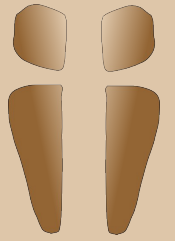


Pythonic Parallelism with CSP

Dr Russel Winder

It'z Interactive Ltd

russel@itzinteractive.com

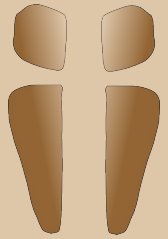


Pythonic Parallelism

Dr Russel Winder

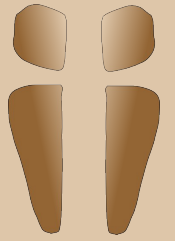
It'z Interactive Ltd

russel@itzinteractive.com



Aims and Objectives

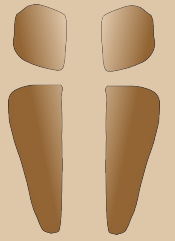
- Convince people that Communicating Sequential Processes (CSP) is *a very good* model for applications programming.
- Show that parallelism is fundamentally straightforward, even with Python.



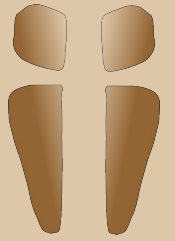
Structure of the Session

- Introduction.
- Provoke discussion.
- Exit stage (left | right).

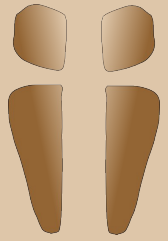
Dynamic binding is mandatory.



Assume everything
remembered from
Monday's presentation.

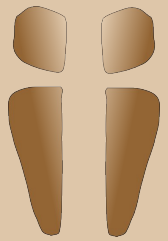


OK, a short resumé.

