Using SimPy to Model AWS Autoscaling for Realtime Computations

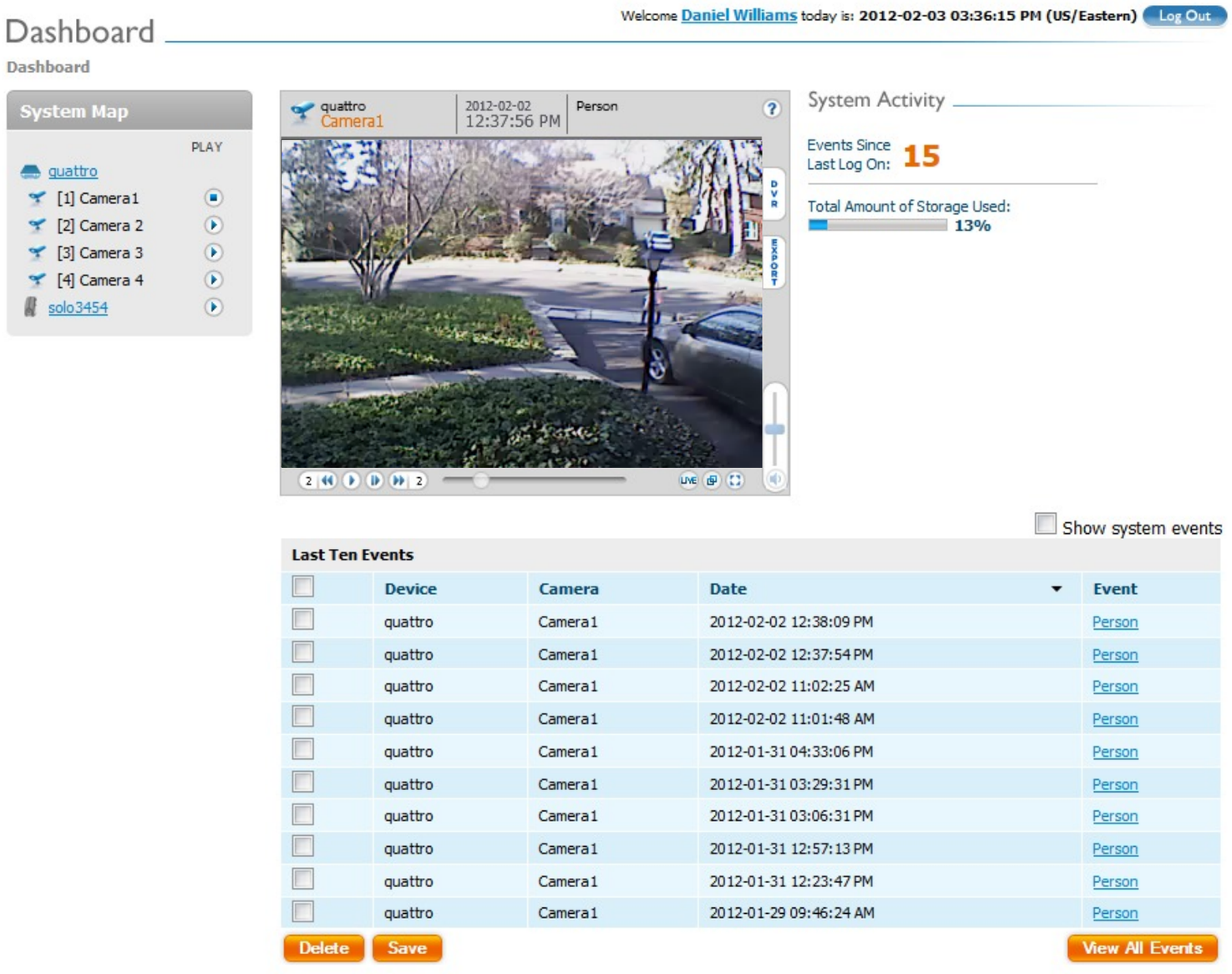
PhillyPUG April 2012

http://danielwilliams.org

