Jeepers GPars, It's All So Easily Parallel

Russel Winder

email: russel@russel.org.uk
xmpp: russel@russel.org.uk
twitter: russel_winder

Aims, Goals and Objectives

- Get people enthusiastic to use dataflow, actors and Communicating Sequential Processes (CSP) as their application structuring architecture.
- Get people enthusiastic to use GPars in all their Java and Groovy programming.

- *Arrive at an hostelry in good time to have a nice drink and good conversation.*

Structure

- The Beginning.
- The Middle.
- The End.
- The Extra Questions.
- *The Whisky.*

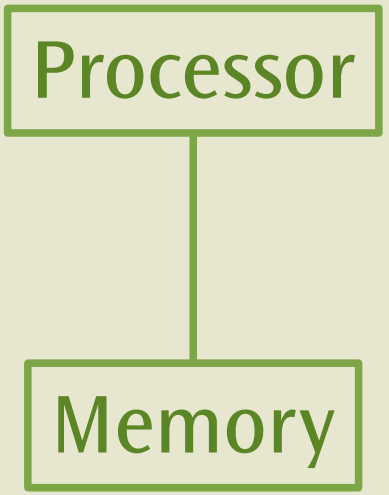
Protocol

- Interaction is allowed. Actually it may well, possibly, be mandatory.
- *If an interjection leads to a "too long" side track, we will stack (or possibly even stash†) it for later.*

†Depending on whether you want a code oriented or a version control oriented metaphor.

The Beginning

In the Beginning: The Hardware



Single ALU

In the Beginning: The Software

Load a program into the memory and run it to completion.

Multitasking Operating Systems

Load many programs into memory and have one of then run at any one time.

Concurrency

- Multitasking operating systems introduce the need for concurrency in a shared memory context.
- Tools for managing this are created:
  - Locks
  - Semaphores
  - Monitors

Higher Level Models

- Dataflow Model:
  - Bert Sutherland, 1966.
- Actor Model:
  - First published in 1973 by Carl Hewett, Peter Bishop and Richard Steiger – IJCAI.
- Communicating Sequential Processes (CSP):
  - First published in a paper by Tony Hoare in 1978, but only really became well known with the 1983 book.

The Interregnum Begins

Programmers were taught that concurrent applications needed the same tools and techniques that operating system implementation needed:
Shared memory multi-threading.

The Interregnum Reified

1995, Java reifies shared memory multithreading as the obviously known right way of dealing with concurrency…

…after all C and C++ have been using pthreads (or the like) for many years.

The Interregnum Reified, A Bit More

1995, Java reifies the mindset that concurrent programming is all about shared memory multithreading by putting it in the language…

…it takes till 2011 for C++ to do the same.

The Interregnum Continues

Programmers discover that shared-memory multithreading is hard[†] to get right: that trying to get things right with lock, semaphores and monitors is not entirely easy[‡].

†By hard, what is actually meant is usually:

Absolutely impossible!

‡Clearly “not entirely easy” is a euphemism,
see the previous slide for the
more appropriate description.

Concurrency Eschewed

Programmers know concurrent and parallel programming is hard, so they don't do it.

