Computer Programming with Scala

Martin Quinson

September 2016

About me

- Prof ENS Rennes depuis 2015 (avant: MCF Université de Lorraine depuis 2005)
- Thèse ENS Lyon, après Deug+Licence+Maîtrise Université St Etienne
- Équipe recherches: Myriads (IRISA = U. Rennes I/INRIA/CNRS/ENS Rennes)

Recherche: Méthodologies expérimentales

- Évaluer des applications distribuées (perfs, bugs)
- Projet SimGrid: Simulateur de systèmes distribués
- Vérification formelle (model-checking), virtualisation

Enseignements: Algorithmique et programmation

- Initiation, Java/Scala, AlgoProg, C seconde langue
- Prog Système; Algo dist; P2P; Prog répartie

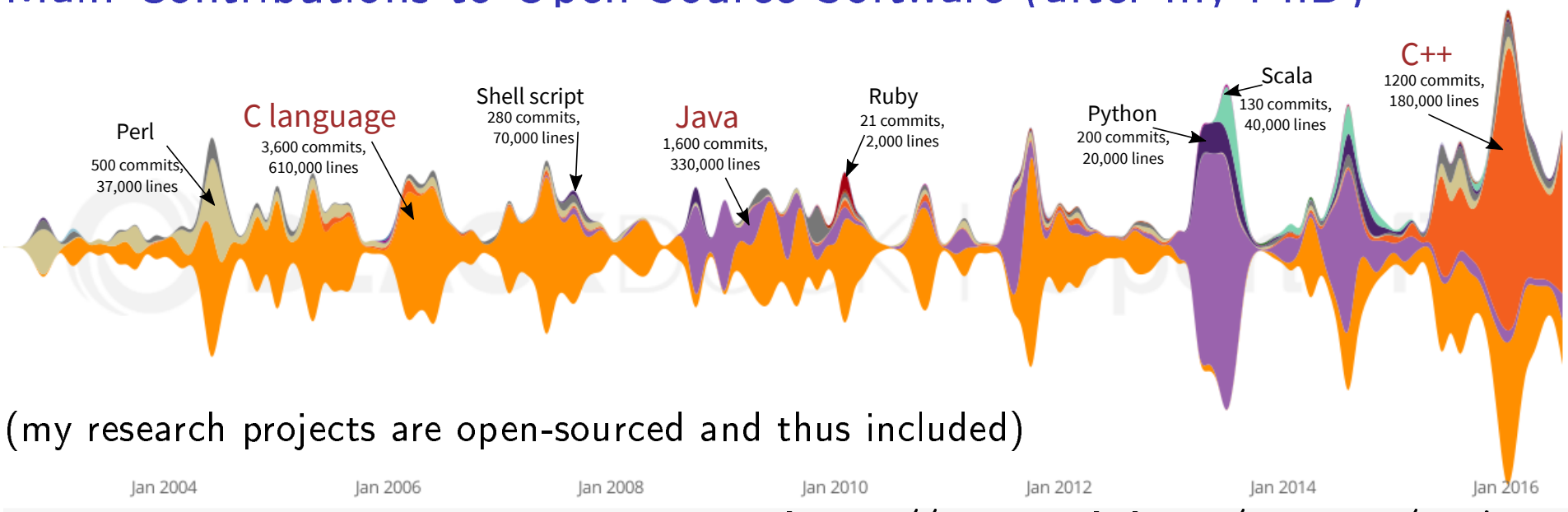
Activités complémentaires:

- Beaucoup de médiation scientifique & pédagogie
- PLM: Exerciseur de programmation
- Developpeur Debian depuis 2004 (l10n, quilt, jeux)

Me as a Programmer

- First language: Logo (1983? 1984?). Self taught in Basic (I was young!)
- A lot of small Pascal programms before my first CS class (at University)
- Graduation project in OCaml (1999)

Main Contributions to Open Source Software (after my PhD)



(my research projects are open-sourced and thus included)

`https://www.openhub.net/accounts/mquinson`

I love building large systems, I must confess

(but I'm not that fluent in FP)

About this course

General Objective

- Be prepared to be a Computer Scientist, in a few years
- Get the grasp on programming (?)

Practical Objective

- Learn a new language that you shall soon all love: Scala
- Learn how to organize your code (mostly OOP... with bits of FP)

Agenda and Organization

- Courses + Practical, as usual
- Evaluation: A "big" project, to be announced
- Please be patient: that's my first year here (report typo, comments per email)

Today

- Try to convince you that programming is both important and interesting
- Basics of Scala

Informatics? What does that mean?

Comment définiriez vous «l'informatique»? Et la science afférente?

Informatics? What does that mean?

Comment définiriez vous «l'informatique»? Et la science afférente?

Wikipédia + Google translate

- Anglais, Hébreux: ingénierie (Institute of Electrical and Electronics Engineers)
- Portuguais: Shannon, théorie de l'information
- Allemand, Russe: linguistique (chomsky comme papa de l'informatique) (Bielefeld)
- Français, Italien: calculabilité, théorie des graphes, décidabilité

Informatics? What does that mean?

Comment définiriez vous «l'informatique»? Et la science afférente?

Wikipédia + Google translate

- Anglais, Hébreux: ingénierie (Institute of Electrical and Electronics Engineers)
- Portuguais: Shannon, théorie de l'information
- Allemand, Russe: linguistique (chomsky comme papa de l'informatique) (Bielefeld)
- Français, Italien: calculabilité, théorie des graphes, décidabilité

Et vis-à-vis des maths ? De la physique? Des autres disciplines?

Informatics? What does that mean?

Comment définiriez vous «l'informatique»? Et la science afférente?

Wikipédia + Google translate

- Anglais, Hébreux: ingénierie (Institute of Electrical and Electronics Engineers)
- Portuguais: Shannon, théorie de l'information
- Allemand, Russe: linguistique (chomsky comme papa de l'informatique) (Bielefeld)
- Français, Italien: calculabilité, théorie des graphes, décidabilité

Et vis-à-vis des maths ? De la physique? Des autres disciplines?