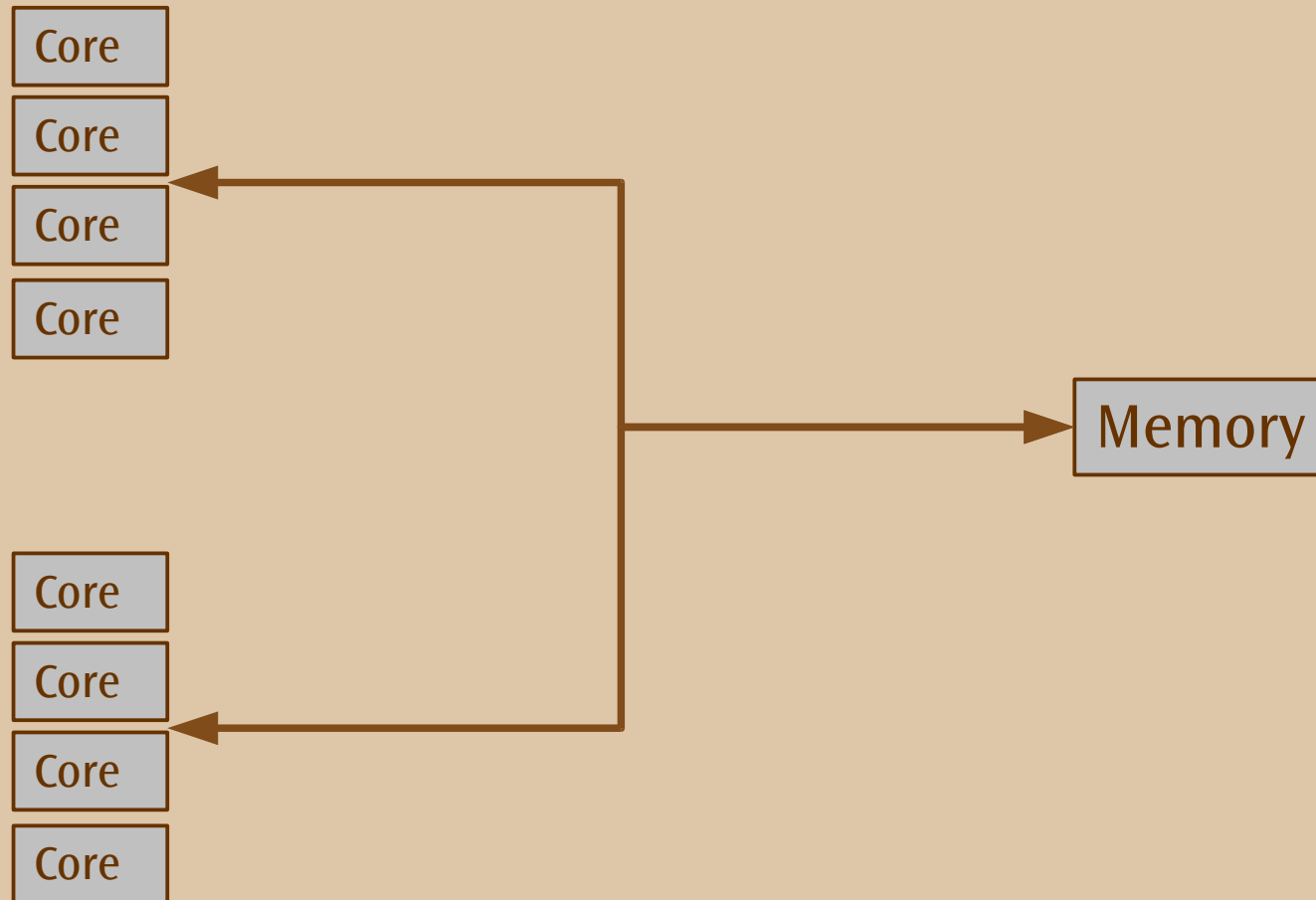


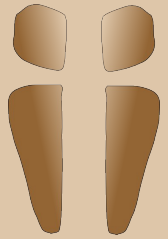
1950 – 2005 (ish)



