

An Open Framework for Extensible Multi-Stage Bioinformatics Software

Johan Nyström-Persson¹, Gabriel Keeble-Gagnère², Matthew I Bellgard², Kenji Mizuguchi¹

1. Nat. Inst. of Biomedical Innovation, Osaka, Japan

2. Centre for Comparative Genomics, Murdoch U, Perth, Australia

9 Nov 2012
PRIB, Tokyo, Japan

This talk

- New principles for bioinformatics software
- The Scala programming language
- The Friedrich framework

Bioinformatics software today

Bioinformatics software

- Software development for data-intensive experimental science is very difficult
- However, we have inadequate tools, frameworks, common practices and developer skills

Bioinformatics software development is special

- One of the main sources difficulty is **change**
 - Technology and data formats (for example, sequencing)
 - Scientific methods
 - Availability and nature of data sources (databases etc) being used
- But there's also **scale**
 - Enormous data sets, constantly growing larger
 - Need for parallelism

Additional problems

- R and Perl were good choices 10-15 years ago - not anymore!
 - But there's still a huge repository of bioconductor and CPAN packages - massive network effect, **hard to leave old technology**
- What about big data and concurrency?
 - There's a need for accessible solutions that anybody can program, making use of **all the idle CPU cores that are around us in our labs**
 - **Concurrency is very hard with Perl and C but much easier with modern languages**

New principles

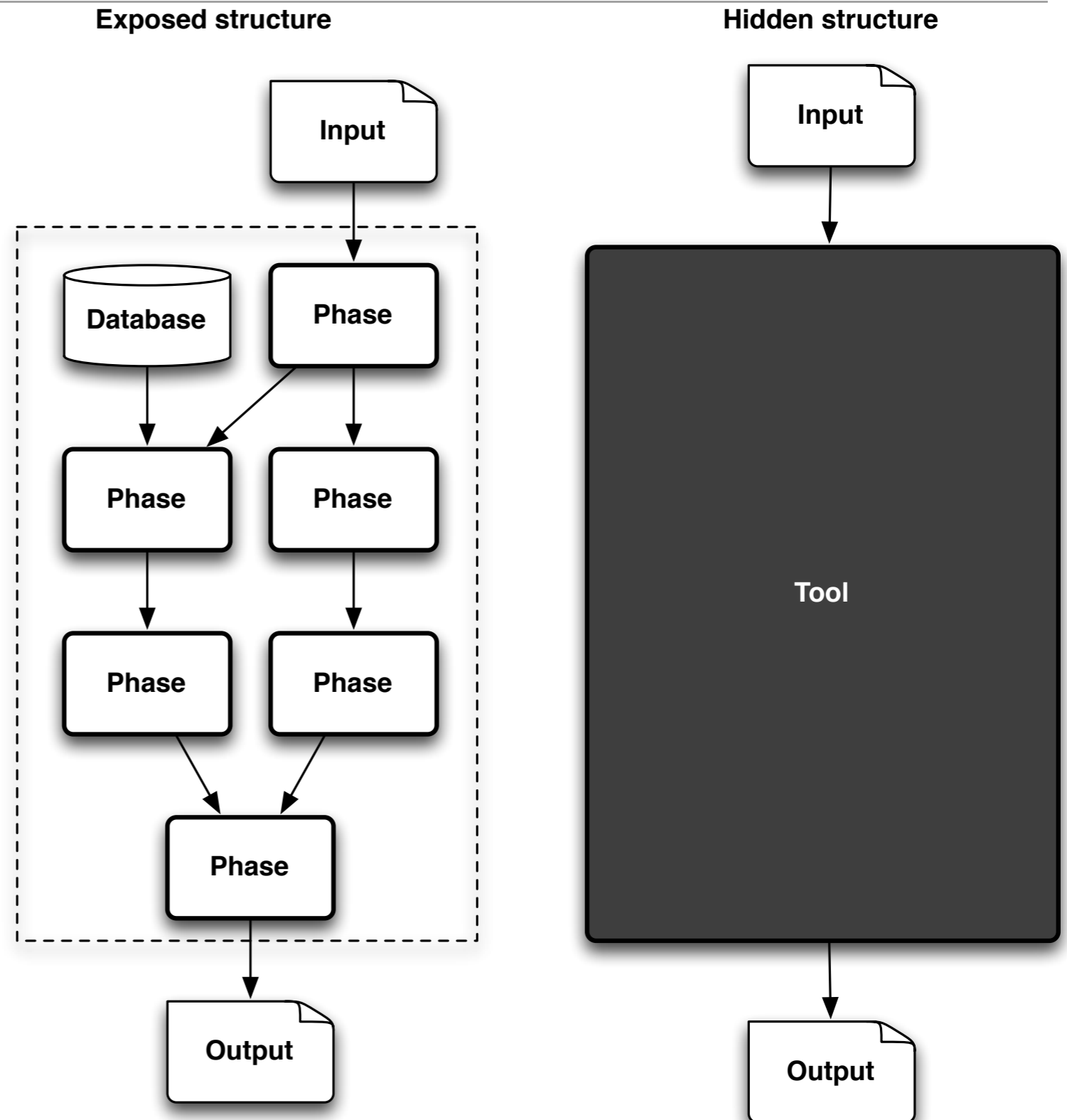
- In the following, we make a set of **observations** about the current situation and as a result, recommend **a new set of principles**
 - The principles were arrived at while developing a de novo genome assembler

