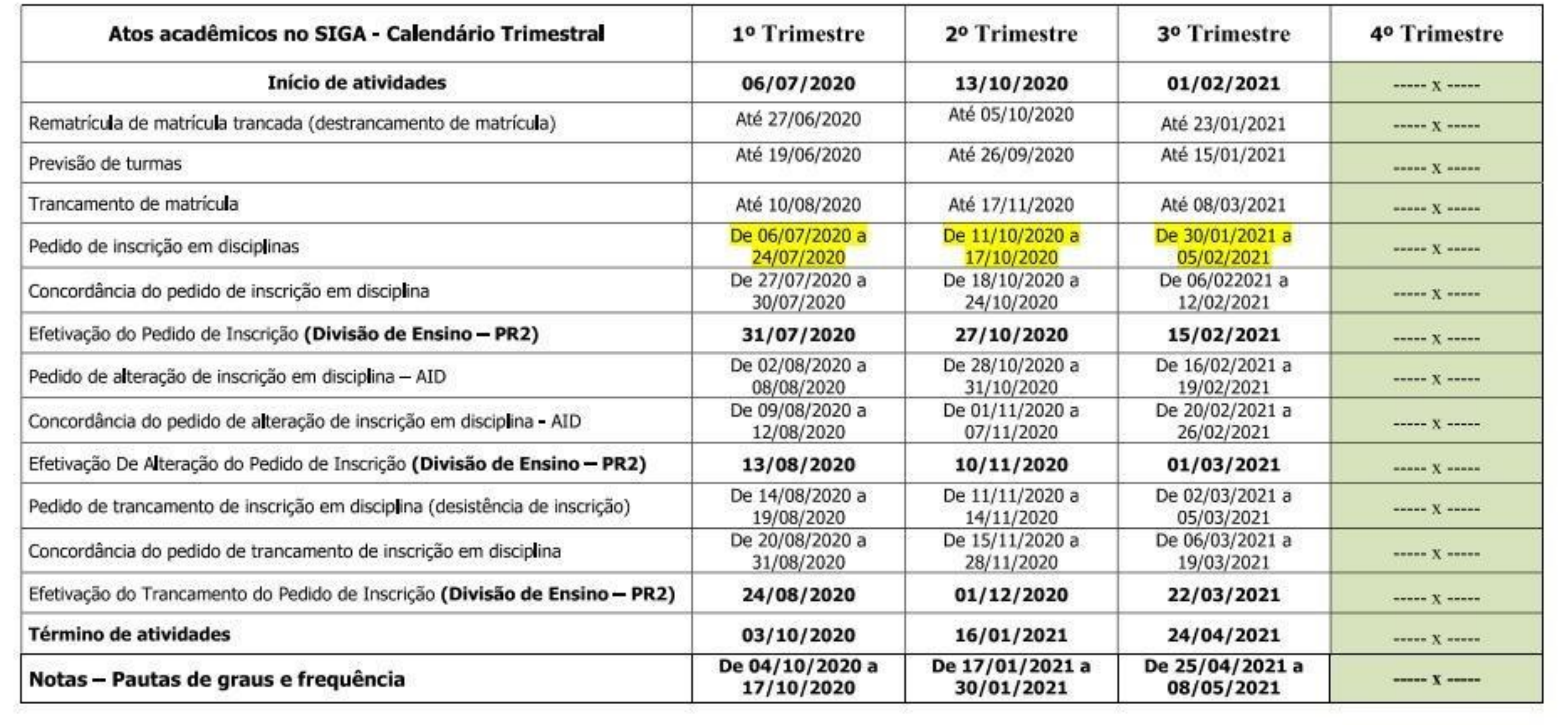
Datas Importantes



Atos acadêmicos no SIGA - Calendário Trimestral	**1º Trimestre**	**2º Trimestre**	**3º Trimestre**	**4º Trimestre**
Início de atividades	**06/07/2020**	**13/10/2020**	**01/02/2021**	----- x -----
Rematrícula de matrícula trancada (destrancamento de matrícula)	Até 27/06/2020	Até 05/10/2020	Até 23/01/2021	----- x -----
Previsão de turmas	Até 19/06/2020	Até 26/09/2020	Até 15/01/2021	----- x -----
Trancamento de matrícula	Até 10/08/2020	Até 17/11/2020	Até 08/03/2021	----- x -----
Pedido de inscrição em disciplinas	De 06/07/2020 a 24/07/2020	De 11/10/2020 a 17/10/2020	De 30/01/2021 a 05/02/2021	----- x -----
Concordância do pedido de inscrição em disciplina	De 27/07/2020 a 30/07/2020	De 18/10/2020 a 24/10/2020	De 06/022021 a 12/02/2021	----- x -----
Efetivação do Pedido de Inscrição **(Divisão de Ensino – PR2)**	**31/07/2020**	**27/10/2020**	**15/02/2021**	----- x -----
Pedido de alteração de inscrição em disciplina – AID	De 02/08/2020 a 08/08/2020	De 28/10/2020 a 31/10/2020	De 16/02/2021 a 19/02/2021	----- x -----
Concordância do pedido de alteração de inscrição em disciplina - AID	De 09/08/2020 a 12/08/2020	De 01/11/2020 a 07/11/2020	De 20/02/2021 a 26/02/2021	----- x -----
Efetivação De Alteração do Pedido de Inscrição **(Divisão de Ensino – PR2)**	**13/08/2020**	**10/11/2020**	**01/03/2021**	----- x -----
Pedido de trancamento de inscrição em disciplina (desistência de inscrição)	De 14/08/2020 a 19/08/2020	De 11/11/2020 a 14/11/2020	De 02/03/2021 a 05/03/2021	----- x -----
Concordância do pedido de trancamento de inscrição em disciplina	De 20/08/2020 a 31/08/2020	De 15/11/2020 a 28/11/2020	De 06/03/2021 a 19/03/2021	----- x -----
Efetivação do Trancamento do Pedido de Inscrição **(Divisão de Ensino – PR2)**	**24/08/2020**	**01/12/2020**	**22/03/2021**	----- x -----
Término de atividades	**03/10/2020**	**16/01/2021**	**24/04/2021**	----- x -----
Notas – Pautas de graus e frequência	**De 04/10/2020 a 17/10/2020**	**De 17/01/2021 a 30/01/2021**	**De 25/04/2021 a 08/05/2021**	**----- x -----**

Avaliação e Atendimento

Critérios de aprovação são os do PPGI/UFRJ.

A avaliação da disciplina consiste em participação em sala de aula (P); exercícios e/ou protótipos desenvolvidos (E); apresentações/ /escritas de artigos (A).

$$MF = 0.2 * P + 0.2 * E + 0.6 * A$$

O aluno que desejar atendimento deverá requisitar o mesmo por e-mail e um horário será agendado pelos responsáveis para o atendimento.

serra@pet-si.ufrrj.br

zavaleta@pet-si.ufrrj.br

Entregáveis – Março

4/3 - Aula + Definição dos grupos + PIT 5 min (Apresentação geral em PPT - DataSet + Problema + Objetivo)

11/3 - Aula + Descrição do projeto de DS - Tudo é via GIT!

- Entregar o projeto contendo (V1): Detalhamento e descrição textual da definição do problema a ser explorado pela equipe; descrever tecnicamente o dataset e sua fonte, o que pretendem fazer e o que vão e como extrair (plano dos experimentos). Apresentar os objetivos geral e específico do projeto de DS; apresentar métodos de data cleaning/tratamento de dados que serão usados, apresentar a proposta de modelo de extração de conhecimento e visualização de dados que será adotado.

18/3 - Aula + Refinamento do projeto de DS

- Agregar ao projeto (V2) : Plano do experimento a ser executado (scripts no Colab x Jupyter x IDE), projeto de coleta de metadados da proveniência dos experimentos, projeto para tornar seu experimento reprodutível, adicionar qualquer outro melhoria ou diferencial que julguem necessário

25/3 – Aula + entregar da 1a versão do artigo – Recomenda-se usar o template da LNCS e suas regras de formatação.

- Texto final (15-20) páginas com referências (pode ser em português ou inglês, escolha do grupo)

Entregáveis – Abril

1/4 - Aula

8/4 - Aula - + Sorteio ordem de apresentação dos grupos

15/4 - Entregar da 2a versão do artigo + Apresentação trabalho alunos (1ª. Parte dos grupos)

22/4 - Apresentação - trabalho alunos (2ª. Parte dos grupos)

10/5 - Entrega da versão final do artigo + projeto completo (V3) + scripts com provenance + datasets anotados+ resultados de DS+ Depósito do notebook reprodutível e executáveis no GIT

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

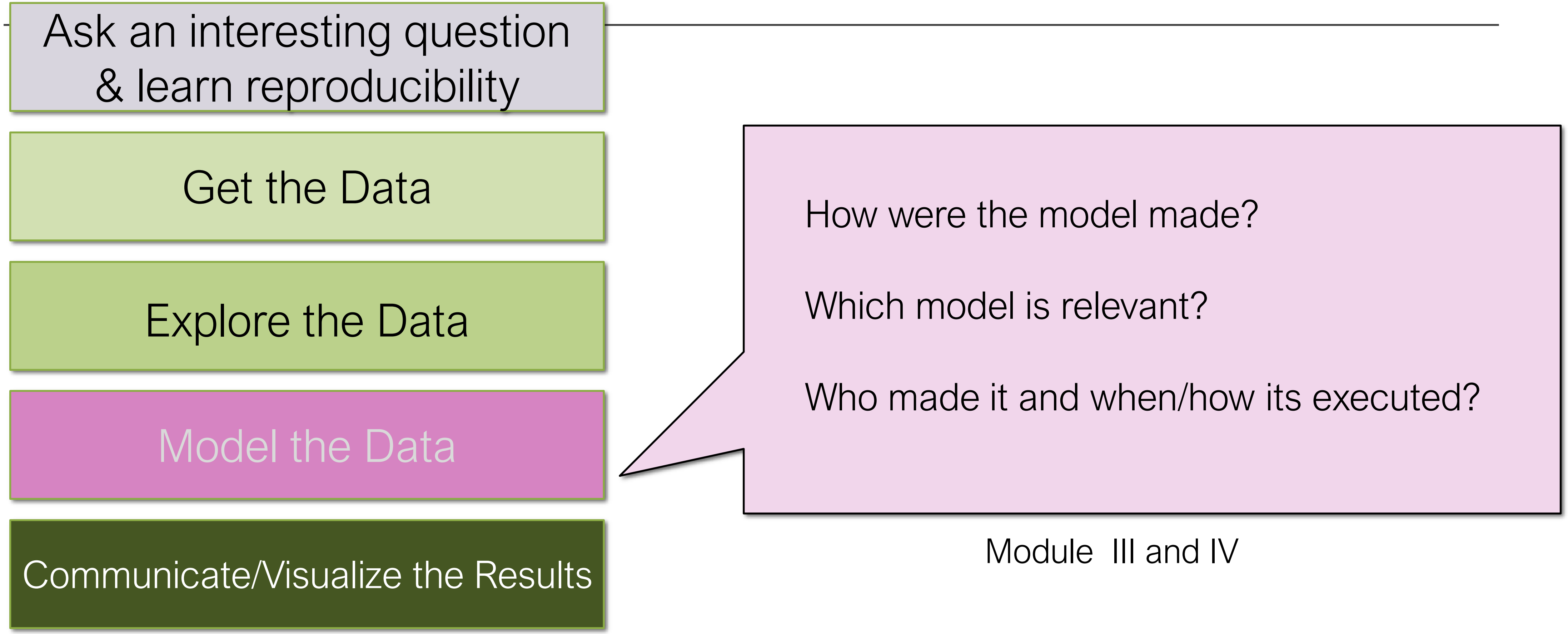
Introduction to Data Science

MODULE IV – PART I

Scientific in Silico Experiments as Workflows and Scripts

Prof Sergio Serra e Jorge Zavaleta

What? The Modeling Process



Ask an interesting question & learn reproducibility

Get the Data

Explore the Data

Model the Data

Communicate/Visualize the Results

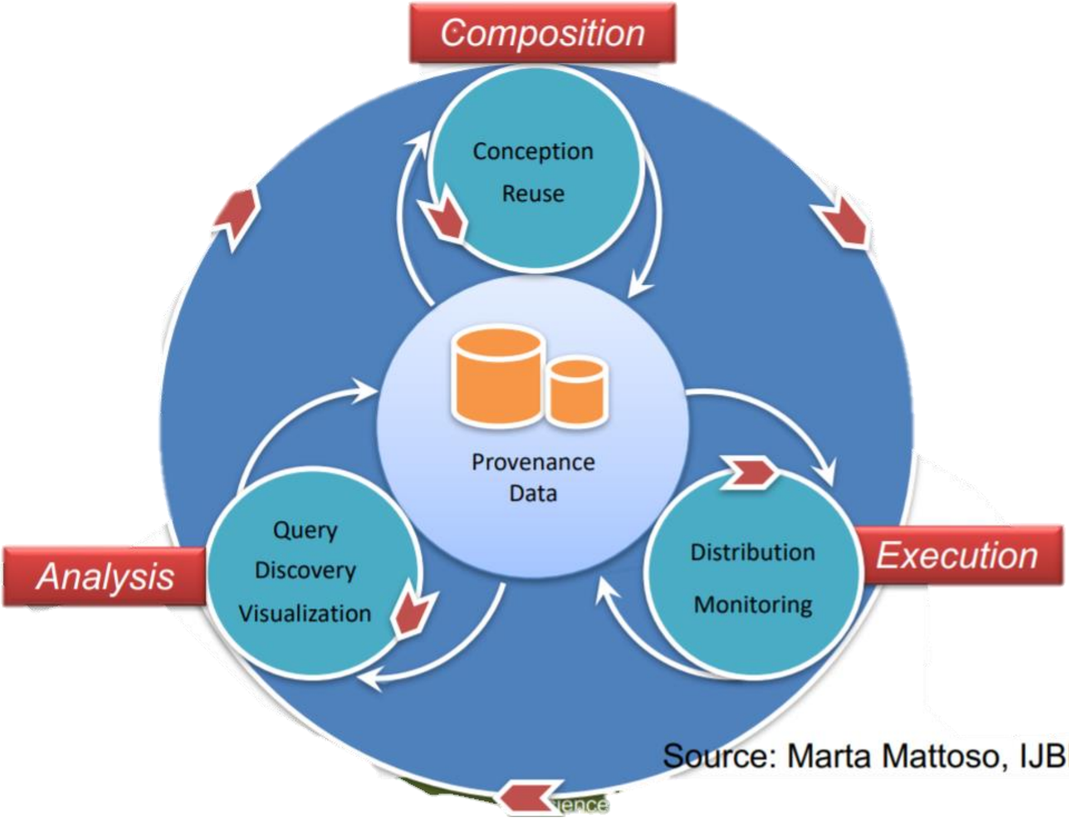
How were the model made?

