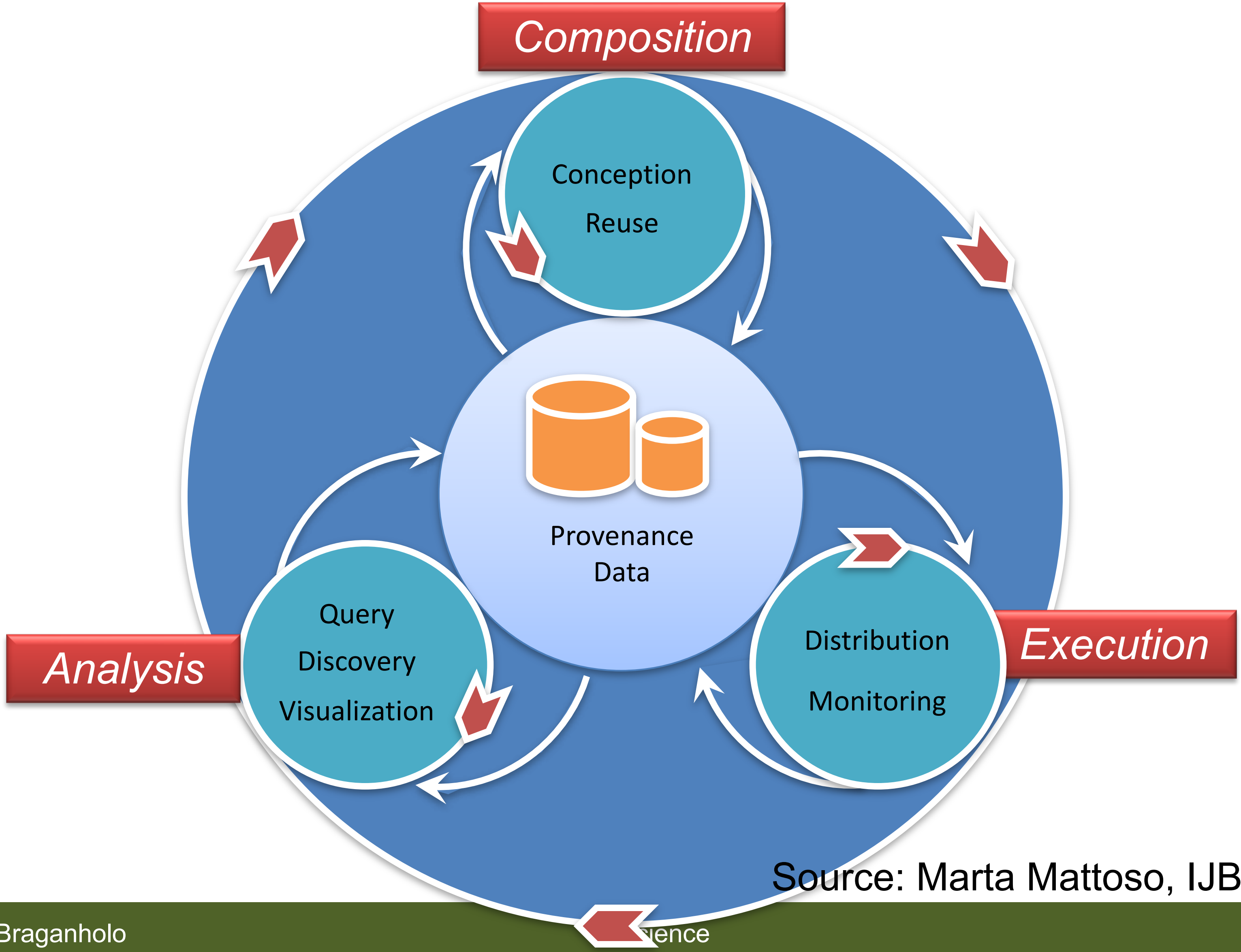
Scientific Experiments as Workflows and Scripts

Instituto de computação

Vanessa Braganholo

The experiment life cycle



Source: Marta Mattoso, IJBPIM, 2010

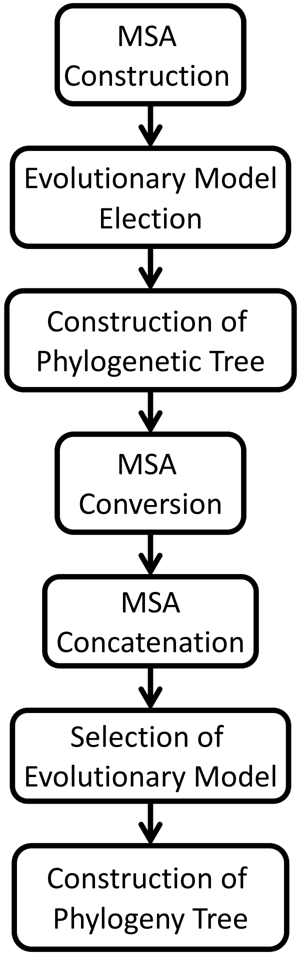
Agenda

- Abstract Representation of Scientific Experiments
- Workflows
- Scripts
- Black Boxes X White Boxes
- Workflow Management Systems
- Provenance Management Systems for Scripts

Composition: Conceiving Scientific Experiments

- Scientists usually design an experiment using a **high abstraction level representation** that is later mapped into a workflow or script

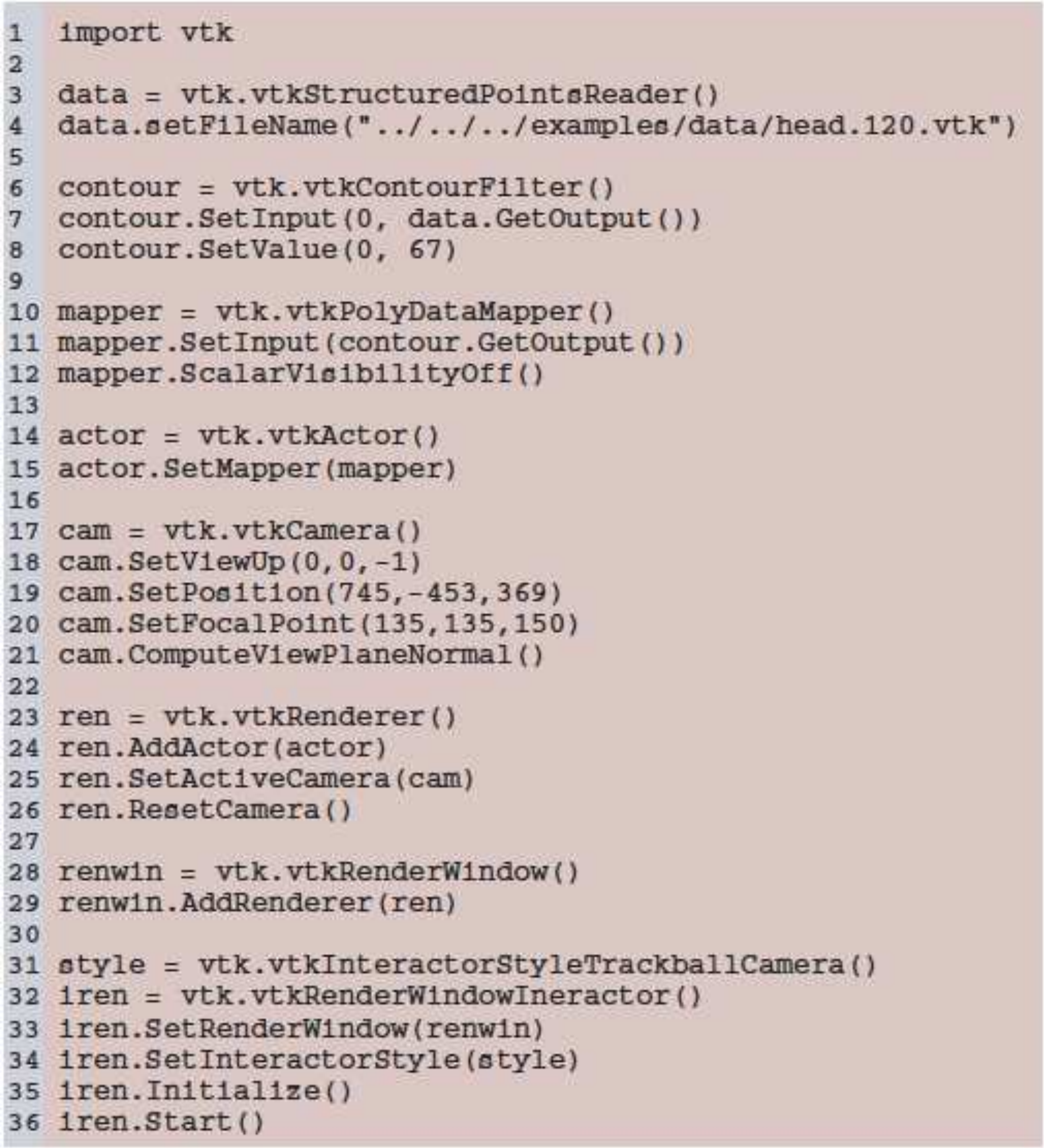
Phylogeny Analysis Experiment (Abstract Workflow)


MSA Construction
Evolutionary Model Election
Construction of Phylogenetic Tree
MSA Conversion
MSA Concatenation
Selection of Evolutionary Model
Construction of Phylogeny Tree

