Distributed SAGE

Yi Qiang
Mathematics
University of Washington

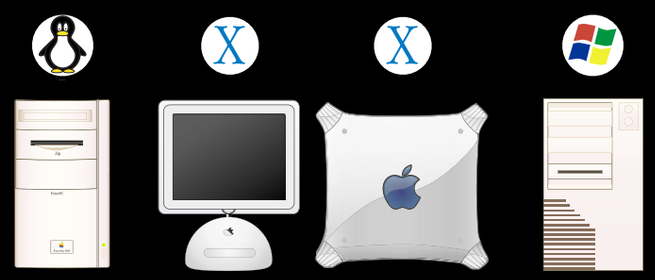
Objective

- Accessible
- Scalable
- Cross platform (any platform that SAGE supports)
- Make use of local and non local resources
  - Department of Hidden Resources

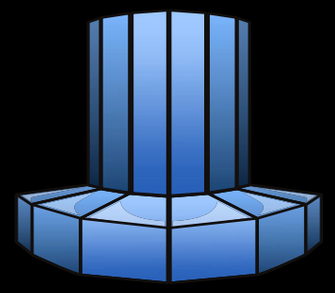
Brain dead simple
and yet reasonably powerful

Design

- Suited for *coarse* parallel tasks
  - Not e*mbarrassingly* parallelizable
  - PP: **Proudly Parallelizable**
- Low requirements for synchronization



