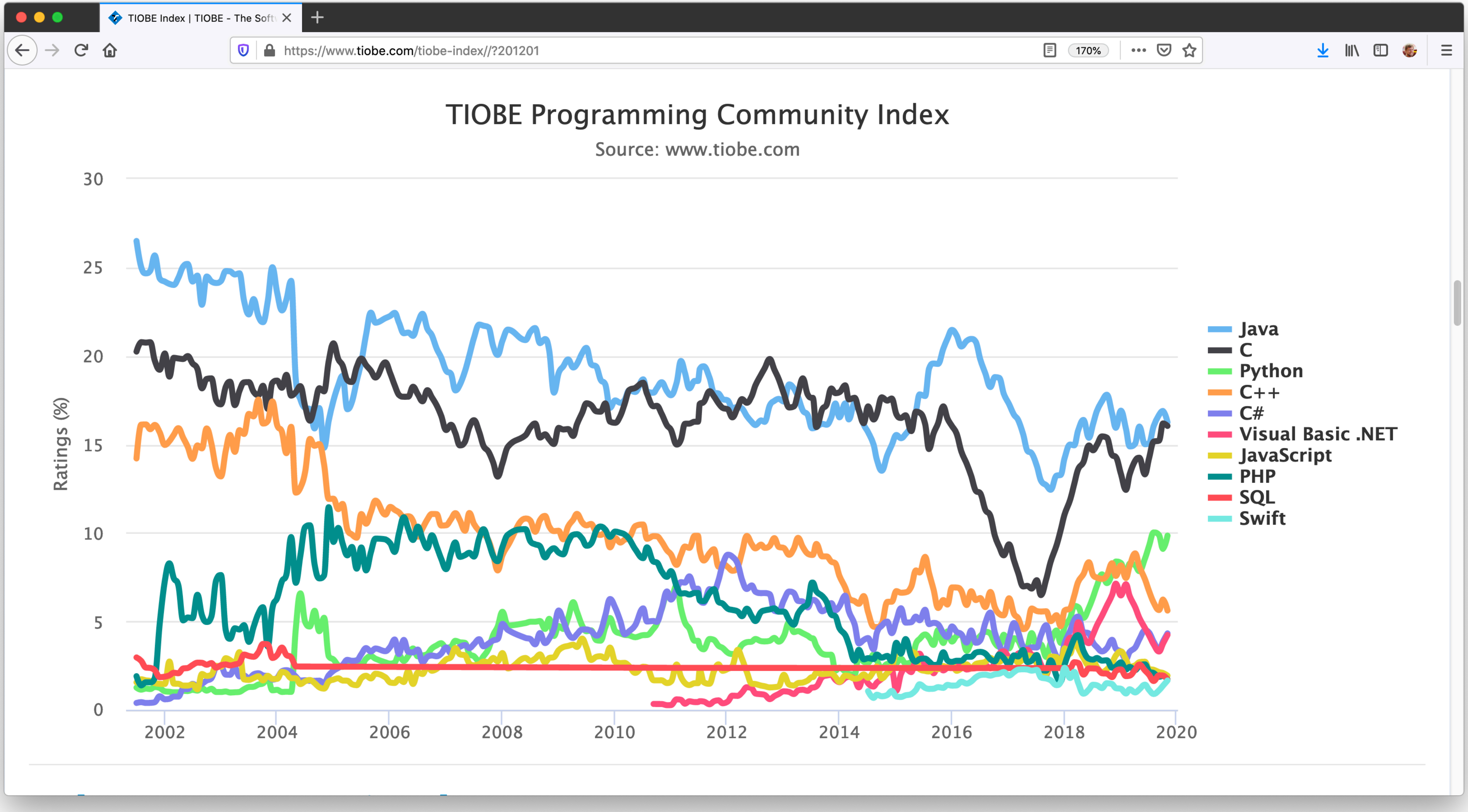
Python: an experiment in language design by community