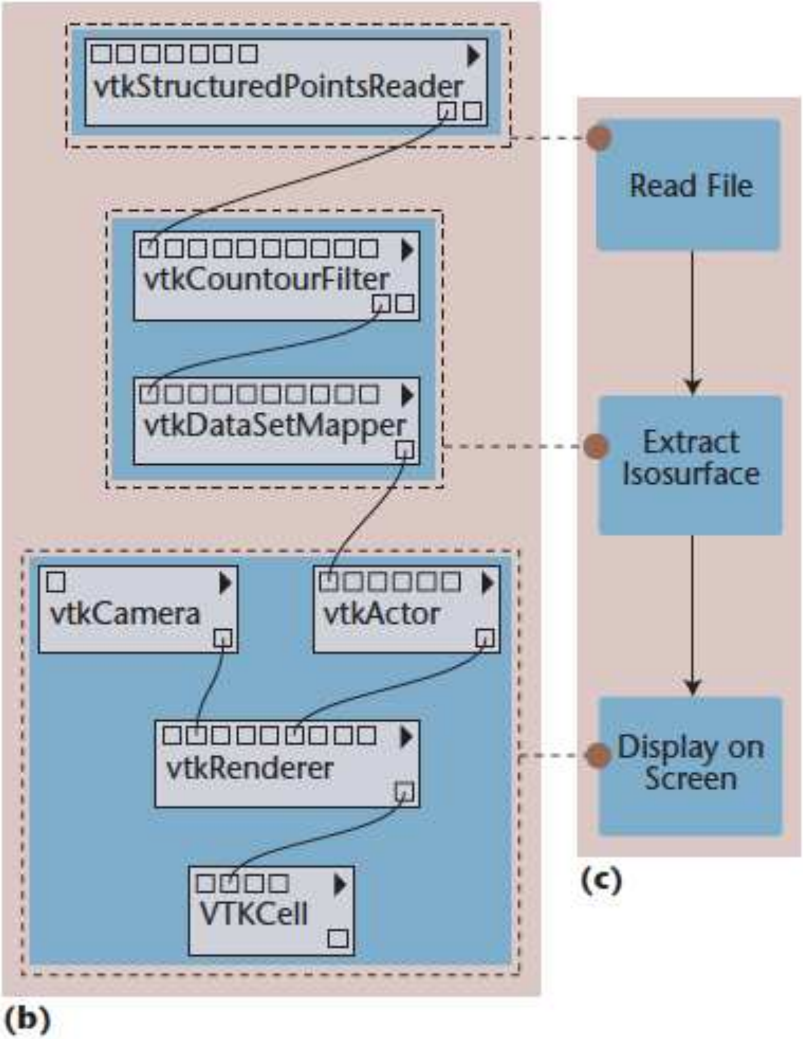
Source: MARINHO et al. Deriving scientific workflows from algebraic experiment lines: A practical approach. Future Generation Computer Systems, v. 68, p. 111-127, 2017.

Abstract x Concrete

- The **abstract** workflow is later mapped into a **concrete** workflow or script



```
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()
```



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

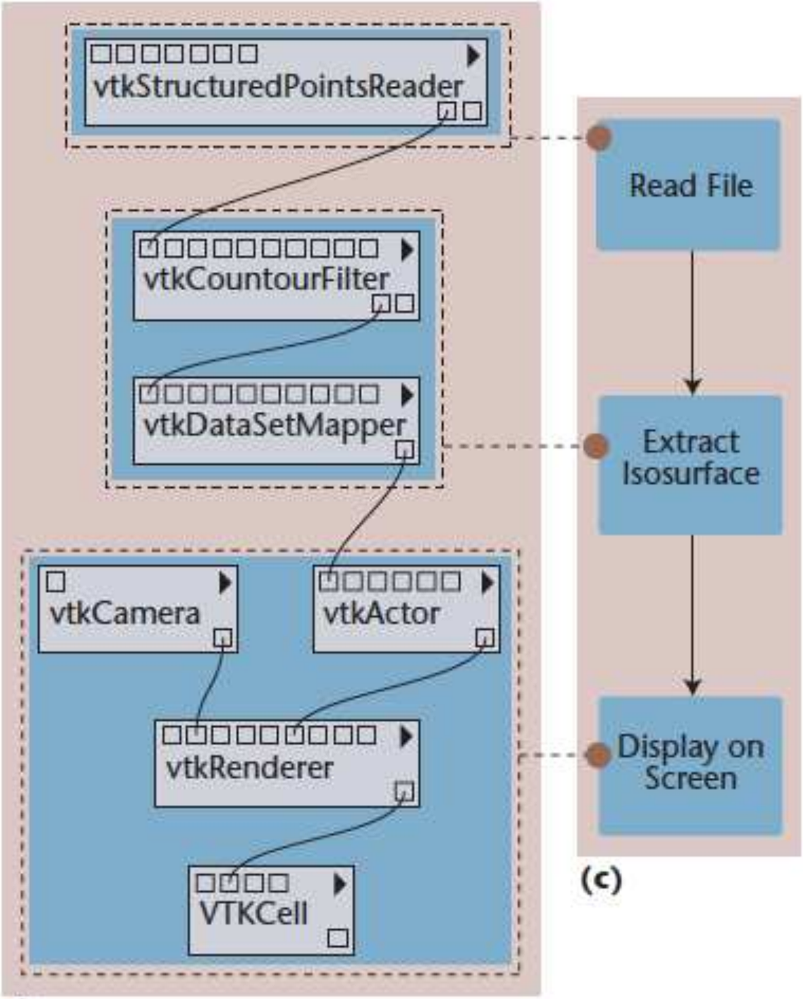
Scientific Workflow

- A scientific workflow is a **chain of activities** organized in the form of a **data flow**

Data Flow

- In a data flow, the **execution is guided by the data**
- As soon as all the input data of an activity is available, it starts executing

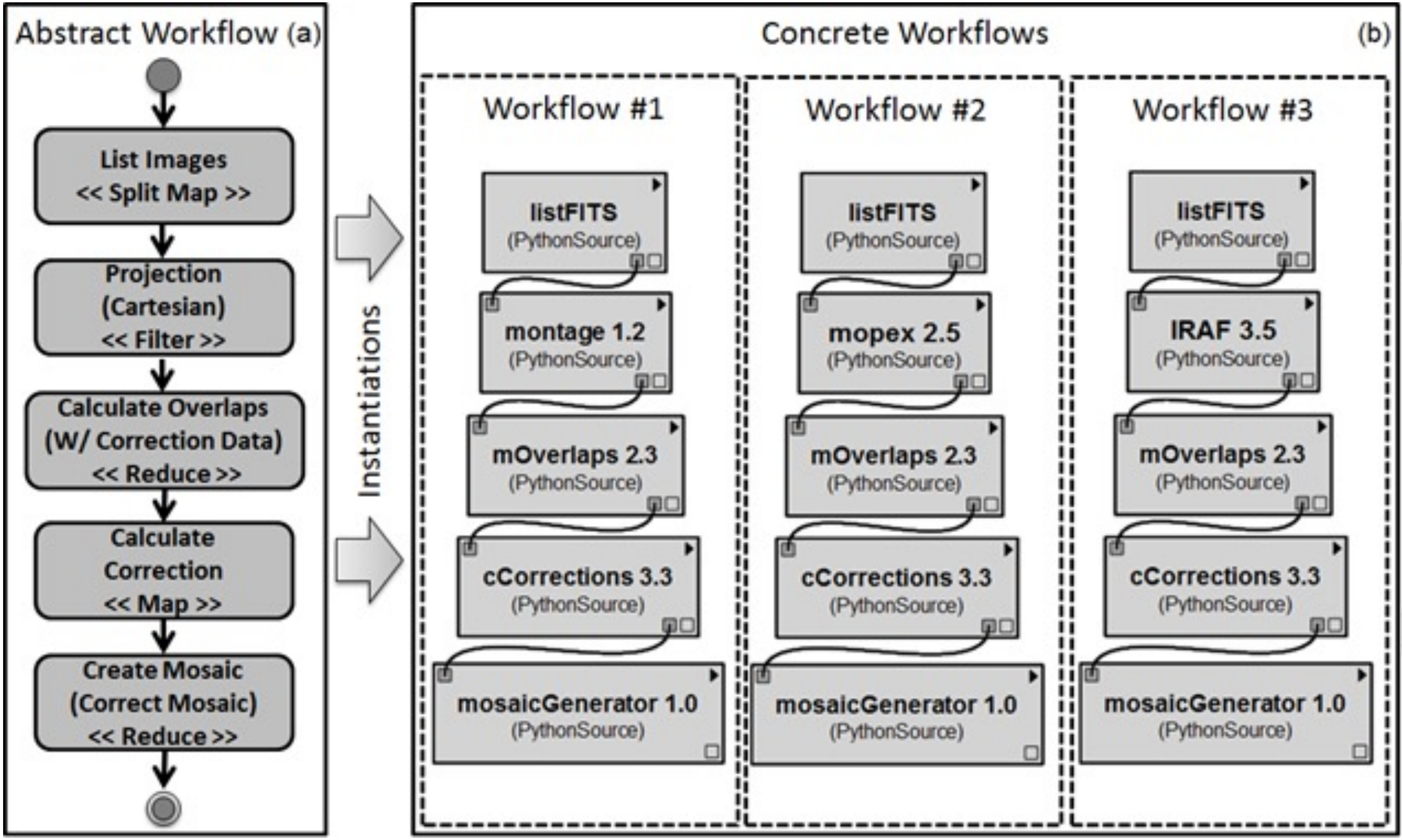
Example



- Activities **vtkStructuredPointsReader** and **vtkCamera** do not depend on other activities data, so they can start executing right away

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

One Abstract to (possibly) Several Concretes

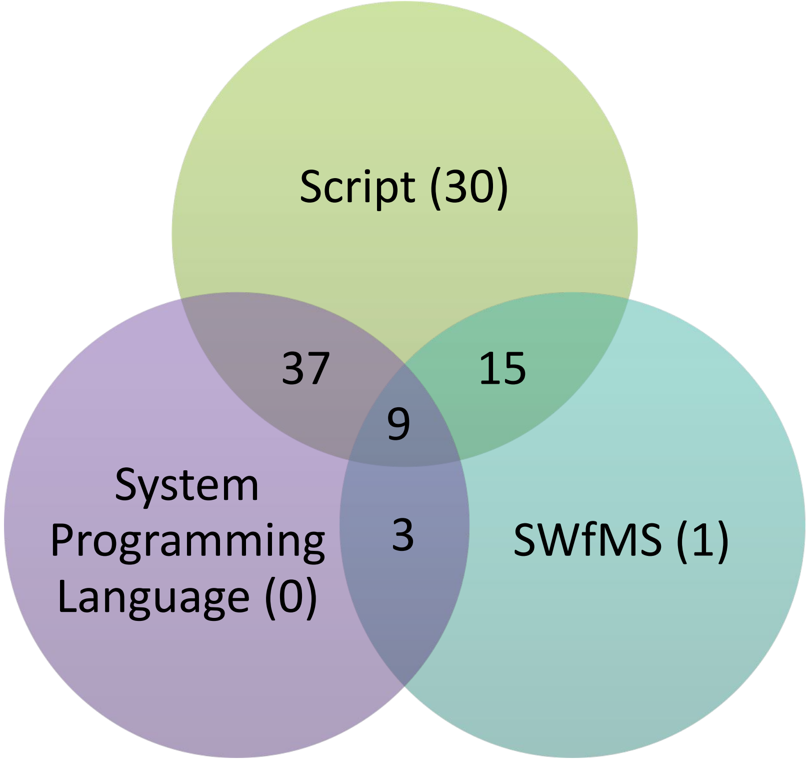


Source: MARINHO, A. Algebraic Experiment Line: an approach to represent scientific experiments based on workflows. PhD Thesis. UFRJ, 2015.

