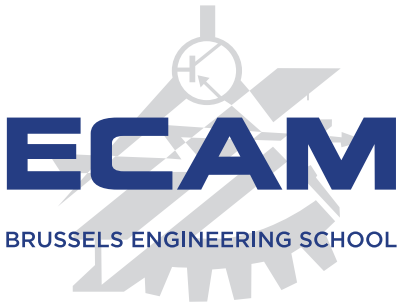


I402A Software Architecture
and Quality Assessment

Session 5

Programming Paradigms

Sébastien Combéfis

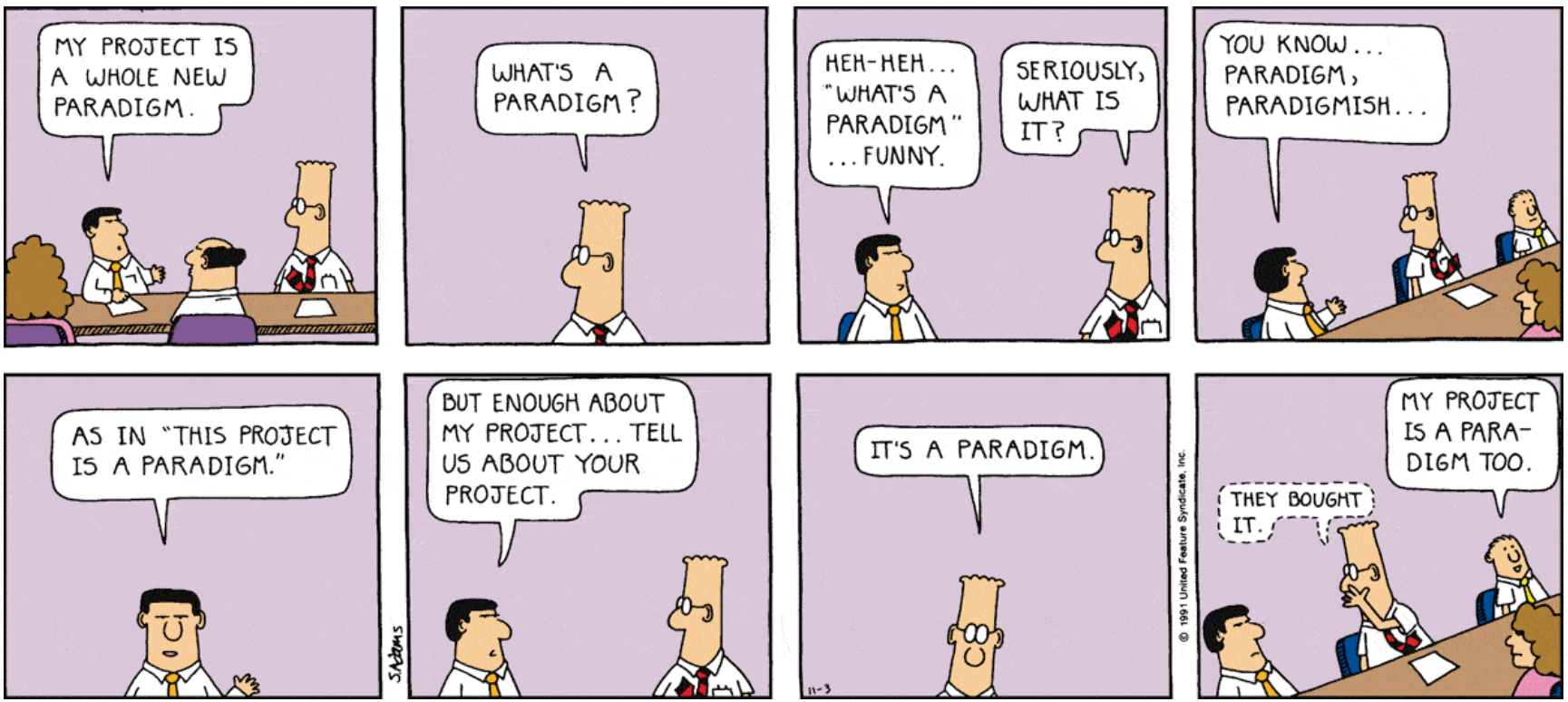
Fall 2019

Objectives

- Understand what a programming paradigm is
  - Definition and characterisation
  - From declarative to imperative programming
  - Functional, logic and object oriented programming languages
- Programming language examples
  - Characterisation of languages and constructions
  - Choosing a programming language


Main Programming Paradigm

A paradigm is...



