

# ABM-BaF

## Tools: the Swarm protocol and a possible new implementation in Python

Pietro TERNA, University of Torino and ISI, [terna@econ.unito.it](mailto:terna@econ.unito.it),

<http://web.econ.unito.it/terna>

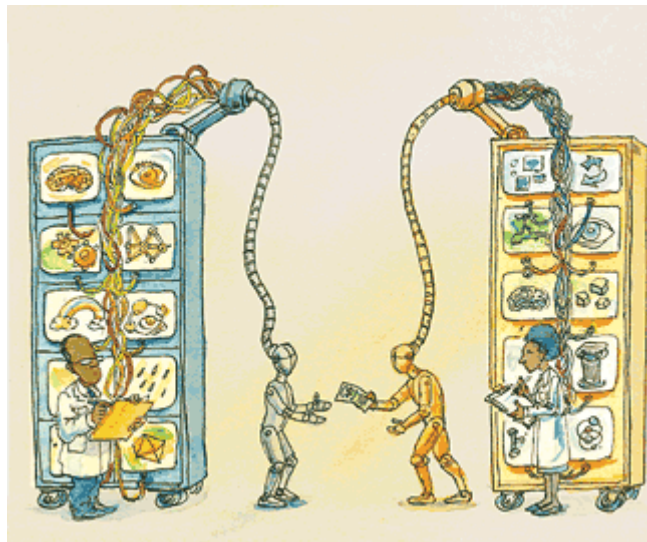
---

# A general structure for agent-based simulation models

---

Social simulation as a computer based way  
to execute complex mental experiments,  
but also as a way to represent the complexity of real world

**simulation = agent-based models**



---

# Building models: three ways

---

## Three different symbol systems:

- verbal argumentations
- mathematics
- computer simulation (agent based)

---

# How to use agents in simulation models: a radical view

---

The radical characterization of an ABM must be found

- (1) into the possibility of real – direct or indirect – interaction amid the agents,
- (2) instead of modeling that interaction in a simplified way, with aggregated simultaneous equations

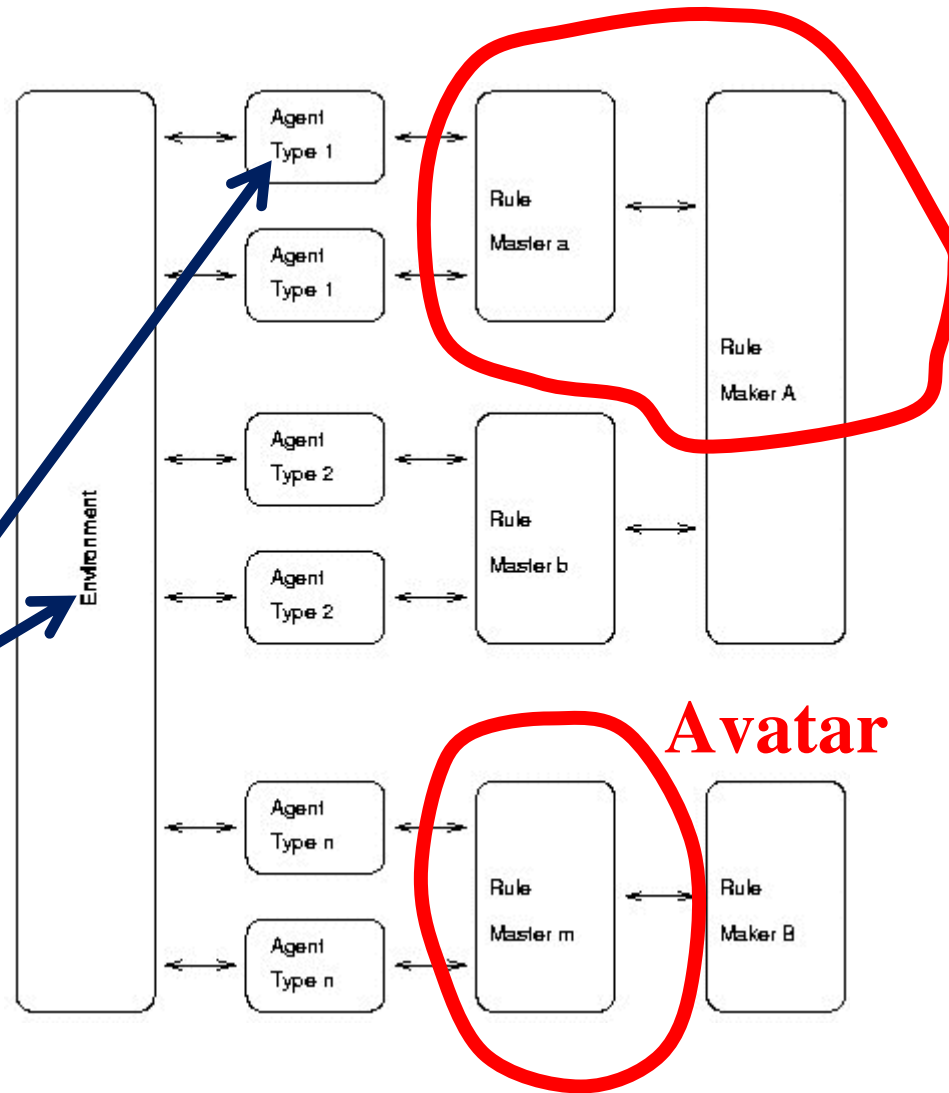
To build (1) type models we need sophisticated tools,  
but also simple and transparent