However, lots of people still use scripts

95%

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



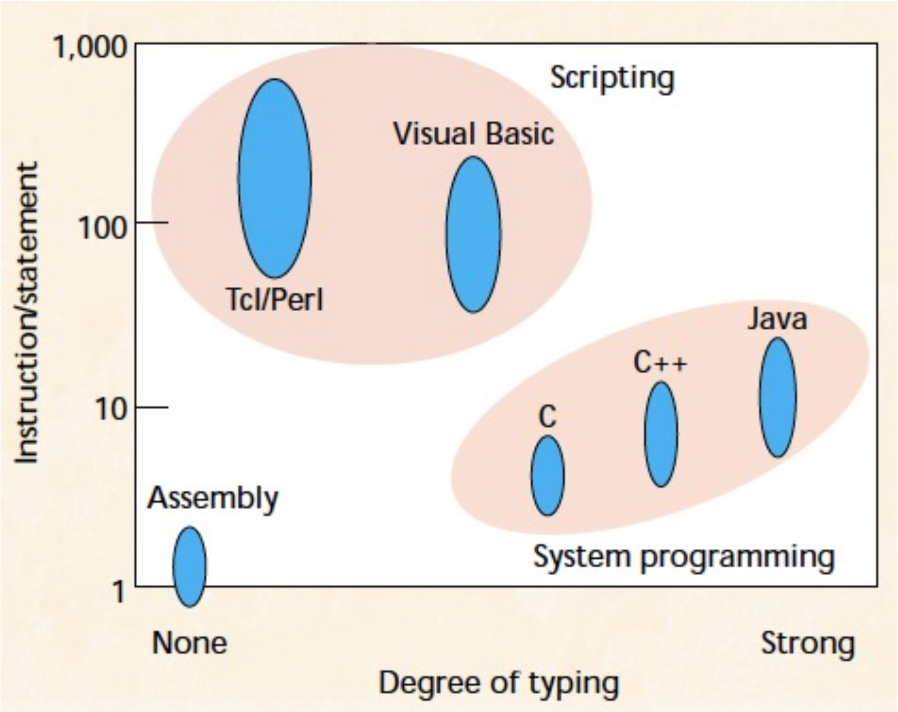
*Survey sent in **2017** to AMC@UvA (Olabarriaga), UFRJ (Mattoso), DATAONE (newsletter), DBBras (mailing list), FIOCRUZ (Davila), USP (Traina), INRIA-Montpellier (Zenith group), LNCC (Ocana), PW 2016 TPC, SciPyLA (Telegram), Software Carpentry (mailing list), U. Nantes (Gaignard), UPENN (Davidson), receiving **120 answers**.

But what exactly are Scripts?

- There is no robust definition in the literature!
- Our to-be-improved definition:
  - "A **script** is a program **conceived for gluing components**, which may have been written in different programming languages" (Leonardo Murta)
- Actually, it does not matter much…
  - "When I see a bird that walks like a duck, swims like a duck and quacks like a duck, I call that bird a duck" (James Whitcomb Riley)

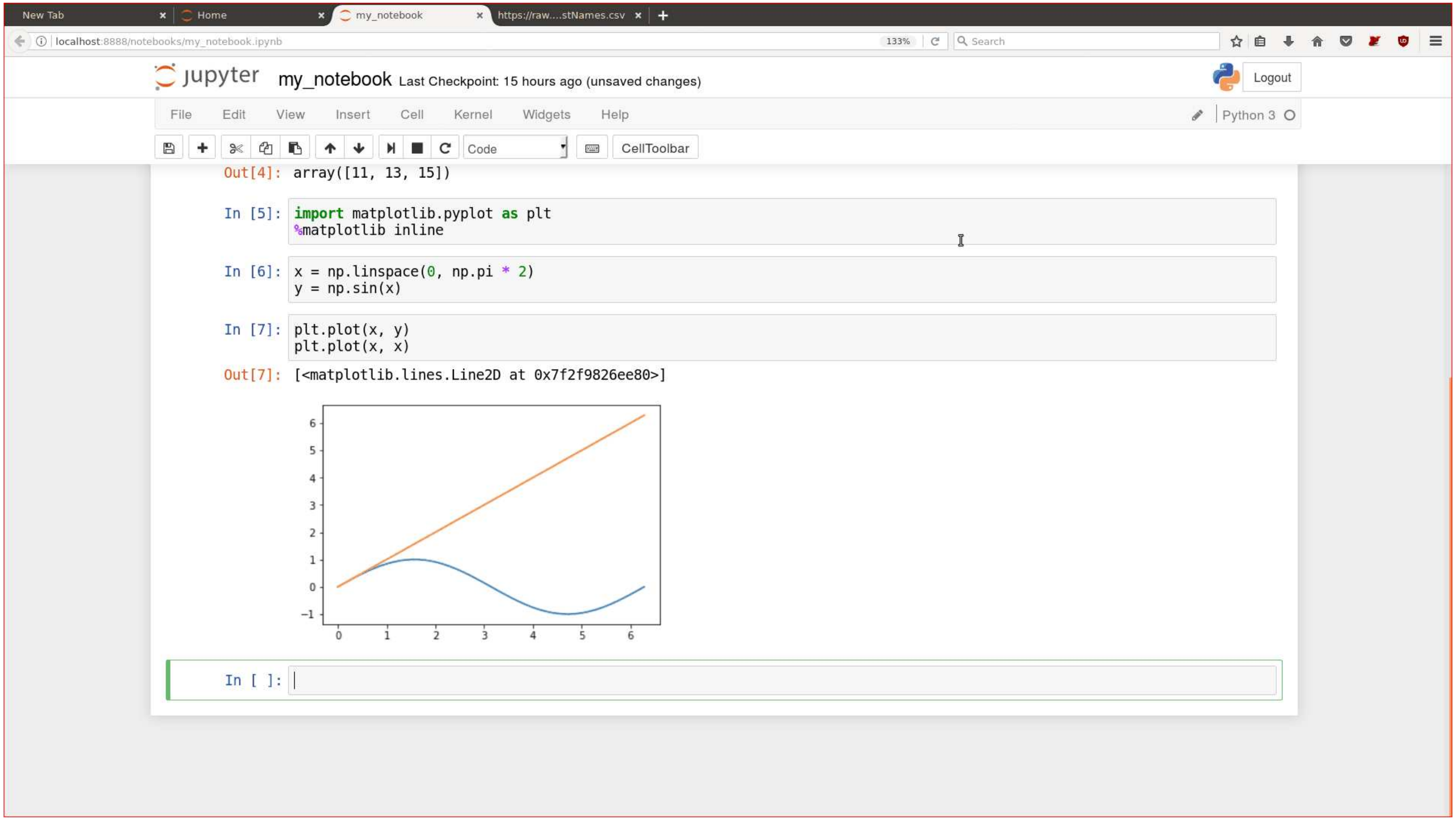
Scripts are high-level programs

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



Ousterhout "Scripting: Higher level programming for the 21st century." Computer 31(3) 1998

Scripts are interactive



New Tab
Home
my_notebook
https://raw....stNames.csv
localhost:8888/notebooks/my_notebook.ipynb
133%
Search
jupyter
my_notebook Last Checkpoint: 15 hours ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Python 3
Code
CellToolbar
6
5
4
3
2
1
0
−1
0
1
2
3
4
5
6

```
Out[4]: array([11, 13, 15])

In [5]: import matplotlib.pyplot as plt
        %matplotlib inline

In [6]: x = np.linspace(0, np.pi * 2)
        y = np.sin(x)

In [7]: plt.plot(x, y)
        plt.plot(x, x)

Out[7]: [<matplotlib.lines.Line2D at 0x7f2f9826ee80>]

In [ ]:
```

http://n-s-f.github.io/2017/03/25/r-to-python.html

Some popular scripting languages

python™
$_
BASH
THE BOURNE-AGAIN SHELL
R
tcl
POWERED
MATLAB®
JS
programming republic of
Perl®

Script

- Execution follows a **control flow** instead of a data flow
  - Commands explicitly define the execution order

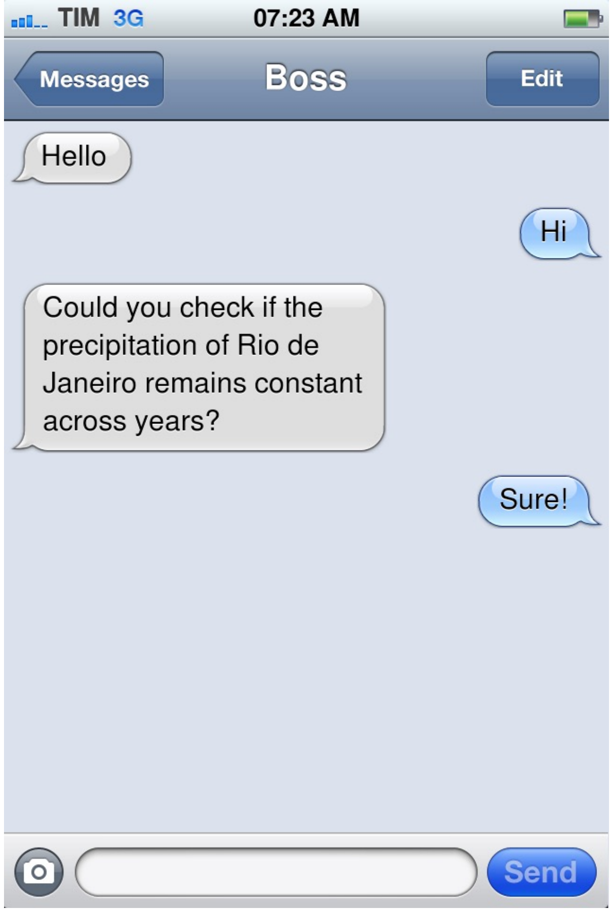
experiment.py

```
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)
```

Running an Experiment

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

New experiment!