Monolithic software

- When we optimise for performance at any cost, software can become difficult to change - “*monolithic*”
- Monolithic software resembles a black box that cannot be changed or is very difficult to change
- Encapsulation may not be well suited to bioinformatics
- Principle: expose internal structure

Principle: Expose internal structure

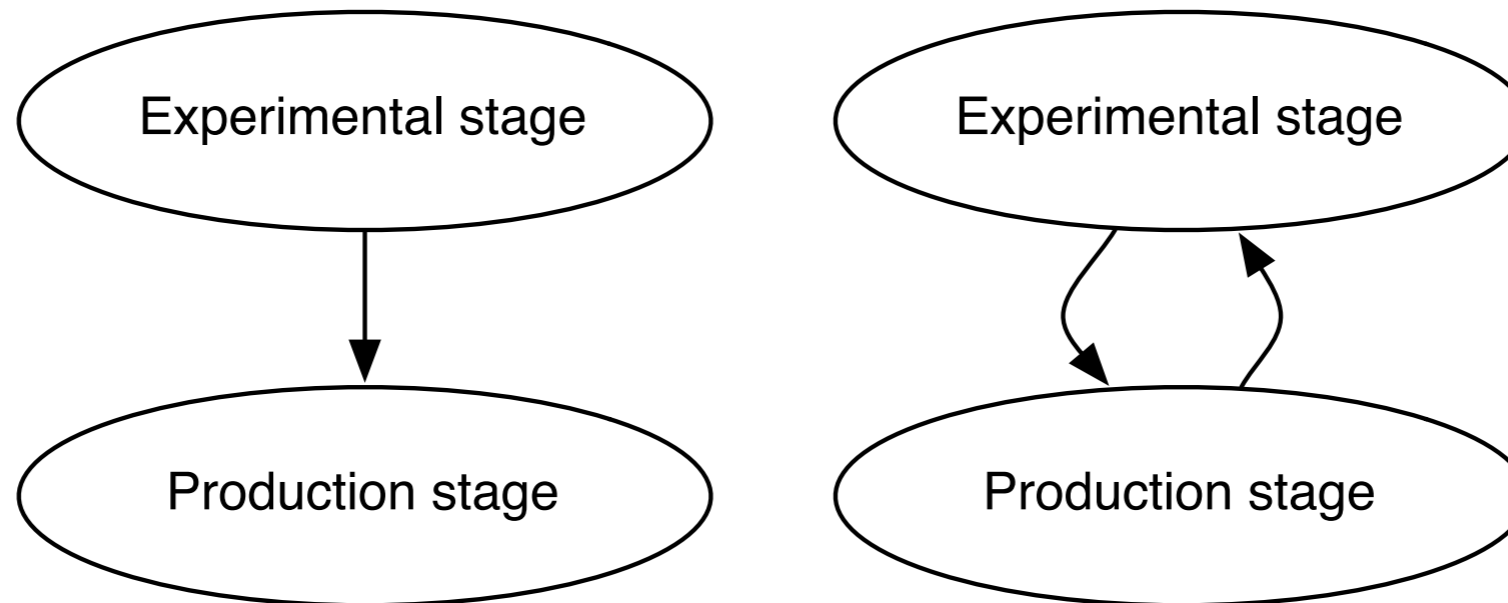
- Software tools should expose each internal processing step
- Allow for manipulation and inspection of each one independently
- Allow for reconfiguration of data flow



Multiple stages

- Conventional view: software is a product (scientific tool) that is delivered to a customer (a researcher)
 - First method development (experimentation), then method usage (production)
- Realistic view: scientific software is never finished, and needs to evolve along with the research
 - Moving from experimentation to production, and back again (multi-stage software)

Multi-stage software



- Avoid designing for either the experimental stage (flexible, slow) or for the production stage (inflexible, fast).
- Assume that we have to move back and forth continuously between **applying** the method and **modifying** the method!