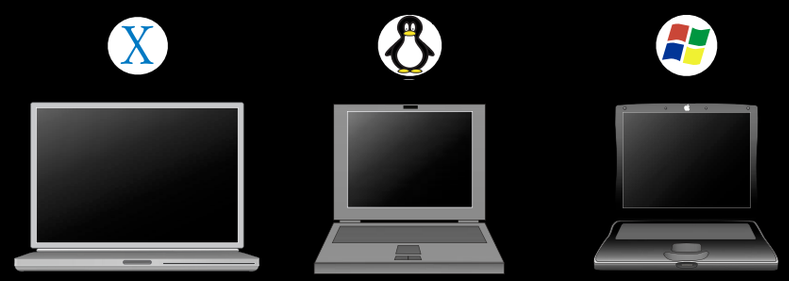
Workers



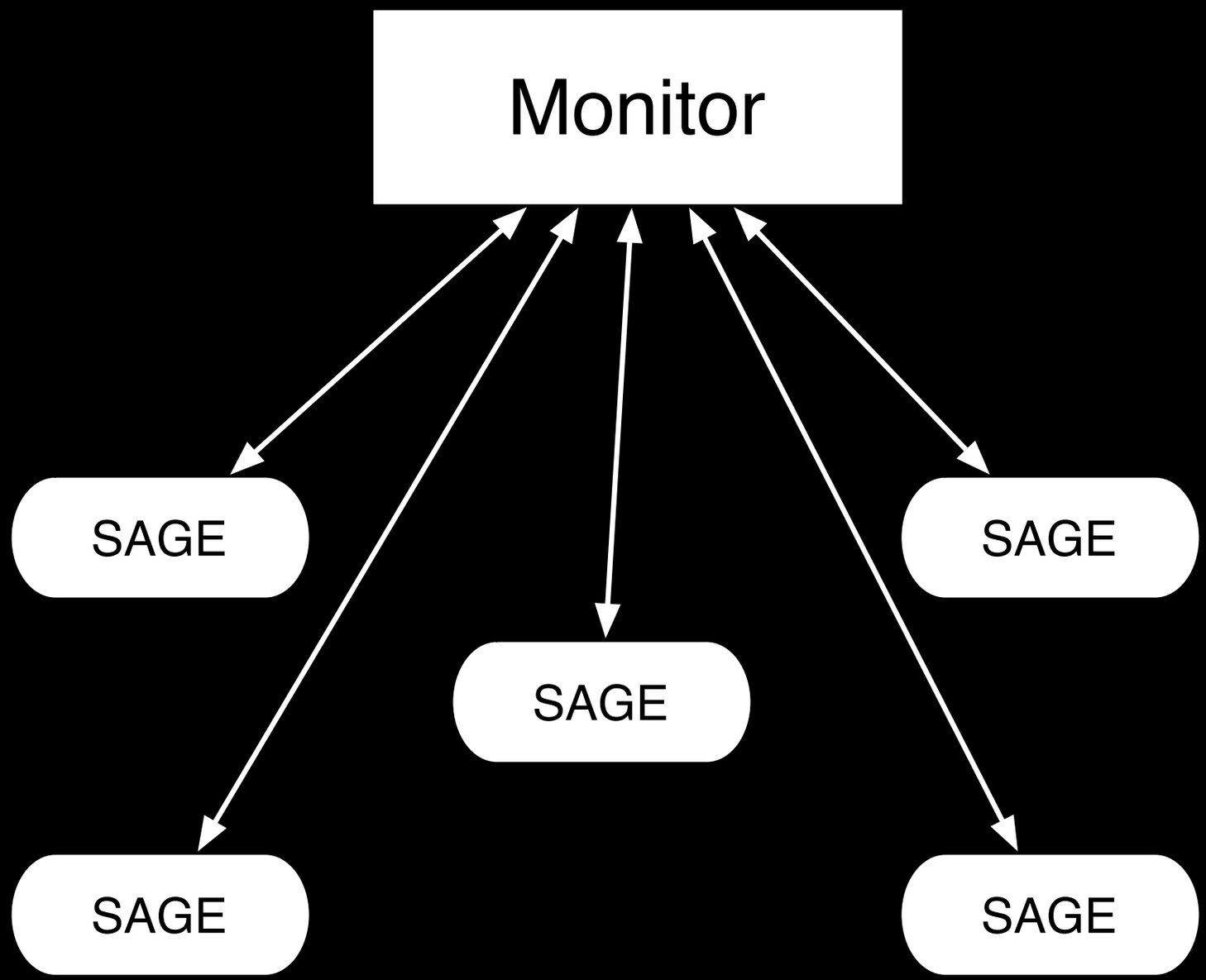
SERVER

Internet

Clients



Workers



Technologies

- Python
- Twisted
  - “Build the engine of *your* Internet”
- ZODB
  - Zope Object Database

Why DSage?

- Xgrid
- BOINC
- Chainsaw (AKA IPython1)
- Commercial Grid Computing solutions

Xgrid

and other commercial solutions

- Proprietary
- Only useful for long running jobs
- Knows nothing about math/science
- Only works on OSX

BOINC

- Barrier of entry is too high
- Suitable for one type of job

Chainsaw

- No fault tolerance (yet)
- No authentication (yet)
- Not tightly integrated with SAGE

Fault tolerance

- Servers can disappear at any time
- Workers can disappear at any time
- Clients can disappear at any time