Hello

Hi

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

Sure!

http://www.ifaketext.com/

1st iteration – experiment.py

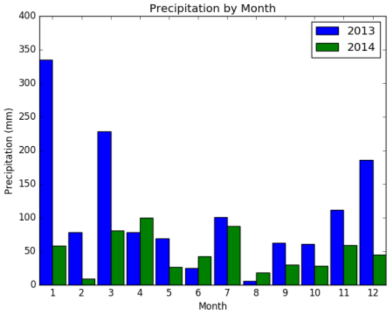
```python
import numpy as np
from precipitation import read, sum_by_month
from precipitation import create_bargraph

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)
```

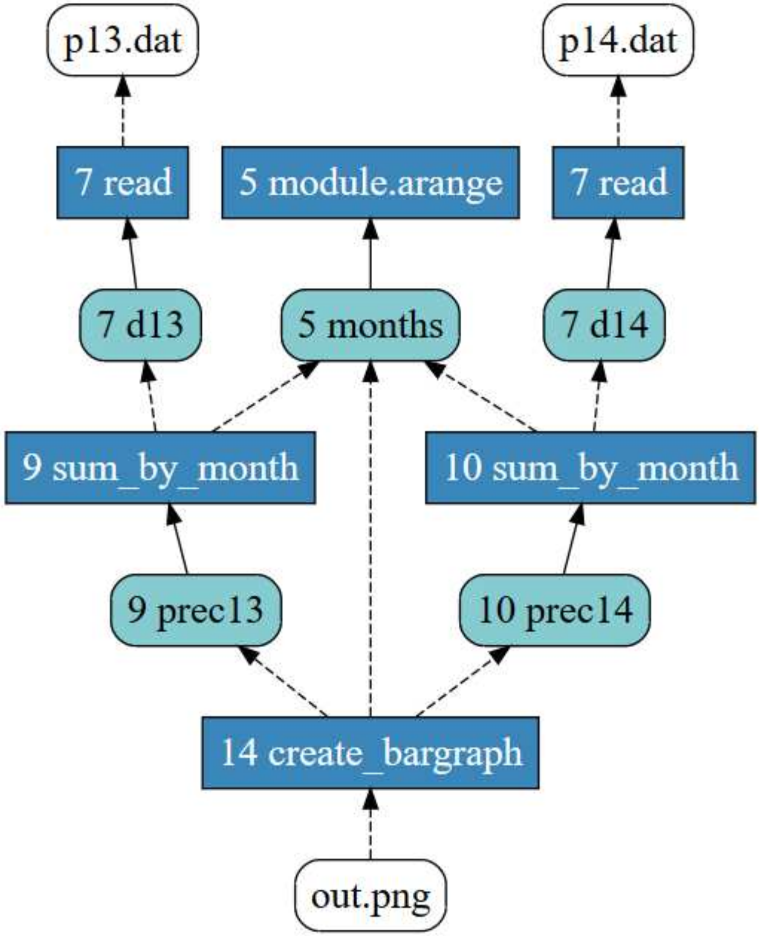
Result and Provenance

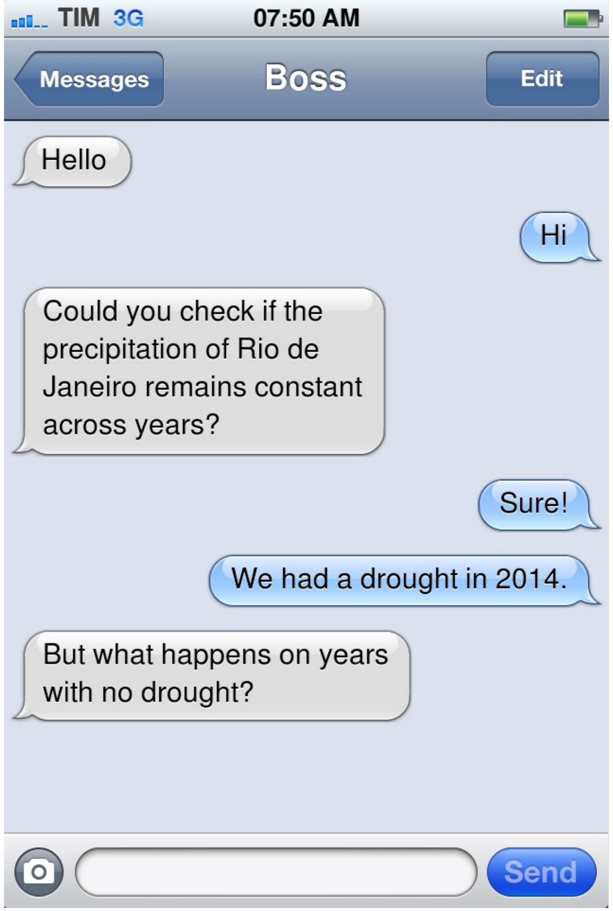
P13.dat

P14.dat

```
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)
```







Hello

Hi

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

Sure!

We had a drought in 2014.

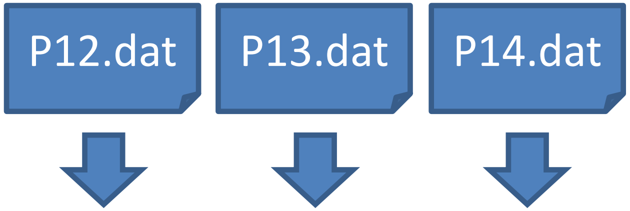
But what happens on years with no drought?

http://www.ifaketext.com/

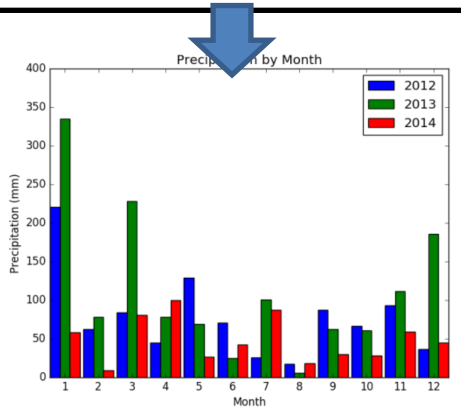
2nd Iteration – experiment.py

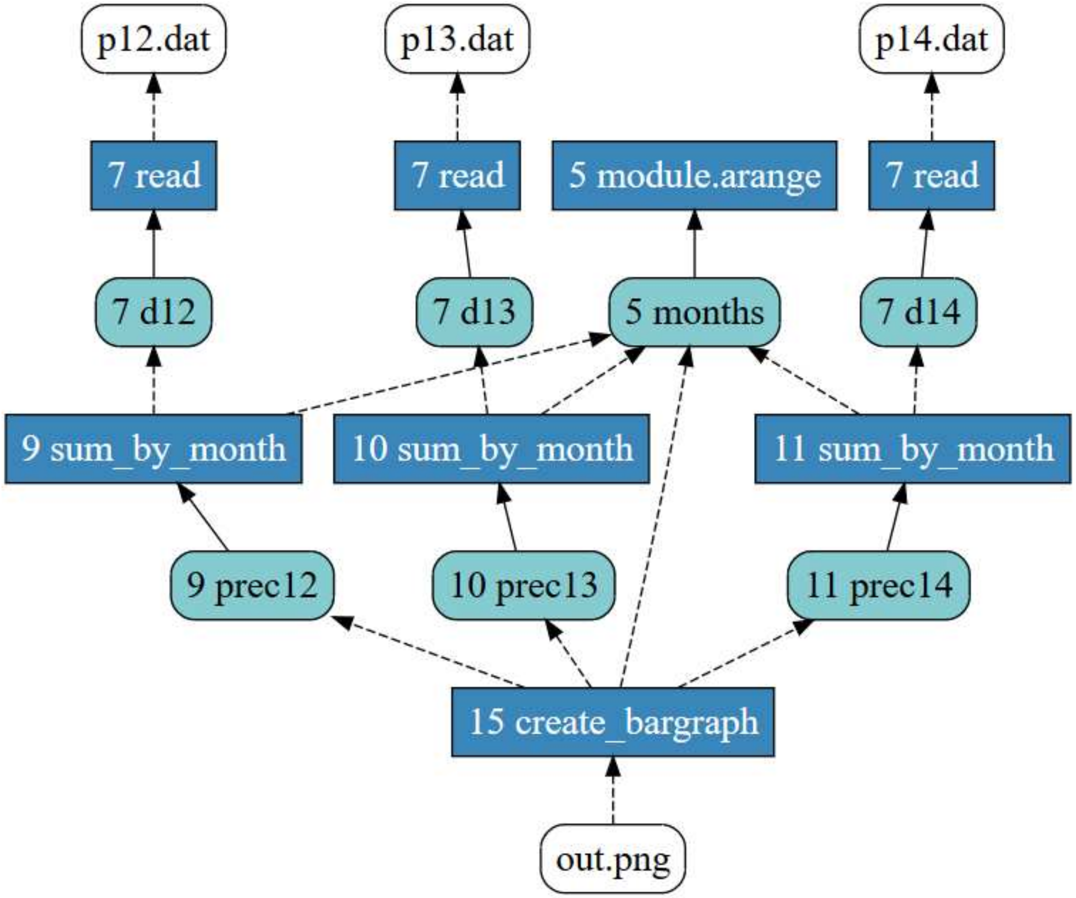
```
import numpy as np
from precipitation import read, sum_by_month
from precipitation import create_bargraph

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)
```

Result and Provenance

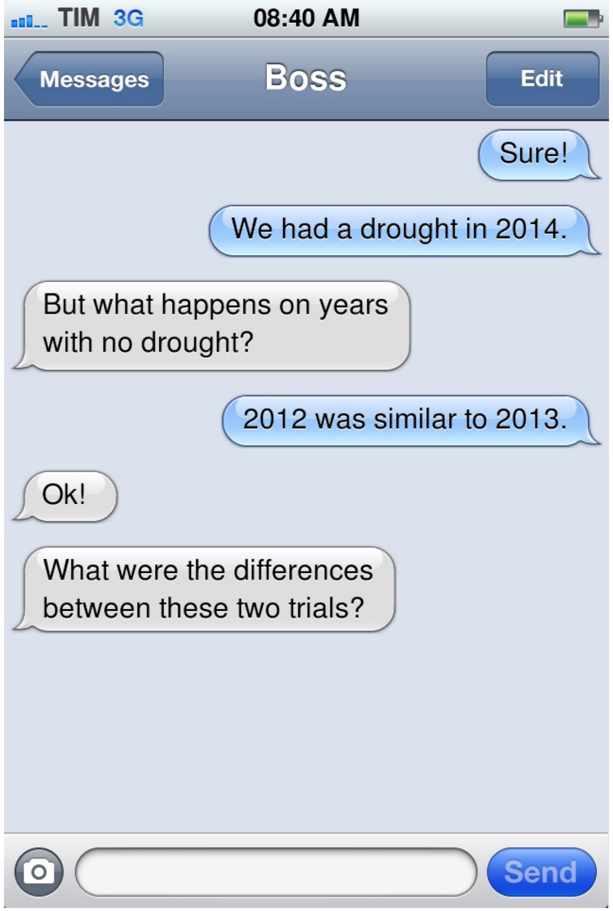


```
import numpy as np
from precipitation import read, sum_by_month
from precipitation import create_bargraph

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)
```





More provenance analyses!