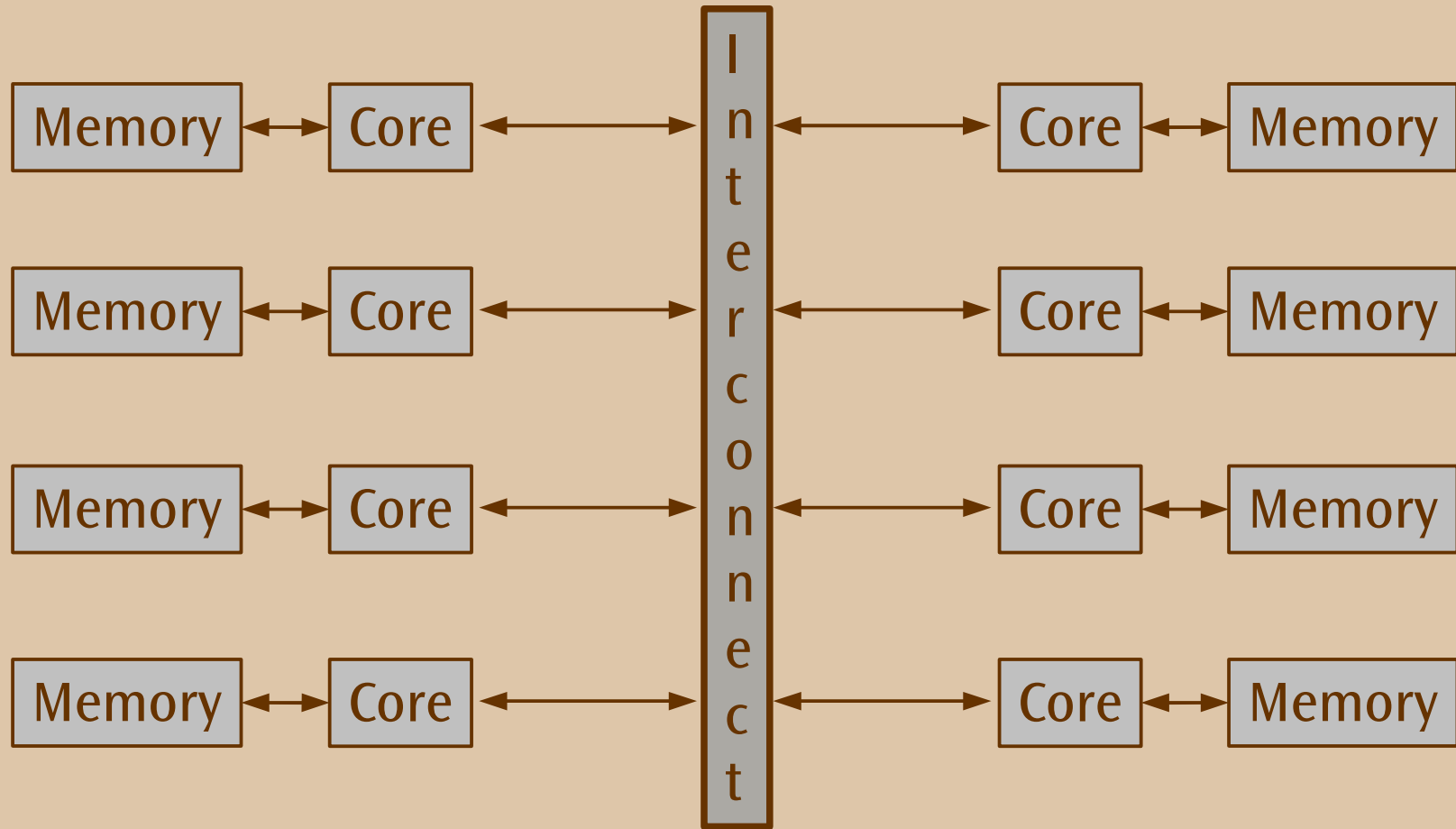


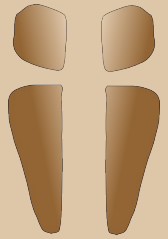
2005 (ish) – 2012 (ish)





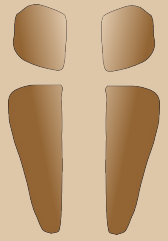
2011 –





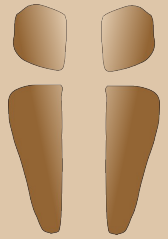
Parallelism: The Standard Model

- Fortran or C++ (or C if you have to) as the programming language.
- MPI
 - Distributed clustering.
 - Can harness multicores.
 - Single program multiple data (SPMD).
- OpenMP
 - Shared-memory multi-threading control.
 - Harnesses multicores.
- CUDA, OpenCL



Python: Working the Standard Model

- Python + multiprocessing
- MPI4Py
- *PyCUDA*
- *PyOpenCL*



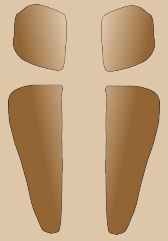
The Heretical Alternative View

And (almost) completely wrong.

- Shared-memory multi-threading is a good way of structuring code to harness parallelism.
- *Except in Python, OCaml, Ruby, etc.*

*These have a GIL to serialize
all thread activity.*

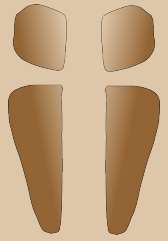
*1.8.x uses an interpreter.
1.9.x uses a bytecode machine.*



Threads are not Evil – per se

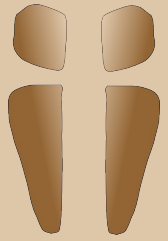
- Threads are:
 - A hardware feature.
 - An operating system feature.
 - A way of exposing hardware parallelism to applications.

So did Java, C++0x, and Python get it wrong?