---

# Agent based simulation and real world representation

---

**Microstructures,  
mainly related to  
time and  
parallelism**



**Fixed  
rules**

**NN**

**CS**

**GA**

**Reinforcement  
learning**

**Avatar**

<http://web.econ.unito.it/terna/ct-era/ct-era.html>

---

# A dictionary

---



A dictionary, from  
Conte R, Edmonds B, Moss S., Sawyer R.K., Sociology and Social Theory in Agent  
Based Social Simulation: A Symposium  
Computational & Mathematical Organization Theory 7, 183-205,2001

- “1. The purpose of Agent Based Social Simulation (ABSS) is to analyse the properties of social systems defined by dense patterns of interaction among autonomous, cognitive individuals.
2. The same modelling techniques that are intended to represent real social systems can also represent software systems such as the Internet and large federated data bases populated by intelligent information agents or, indeed, any other large, complex multi agent system. Multi agent based simulations (MABS) of such systems share the techniques of ABSS.”

My note:

- we use frequently the name of Agent Based Model (ABM) or Agent Computational Economics (ACE) instead of ABSS;
- in computer science the attention is devoted to Multi Agent Systems, MAS; adding “simulation” we have MABS and, in some way, ABSS.

---

# Tools

---

**Swarm**, <http://www.swarm.org>

**SLAPP**, Swarm-Like Agent Protocol in Python, temporary at  
<http://eco83.econ.unito.it/terna/slapp> ; Python at [www.python.org](http://www.python.org)

**JAS**, <http://jaslibrary.sourceforge.net/>

**Ascape**, <http://www.brook.edu/dynamics/models/ascape/>

**Repast**, <http://repast.sourceforge.net/>

**StarLogo**, <http://education.mit.edu/starlogo/>

**StarLogo TNG**, <http://education.mit.edu/starlogo-tng/>

**NetLogo**, <http://ccl.northwestern.edu/netlogo/>

**FLAME**, <https://trac.flame.ac.uk/wiki>

**MetaABM**, <http://www.metascapeabm.com/>

**SDML** (based upon SmallTalk, as a declarative programming tool):  
<http://www.cpm.mmu.ac.uk/sdml/>

See also **ABLE**, <http://www.research.ibm.com/able/>

**JADE**, <http://jade.tilab.com/>

or **DAML**, [www.daml.org](http://www.daml.org)

Also useful in  
didactical  
perspective

nearly  
videogames

---

Why a new tool and why **SLAPP** (Swarm-Like Agent Based Protocol in Python) as a preferred tool?

---

- For didactical reasons, applying a such rigorous and simple object oriented language as Python
- To build models upon transparent code: Python does not have hidden parts or feature coming from magic, it has no obscure libraries
- **To take advantage of the openness of Python**
- **To apply easily the SWARM protocol**

## The openness of Python ([www.python.org](http://www.python.org))

- ... going from Python to R  
(R is at <http://cran.r-project.org/> ; rpy library is at <http://rpy.sourceforge.net/>)
- ... going from OpenOffice (Calc, Writer, ...) to Python and viceversa (via the Python-UNO bridge, incorporated in OOO)
- ... doing symbolic calculations in Python (via <http://code.google.com/p/sympy/>)
- ... doing declarative programming with PyLog, a Prolog implementation in Python (<http://christophe.delord.free.fr/pylog/index.html>)
- ... using Social Network Analysis from Python; examples:
  - Igraph library <http://cneurocv.s.rmki.kfki.hu/igraph/>
  - libsna <http://www.libsna.org/>
  - pySNA <http://www.menslibera.com.tr/pysna/>
- ... building videogames in Python, with <http://www.pygame.org>
-

## The SWARM protocol

What's SLAPP: basically a **demonstration that we can easily implement the Swarm protocol** [Minar, N., R. Burkhart, C. Langton, and M. Askenazi (1996), *The Swarm simulation system: A toolkit for building multi-agent simulations*. Working Paper 96-06-042, Santa Fe Institute, Santa Fe (\*)] **in Python**

