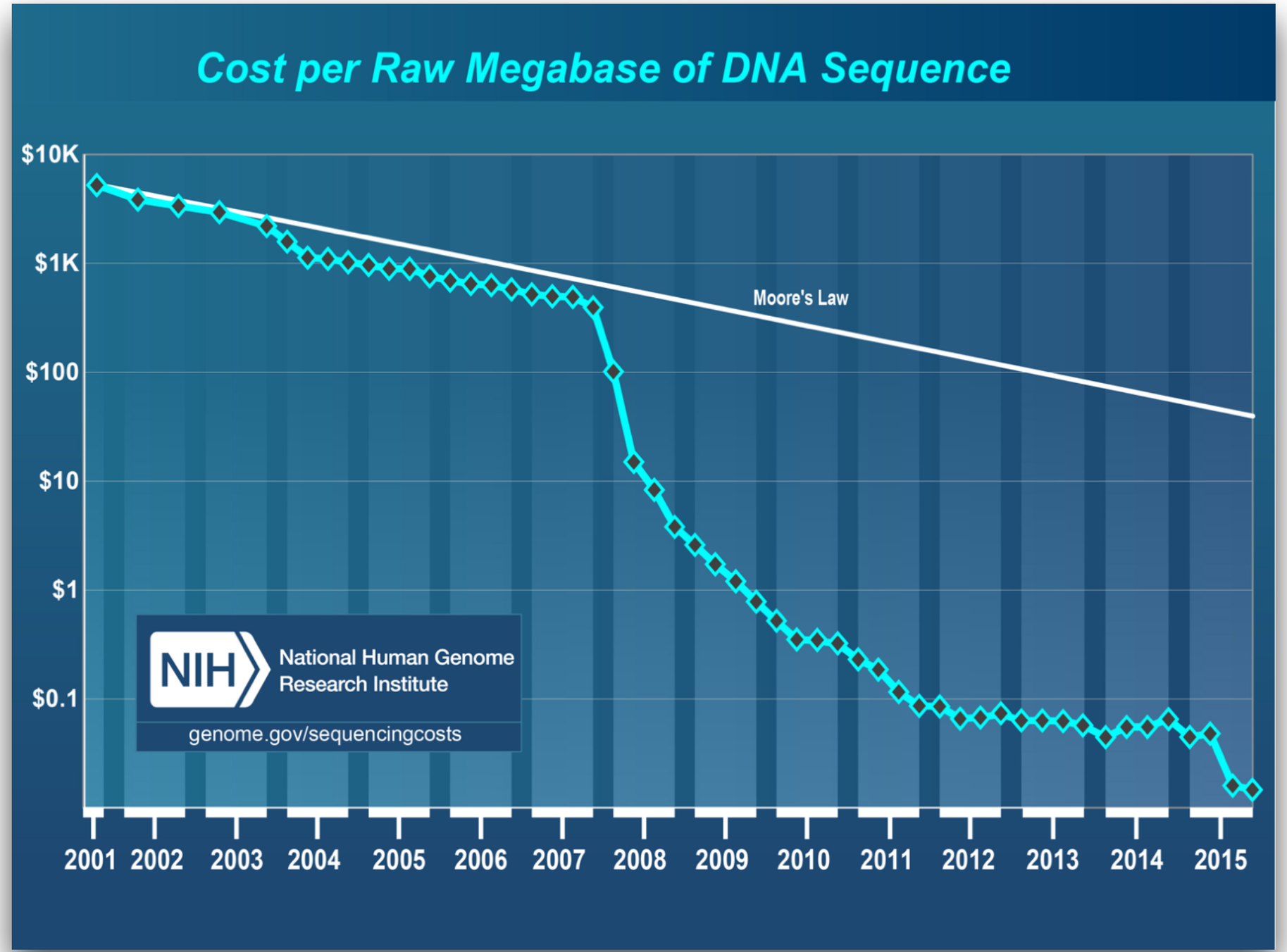
Computing through Biology with Jupyter

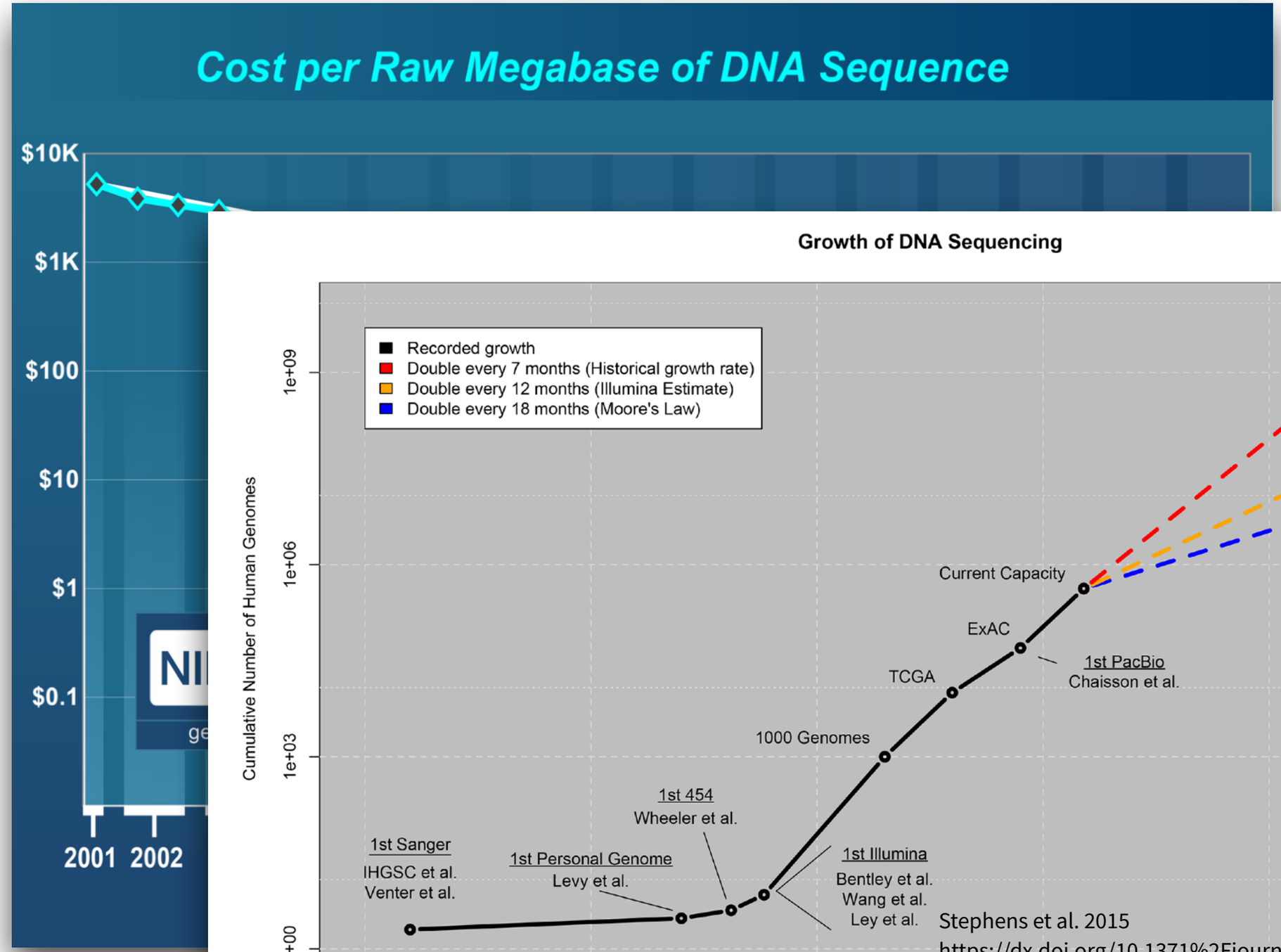
Joshua A Shapiro
Bryn Mawr College
@jashapiro