...a worldview

A paradigm is a representation of the world, a way of seeing things, a coherent model of worldview based on a definite basis (disciplinary matrix, theoretical model or current of through).

Analogy

- **Software engineering**
  - Defines the processes for making softwares
  - Several existing methodologies
- **Programming language**
  - Defines a computation model for a computer
  - Several existing programming paradigms

Imperative Programming (1)

- Focus on how a program works

  "First do this and then do that" *(Cooking recipe)*

- The program is at all times in a state

  Corresponding essentially to values in memory

- A statement triggers a state change

  The order of execution of statements is important

- Fortran, Algol, Pascal, Basic, C, etc.

Imperative Programming Example

```
1  int fact(int n)
2  {
3      int result = 1;
4      int i;
5      for (i = 1; i <= n; i++)
6          result *= i;
7      return result
8  }
```

- Program execution for the function call `fact(3)`

Line	`n`	`result`	`i`	*Comment*
1	3	–	–	*passing parameters*
3	3	1	–	*declaring and initialising variable* `result`
4	3	1	undefined	*declaring variable* `i`
5	3	1	1	*initialising variable* `i`
6	3	1	1	*multiplying* `result` *by* `i`
5	3	1	2	*incrementing* `i` *by 1*
6	3	2	2	*multiplying* `result` *by* `i`
5	3	2	3	*incrementing* `i` *by 1*
6	3	6	3	*multiplying* `result` *by* `i`
5	3	6	4	*incrementing* `i` *by 1*

Imperative Programming (2)

- Basic bricks of imperative programming
  - Choice between several subsequences (`if-else`, `switch`, etc.)
  - Repetition of a subsequence (`for`, `while`, `do`, etc.)
- Several variables types to represent the state of the program

  Primitive type, pointer, array, structure, record, etc.
- Additional structuring using procedures

  Arguments passing, local variable, function with return value

Maximum of Three Numbers (1)

- `max` function to find the maximum of three numbers

  Decomposition in two functions `max2` and `max3`

```
1  def max2(a, b):
2      if a > b:
3          return a
4      return b
5
6  def max3(a, b, c):
7      return max2(a, max2(b, c))
8
9  print(max2(12, -2))
10 print(max3(-3, 8, 3))
```

Declarative Programming

- Focus on what a program should do

  In opposition with imperative programming

- Description of the expected results of the program
  - High level description of algorithms
  - No side effects
- XML, SQL, Prolog, Haskell, etc.

Declarative Programming Example

```sql
CREATE TABLE Persons (
    id INT PRIMARY KEY,
    lastName VARCHAR(255),
    firstName VARCHAR(255),
    birthyear INT
);
INSERT INTO Persons VALUES (1, "John", "Doe", 1970);
INSERT INTO Persons VALUES (2, "Jane", "Doe", 1963);

SELECT *, (SELECT MAX(birthyear) FROM Persons)-birthyear AS diff
FROM Persons
```

- "Program" (request) execution decomposed in two steps

  Primary and subquery selections on the `Persons` table

Functional Programming

- Computations are evaluation of mathematical functions

  "Evaluate an expression and use the result for something"

- Execution of a program is a sequence of evaluations

  No states and no mutable data

- The final result only depends on inputs

  $f(x)$ *always gives the same result, for the same* x

- Common Lisp, Erlang, Haskell, Scheme, etc.

Functional Programming Example

```
1  fact :: Int -> Int
2  fact n
3    | n == 0     = 1
4    | otherwise = n * fact (n-1)
5
6  main = do
7     print (fact 0)
8     print (fact 3)
```

- Program execution for the function call `fact 3`

  "fact 3" $\rightarrow$ *"3 * fact (2)"* $\rightarrow$ *"3 * (2 * fact (1))"*
  $\rightarrow$ *"3 * (2 * fact (1))"* $\rightarrow$ *"3 * (2 * (1 * fact (0)))"*
  $\rightarrow$ *"3 * (2 * (1 * 1))"* $\rightarrow$ *"3 * (2 * 1)"* $\rightarrow$ *"3 * 2"* $\rightarrow$ *"6"*

Haskell...


CODE WRITTEN IN HASKELL IS GUARANTEED TO HAVE NO SIDE EFFECTS.
...BECAUSE NO ONE WILL EVER RUN IT?