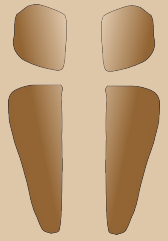
Shared-memory Multi-threading Reprise

- Shared-memory multi-threading is hard to get right . . .
- . . . even for those people who really do know what they are doing.



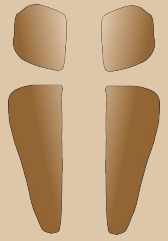
On Thread Safety

- Working with semaphores, monitors, locks, etc. to try and ensure thread safety is nigh on impossible even for super programmers.
- If anyone tries to tell you this is not the case, then they are just deluding themselves.
- Lock-free programming is even harder.



Managing Concurrency

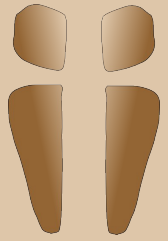
- It's been a problem for decades.
 - In operating systems, you have to deal with this using semaphores, locks, monitors, etc.
- Big Question:
 - *Why has this model of managing concurrency been forced on applications programmers?*



Are Futures the Future?

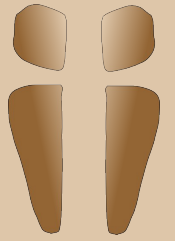
- **Yes** because it moves synchronization from being about control flow to being about data readiness; and
- **No** because futures are only really applicable for data-focused computations.

*There is an important point here:
if you make synchronization about
data readiness and not control,
parallelism all gets a lot easier.*



Message Passing a Better Way?

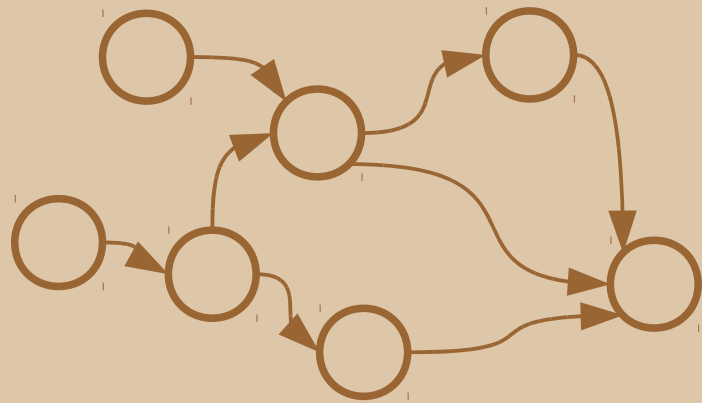
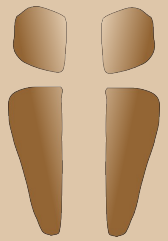
- Yes, definitely.
 - MPI is a proof of this.
 - It's just awkward using SPMD.

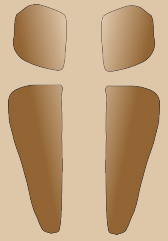


Solutions for Today

- Actor Model
- Dataflow Architectures
- Communicating Sequential Processes (CSP)
- *Data parallelism*

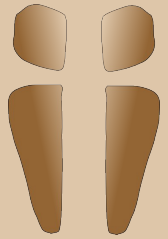
*Return to a **process** view of applications.*





A Message on Message Passing History

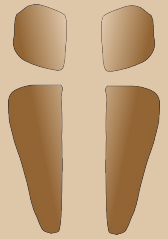
- Actors
 - ABCL
 - Erlang
 - Scala
- CSP (1978)
- CSP (1984)
- CSP (2010–)
- occam
- occam- π
- Newsqueak
- Alef
- Limbo
- Go



Parallel Development

- **occam**
- **Python**
 - processing
 - multiprocessing
- **JCSP**
- **PyCSP**
- **Python-CSP**
- **Go**

Processes and channels with slightly different semantics.

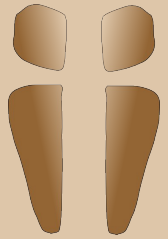


CSP is . . .