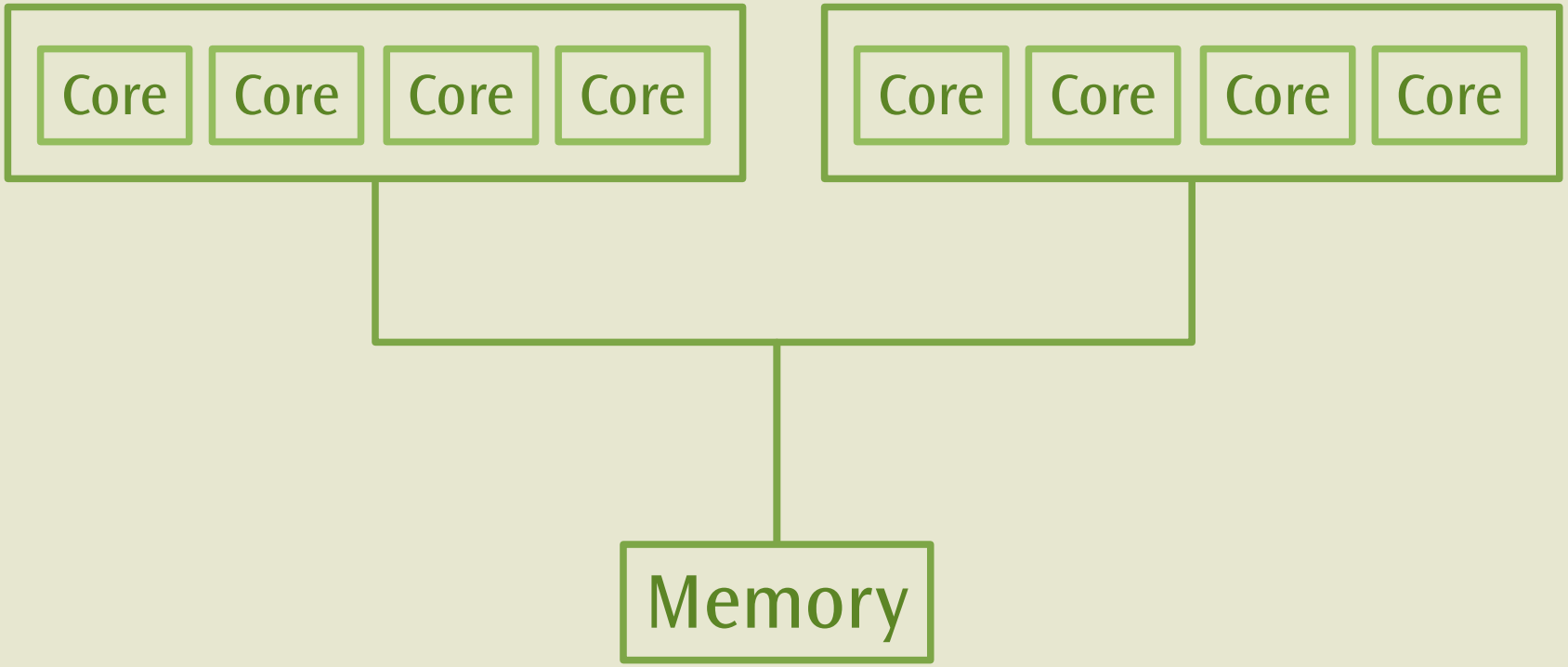
Why Bother?

Processor speeds double every couple of years, so single thread applications get twice as fast every two years, so who cares about concurrency and parallelism?

Oh Dear

Processor speeds have to stop getting faster, Moore's Law still working, processor manufacturers start increasing the core count to use all the extra transistors they have.

Multicore Revolution: The Early Period



The Hardware Con Job

Each processor has *N* ALUs and so executes *N* instructions per unit time so is *N* times faster than a single core processor.

The Realization

For compute intensive applications, increased parallelism is now the only way to create increased application performance.

The Mechanisms

Kernel threads mean that applications can harness real parallelism with threads not just time-division multiplexing concurrency.

The Problem

Creating large, correct programs using shared memory multi-threading is:

Absolutely impossible!

The Middle

Strategy

- Use high-level concurrency structures:
  - Actors
  - Dataflow
  - Communicating Sequential Processes (CSP)
  - Data Parallelism

Actor Model

- A collection of processes that communicate by sending messages to each other.
- No global shared state.

Dataflow Model

- A collection of processes that communicate by sending messages to each other.
- No global shared state.

