IPython

Enjoying interactive and high-level distributed computing in Python

Fernando Pérez

<Fernando.Perez@berkeley.edu>

Helen Wills Neuroscience Instiute, U.C. Berkeley

23andMe, Mountain View
September 24, 2008

Outline

1. Interactive Computing and Python
2. IPython: interactive++
3. Moving forward: IPython and parallel computing

Outline

1. Interactive Computing and Python
2. IPython: interactive++
3. Moving forward: IPython and parallel computing

Outline

1. Interactive Computing and Python
2. IPython: interactive++
3. Moving forward: IPython and parallel computing

Outline

1 Interactive Computing and Python

2 IPython: interactive++

3 Moving forward: IPython and parallel computing

Interactive computing?

- The computer as “microscope”
- Explore:
  - The language
  - Algorithms
  - Libraries
  - Data
  - Objects
- Debug
- Profile
- Interact with the OS, the network, etc.

Python
A good balance of features for this work style

- The interactive prompt: one of Python's greatest strengths.
  - But: it feels like a half-implemented idea.
- Dynamic typing: high "keystroke to power" ratio.
- Powerful introspection: `inspect` module, `foo.__dir__`, etc.
- Great string processing, OS access.
- Reloading of modules (though not perfect)
- Good GUI integration

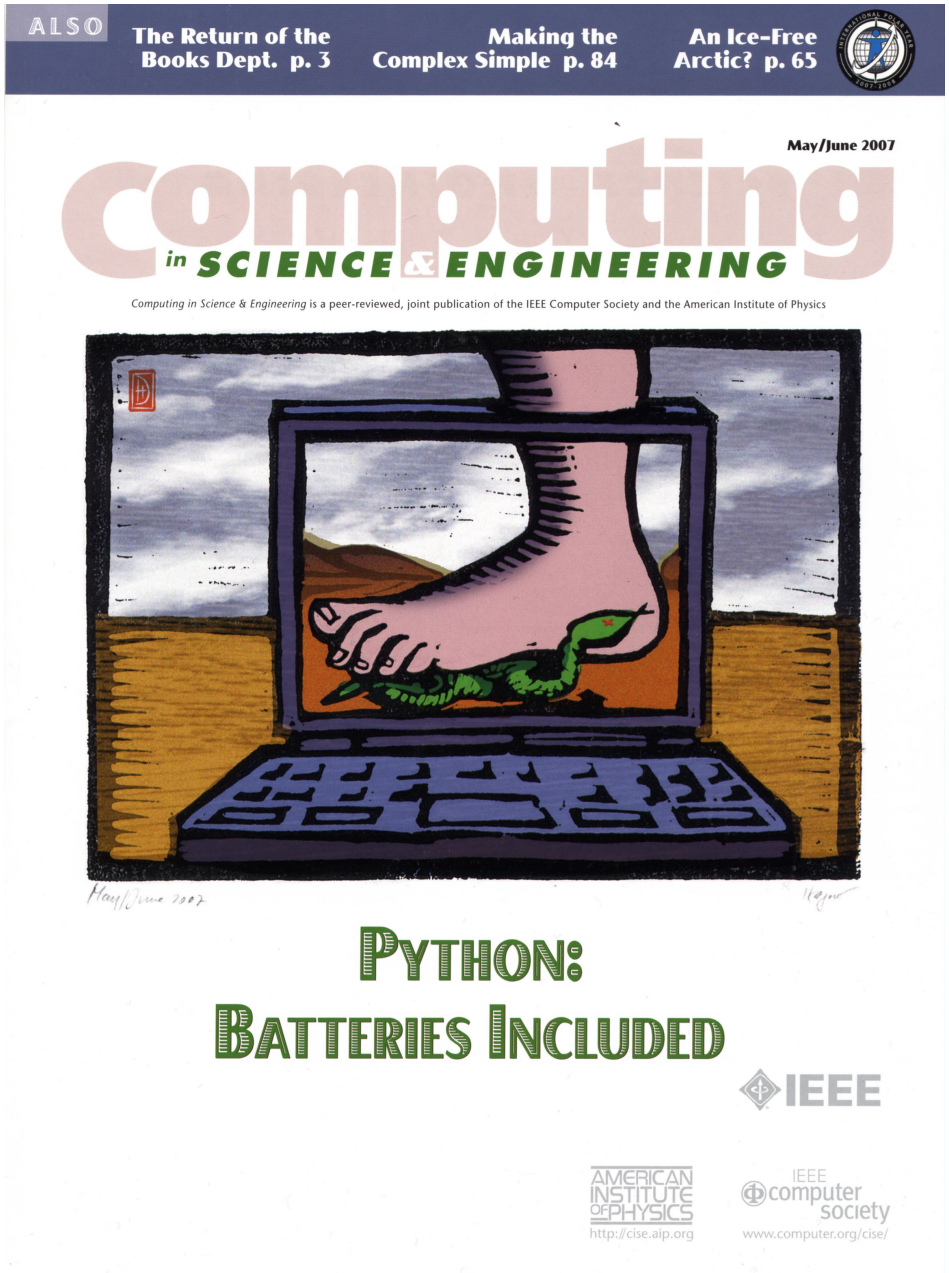
Not the famed 70's LISP machine, but we'll take what we can get...