- . . . mathematics:

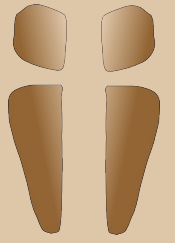
$$\text{VMS} = \mu X. (\text{in2p} \rightarrow (\text{chocolate} \rightarrow X \mid \text{out1p} \rightarrow \text{toffee} \rightarrow X) \\ \mid \text{in1p} \rightarrow (\text{toffee} \rightarrow X \mid \text{in1p} \rightarrow (\text{chocolate} \rightarrow X \mid \text{in1p} \rightarrow \text{STOP} \alpha X)))$$

- . . . not scary since the mathematics can be hidden in an API, so it just becomes a programming tool.

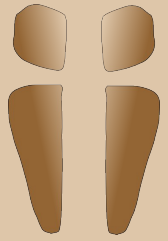


Example Problem

- Something small, so the code is small.
- Something not too “toy”.
- Something with good parallelism.
 - *Embarrassingly parallel* to allow checking of *scaling*.

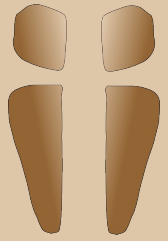


π



What is the Value of π ?

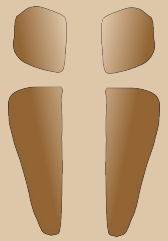
- The value of π is known exactly, it's π (obviously).



Approximating π

- What is value of π is represented as a floating point number?
 - We can only obtain an approximation.
 - A plethora of possible algorithms to choose from, a popular one is to employ the following integral equation.

$$\frac{\pi}{4} = \int_0^1 \frac{1}{1+x^2} dx$$

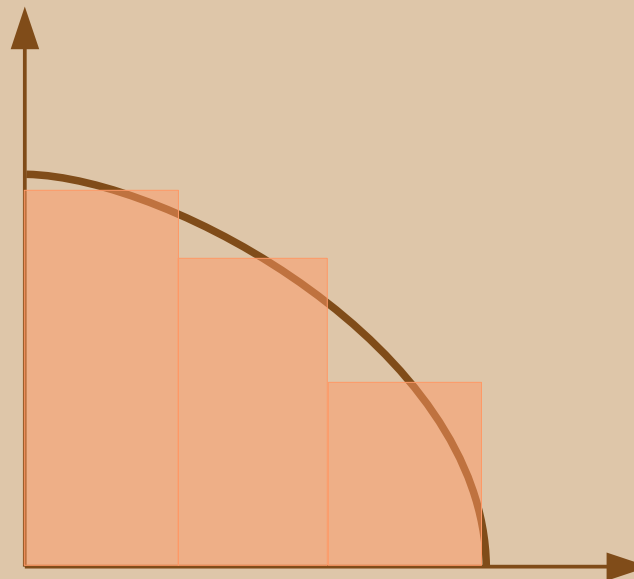


One Possible Algorithm

- Use quadrature to estimate the value of the integral – which is the area under the curve.

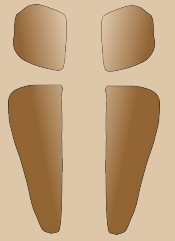
$$\pi = \frac{4}{n} \sum_{i=1}^n \frac{1}{1 + \left(\frac{i-0.5}{n}\right)^2}$$

There is an interesting disconnect between the equation and the diagram.



Embarrassingly parallel.

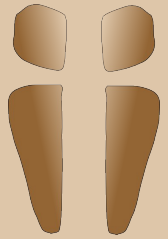
With $n = 3$ not much to do, but potentially lots of error. With $n = 10^9$ things get more interesting.



The Machine

- Twin Xeon E5410 2.33GHz.
- 4GB memory.
- Ubuntu 10.04 Lucid Lynx AMD64.