L'informatique fait partie des mathématiques!!! (air connu :-)

- On peut retourner l'argument pour le plaisir du troll
- Modèles algorithmiques $\supseteq$ modèles équationnels $\Rightarrow$ maths $\subseteq$ informatique
- Design de processeur $\notin$ maths $\Rightarrow$ maths $\subset$ informatique :D

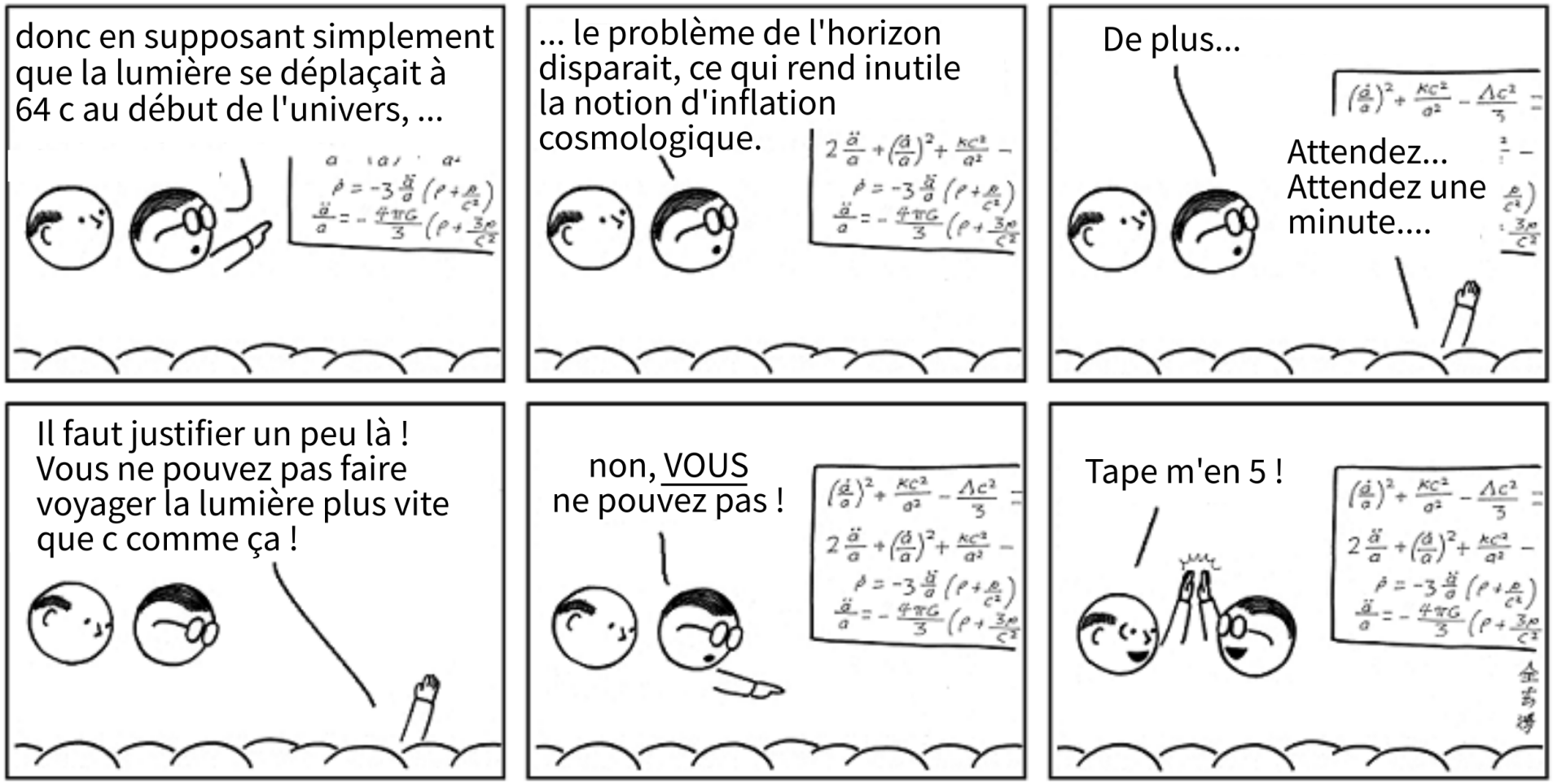
C'est juste pour le plaisir du troll

- Je n'ai rien contre les maths au contraire, elles m'ont énormément apporté
- "On peut écrire des logiciels sans réaliser que l'on fait [aussi] des maths" ;)
- Un troll, ce n'est pas sérieux et il ne faut pas en abuser

Vers une épistémologie de l'informatique

- [Dowek] La science des algorithmes, information, langages, machines
- Physique: teste les contingences; Mathématiques: démontre les nécessités
  Informatique: teste les nécessités, ou démontre les contingences (ou autre)
  - IMRAD vs Problème, solution, éval, conclusion vs Axiome, théorèmes, preuve

Vers une épistémologie de l'informatique



Les matheux ne sont plus admis dans les conférences de physiciens.

http://abstrusegoose.com/316

Vers une épistémologie de l'informatique

- [Dowek] La science des algorithmes, information, langages, machines
- Physique: teste les contingences; Mathématiques: démontre les nécessités
  Informatique: teste les nécessités, ou démontre les contingences (ou autre)
  - IMRAD vs Problème, solution, éval, conclusion vs Axiome, théorèmes, preuve
- [Denning] confluence de maths, ingénierie et sciences naturelles
  - U. Heidelberg: faculté "Maths, Sciences nat et Info" (similaires et différents)
  - (on y revient)

Le diable est dans les détails [Varenne 2009]

- En maths, l'abstraction supprime les détails pour ne laisser que la généralité
- En science nat, on catégorise selon les détails puis on raisonne sur les groupes
- En informatique, l'abstraction *masque* les détails, mais ils restent présents dans la **pyramide des symboles** mise en œuvre
- C'est à mes yeux l'essence même de la programmation: définir des méthodes
  (et c'est pour ça que même les théoriciens doivent apprendre à programmer)

Une preuve d'informaticien

Le théorème de Feit-Thompson (1963)

- Tout groupe fini dont le nombre d'éléments est impair est résoluble
- Le groupe de Fischer-Griess compte environ $8 \cdot 10^{53}$ éléments (mais c'est pair)
- Hypothèse géométrique, conclusion algébrique, preuve: théorie représentations
- La preuve originelle: 255 pages, étendues depuis en deux livres

Breaking News (2012)

- Théorème de Feit-Thompson redémontré formellement [Inria/MSR, Gonthier]
- La nouvelle preuve est entièrement mécanique (grâce à Coq)
- 6 ans de travail, 15000 définitions, 4200 théorèmes, 170000 lignes de code...