Python
A good balance of features for this work style

- The interactive prompt: one of Python's greatest strengths.
  - But: it feels like a half-implemented idea.
- Dynamic typing: high "keystroke to power" ratio.
- Powerful introspection: `inspect` module, `foo.__dir__`, etc.
- Great string processing, OS access.
- Reloading of modules (though not perfect)
- Good GUI integration

Not the famed 70's LISP machine, but we'll take what we can get...

Python is growing in scientific computing



Outline

1 Interactive Computing and Python

2 IPython: interactive++

3 Moving forward: IPython and parallel computing

IPython? An overview

BSD tools for interactive work and high level parallel/distributed computing.

- A better Python shell: object introspection, system access, `%magic` commands, ...
- An embeddable interpreter: debugging, mixing batch-processing with interactive work.
- Flexible tools: easy to customize for interactive systems that may be Python-based but extend/modify the command set.

More recently, as we've tried to better generalize the key ideas:

- Talk to the keyboard: command line shell
- Talk to a GUI toolkit: embeddable interactive components
- Talk to the network: instant interactive distributed and parallel computing.

IPython? An overview

BSD tools for interactive work and high level parallel/distributed computing.

- A better Python shell: object introspection, system access, `%magic` commands, ...
- An embeddable interpreter: debugging, mixing batch-processing with interactive work.
- Flexible tools: easy to customize for interactive systems that may be Python-based but extend/modify the command set.

More recently, as we've tried to better generalize the key ideas:

- Talk to the keyboard: command line shell
- Talk to a GUI toolkit: embeddable interactive components
- Talk to the network: instant interactive distributed and parallel computing.

Cast of Characters

- **Brian Granger** - Physics, Cal State San Luis Obispo
- **Ville Vainio** - CS, Tampere University of Technology, Finland
- **Min Ragan-Kelley** - Plasma physics, UC Berkeley
- **Gael Varoquaux** - Neurospin (Orsay, France)
- **Robert Kern** - Enthought Inc (Austin, TX)
- **Stefan van der Walt** - Applied Math, U. Stellenbosch, South Africa
- **Barry Wark** - Neuroscience, U. Washington.
- **Jorgen Stenarson** - Sweden
- **Ondrej Certik** - Physics, Czech Republic
- **Laurent Dufrechou** - France
- **Vivian De Smedt** - Belgium
- Many more I'm forgetting...

Original design ideas

- Make every keystroke count: do the most with the least typing.
- Meta-control: `%magics` control and extend IPython itself.
- System access: why should your shell shield you from the OS?
- Efficient development:
  - Object introspection: TAB-completion, '`?`', '`??`', '`%p...`' functions.
  - Better tracebacks: colored, longer and with data details.
  - `%run: execfile()` on steroids.
  - Profiler: quick and easy profile access via `%prun` and `%run -p`.
  - Debugger: automatic `pdb` triggering on uncaught exceptions.
- We don't know what you want to do: easy to extend and customize.

Development features

- Code execution: `%run`

  `%run [options] your_file [args to your program]`

  - IPython's exception tracebacks.
  - Easy reloading of code (top-level modules, at least).
- The debugger: `%pdb` and %debug. Start pdb in post-mortem mode.
  - The pdb interactive prompt sees the local namespace.
  - Walk up and down the stack, print variables, call code, ...
  - Very efficient debugging.
- The profiler:
  - `%run -p`: profile complete programs.
  - `%prun`: profile single Python expressions (like function calls).
- Recursive reloading: `%dreload`. (Good, but not perfect)