Maximum of Three Numbers (2)

- `max` function to find the maximum of three numbers

  Decomposition in two functions max2 and max3

```
1  max2 :: Int -> Int -> Int
2  max2 a b
3   | a > b      = a
4   | otherwise = b
5
6  max3 :: Int -> Int -> Int -> Int
7  max3 a b c = max2 a (max2 b c)
8
9  main = do
10    print (max2 12 (-2))
11    print (max3 (-3) 8 3)
```

Maximum of Three Numbers (3)

- `max` function to find the maximum of three numbers

  Decomposition in two functions `max2` and `max3`

$$
\begin{array}{rccl}
max2: & \mathbb{Z} \times \mathbb{Z} & \rightarrow & \mathbb{Z} \\
 & (a,b) & \mapsto & max2(a,b) = \begin{cases} a & \text{if } a > b \\ b & \text{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{rccl}
max3: & \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} & \rightarrow & \mathbb{Z} \\
 & (a,b,c) & \mapsto & max3(a,b,c) = max2(a, max2(b,c))
\end{array}
$$

Object Oriented Programming

- Behaviour associated to structures called objects

  Objects belong to classes, organised as hiearchies

- Objects expose behaviours

  Data and methods to manipulate the objects in a single entity

- Smalltalk, Delphi, Java, C++, C#, etc.

Example: Pure Imperative

- A sequence of instructions executed one after each other

  Each instruction changes the state of the program

```
1  # Information about the product
2  itemname = "Viru Valge Vodka (500 ml)"
3  itemprice = 7.55
4  vat = 0.19
5
6  # Computation of the total price
7  quantity = 10
8  unitprice = (1 + vat) * itemprice;
9  totalprice = quantity * unitprice
10 print("{:d} x {:s} : {:.2f} euros"
11                  .format (quantity, itemname, totalprice))
```

Example: Procedural

- Structuring the program thanks to procedures and functions

  Building reusable units of code

```
1  # Computation of the price including taxes
2  def unitprice(price, vat):
3      return (1 + vat) * price
4
5  # Displaying the total price
6  def printprice(name, unitprice, quantity):
7      totalprice = quantity * unitprice
8      print("{:d} x {:s} : {:.2f} euros"
9                    .format (quantity, name, totalprice))
10
11 # Information about the product
12 itemname = "Viru Valge Vodka (500 ml)"
13 itemprice = 7.55
14
15 if __name__ == '__main__':
16     unitprice = unitprice(itemprice, 0.19)
17     printprice(itemname, unitprice, 10)
```

Example: Object Oriented

- Defining objects to put data and behaviour together

  An item is responsible for its own price

```
1  class Item:
2      def __init__(self, name, price):
3          self.name = name
4          self.price = price
5
6      def unitprice(self, vat):
7          return (1 + vat) * self.price
8
9      def printprice(self, vat, quantity):
10         totalprice = quantity * self.unitprice(vat)
11         print("{:d} x {:s} : {:.2f} euros"
12                   .format (quantity, self.name, totalprice))
13
14 if __name__ == '__main__':
15     vodka = Item("Viru Valge Vodka (500 ml)", 7.55)
16     vodka.printprice(0.19, 10)
```