Guido van Rossum
Principal Engineer, Dropbox (retired)
emeritus Python BDFL

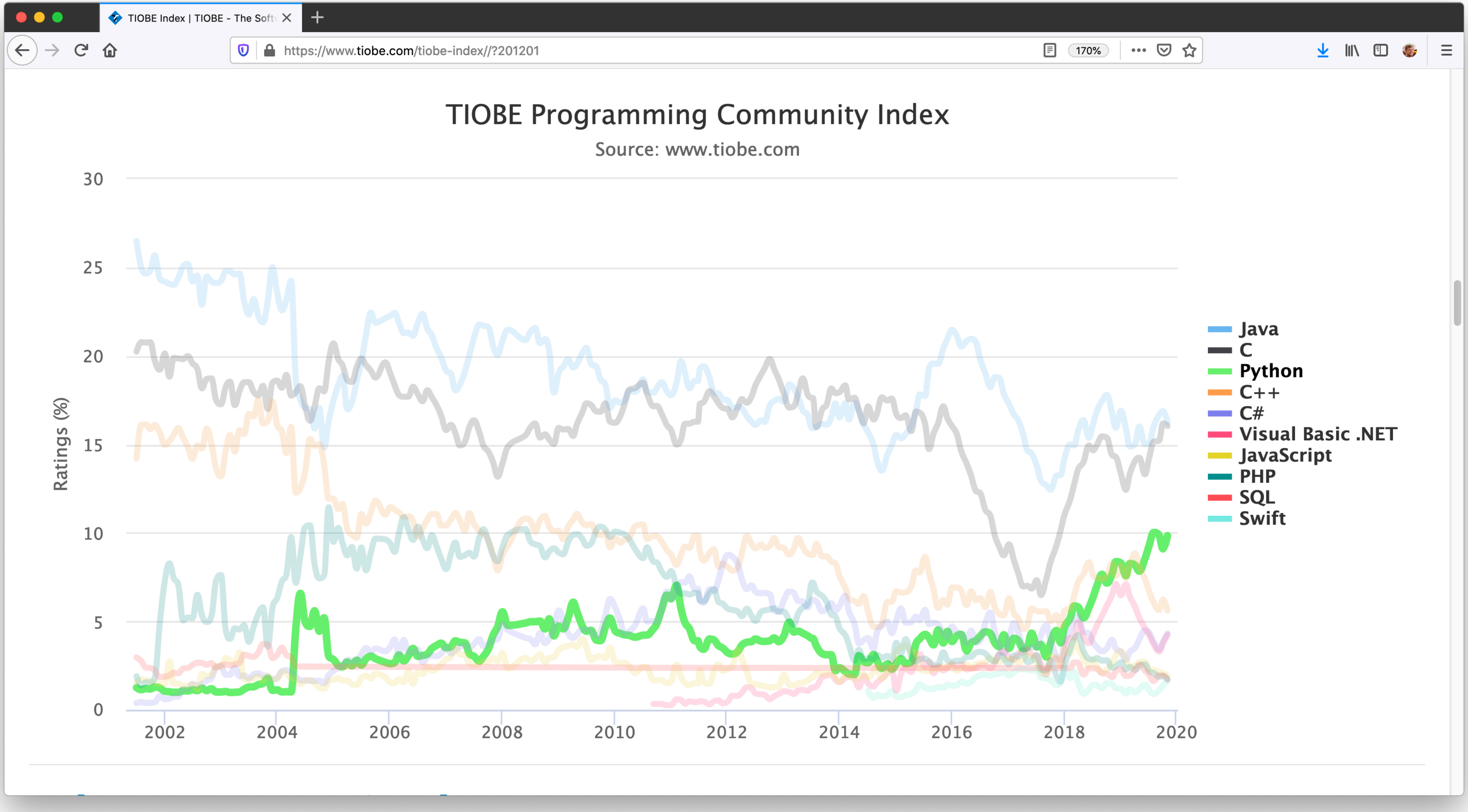
Why are we here?

"Let's Google that."



TIOBE Programming Community Index

Source: www.tiobe.com



TIOBE Programming Community Index

Source: www.tiobe.com

In other sites' words

- StackOverflow on https://insights.stackoverflow.com/survey/2019:

  "For the seventh year in a row, JavaScript is the most commonly used programming language, but Python has risen in the ranks again. This year, Python just edged out Java in overall ranking, much like it surpassed C# last year and PHP the year before. *Python is the fastest-growing major programming language today.*"

- GitHub on https://octoverse.github.com/ in 2019:

  "And for the first time, Python outranked Java as the second most popular language on GitHub by repository contributors."

Blah, blah, data science, blah

- GitHub again:

  "Behind Python’s growth is a speedily-expanding community of data science professionals and hobbyists—and the tools and frameworks they use every day. These include the many core data science packages powered by Python that are both lowering the barriers to data science work and proving foundational to projects in academia and companies alike."

Blah, blah, AI, blah, blah

- And GitHub again:

  "Beyond Python, repositories with topics like “deep learning”, “natural language processing”, and “machine learning” have become more popular over the years with growing communities focused on data science. Among the most popular (based on star counts) public repositories labelled with the topic, over half of them are built on numpy, and many of them depend on scipy, scikit-learn, and TensorFlow. We’ve also seen non-code contributions from the data science field, including academic papers."

It wasn't always that way!

- Python started with a more modest goal:

  Python is a simple, yet powerful programming language that bridges the gap between C and shell programming, and is thus ideally suited for "throw-away programming" and rapid prototyping.

  -- Python 0.9.8 reference manual (1992)

- "In the '90s, our goal was to be more popular than Perl."

