ACM/CS 114
Parallel algorithms for scientific applications

Michael A. G. Aïvázis

California Institute of Technology

Spring 2012

Creating an application

```
10 # externals
11 import sys
12 import pyre
13 import gauss
14
15 # the application
16 class Quad(pyre.application):
17     """
18     A harness for {gauss} integrators
19     """
20
21     # public state
22     samples = pyre.properties.int(default=10**5)
23     integrator = pyre.facility(interface=gauss.integrator)
24
25     # required interface
26     @pyre.export
27     def main(self, *args, **kwds):
28         # pass the requested number of samples to the integrator
29         self.integrator.samples = self.samples
30         # get it to integrate
31         integral = self.integrator.integrate()
32         # print the answer
33         print("integral = {}".format(integral))
34         # return success
35         return 0
```

Auto-launching

instantiating and launching the application

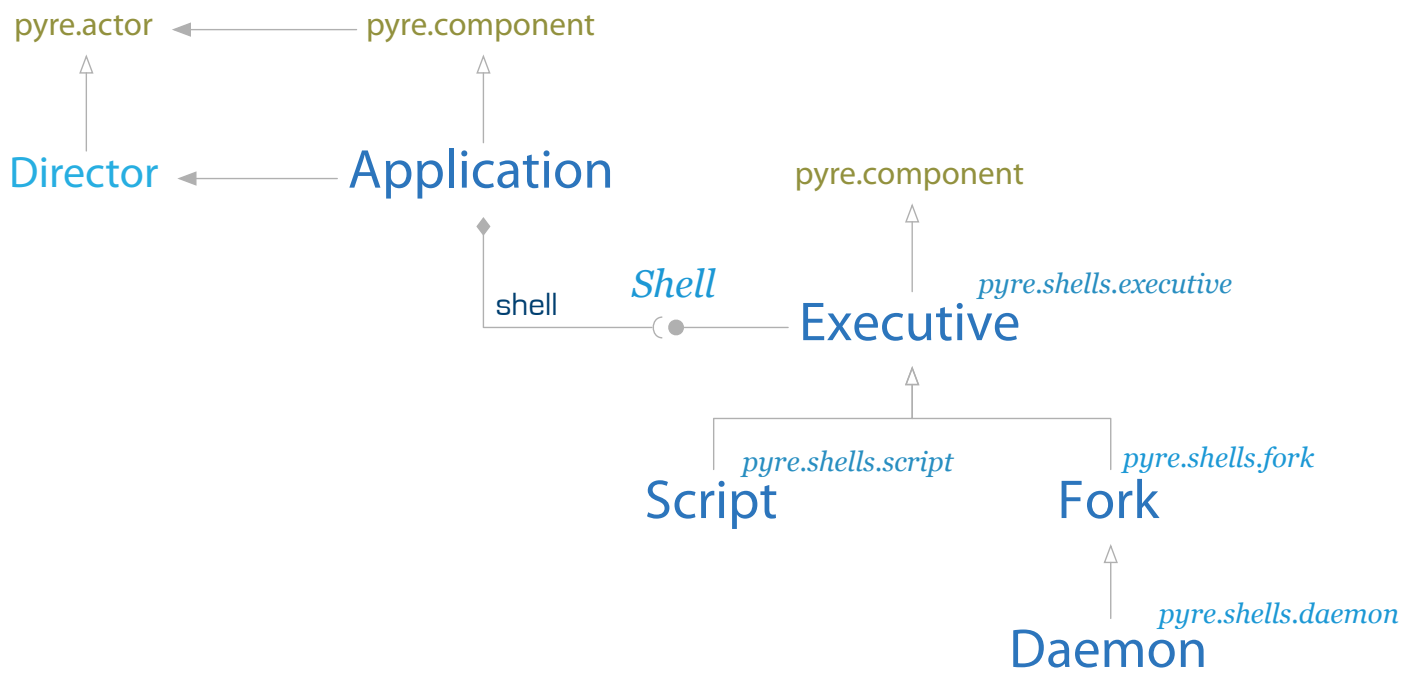
```
55 # main
56 if __name__ == "__main__":
57     # instantiate the application
58     q = Quad(name='quad')
59     # run it and return its exit code to the os
60     sys.exit(q.run())
```

a sample configuration file

```
8  ; application settings
9  [ quad ]
10 samples = 10**6
11
12 ; cconfiguration for the integrator
13 [ gauss.integrators.montecarlo # quad.integrator ]
14 box.diagonal = ((-1,-1), (1,1))
15 region = ball
16 integrand = constant
17
18 ; when the integrand is the constant functor
19 [ gauss.functors.constant # quad.integrator.integrand ]
20 value = 1
```

The application component

quick introduction to pML



our `Quad` derives from `Application`, so it has a `shell`

Parallel integration

the `mpi` entry point

```
37      @pyre.export
38      def main_mpi(self, *args, **kwds):
39          # access the mpi package
40          import mpi
41          # find out how many tasks were launched
42          size = mpi.world.size
43          # find out my rank
44          rank = mpi.world.rank
45          # figure out how many samples to do and pass that on to my integrator
46          self.integrator.samples = self.samples / size
47          # integrate: average the estimates produced by each task
48          integral = mpi.sum(self.integrator.integrate())/size
49          # node 0: print the answer
50          if rank == 0: print("integral = {}".format(integral))
51          # all done
52          return 0
```

the `mpi` package is part of the pyre distribution

- handles initialization and finalization of `MPI`
- simplifies most of the “overhead” activities
- provides an OO veneer

Running in parallel

minor modifications to the configuration file...

```
8  ; application settings
9  [ quad ]
10 samples = 10**6
11
12 ; cconfiguration for the integrator
13 [ gauss.integrators.montecarlo # quad.integrator ]
14 box.diagonal = ((-1,-1), (1,1))
15 region = ball
16 integrand = constant
17
18 ; when the integrand is the constant functor
19 [ gauss.functors.constant # quad.integrator.integrand ]
20 value = 1
21
22 ; for MPI
23 [ quad ]
24 shell = mpi
25
26 [ mpi.shells.mpirun # quad.shell ]
27 tasks = 8
28 launcher = openmpirun
```