Sure!

We had a drought in 2014.

But what happens on years with no drought?

2012 was similar to 2013.

Ok!

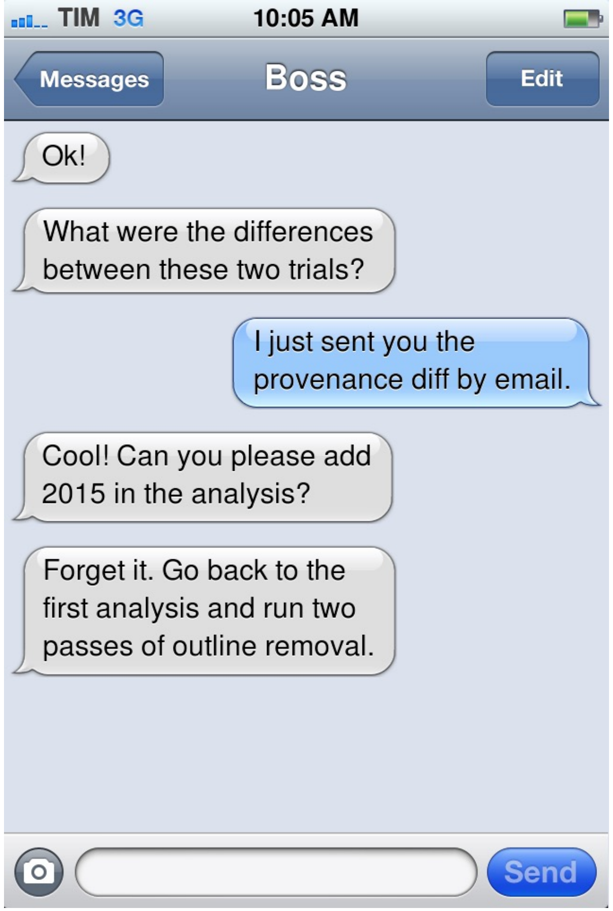
What were the differences between these two trials?

http://www.ifaketext.com/

Textual Diff produced by noWorkflow

```
$ now diff 1 2 -f –brief

[now] trial diff:
  Start changed from 2016-05-30 7:33:26.105716
                  to 2016-05-30 7:55:26.276369
  Finish changed from 2016-05-30 7:34:27.729060
                   to 2016-05-30 7:56:24.863268
  Duration changed from 0:01:01.623344 to 0:00:58.586899
  Code hash changed from a66f3052414673feed5e49812e6940a92bba7679
                      to ff62d0f369315fbc209c39379ccf93437725fa31
  Parent id changed from <None> to 1

[now] Brief file access diff
[Additions]    | [Removals]            | [Changes]
(r) p12.dat    | (wb) out.png (new)    |
(wb) out.png   |                       |
```

After some other requests…



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.

http://www.ifaketext.com/

Restore Trial 1

- Scientists should be able to restore a previous version of the experiment

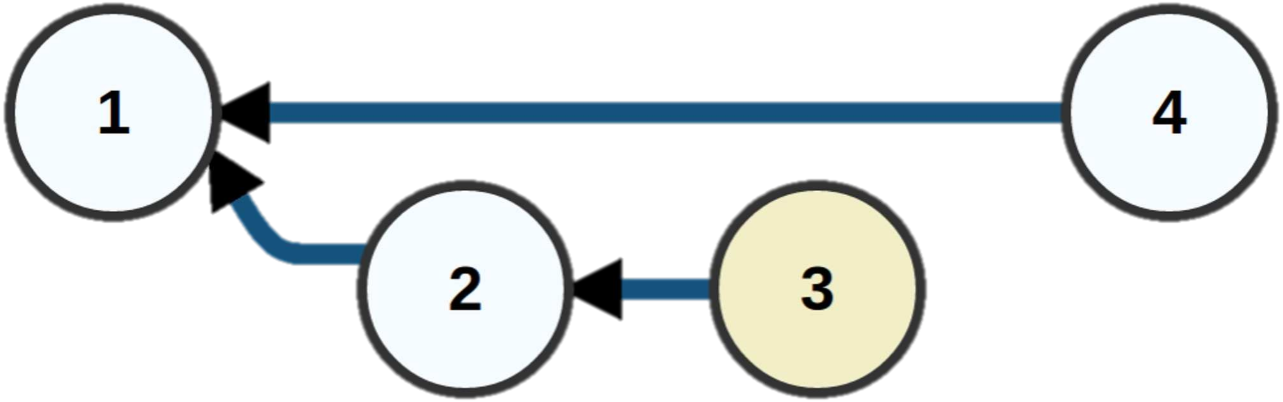
- Example of a command to do that in noWorkflow:

```
$ now restore 1
```

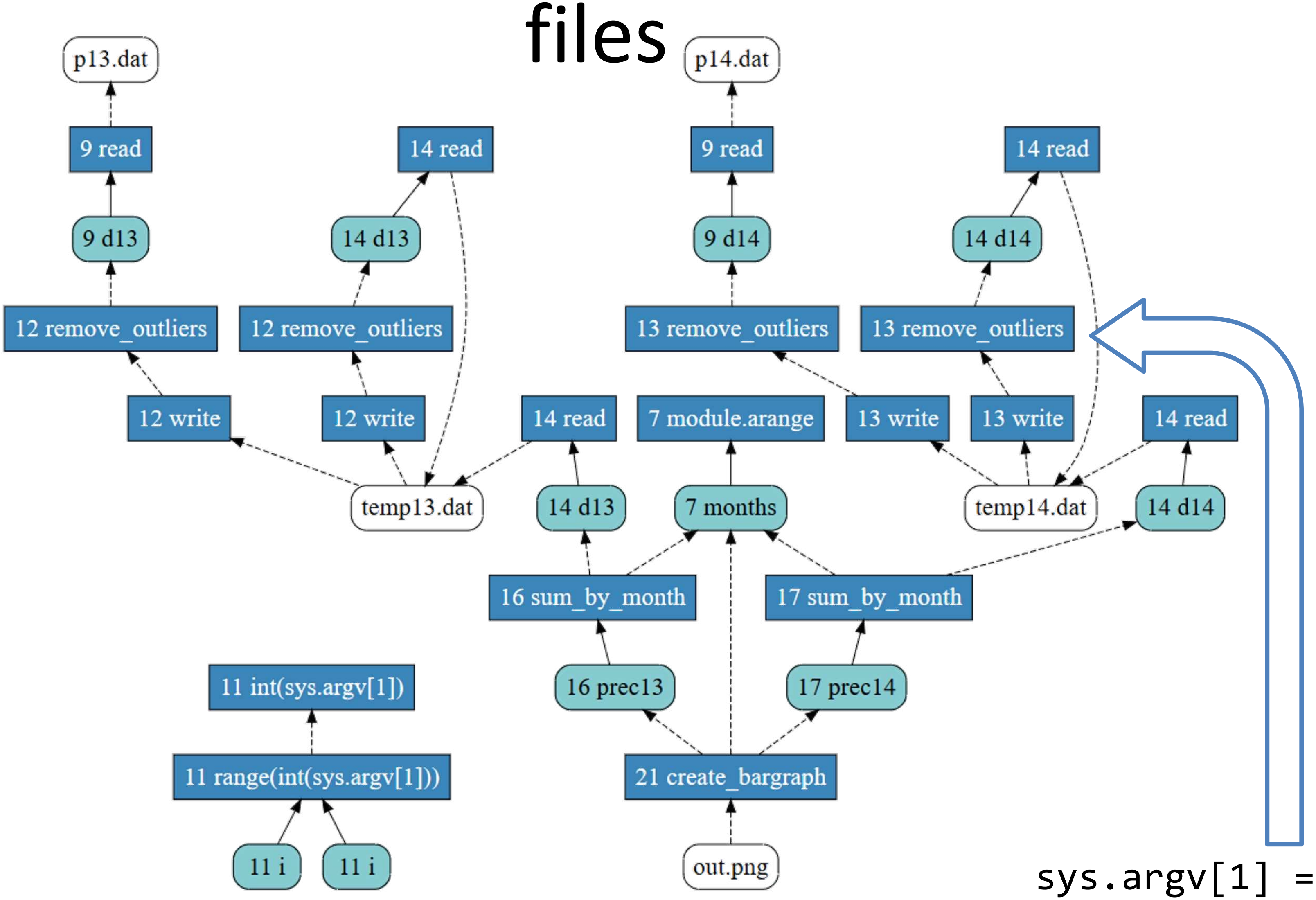
4th iteration – experiment.py

```
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)
```