  -- Jeremy Hylton, core developer since 1995

Early days

ABC

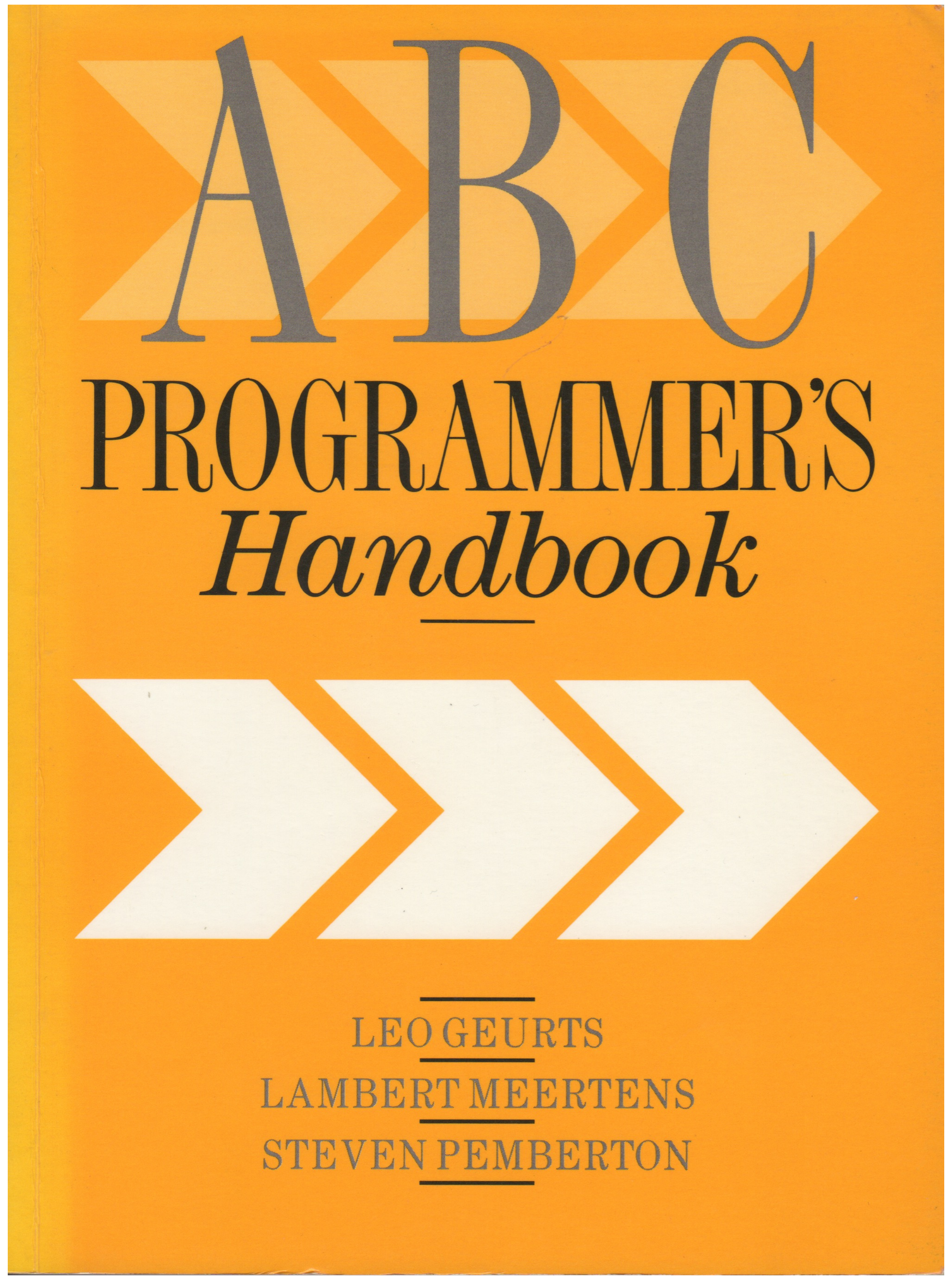
- Before Python, there was ABC

```
HOW TO PROCRASTINATE:
   FOR i IN {1..1000000000}:
      WRITE "Almost ready!" /
```

- Project ran from 1981-1987 (I think)

- Lambert Meertens, Leo Geurts, Steven Pemberton
- Timo Krijnen, Frank van Dijk, Guido van Rossum