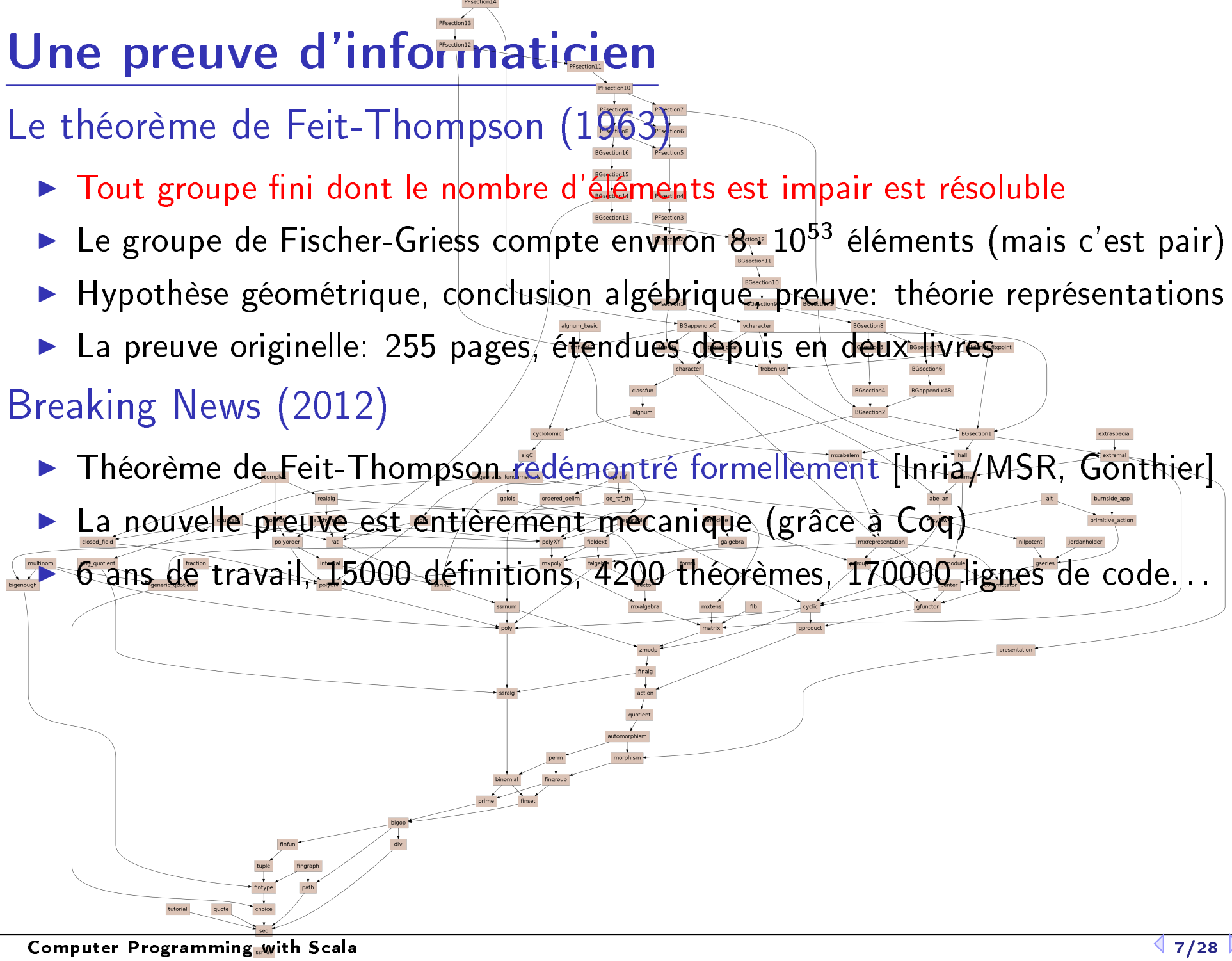
Une preuve d'informaticien

Le théorème de Feit-Thompson (1963)

- Tout groupe fini dont le nombre d'éléments est impair est résoluble
- Le groupe de Fischer-Griess compte environ $8 \cdot 10^{53}$ éléments (mais c'est pair)
- Hypothèse géométrique, conclusion algébrique, preuve: théorie représentations
- La preuve originelle: 255 pages, étendues depuis en deux livres

Breaking News (2012)

- Théorème de Feit-Thompson redémontré formellement [Inria/MSR, Gonthier]
- La nouvelle preuve est entièrement mécanique (grâce à Coq)
- 6 ans de travail, 15000 définitions, 4200 théorèmes, 170000 lignes de code...



Une preuve d'informaticien

Le théorème de Feit-Thompson (1963)

- Tout groupe fini dont le nombre d'éléments est impair est résoluble
- Le groupe de Fischer-Griess compte environ $8 \cdot 10^{53}$ éléments (mais c'est pair)
- Hypothèse géométrique, conclusion algébrique, preuve: théorie représentations
- La preuve originelle: 255 pages, étendues depuis en deux livres

Breaking News (2012)

- Théorème de Feit-Thompson redémontré formellement [Inria/MSR, Gonthier]
- La nouvelle preuve est entièrement mécanique (grâce à Coq)
- 6 ans de travail, 15000 définitions, 4200 théorèmes, 170000 lignes de code...

Pourquoi? Et après? Et alors?

- Personne ne doutait de la preuve, mais maintenant on est sûrs :)
- Formalisation de l'univers mathématique de la preuve, réutilisable
- La classification des groupes finis simples: 10,000 à 20,000 pages manuelles
- Vous voyez ce que Varenne veut dire par le traitement spécifique du détail?

http://images.math.cnrs.fr/Coq-et-caracteres.html

Foundations of Computing

Fundamental mathematical and logical structures

- To understand computing [Turing 36], and its difficulties
- To analyze and verify the correctness of software and hardware

Main issues of interest in Computer Science in France

- Calculability
  - Given a problem, can we show whether there exist an algorithm solving it?
  - Which are the problems for which no algorithm exist? How to categorize them?
- Complexity
  - How long does my algorithm need to answer? (as function of input size)
  - How much memory does it take?
  - Is my algorithm optimal, or does a better one exist?
- Correctness
  - Can we be certain that a given algorithm always reaches a solution?
  - Can we be certain that a given algorithm always reaches the right solution?

But don't forget Denning!

- Computer Science: convergence of Maths, Natural Science and Engineering

Computational Thinking [Wing 2005]

Thinking like a computer scientist means more than programming a computer. It requires thinking at multiple levels of abstraction.

- Methods and models to solve problems efficiently: Récursivity, reductions, etc.
- *Complements and combines mathematical and engineering thinking*
- *A way that humans, not computers, think*

Equipped with computing devices, we use our cleverness to tackle problems we would not dare take on before the age of computing and build systems with functionality limited only by our imaginations.

`https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf`

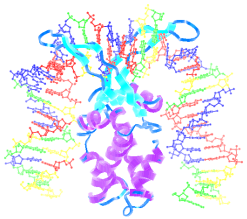
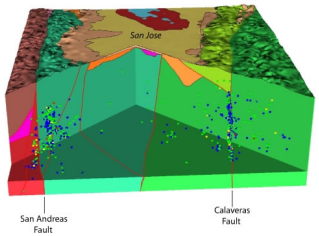
- Impact of Informatics on Sciences $\gg$ Impact of Computers
  - Biologists see the genom as a Code
  - Lawyers (should) write unambiguous laws, that is, algorithmic laws
- Conclusion: Computer Science has an important heritage from Enginering

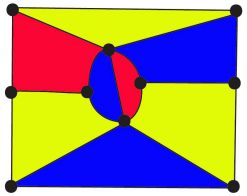
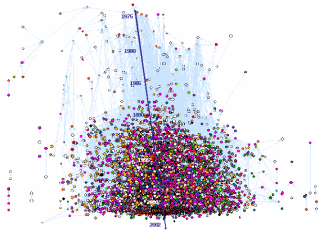
Computational Science

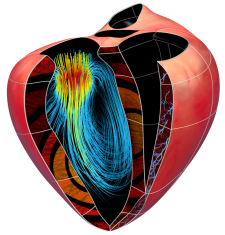
Computational Science in a Nutshell

- Mathematical Models of Complex Phenomenons
- Simulation on Super-Computers
- (In)validation: compare predictions vs. observations
- Then, get results without doing any experience (!)