Communicating Sequential Processes

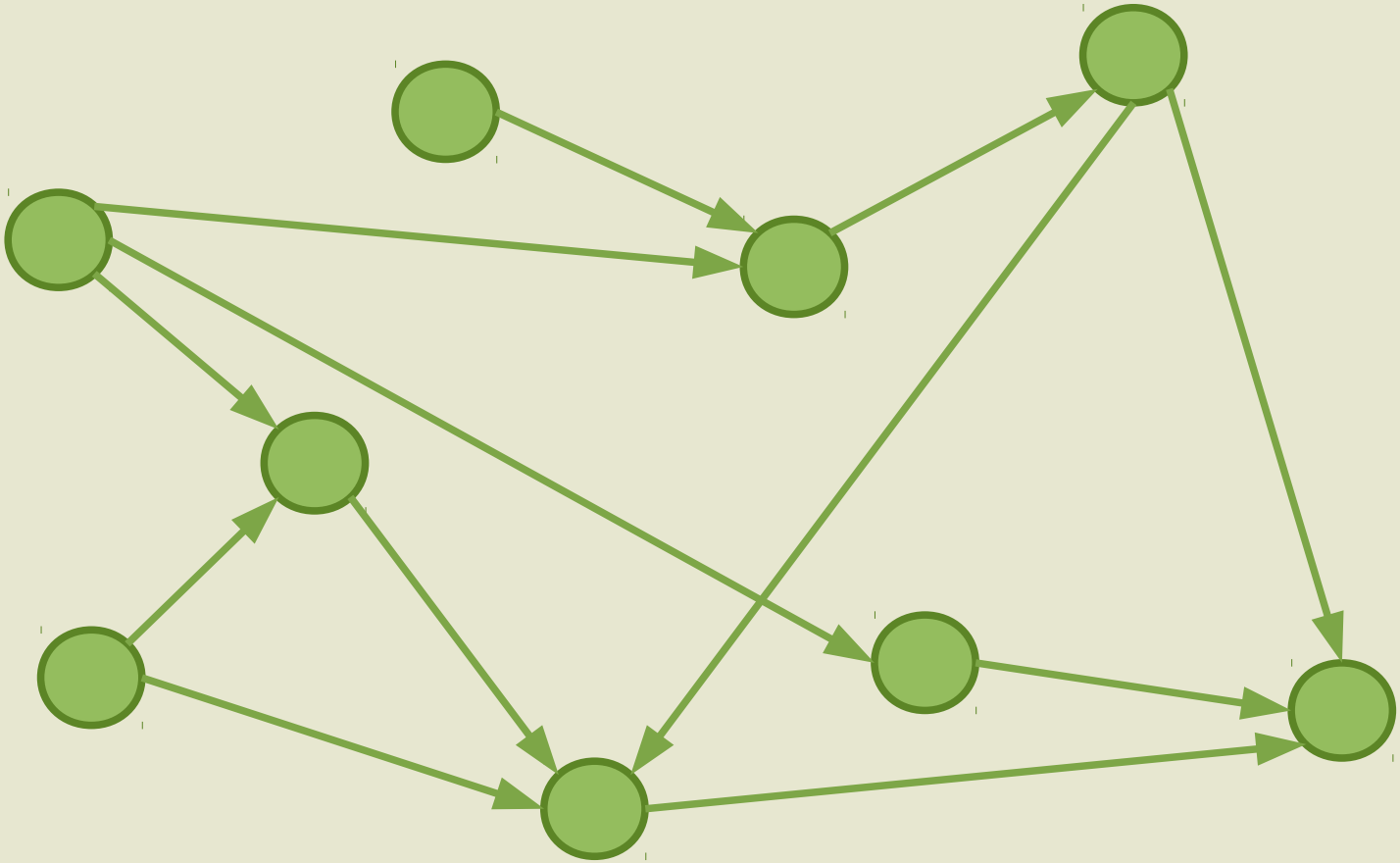
- A collection of processes that communicate by sending messages to each other.
- No global shared state.

So what is the difference?

It's all in the message passing and hence synchronization.

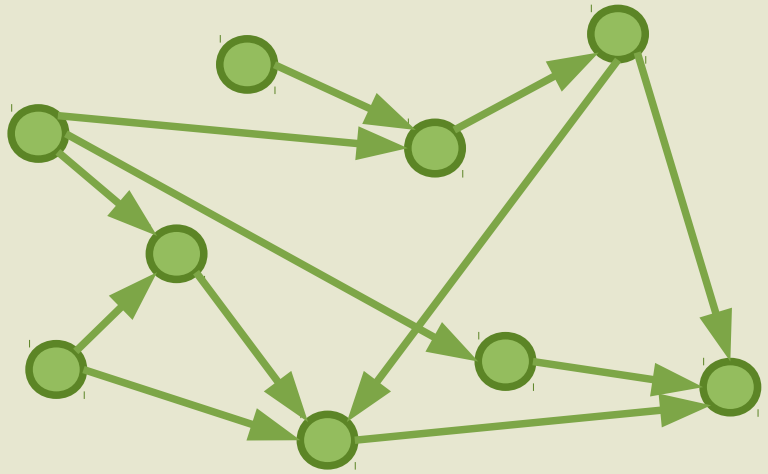
And turning threads into a hidden and managed resource.