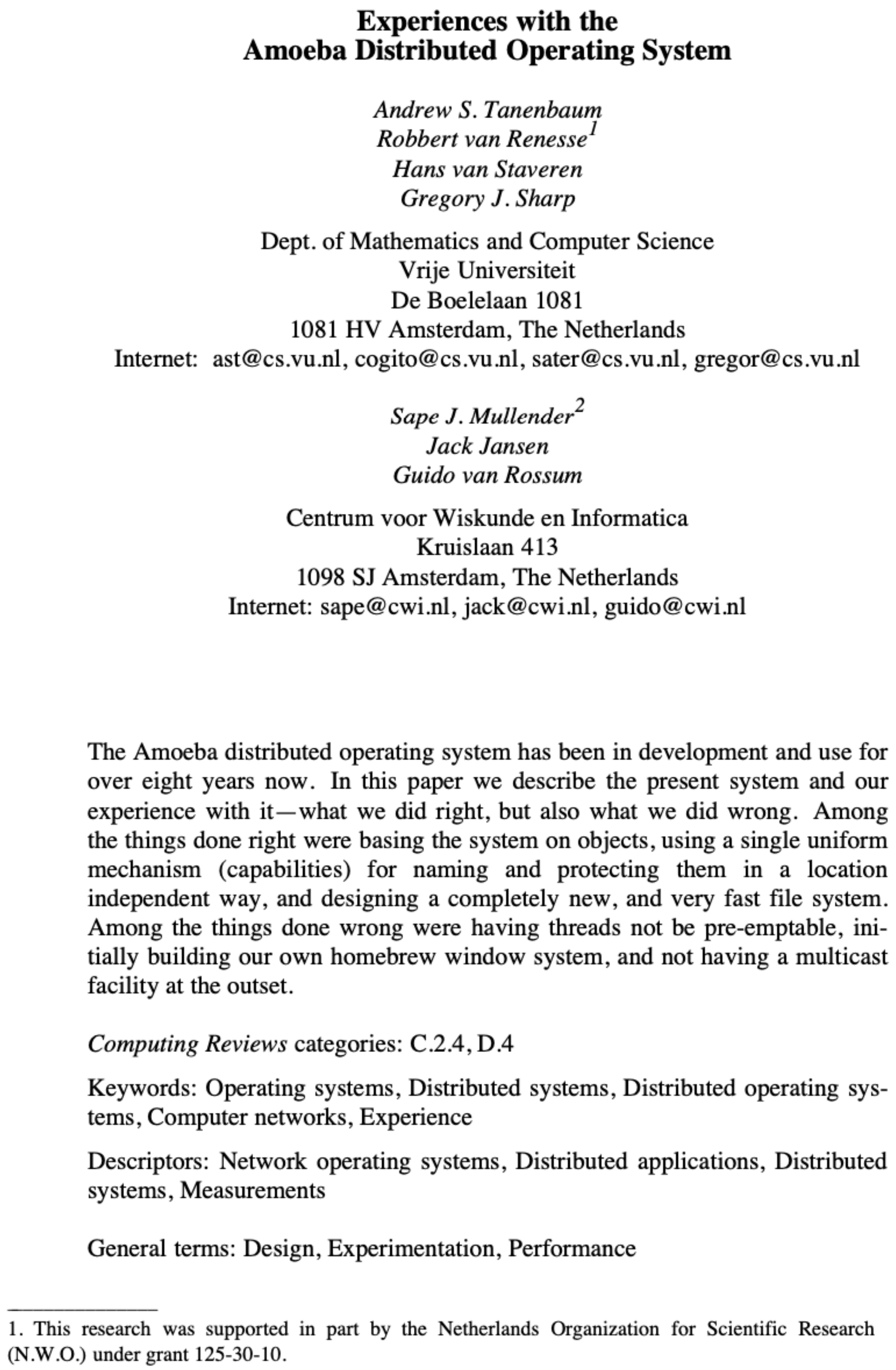
- Cancelled for lack of impact



Prentice Hall

Amoeba

- Distributed system project, started at Vrije Universiteit
  - Andrew Tanenbaum and his crew
- CWI team:
  - Dr. Sape Mullender
  - Jennifer Steiner, Jack Jansen, Sjoerd Mullender, myself
- We wanted to write a series of utilities and apps
- I got bored writing those in C



Experiences with the
Amoeba Distributed Operating System

Andrew S. Tanenbaum
Robbert van Renesse[1]
Hans van Staveren
Gregory J. Sharp

Dept. of Mathematics and Computer Science
Vrije Universiteit
De Boelelaan 1081
1081 HV Amsterdam, The Netherlands
Internet: ast@cs.vu.nl, cogito@cs.vu.nl, sater@cs.vu.nl, gregor@cs.vu.nl

Sape J. Mullender[2]
Jack Jansen
Guido van Rossum

Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam, The Netherlands
Internet: sape@cwi.nl, jack@cwi.nl, guido@cwi.nl

The Amoeba distributed operating system has been in development and use for over eight years now. In this paper we describe the present system and our experience with it—what we did right, but also what we did wrong. Among the things done right were basing the system on objects, using a single uniform mechanism (capabilities) for naming and protecting them in a location independent way, and designing a completely new, and very fast file system. Among the things done wrong were having threads not be pre-emptable, initially building our own homebrew window system, and not having a multicast facility at the outset.

Computing Reviews categories: C.2.4, D.4

Keywords: Operating systems, Distributed systems, Distributed operating systems, Computer networks, Experience

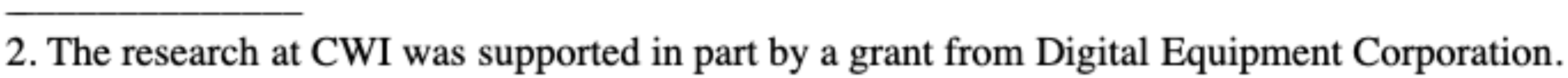
Descriptors: Network operating systems, Distributed applications, Distributed systems, Measurements

General terms: Design, Experimentation, Performance

[1] This research was supported in part by the Netherlands Organization for Scientific Research (N.W.O.) under grant 125-30-10.

Python!

- The way I recall it...
- Sape[2] was on a sabbatical in Palo Alto, Calif.
- Starting Xmas 1989 I spent most of my spare time on Python
- Used the UNIX "worse is better" philosophy
- First version was ready to demo in three months
- An early demo to Lambert went less than stellar :-)

