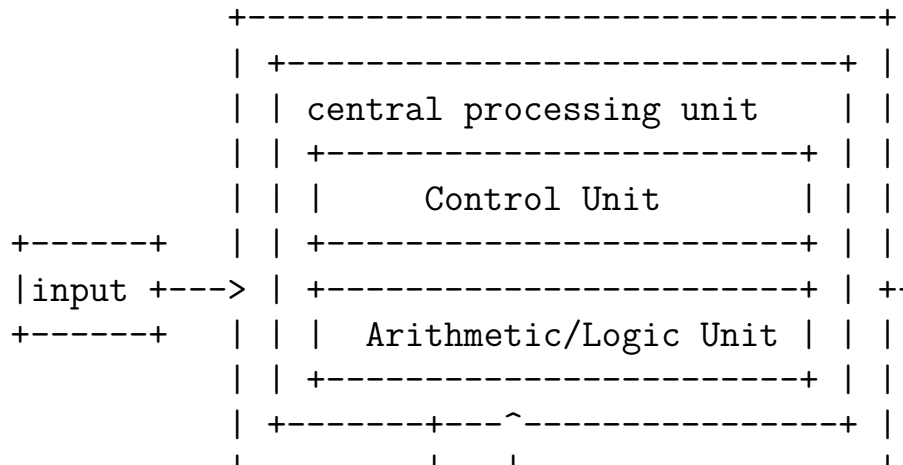
# Qu'est-ce que la programmation fonctionnelle?

## Von Neumann Architecture



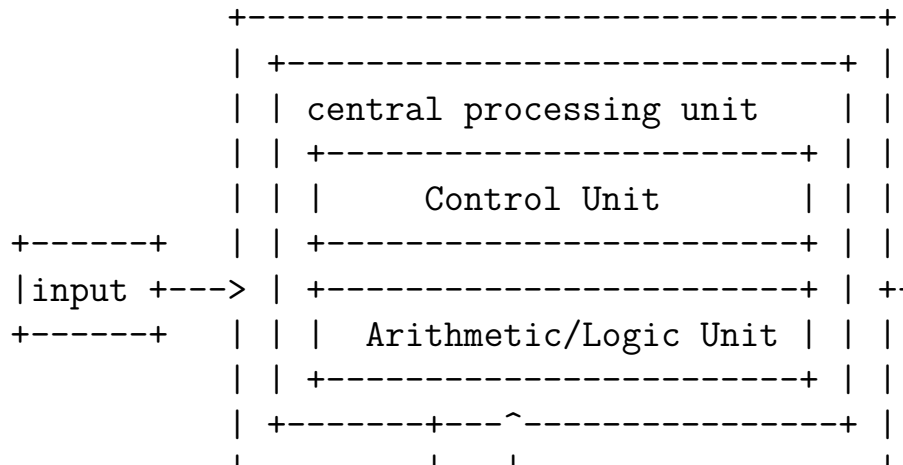
# Qu'est-ce que la programmation fonctionnelle?

## Von Neumann Architecture



# Qu'est-ce que la programmation fonctionnelle?

## Von Neumann Architecture



# Pourquoi Haskell?

## Simplicité par l'abstraction

**/!\ SIMPLICITÉ FACILITÉ /!\**

- ▶ mémoire (garbage collection)
- ▶ ordre d'évaluation (non strict / lazy)
- ▶ effets de bords (pur)
- ▶ manipulation de code (referential transparency)

## Production Ready™

- ▶ rapide
  - ▶ équivalent à Java (~ x2 du C)
  - ▶ parfois plus rapide que C
  - ▶ bien plus rapide que python et ruby

# Pourquoi Haskell?

## Simplicité par l'abstraction

**/!\ SIMPLICITÉ FACILITÉ /!\**

- ▶ mémoire (garbage collection)
- ▶ ordre d'évaluation (non strict / lazy)
- ▶ effets de bords (pur)
- ▶ manipulation de code (referential transparency)

## Production Ready™

- ▶ rapide
  - ▶ équivalent à Java (~ x2 du C)
  - ▶ parfois plus rapide que C
  - ▶ bien plus rapide que python et ruby

# Pourquoi Haskell?

## Simplicité par l'abstraction

**/!\ SIMPLICITÉ FACILITÉ /!\**

- ▶ mémoire (garbage collection)
- ▶ ordre d'évaluation (non strict / lazy)
- ▶ effets de bords (pur)
- ▶ manipulation de code (referential transparency)

## Production Ready™

- ▶ rapide
  - ▶ équivalent à Java (~ x2 du C)
  - ▶ parfois plus rapide que C
  - ▶ bien plus rapide que python et ruby

