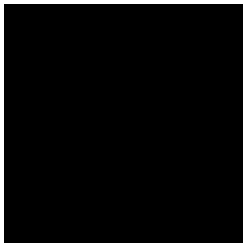
Seattle: Open P2P Computing

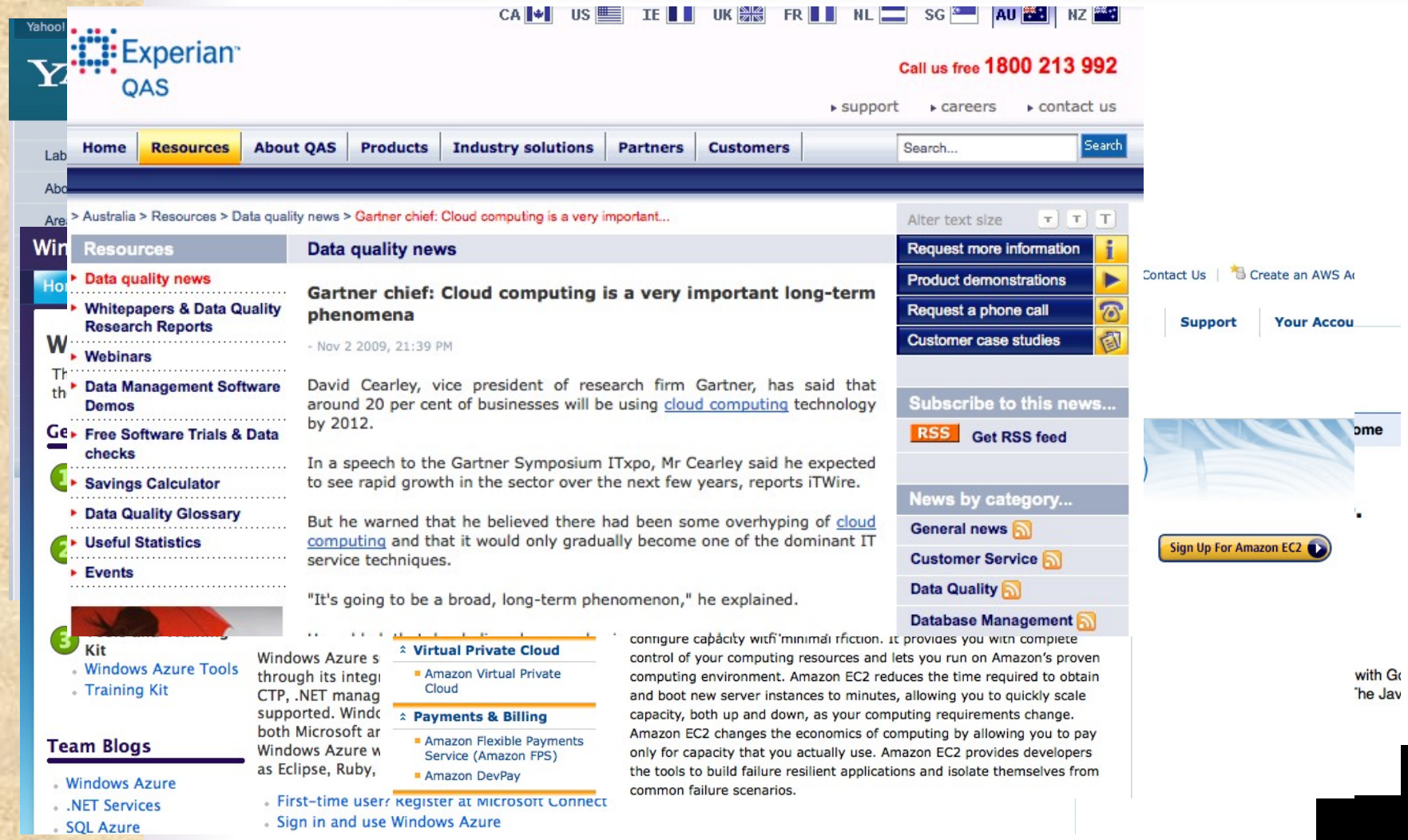
Justin Cappos

Computer Science and Engineering

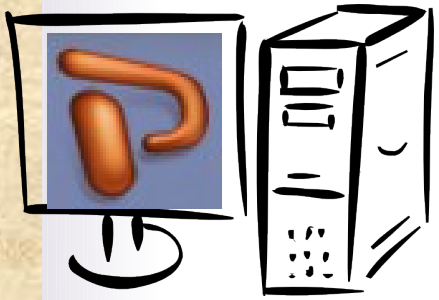
University of Washington



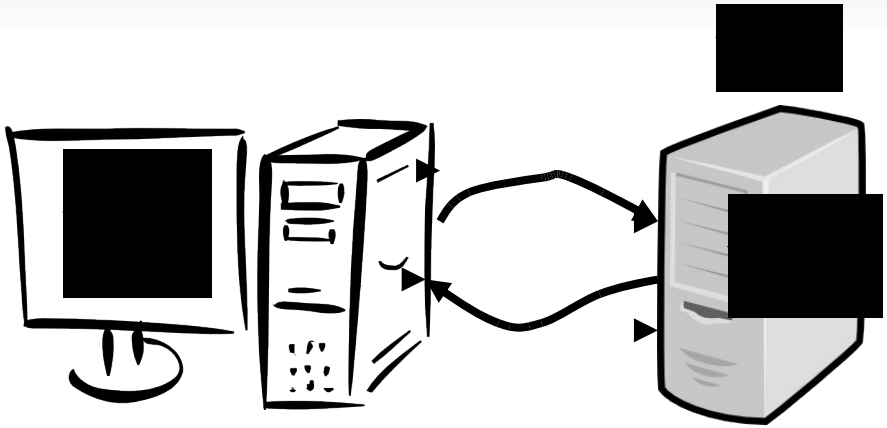
Cloud Computing



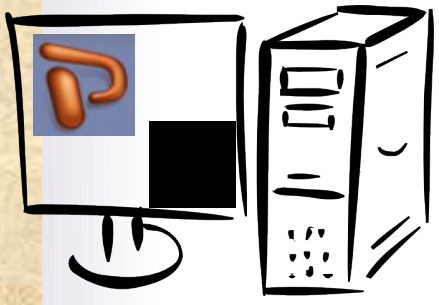
What Cloud Computing Really Is

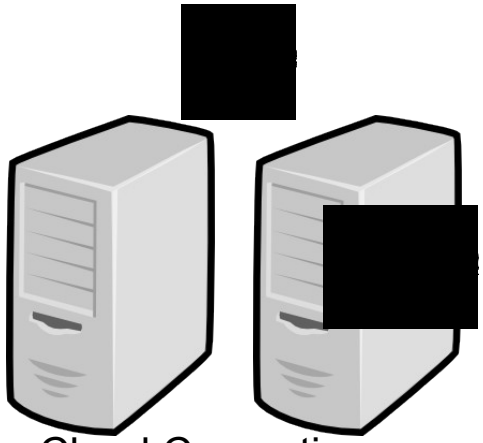


Workstation

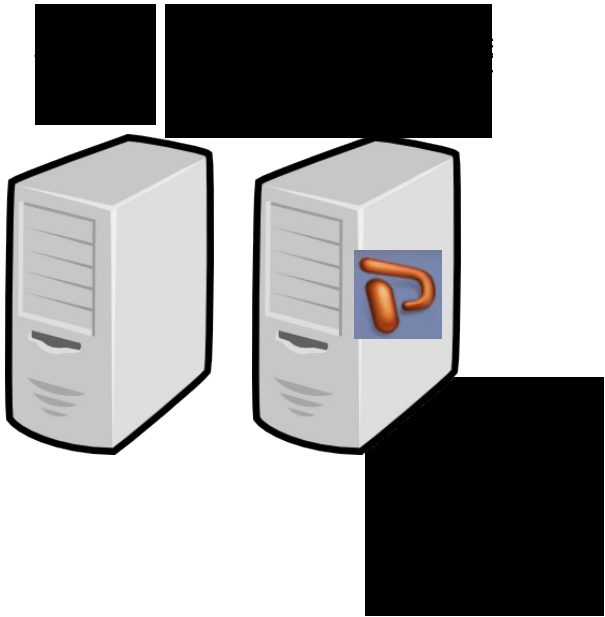


Client / Server





Cloud Computing



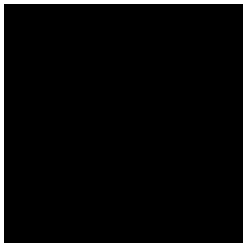
The Silver Lining

- Computation can scale with demand
- No hardware purchases
- Minimizes IT overhead
- APIs provide
  - Fault tolerance
  - Scalability

The Dark Cloud