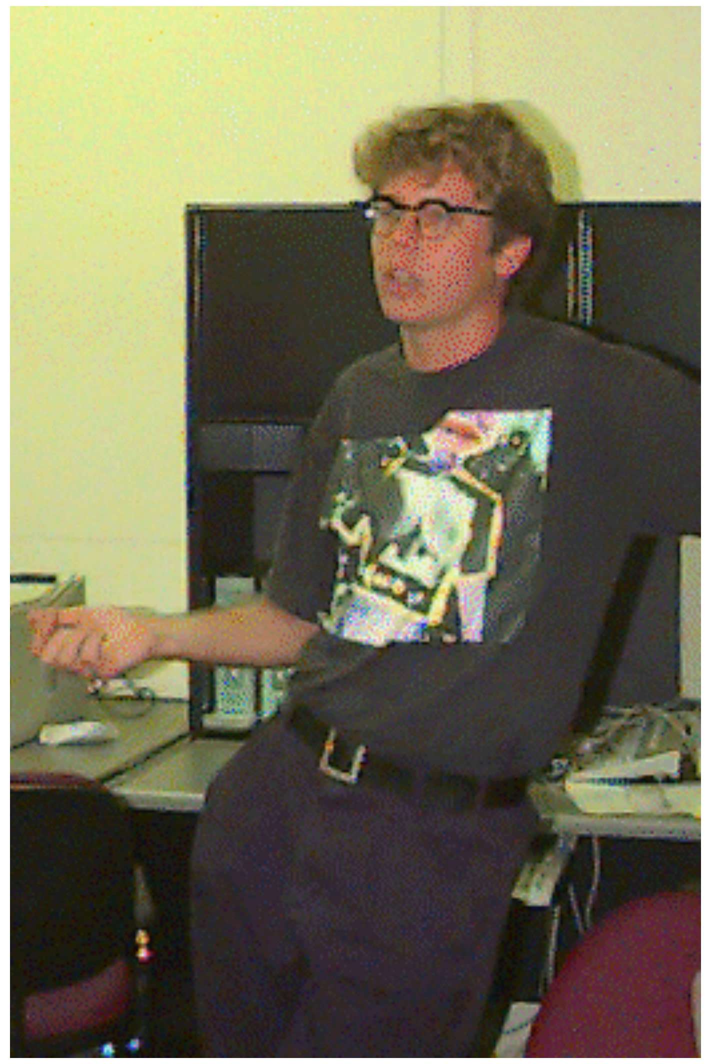


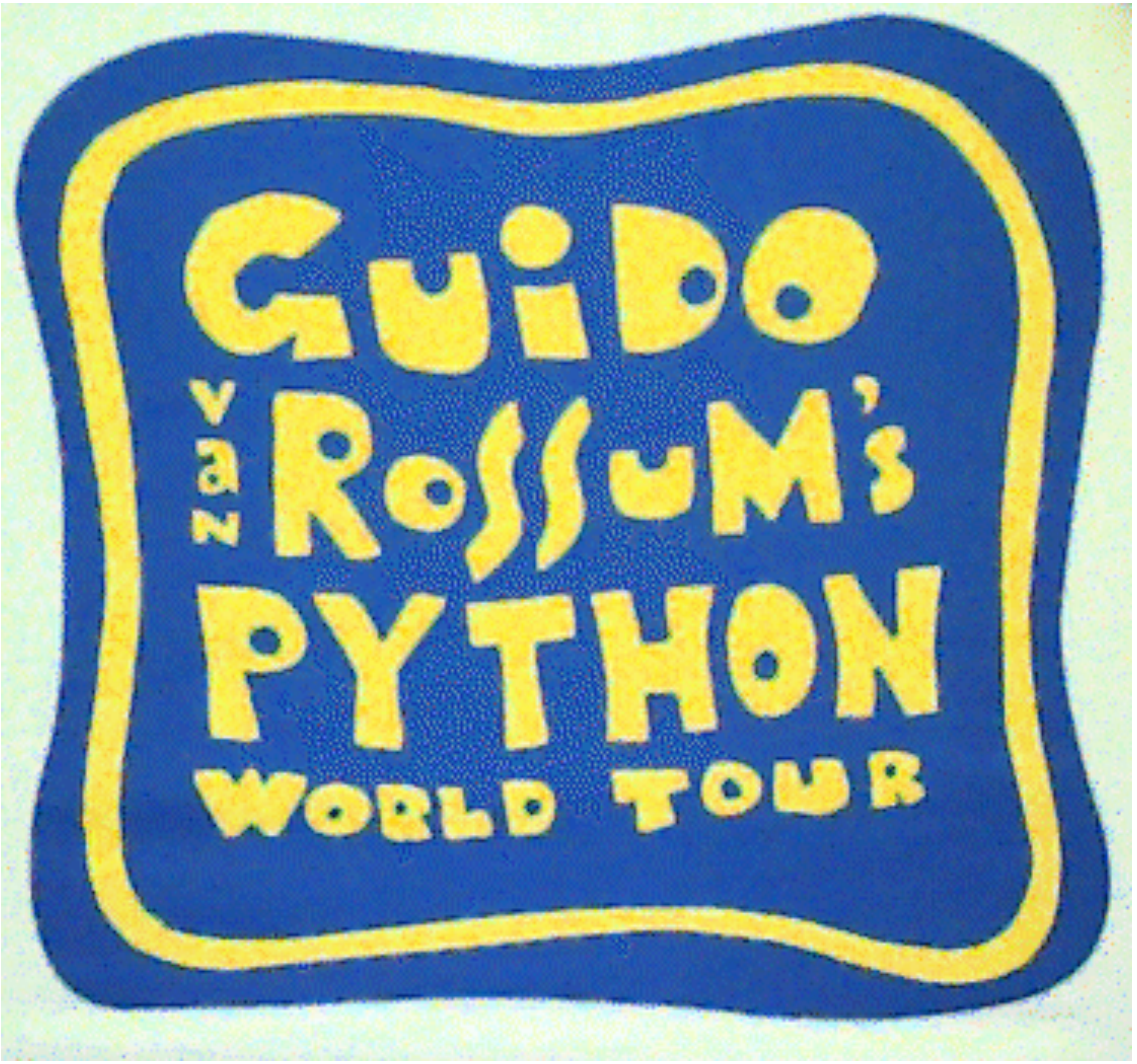
[2] The research at CWI was supported in part by a grant from Digital Equipment Corporation.