OS access

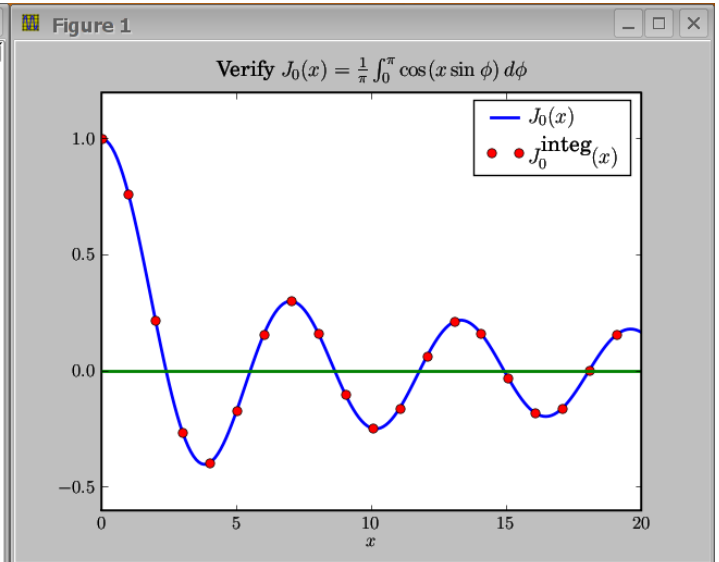
- Magics which mimic system commands (`%cd, %ls, %env, ...`)
- Support for directory traversal (`%cd, %dhist, %popd, %pushd, %dirs`).
- Define new system aliases with `%alias`
- `%bookmark` system: access to often visited directories.
- Lines starting with `'!'` go to the OS.
- `a=!ls *.py` to capture the output of `'ls *.py'`.
- `!cmd $foo` expands Python var `foo` for the shell.