Finality

- Software developers often make various final assumptions about input formats, users' intentions, and so on, and hardcode them throughout the software
- Avoid a final design or final assumptions about how the software is to be used ultimately: minimal finality

Loss of intermediate data

- Many tools naturally perform computation in multiple stages
- However, after each stage has been computed, **data is discarded and it is impossible to go back to the previous stage**
- **Principle: retain intermediate data maximally, so that it is efficient to go back and explore previous stages with other parameters**
 - Mathematically: preserve dimensionality maximally

Preserving intermediate data

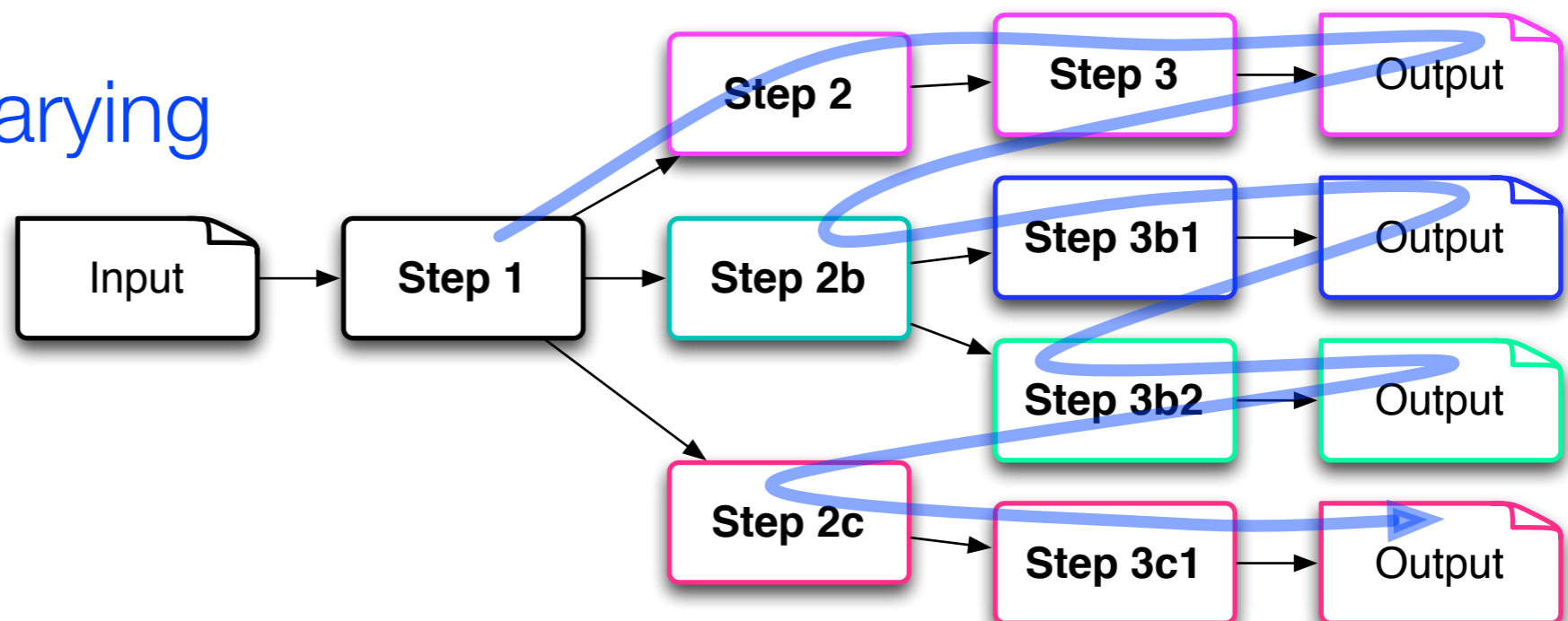
Traditional: start to finish



With preservation



Enables efficient exploration with varying parameters



Performance bias OR flexibility bias

- Traditionally, bioinformatics programmers tend to write **either very flexible software (experimental stage) or very fast software (production stage)**
- Instead of optimising **$\max(f, p)$** , optimise **$f \cdot p$** !
 - Aim for acceptable performance and acceptable flexibility

