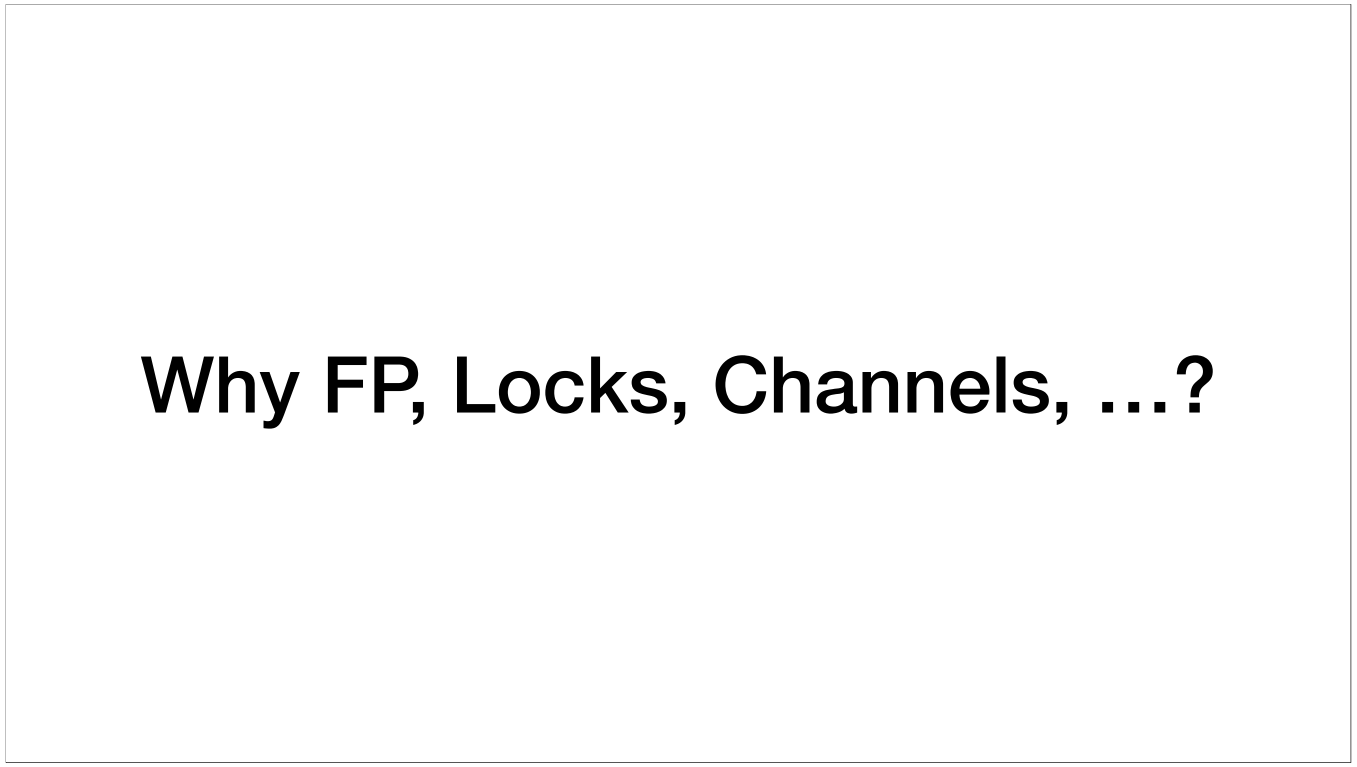
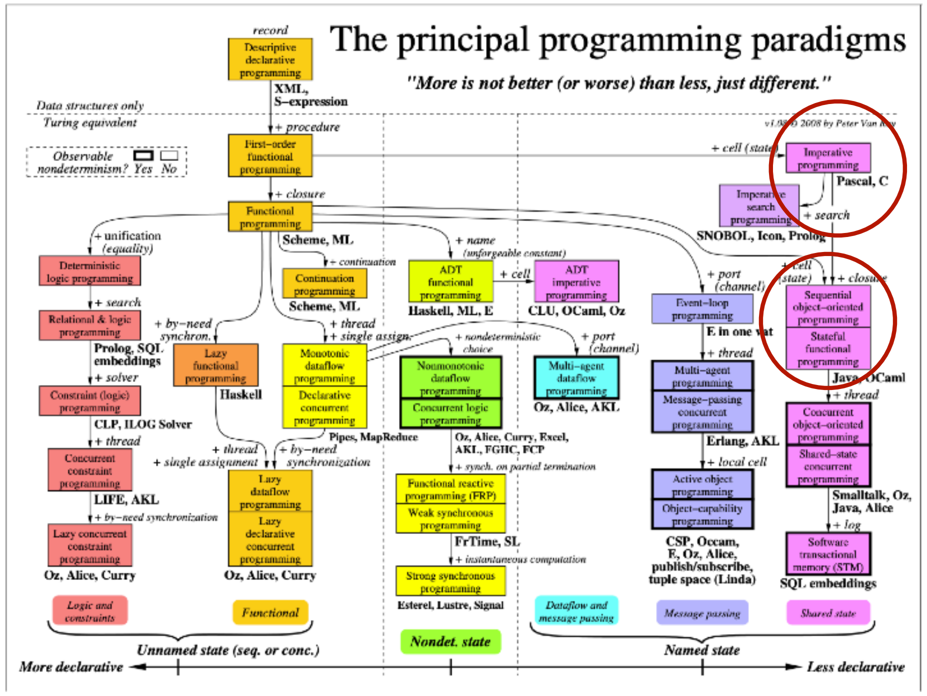
Conclusion