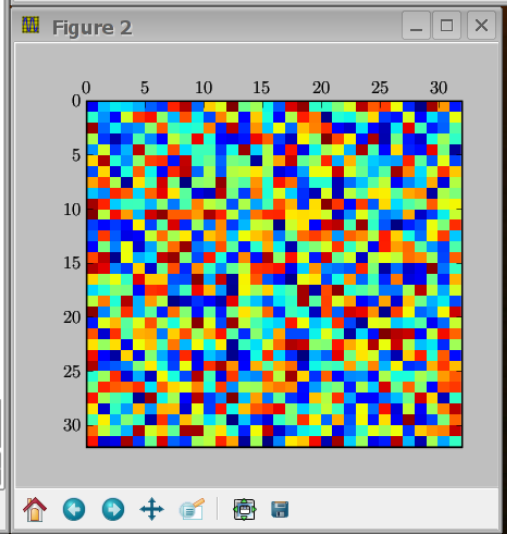
Matlab-like interactive usage

fperez@longs:/home/fperez - Shell - Konsole

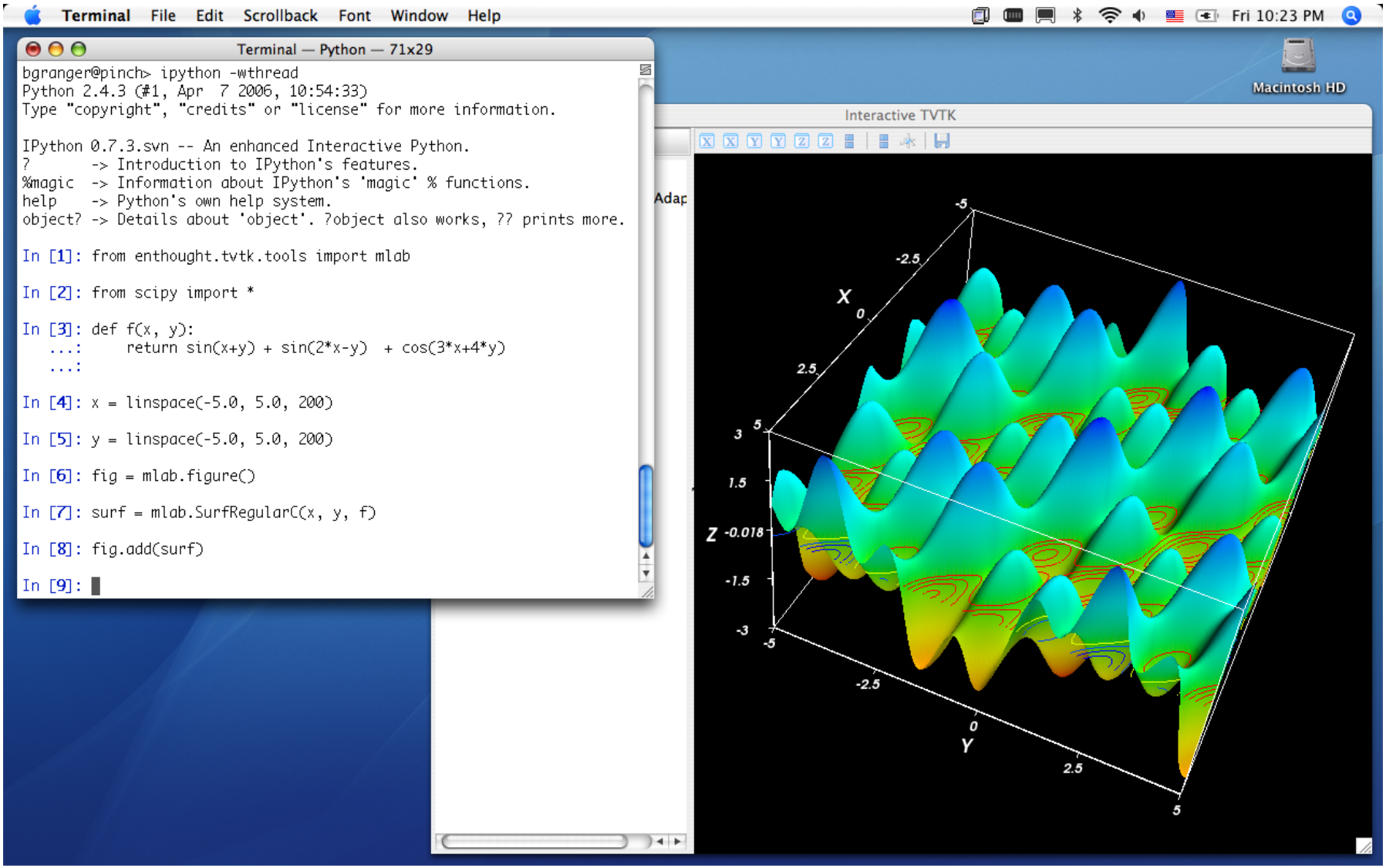
```
longs[~]> ipython -pylab
Python 2.4.3 (#2, Apr 27 2006, 14:43:58)
Type "copyright", "credits" or "license" for more information.

IPython 0.7.3.svn -- An enhanced Interactive Python.
?       -> Introduction to IPython's features.
%magic  -> Information about IPython's 'magic' % functions.
help    -> Python's own help system.
object? -> Details about 'object'. ?object also works, ?? prints more.

  Welcome to pylab, a matplotlib-based Python environment.
  For more information, type 'help(pylab)'.

In [1]: import math, numpy

In [2]: from scipy.integrate import quad

In [3]: from scipy.special import j0

In [4]: def j0i(x):
   ...:     """Integral form of J_0(x)"""
   ...:     def integrand(phi):
   ...:         return math.cos(x*math.sin(phi))
   ...:     return (1.0/math.pi)*quad(integrand,0,math.pi)[0]
   ...:

In [5]: x = numpy.linspace(0,20,200) # sample grid: 200 points between 0 and 20

In [6]: y = j0(x)  # sample J0 at all values of x

In [7]: x1 = x[::10]  # subsample the original grid every 10th point

In [8]: y1 = map(j0i,x1)  # evaluate the integral form at all points in x1

In [9]: # Make a plot with these values (the ; suppresses output)

In [10]: plot(x,y,label=r'$J_0(x)$');

In [11]: plot(x1,y1,'ro',label=r'$J_0^{\textrm{integ}}(x)$');

In [12]: axhline(0,color='green',label='_nolegend_');

In [13]: title(r'Verify $J_0(x)=\frac{1}{\pi}\int_0^{\pi}\cos(x \sin\phi)\,d\phi$');

In [14]: xlabel('$x$');

In [15]: legend();

In [16]: matshow(numpy.random.random((32,32)))
Out[16]: <matplotlib.figure.Figure instance at 0x4630042c>
```

Shell





Sophisticated 3d visualizations with VTK



Who uses IPython?

IPython is available for all Linux distributions, OS X and Windows.

- **SAGE**: open source mathematics, very ambitious project (U. Washington).
- **PyRAF**: astronomical image analysis (Space Telescope Science Institute).
- **CASA**: Common Astronomy Software Applications (National Radio Astronomy Observatory).
- **Ganga**: job control for the LHCb and ATLAS experiments at CERN.
- **PyMAD**: a neutron spectrometer at the Institut Laue Langevin (Grenoble).
- **Pymerase**: microarray gene expression databases.
- Many others...

