Interactive scientific computing environments

IPython's design and future directions

Fernando Pérez[1]
Brian Granger[2]
Robert Kern[3]

[1]Applied Math, U. Colorado; [2]Physics, U. Santa Clara; [3] Enthought.

SAGE Days, UCSD
February 4, 2006

Outline

Outline

Outline

Outline

IPython: 2001-Today

- Nov. 2001: I discover Python. Fun, interactive, but I miss the Mathematica and IDL prompts.
- `sys.ps{1,2}` and `sys.displayhook`: the door to cached prompts.
- LazyPython (Nathan Gray, Caltech) and IPP - Interactive Python Prompt (Janko Hauser, at a German oceanographic institute).
- Six intense weeks and 7000 LOC later: IPython 0.2.0 is announced. A blend of my own IPython and LazyPython, inserted into IPP.
- IPP subclassed `code.InteractiveConsole`. So did we until v. 0.7.0.

SAGE & IPython

Scientific computing is *exploratory*: a good interactive environment is a *necessity*.

What is IPython?

- Besides a (much) better interactive shell, with threading support (GTK, WX and Qt), system access, etc...
- It is a base layer for building customized interactive environments.

How?

- Input preprocessing: $2/3 \rightarrow ZZ(2)/ZZ(3)$
- Output preprocessing.
- Customized tab-completers: PyMad (Institut Laue Langevin-CEA Grenoble), tab-completion over the network for proxied objects.
- Customized exception handlers. And a lot more...

SAGE: the first widely available system to really take advantage this (others: Ganga/Clip-CERN, CASA-NRAO, iPymerase, PyMad,...)

SAGE & IPython

Scientific computing is *exploratory*: a good interactive environment is a *necessity*.

What is IPython?

- Besides a (much) better interactive shell, with threading support (GTK, WX and Qt), system access, etc...
- It is a base layer for building customized interactive environments.

How?

- Input preprocessing: $2/3 \to ZZ(2)/ZZ(3)$
- Output preprocessing.
- Customized tab-completers: PyMad (Institut Laue Langevin-CEA Grenoble), tab-completion over the network for proxied objects.
- Customized exception handlers. And a lot more...

SAGE: the first widely available system to really take advantage this (others: Ganga/Clip-CERN, CASA-NRAO, iPymerase, PyMad,...)

SAGE & IPython

Scientific computing is *exploratory*: a good interactive environment is a *necessity*.

What is IPython?

- Besides a (much) better interactive shell, with threading support (GTK, WX and Qt), system access, etc...
- It is a base layer for building customized interactive environments.

How?

- Input preprocessing: $2/3 \rightarrow ZZ(2)/ZZ(3)$
- Output preprocessing.
- Customized tab-completers: PyMad (Institut Laue Langevin-CEA Grenoble), tab-completion over the network for proxied objects.
- Customized exception handlers. And a lot more...

SAGE: the first widely available system to really take advantage this (others: Ganga/Clip-CERN, CASA-NRAO, iPymerase, PyMad,...)

A few useful interactive tools

- `%edit`: call your favorite `$EDITOR` on the spot.
- `%macro`: interactively recall groups of lines quickly (you can `%edit` them)
- `%save`: save a group of lines to a named file.
- `%store`: lightweight persistence for any variable (including macros).
- `%pdb`: automatic invocation of a debugger (IPython-enhanced pdb)
- Embedding IPython: open an interactive shell inside any program you want.
- Shell access: direct access to the underlying OS. Use Python for shell-like tasks (*much* nicer syntax than bash).

A few useful interactive tools

- `%edit`: call your favorite `$EDITOR` on the spot.
- `%macro`: interactively recall groups of lines quickly (you can `%edit` them)
- `%save`: save a group of lines to a named file.
- `%store`: lightweight persistence for any variable (including macros).
- `%pdb`: automatic invocation of a debugger (IPython-enhanced pdb)
- Embedding IPython: open an interactive shell inside any program you want.
- Shell access: direct access to the underlying OS. Use Python for shell-like tasks (*much* nicer syntax than bash).

A few useful interactive tools

- `%edit`: call your favorite `$EDITOR` on the spot.
- `%macro`: interactively recall groups of lines quickly (you can `%edit` them)
- `%save`: save a group of lines to a named file.
- `%store`: lightweight persistence for any variable (including macros).
- `%pdb`: automatic invocation of a debugger (IPython-enhanced pdb)
- Embedding IPython: open an interactive shell inside any program you want
- Shell access: direct access to the underlying OS. Use Python for shell-like tasks (*much* nicer syntax than bash).