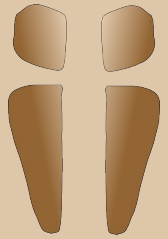
*She's called Anglides,
do say hello.*



The Laptop

- Core2 Duo T9550 2.66GHz
- 4GB memory
- Debian Testing (aka Squeeze) AMD64; or Ubuntu 10.04 Lucid Lynx AMD64.

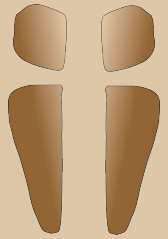
*He's called Launcelot,
do say hello.*



Sequential

```
import time

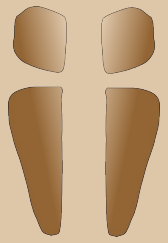
n = 10000000 # 100 times fewer due to speed issues.
delta = 1.0 / n
startTime = time.time ()
sum = 0.0
for i in xrange ( 1 , n + 1 ) :
    x = ( i - 0.5 ) * delta
    sum += 1.0 / ( 1.0 + x * x )
pi = 4.0 * sum * delta
elapsedTime = time.time () - startTime
print ( "==== Python Sequential For/Xrange pi = " + str ( pi ) )
print ( "==== Python Sequential For/Xrange iteration count = " + str ( n ) )
print ( "==== Python Sequential For/Xrange elapse = " + str ( elapsedTime ) )
```

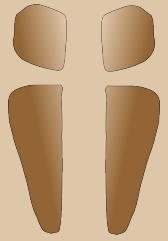


Multithreaded

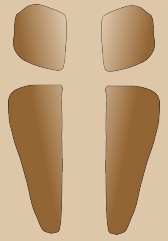
- Absolutely no point because of the GIL.

Multiprocess





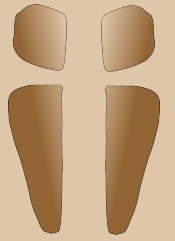
Lots of Other Solutions . . .



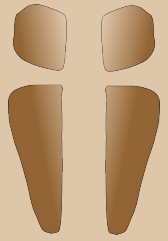
Performance isn't Everything, But . . .

- If performance is critical, write the critical tight loops in C++ (or C if you really have to).
- Sequential C++ computations managed by Python coordinating programs are (almost) as fast as pure C++.
- Python is much easier to create visualization codes with.

The scientific community are learning this very rapidly.



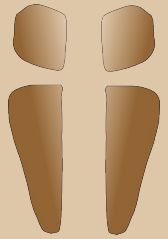
Go from here and
learn about CSP-based
frameworks.



Summary

- Models such as:
 - Actor
 - Dataflow
 - **CSP**
 - Data parallelism

are the future models of application architecture.

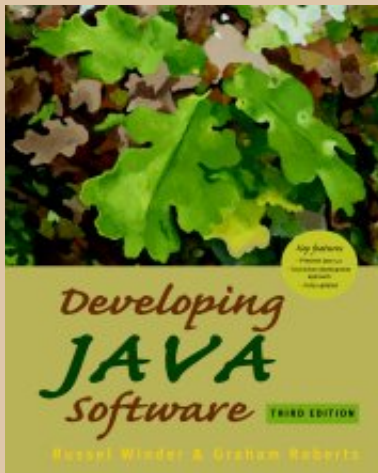
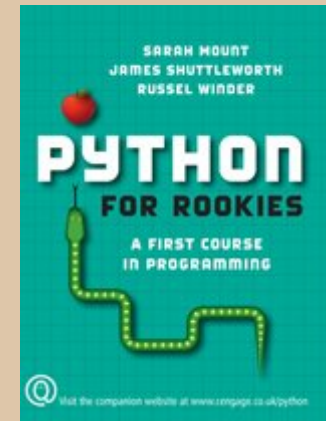


Blatant Advertising

Python for Rookies

Sarah Mount, James Shuttleworth and
Russel Winder

Thomson Learning *Now called Cengage Learning.*



Developing Java Software Third Edition

Russel Winder and Graham Roberts

Wiley

Buy these books!