with Doug Blank

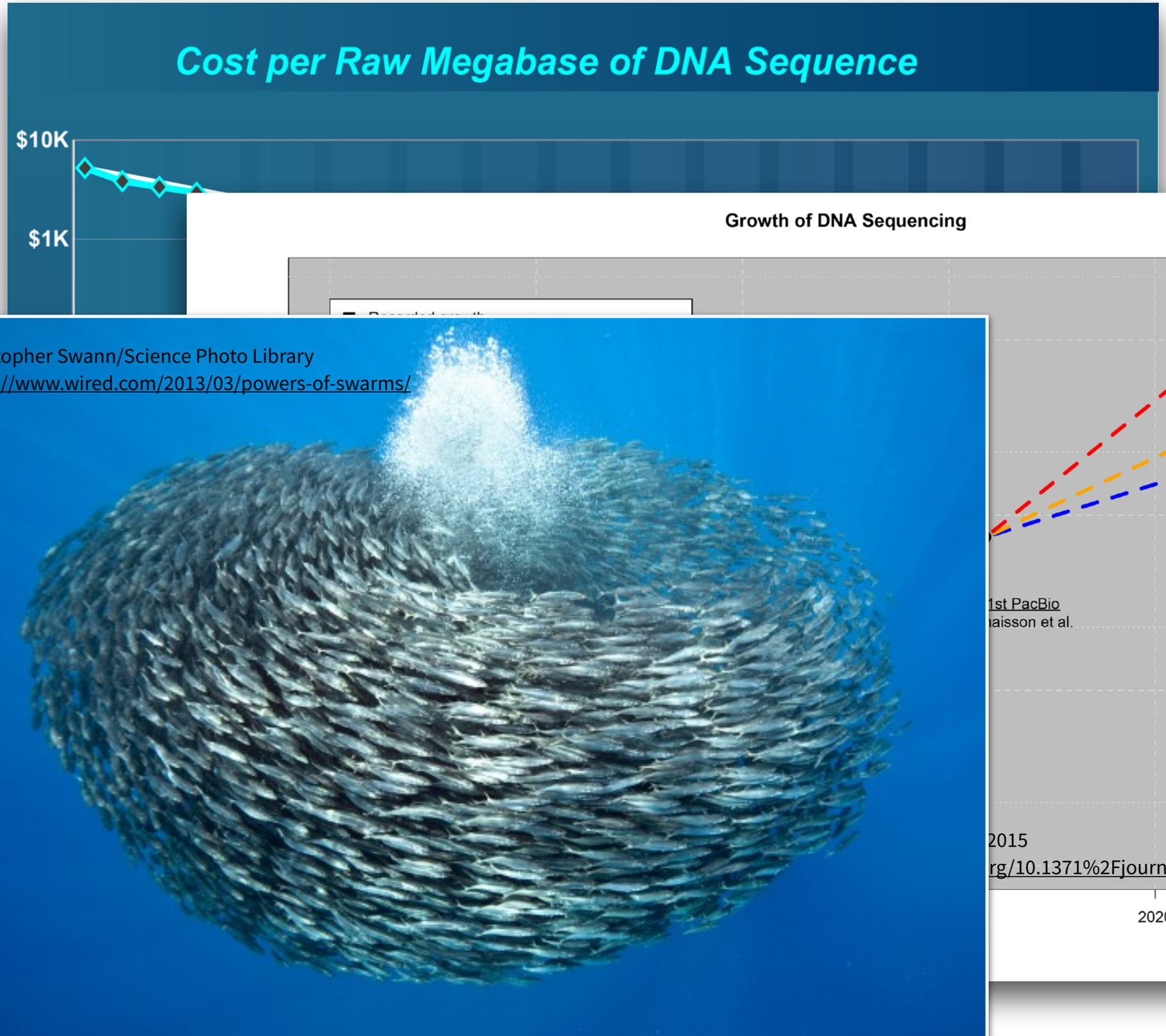
The New Era of Biology



The New Era of Biology

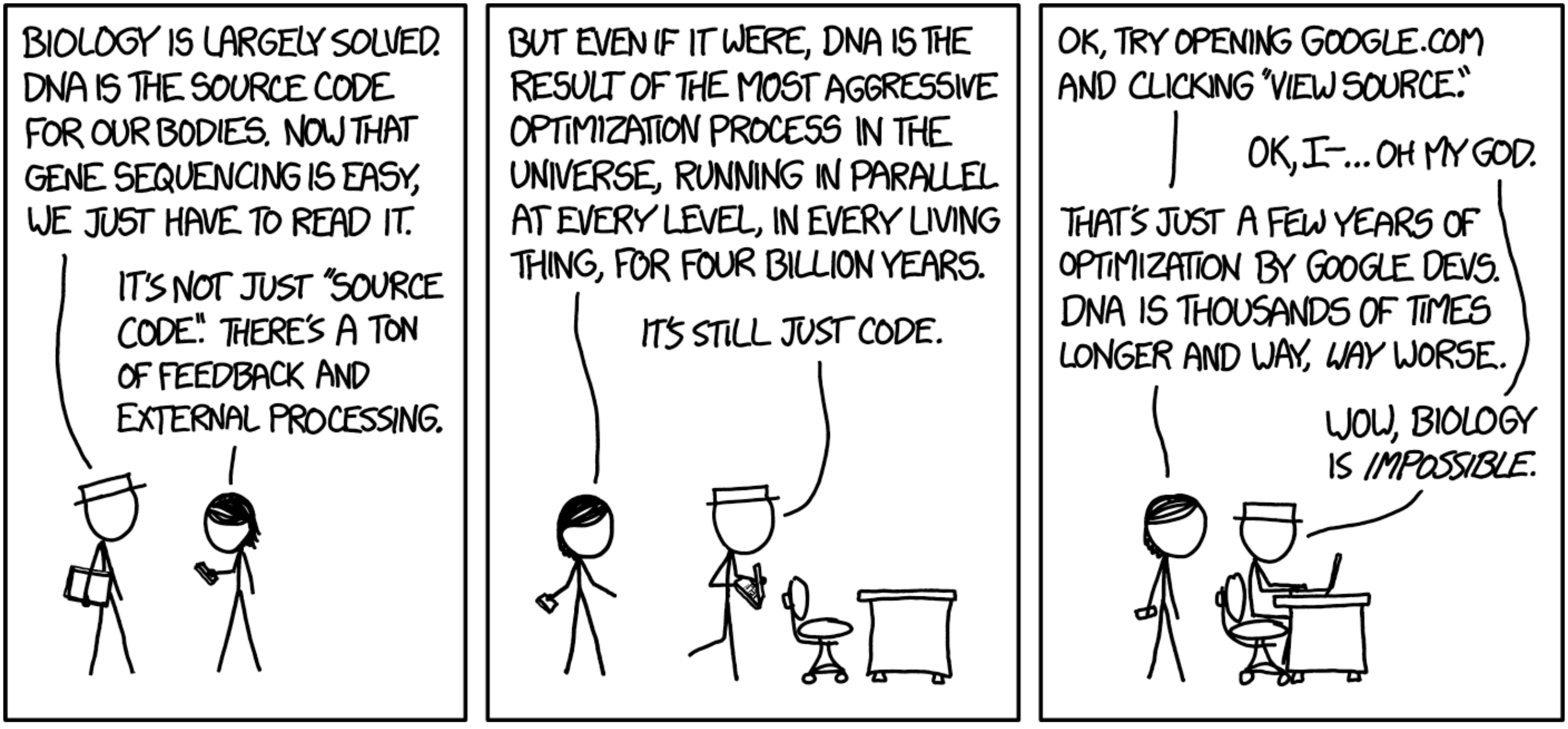


The New Era of Biology



Christopher Swann/Science Photo Library
https://www.wired.com/2013/03/powers-of-swarms/

The New Era of Biology



https://xkcd.com/1605/

Current Bryn Mawr Biology Major Computational Requirements

This slide intentionally left blank

Current Bryn Mawr Biology Major Options for Computing

- **Intro CS**
- **Experimental Design and Statistics**
  - (Biostatistics)
  - coding in R
- **Computational Methods in the Sciences**
  - also R (Python in the past)