Trial History



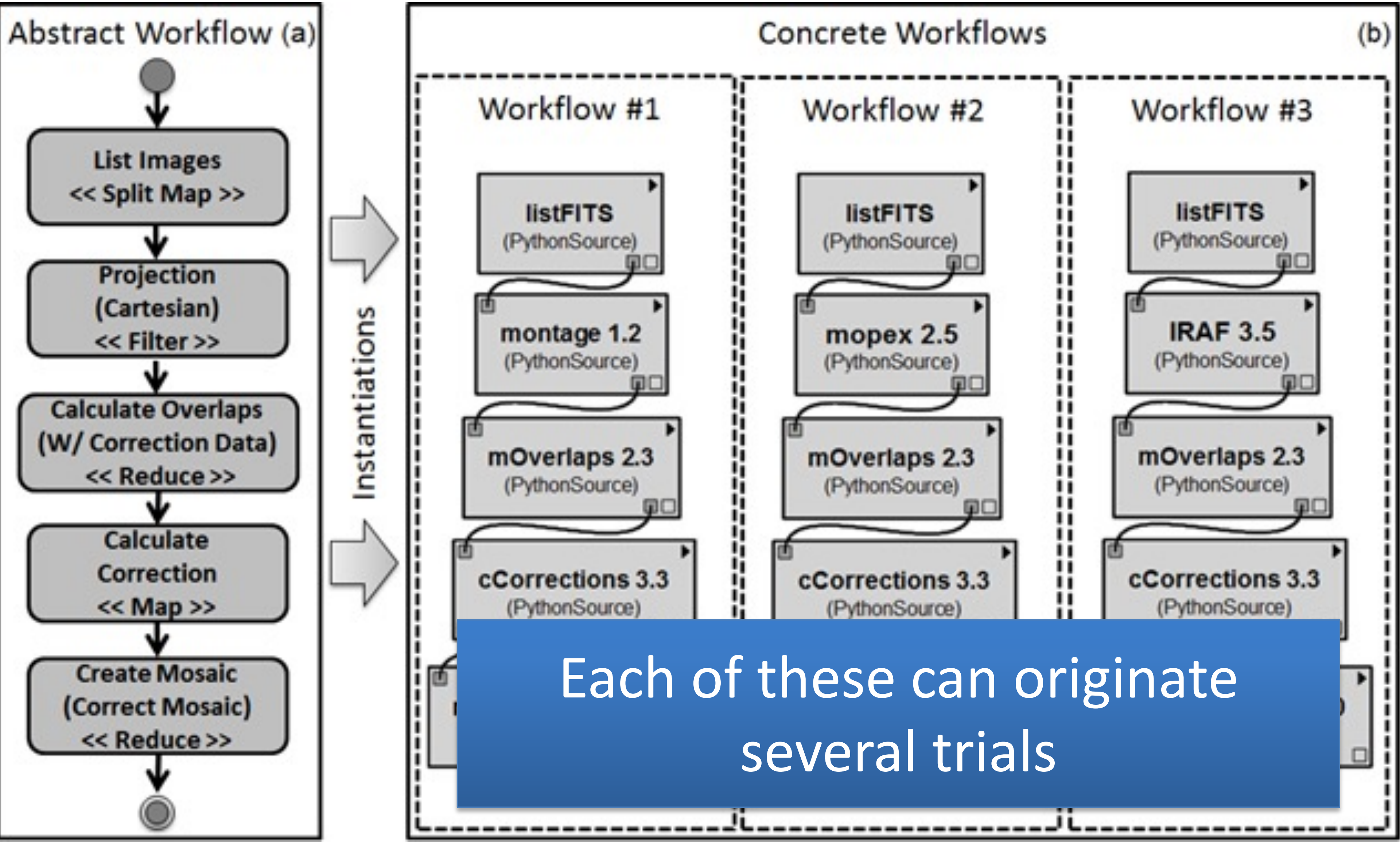
4th iteration – provenance with temp files



sys.argv[1] = 2

This can also be done for workflows...

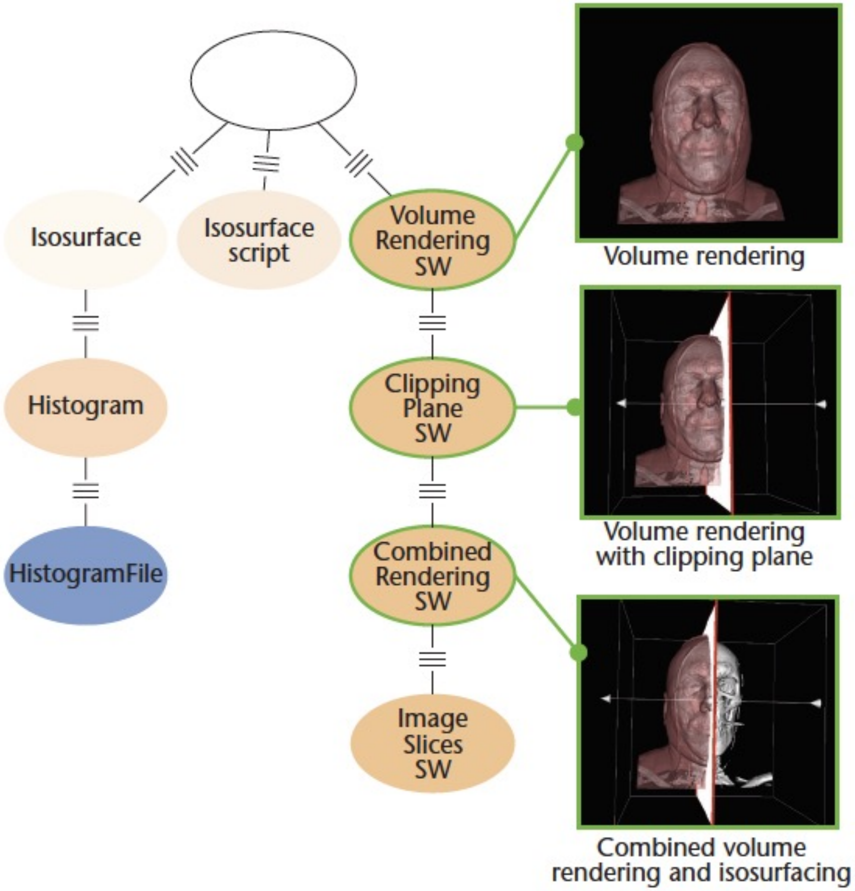
Workflow Trials



Each of these can originate several trials

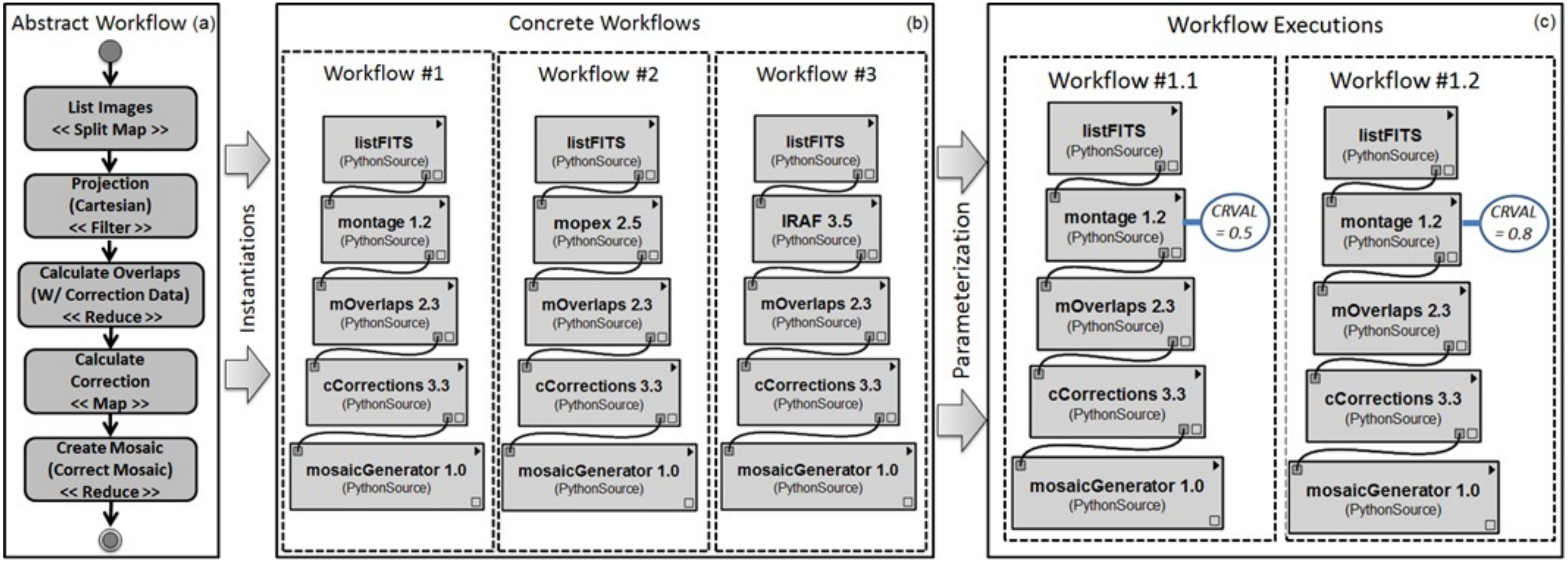
Source: MARINHO, A. Algebraic Experiment Line: an approach to represent scientific experiments based on workflows. PhD Thesis, UFRJ, 2015.

History Graph (VisTrails)



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

Trials in Workflows



Abstract Workflow (a)
List Images << Split Map >>
Projection (Cartesian) << Filter >>
Calculate Overlaps (W/ Correction Data) << Reduce >>
Calculate Correction << Map >>
Create Mosaic (Correct Mosaic) << Reduce >>
Instantiations
Concrete Workflows (b)
Workflow #1
listFITS (PythonSource)
montage 1.2 (PythonSource)
mOverlaps 2.3 (PythonSource)
cCorrections 3.3 (PythonSource)
mosaicGenerator 1.0 (PythonSource)
Workflow #2
listFITS (PythonSource)
mopex 2.5 (PythonSource)
mOverlaps 2.3 (PythonSource)
cCorrections 3.3 (PythonSource)
mosaicGenerator 1.0 (PythonSource)
Workflow #3
listFITS (PythonSource)
IRAF 3.5 (PythonSource)
mOverlaps 2.3 (PythonSource)
cCorrections 3.3 (PythonSource)
mosaicGenerator 1.0 (PythonSource)
Parameterization
Workflow Executions (c)
Workflow #1.1
listFITS (PythonSource)
montage 1.2 (PythonSource)
CRVAL = 0.5
mOverlaps 2.3 (PythonSource)
cCorrections 3.3 (PythonSource)
mosaicGenerator 1.0 (PythonSource)
Workflow #1.2
listFITS (PythonSource)
montage 1.2 (PythonSource)
CRVAL = 0.8
mOverlaps 2.3 (PythonSource)
cCorrections 3.3 (PythonSource)
mosaicGenerator 1.0 (PythonSource)