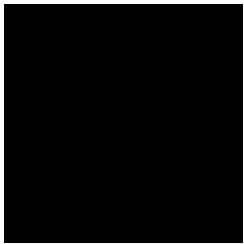
- Resources cost money
  - Sponsor w/ advertisements?
- Loss of privacy
- Censorship
- Vendor API lock-in

Cloud computing erodes software freedom



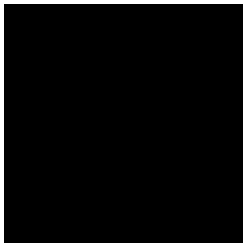
The Peer-to-Peer Alternative

- P2P leverages end user computation
  - Avoid lock-in through open protocols
  - Free
  - Scalability
- P2P challenges
  - Heterogeneity
  - Many apps have short use patterns
  - Fault tolerance
  - Distributed state management



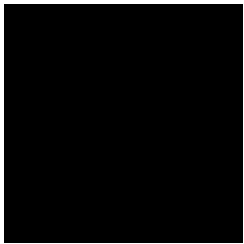
The Peer-to-Peer Alternative

- P2P leverages end user computation
  - Avoid lock-in through open protocols
  - Free
  - Scalability
- P2P challenges
  - Heterogeneity
  - Many apps have short use patterns
  - Fault tolerance
  - Distributed state management



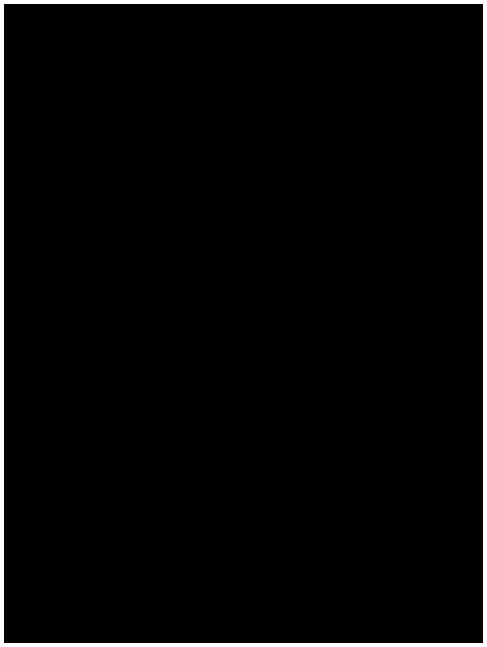
Seattle: Making P2P Easier

- Heterogeneity
  - Provide a portable / virtualized programming abstraction
- Many apps have short use patterns
  - Platform is always-on



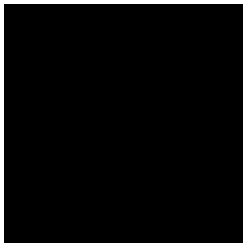
What is Seattle?

- P2P hosting platform (it's a potluck!)
  - You contribute some resources
  - You consume what others provide
  - Others consume what you provide
- Contributed resources
  - Remotely accessible
  - Requested through a sharing site
  - Run in a virtual machine (vessel)



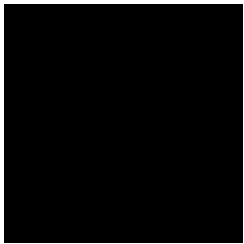
What is a Seattle Vessel?