wikipedia.org

Three words: "community"*3

- Python spread like an oil slick:
- First Jack and Sjoerd
- Then other programmers and researchers at CWI
- Then USENET (first release on comp.sources 1991)
- Our own newsgroup: comp.lang.python (1994)
- Workshops and conferences (1994 onwards)





python.org

Python Enhancement Proposals (PEPs)

- Modeled after Internet RFC process, seen at IETF meetings
- Championed by Barry Warsaw (2000)
- Strike balance between BDFL and community input
- Anyone can propose, BDFL accepts or rejects...
- ...But only after ample debate!
- BDFL role is primarily review
- (And keeper of Python philosophy)

Intermezzo:
From 'for' to 'async def' in half a dozen PEPs

The lowly for-loop (1990)

- In the beginning was the for-loop

```
for x in a:
    print(x)
```

- Innovation: x iterates over the values in the sequence; avoid

```
for i := 1 to n do ...a[i]...
```

- The (bytecode) implementation does something like

```
for (int i = 0; i < len(a); i++) { x = a[i]; print(x); }
```

PEP 234: Iteration protocol (2001)

- Goal: allow iteration without a[i];
  move iteration state into Iterator object

-
  ```
  for line in file:
      process(line)
  ```

- Settled on a.__iter__() and it.next()

-
  ```
  it = file.__iter__()
  while True:
      try:
          line = it.next()
          process(line)
      except StopIteration:
          break
  ```

-
  ```
  class MyIterator:
      def __init__(self, a):
          self.a = a
          self.i = 0

      def next(self):
          if self.i >= len(self.a):
              raise StopIteration
          x = self.a[self.i]
          self.i += 1
          return x
  ```

-
  ```
  class MyList:
      def __iter__(self):
          return MyIterator(self)

      def __getitem__(self, i): ...
  ```

PEP 255: Generator functions (2001)

- Goal: avoid clunky iterator objects; allow user to move iteration state into local variables of a function

-
  ```
  def listiter(a):
      i = 0
      while i < len(a):
          yield a[i]
  ```

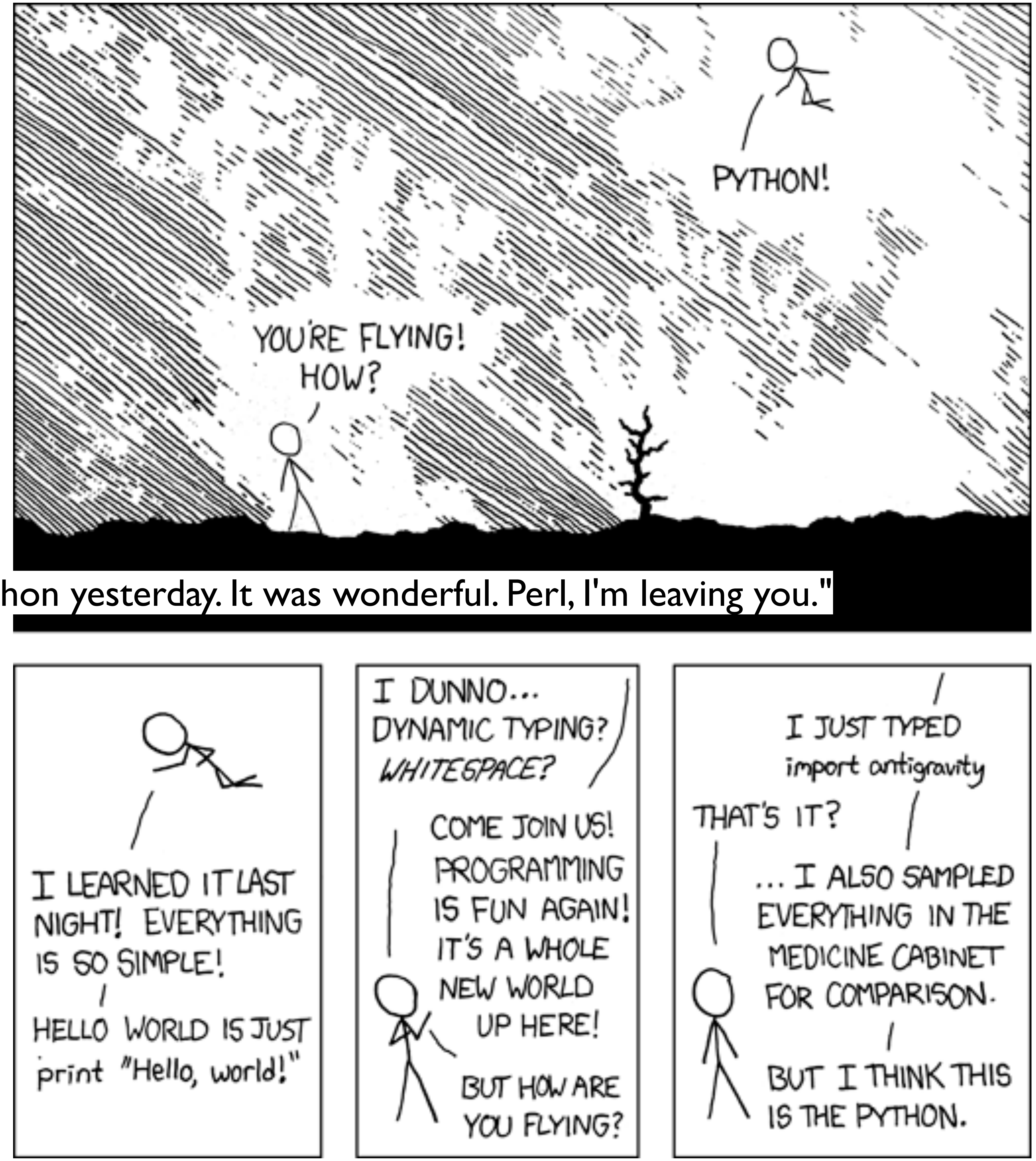
-
  ```
  class MyList:
      def __iter__(self):
          return listiter(self)
  ```

-
  ```
  def fileiter(f):
      while True:
          line = f.readline()
              if not line:
                  break
              yield line
  ```