Which model is relevant?

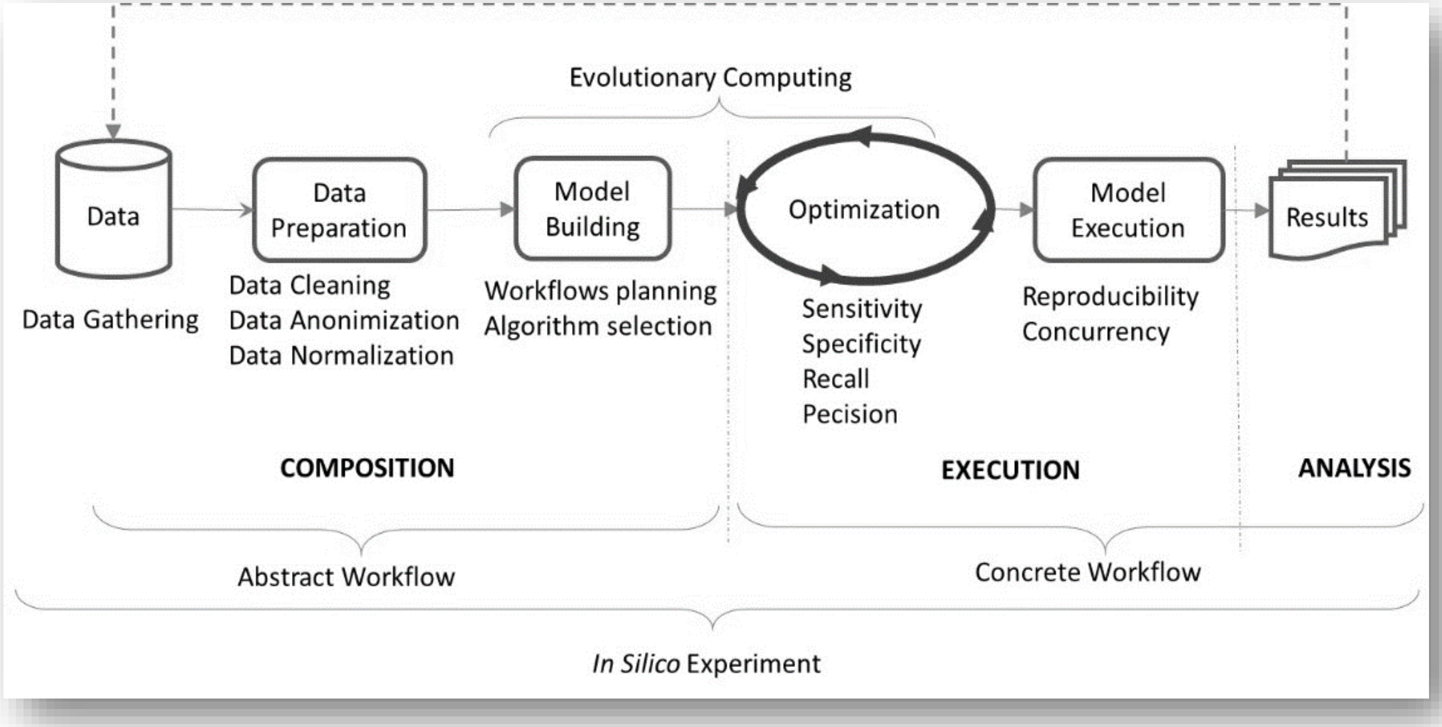
Who made it and when/how its executed?

Module III and IV

The *in silico* experiment lifecycle



Source: Marta Mattoso, IJBPIM, 2010



Evolutionary Scientific Workflows
Sergio Manuel Serra da Cruz; Anderson de Oliveira; Fabricio Firmino de Faria
Congress on Evolutionary Computation, 2018

0International Journal of Business Process Integration and Management 5(1):79-92
DOI: 10.1504/IJBPIM.2010.033176

Scientific Workflow

The scientific workflow represents the automation of a scientific hypothesis, in which tasks (chain of activities) compute the data resources according to a set of user-defined rules, producing the scientific results (data products)

A scientific workflow is a chain of activities → a.k.a **data flow**

- In a data flow, the execution is guided by the data

Scientific workflows are supported by Scientific Workflow Management Systems (SWfMS),

- exhibit several functionalities, namely: composition and execution of the workflows,
- distribution of its execution in parallel processing environments,
- collection of data provenance,

Scientific Workflow

- Abstract
- Concrete

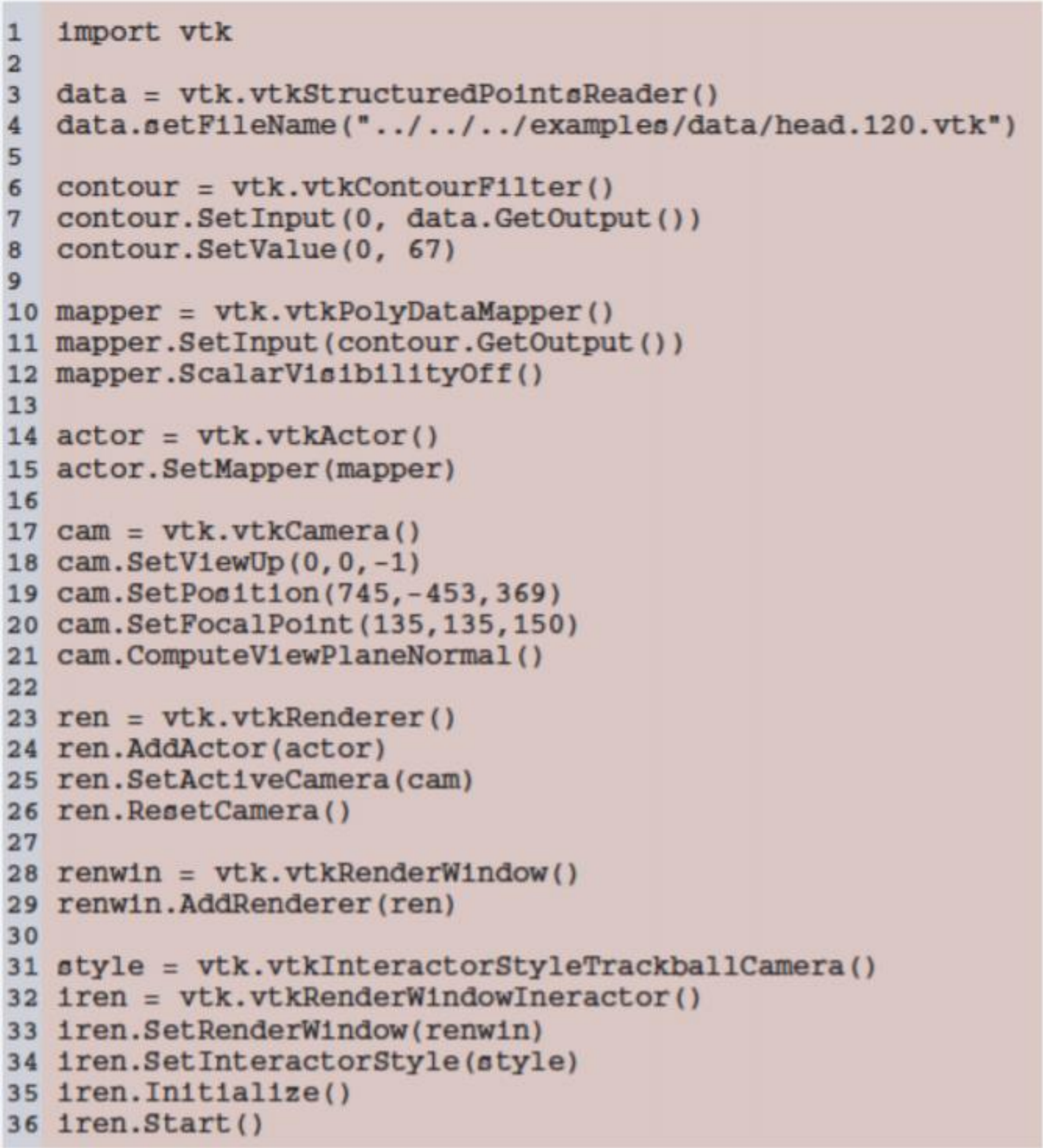
Abstract x Concrete

Abstract workflow simplifies the research process;

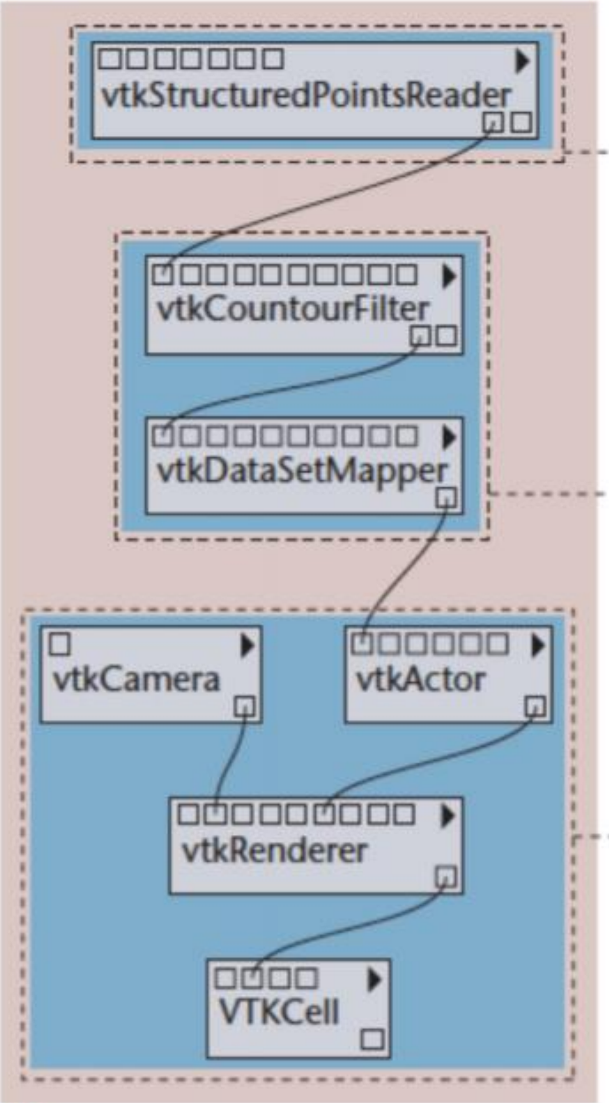
- Enables the research team to foresee a systematic approach of the *in silico* experiment, providing flexibility to replace components with alternative implementations
- The abstract workflow is later mapped into a concrete workflow or script

Concrete workflows describe the sequences of modules (tasks) to be executed and details the computational resources that would be used.

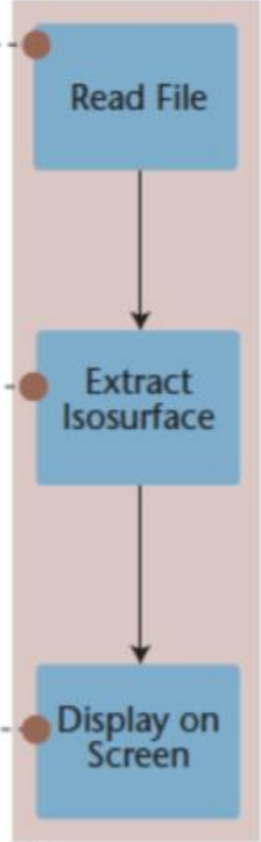
- tasks may vary from data cleaning to building, executing, evaluating and training the computing models



```
import vtk

data = vtk.vtkStructuredPointsReader()
data.setFileName("../../../examples/data/head.120.vtk")

contour = vtk.vtkContourFilter()
contour.SetInput(0, data.GetOutput())
contour.SetValue(0, 67)

mapper = vtk.vtkPolyDataMapper()
mapper.SetInput(contour.GetOutput())
mapper.ScalarVisibilityOff()

actor = vtk.vtkActor()
actor.SetMapper(mapper)

cam = vtk.vtkCamera()
cam.SetViewUp(0,0,-1)
cam.SetPosition(745,-453,369)
cam.SetFocalPoint(135,135,150)
cam.ComputeViewPlaneNormal()

ren = vtk.vtkRenderer()
ren.AddActor(actor)
ren.SetActiveCamera(cam)
ren.ResetCamera()

renwin = vtk.vtkRenderWindow()
renwin.AddRenderer(ren)

style = vtk.vtkInteractorStyleTrackballCamera()
iren = vtk.vtkRenderWindowIneractor()
iren.SetRenderWindow(renwin)
iren.SetInteractorStyle(style)
iren.Initialize()
iren.Start()
```



(b)



(c)

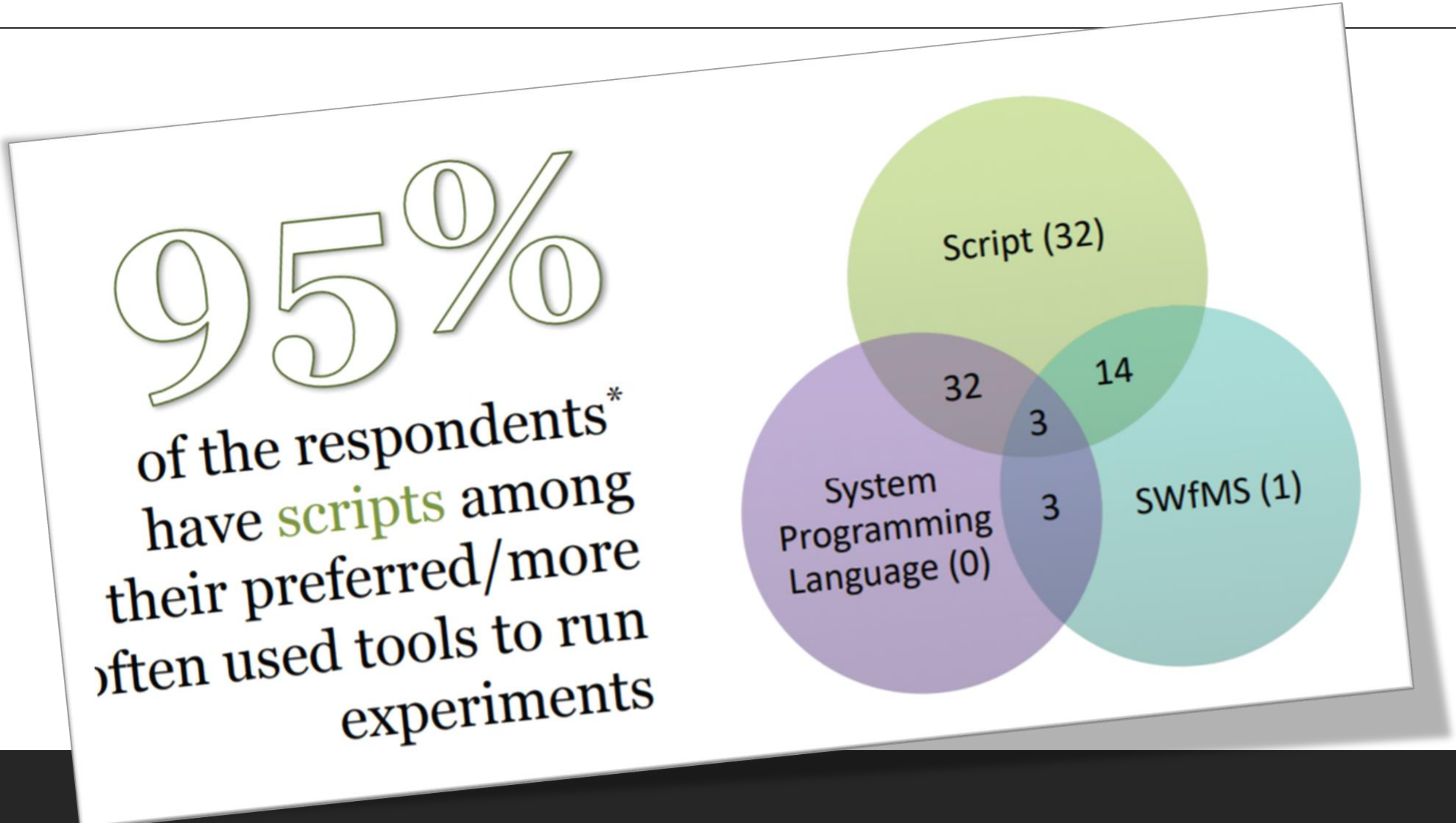
One Abstract to (possibly) Several Concretes !