(\*) <http://www.swarm.org/images/b/bb/MinarEtAl96.pdf>

Swarm key points (quoting from that paper):

- *Swarm defines a structure for simulations, a framework within which models are built.*
- *The core commitment is to a discrete-event simulation of multiple agents using an object-oriented representation.*
- *To these basic choices Swarm adds the concept of the "swarm," a collection of agents with a schedule of activity.*



## The SWARM protocol

An absolutely clear and rigorous application of the SWARM protocol is contained in the original SimpleBug tutorial (1996?) with ObjectiveC code and text by Chris Langton & Swarm development team (Santa Fe Institute), on line at <http://ftp.swarm.org/pub/swarm/apps/objc/sdg/swarmapps-objc-2.2-3.tar.gz> (into the folder “tutorial”, with the text reported into the README files in the tutorial folder and in the internal subfolders)

The same has also been adapted to Java by Charles J. Staelin (*jSIMPLEBUG, a Swarm tutorial for Java*, 2000), at <http://www.cse.nd.edu/courses/cse498j/www/Resources/jsimplebug11.pdf> (text) or <http://eco83.econ.unito.it/swarm/materiale/jtutorial/JavaTutorial.zip> (text and code)

At <http://eco83.econ.unito.it/terna/slapp> you can find the same structure of files, but now implementing the SWARM protocol using Python

**The SWARM protocol as *lingua franca* in agent based simulation models**

---

## Have a look to Swarm basics

---

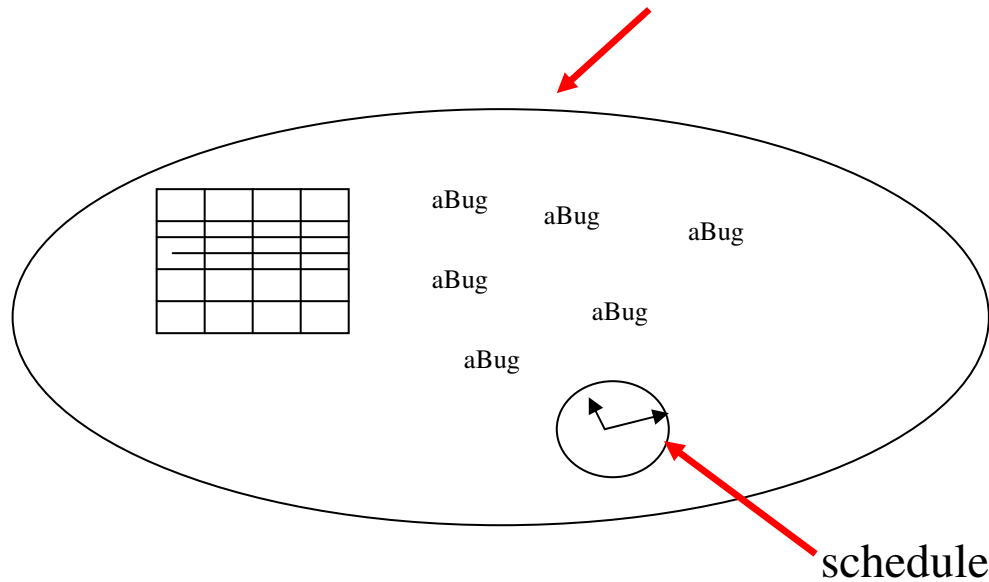
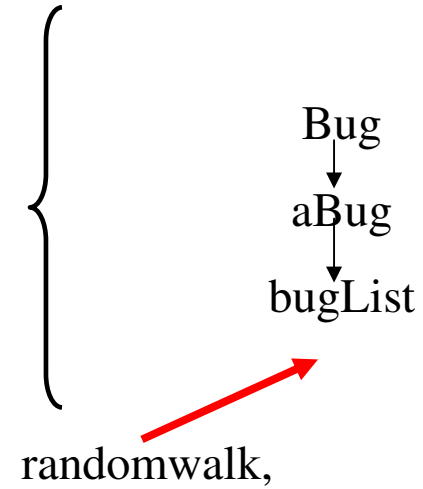
# Swarm = a library of functions and a **protocol**