Security

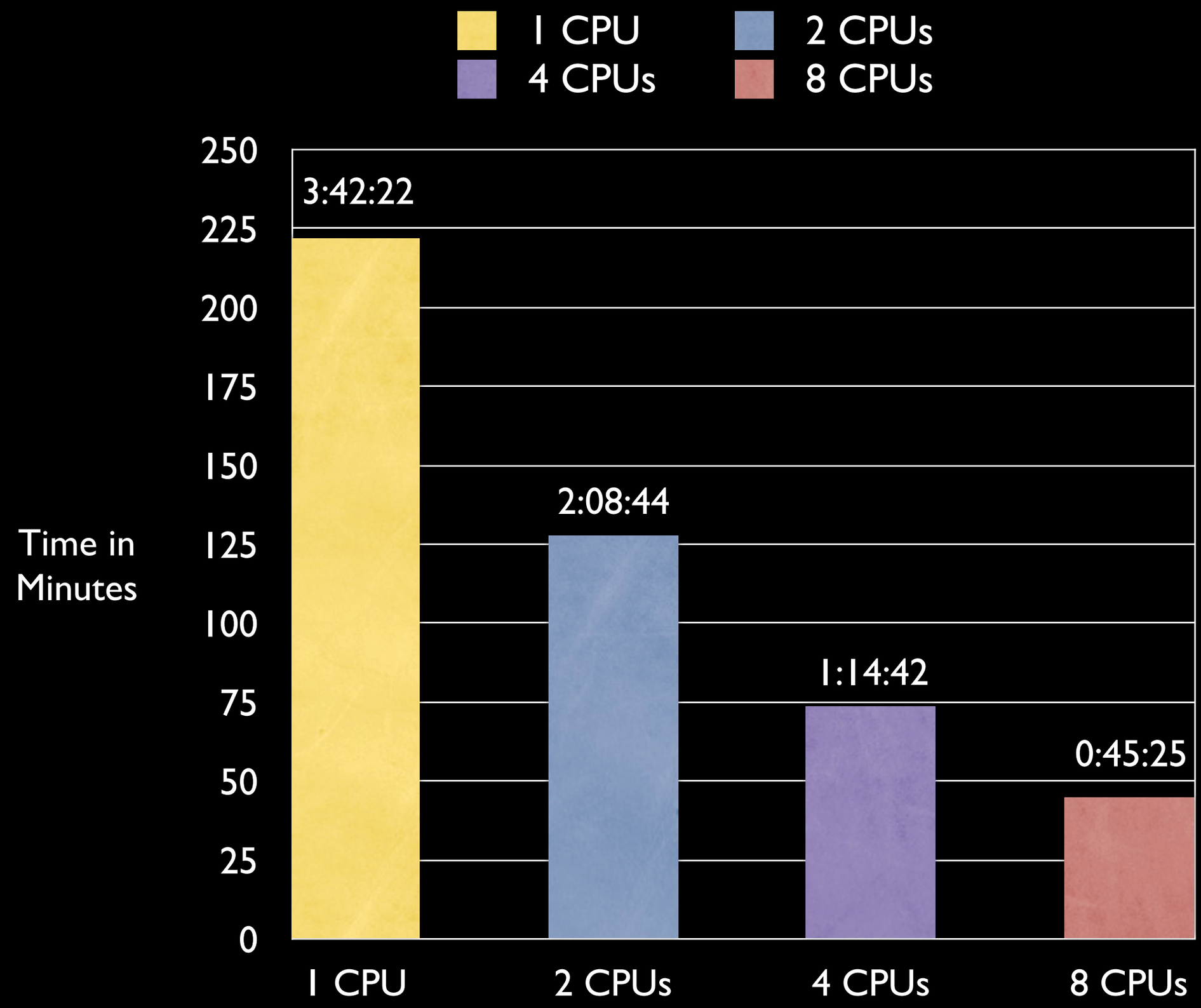
- SSL communication by default
- Public key authentication of clients
- Run workers in a virtual machine (in the near future)

Applications

- Implemented DistributedFunctions
  - Distributed Ray-Tracing
  - Distributed Integer Factorization
- Many other possibilities
  - http://www.llnl.gov/computing/tutorials/parallel_comp/#Examples

Benchmarks

POVRay


1 CPU
2 CPUs
4 CPUs
8 CPUs
Time in Minutes
250
225
200
175
150
125
100
75
50
25
0
3:42:22
2:08:44
1:14:42
0:45:25
1 CPU
2 CPUs
4 CPUs
8 CPUs