- Portable / interoperable
  - Mac / Linux / BSD / Windows / Nokia devices, etc.
- Performance isolated
  - CPU, memory, disk, network b/w, etc.
- Safe
  - Isolated directory, can do no harm

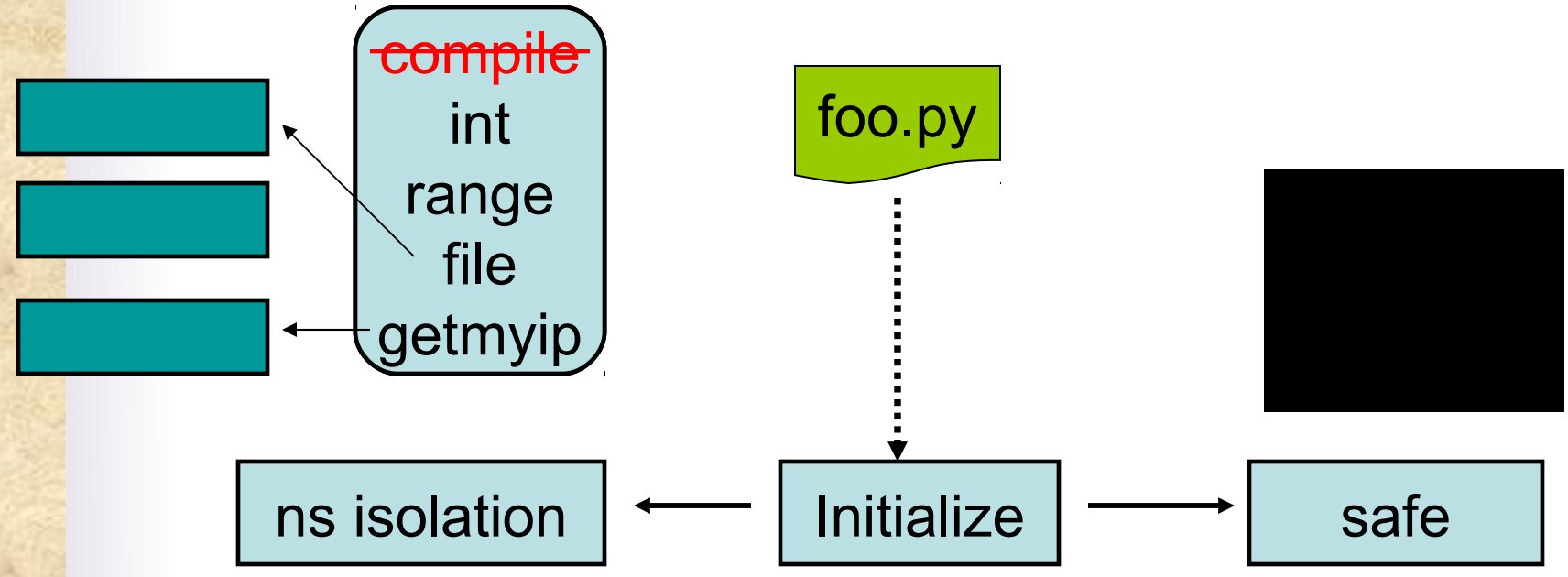


Programming Seattle Vessels

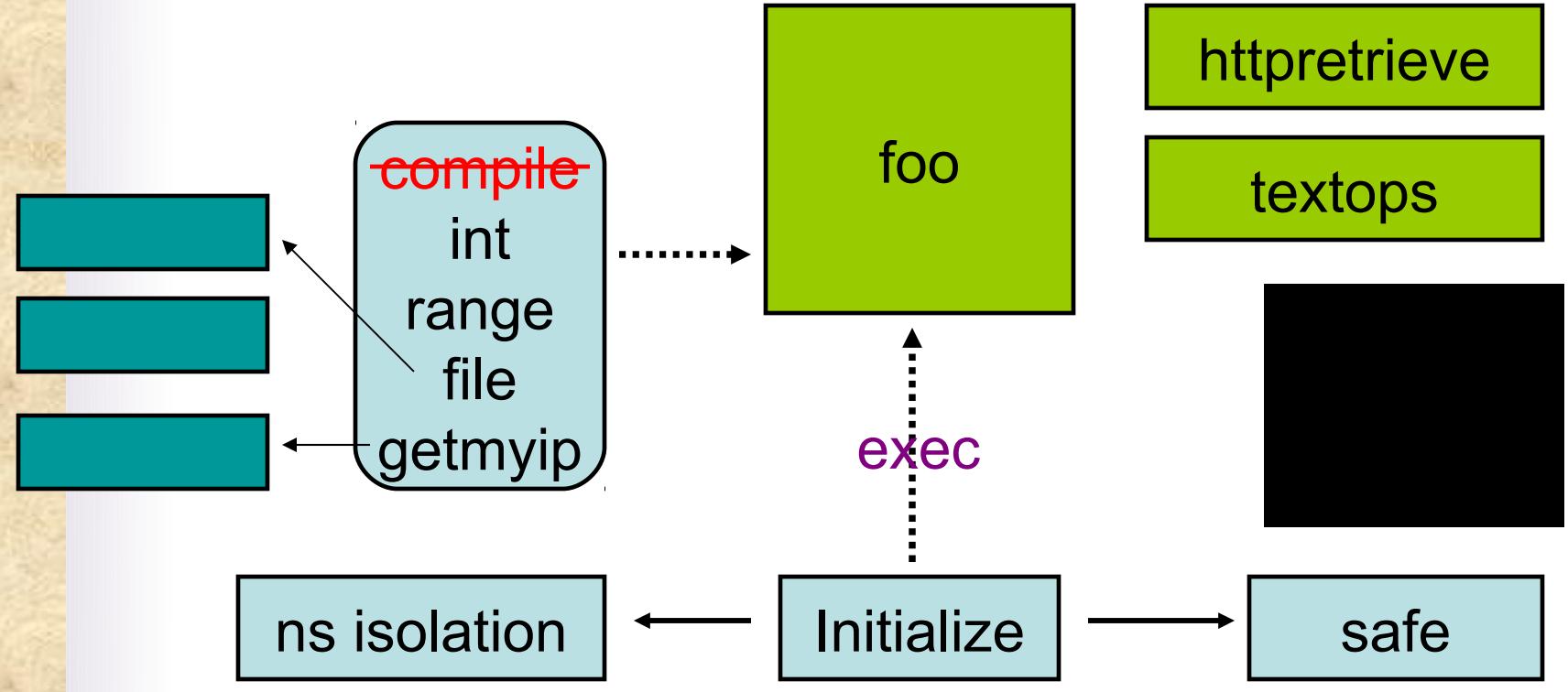
- Executes a subset of Python
  - Missing some built-ins, __slots__, etc.
  - Replacements for import, eval, standard libs, etc.
- Implemented in Python
  - Operating system ugliness masked
  - Performance isolation at call time



Executing Python Code Safely



Executing Python Code Safely



Easy To Code

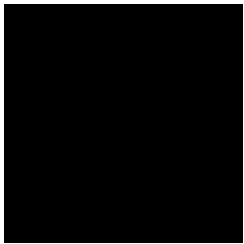
UDP ping server (4 LOC)

```
def got_message(srcip,srcport,mess,ch):
  sendmess(srcip,srcport,mess)
if callfunc == 'initialize':
  recvmess(getmyip(),54321,got_message)
```

UDP ping client (6 LOC)

```
def got_reply(srcip,srcport,mess,ch):
  print 'received:',mess,'from',srcip,srcport
if callfunc == 'initialize':
  recvmess(getmyip(),43210,got_reply)
  # send the second arg to the first arg's IP
  sendmess(callargs[0],54321,callargs[1],getmyip(),43210)
  # exit in one second
  settimer(1,exitall,())
```

UDP multicast (<20 LOC), TCP tunneling (<20 LOC), Chord (~300 LOC)



Demonstration

- Registration
- Download Installer
- Acquire resources
  - Use seattlegeni website
- Deploy all pairs ping
  - Use shell to locate and control resources
- Automated deployment tools