Outline

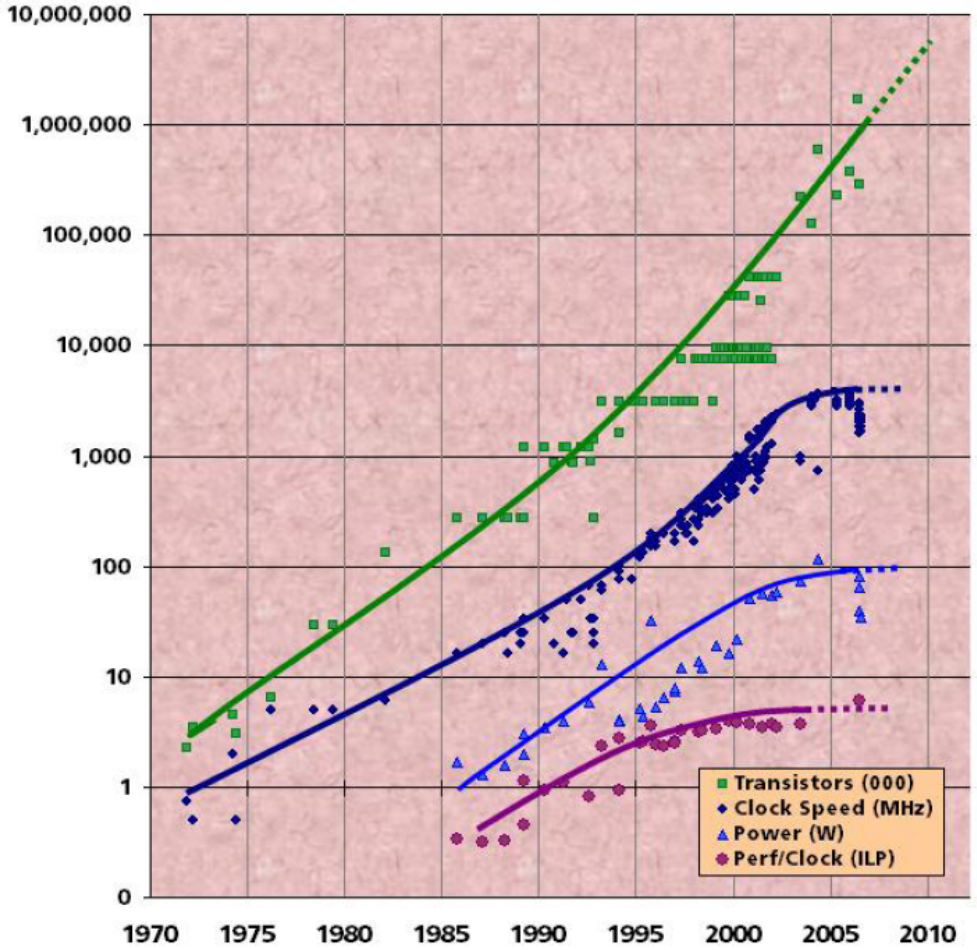
1 Interactive Computing and Python

2 IPython: interactive++

3 Moving forward: IPython and parallel computing

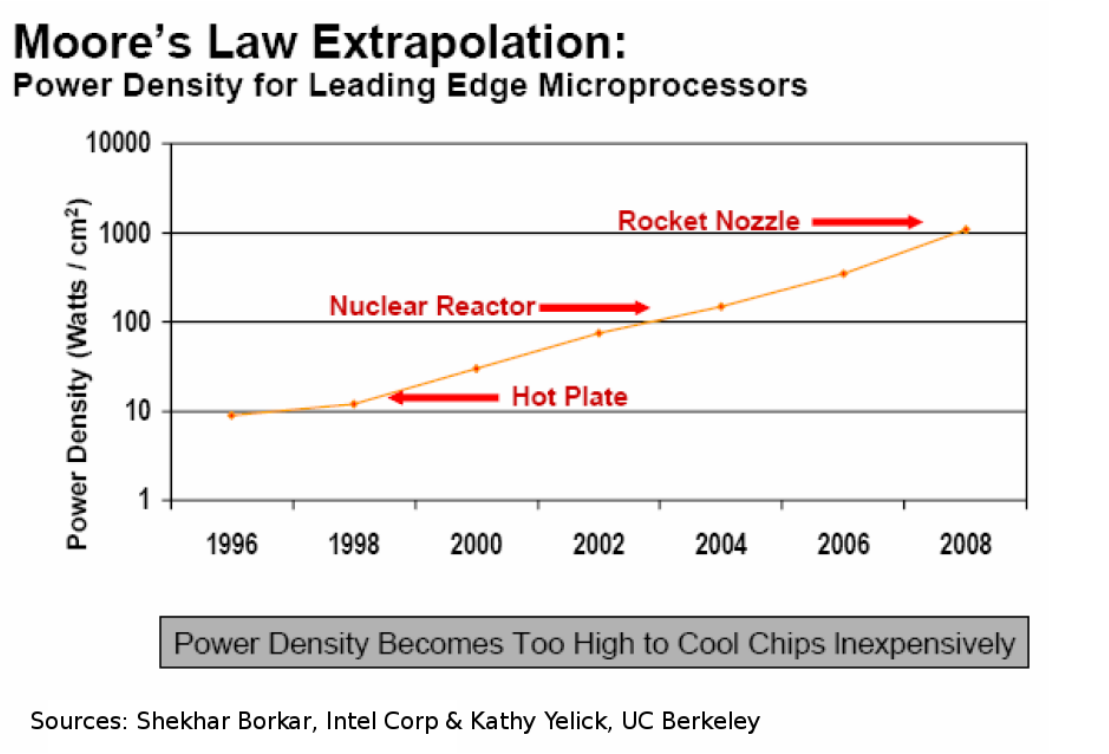
Parallel computing: why should we care?