Several ways to go from abstract to concrete

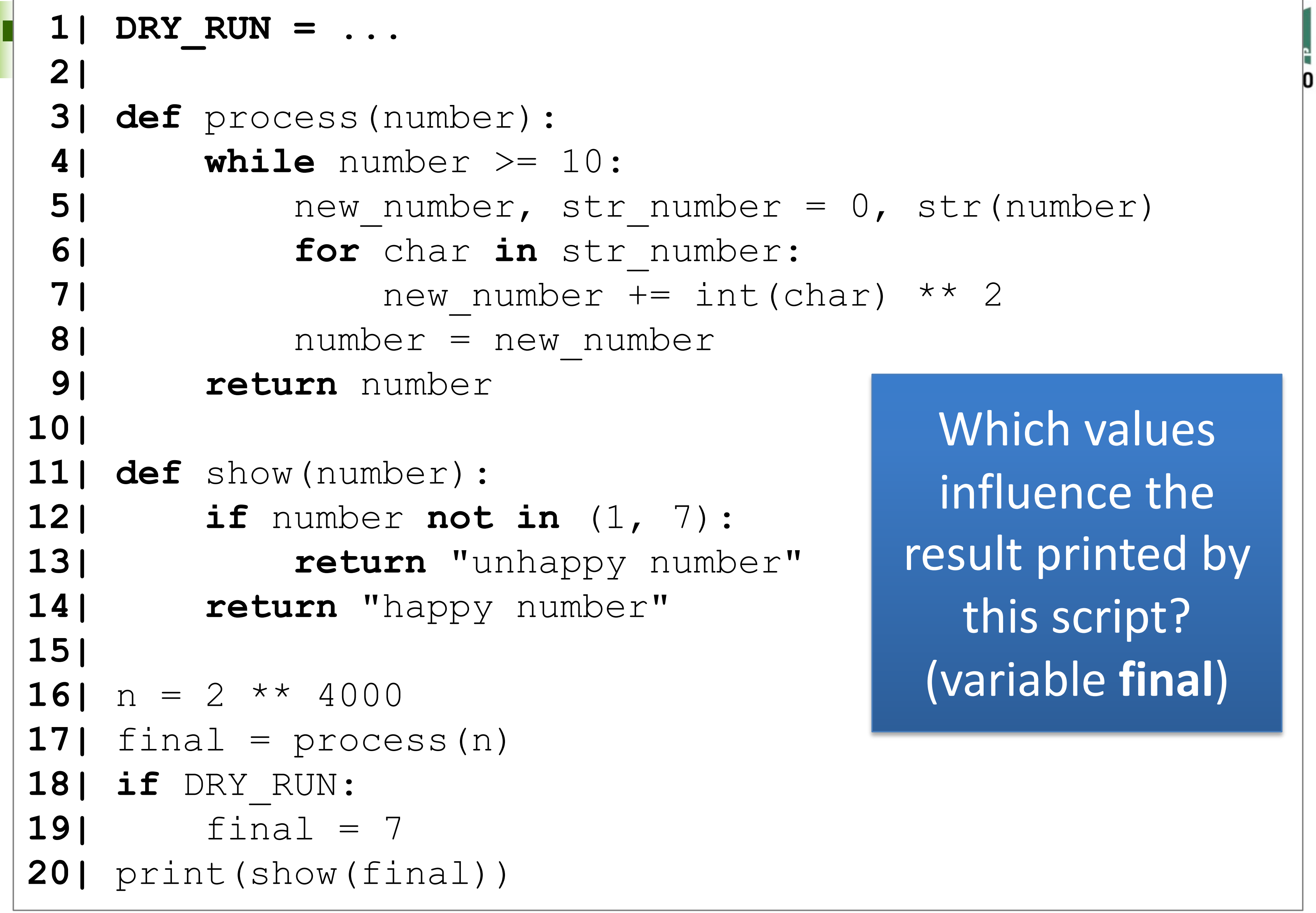
- When using scripts, there are several ways to go from abstract to concrete workflows
  - Activities are implemented one after the other in the script (no functions)
  - Activities are mapped into functions (each activity becomes one or more functions)

Black Box X White Box

- In Workflow systems, activities 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 (in the case of noWorkflow), it is a white box

Black Box X White Box

- Black boxes have implications in provenance analysis



```python
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**)

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

```python
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**

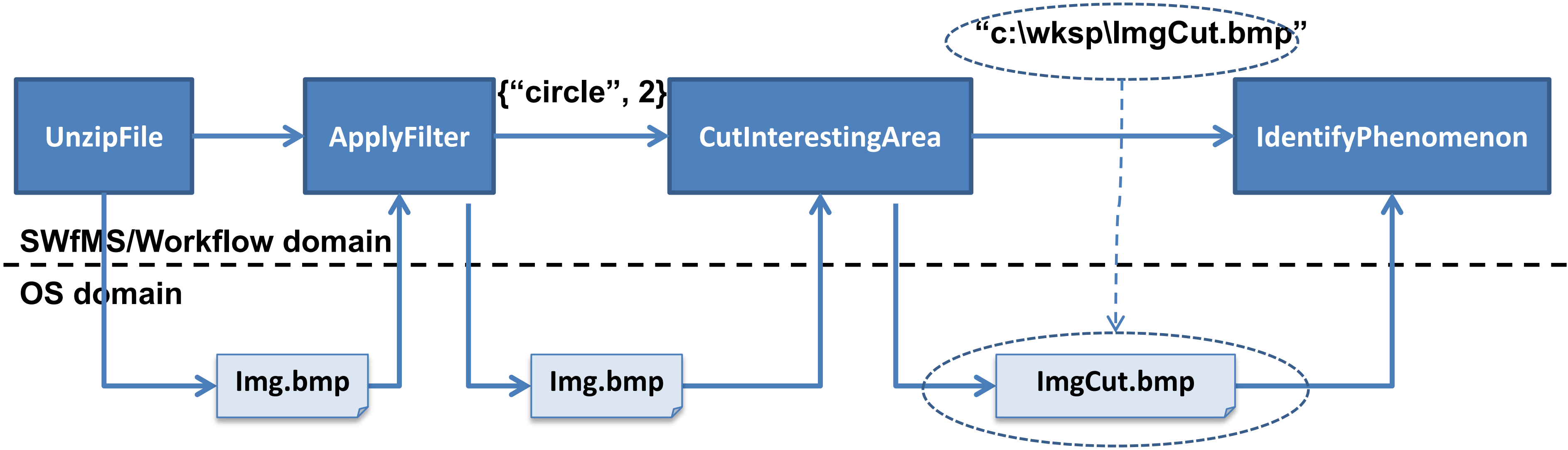
If not, then **final** also depends on **n**

Source: 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

Implicit Provenance



Sources:
Neves et al., 2017. Managing Provenance of Implicit Data Flows in Scientific Experiments.
Marinho et al., 2011. Challenges in managing implicit and abstract provenance data: experiences with ProvManager.

Implicit Provenance

- OS-Based approaches are able to capture this kind of provenance
- Other approaches need special components to handle it (e.g. PROVMONITOR)

Neves et al., 2017. Managing Provenance of Implicit Data Flows in Scientific Experiments

Overview of Existing Systems

- Workflow Management Systems
- Provenance Management Systems for Scritps

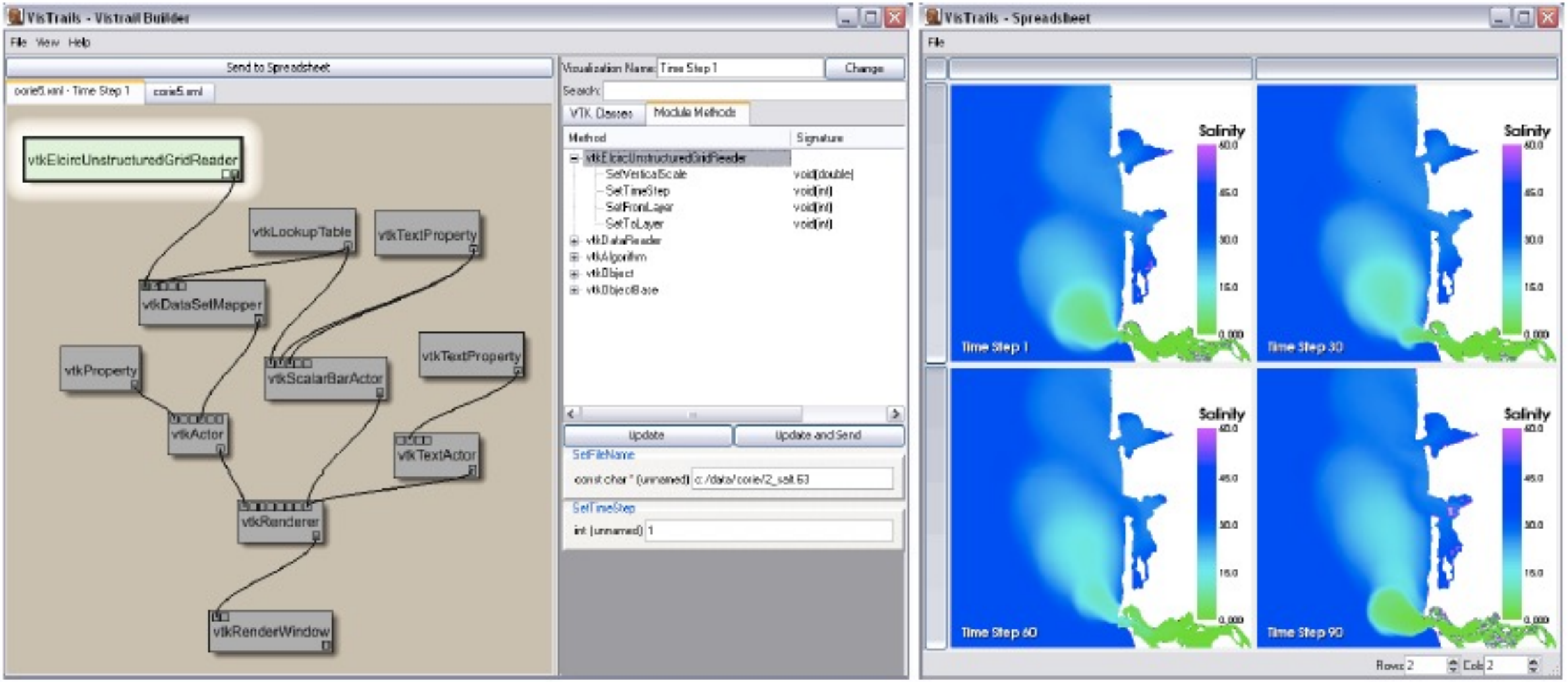
Workflow Management Systems

Among many others…

VisTrails

- 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

VisTrails



Visualization Name: Time Step 1

Method	Signature
vtkElcircUnstructuredGridReader	
SetVerticalScale	void(double)
SetTimeStep	void(int)
SetFromLayer	void(int)
SetToLayer	void(int)
vtkDataReader	
vtkAlgorithm	
vtkObject	
vtkObjectBase	