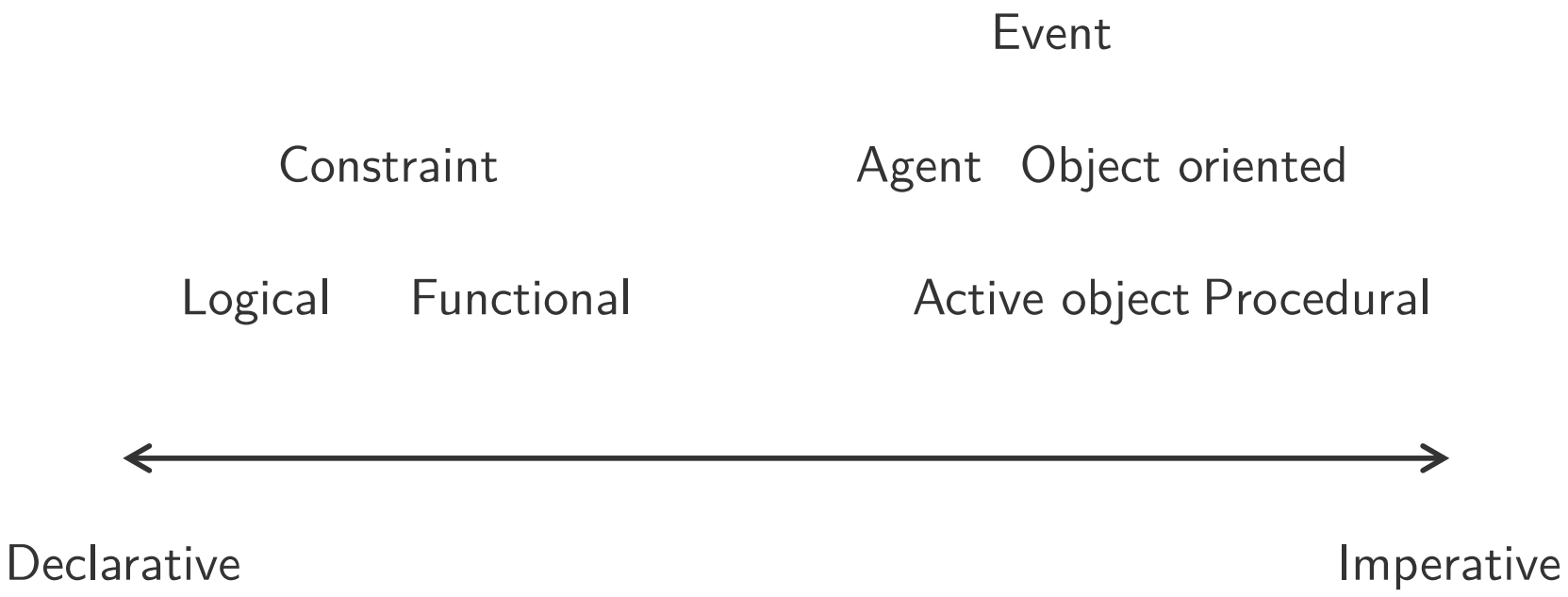
Example: Functional

- Split computations according to functions

  Executing the program is just a function evaluation

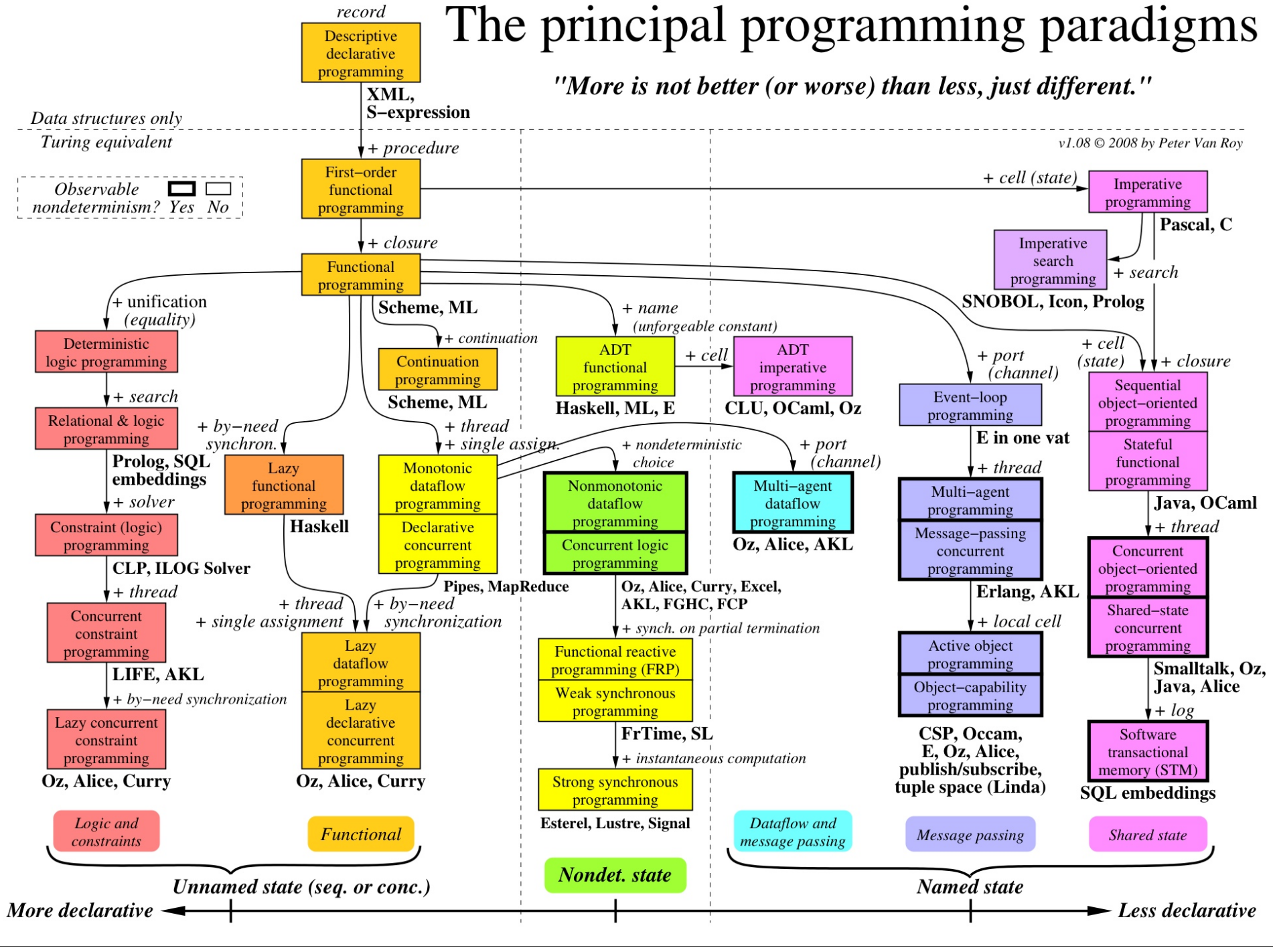
```
1  -- Computation of the price including taxes
2  unitprice :: Double -> Double -> Double
3  unitprice price vat = (1 + vat) * price
4
5  -- Displaying the total price
6  printprice :: String -> Double -> Int -> IO ()
7  printprice name unitprice quantity = do
8      putStr (show quantity ++ " x " ++ name ++ " : " ++ show (
       fromIntegral quantity * unitprice) ++ " euros\n")
9
10 -- Information about the product
11 itemname = "Viru Valge Vodka (500 ml)"
12 itemprice = 7.55
13
14 main = do
15     printprice itemname (unitprice itemprice 0.19) 10
```