Source: Freire et al., 2008. Provenance for Computational Tasks: A Survey.

How to cross the bridge from "abstract heaven" to "concrete land"

- Several ways to go from abstract to concrete
- When using scripts, there are several ways to go from abstract to concrete workflows
  - Activities are implemented one after the other in the script (no functions)
  - Activities are mapped into functions (each activity becomes one or more functions)
- Black Box X White Box (next slides...)

But….lots of researchers still use **scripts**!



95%

of the respondents* have scripts among their preferred/more often used tools to run experiments

Scripts x Workflows

There is no robust definition in the literature

Execution follows a **control flow** instead of a data flow

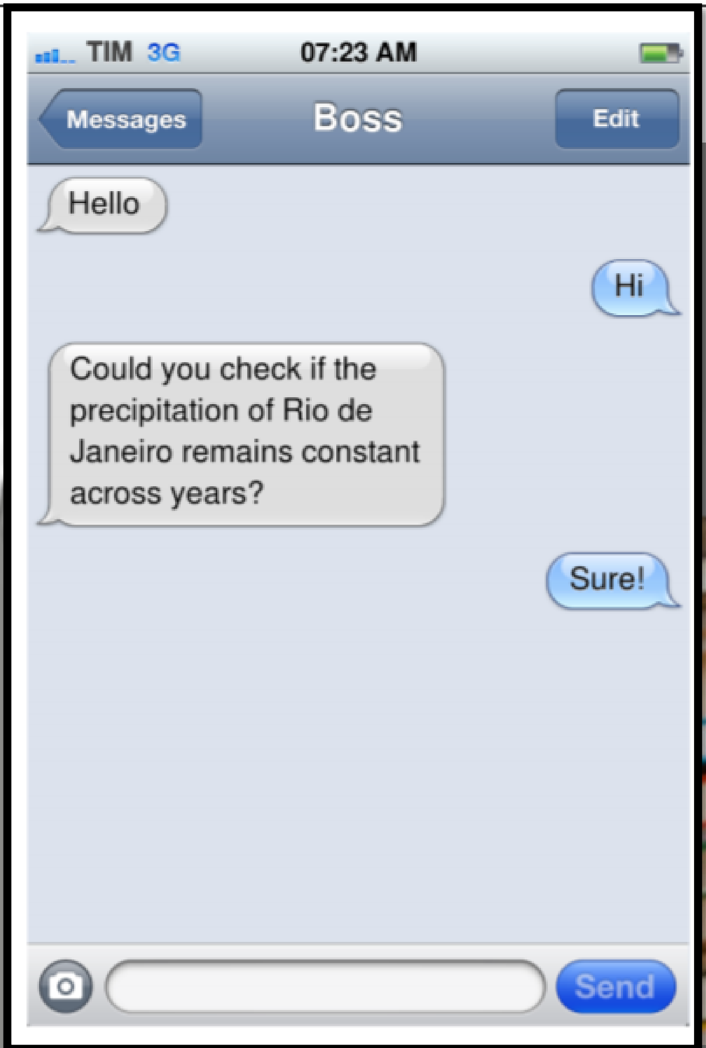
- Commands explicitly define the execution order

Characteristics

- Everything is Object
- Multiparadigm
- Typeless (dynamically typed)
- Interpreted
- Automatic memory management
- Extensive component library

Scripts are interactive!
(see our DS classes!)

Running a Computational Experiment

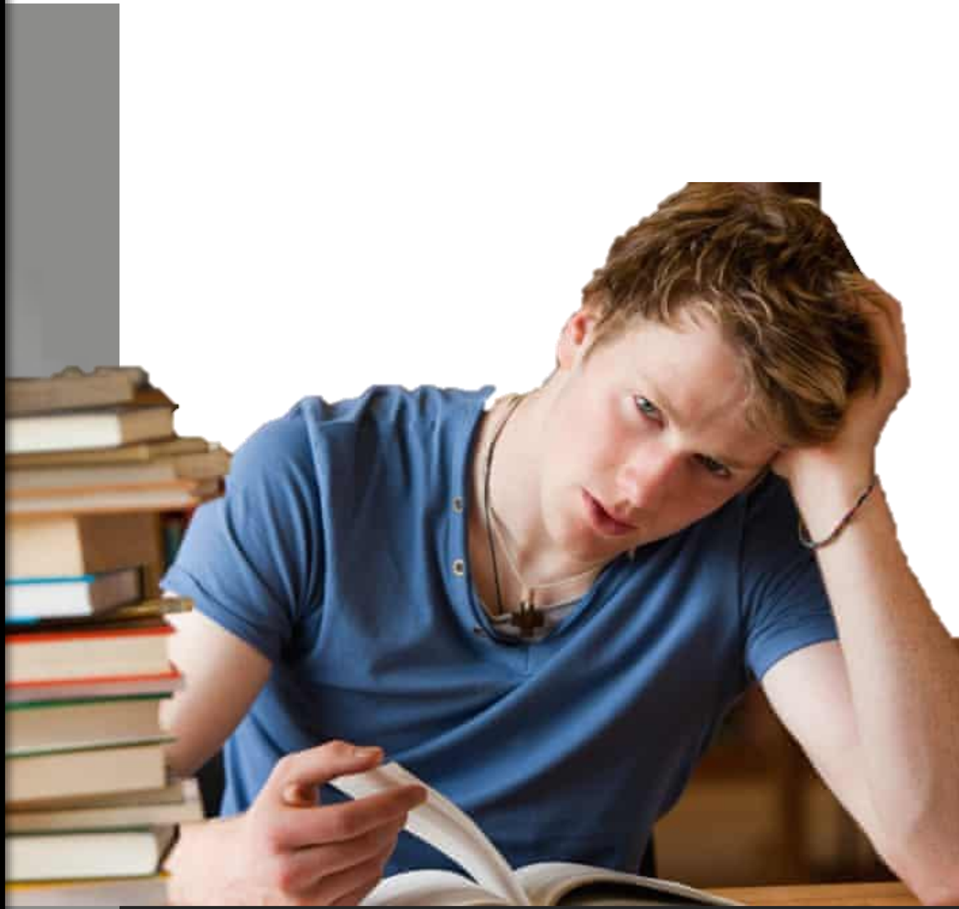


Hello

Hi

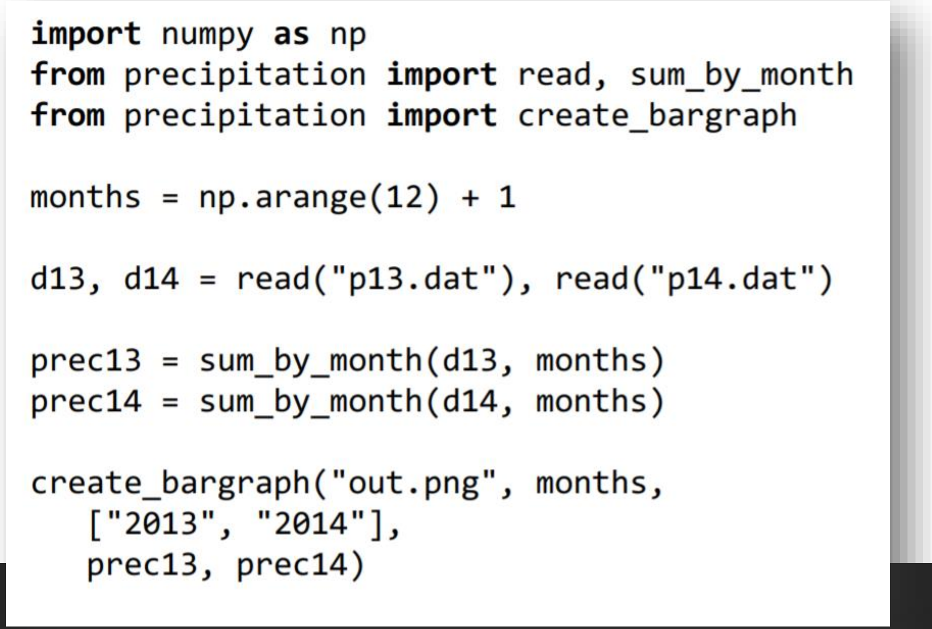
Could you check if the precipitation of Rio de Janeiro remains constant across years?

Sure!



Running a Computational Experiment

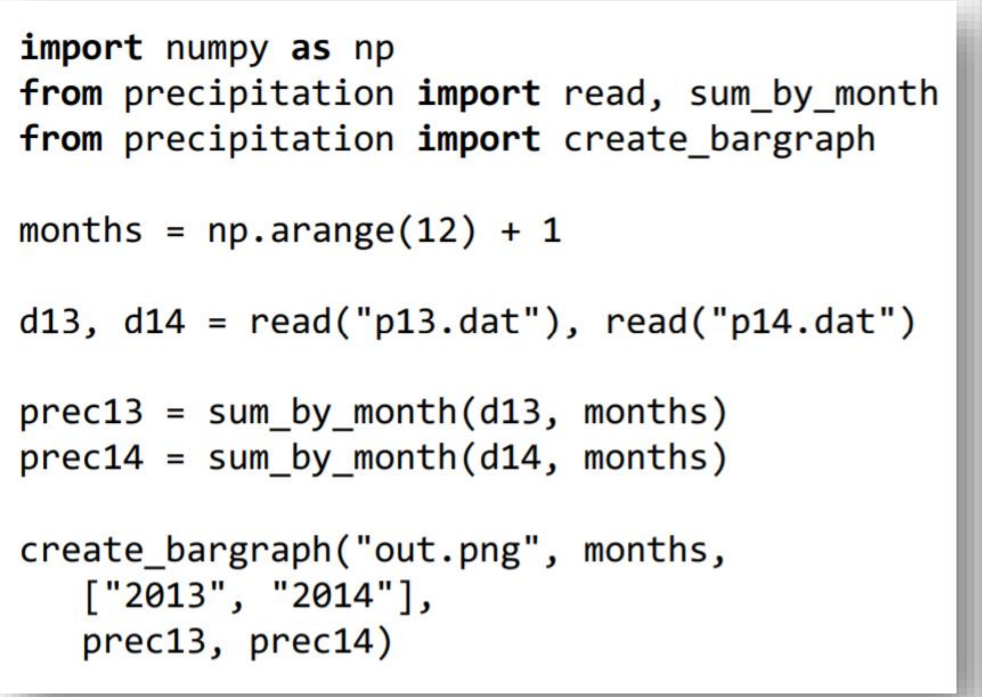
- A **workflow or script** is just part of an experiment
- In order to prove or refute an hypothesis, it is usually necessary to model and run the workflow or script varying inputs, parameters and programs and finally analyze the outputs
  - Each of those runs is called a trial of the experiment



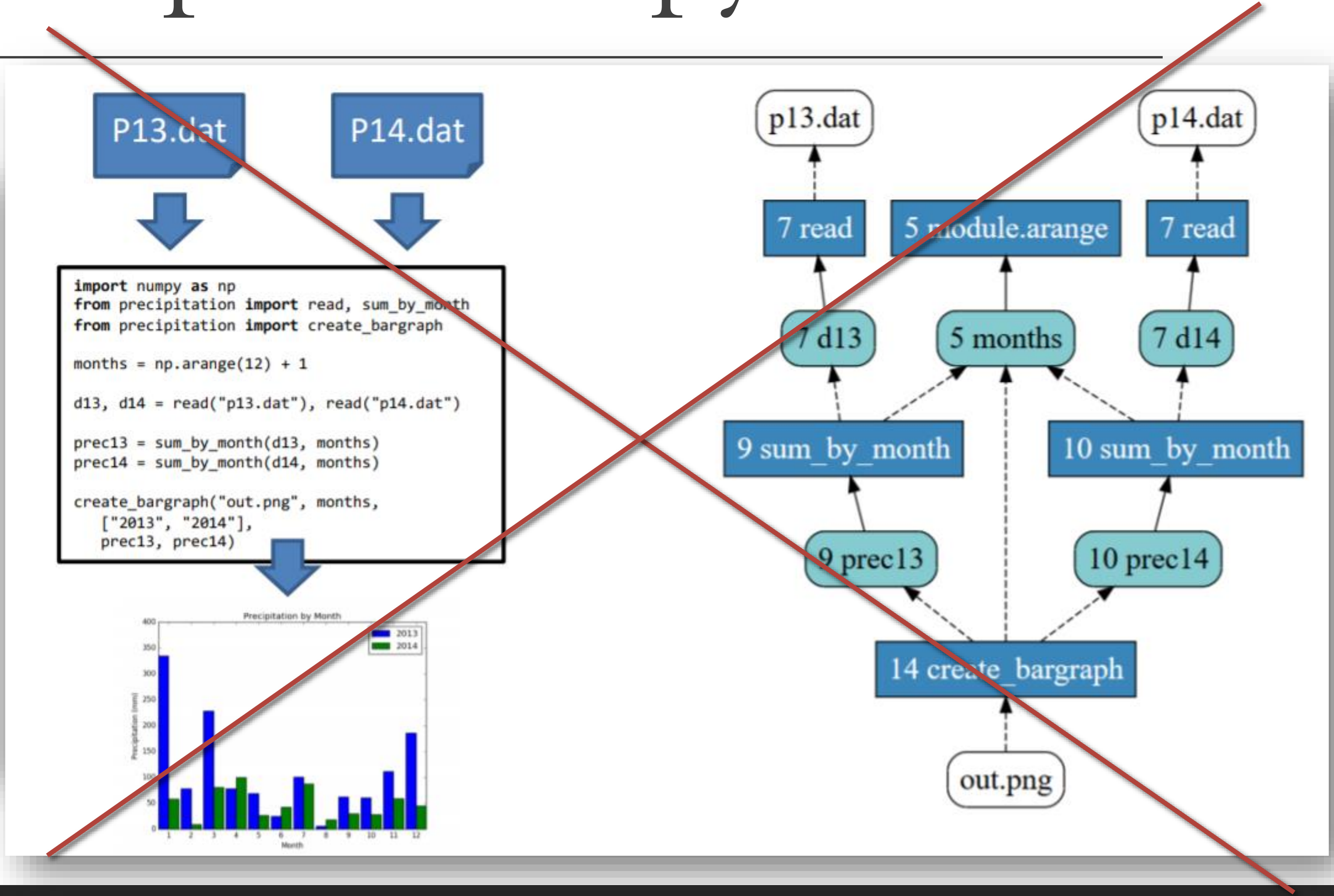
```
import numpy as np
from precipitation import read, sum_by_month
from precipitation import create_bargraph

months = np.arange(12) + 1

d13, d14 = read("p13.dat"), read("p14.dat")

prec13 = sum_by_month(d13, months)
prec14 = sum_by_month(d14, months)

create_bargraph("out.png", months,
   ["2013", "2014"],
   prec13, prec14)
```