Because reality looks like this:



Sources: Intel, Microsoft (Sutter), Stanford (Olukotun, Hammond) & Berkeley (Yelick)

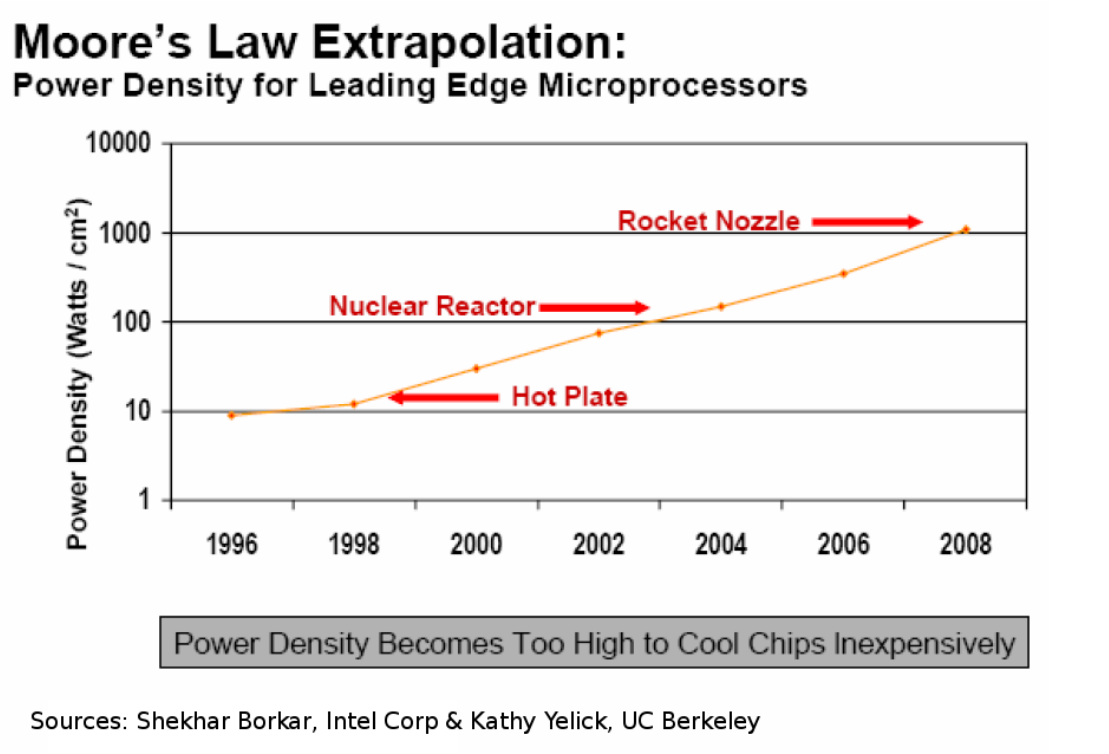
We can't escape thermodynamics



The vendor's solutions: Heterogeneous and complex landscape!

- Multicore chips: everywhere now.
- GPUs for general computing: $>$ 128 'processors' per card.
- High-density clusters: SiCortex ($>$ 5000 processors in a cabinet).

We can't escape thermodynamics



The vendor's solutions: Heterogeneous and complex landscape!

- Multicore chips: everywhere now.
- GPUs for general computing: $>$ 128 'processors' per card.
- High-density clusters: SiCortex ($>$ 5000 processors in a cabinet).

Parallel programming?

There are plenty of bad news

- It is in general, *extremely* difficult.
- Productivity plummets with enormous up-front efforts.
- Development, debugging, running is hard and cumbersome.
- With propietary tools, licensing costs can go through the roof.

But not all is doom and gloom

- Many problems are *embarrassingly parallel*: uncoupled components.
- This is common in science:
  - Analyze many data sets with the same algorithm.
  - Global parameter searches.
- Even not-so-embarrassingly parallel problems can be tractable...
  - ... with the right tools.

Parallel programming?