# Pourquoi Haskell?

## Simplicité par l'abstraction

**/!\ SIMPLICITÉ FACILITÉ /!\**

- ▶ mémoire (garbage collection)
- ▶ ordre d'évaluation (non strict / lazy)
- ▶ effets de bords (pur)
- ▶ manipulation de code (referential transparency)

## Production Ready™

- ▶ rapide
  - ▶ équivalent à Java (~ x2 du C)
  - ▶ parfois plus rapide que C
  - ▶ bien plus rapide que python et ruby



# Pourquoi Haskell?

## Simplicité par l'abstraction

**/!\ SIMPLICITÉ FACILITÉ /!\**

- ▶ mémoire (garbage collection)
- ▶ ordre d'évaluation (non strict / lazy)
- ▶ effets de bords (pur)
- ▶ manipulation de code (referential transparency)

## Production Ready™

- ▶ rapide
  - ▶ équivalent à Java (~ x2 du C)
  - ▶ parfois plus rapide que C
  - ▶ bien plus rapide que python et ruby

# Pourquoi Haskell?

## Simplicité par l'abstraction

**/!\ SIMPLICITÉ FACILITÉ /!\**

- ▶ mémoire (garbage collection)
- ▶ ordre d'évaluation (non strict / lazy)
- ▶ effets de bords (pur)
- ▶ manipulation de code (referential transparency)

## Production Ready™

- ▶ rapide
  - ▶ équivalent à Java (~ x2 du C)
  - ▶ parfois plus rapide que C
  - ▶ bien plus rapide que python et ruby

# Pourquoi Haskell?

## Simplicité par l'abstraction

**/!\ SIMPLICITÉ FACILITÉ /!\**

- ▶ mémoire (garbage collection)
- ▶ ordre d'évaluation (non strict / lazy)
- ▶ effets de bords (pur)
- ▶ manipulation de code (referential transparency)

## Production Ready™

- ▶ rapide
  - ▶ équivalent à Java (~ x2 du C)
  - ▶ parfois plus rapide que C
  - ▶ bien plus rapide que python et ruby

# Pourquoi Haskell?

## Simplicité par l'abstraction

**/!\ SIMPLICITÉ FACILITÉ /!\**

- ▶ mémoire (garbage collection)
- ▶ ordre d'évaluation (non strict / lazy)
- ▶ effets de bords (pur)
- ▶ manipulation de code (referential transparency)

## Production Ready™

- ▶ rapide
  - ▶ équivalent à Java (~ x2 du C)
  - ▶ parfois plus rapide que C
  - ▶ bien plus rapide que python et ruby

# Premiers Pas en Haskell

# What is your name?

## What is your name? (1/3)

```
module Main where
```

```
main :: IO ()
```

```
main = do
```

```
    putStrLn "Hello! What is your name?"
```

```
    name <- getLine
```

```
    let output = "Nice to meet you, " ++ name ++ "
```

```
    putStrLn output
```

file:pres-haskell/name.hs

## What is your name? (2/3)

# Erreurs classiques

## Erreur classique #1

```
module Main where
```

```
main :: IO ()
```

```
main = do
```

```
    putStrLn "Hello! What is your name?"
```

```
    let output = "Nice to meet you, " ++ getLine
```

```
    putStrLn output
```

/Users/yaesposi/.deft/pres-haskell/name.hs:6:4

- Couldn't match expected type '[Char]'  
with actual type 'IO String'
- In the first argument of '(++)', namely

# Concepts avec exemples



# Composabilité

## Composabilité vs Modularité

Modularité: soit un a et un b, je peux faire un c. ex:  
x un graphique, y une barre de menu  $\Rightarrow$  une page  
`let page = mkPage ( graphique, menu )`

Composabilité: soit deux a je peux faire un autre a.  
ex: x un widget, y un widget  $\Rightarrow$  un widget  
`page = x <+> y`

Gain d'abstraction, moindre coût.

**Hypothèses fortes sur les a**

## Exemples

- ▶ **Semi-groupes  $\langle + \rangle$**

# Catégories de bugs évités avec Haskell

# Organisation du Code

# Règles pragmatiques

## Organisation en fonction de la complexité

*Make it work, make it right, make it fast*

- ▶ Simple: directement IO (YOLO!)
- ▶ Medium: Haskell Design Patterns: The Handle Pattern: <https://jaspervdj.be/posts/2018-03-08-handle-pattern.html>
- ▶ Medium (bis): MTL / Free / Freeer / Effects...
- ▶ Gros: Three Layer Haskell Cake:  
[http://www.parsonsmatt.org/2018/03/22/three\\_layer\\_haskell\\_cake.html](http://www.parsonsmatt.org/2018/03/22/three_layer_haskell_cake.html)
  - ▶ Layer 1: Imperatif
  - ▶ Orienté Objet (Level 2 / 2')

# Conclusion

# Appendix