florian.felten@uni.lu



Why FP, Locks, Channels, …?


The principal programming paradigms
"More is not better (or worse) than less, just different."
v1.08 © 2008 by Peter Van Roy
record
Descriptive declarative programming
XML, S–expression
Data structures only
Turing equivalent
Observable nondeterminism? Yes No
+ procedure
First–order functional programming
+ cell (state)
Imperative programming
Pascal, C
Imperative search programming
+ search
SNOBOL, Icon, Prolog
+ closure
Functional programming
Scheme, ML
+ unification (equality)
Deterministic logic programming
+ search
Relational & logic programming
Prolog, SQL embeddings
+ solver
Constraint (logic) programming
CLP, ILOG Solver
+ thread
Concurrent constraint programming
LIFE, AKL
+ by–need synchronization
Lazy concurrent constraint programming
Oz, Alice, Curry
+ continuation
Continuation programming
Scheme, ML
+ by–need synchron.
Lazy functional programming
Haskell
+ thread
+ single assign.
Monotonic dataflow programming
Declarative concurrent programming
Pipes, MapReduce
+ thread
+ single assignment
+ by–need synchronization
Lazy dataflow programming
Lazy declarative concurrent programming
Oz, Alice, Curry
+ name (unforgeable constant)
ADT functional programming
Haskell, ML, E
+ cell
ADT imperative programming
CLU, OCaml, Oz
+ nondeterministic choice
Nonmonotonic dataflow programming
Concurrent logic programming
Oz, Alice, Curry, Excel, AKL, FGHC, FCP
+ synch. on partial termination
Functional reactive programming (FRP)
Weak synchronous programming
FrTime, SL
+ instantaneous computation
Strong synchronous programming
Esterel, Lustre, Signal
+ port (channel)
Multi–agent dataflow programming
Oz, Alice, AKL
+ port (channel)
Event–loop programming
E in one vat
+ thread
Multi–agent programming
Message–passing concurrent programming
Erlang, AKL
+ local cell
Active object programming
Object–capability programming
CSP, Occam, E, Oz, Alice, publish/subscribe, tuple space (Linda)
+ cell (state)
+ closure
Sequential object–oriented programming
Stateful functional programming
Java, OCaml
+ thread
Concurrent object–oriented programming
Shared–state concurrent programming
Smalltalk, Oz, Java, Alice
+ log
Software transactional memory (STM)
SQL embeddings
Logic and constraints
Functional
Unnamed state (seq. or conc.)
Nondet. state
Dataflow and message passing
Message passing
Shared state
Named state
More declarative
Less declarative