- **Computational Methods Minor**
  - Standard CS intro + some advanced courses
  - Applied courses in other science departments

Current Bryn Mawr CS Major Options for Biology

- Intro Bio
- Evolution
- Experimental Design and Statistics
- Computational Methods in the Sciences

Challenges

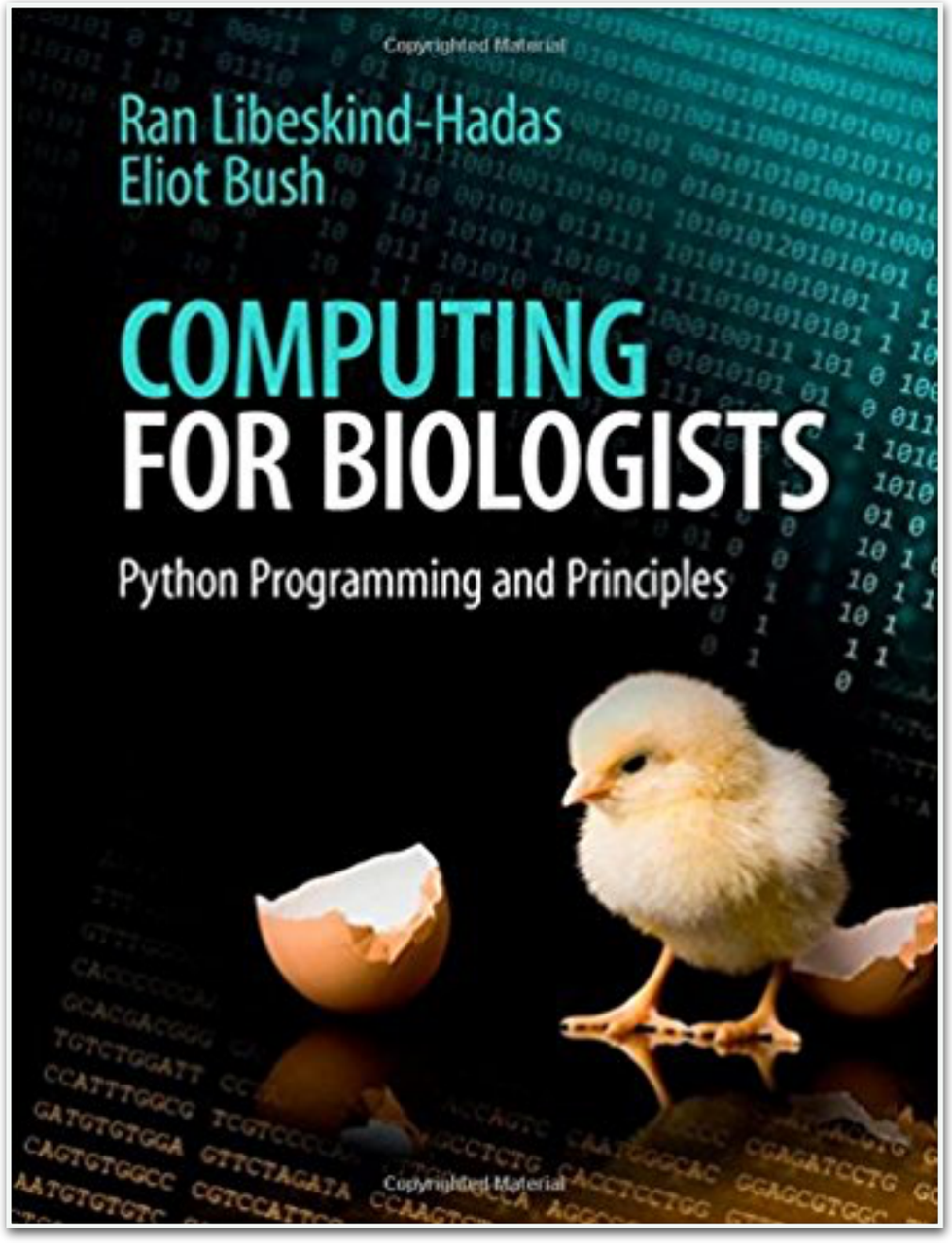
- **Exposure**
  - Most bio-inclined students have not seen much computer science
  - Most CS-inclined students have not seen biological applications
- **Course planning**
  - Too many intro courses to take them all early
  - Overenrolled intros
  - Dueling schedules
- **Fear, discomfort, prior experience** 😱