Source: Braganholo, V. Scientific Experiments as Workflows and Scripts

1st iteration – experiment.py



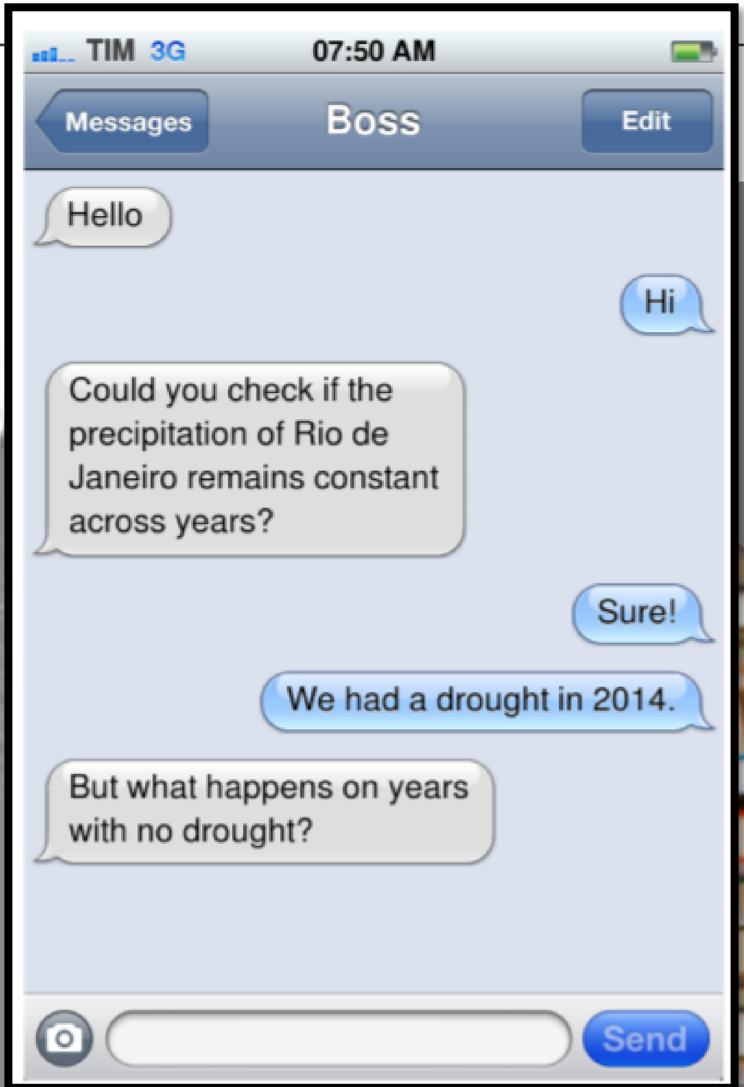
```
import numpy as np
from precipitation import read, sum_by_month
from precipitation import create_bargraph

months = np.arange(12) + 1

d13, d14 = read("p13.dat"), read("p14.dat")

prec13 = sum_by_month(d13, months)
prec14 = sum_by_month(d14, months)

create_bargraph("out.png", months,
   ["2013", "2014"],
   prec13, prec14)
```



```
import numpy as np
from precipitation import read, sum_by_month
from precipitation import create_bargraph

months = np.arange(12) + 1

d13, d14 = read("p13.dat"), read("p14.dat")

prec13 = sum_by_month(d13, months)
prec14 = sum_by_month(d14, months)

create_bargraph("out.png", months,
   ["2013", "2014"],
   prec13, prec14)
```

Source: Braganholo, V. Scientific Experiments as Workflows and Scripts

Running a Computational Experiment



Hello

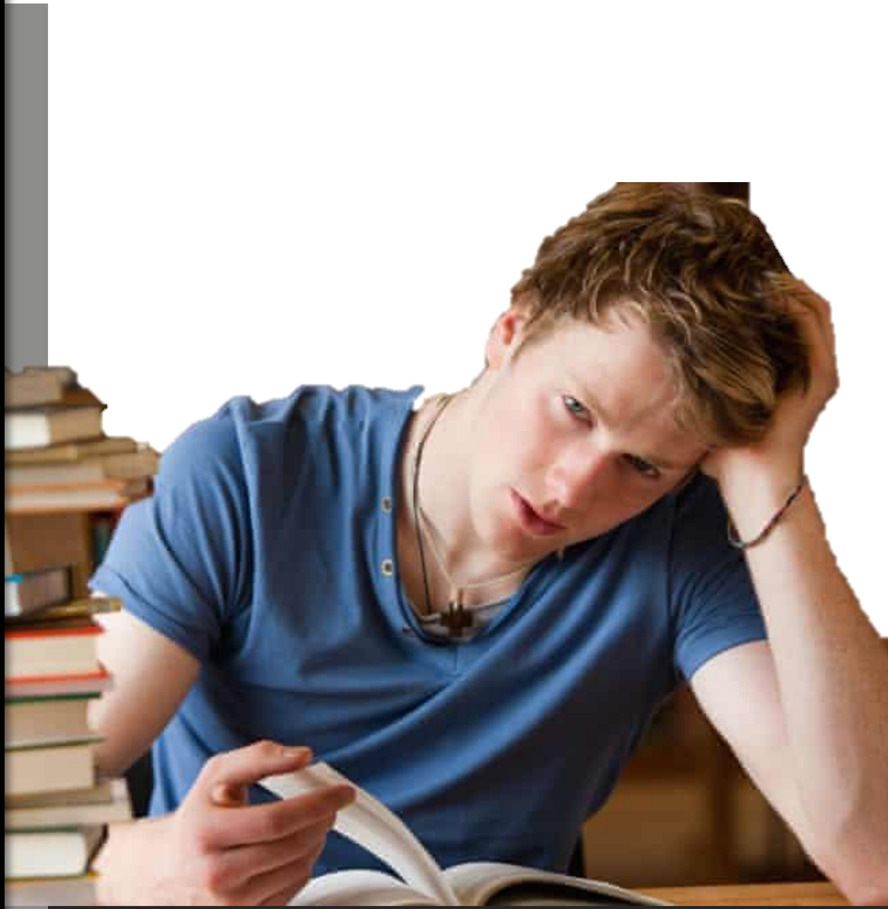
Hi

Could you check if the precipitation of Rio de Janeiro remains constant across years?

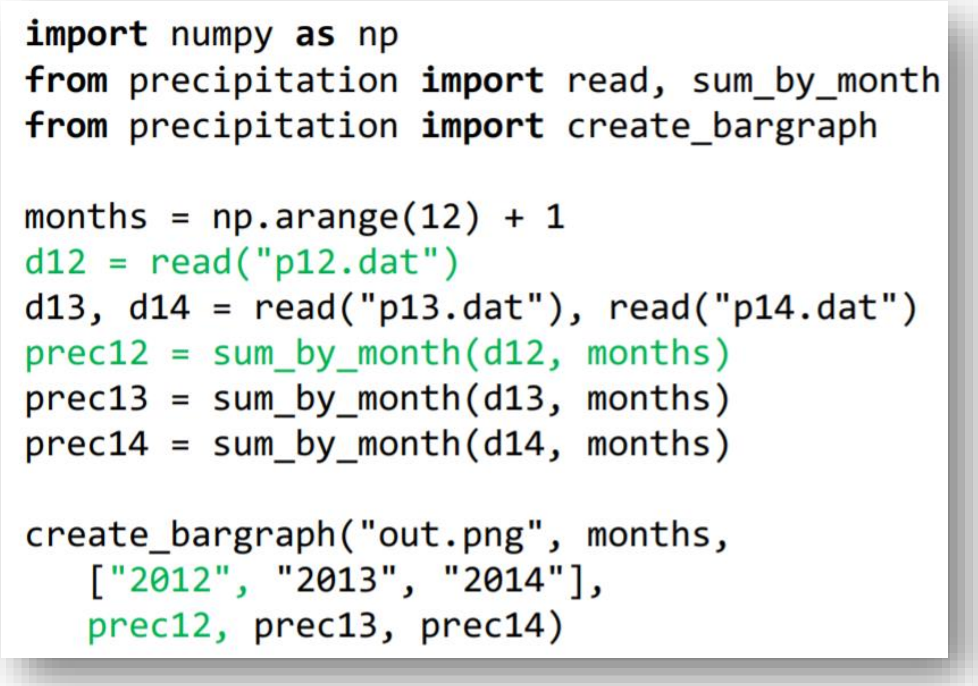
Sure!

We had a drought in 2014.

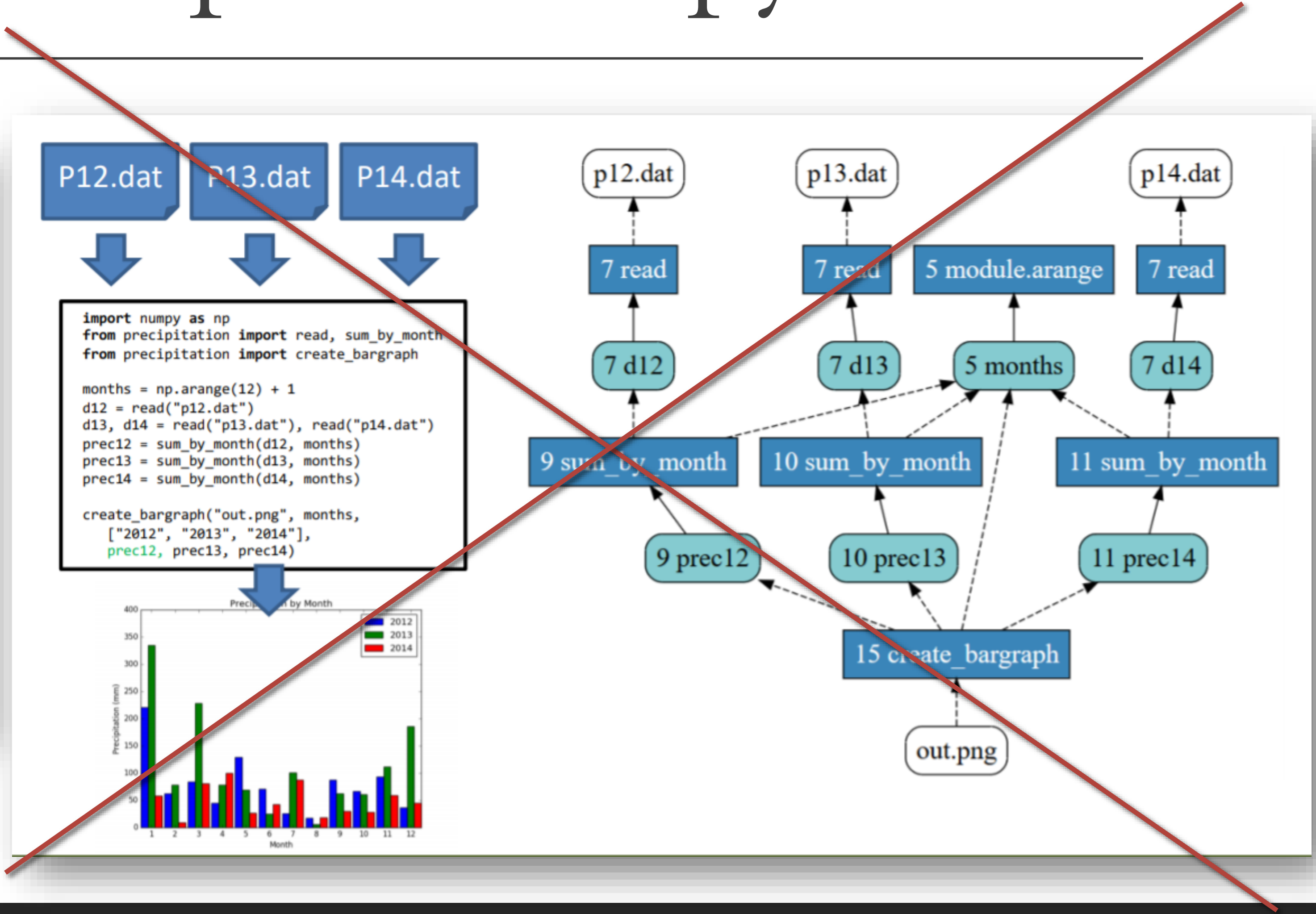
But what happens on years with no drought?