Prof. Peter Van Roy
(eating cake)

What you had seen before

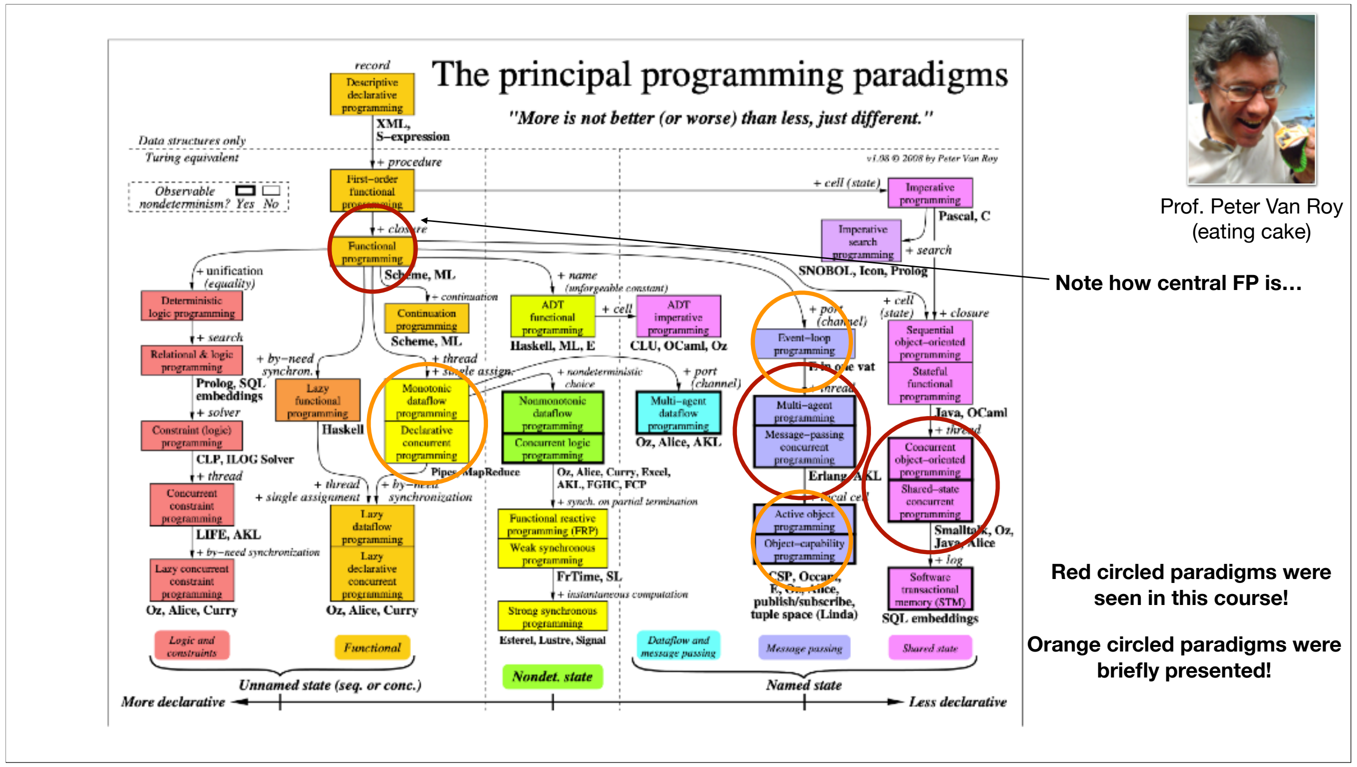
A *programming paradigm* is a style of programming a computer that is defined by a specific set of programming concepts and techniques.

You might have now realised that there are many ways to think when programming.