All Pairs Ping

Send periodic UDP pings
15 LOC

```
# send a probe message to each neighbor
def probe_neighbors(port):

  for neighborip in mycontext["neighborlist"]:
    mycontext['sendtime'][neighborip] = getruntime()
    sendmess(neighborip, port, 'ping',getmyip(),port)

    sendmess(neighborip, port,'share'+encode_row(getmyip(), mycontext["neighborlist"], mycontext['latency'].copy()))
    # sleep in between messages to prevent us from getting a huge number of
    # responses all at once...
    sleep(.5)

  # Call me again in 10 seconds
  while True:
    try:
      settimer(10,probe_neighbors,(port,))
      return
    except Exception, e:
      if "Resource 'events'" in str(e):
        # there are too many events scheduled, I should wait and try again
        sleep(.5)
        continue
      raise
```

Handle incoming UDP pings
7 LOC

```
# Handle an incoming message
def got_message(srcip,srcport,mess,ch):
  if mess == 'ping':
    sendmess(srcip,srcport,'pong')
  elif mess == 'pong':
    # elapsed time is now - time when I sent the ping
    mycontext['latency'][srcip] = getruntime() - mycontext['sendtime'][srcip]

  elif mess.startswith('share'):
    mycontext['row'][srcip] = mess[len('share'):]
```

Format latency data into HTML
9 LOC

```
def encode_row(rowip, neighborlist, latencylist):

  retstring = "<tr><td>"+rowip+"</td>"
  for neighborip in neighborlist:
    if neighborip in latencylist:
      retstring = retstring + "<td>"+str(latencylist[neighborip])[:4]+"</td>"
    else:
      retstring = retstring + "<td>Unknown</td>"

  retstring = retstring + "</tr>"
  return retstring
```

Return a webpage
11 LOC

```
def show_status(srcip,srcport,connobj, ch, mainch):

  webpage = "<html><head><title>Latency Information</title></head><body><h1>Latency information from "+getmyip()+' </h1><table border="1">'

  webpage = webpage + "<tr><td></td><td>"+ "</td><td>".join(mycontext['neighborlist'])+"</td></tr>"

  for nodeip in mycontext['neighborlist']:
    if nodeip in mycontext['row']:
      webpage = webpage + mycontext['row'][nodeip]+'\n'
    else:
      webpage = webpage + '<tr><td>'+nodeip+'</td><td>No Data Reported</td></tr>\n'

  # now the footer...
  webpage = webpage + '</table></html>'

  # send the header and page
  connobj.send('HTTP/1.0 200 OK\nContent-Length: '+str(len(webpage))+'\nDate: Fri, 31 Dec 1999 23:59:59 GMT\nContent-Type: text/html\n\n'+webpage)

  # and we're done, so let's close this connection...
  connobj.close()
```

Initialization
15 LOC

```
if callfunc == 'initialize':

  # this holds the response information (i.e. when nodes responded)
  mycontext['latency'] = {}
  # this remembers when we sent a probe
  mycontext['sendtime'] = {}
  # this remembers row data from the other nodes
  mycontext['row'] = {}

  # get the nodes to probe
  mycontext['neighborlist'] = []
  for line in file("neighboriplist.txt"):
    mycontext['neighborlist'].append(line.strip())

  ip = getmyip()
  if len(callargs) != 1:
    raise Exception, "Must specify the port to use"
  pingport = int(callargs[0])

  # call gotmessage whenever receiving a message
  recvmess(ip,pingport,got_message)

  probe_neighbors(pingport)

  # we want to register a function to show a status webpage (TCP port)
  pageport = int(callargs[0])
  waitforconn(ip,pageport,show_status)
```

Potential Apps

- File backup
- Photo sharing
- Web caching
- Distributed web server
- Email Service
- Web proxy
- … etc.

Summary

Seattle makes p2p easy

- Safely executes Python
- Portable / interoperable
- Performance isolation
- Open source (MIT license)
- As of Oct 2009, we have resources on over 2000 computers, 100s of cities, 6 continents

P2P keeps free software free!

https://seattle.cs.washington.edu/