Computational Science

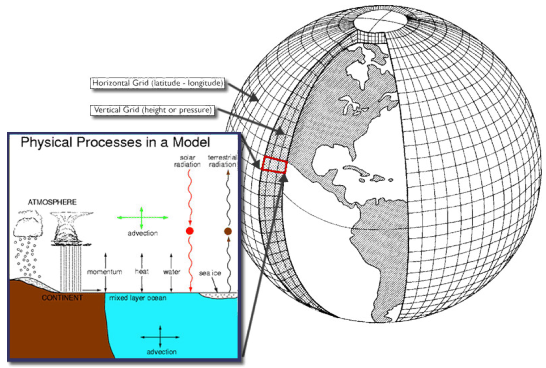



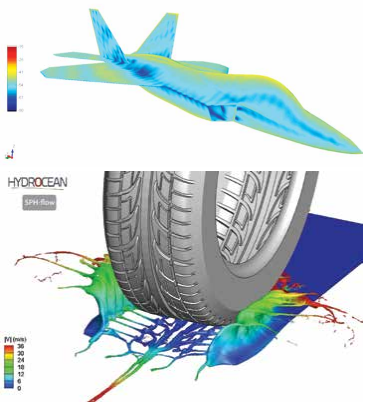



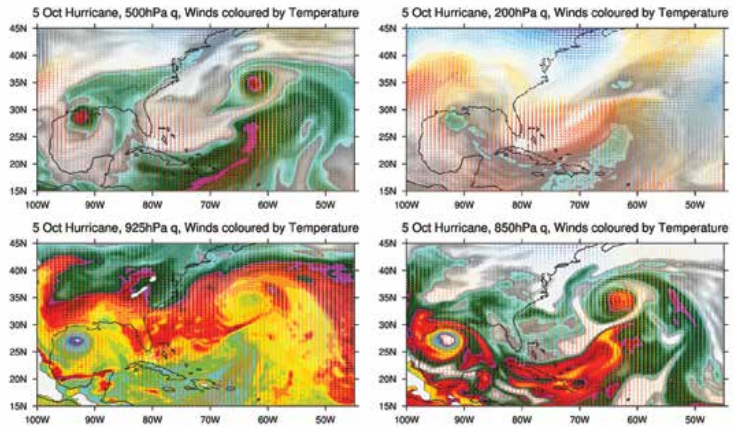


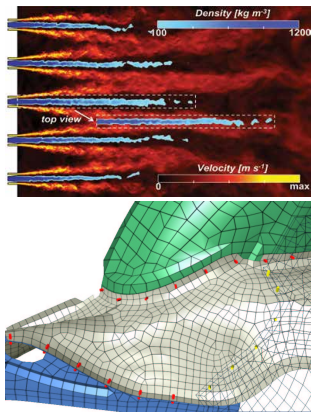
Computational Science in a Nutshell

- Mathematical Models of Complex Phenomenons
- Simulation on Super-Computers
- (In)validation: compare predictions vs. observations
- Then, get results without doing any experience (!)





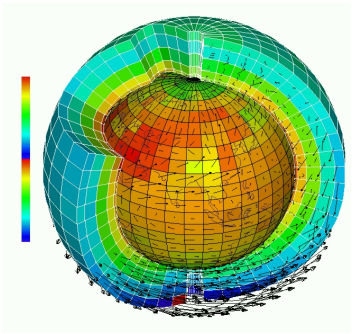




The third Pillar of Science

Doing Science = Acquiring Knowledge

$$\frac{\partial}{\partial x_j}\left(\frac{\partial \Phi}{\partial x_i}\right) = \frac{\partial}{\partial x_i}\left(\frac{\partial \Phi}{\partial x_j}\right)$$



Experimental Science

- Thousand years ago
- Observations-based
- Can describe
- Prediction tedious

Theoretical Science

- Last few centuries
- Equations-based
- Can understand
- Prediction long

Computational Science

- Nowadays
- Compute-intensive
- Can simulate
- Prediction easier

Prediction is very difficult, especially about the future. – Niels Bohr

Observation still bases Science (to feed models)