modelSwarm

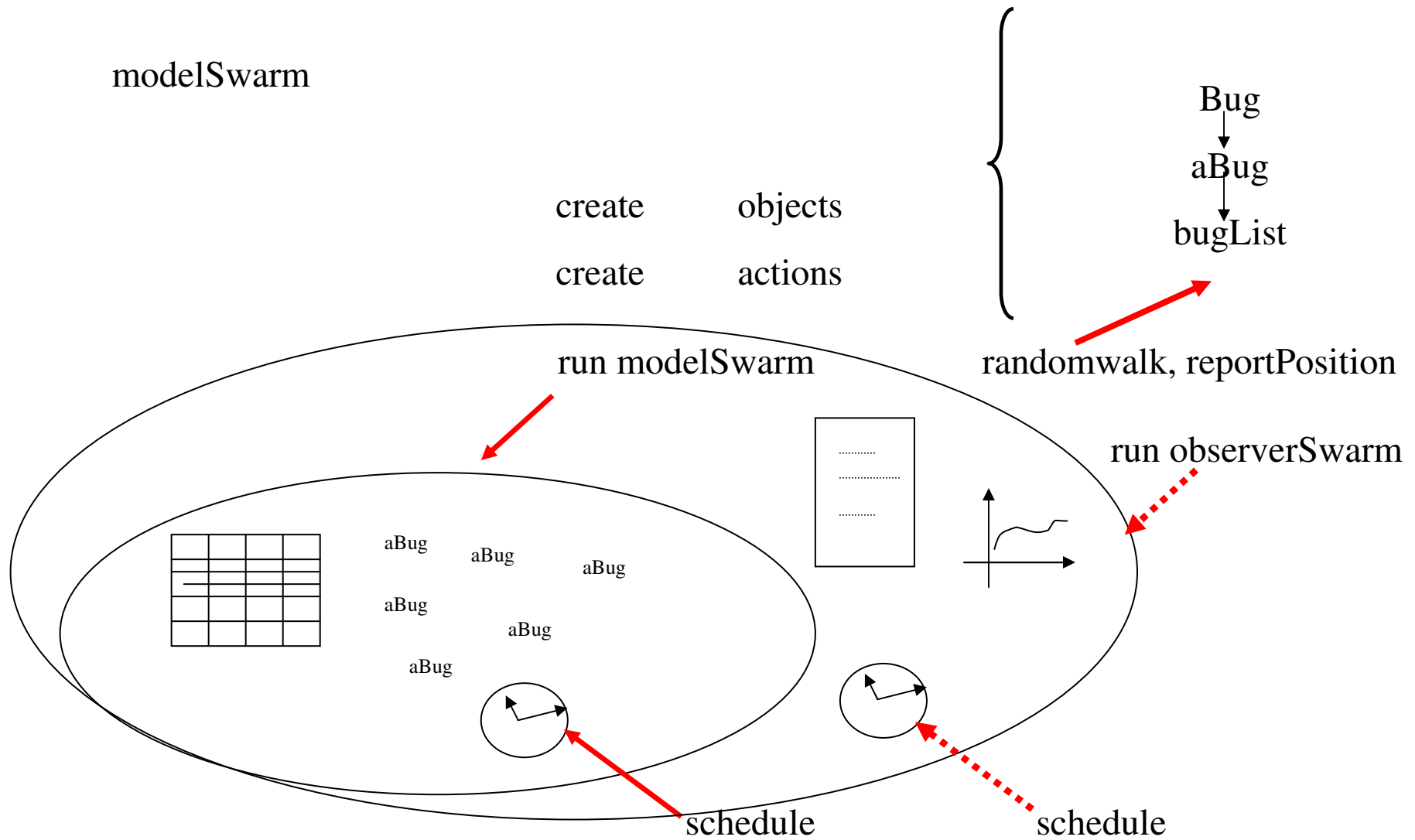
create objects

create actions

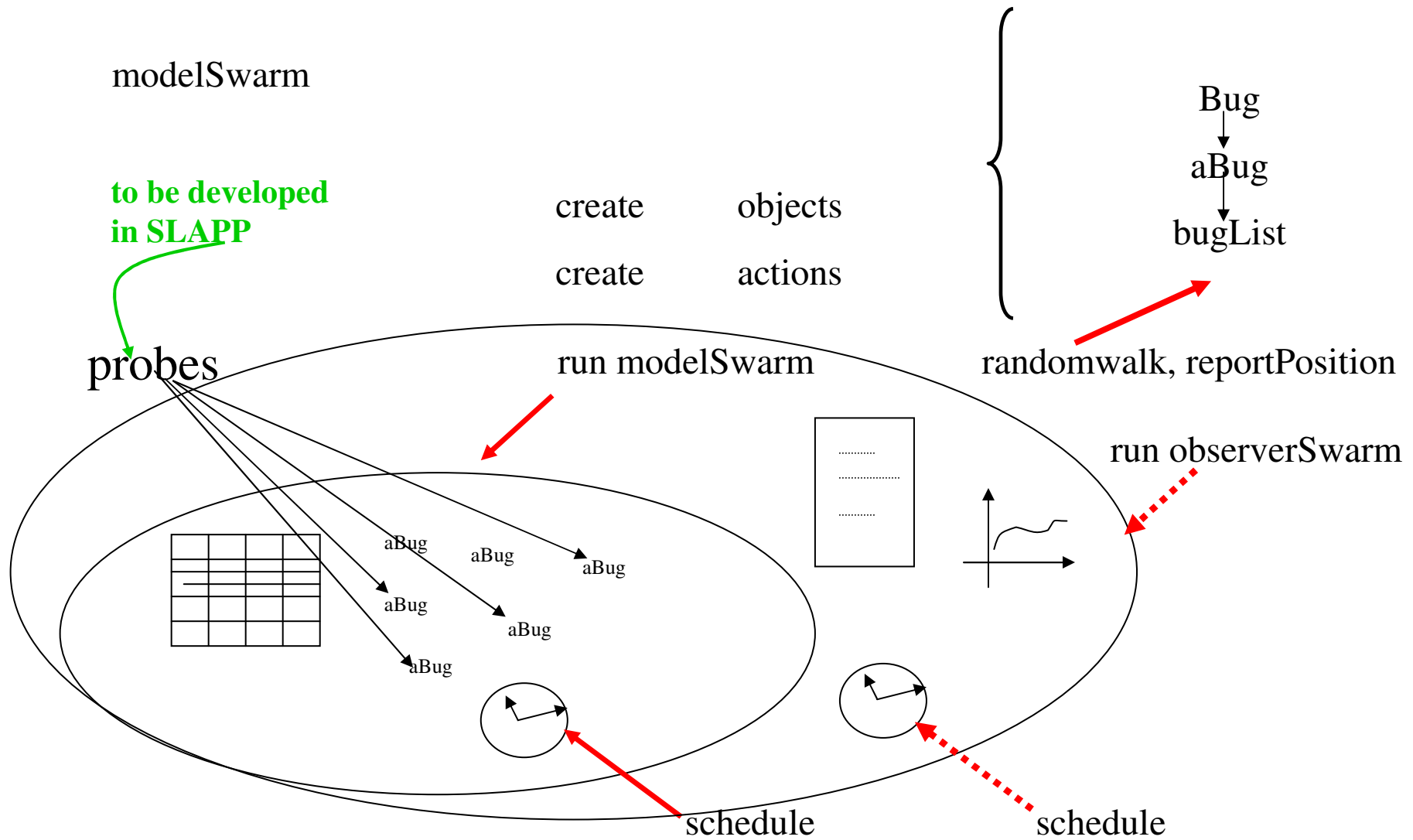
run modelSwarm



# Swarm = a library of functions and a **protocol**



# Swarm = a library of functions and a **protocol**



---

## Eating the pudding

---

## The Proof is in the Pudding!

What does that mean? This is an old proverb that has joined the microwave generation!

It has been clipped down from the original phrase which was: "The proof of the pudding is in the eating."

It means that the true value or quality of something can only be judged when it's put to use or tried and tested.

(For this study...we aren't talking about 'instant pudding'! We are talking about the pudding that is cooked and stirred on top of the stove....**takes time!**)



---

## Eating the pudding

### The surprising world of the Chameleons, with SLAPP

From an idea of Marco Lamieri, a project work with Riccardo Taormina