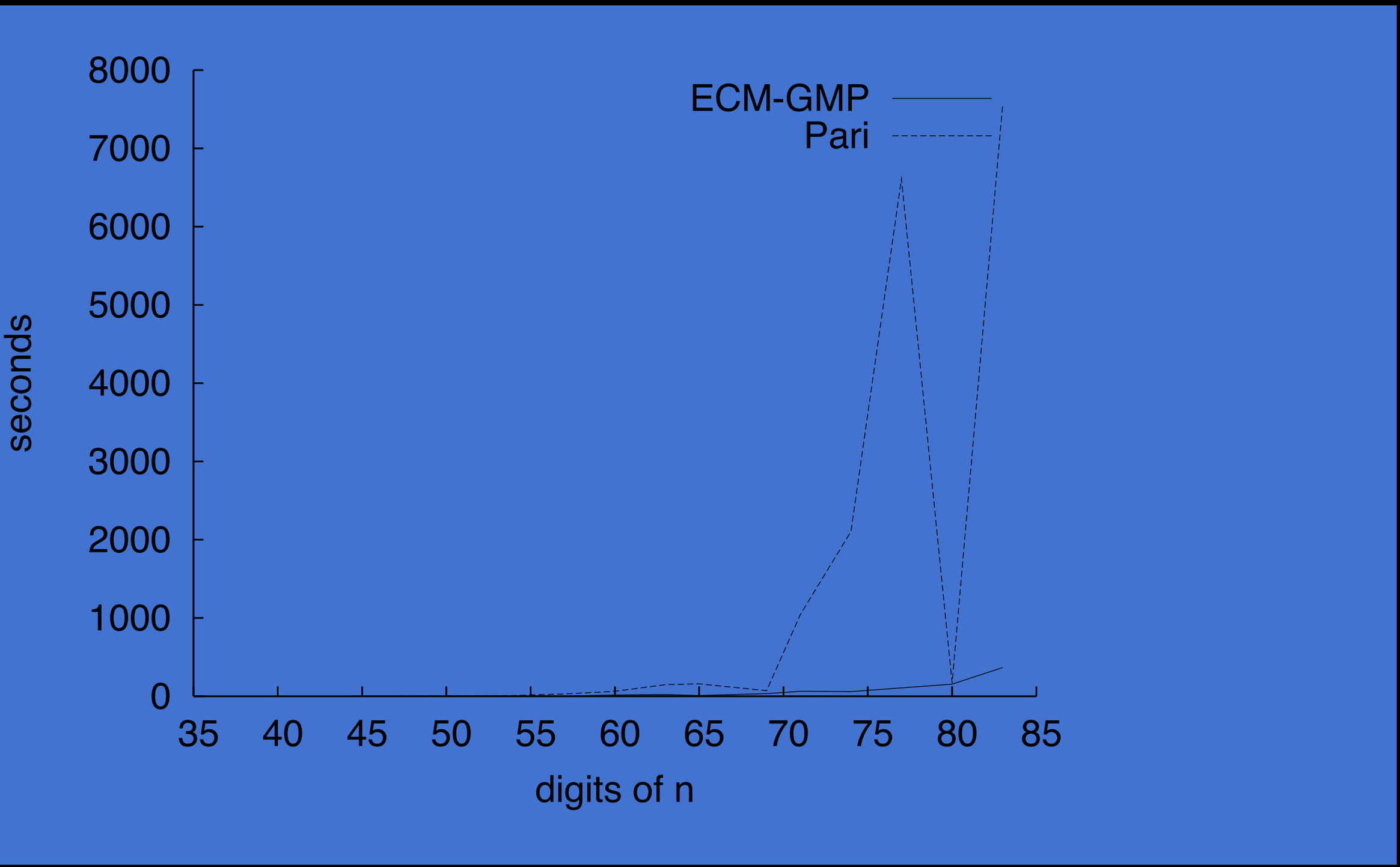

POV-Ray
POV-Ray

Distributed Factorization

- Run trial division up to n (default=10000)
- Perform ECM
  - Also do qsieve (implemented yesterday)

ECM

- 11^150+1
- DistributedFactor
  - Time: 0:02:08
- SAGE factor (PARI)
  - Time: 4:10:29



Benchmark by Robert Bradshaw

Writing your own

- Simply subclass DistributedFunction
  - *class MyDistributedJob(DistributedFunction):*
- Most things are taken care for you
  - Submission
  - Collecting results
  - Saving jobs
  - Restoring jobs later on

- Implement your own '*process_result()*'
  - Checks result of jobs
  - Marks job as complete/incomplete
- Implement your own '*next_job()*'
  - Creates new jobs
  - Keeps track of job parameters

Simple example

```
class DistributedTestFunction(DistributedFunctions):
    def __init__(self, DSage, n, name='DistributedTestFunction'):
        DistributedFunctions.__init__(self, DSage)
        self.n = n
        self.name = name
        self.result = 0
        self.results = []
        self.jobs = ["print %s"%i for i in range(1,n+1)]

    def process_result(self, job):
        if self.result == ((self.n*self.n+1) / 2):
            self.done = True
            return
        self.result += job.result
```

Demo

Future development

- Tracking and scoring of users
  - Possible reward system
- Integrity checking schemes
  - Redundancy
- Run workers SETI style
- More distributed computation examples!

Accessibility

- Included in SAGE since 1.7
- Latest version: dsage-0.2
- Get latest spkg from:
  - http://www.yiqiang.net/dsage

Thanks

- MSRI
- William Stein
- VIGRE