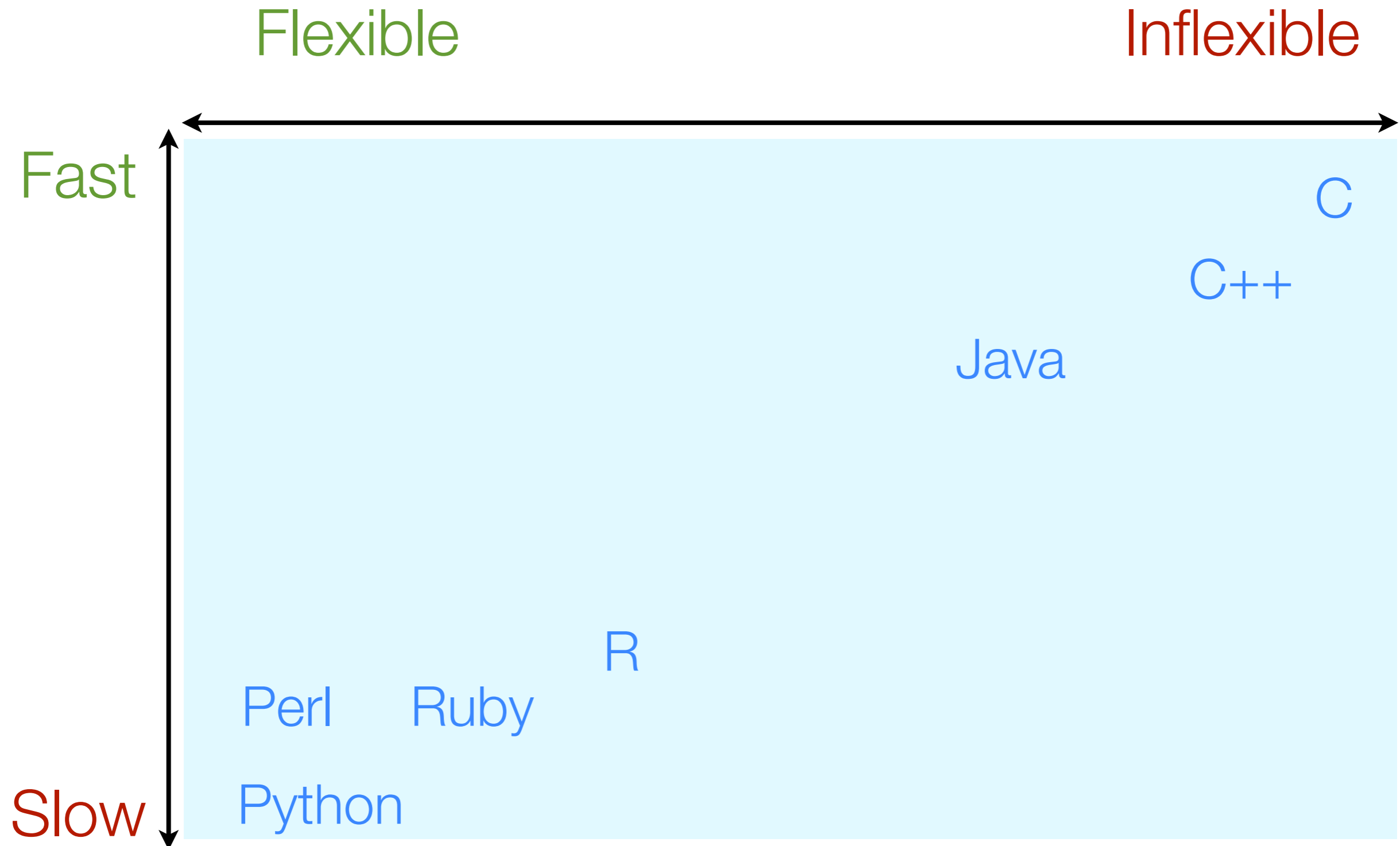
Context	Performance	Flexibility	Monolithic	Examples
Experimentation	Low	Very high	No	BioPerl, BioPython
Production	Very high	Low	Possibly	Velvet, Abyss, BioJava
Proposed ideal	High	High	No	