<http://eco83.econ.unito.it/terna/chameleons/chameleons.html>

---

The metaphorical models we use here is that of the **changing color chameleons**

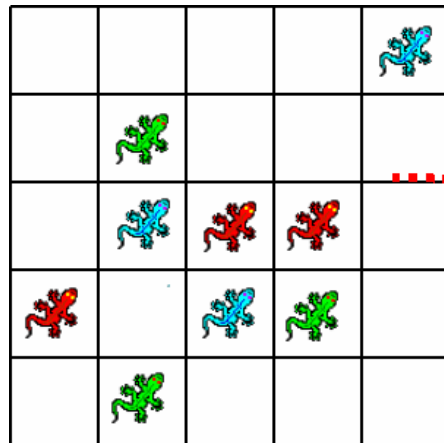
We have chameleons of three colors: red, green and blue

When two chameleons of different colors meet, they both change their color, assuming the third one (if all the chameleons get the same color, we have a steady state situation)

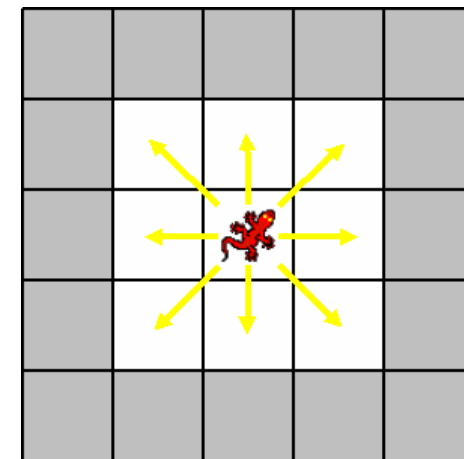
(The metaphor can also be interpreted in the following way: an agent diffusing innovation or ideas (or political ideas) can change itself via the interaction with other agents: as an example think about an academic scholar working in a completely isolated context or interacting with other scholars or with private entrepreneurs to apply the results of her work)

The simple model moves agents and changes their colors, when necessary

But what if the chameleons of a given color want to preserve their identity?