2nd Iteration – experiment.py



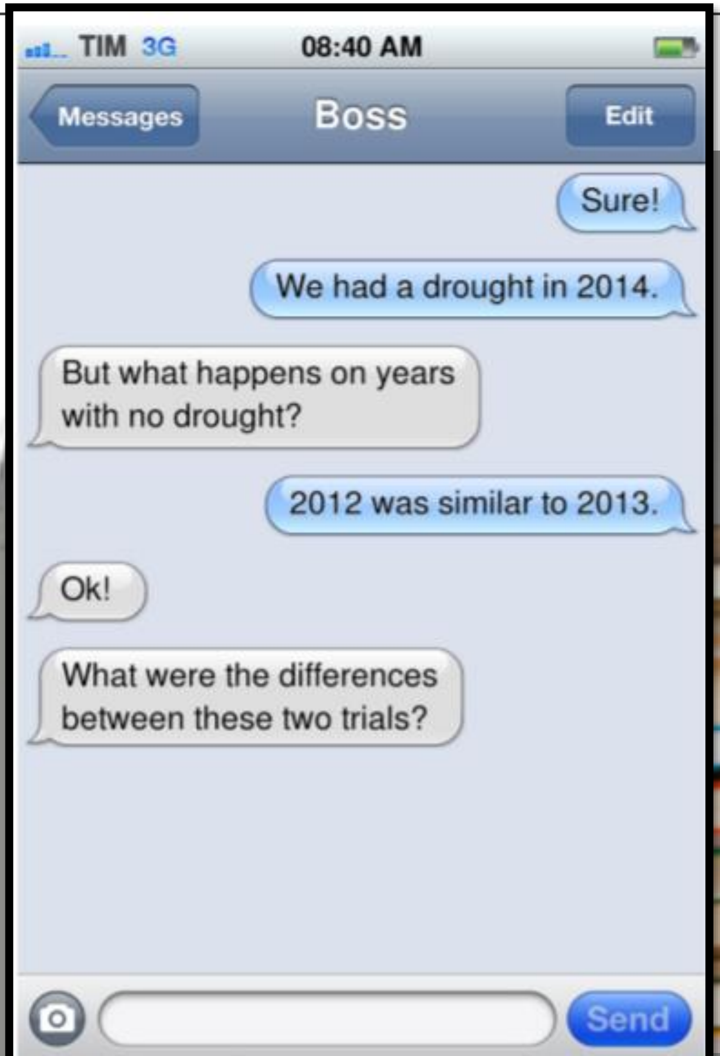
```
import numpy as np
from precipitation import read, sum_by_month
from precipitation import create_bargraph

months = np.arange(12) + 1
d12 = read("p12.dat")
d13, d14 = read("p13.dat"), read("p14.dat")
prec12 = sum_by_month(d12, months)
prec13 = sum_by_month(d13, months)
prec14 = sum_by_month(d14, months)

create_bargraph("out.png", months,
   ["2012", "2013", "2014"],
   prec12, prec13, prec14)
```



```
import numpy as np
from precipitation import read, sum_by_month
from precipitation import create_bargraph

months = np.arange(12) + 1
d12 = read("p12.dat")
d13, d14 = read("p13.dat"), read("p14.dat")
prec12 = sum_by_month(d12, months)
prec13 = sum_by_month(d13, months)
prec14 = sum_by_month(d14, months)

create_bargraph("out.png", months,
   ["2012", "2013", "2014"],
   prec12, prec13, prec14)
```

Source: Braganholo, V. Scientific Experiments as Workflows and Scripts

Running a Computational Experiment



Sure!

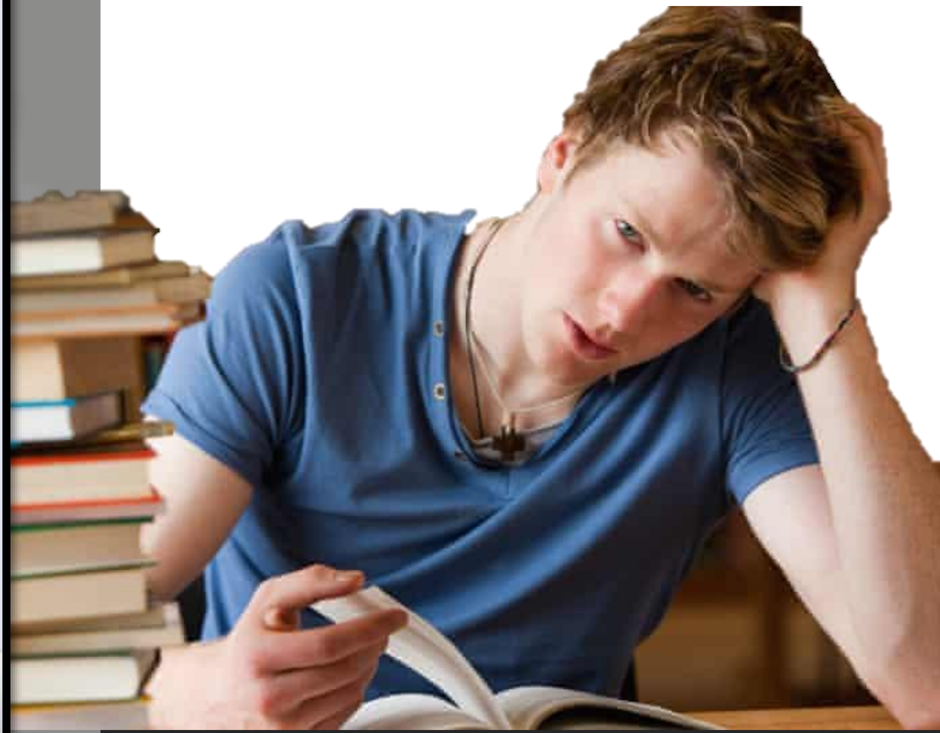
We had a drought in 2014.

But what happens on years with no drought?

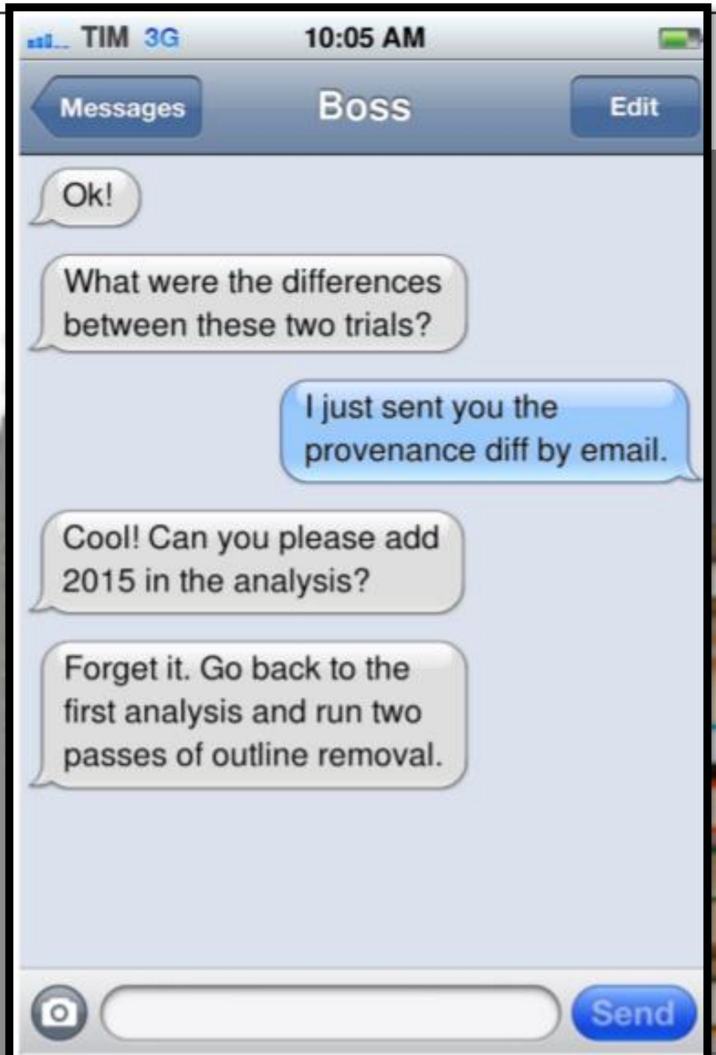
2012 was similar to 2013.

Ok!

What were the differences between these two trials?



Running a Computational Experiment



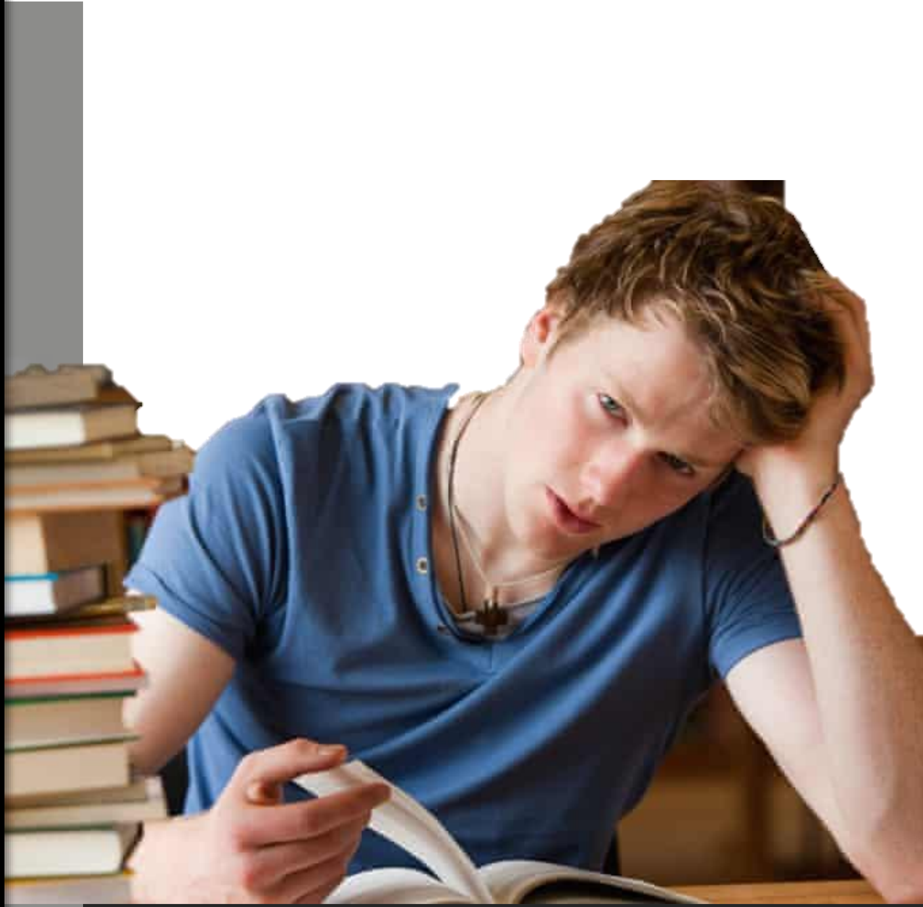
Ok!

What were the differences between these two trials?

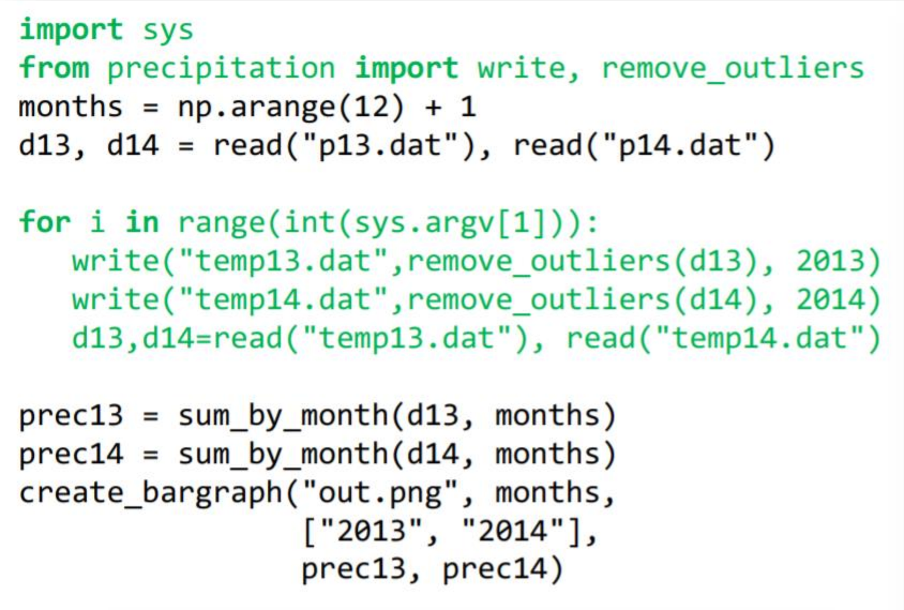
I just sent you the provenance diff by email.

Cool! Can you please add 2015 in the analysis?

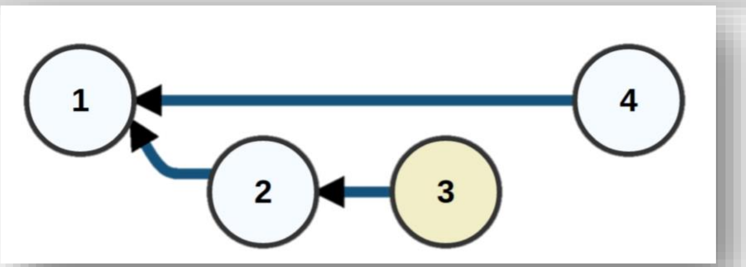
Forget it. Go back to the first analysis and run two passes of outline removal.