Principles for bioinformatics software: summary

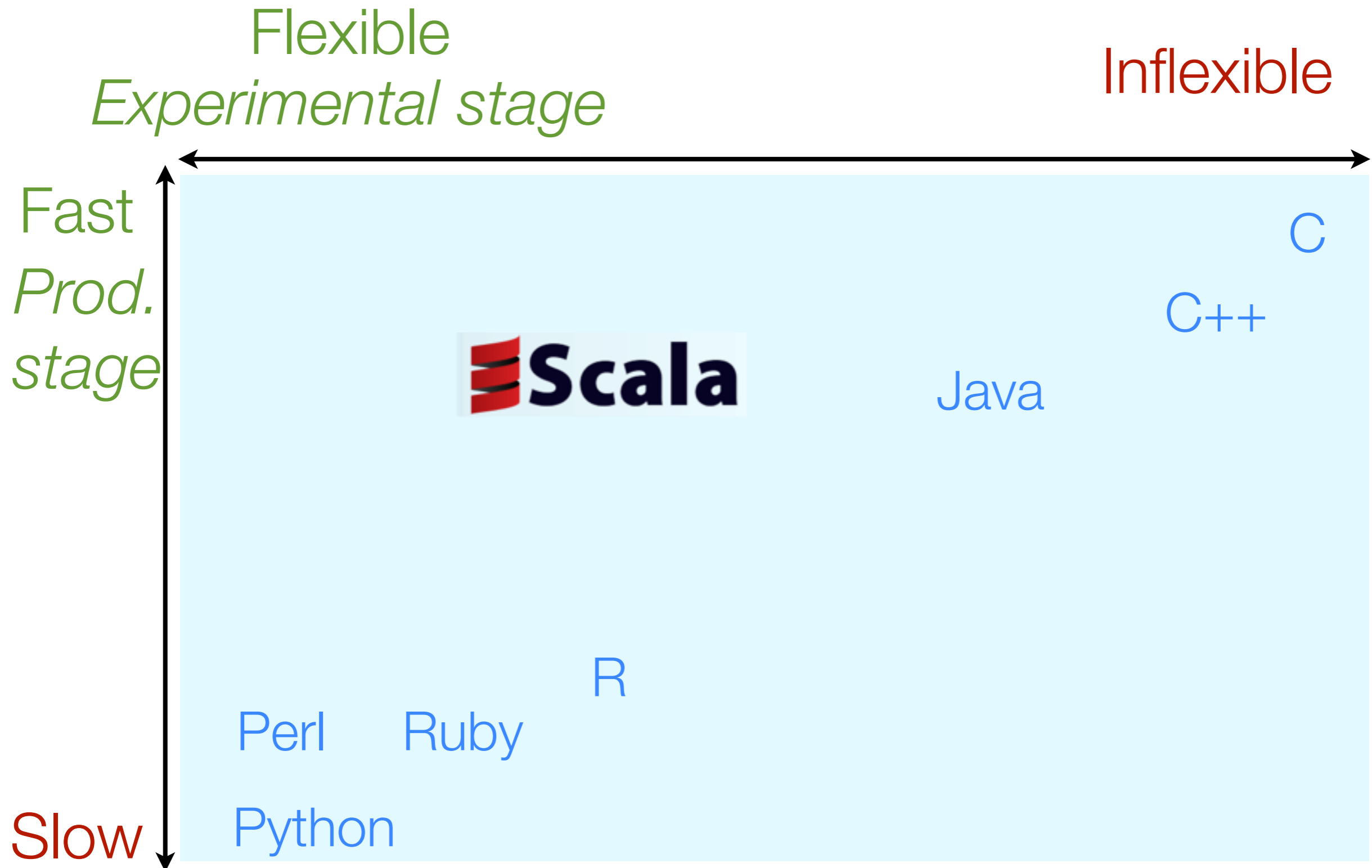
- Expose internal structure
- Multi-stage applications
- Conserve dimensionality maximally
(retain intermediate data)
- Minimal finality
- Flexibility with performance
- (Ease of use)

The Scala programming language

Programming languages



A new kind of compromise?