0	0	0	0	1
0	1	0	0	0
0	1	0	0	0
0	0	1	1	0
0	1	0	0	0

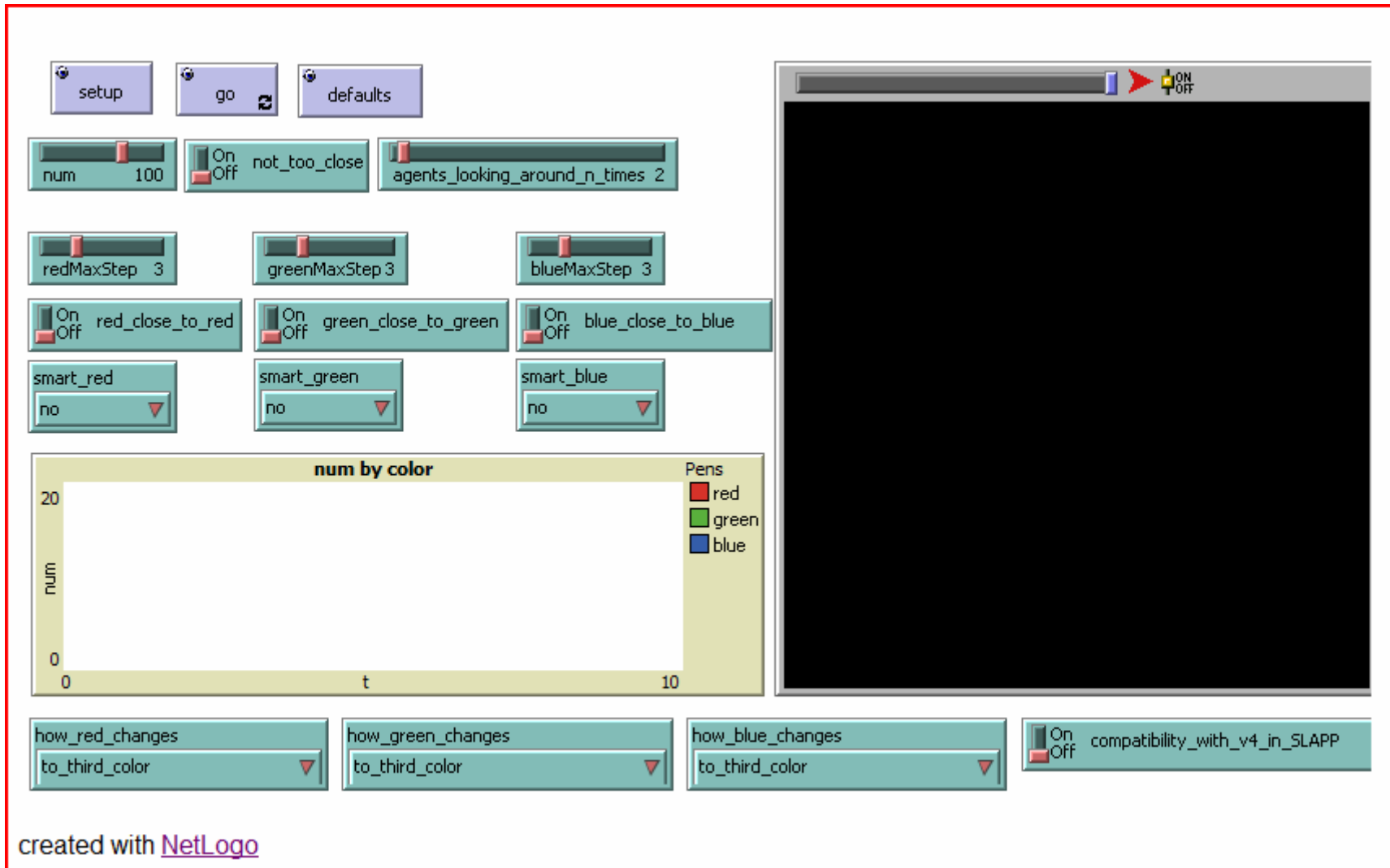


Preserving identity!