A New Course

- **Bio/CS 115: Computing Through Biology**
- **Alternative to second semester of Intro Bio**
- **Alternative to Intro CS**
- **Counts in both departments**
  - students can continue to upper level courses in either or both subjects

Not “new”

- Harvey Mudd CS 5 Green



Course Topics

- **CS topics**
  - variables, functions, loops, etc.
  - recursion
  - object-oriented design
  - algorithm performance
- **Bio topics**
  - Bioinformatics (DNA sequence analysis)
  - Population Genetics/Evolution
  - Ecological Dynamics

Course Participants (S2016)

- **2 faculty**
  - Joshua Shapiro (Biology)
  - Doug Blank (Computer Science)
- **16 diverse students**
  - 4 seniors, 3 juniors, 2 sophomores, 7 freshman
  - Majors: bio, biochem, chem, math, psych, anthropology, classics, music
  - 9 registered through Bio, 7 through CS
- **Prior exposures**
  - 3 intro CS
  - 8 intro Bio (1 semester), 4 advanced Bio

Use of Jupyter

- **All course content on JupyterHub**
- **Live coding lectures**
  - Notebooks available before (at the start of) class
  - Add and modify code during class
  - Students experiment in parallel with lecture
  - Save final notebook for students to reference
- **Lab assignments**
  - nbgrader
    - *combination of auto and manual grading*

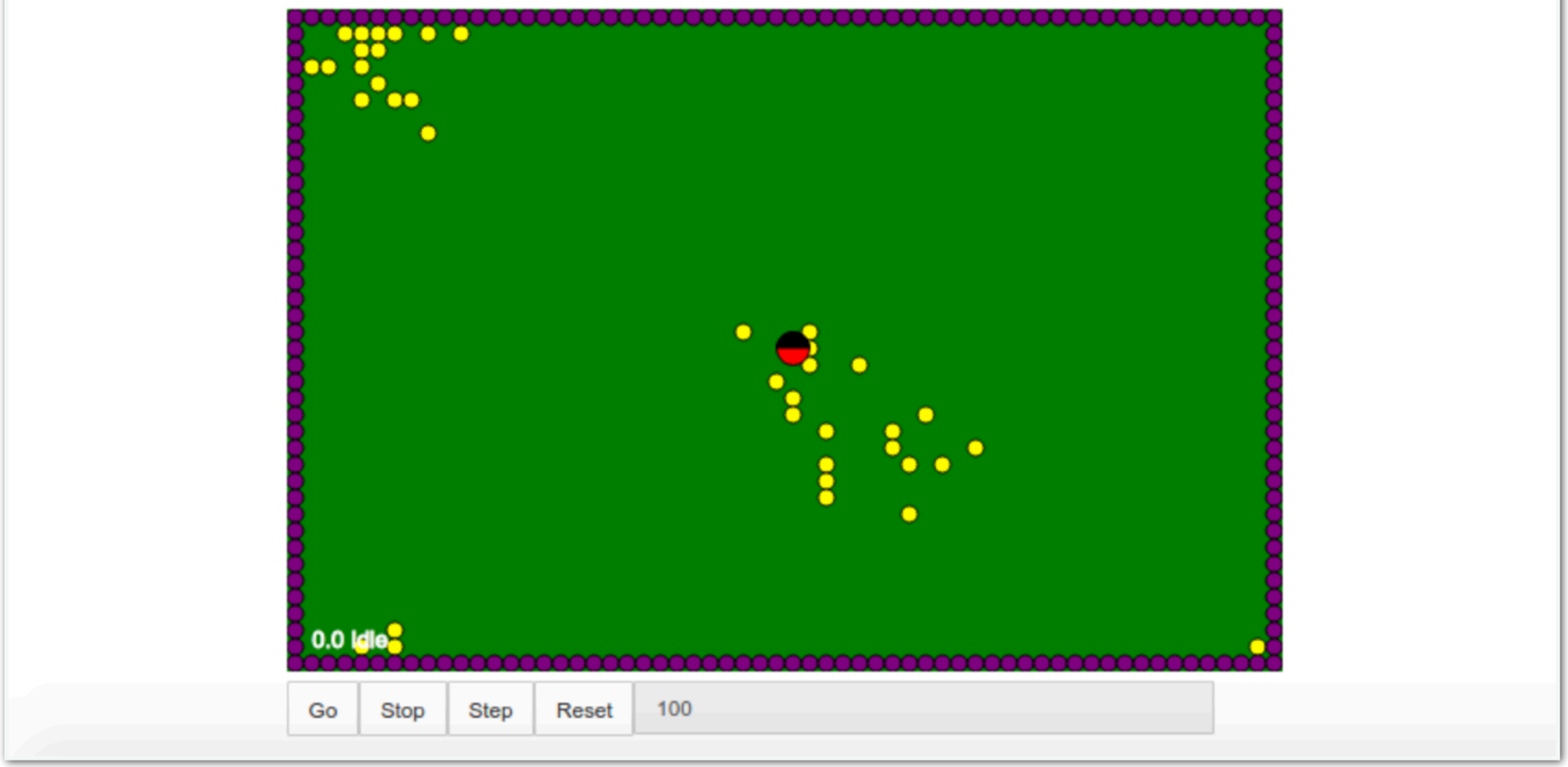
Lab Assignments

As exploratory as possible

1.2 Ladybug Simulation

Our first simulation is a discrete simulation: the world is divided into discrete locations (called patches) and the world operates in discrete time steps.