PEP 342: Coroutines (2005)

- Goal: support use of generators as coroutines; allow callers to pass a value into the next iteration
- Also allow throwing exceptions into generators
- Ideas initially developed in PEP 288, PEP 325, PEP 334
- PEP 342 combined those ideas, superseding the PEPs
- Coroutines were adopted by Twisted and Tornado

2007


YOU'RE FLYING! HOW?
PYTHON!
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST
print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED
import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

"I wrote 20 short programs in Python yesterday. It was wonderful. Perl, I'm leaving you."

https://www.xkcd.com/353/

PEP 380: 'yield from' (2009-2011)

- Ostensibly just introduced a shorthand for

```
    for x in a:
        yield x
-->
    yield from a
```

- In reality created much more flexible coroutine machinery
- Initially targeted Python 2.7, finally landed in 3.3 (oops)
- Became the basis of PEP 3156: asyncio (2012-2013)

PEP 492: 'async def' (2015)

- Following the success of asyncio, streamline coroutines from

```
@coroutine
def read_text(s):
    data = yield from recv(s)
    return data.encode("utf-8")
```

-->

```
async def read_text(s):
    data = await recv(s)
    return data.encode("utf-8")
```

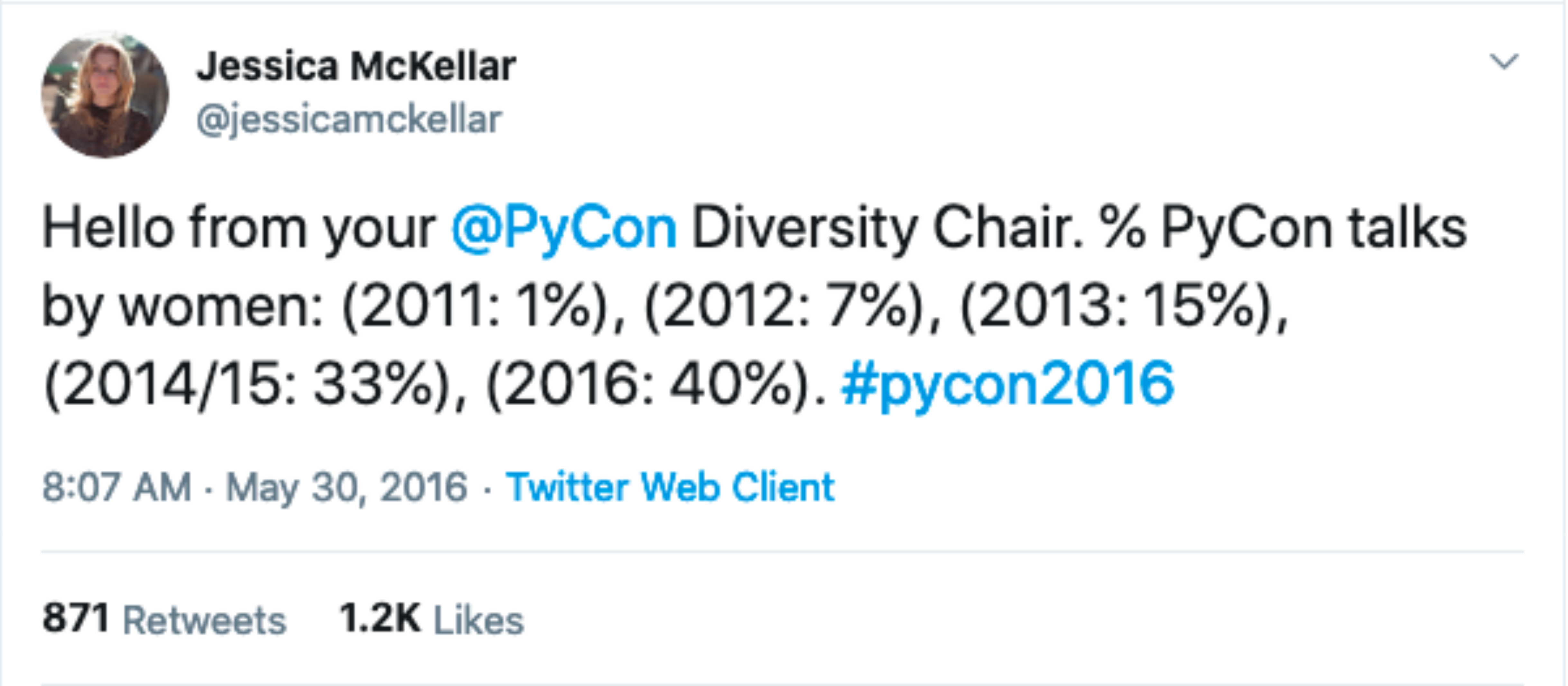
Moral

- Je weet nooit hoe een koe een haas vangt
- (A blind man may sometimes hit the crow)

http://ikbenpoeslief.blogspot.com/2012/04/hoe-een-koe-een-paashaas-vangt-2.html

Diversity

- I thought it wasn't an issue
- I didn't discriminate on race, gender, etc. ("meritocracy")
- So how did we end up with mostly white men?
- And what did we do about it?
- Mostly the work of a bunch of motivated women!
- PyLadies, Django Girls