- **Reinforcement learning and pattern recognition, with bounded rationality**
- **Agent brain built upon 9 ANN**
- **In perspective, ANN with CTs**

The evaluation of the rewards is very simple: considering only a  $3 \times 3$  space, the red chameleons occupying the center in the red square has three enemies around it; moving to the north and being now in the center of the green square, it has only two (visible) enemies around it, with a +1 reward. reward can also be negative





setup go defaults

num 100  On  Off not\_too\_close agents\_looking\_around\_n\_times 2

redMaxStep 3 greenMaxStep 3 blueMaxStep 3

On  Off red\_close\_to\_red  On  Off green\_close\_to\_green  On  Off blue\_close\_to\_blue

smart\_red smart\_green smart\_blue

no no no

**num by color**

num 20 0

t 0 10

Pens  
red  
green  
blue

how\_red\_changes how\_green\_changes how\_blue\_changes compatibility\_with\_v4\_in\_SLAPP

to\_third\_color to\_third\_color to\_third\_color  On  Off

created with [NetLogo](#)

---

Have a look to SLAPP

---

# The secrets of Python and the structures of SLAPP

- Variables and dictionaries
- Classes, instances and collections of instances
- Direct access to methods

# Variables and dictionaries

a=1

a=[1,2,3]

a[0]

globals()

a=1.23

globals()

# Classes, instances and collections

collections.py

```
class A:  
    def setId(self,id):  
        self.id=id  
  
    def displayId(self):  
        print self.id  
  
a_collection=[]  
  
for i in range(1,11):  
    a=A()  
    a.setId(i*10)  
    a_collection.append(a)  
  
for a in a_collection:  
    a.displayId()  
  
for i in range(len(a_collection)):  
    a_collection[i].displayId()
```

let try

*globals(); vars(a\_collections[0]);*

```
class A:  
    def p(self,**k):  
        """ use with x= and y= """  
        print len(k)  
        self.x=k["x"]  
        self.y=k["y"]  
        print self.x + self.y
```

```
a=A()
```

```
A.p(a,x=222,y=111)
```

```
m=A.p
```

```
def f(instance,method,**dictionary): method(instance,**dictionary)
```

```
f(a,m,x=222,y=111)
```

let try

globals(); vars(a); a

A lot of handbooks

Two significant papers:

Isaac A. G. (2008), Simulating Evolutionary Games: A Python-Based Introduction. *Journal of Artificial Societies and Social Simulation*, 11, 3.  
<http://jasss.soc.surrey.ac.uk/11/3/8.html>.

(Patrick O'Brien (2002), Guide to Python introspection,  
<http://www.ibm.com/developerworks/library/l-pyint.html>