Scala



- Hybrid functional/object-oriented language for the Java VM
- In development for the last 10 years or so, now becoming popular and widely used in academia, industry
- Very easy to combine with Java code and libraries
- Based on **modern computer science**
 - Simple, efficient concurrency and parallelism (Erlang-style model with actors)
- www.scala-lang.org

Scala (2)



- Industrial users include The Guardian, LinkedIn, Xerox, Twitter, Siemens, ...
- Code is very compact, flexible style, often 3x smaller than Java
- Interactive console (like python, perl etc) (“REPL”)
- The performance of Java and the compactness of Python/Perl!

Scala in bioinformatics



- **BioScala** - 2 year old project, now dormant, by Pjotr Prins (<https://github.com/bioscala/bioscala>)
- **(BioJava)** - substantial, can be used very easily from Scala (<http://biojava.org>)
- **Scabio** - small bioinfo algorithms collection, by Markus Gumbel (<http://www.mi.hs-mannheim.de/gumbel/en/forschung/scabio>)
- **Friedrich** - our contribution in this talk
- Also: use any maths/visualisation/DB etc library you want from Java

Bioinformatics needs Scala



- Several small libraries and projects exist, but none has become very big yet - hopefully this will change soon
- **Scala may be the best language to support the experimental stage and the production stage simultaneously:** good fit for scientific research
- *Why everyone in bioinformatics should learn Scala* by Pjotr Prins (<http://blog.thebird.nl/?p=26>)

Friedrich: A software framework

Friedrich

- A framework that supports bioinformatics software development in Scala according to the principles we have outlined (“The Friedrich principles”)

Friedrich features (current)

- Phases

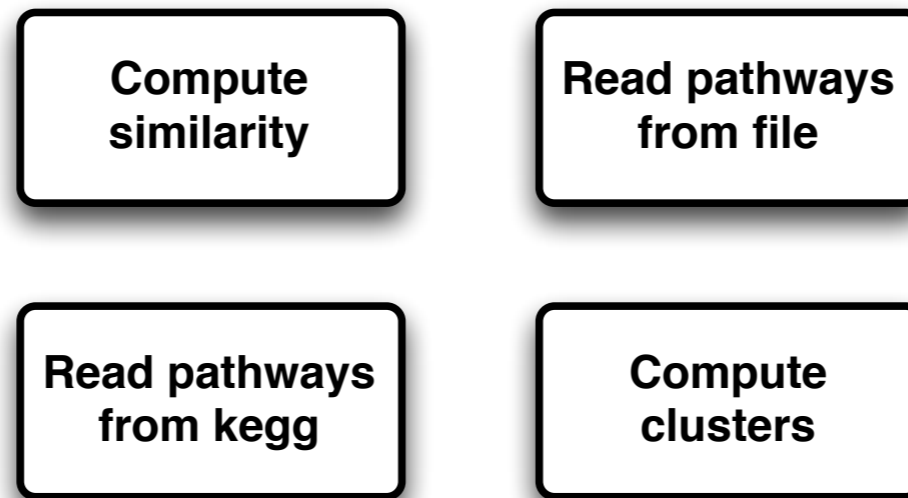
- An “application” is structured as a set of phases that operate on data
- Just as in Scala, there is support for both **mutable** and **immutable** data