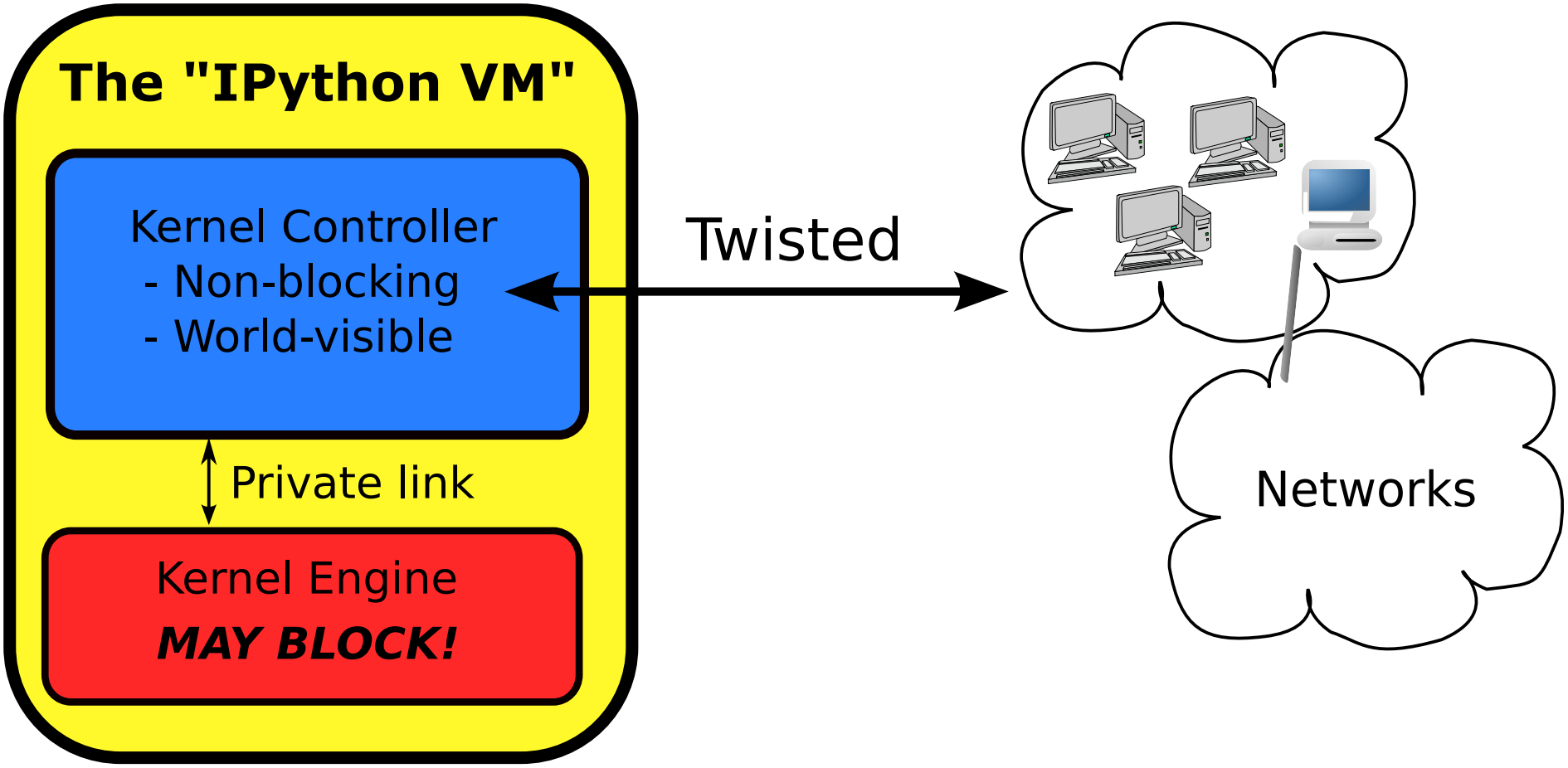
There are plenty of bad news

- It is in general, *extremely* difficult.
- Productivity plummets with enormous up-front efforts.
- Development, debugging, running is hard and cumbersome.
- With propietary tools, licensing costs can go through the roof.