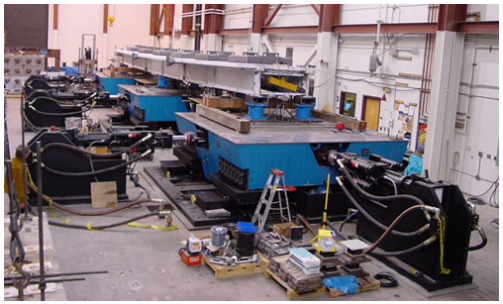
Space telescope

Large Hadron Collider

Mars Explorer

Tsunamis

Earthquake vs. Bridge



Climate vs. Ecosystems

NMR Spectroscope

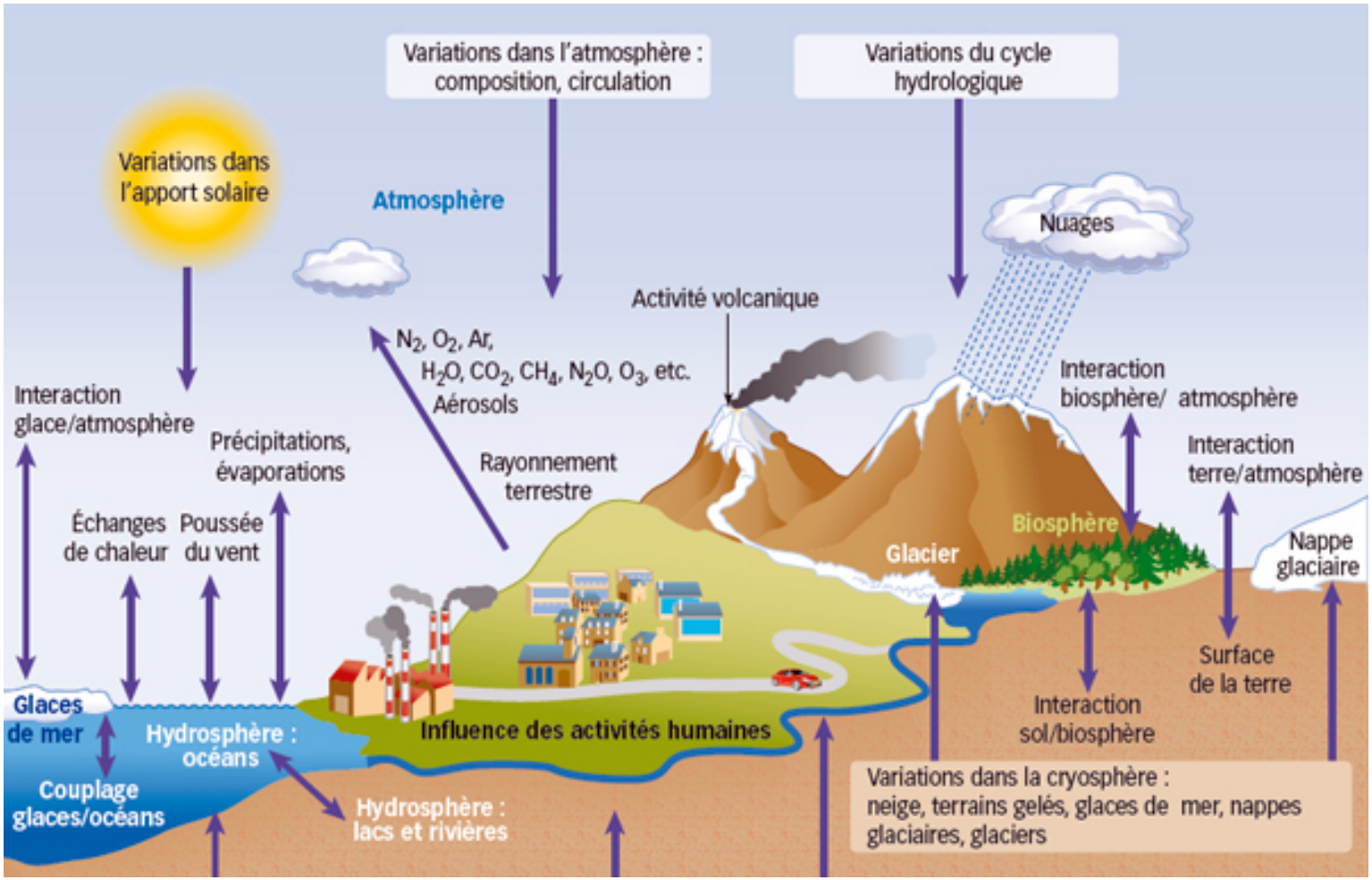
Synchrotrons

Turntable

(who said that science is not fun??)

Computational Science in Practice

How do we build a Model in Climatology?



[Eric Blayo]

1 - Modeling: translate reality into equations

$$\frac{\partial u}{\partial t} + \mathbf{U}.\nabla u - fv + \frac{1}{\rho_0}\frac{\partial p}{\partial x} - \mathcal{F}(u) = 0 \qquad \frac{\partial T}{\partial t} + \mathbf{U}.\nabla T - \mathcal{D}_T(T) = 0$$

$$\frac{\partial v}{\partial t} + \mathbf{U}.\nabla v + fu + \frac{1}{\rho_0}\frac{\partial p}{\partial y} - \mathcal{F}(v) = 0 \qquad \frac{\partial S}{\partial t} + \mathbf{U}.\nabla S - \mathcal{D}_S(S) = 0$$

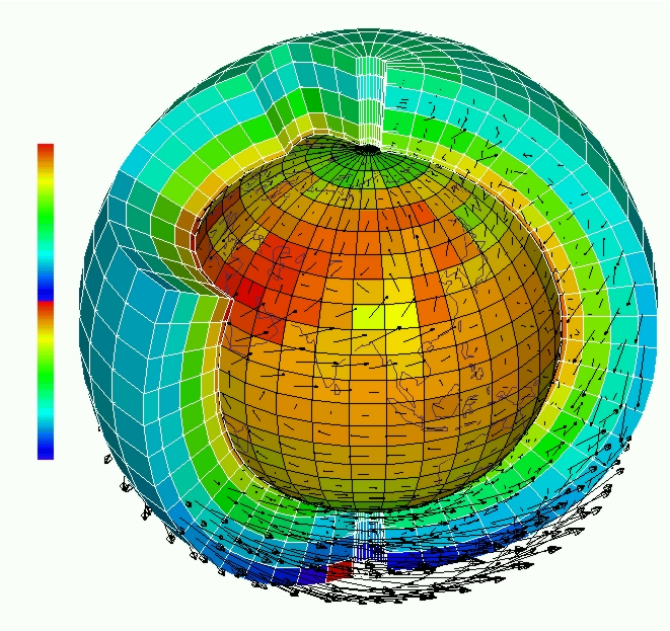
$$\frac{\partial p}{\partial z} = -\rho\, g \qquad \rho = \rho(T, S, p)$$

$$\text{div}\ \mathbf{U} = 0 \qquad + \text{ conditions aux limites}$$

The Ocean, as seen by a mathematician

[Eric Blayo]

2 - Simulating: solve Equations on Computers

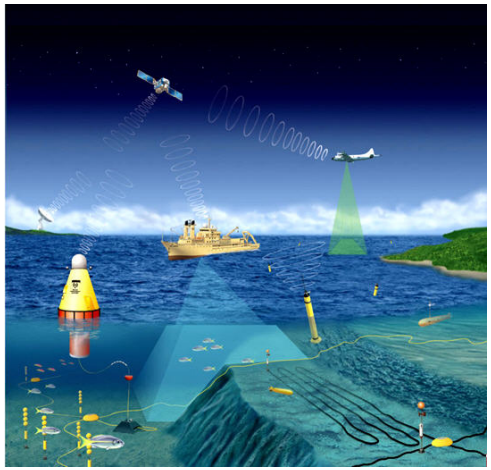


+

[Eric Blayo]

3 - Tuning the model with the Observations

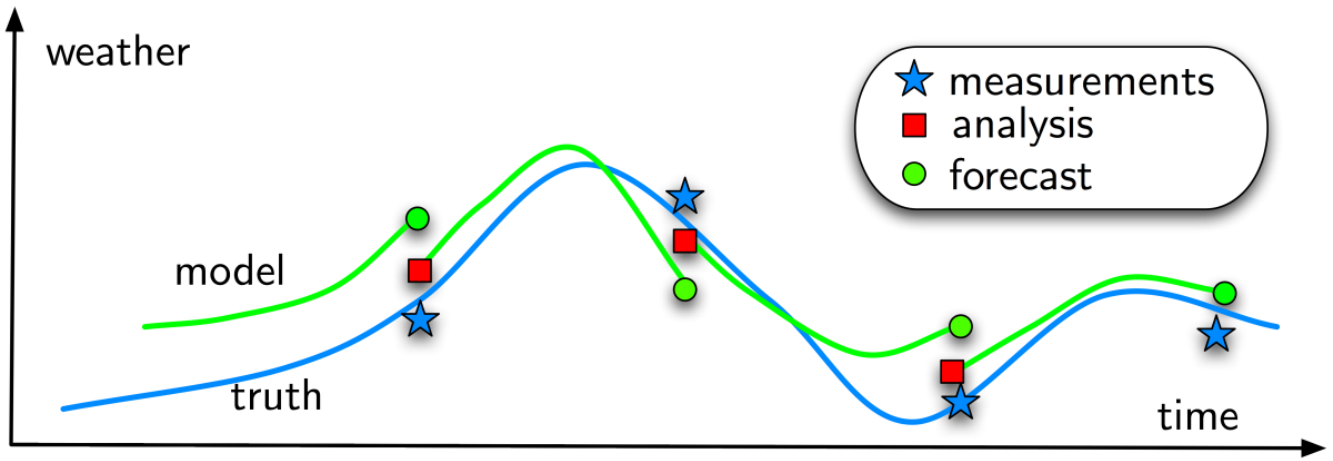
Reconstitute the current situation from the available information