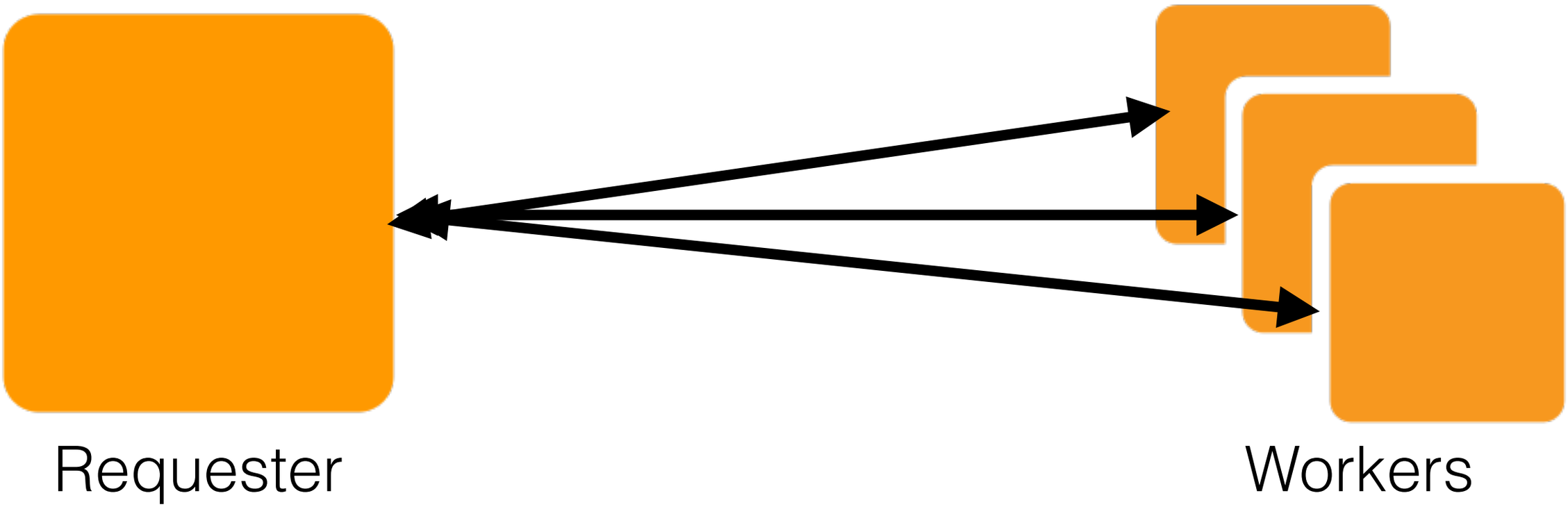
Wireless Video Surveillance

Archerfish Video Camera

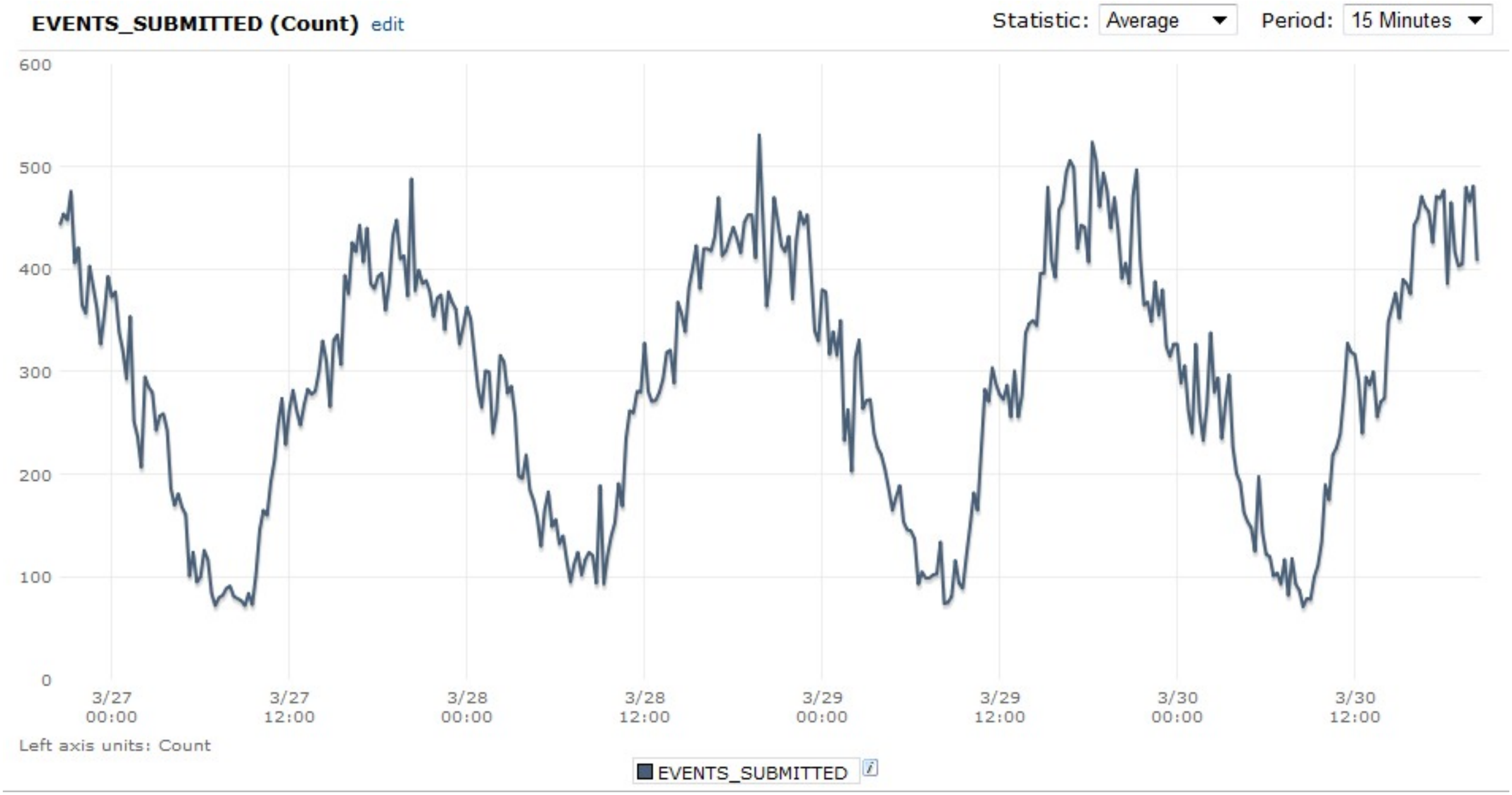
Interact using web...