A few useful interactive tools

- `%edit`: call your favorite `$EDITOR` on the spot.
- `%macro`: interactively recall groups of lines quickly (you can `%edit` them)
- `%save`: save a group of lines to a named file.
- `%store`: lightweight persistence for any variable (including macros).
- `%pdb`: automatic invocation of a debugger (IPython-enhanced pdb)
- Embedding IPython: open an interactive shell inside any program you want
- Shell access: direct access to the underlying OS. Use Python for shell-like tasks (*much* nicer syntax than bash).

A few useful interactive tools

- `%edit`: call your favorite `$EDITOR` on the spot.
- `%macro`: interactively recall groups of lines quickly (you can `%edit` them)
- `%save`: save a group of lines to a named file.
- `%store`: lightweight persistence for any variable (including macros).
- `%pdb`: automatic invocation of a debugger (IPython-enhanced pdb)
- Embedding IPython: open an interactive shell inside any program you want
- Shell access: direct access to the underlying OS. Use Python for shell-like tasks (*much* nicer syntax than bash).

A few useful interactive tools

- `%edit`: call your favorite `$EDITOR` on the spot.
- `%macro`: interactively recall groups of lines quickly (you can `%edit` them)
- `%save`: save a group of lines to a named file.
- `%store`: lightweight persistence for any variable (including macros).
- `%pdb`: automatic invocation of a debugger (IPython-enhanced pdb)
- Embedding IPython: open an interactive shell inside any program you want
- Shell access: direct access to the underlying OS. Use Python for shell-like tasks (*much* nicer syntax than bash).

A few useful interactive tools

- `%edit`: call your favorite `$EDITOR` on the spot.
- `%macro`: interactively recall groups of lines quickly (you can `%edit` them)
- `%save`: save a group of lines to a named file.
- `%store`: lightweight persistence for any variable (including macros).
- `%pdb`: automatic invocation of a debugger (IPython-enhanced pdb)
- Embedding IPython: open an interactive shell inside any program you want
- Shell access: direct access to the underlying OS. Use Python for shell-like tasks (*much* nicer syntax than bash).

Outline

Lessons from IPython

- IPython has been pushed by some almost to replace Python as the user-visible system (better exceptions, control via `%magics`, etc.)
- If everyone wants it so bad, let's give it to them!
- Any good interactive system should have *two* levels of operation:
  - The actual execution language: in our case, Python.
  - A control mechanism: the `%magics`.
- But we'll make all of this available *over the network.*
- And it will be *non-blocking* (so your extension C code doesn't freeze your sessions).
- We're going to clarify, organize and improve all the public extension points, so extension authors (e.g. SAGE) have an easier time.

Lessons from IPython

- IPython has been pushed by some almost to replace Python as the user-visible system (better exceptions, control via `%magics`, etc.)
- If everyone wants it so bad, let's give it to them!
- Any good interactive system should have *two* levels of operation:
  - The actual execution language: in our case, Python.
  - A control mechanism: the `%magics`.
- But we'll make all of this available *over the network.*
- And it will be *non-blocking* (so your extension C code doesn't freeze your sessions).
- We're going to clarify, organize and improve all the public extension points, so extension authors (e.g. SAGE) have an easier time.

Lessons from IPython

- IPython has been pushed by some almost to replace Python as the user-visible system (better exceptions, control via `%magics`, etc.)
- If everyone wants it so bad, let's give it to them!
- Any good interactive system should have *two* levels of operation:
  - The actual execution language: in our case, Python.
  - A control mechanism: the `%magics`.
- But we'll make all of this available *over the network.*
- And it will be *non-blocking* (so your extension C code doesn't freeze your sessions).
- We're going to clarify, organize and improve all the public extension points, so extension authors (e.g. SAGE) have an easier time.