Jessica McKellar
@jessicamckellar

Hello from your @PyCon Diversity Chair. % PyCon talks by women: (2011: 1%), (2012: 7%), (2013: 15%), (2014/15: 33%), (2016: 40%). #pycon2016

8:07 AM · May 30, 2016 · Twitter Web Client

871 Retweets 1.2K Likes

Python is for Girls!

https://esmithy.net/2016/06/01/pycon-2016-day-2/

Government

- 1995: Python Software Activity (PSA) "user group"

  "First Interim Chairman: Mike McLay
  First Interim Keepers of python.org: 1st interim board, @CNRI
  First Interim Keeper of the Notes: Ken Manheimer
  First Interim Keeper of the python.org Materials Index: Paul Everitt
  First Interim Treasurer: decision postponed until there's money
  First Interim Workshop Coordinator: Paul Everitt
  First Interim **Benevolent Dicator for Life**: Guido van Rossum"

- Most important activity: creation of python.org

- 2001: Python Software Foundation (PSF) incorporated

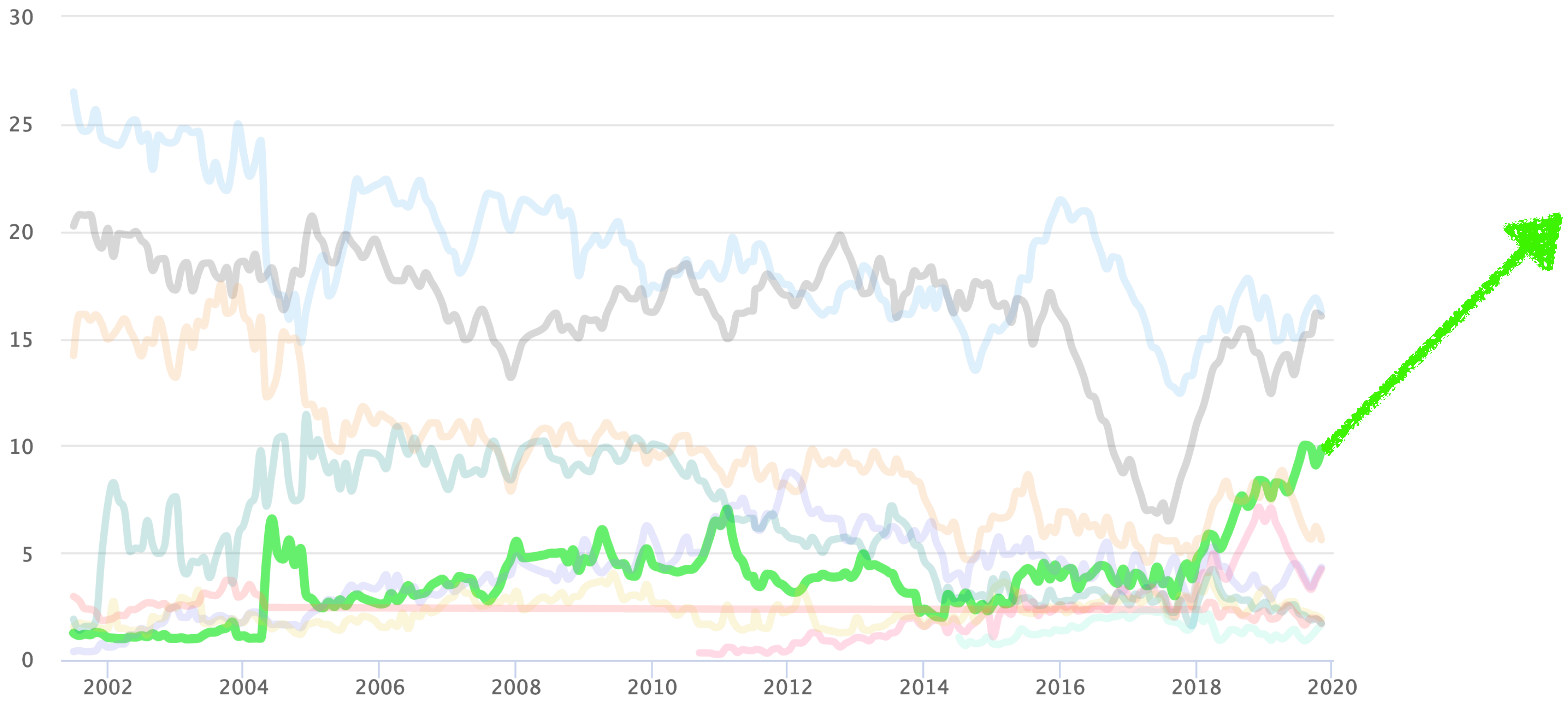
A new sheriff in town

- 2018: BDFL retirement (over PEP 572)
- 2019: Steering Council elected (5 people, guided by PEP 13)

Note:

- PSF and Steering Council are separate bodies
- PSF is for community in general (e.g. PyCon, licensing)
- SC guides development of language and implementation

The Future


30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020

The Zen of Python

By Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!