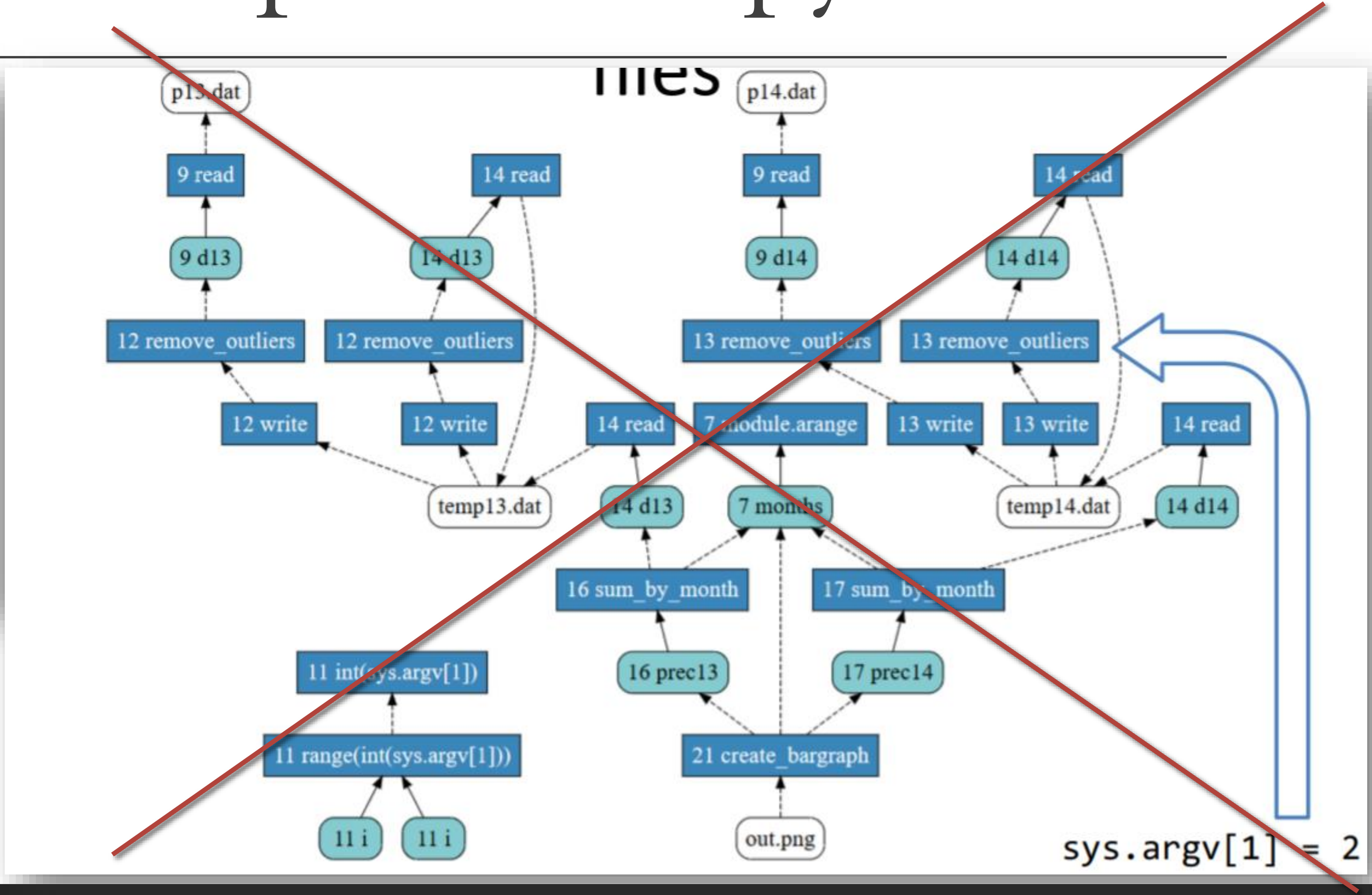


4th iteration – experiment.py



```
import sys
from precipitation import write, remove_outliers
months = np.arange(12) + 1
d13, d14 = read("p13.dat"), read("p14.dat")

for i in range(int(sys.argv[1])):
   write("temp13.dat",remove_outliers(d13), 2013)
   write("temp14.dat",remove_outliers(d14), 2014)
   d13,d14=read("temp13.dat"), read("temp14.dat")

prec13 = sum_by_month(d13, months)
prec14 = sum_by_month(d14, months)
create_bargraph("out.png", months,
                ["2013", "2014"],
                prec13, prec14)
```



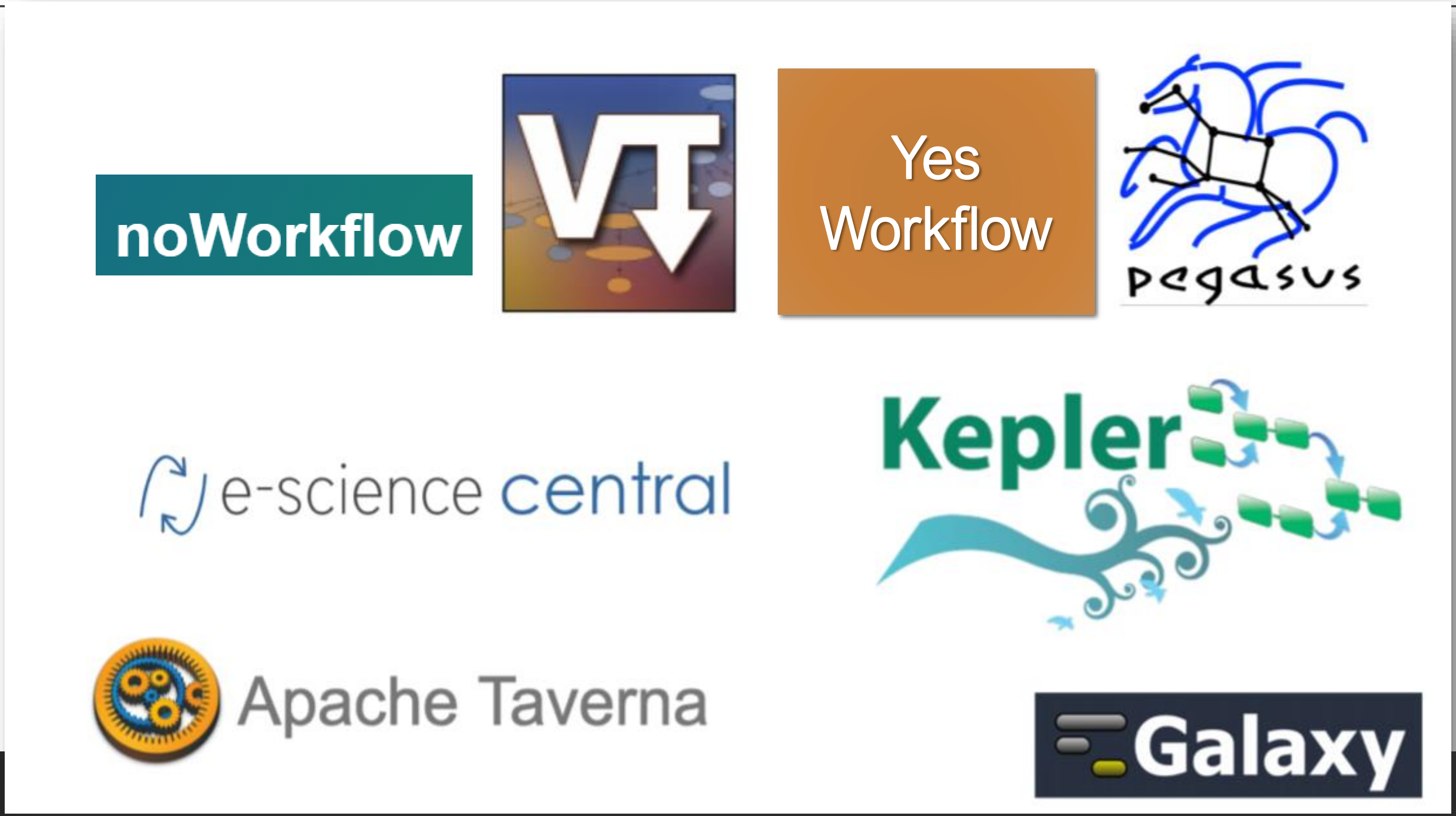


sys.argv[1] = 2

Source: Braganholo, V. Scientific Experiments as Workflows and Scripts

Scripts & Workflows without provenance...

Workflow Management Systems



VisTrails

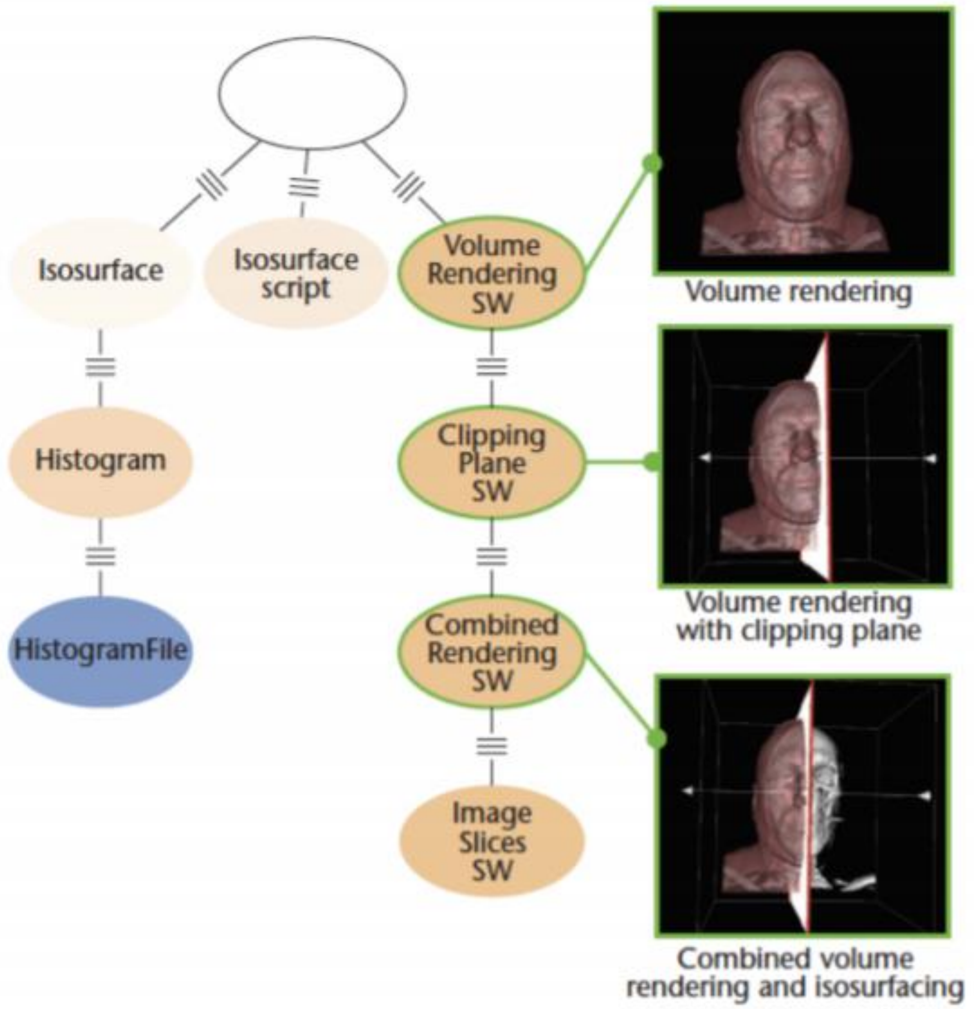
Python

Visual drag and drop interface for workflow composition

Captures history of changes in the workflow structure

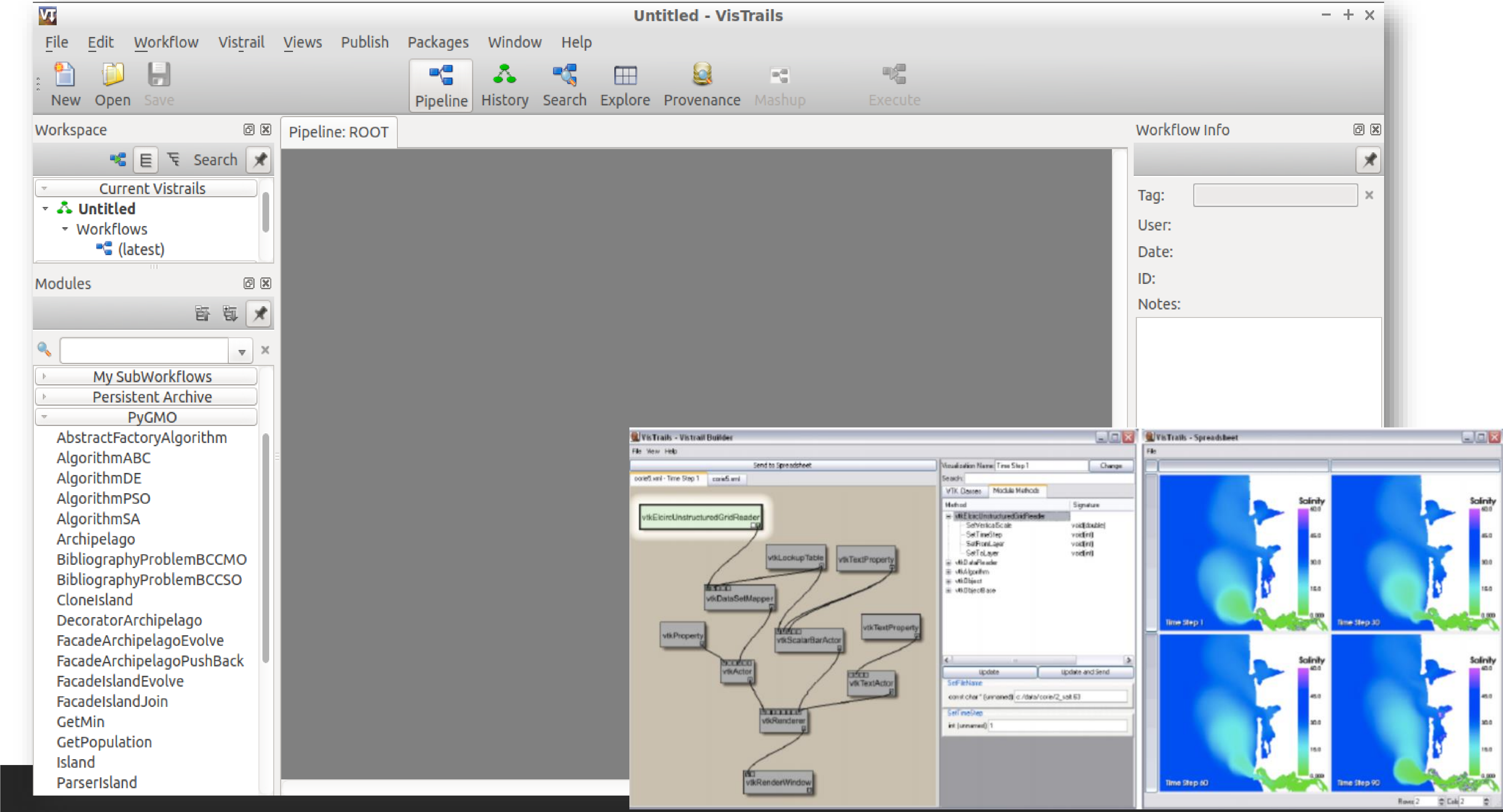
Allows comparing results side-by-side

Focus on visualization



Source: Freire et al., 2008. Provenance for Computational Tasks: A Survey.