But not all is doom and gloom

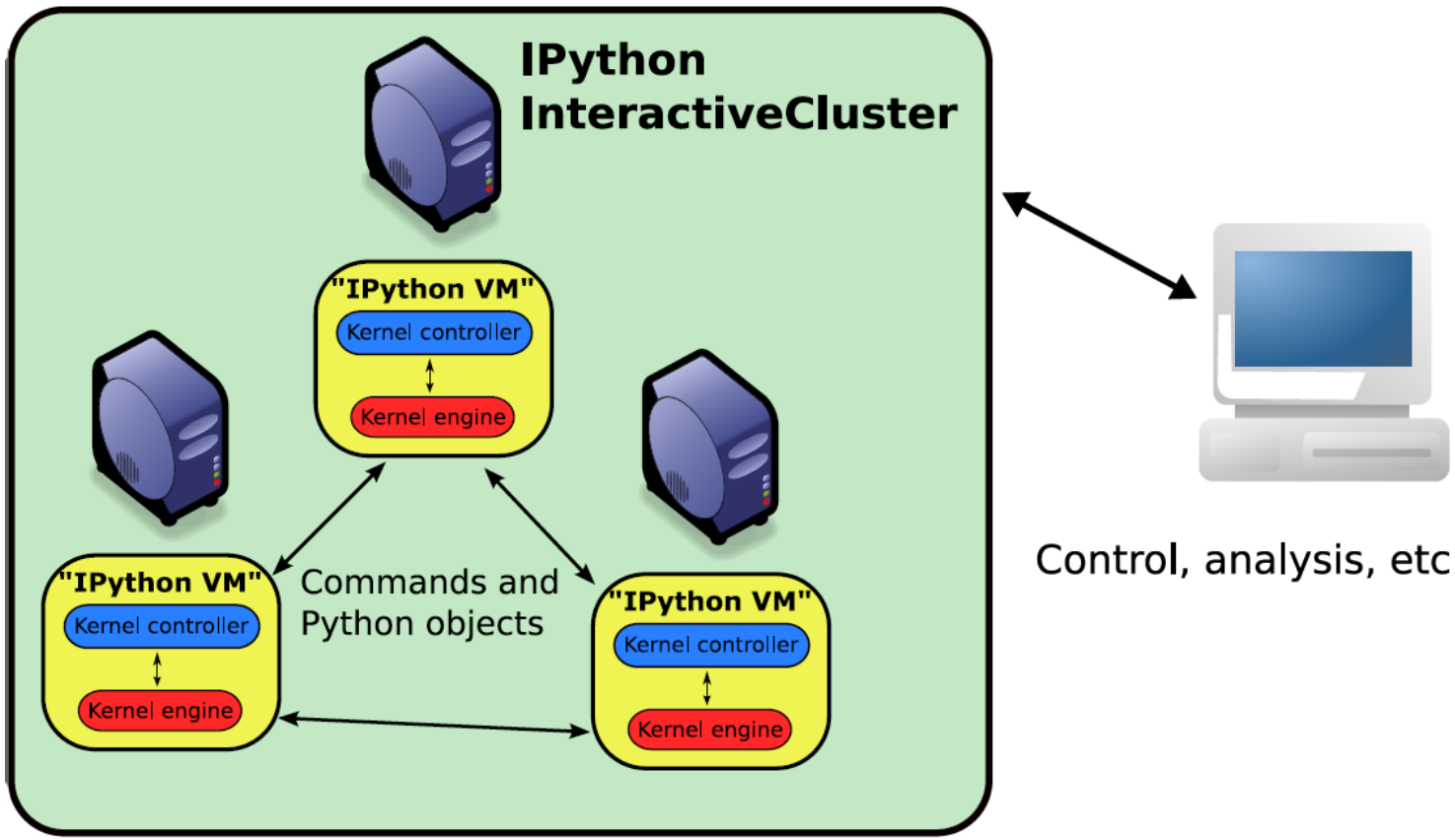
- Many problems are *embarrassingly parallel*: uncoupled components.
- This is common in science:
  - Analyze many data sets with the same algorithm.
  - Global parameter searches.
- Even not-so-embarrassingly parallel problems can be tractable...
  - ... with the right tools.

Network-aware IPython



IPython for distributed and parallel computing

- Think of Python as ‘the CPU’.
- IPython abstracts them over the network.
- Use interactively **or not**.



What does IPython offer here?

- Easy reuse and distribution of existing serial ('normal') codes.
- High-level abstractions for 'embarrassingly parallel' problems.
  - Direct execution of code over the network: `multiengine` interface.
  - Out-of-the box *task farming* tools: `task` interface.
- Task farming system is low-latency.
  - can be integrated into more complex codes.
- Implement any approach to parallelism you want:
  - Synchronous or asynchronous execution of code on nodes.
  - Task farming.
  - Traditional Message Passing (MPI).
  - Integrate hybrid codes.
  - BYO.
- Actively developed.

Some technical notes

- Networking: Twisted
- High-level interfaces: no need to learn Twisted.
- RPC: Twisted's `foolscap`.
- Security: foolscap supports SSL (pyOpenSSL) and encodes IP/port/service info into "FURLS" (foolscap URLS), akin to SSH keys.
  - Review/improvements welcome, we're not security experts!
- MPI support is there, use mpi4py bindings.
- Integration with queuing systems, better process control coming: next release.

Status?

- Actively developed
- Release 0.9.1 made recently, working on 0.10
- Launchpad/Bazaar: distributed VC, bugs, code review
- Sphinx: documentation.
- Nose/Twisted.trial: testing
- WX and Cocoa shells in prototype
- More and better docs...
- Cleaner process control, PBS (queuing) integration.
- Graphical programming for job control.
- More developers welcome!

A brief demo and questions...