Overview of Programming Paradigms



The principal programming paradigms

"More is not better (or worse) than less, just different."



Other Paradigm

Literate Programming

- Logic in a natural language and integrated macros

  Generating an executable and the documentation

- The order of the source code is no longer important

  The program is written according to the logic of the programmer

- Forces the programmer to clearly articulate his/her ideas

  Has to think about the code while documenting it

- CWEB, FunnelWeb, Haskell, CoffeeScript, Julia, etc.

Example: CWEB

```
1  The foo program implements an approximate solution
2  to the traveling salesman problem.
3  <<*>>=
4  <<Header to include>>
5  <<Global variables>>
6  <<Additional functions>>
7  <<Main function>>
8  @
9
10 Several headers must be included to be able to use
11 the tools from the standard library.
12 <<Header to include>>=
13 #include <stdio.h>
14 @
15
16 We also include a library exclusively developed for
17 this program, containing specific tools.
18 <<Header to include>>+=
19 #include "mylib.h"
20 @
```

- Compiling `.c` file with `ctangle` and `.tex` file with `cweave`

Logic Programming

- Computation are expressed as mathematical logic

  "Answer a question via search for a solution"

- Based on the concept of relation

  And axioms, inference rules, requests

- Expressions of facts and rules on the domain

  $Algorithm = Logic + Control$

- Mercury, Prolog, etc.

Example: Prolog

```
1  % The facts
2  % par(X,Y) means that X is a parent of Y
3  par(lloyd, james).
4  par(lloyd, janet).
5  par(ruth, james).
6  par(ruth, janet).
7  par(emma, lloyd).
8  par(katherine, ruth).
9  par(adolph, lloyd).
10 par(edgar, ruth).
11
12 % The relations
13 % grand(X,Z) means that X is a grand-father of Z
14 grand(X, Z) :- par(Y, Z), par(X, Y).
```

- Execution with `gprolog` and the `-consult-file` option

Agent-Oriented Programming

- Building the software based on software agents
  - Interfaces and message exchange capability
  - Goal, belief, event, plan and action
- Perception of the environment to respond to stimuli
  - `event MyEvent` and `on MyEvent`, `on Initialize`, `on Destroy`
  - Event sent with `emit`
- Jade, SARL, 2APL, etc.