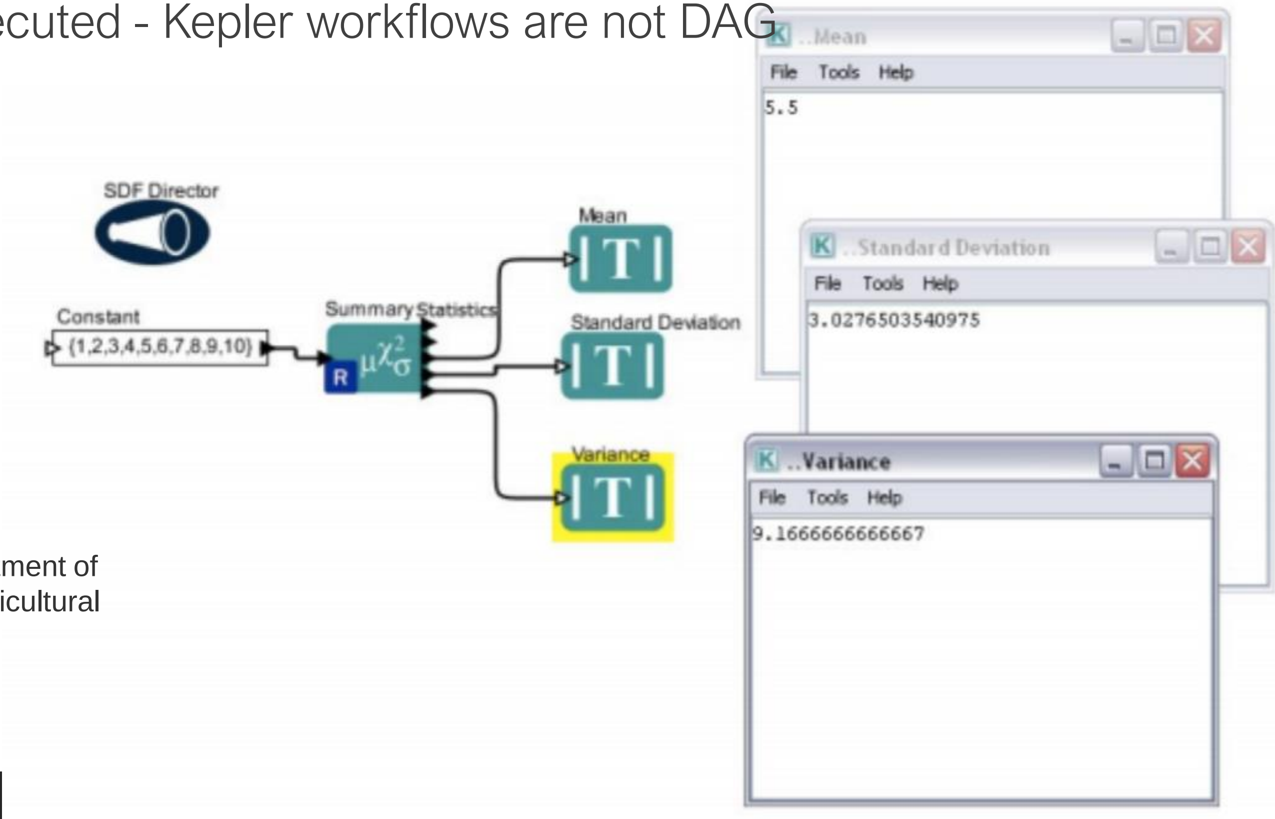
Evolutionary Scientific Workflows
Sergio Manuel Serra da Cruz; Anderson de Oliveira; Fabricio Firmino de Faria


Untitled - VisTrails
File
Edit
Workflow
Vistrail
Views
Publish
Packages
Window
Help
New
Open
Save
Pipeline
History
Search
Explore
Provenance
Mashup
Execute
Workspace
Search
Current Vistrails
Untitled
Workflows
(latest)
Modules
My SubWorkflows
Persistent Archive
PyGMO
AbstractFactoryAlgorithm
AlgorithmABC
AlgorithmDE
AlgorithmPSO
AlgorithmSA
Archipelago
BibliographyProblemBCCMO
BibliographyProblemBCCSO
CloneIsland
DecoratorArchipelago
FacadeArchipelagoEvolve
FacadeArchipelagoPushBack
FacadeIslandEvolve
FacadeIslandJoin
GetMin
GetPopulation
Island
ParserIsland
Pipeline: ROOT
Workflow Info
Tag:
User:
Date:
ID:
Notes:
VisTrails - Vistrail Builder
File
View
Help
Send to Spreadsheet
vtkElcircUnstructuredGridReader
vtkLookupTable
vtkTextProperty
vtkDataSetMapper
vtkProperty
vtkScalarBarActor
vtkTextProperty
vtkActor
vtkTextActor
vtkRenderer
vtkRenderWindow
Visualization Name: Time Step 1
Change
Search:
VTK Classes
Module Methods
Method
Signature
Update
Update and Send
VisTrails - Spreadsheet
File
Salinity
Time Step 1
Time Step 30
Time Step 60
Time Step 90
Rows: 2
Cols: 2

Kepler

Drag and Drop graphical interface for workflow composition

Different actors that rules how the workflow is executed - Kepler workflows are not DAG



```
5.5
```

```
3.0276503540975
```

```
9.1666666666667
```

Enriching Agronomic Experiments with Data Provenance
Sergio Manuel Serra da Cruz (Federal Rural University of Rio de Janeiro (UFRRJ), Department of Mathematics, Rio de Janeiro, Brazil) and Jose Antonio Pires do Nascimento (Brazilian Agricultural Research Corporation (EMBRAPA), Agricultural Sector, Rio de Janeiro, Brazil)
Source Title: International Journal of Agricultural and Environmental Information Systems (IJAEIS)8(3)

Provenance Management Systems for Scripts

noWorkflow – captures provenance for Python scripts

- https://github.com/gems-uff/noworkflow
- Paper: Collecting and Analyzing Provenance on Interactive Notebooks: when IPython meets noWorkflow.

RDataTracker – captures provenance for R scripts

- https://github.com/End-to-end-provenance/RDataTracker
- Paper: RDataTracker: Collecting Provenance in an Interactive Scripting Environment

Sumatra – captures provenance for Python, R and MatLab scripts

- https://pythonhosted.org/Sumatra/
- Paper: Sumatra: a toolkit for reproducible research

Black Box x White Box

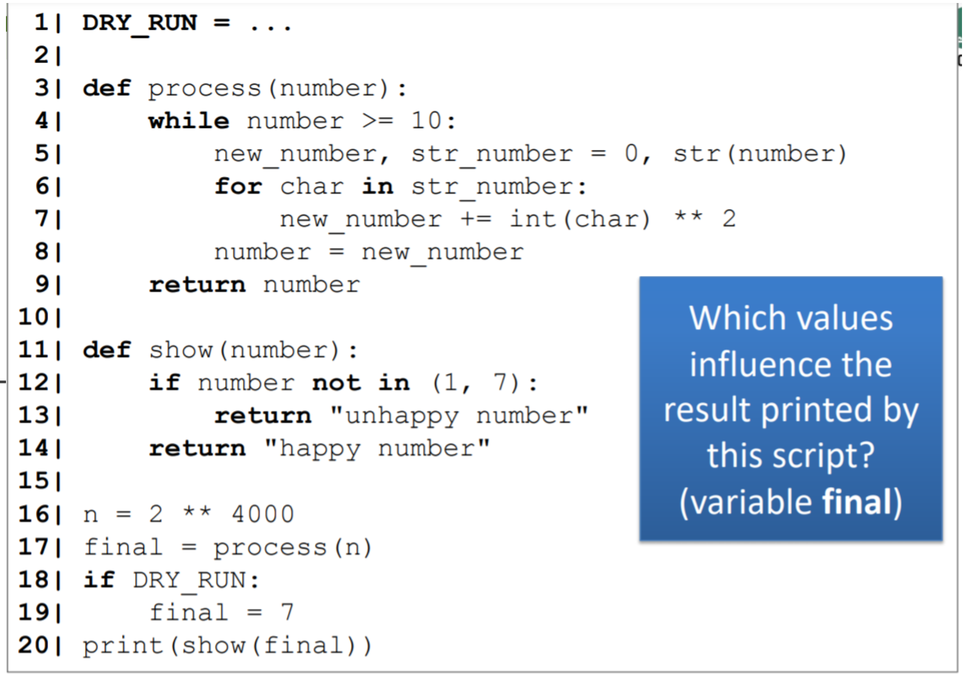
Black Box x White Box

- In **Workflow systems**, activities (tasks) are black boxes
  - What goes in and out are known, but what happens inside is not known

In **scripts**, activities can be black boxes or white boxes

- An activity in a script can call an external program, and in this the activity is a black box
- When the function is implemented in Python, it is a white box

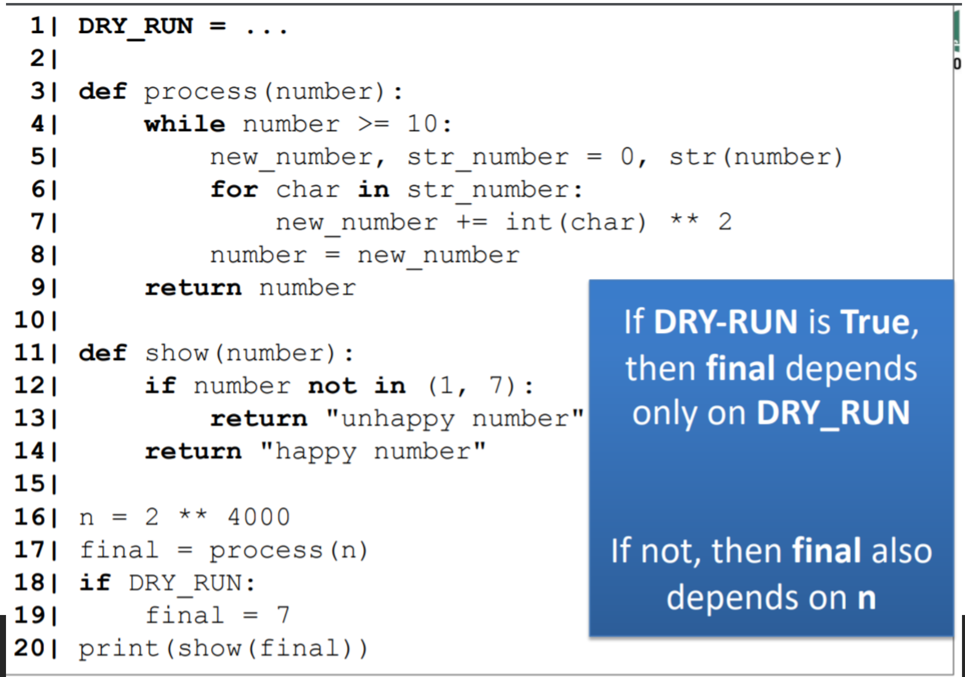
- Black boxes have implications in reproducibility and provenance analysis



```
DRY_RUN = ...

def process(number):
    while number >= 10:
        new_number, str_number = 0, str(number)
        for char in str_number:
            new_number += int(char) ** 2
        number = new_number
    return number

def show(number):
    if number not in (1, 7):
        return "unhappy number"
    return "happy number"

n = 2 ** 4000
final = process(n)
if DRY_RUN:
    final = 7
print(show(final))
```

Which values influence the result printed by this script? (variable **final**)

ource: Pimentel et al., 2016. Fine-grained Provenance Collection over Scripts Through Program Slicing



```
DRY_RUN = ...

def process(number):
    while number >= 10:
        new_number, str_number = 0, str(number)
        for char in str_number:
            new_number += int(char) ** 2
        number = new_number
    return number

def show(number):
    if number not in (1, 7):
        return "unhappy number"
    return "happy number"

n = 2 ** 4000
final = process(n)
if DRY_RUN:
    final = 7
print(show(final))
```

If **DRY-RUN** is **True**, then **final** depends only on **DRY_RUN**

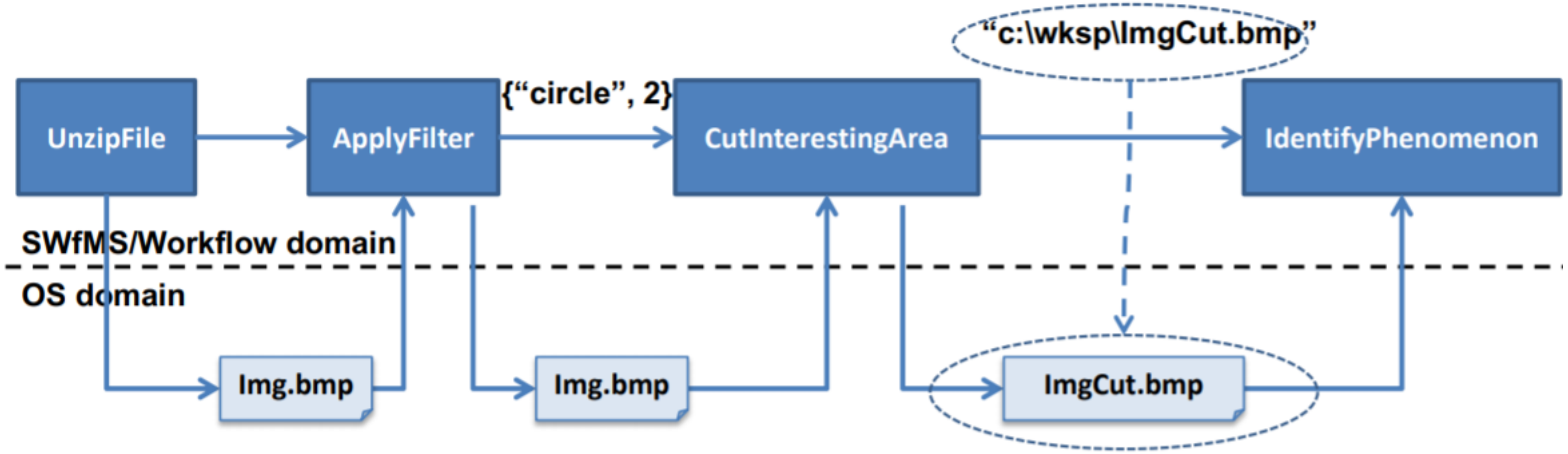
If not, then **final** also depends on **n**

ource: Pimentel et al., 2016. Fine-grained Provenance Collection over Scripts Through Program Slicing

Implications of Black Boxes

If process(number) were a black box, anything could happen inside it

- It could, for example, read a file that could influence the value returned by the function, so dependencies would be missed
- This is a common case of implicit provenance, that is missed by several provenance capturing approaches