A Ladybug world looks like this:

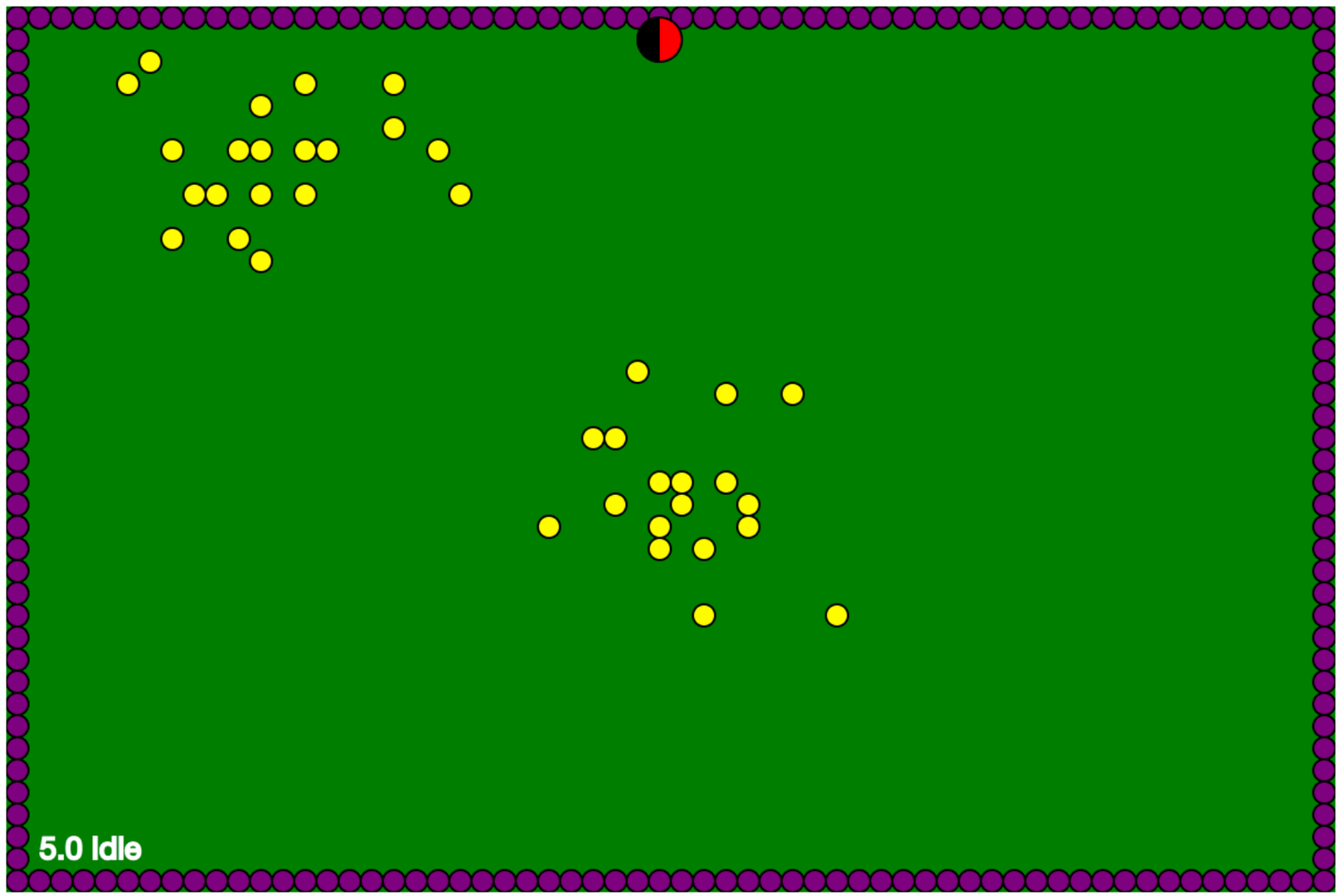


Ladybug by D. Blank, inspired by PicoBot

Start with simple rules

In [1]:
```
%%simulation

0 **w** -> turnLeft(90)   0
0 **f** -> forward(1)     1
0 ***** -> forward(1)     0
1 **f** -> forward(1)     1
1 ****f -> turnRight(90)     1
1 ***** -> turnLeft(90)   0
```