- Pipelines

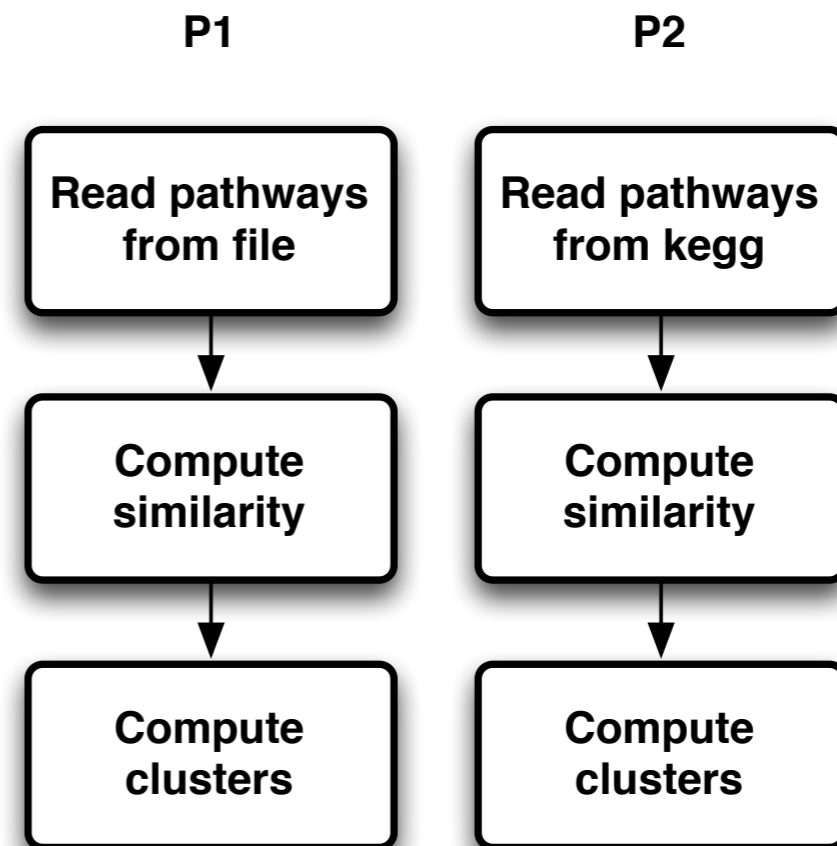
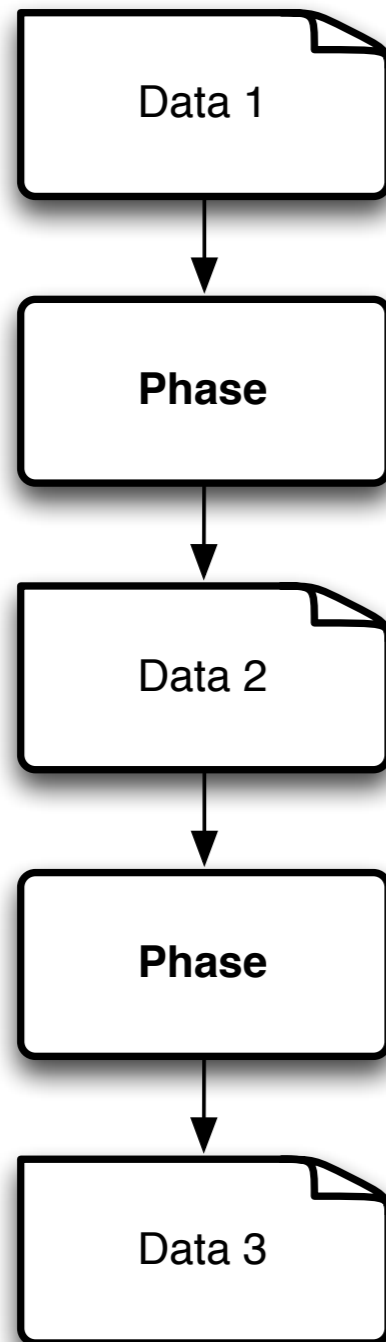
- A pipeline is a sequence of phases that can process data

- Configuration management

Phases



Pipelines



Supporting the principles

- Because phases can be rearranged, **minimal finality** is supported, and **internal structure is exposed**
- Immutable data supports **conservation of intermediate results** (work in progress)
- Scala supports **flexibility with performance**

Friedrich applications

- A simple but functional de novo genome assembler is available
 - Can handle approx. 20 million *k*-mers using 2 GB RAM
- Internal applications at NIBIO, not ready for release
- Feel free to develop your own!
- Get Friedrich: <http://bitbucket.org/jtnystrom/friedrich>

Comparison with...

- Task running frameworks, for example Galaxy and Yabi are focussed on providing a high level interface to existing components
 - User-developer separation
- “Toolkits” e.g. SAMTools, Picard: multiple binaries
 - Not flexible enough, no true interactive experimentation

Ongoing and future work



- Currently in progress
 - Akka, a framework for distributed/concurrent *actor-based* programming
 - Model is similar to Erlang
 - Goal: transparent, location-agnostic concurrency
 - Integration with BioScala/BioJava
- Future goals
 - Downloadable package/module system

Summary

- Bioinformatics software development faces unique challenges, and collectively we are not up to scratch
- New principles show the way to more natural and efficient software development in BI
- Scala may be the language we need next; Perl, R and C++ are not good enough
- Friedrich is a framework that supports our new principles, and you are welcome to use and contribute to it

*“Everyone who has ever built anywhere a new heaven
first found the power thereto in his own hell.” - F. W. Nietzsche*

Thank you !

We would very much appreciate comments and collaboration.
Please contact us: johan@nibio.jp, jtnystrom@gmail.com (me)
gabriel7@gmail.com (Gabriel)