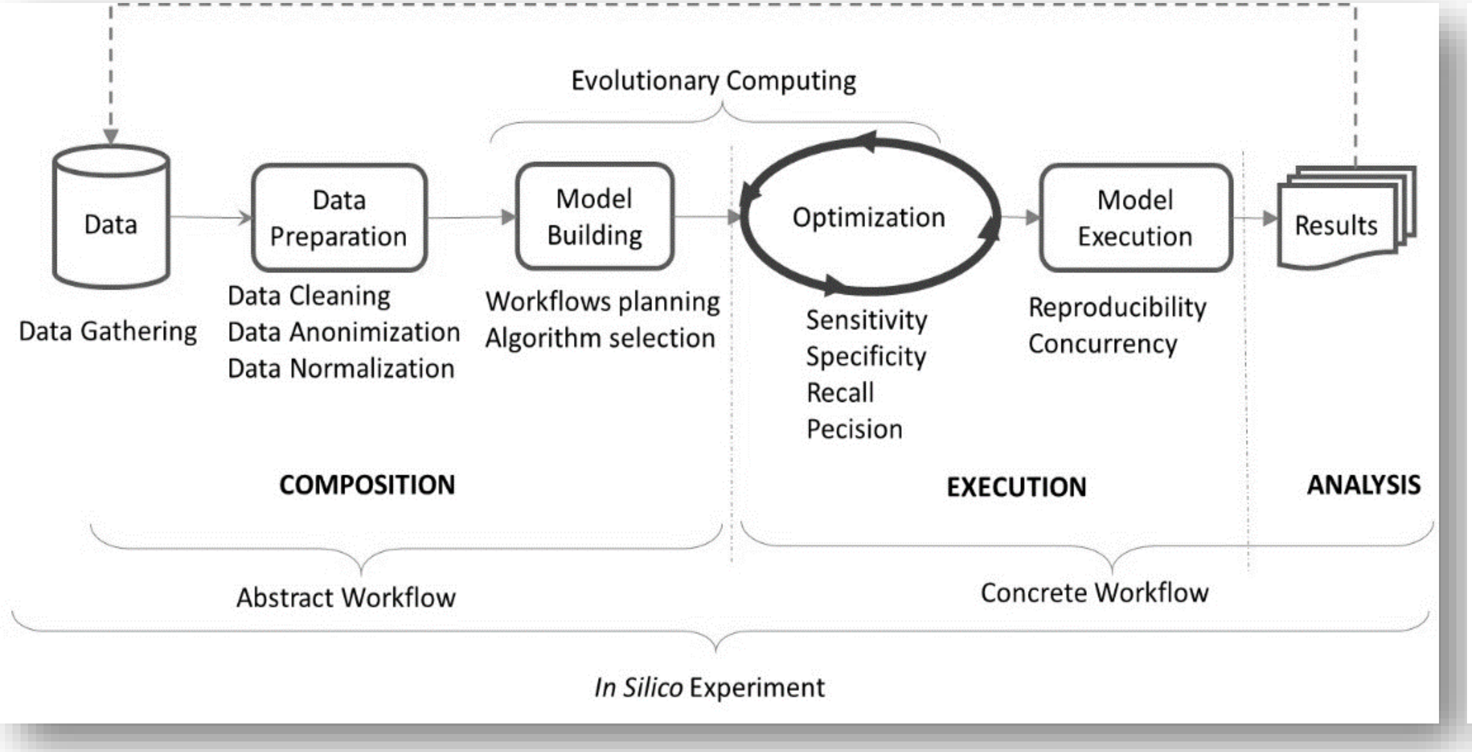
Evolutionary Workflows

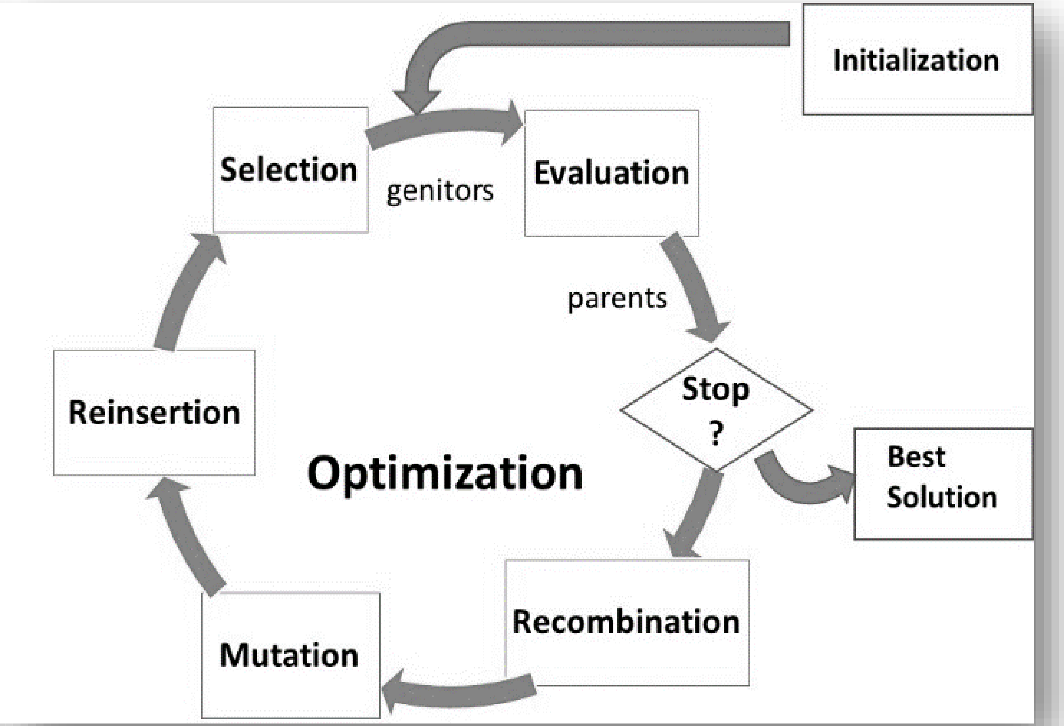
Is a workflow with feedback loops used to represent the computational problem and the optimization cycle supporting the multiple patterns required by current optimization problems.

- Are modular artifacts that read a considerable number of input files and typically produce one output; the trained model that can be used to solve optimization problems.
- Compared to monolithic codes, they provide users with a clear view of the flow within the optimization process, making it easier to test and substitute those optimization modules.

Evolutionary Scientific Workflows
Sergio Manuel Serra da Cruz; Anderson de Oliveira; Fabricio Firmino de Faria
Congress on Evolutionary Computation, 2018

Evolutionary Workflows



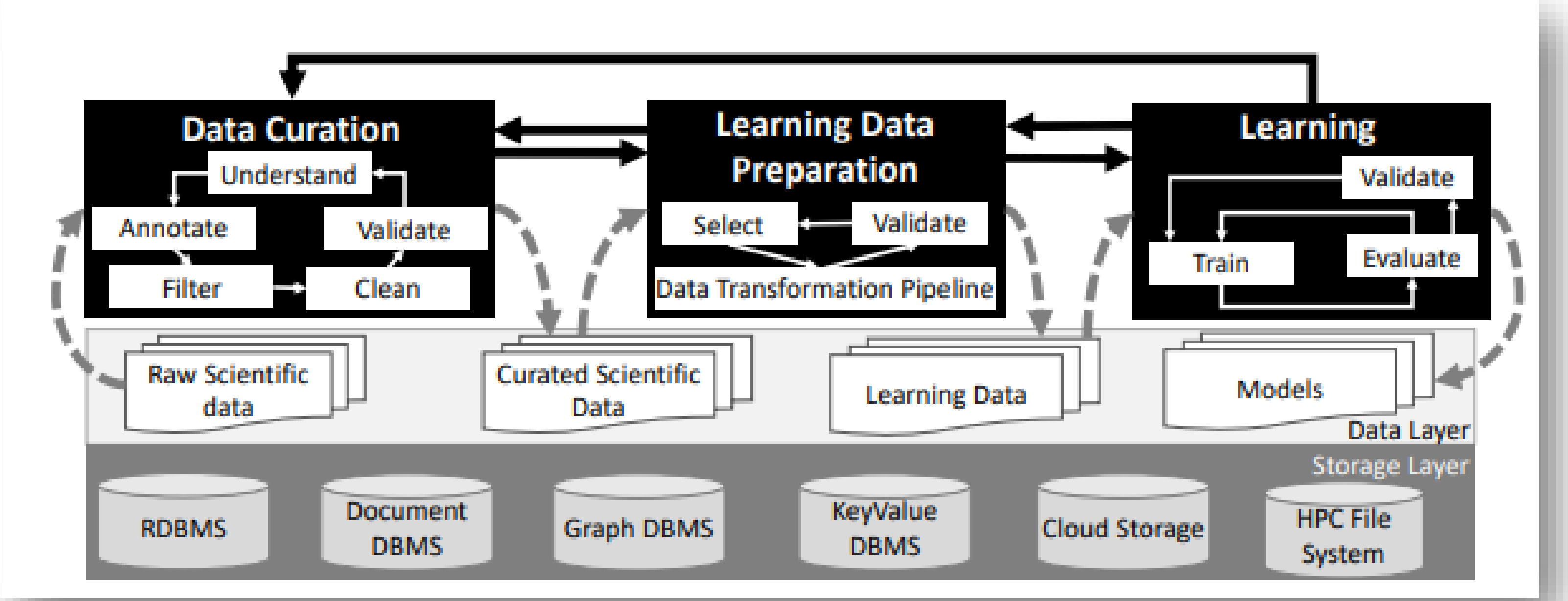


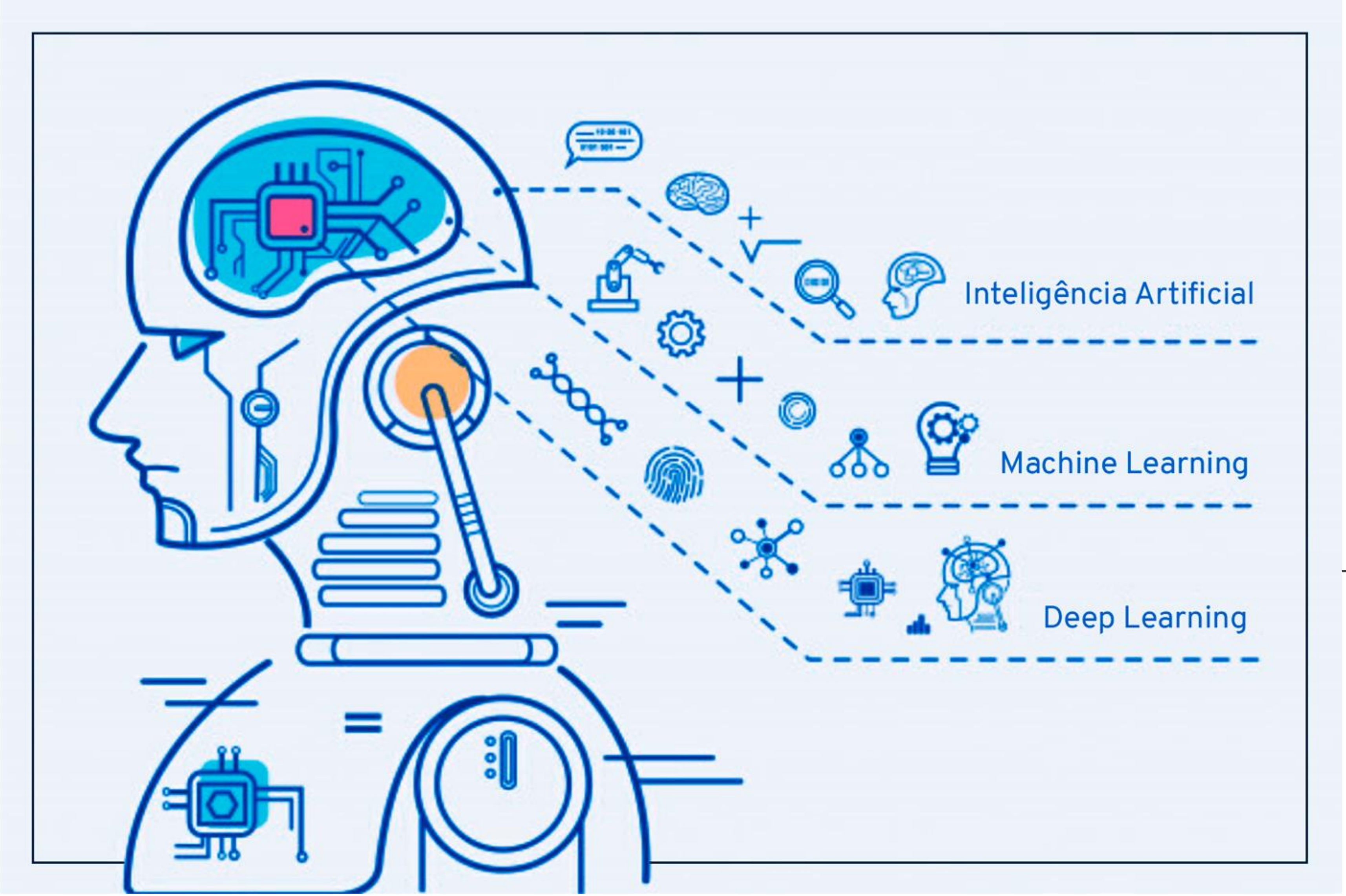
Evolutionary Workflows – Features

- **Randomness in data access pattern: S**how more randomness in access pattern, caused by the sampling of input files for achieving convergence of the optimization process;
- **Data optimization**: The workflow notation needs to support data objects that can represent the experiment, optimization plan, and datasets.
- **Asynchronous communication**: EA (or robust optimization activities) require the evaluation of designs in asynchronous terms so that evaluation of several designs can be requested without blocking the execution of the algorithms;
- **Concurrent execution**: mechanisms must be provided to support the solver's parallel execution and other modules, like robust optimization;
- **Instance routing:** many instances of the same process running concurrently, the messaging system must deliver the instances' messages, being handled through an adequate correlation model, which can ensure that data will flow between the modules as required.

Machine Learning Workflows

Describes the processes involved in machine learning work. Various stages help to universalize the process of building and maintaining machine learning networks.





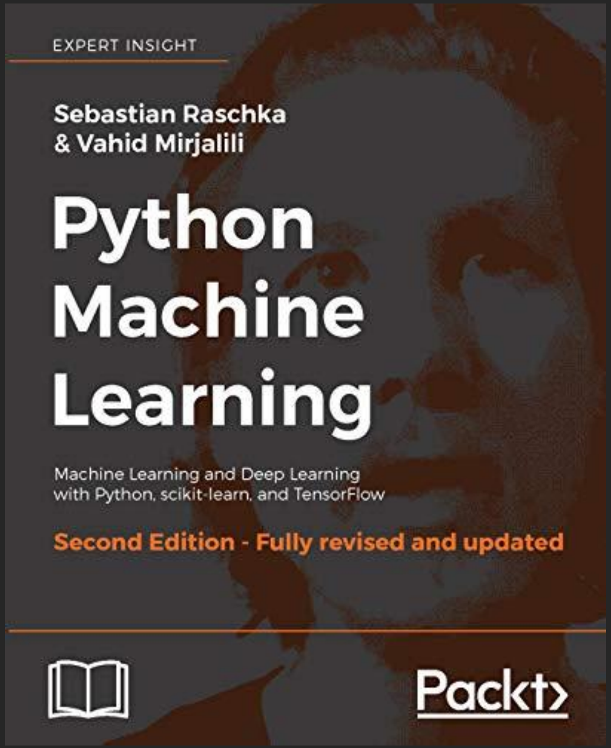
Hands on...

NOTEBOOK:

MACHINE LEARNING

Scikit learn

References



EXPERT INSIGHT

Sebastian Raschka
& Vahid Mirjalili

Python Machine Learning

Machine Learning and Deep Learning
with Python, scikit-learn, and TensorFlow

Second Edition - Fully revised and updated

Packt>