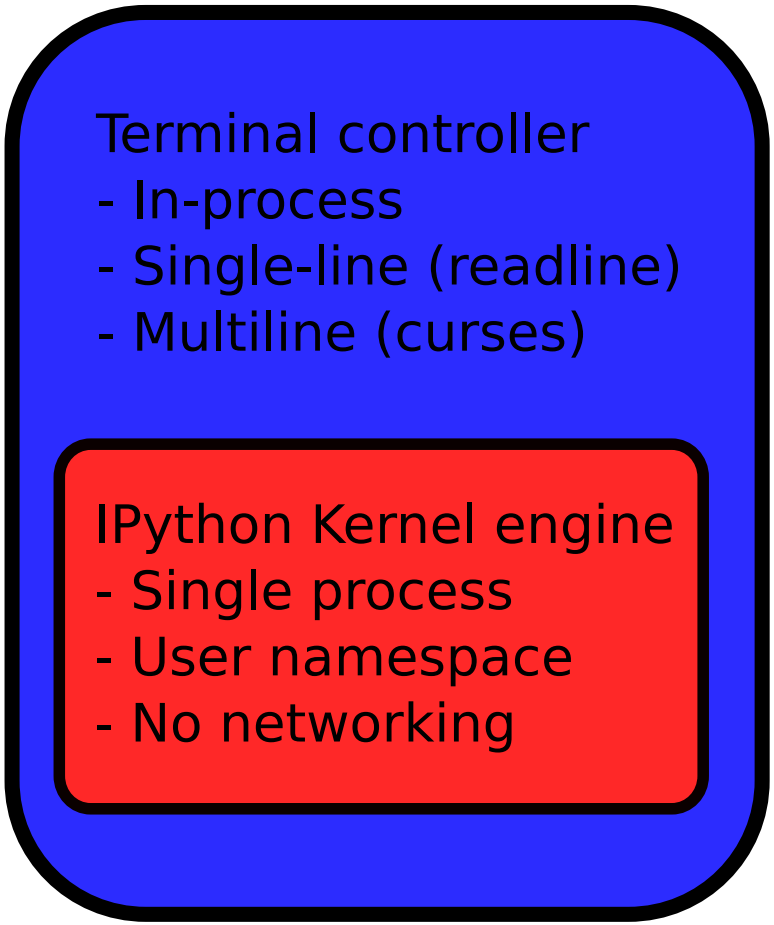
The IPython Kernel

- The kernel is an IPython instance that listens on a network port rather than to an interactive prompt.
- It has a control protocol for commands.
- And it can also pass any object which can be serialized (pickle for now).
- Multi-threaded with an execution queue.
- Developed using Twisted and non-blocking sockets.
- Can be started at any time using various means (SSH, Xgrid, GridEngine, Condor, etc.)
- Eventually, this kernel will be the core of IPython.

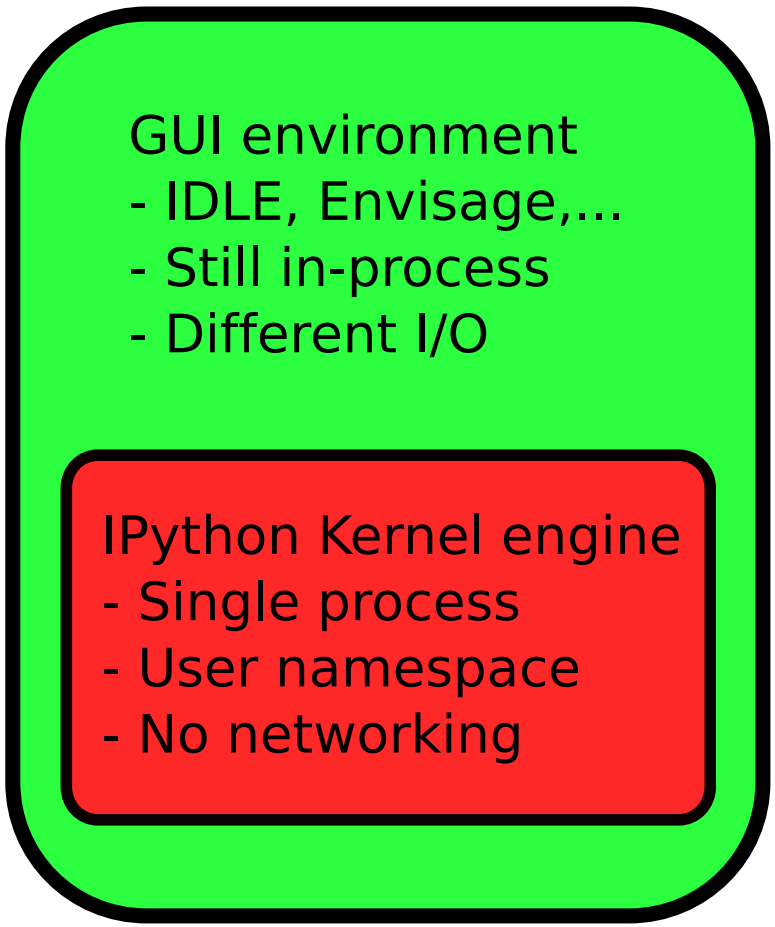
IPython as we know it will continue to exist