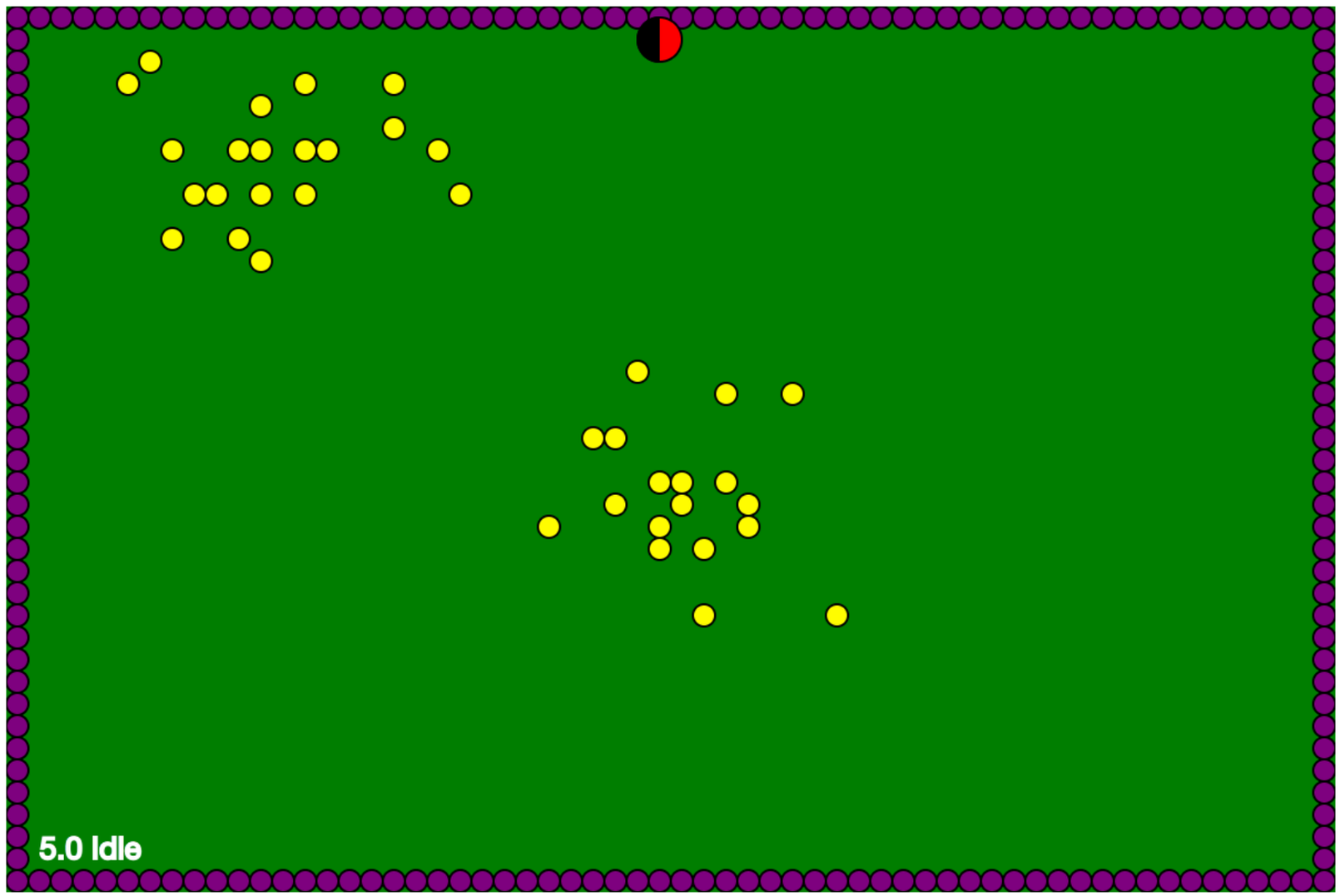


Go | Stop | Step | Restart | 112.75

Accessible internals

In [1]:
```
%%simulation

0 **w** -> turnLeft(90)   0
0 **f** -> forward(1)     1
0 ***** -> forward(1)     0
1 **f** -> forward(1)     1
1 ****f -> turnRight(90)     1
1 ***** -> turnLeft(90)   0
```

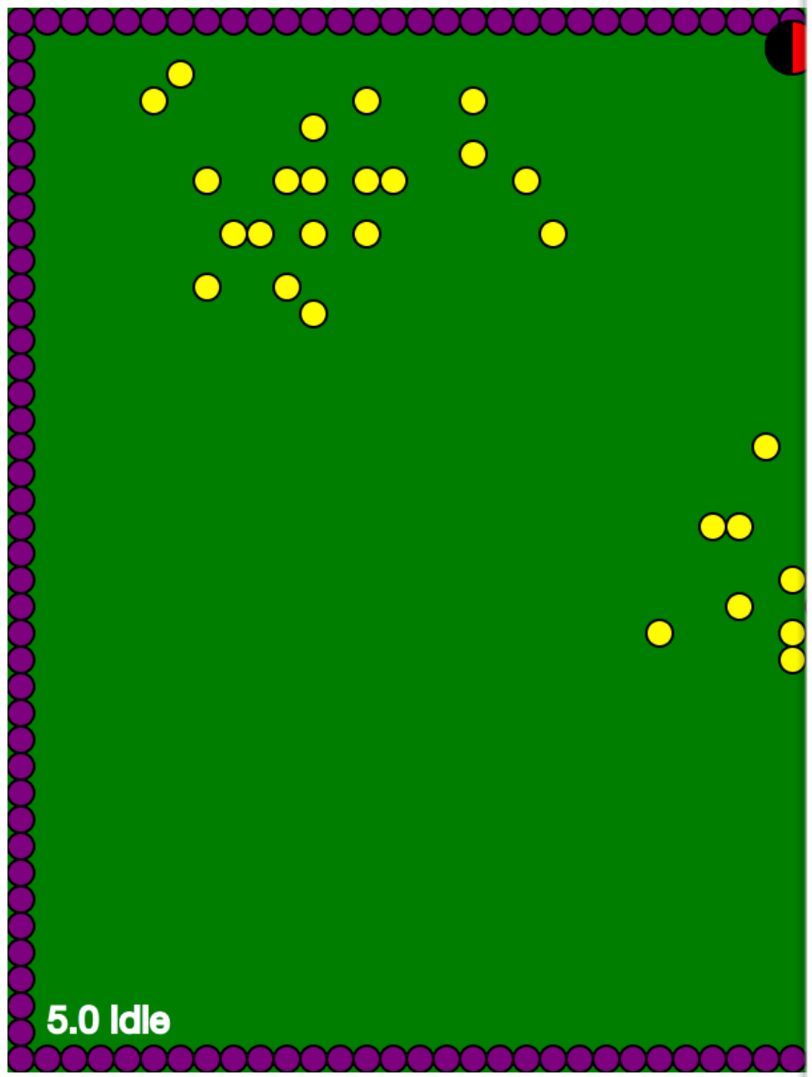


Go | Stop | Step | Restart | 112.75

Accessible internals

In [1]:
```
%%simulation

0 **w** -> turnLeft(90)   0
0 **f** -> forward(1)     1
0 ***** -> forward(1)     0
1 **f** -> forward(1)     1
1 ****f -> turnRight(90)     1
1 ***** -> turnLeft(90)   0
```

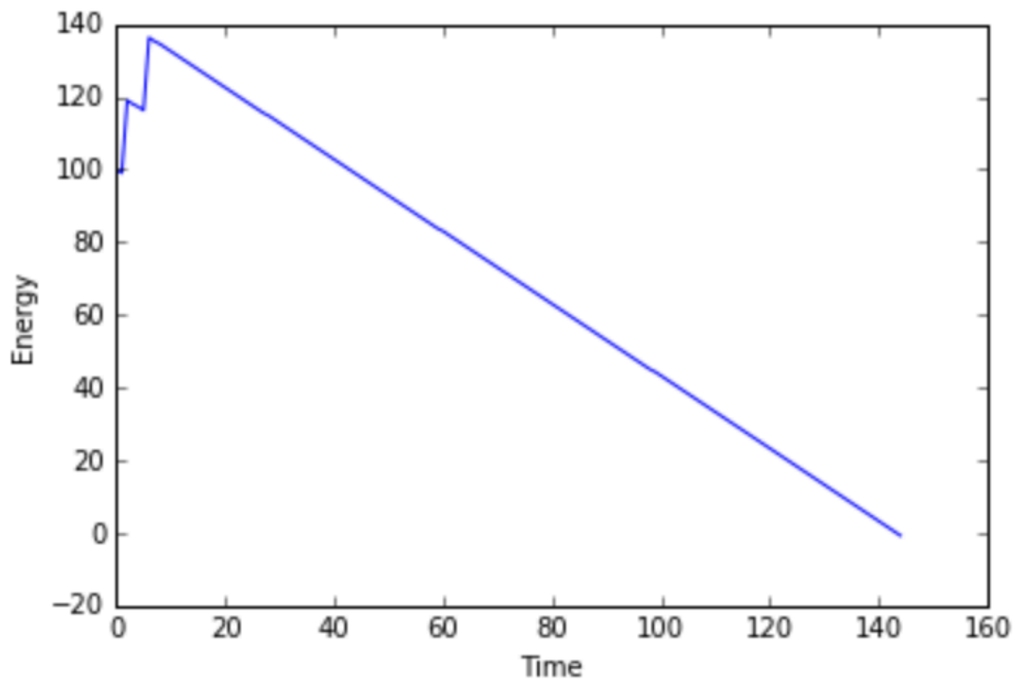


Finally, we can create a quick visualization of the energy level over time:

```
%matplotlib inline

import matplotlib.pyplot as plt

plt.plot(ladybug.history)
plt.ylabel("Energy")
plt.xlabel("Time")
plt.show()
```