Paradigms are based on different concepts. For example, functional programming doesn't allow mutability, while object-oriented does. Yet **they are equally powerful**! In this case, we say the paradigms are *Turing equivalent.*

Classical programming languages generally support one main paradigm e.g. C, Haskell, Java. However, there exists languages which support multiple paradigms natively e.g. Scala, Oz, Alice.

This article by Prof. Van Roy discusses the matter: https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf.


The principal programming paradigms
"More is not better (or worse) than less, just different."
v1.08 © 2008 by Peter Van Roy
record
Descriptive declarative programming
XML, S–expression
Data structures only
Turing equivalent
Observable nondeterminism? Yes No
+ procedure
First–order functional programming
+ cell (state)
Imperative programming
Pascal, C
Imperative search programming
+ search
SNOBOL, Icon, Prolog
+ closure
Functional programming
Scheme, ML
+ unification (equality)
Deterministic logic programming
+ search
Relational & logic programming
Prolog, SQL embeddings
+ solver
Constraint (logic) programming
CLP, ILOG Solver
+ thread
Concurrent constraint programming
LIFE, AKL
+ by–need synchronization
Lazy concurrent constraint programming
Oz, Alice, Curry
+ continuation
Continuation programming
Scheme, ML
+ by–need synchron.
Lazy functional programming
Haskell
+ thread
+ single assign.
Monotonic dataflow programming
Declarative concurrent programming
Pipes, MapReduce
+ thread
+ single assignment
+ by–need synchronization
Lazy dataflow programming
Lazy declarative concurrent programming
Oz, Alice, Curry
+ name (unforgeable constant)
ADT functional programming
Haskell, ML, E
+ cell
ADT imperative programming
CLU, OCaml, Oz
+ nondeterministic choice
Nonmonotonic dataflow programming
Concurrent logic programming
Oz, Alice, Curry, Excel, AKL, FGHC, FCP
+ synch. on partial termination
Functional reactive programming (FRP)
Weak synchronous programming
FrTime, SL
+ instantaneous computation
Strong synchronous programming
Esterel, Lustre, Signal
+ port (channel)
Multi–agent dataflow programming
Oz, Alice, AKL
+ port (channel)
Event–loop programming
E in one vat
+ thread
Multi–agent programming
Message–passing concurrent programming
Erlang, AKL
+ local cell
Active object programming
Object–capability programming
CSP, Occam, E, Oz, Alice, publish/subscribe, tuple space (Linda)
+ cell (state)
+ closure
Sequential object–oriented programming
Stateful functional programming
Java, OCaml
+ thread
Concurrent object–oriented programming
Shared–state concurrent programming
Smalltalk, Oz, Java, Alice
+ log
Software transactional memory (STM)
SQL embeddings
Logic and constraints
Functional
Nondet. state
Dataflow and message passing
Message passing
Shared state
Unnamed state (seq. or conc.)
Named state
More declarative
Less declarative

Prof. Peter Van Roy
(eating cake)

Note how central FP is…

Red circled paradigms were seen in this course!

Orange circled paradigms were briefly presented!

So far, we have seen some paradigms but we cannot see all of them in one course! This slide is just to give an idea that once again, it is just the tip of the iceberg 😢.

In the end, there is no “best paradigm”, and there is no “best language”. As computer scientists, our job is to be aware of the variety of techniques out there and to **use the one which fits the most** to the problem. Also, we should be careful to **use just the right amount of expressiveness** to solve our problems.

“If you just need a banana, don’t import the whole jungle. The gorillas won’t serve you and might cause trouble” 😉.

This article by Prof. Van Roy discusses the matter: https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf.

This course

- ✓ **Introduce new paradigms (programming styles)**
- ✓ **Insights on how vast the programming world is**
- ✓ **Insights on some key techniques coming in various languages and systems**

Don't master a language, master the concepts!
Good programmers adapt quickly to new
environments.

Fill the course feedback!

Good luck for the exams