Past and Present Observations

$$
\begin{cases}
\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \dot{\sigma}\frac{\partial u}{\partial \sigma} - f(v - v_g) + \frac{\partial \phi}{\partial x}|_p + F_u = 0 \\
\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \dot{\sigma}\frac{\partial v}{\partial \sigma} + f(u - u_g) + \frac{\partial \phi}{\partial y}|_p + F_v = 0 \\
\frac{\partial \phi}{\partial (T/\theta)} = -C_p\theta(1 + 0.85q_v) \\
\frac{\partial p_\star}{\partial t} = -\int_0^1 [\frac{\partial (p_\star u)}{\partial x} + \frac{\partial (p_\star v)}{\partial y}]d\sigma \\
\frac{\partial \sigma}{\partial t} = \frac{1}{p_\star}\{\sigma \int_0^1 [\frac{\partial (p_\star u)}{\partial x} + \frac{\partial (p_\star v)}{\partial y}]d\sigma - \int_0^\sigma [\frac{\partial (p_\star u)}{\partial x} + \frac{\partial (p_\star v)}{\partial y}]d\sigma\} \\
\frac{\partial \theta}{\partial t} + u\frac{\partial \theta}{\partial x} + v\frac{\partial \theta}{\partial y} + \dot{\sigma}\frac{\partial \theta}{\partial \sigma} + \frac{\partial R_{rad}}{\partial \sigma} + F_\theta + p_\theta = 0 \\
\frac{\partial q_v}{\partial t} + u\frac{\partial q_v}{\partial x} + v\frac{\partial q_v}{\partial y} + \dot{\sigma}\frac{\partial q_v}{\partial \sigma} + F_{q_v} + p_{q_v} = 0
\end{cases}
$$

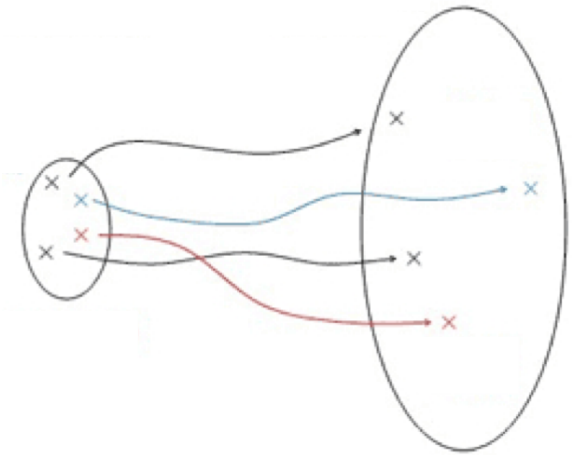
Mathematical Models

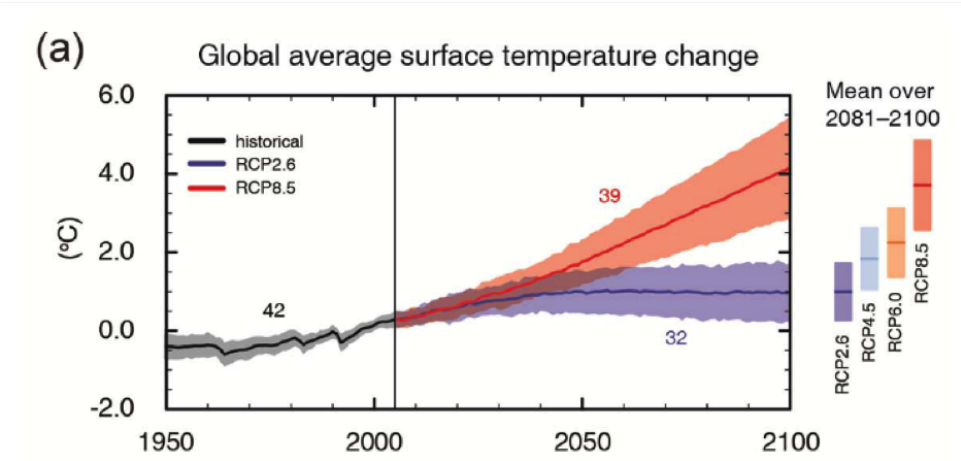


[Eric Blayo]

4 - Estimate the uncertaincy

Noised Initial Conditions + Numerous Simulations $\rightsquigarrow$ Confidence Intervals

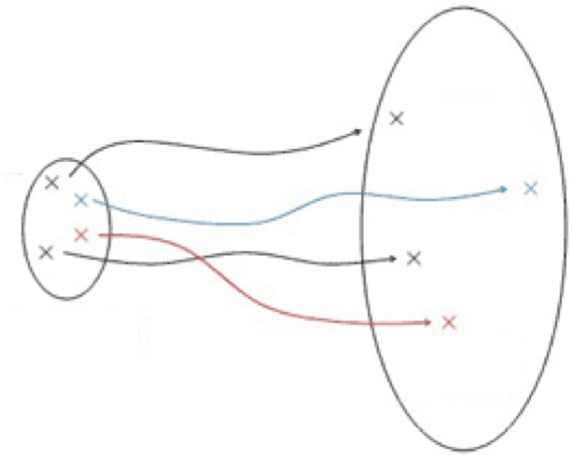


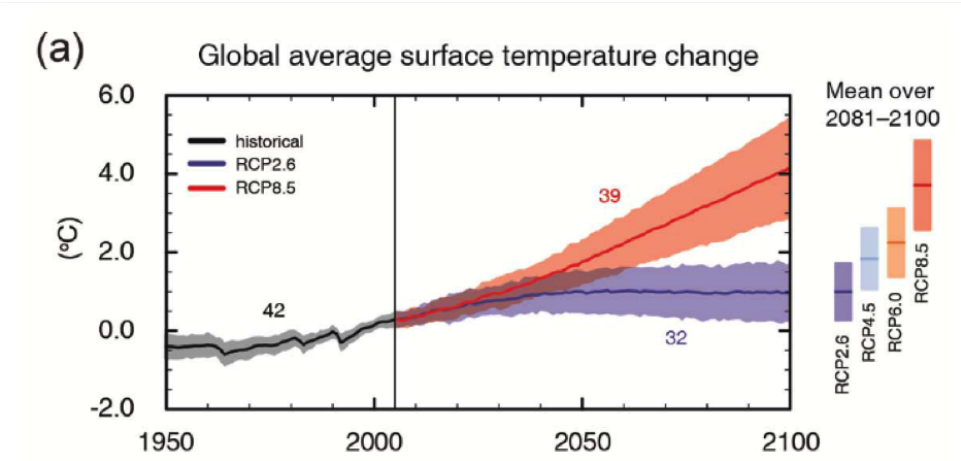


[Eric Blayo]