But better, cleaner, and embeddable in GUIs

Today's IPython



What we wish we could do with today's IPython

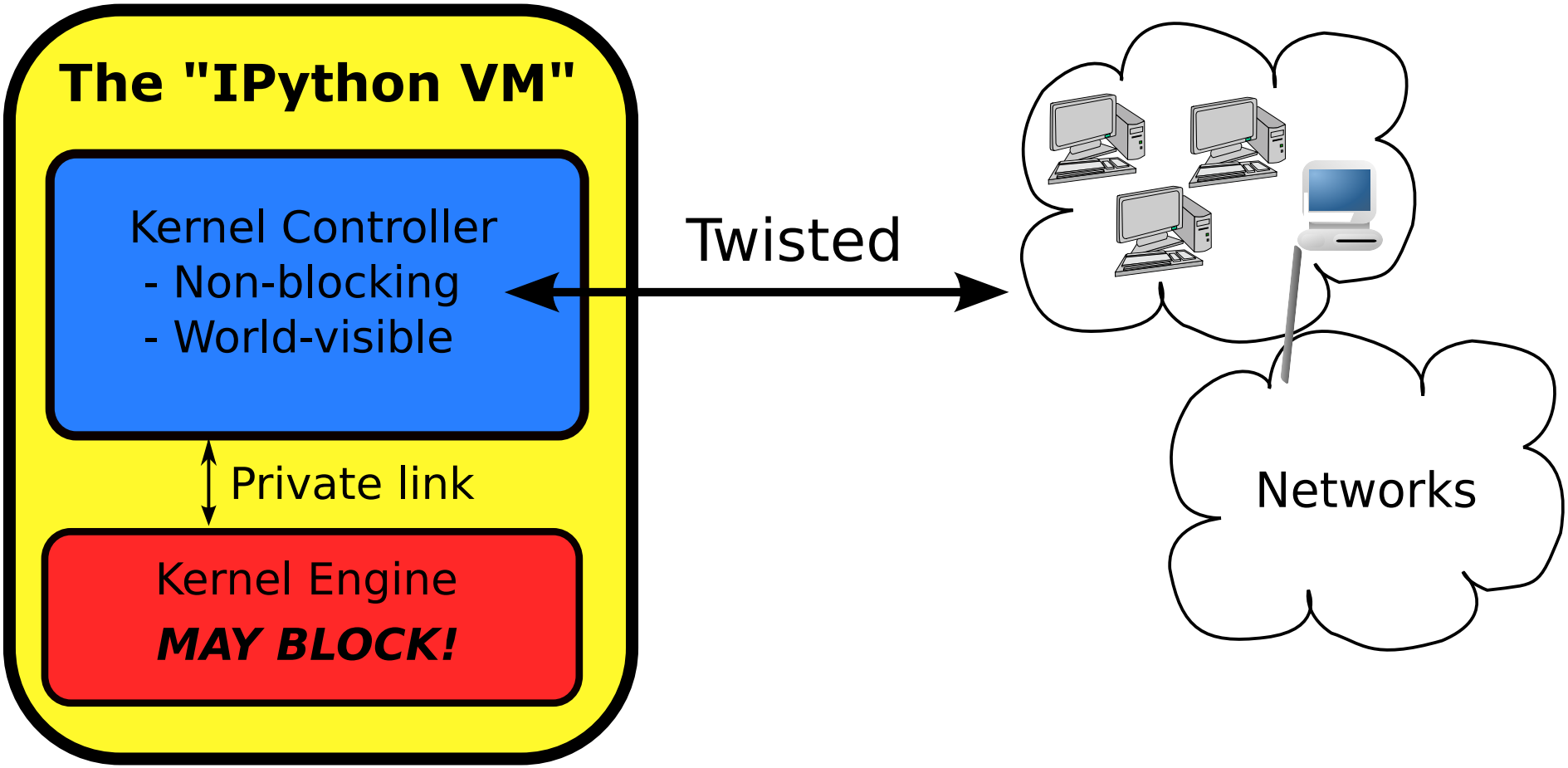


A 2-process kernel

Why do we need this?