# The secrets of Python and the structures of SLAPP

start 1 plainProgrammingBug

start 2 basicObjectProgrammingBug

start 3 basicObjectProgrammingManyBugs

start 4 basicObjectProgrammingManyBugs\_bugExternal+\_shuffle

start 5 objectSwarmModelBugs.py

start 6 objectSwarmObserverBugs.py

start 7 objectSwarmObserverTkBugs.py

```
#start 6 objectSwarmObserverBugs.py
from Tools import *
from ObserverSwarm import *

observerSwarm = ObserverSwarm()

# create objects
observerSwarm.buildObjects()

# create actions
observerSwarm.buildActions()

# run
observerSwarm.run() |
```

```
#ObserverSwarm.py|
from ModelSwarm import *
from Tools import *
from Bug import *
from ActionGroup import *

class ObserverSwarm:

    # create objects
    def buildObjects(self):
        self.nBugs = input("How many bugs? ")
        self.worldXSize= input("X Size of the world? ")
        self.worldYSize= input("Y Size of the world? ")
        self.nCycles = input("How many cycles? (0 = exit) ")
        self.conclude=False
        self.t=0 # time will start with a 0 value in the first step

        self.modelSwarm = ModelSwarm(self.nBugs,
                                     self.worldXSize, self.worldYSize)
        self.modelSwarm.buildObjects()
```

```
# actions
def buildActions(self):

    self.modelSwarm.buildActions()

    self.actionGroup1 = ActionGroup ("clock")
    def do1(address, nCycles, actionList):

        self.t+=1 #the clock running
        if self.t+1==nCycles:
            insertASubStepElementInNextStep_firstPosition(actionList,"end")

    self.actionGroup1.do = do1 # do is a variable linking a method

    self.actionGroup2a = ActionGroup ("talk all")
    def do2a(address):

        # ask each agent, without parameters

        print "Time = ", self.t, "ask all agents to report position"
        askEachAgentIn(address.modelSwarm.getBugList(),Bug.reportPosition)

    self.actionGroup2a.do = do2a # do is a variable linking a method

    self.actionGroup2b = ActionGroup ("talk one")
    def do2b(address):

        # ask a single agent, without parameters
        print "Time = ",self.t,"ask first agent to report position"
        askAgent(address.modelSwarm.getBugList()[0],Bug.reportPosition)

    self.actionGroup2b.do = do2b # do is a variable linking a method
```

```
self.actionGroup3 = ActionGroup ("end")
def do3(address):
    self.conclude=True
self.actionGroup3.do = do3 # do is a variable linking a method

# schedule
self.actionList = [{"talk all", "clock"}, \
                   ["talk one", "clock"], \
                   ["talk one", "clock"] ]

#or
#self.actionList = [{"talk all"}, ["clock"], \
#                   ["talk one"], ["clock"], \
#                   ["talk one"], ["clock"] ]
```

```
# run
def run(self):

    while not self.conclude:

        self.modelSwarm.step() # model steps before than observer looks
                               # at the effects

        step=extractAStepAndRotate(self.actionList)

        while len(step)>0:
            subStep=extractASubStep(step)

            if subStep == "clock":
                self.actionGroup1.do(self, self.nCycles, self.actionList)
                # self here is the model env.
                # not added automatically
                # being do a variable

            if subStep == "talk all":
                self.actionGroup2a.do(self)

            if subStep == "talk one":
                self.actionGroup2b.do(self)

            if subStep == "end":
                self.actionGroup3.do(self)
```

```
#ModelSwarm.py
import Tools
from Bug import *
from ActionGroup import *

class ModelSwarm:
    def __init__(self, nBugs, worldXSize, worldYSize):
        # the environment
        self.nBugs = nBugs
        self.bugList = []

        self.worldXSize= worldXSize
        self.worldYSize= worldYSize

    # objects
    def buildObjects(self):
        for i in range(self.nBugs):
            aBug = Bug(i, random.randint(0,self.worldXSize-1),
                       random.randint(0,self.worldYSize-1), self.worldXSize,
                       self.worldYSize)
            self.bugList.append(aBug)
        print
```

```
# actions
def buildActions(self):

    self.actionGroup1 = ActionGroup ("move")
    def do1(address):

        actionDictionary[self.actionGroup1.getName()]=self.actionGroup1

        # keep safe the original list
        address.bugListCopy=address.bugList[:]
        # never in the same order (please comment if you want to keep
        # always the same sequence
        random.shuffle(address.bugListCopy)
        # move with a jump, to have to transfer a parameter
        # the format is: collection, method, parameters by name
        # ask each agent, without parameters
        # the potential jump is the same for all the agents
        askEachAgentIn(address.bugListCopy,Bug.randomWalk,
                        jump=random.uniform(0,5))

        self.actionGroup1.execLocalCode()

    self.actionGroup1.do = do1 # do is a variable linking a method
```

```
# schedule
self.actionList = ["move"]

# or
#self.actionList = ["move"], []

# run a step
def step(self):

    step=extractAStepAndRotate(self.actionList)


    while len(step)>0:
        subStep=extractASubStep(step)

        if subStep == "move":
            self.actionGroup1.do(self)
            # self here is the model env.
            # not added automatically
            # being do a variable
```

```
#ActionGroup.py
from Tools import *

class ActionGroup:
    def __init__(self, groupName = " "): # the name is optional
        self.groupName = groupName
        self.localCode=""

    # reporting name
    def getName(self):
        return self.groupName

    # place into the instances of this class parts of code generated
    # by the agents, in the form 

    # exec("instruction 1; instruction 2; ...")
    # pay attention to the semicolon
    def execLocalCode(self):
        exec(self.localCode)
        self.cleanLocalCode()

    def cleanLocalCode(self):
        self.localCode=""

    def getLocalCode(self):
        return self.localCode

    def setLocalCode(self, code):
        self.localCode=code
```

```
# the action, now jumping
def randomWalk(self, **k):
    print "bug # %2d moving" % self.number
    self.jump=k["jump"]
    dx=randomMove(self.jump)
    self.xPos +=dx
    dy=randomMove(self.jump)
    self.yPos += dy
    self.xPos = (self.xPos + self.worldXSize) % self.worldXSize
    self.yPos = (self.yPos + self.worldYSize) % self.worldYSize

    if abs(dx) > 4 and abs(dy) > 4 :

# modify actionGroup "move"
    actionGroup=actionDictionary.get("move", "none")
        # in case, return "none"

    if actionGroup=="none": print "warning, actionGroup not found "+\
        "in agent "+ str(self.number)

    else: actionGroup.setLocalCode(actionGroup.getLocalCode()+\
        "print 'agent %d made a big jump';" % self.number)
```