The Abstract Model



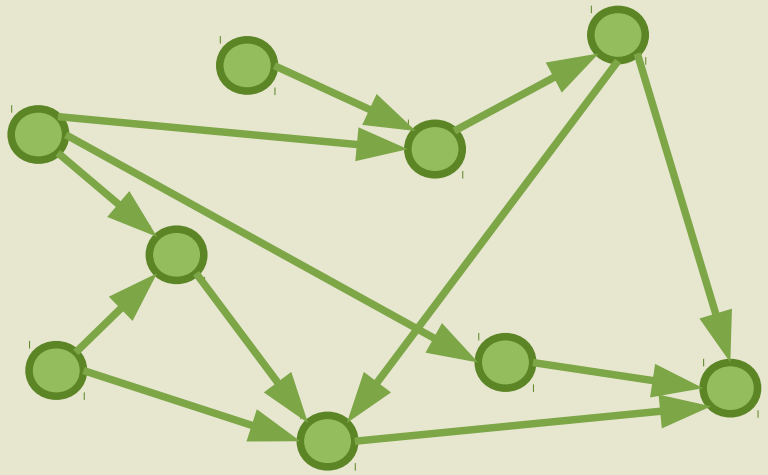
Actor Model

- Each actor has a message queue.
- Actors can send messages asynchronously to any other actor.
- Actors read messages from their message queues, do some work and send messages to other actors.



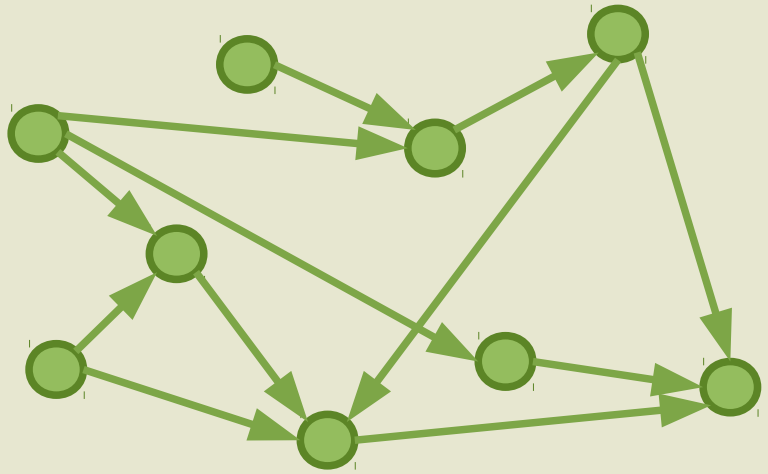
Dataflow Model

- Each *operator* has a set of inputs: single assignment variables, or a queue of such things.
- Operator block until a given state of its inputs and then “fires” creating values on its outputs.



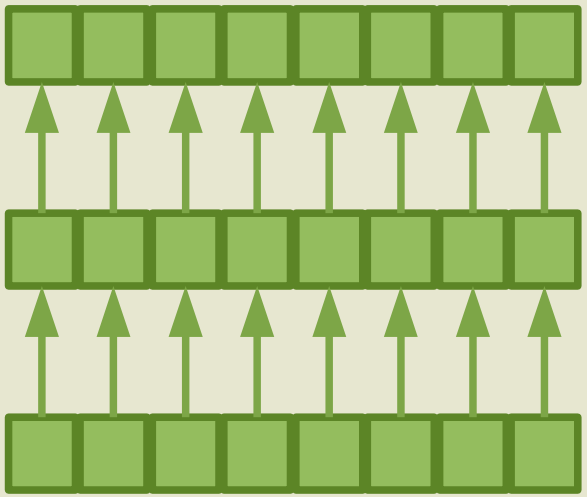
Communicating Sequential Processes

- Each process has a set of input channels.
- A process takes data from one of its channels synchronously (rendezvous), computes and then writes to one of its output channels.