Example: SARL

```
1  event MyEvent
2
3  agent MyAgent
4  {
5      uses Logging
6      on MyEvent
7      {
8          println("Event received")
9      }
10
11     on Initialize
12     {
13         println("Initialization")
14     }
15
16     on Destroy
17     {
18         println("Destruction")
19     }
20 }
```

Aspect-Oriented Programming

- Increasing the modularity with aspects

  By handling separately the transversal concerns

- Pointcuts to modify a code by adding an aspect

```
call(void Point.setX(int))
```

- Advices to alter a behaviour

```
before():  move() {
   System.out.println("about to move");
}
```

- AspectJ, AspectC, pytilities, etc.

Example: AspectJ

```
1  public aspect Logging
2  {
3     before(): call(void Plotter.plot())
4     {
5        System.out.println ("About to plot...");
6     }
7
8     after() returning: call(void Plotter.plot())
9     {
10       System.out.println ("Plot finished...");
11    }
12 }
```

Language Concept

Concept

> More is not better (or worse) than less, just different.
>
> —The paradigm paradox

- Record
- Closure
- Independence
- Named state

Programming Language Choice

- Different programming problems

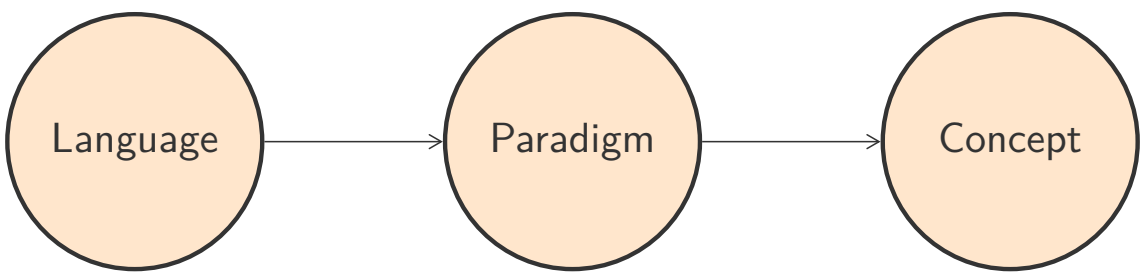
  $\Longrightarrow$ different programming **concepts**