SetFileName

const char * (unnamed) c:/data/corie/2_salt.63

SetTimeStep

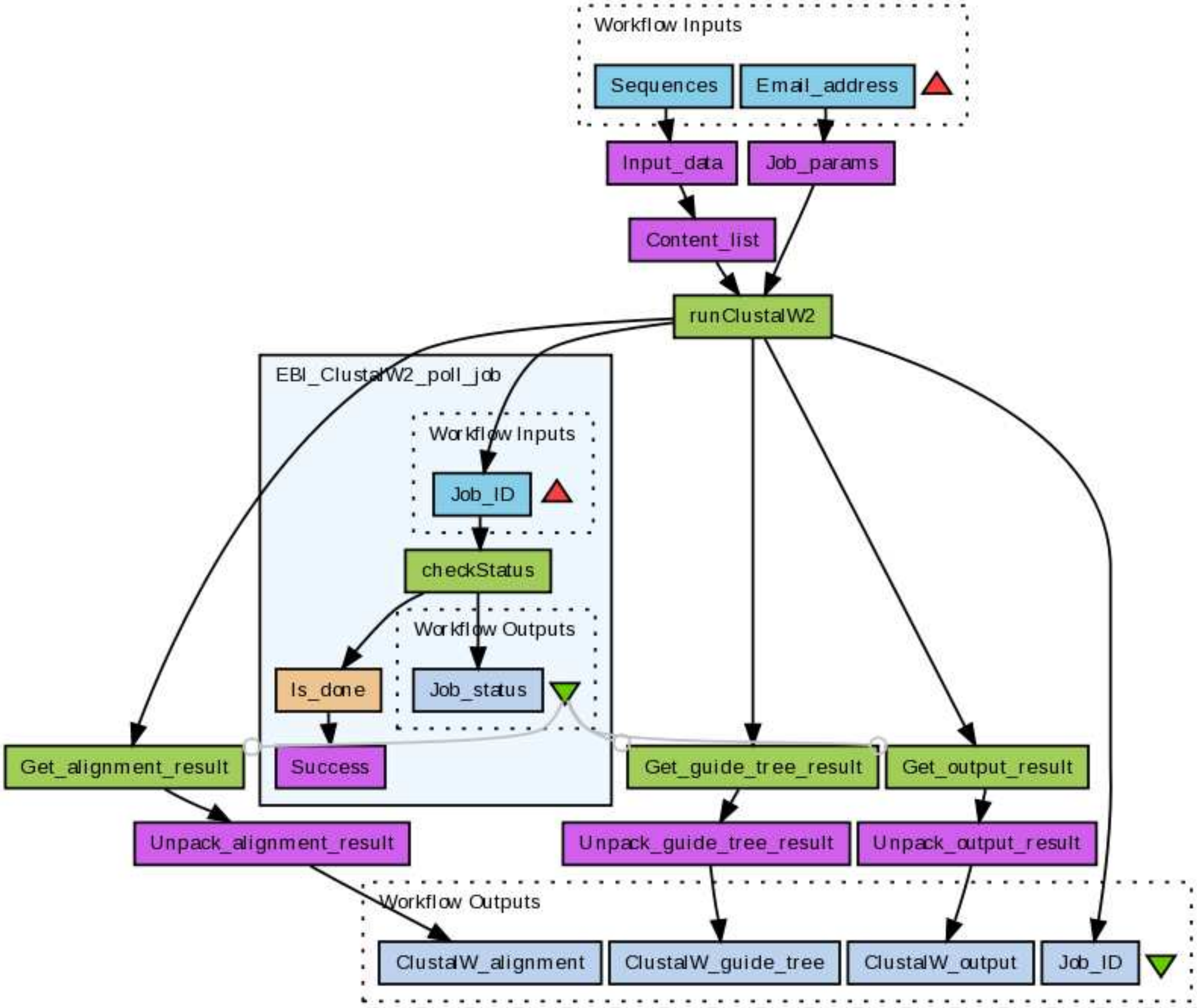
int (unnamed) 1

Rows: 2 Cols: 2

Taverna

- Focus on Bioinformatics
- Several ready-to-use bioinformatics services
- Drag and Drop graphical interface for workflow composition

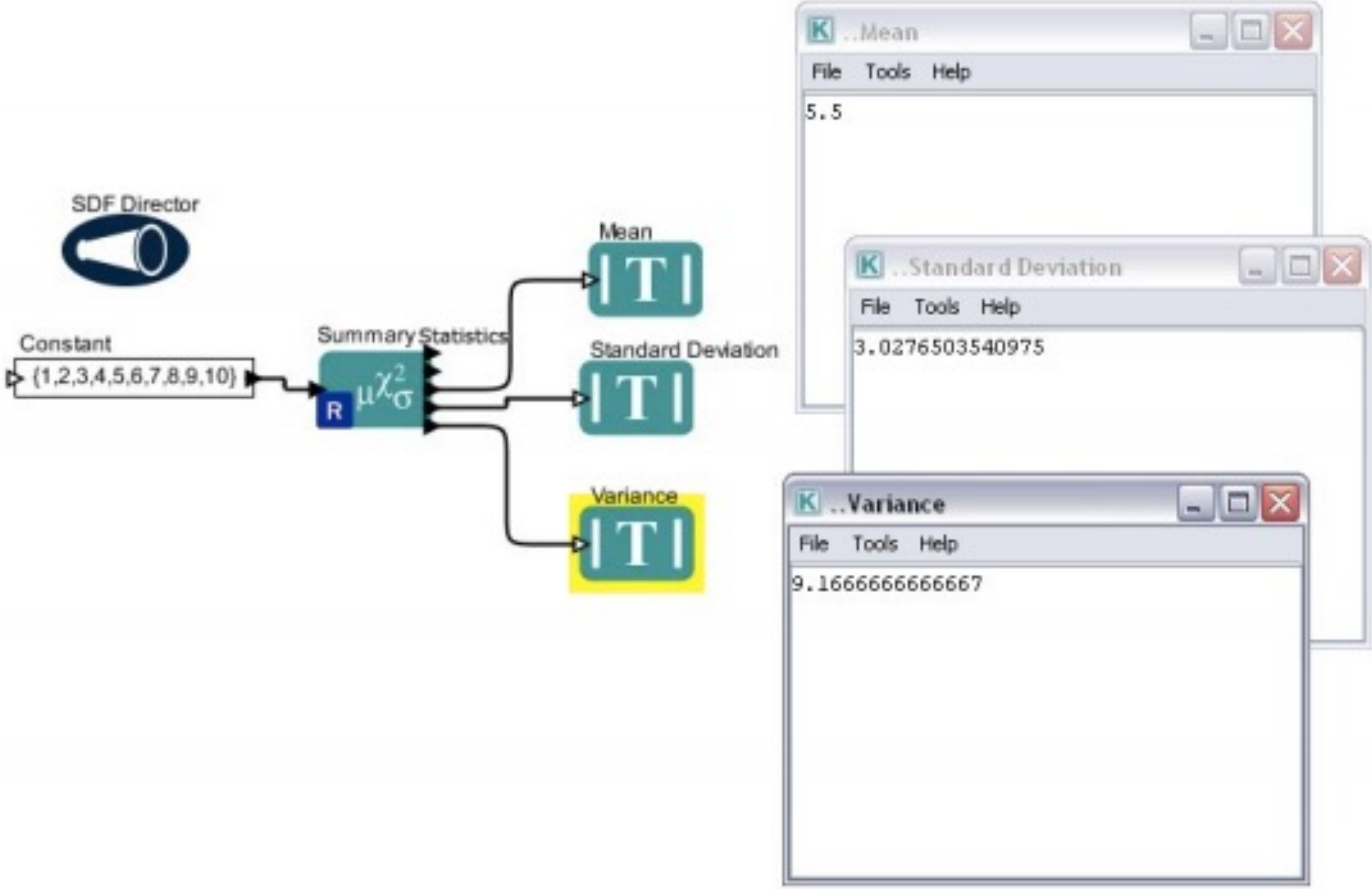
http://www.taverna.org.uk/


Workflow Inputs
Sequences
Email_address
Input_data
Job_params
Content_list
runClustalW2
EBI_ClustalW2_poll_job
Workflow Inputs
Job_ID
checkStatus
Workflow Outputs
Is_done
Job_status
Success
Get_alignment_result
Get_guide_tree_result
Get_output_result
Unpack_alignment_result
Unpack_guide_tree_result
Unpack_output_result
Workflow Outputs
ClustalW_alignment
ClustalW_guide_tree
ClustalW_output
Job_ID

Kepler

- Drag and Drop graphical interface for workflow composition
- Different actors that rules how the workflow is executed – Kepler workflows are not DAG

https://kepler-project.org/



```
5.5
```

```
3.0276503540975
```

```
9.1666666666667
```

Swift, SciCumulus and Pegasus

- Focus on High Performance
- Workflows are specified in XML (no graphical interface) in SciCumulus and Pegasus
- In Swift, workflows are specified as scripts in a specific language

http://swift-lang.org/main/index.php
https://scicumulusc2.wordpress.com/
https://pegasus.isi.edu/

Provenance Management Systems for Scripts

- noWorkflow
  - captures provenance for Python scripts
- RDataTracker
  - captures provenance for R scripts
- Sumatra
  - captures provenance for Python, R and MatLab scripts

Exercise

- Choose one of the systems presented in today's class and search the Web to find:
  - What is the format in which provenance is stored
  - If they export provenance in the PROV format
  - Post your answer in our class in Google Classroom

Provenance of these slides

- A number of these slides were obtained from a **keynote** at BreSci 2017 presented by **Leonardo Murta** "Provenance Gathering from scripts: challenges and opportunities"

Provenance of these slides

- MARINHO, A. ; WERNER, C. M. L. ; MATTOSO, M. L. Q. ; BRAGANHOLO, V. ; MURTA, L. G. P. . Challenges in managing implicit and abstract provenance data: experiences with ProvManager. In: USENIX Workshop on the Theory and Practice of Provenance (TaPP), 2011, Heraklion, Creta, Grécia, p. 1-6.
- MATTOSO, M. L. Q. ; WERNER, C. M. L. ; TRAVASSOS, G. H. ; BRAGANHOLO, V. ; MURTA, L. G. P. ; OGASAWARA, E. ; OLIVEIRA, D. ; CRUZ, S. ; MARTINHO, W. . Towards Supporting the Life Cycle of Large Scale Scientific Experiments. International Journal of Business Process Integration and Management (Print), v. 5, p. 79-92, 2010.
- NEVES, V. C. ; OLIVEIRA, D. ; OCANA, K. A. ; BRAGANHOLO, V. ; MURTA, L. G. P. . Managing Provenance of Implicit Data Flows in Scientific Experiments. ACM Transactions on Internet Technology, 2017.
- PIMENTEL, J. F. N. ; FREIRE, J. ; BRAGANHOLO, V. ; MURTA, L. G. P. . Tracking and Analyzing the Evolution of Provenance from Scripts. In: International Provenance and Annotation Workshop (IPAW), 2016, Washington, D.C., v. 9672. p. 16-28.
- PIMENTEL, J. F. N. ; FREIRE, J. ; MURTA, L. G. P. ; BRAGANHOLO, V. . Fine-grained Provenance Collection over Scripts Through Program Slicing. In: International Provenance and Annotation Workshop (IPAW), 2016, Washington D.C., v. 9672. p. 199-203.