Go | Stop | Step | Restart | 112.75

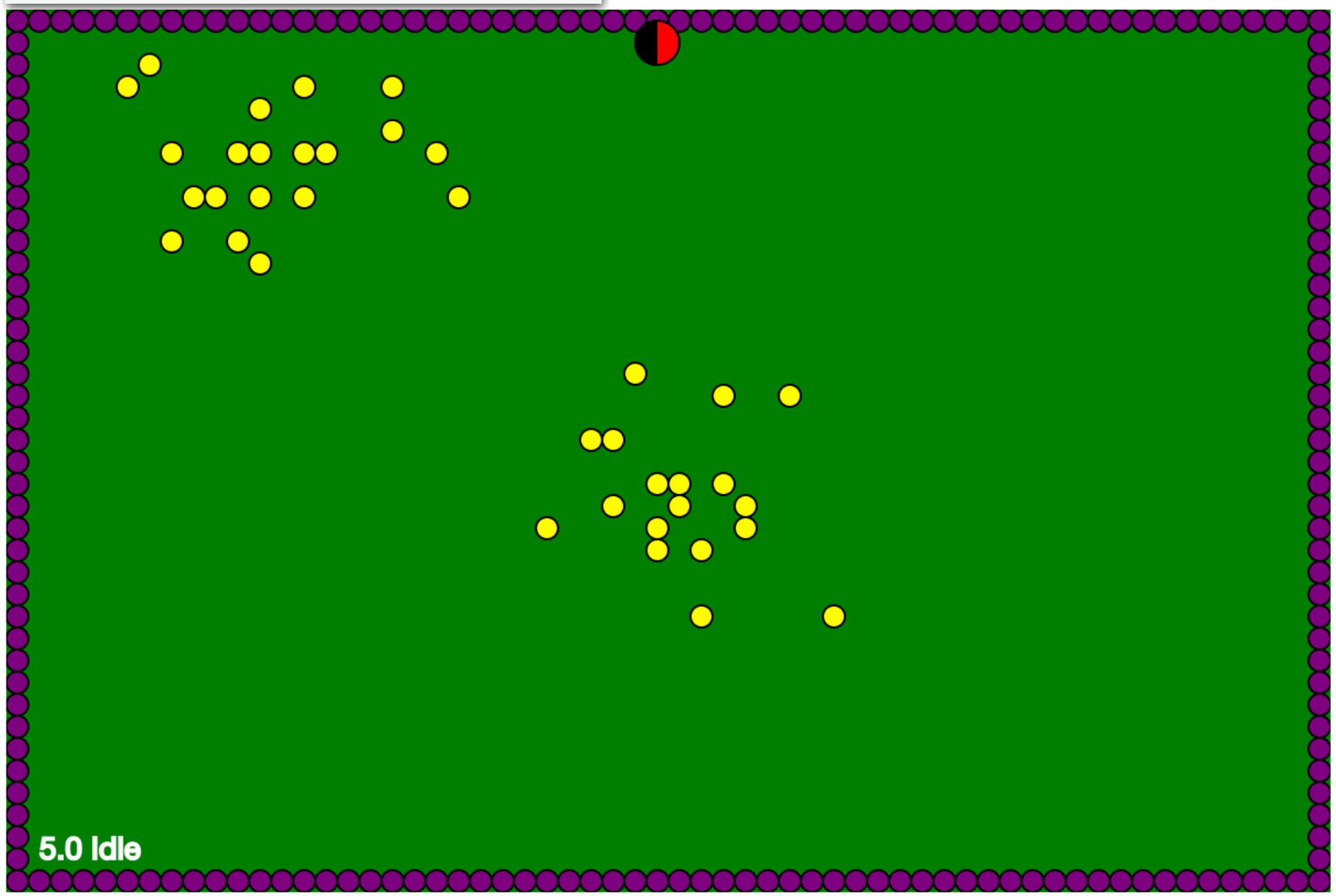
Build to python control

In [1]:

```
%%simulation

# My first brain!

def brain(robot):
    senses = robot.getSenses()
    if senses[2] == 'f':
        robot.forward(1)
    else:
        robot.turnLeft(90)
```



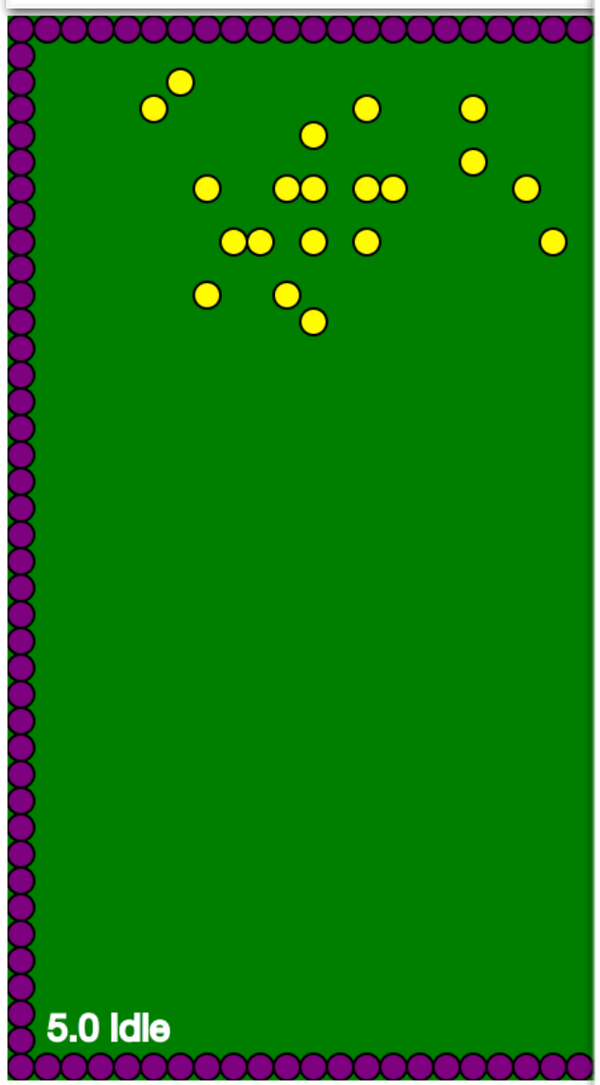
Go | Stop | Step | Restart | 112.75

Build to python control

In [1]:
```
%%simulation

# My first brain!

def brain(robot):
    senses = robot.getSenses()
    if senses[2] == 'f':
        robot.forward(1)
    else:
        robot.turnLeft(90)
```