- Modern languages support only a few paradigms

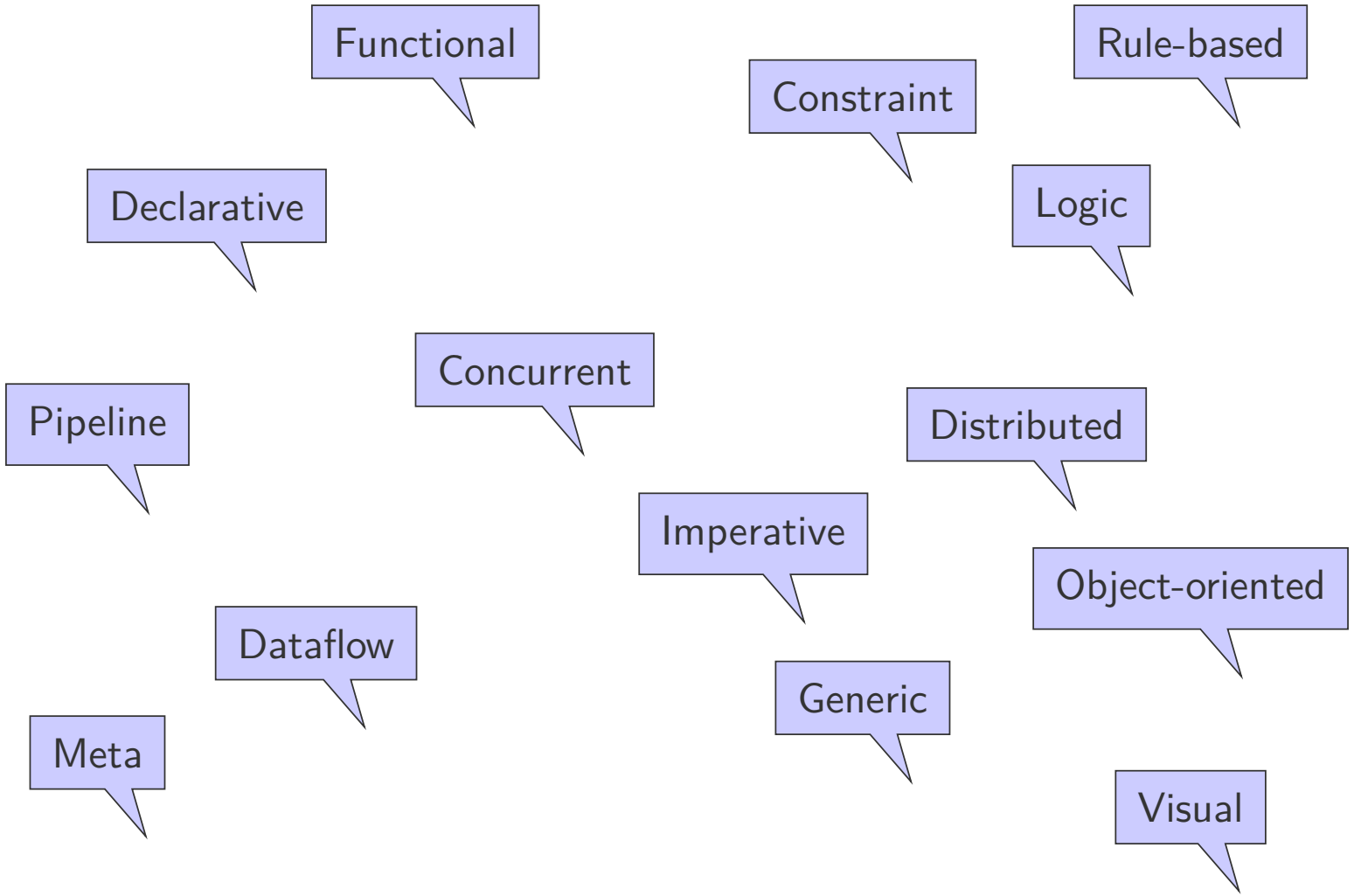
  Multi-paradigms programming

- There are fewer paradigms than languages

  And also fewer concepts than paradigms



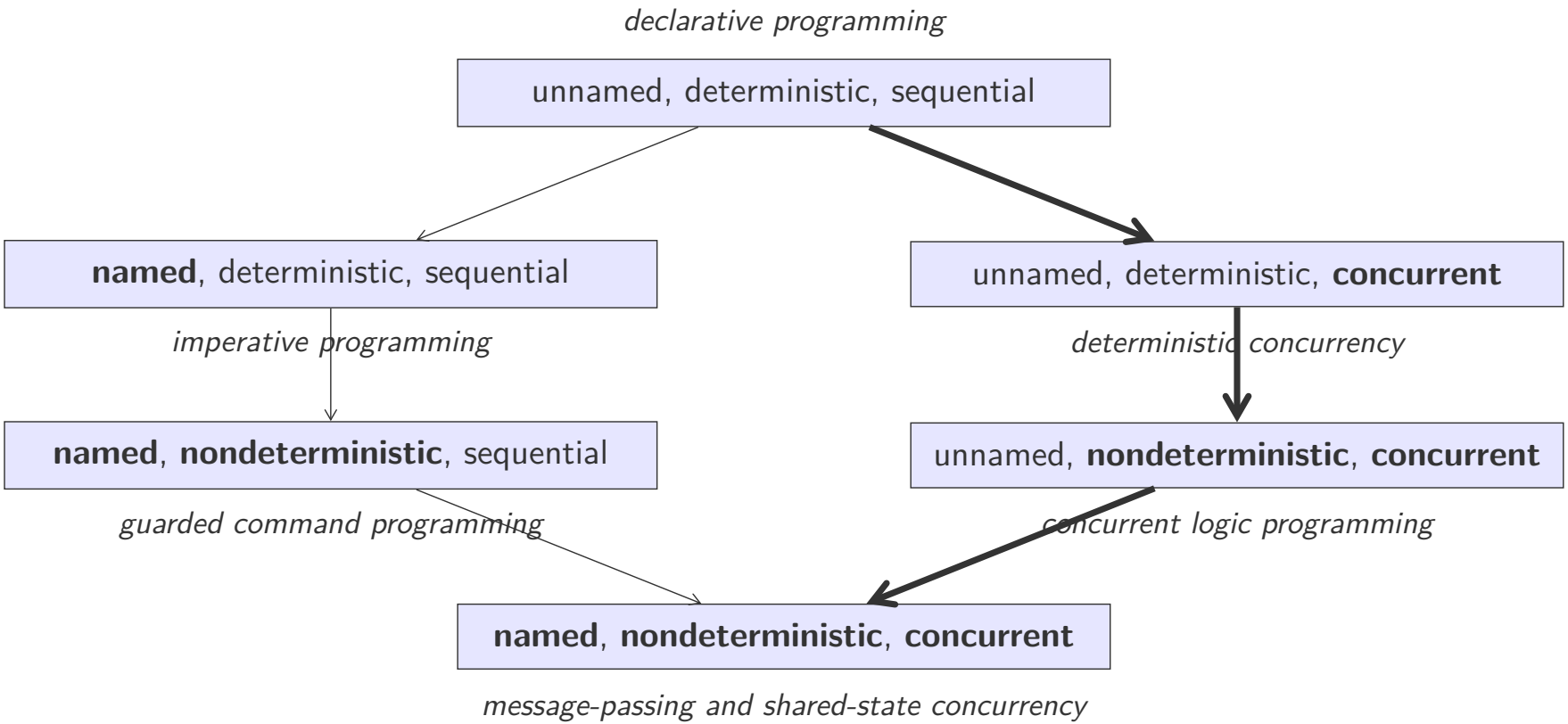
Paradigm Jungle



Two Important Properties

- **Observable non-determinism**
  - Execution not completely determined by its specification
  - Execution choice made by a runtime scheduler
  - The user can identify different executions
  - Not desired, e.g. with timing resulting in race condition
- **Named state**
  - The state allows you to remember information (history)
  - Named or not, deterministic or not, sequential or concurrent

State Support Level



Paradigm Extension

- Modelling independent activities

  Adding concurrency

- Modelling alterable memory

  Adding named states

- Modelling error detection and their correction

  Adding exceptions

Multi-Paradigm Language

- Prolog

  Logical and imperative programming

- Modelling languages

  Solver and object-oriented programming

- Language incorporation

  Embedded language paradigm and host paradigm

Record

- Data structure

  Group of references to data, with access by index

- Basic concept of symbolic programming

  A program can manipulate its own formulas

```
R=food(name:  "Cherry pie", calories:  312.9, category:  dessert)
```

Closure

- Combines a procedure with its external references

  References used during its definition

- Instantiation and genericity, separation of concerns

  Component-based programming

```
1  def build_multiplier(nb):
2      def multiplier(x):
3          return x * nb
4      return multiplier
5
6  mult2 = build_multiplier(2)
7  for i in range(1,11):
8      print("{:d} x 2 = {:d}".format(i, mult2(i)))
```

Independence

- Program consisting of independent parts

  Can communicate at very specific moments

- Concurrency / Parallelism

  Language / hardware related concepts