4 - Estimate the uncertaincy

Noised Initial Conditions + Numerous Simulations $\rightsquigarrow$ Confidence Intervals





[Eric Blayo]

Conclusions

- Major revolution: ever heard of experimental maths? or experimental history?
- No "Good Model". There is only models more or less adapted to the study
- Open Science: Ensuring the Experimental Reproducibility, as in Natural Science
- Conclusion 1: Computer Science has an heritage in Natural Science
  (also in Cognitive Science and IA and other parts of CS)
- Conclusion 2: Computer Scientists are very bad in Computational Science!
  But things slowly improve, so get ready! **Learn programming! Now!**

Crash Course on Scala

Scala is Java as it *should* be

What is all the hype about Scala?

- Pleasant: The syntax is simple and elegant
- Multi-paradigm: Functional (+ properties) and OOP (+ mixin and singleton)
- Typed: not as strongly as Caml, but implicits make the life much easier
- JVM-based: You gain Java libs, aggressive optimizations and portability
- Actor models: At least, parallelism made easier than threads

The Bad Points

- Mixing paradigms $\rightsquigarrow$ Frankenstein effect, tricky to go along
- The error messages are sometimes pure hells
- The language and runtime are still moving targets (longevity?)

Why you will love it?

- Often clever and sometimes tricky. Who don't like mind challenges?
- Pretend you coded it in Java, and enjoy the beauty of Scala

Starting Scala

Installation: Get it from `http://scala-lang.org/` (version 2.11 at best)

Executing your code

myfile.scala

```
println("Hello, friends")
```

Run directly

```
$ scala myfile.scala
Hello, friends
$
```

Compile first

```
$ scalac -Xscript toto myfile.scala
$ scala toto
Hello, friends
$
```

myscript

```
#!/usr/bin/scala
!#
println("Hello, friends")
```

Turn it into a script

```
$ chmod +x myscript
$ ./myscript
Hello, friends
$
```

Run interactively (REPL)

```
$ scala
Welcome to Scala [...]

scala> println("Hello, friends")
Hello, friends

scala> :load myfile.scala
Loading toto.scala...
Hello, friends
```

The REPL is your friend to play with the concepts

Getting Started in Scala

Declaring a variable: `var x:Int = 0`

`var` $\rightsquigarrow$ because that's a **var**iable
`x` $\rightsquigarrow$ name of that variable (its label)
`:Int` $\rightsquigarrow$ type of this variable (what it can store)
`= 0` $\rightsquigarrow$ initial value (mandatory)

- You can often omit the type (it's inferred): `var x = 0`

Some Scala data types

- Int: for integer values, Double: for dot numbers
- Boolean: `true/false`, String: `"some chars together"`

Getting Started in Scala

Declaring a variable: `var x:Int = 0`

`var` ⇝ because that's a **var**iable
`x` ⇝ name of that variable (its label)
`:Int` ⇝ type of this variable (what it can store)
`= 0` ⇝ initial value (mandatory)

- You can often omit the type (it's inferred): `var x = 0`

Some Scala data types

- Int: for integer values, Double: for dot numbers
- Boolean: `true/false`, String: `"some chars together"`

Declaring a value

- If your "variable" is constant, make it a value: `val answer:Int = 42`
- Seen as good style in Scala *mutable stateful objects are the new spaghetti code*
- Allows to detect errors, may produce faster code, easy multithreading.
- Use values unless you must use variables

The Scala Syntax

Looping

```
while (condition) {
  instructions
}
```

```
do {
  instructions
} while (condition)
```

```
for (i <- 0 to 10 by 2) {
  // i in 0,2,4,6,8,10
}
```

Methods and functions

```
def sayIt(msg:String):Unit= {
  print(msg)
}
```

```
def max3(x:Int, y:Int, z:Int):Int = {
  val m = if (x>y) x else y
  if (m>z) {
    m          // explicit return not need
  } else {     // and sometimes considered
    z          // harmfull (but YMMV)
  }
}
```

Nice Aspects

- Semicolons (;) are optional
- Much of C scoria removed

Pitfalls

- The interpreter may start too early
- Shorter is not always clearer

Pattern matching: cascading if / else if are over

```
name match {
  case "Martin" => println("Hey there")
  case "Luc" => println("Hello")
  case _ => println("Gnii?")
}
```

- Veeery powerful construct
- Any expression can be filtered
- The default case is mandatory

```
name match {
  case "Martin" | "Luc" => println("Hey there")
  case _ => println("Gniii?")
}
```

```
age match {
  case i if i<20 => println("Hey dude!")
  case i if i<30 => println("Hello young man")
  case _ => println("Hello Sir")
}
```

```
(x,y) match {
  case (0,0) => println("Origin")
  case (_,0) => println("Abscissa")
  case (0,_) => println("Ordinate")
  case (_,_) => println("Random")
}
```

Lists: central data container in functional world

Scala lists are homogeneous (all elements of same type)

```
scala> val l0 = Nil // the empty list
res1: scala.collection immutable.Nil.type = List ()

scala> val l = 1 :: 2 :: Nil   // :: is pronounced " cons "
l: List[Int] = List (1 , 2)

scala> val m = List (3 , 4 , 5)
m: List[Int] = List (3 , 4 , 5)

scala> l ::: m
res2: List[Int] = List (1 , 2 , 3 , 4 , 5)
```

Scala lists are immutable (cannot change value)

```
scala> val l = List (1 ,2 ,3 ,4)
l: List [Int] = List (1 , 2 , 3 , 4)

scala> l (3)
res1: Int = 4

scala> l (3) = 100
<console>:9: error: value update is not a member of List[Int]
```

Arrays: central data container in imperative world

Scala arrays are homogeneous, but mutable

```
scala> val a = Array (1 ,2 ,3 ,4)
a: Array[Int] = Array (1 , 2 , 3 , 4)

scala> a(3) = 100

scala> a
res1: Array [ Int ] = Array (1 , 2 , 3 , 100)
```