Data Parallelism

- Data is in some array-like data structure.
- At each stage of a computation, a transformation is applied to all the items in the data structure.



Sample Problems

- Sleeping Barber
- π by Quadrature

The Sleeping Barber Problem

A barber sleeps in the cutting chair unless cutting someone's hair. Customers enter the shop: if the barber is asleep, the customer awakens the barber, sits in the chair and gets a cut; if the barber is cutting the customer checks to see if there is a free waiting chair, and if there is sits to wait their turn or if not leaves the shop, uncut. On finishing a cut, the barber checks the waiting chairs to see if there is a new customer to cut. If there is, the customer moves to the cutting chair and gets a cut, if there isn't the barber takes the cutting chair and sleeps.

Problem believed to be originally due to Edsger Dykstra, 1965.
It is a model of a process management problem in operating systems.

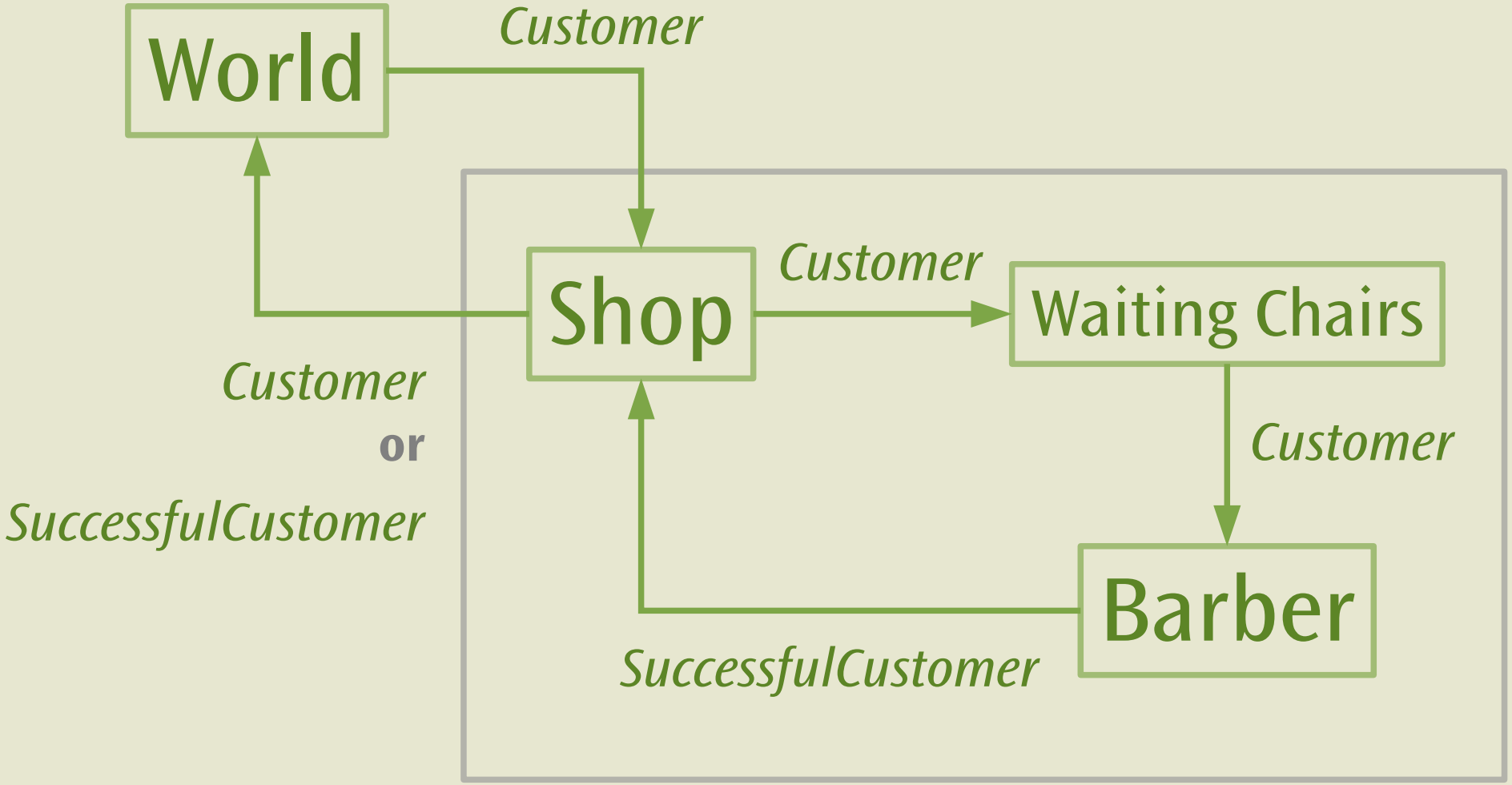
http://en.wikipedia.org/wiki/Sleeping_barber_problem