- Three levels of concurrency

  Distributed systems, OS, activities in a process

Named State

- Abstract notion of time in a program

  We can follow the evolution of the state

- Modelling an entity whose behaviour changes

  Sequence of values in time with the same name

- Major importance for the modularity of a program

  Update of the part of a system, without affecting the rest

References

- Darrion Ramdin, *What is meant by ?Programming Paradigms??*, October 7, 2017. https://medium.com/@darrion/what-is-meant-by-programming-paradigms-9b965a62b7c7
- Adrian Colyer, *Programming paradigms for dummies: what every programmer should know*, January 25, 2019. https://blog.acolyer.org/2019/01/25/programming-paradigms-for-dummies-what-every-programmer-should-know
- Peter Van Roy, *The principal programming paradigms: "More is not better (or worse) than less, just different."*, 2008. http://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng108.pdf.
- Patrick Smyth, *An Introduction to Programming Paradigms*, March 12, 2018. https://digitalfellows.commons.gc.cuny.edu/2018/03/12/an-introduction-to-programming-paradigms
- Yevgeniy Brikman, *Six programming paradigms that will change how you think about coding*, April 9, 2014. https://www.ybrikman.com/writing/2014/04/09/six-programming-paradigms-that-will/
- Jerry Reghunadh, & Neha Jain, *Selecting the optimal programming language: Factors to consider*, September 13, 2011. https://www.ibm.com/developerworks/library/wa-optimal

Credits

- island home, March 2012, https://www.flickr.com/photos/lawson_matthews/7277768736.
- Scott Adams (Dilbert), November 3, 1991, http://dilbert.com/strip/1991-11-03.
- XKCD, January 3, 2014, http://www.explainxkcd.com/wiki/index.php/1312:_Haskell.
- Goran Konjevod, June 30, 2009, https://www.flickr.com/photos/23913057@N05/3718793346.
- adchen, April 1, 2009, https://www.flickr.com/photos/adchen00/4374846426.