- Scala defines immutable and mutable versions of many data containers
- Always prefer the immutable version

The content of an immutable container may be mutable

```
scala> val a = List(Array(1,2,3), Array(4,5), Array(6))
a: List[Array[Int]] = List(Array(1, 2, 3), Array(4, 5), Array(6))

scala> a(0)(0) = 1000

scala> a
res2: List[Array[Int]] = List(Array(1000, 2, 3), Array(4, 5), Array(6))

scala> a(0) = Array(1,2,3)
<console>:12: error: value update is not a member of List[Array[Int]]
```

Advanced for loops

```
scala> 1 to 10 // or 1.to(10)
res1: scala.collection.immutable.Range.Inclusive =
   Range (1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10)

scala> 10 to (0 , -2)
res2: scala.collection.immutable.Range.Inclusive =
   Range (10 , 8 , 6 , 4 , 2 , 0)

scala> for ( x <- List (1 ,2 ,3)) yield x *2
res3: List[Int] = List(2 , 4 , 6)

scala> for { x <- 1 to 5  // generator
             y = x % 2;   // definition
             if ( y == 0) // filter
           } yield {
             println ( x )
             x
           }
2
4
res4:scala.collection.immutable.IndexedSeq[Int] = Vector (2 , 4 )

scala> for { x <- List (1 ,2 ,3); y <- List (4 ,5)) } yield x * y
res5: List[Int] = List (4 , 5 , 8 , 10 , 12 , 15)
```

More on Scala usage

Compile once, use many times

Hello.scala

```
object Hello {
  def main(args: Array[String]) =
      println("Hello, friends")
}
```

```
$ scalac Hello.scala
$ ls
Hello$.class    Hello.class   Hello.scala
$ scala Hello # no recompilation
Hello, friends
$
```

Interfacing with Java libraries

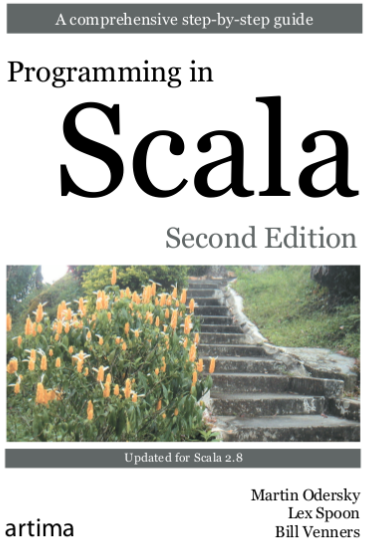
```
import java.awt._
import scala.swing._
import scala.swing.event._

object MyGUI extends SimpleSwingApplication {
  lazy val ui: scala.swing.Panel = new Panel {
    background = Color.white
    preferredSize = new Dimension(800, 600)
    override def paintComponent(g: Graphics2D)={
      super.paintComponent(g)
      g.drawString("Hello, world.",
                   10, size.height - 10)
   }
 }
```

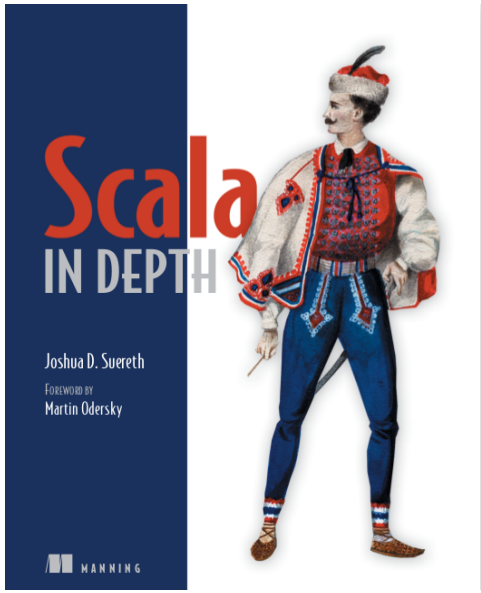
(continued)

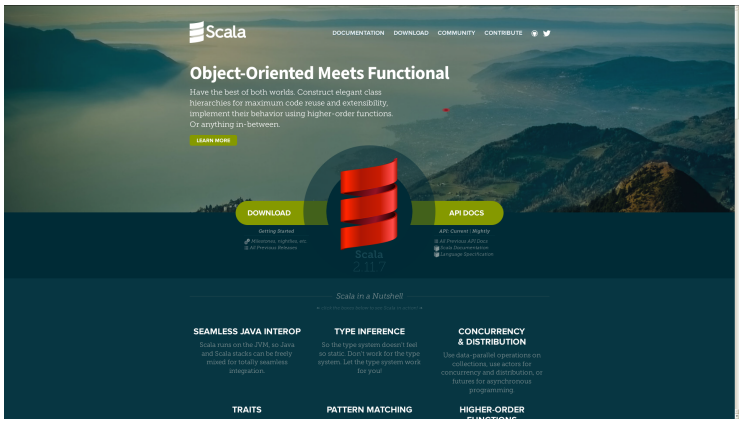
```
 def top = new MainFrame {
  title = "My little GUI"
   contents = ui
 }
}
```

```
$ scalac -cp scala-swing.jar:. MyGUI.scala
$ scala -cp scala-swing.jar:. MyGUI
(opens a new windows)
$
```

More Information on Scala







- http://scala-lang.org
- https://twitter.github.io/scala_school
- http://www.cs.columbia.edu/~bauer/cs3101-2
- https://www.irisa.fr/celtique/genet/GEN
- http://www.loria.fr/~oster/pmwiki/index.php/Main/OOP

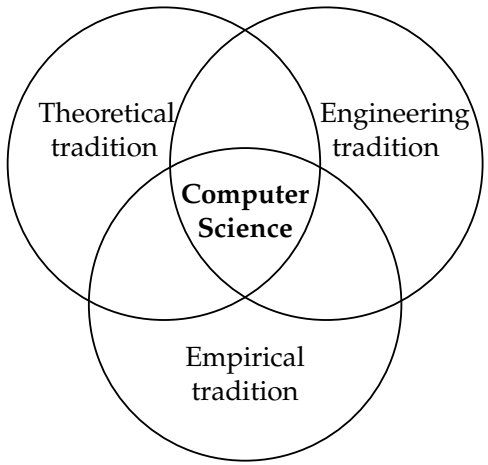
Conclusion

Computer Science and Informatics

- **Science of Abstraction:** building hierarchies of symbols and concepts
  Programming computers: surface activity, but the easiest to practice with
- Computational Sciences: simulation as third pilar (+ observation & theory)

The Historical Heritages of Computer Science

- Maths: proves necessary facts
- Natural Sciences: tests contingent facts
- Engineering: solves problems



Programming Complex Systems is at the core of the discipline

(That's the topic of this course :)

Scala

- Nice little language, that turns out to be a multi-paradigm beauty (beast)
- That's not perfect either, but we will use it in this course