Operating Systems to Simulation

Implementing a solution to the problem in an operating systems context is essentially a "solved" problem.

Extend the problem to be an example of concurrency and possible parallelism in simulation of a queueing problem, and as a vehicle for trying various technologies.

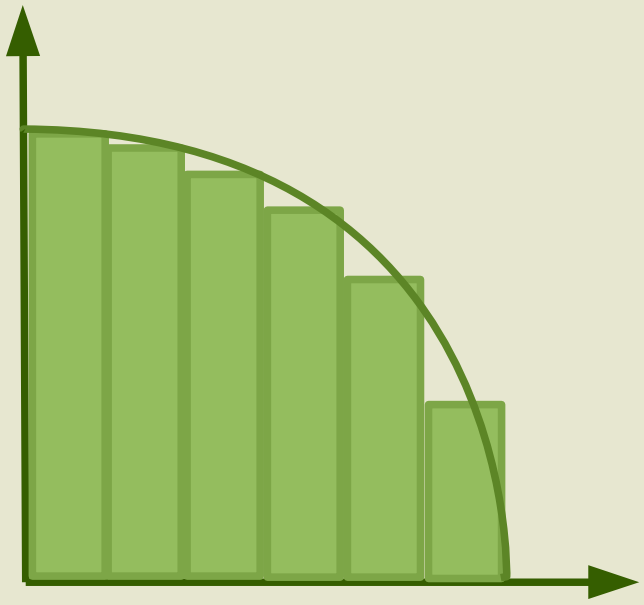
The Abstract Model



π By Quadrature

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

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



The Code

Sleeping Barber : http://www.russel.org.uk/Bazaar/SleepingBarber

π By Quadrature: http://www.russel.org.uk/Bazaar/Pi_Quadrature

The End

Actors, dataflow, CSP, data parallelism are the high-level abstractions.

Shared memory multi-threading is low-level infrastructure.

C++, Java, Groovy, Python, etc. are high-level programming languages.

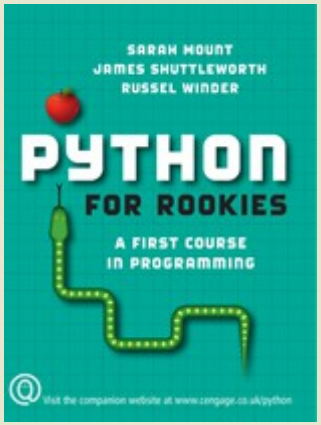
Assembly language is low-level infrastructure.

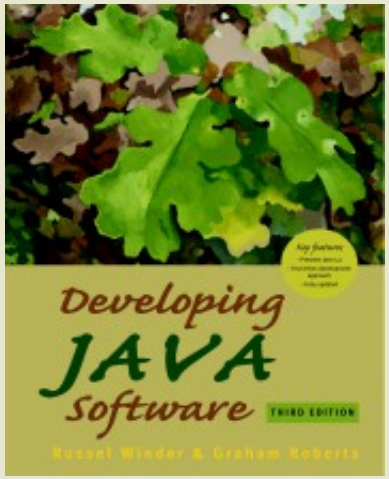
Advertising

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!

The Extra Questions

The Whisky