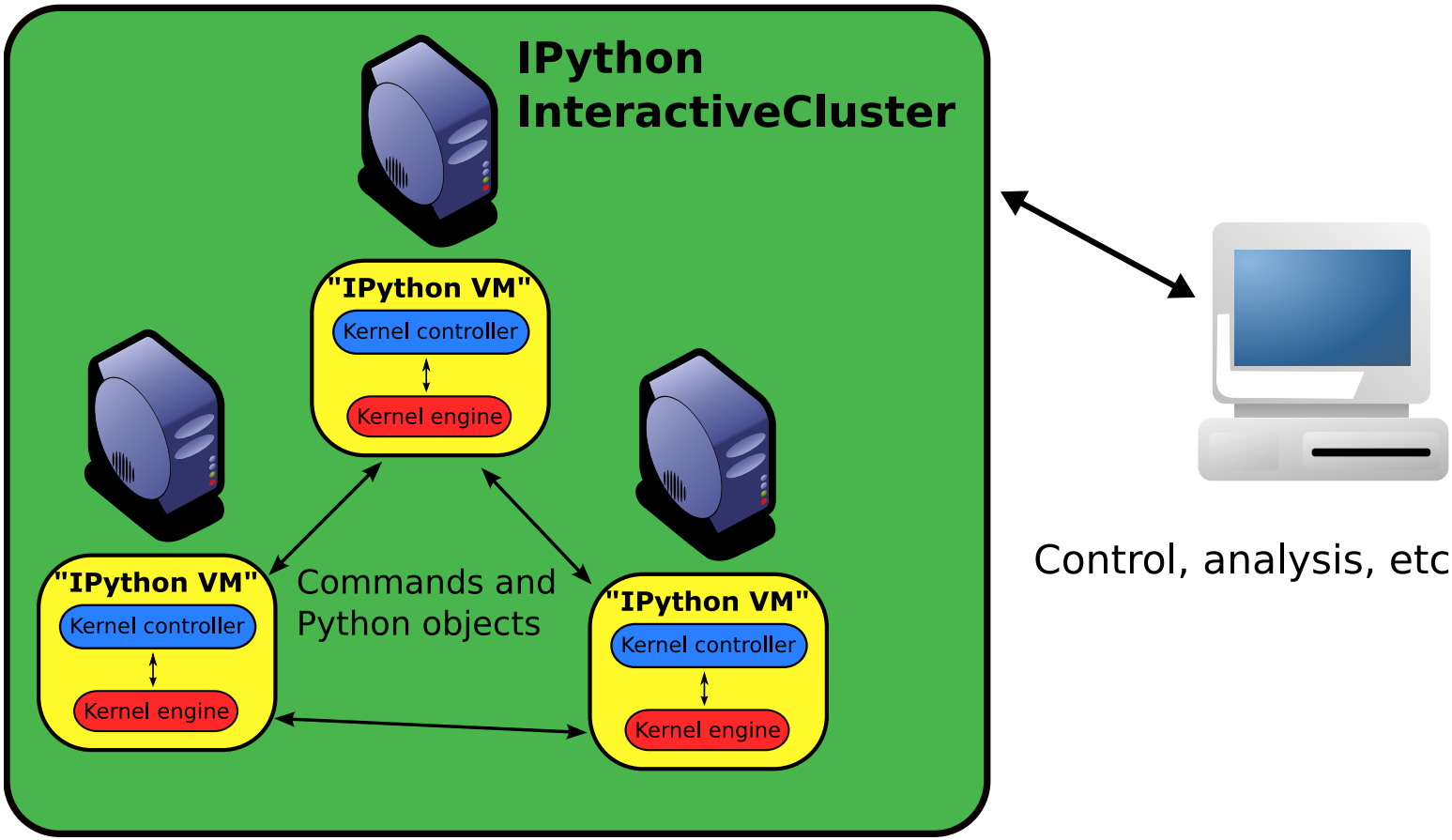
- The Python VM has a global lock (the Global Interpreter Lock – GIL).
- It protects the global state of the interpreter
  - Only one thread can execute Python code at the same time.
  - No Python variables may be modified without holding the GIL.
- Python *does* have threads: they work well for non-CPU bound tasks.
- **BUT**
  - Extensions (C, Fortran) can fully block the VM.
  - And poof goes all hope of the ability to control a cluster

A 2-process kernel (2)



Distributed/parallel computing

- Think of Python as 'the CPU'
- But these souped-up kernels let you talk to it conveniently.



A cluster in San Francisco, monitored from Colorado

- 10 nodes (5 dual G5 Macs) running IPython kernels at the University of Santa Clara.
- I can connect to them from my office (U. Colorado, Boulder) and listen.
- Computation happens on the nodes
- Results are collected for analysis on the controller.
- Plan B: if the network fails, we mimic this on my laptop (with a smaller calculation)

Outline

Don’t come to a conference with an untested laptop...

Unless you bring Robert Kern as well

- I’ll let Robert present this part: since he actually did the real work, it’s only fair.
- And ’`import wx`’ explodes on my laptop right now...