Go

In [74]:
```
%%simulation

def brain(robot):
    senses = robot.getSenses()
    if robot.state == '0':                                  #normal state
        if senses[2] == 'f':
            robot.forward(1)
        elif senses[4] == 'f':
            robot.turnRight(90)
        elif senses[0] == 'f':
            robot.turnLeft(90)
        elif senses[2] == 'w':
            robot.turnLeft(90)
        elif len(robot.history) >= 2 and robot.history[-2] < robot.energy:
            robot.state = '1'
        elif 50 < robot.energy < 51:
            robot.state = '2'
        else:
            robot.forward(1)

    elif robot.state == '1':                                #food cluster state
        if senses[2] == 'f':
            robot.forward(1)
        elif senses[4] == 'f':
            robot.turnRight(90)
        elif senses[0] == 'f':
            robot.turnLeft(90)
        else:
            robot.forward(1)
            robot.forward(1)
            robot.turnRight(90)

        if robot.history[-2] > robot.energy:
            robot.state = '0'

    elif robot.state == '2':                                #hail mary state
            robot.turnLeft(90)
            robot.state = '0'

    else:
        robot.forward(1)
    print (robot.getSenses(), robot.state, robot.energy)
```

Some code has to be correct

- **Solution: provide autograded tests**

-

Once you can work with a sequence, we will want to calculate the GC content for each sequence in the file. This is the number of G nucleotides plus the number of C nucleotides divided by the total number of G, C, A, and T. While we only have one strand of DNA, by combining the G and C counts, we are effectively counting the percentages on both strands combined. Why might that be? (Why might using the full length of the sequence cause problems in some cases?)

Use the space below to write a function to calculate GC content below:

In [3]: ID: calculateGC — Autograded answer

```
def calculateGC(sequence):
    """Computes the GC content of a DNA sequence.
    Ignores ambiguous bases, but allows for lower or upper case letters to
    ### BEGIN SOLUTION
    # Note that this solution does not check for zero length sequences, or
    sequence = sequence.upper()
    A_count = sequence.count("A")
    T_count = sequence.count("T")
    C_count = sequence.count("C")
    G_count = sequence.count("G")
    GC_content = (C_count + G_count) / \
                 (A_count + T_count + C_count + G_count)

    ### END SOLUTION
    return GC_content
```

In [4]: Points: 25 — ID: GC_test — Autograder tests

```
from nose.tools import assert_equal

assert_equal(calculateGC("ATTGCC"), 0.5)
assert_equal(calculateGC("ATTCCNN"), 0.4)
assert_equal(calculateGC("atga"), 0.25)
```

Some code has to be correct

- **Solution: provide autograded tests**

Once you can work with a sequence, we will want to calculate the GC content for each sequence in the file. This is the number of G nucleotides plus the number of C nucleotides divided by the total number of G, C, A, and T. While we only have one strand of DNA, by combining the G and C counts, we are effectively counting the percentages on both strands combined. Why might that be? (Why might using the full length of the sequence cause problems in some cases?)

Use the space below to write a function to calculate GC content below:

In [3]: ID: calculateGC — Autograded answer

```
def calculateGC(sequence):
    """Computes the GC content of a DNA sequence.
    Ignores ambiguous bases, but allows for lower or upper case letters to
    ### BEGIN SOLUTION
    # Note that this solution does
    sequence = sequence.upper()
    A_count = sequence.count("A")
    T_count = sequence.count("T")
    C_count = sequence.count("C")
    G_count = sequence.count("G")
    GC_content = (C_count + G_count
                  (A_count + T_count

    ### END SOLUTION
    return GC_content
```

In [4]: Points: 25

```
from nose.tools import assert_equal

assert_equal(calculateGC("ATTGCC"),
assert_equal(calculateGC("ATTCCNN")
assert_equal(calculateGC("atga"), 0
```

Use the space below to write a function to calculate GC content below:

```
def calculateGC(sequence):
    """Computes the GC content of a DNA sequence.
    Ignores ambiguous bases, but allows for lower or upper
    # YOUR CODE HERE
    raise NotImplementedError()
    return GC_content
```

```
from nose.tools import assert_equal

assert_equal(calculateGC("ATTGCC"), 0.5)
assert_equal(calculateGC("ATTCCNN"), 0.4)
assert_equal(calculateGC("atga"), 0.25)
```

Exploratory questions

1.3.2 Exploration topics

Using the genome or genomes that you downloaded, explore the patterns that you see in GC content within or across genomes. Questions you might want to explore include:

- How much does GC content vary between species?
  - Which species has the highest/lowest GC on average?
- How does GC content vary within a species?
  - For organisms with multiple chromosomes, do they all have the same GC content?
  - How much variation is there in GC content along a chromosome?
    - Note that you will need to select subsegments of a chromosome to do this
    - How big should those subsegments be? Does the scale matter?
- Test Chargaff's second rule:
  - Recall that Chargaff's rule says that G and C counts along a chromosome should be equal (as should A & T).
  - Write a function to calculated these relationships for a sequence
  - Apply that function to test the rule across species, chromosomes, or segments.

You should aim to explore at least two of the topics above.

Reflection

- **Every assignment ends with reflection**
  - What did you learn?
  - What was challenging?
  - What do you want to explore further?

Outcomes

- **???**
- **Only taught once…**
  - 2 semesters of follow up
  - “computational” bio courses not offered
- **of 11 non-seniors**
  - 8 took upper level course(s) in Biology
  - 1 took upper level course(s) in CS
- **of 7 freshman**
  - 5 declared bio or biochem major

Student comments

- “I enjoyed the integration of two different departments and topics. Should be more classes like this in the future”
- “I didn’t find it particularly helpful to have Bio and CS combined.”
- “A fun and constructive way to learn Python and a bit about computer science.”
- “Coding is frustrating 😠”
- “A lot of work”
- “Pace of the course felt very fast, sometimes uncomfortably so.”
- “Some of the topics were too advanced.”
- “All of [the bio] was review… it would have been nice to learn some new stuff.”
- “Some of the assignments felt like too much programming had been done for us.”

The Future…

- **To be taught again spring 2018**
  - just me (no Doug 😢)
- **More recruiting!**
  - recruiting Bio students was “easy”
    - *Most are in Intro Bio in the fall*
  - CS students have to be found “in the wild”
- **Refining topics…**
  - reduce molecular bio content (DNA structure, etc.)
  - increase organismal bio
  - more coding practice

Thank You

- Doug Blank
- Bryn Mawr Biology Department
- Bryn Mawr Computer Science Department
- Project Jupyter

- Students!