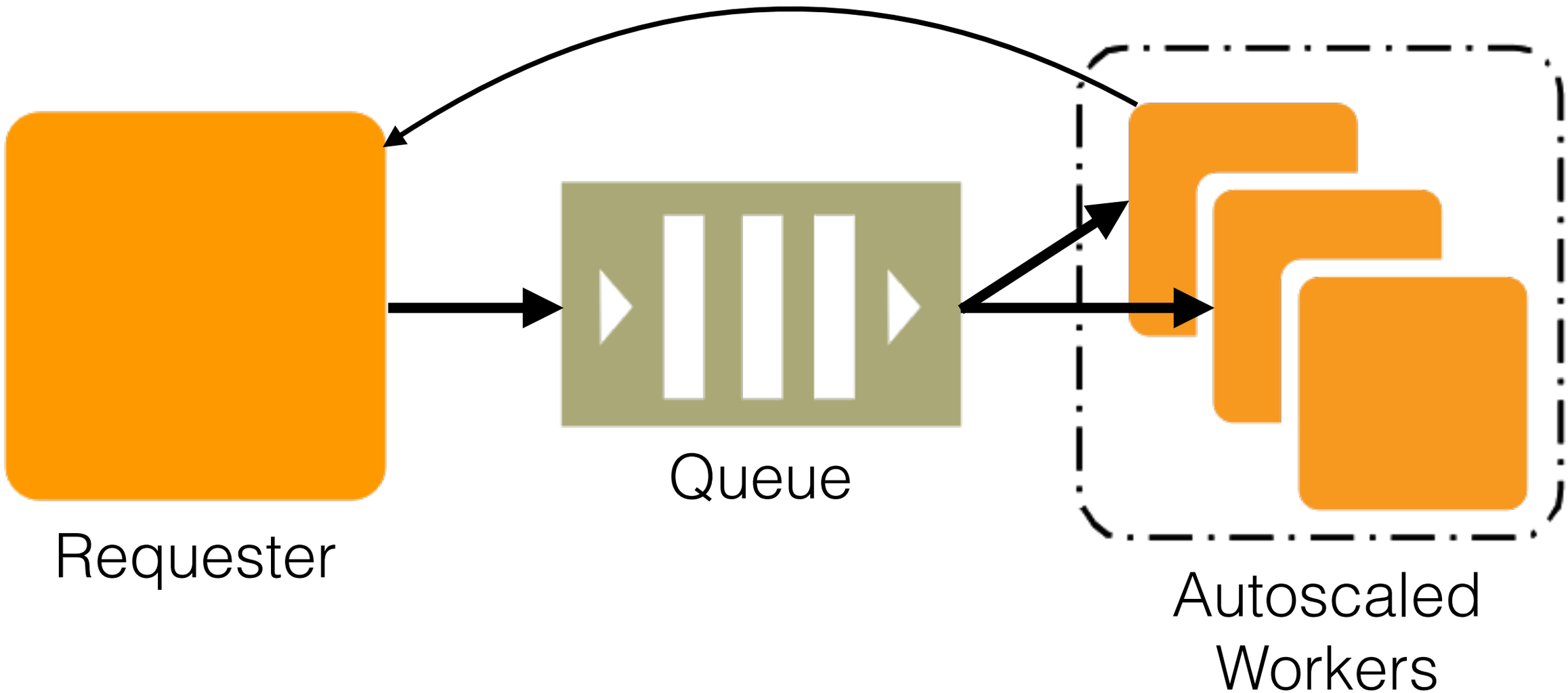


...and mobile devices


Requester
Workers

Our event load is cyclical

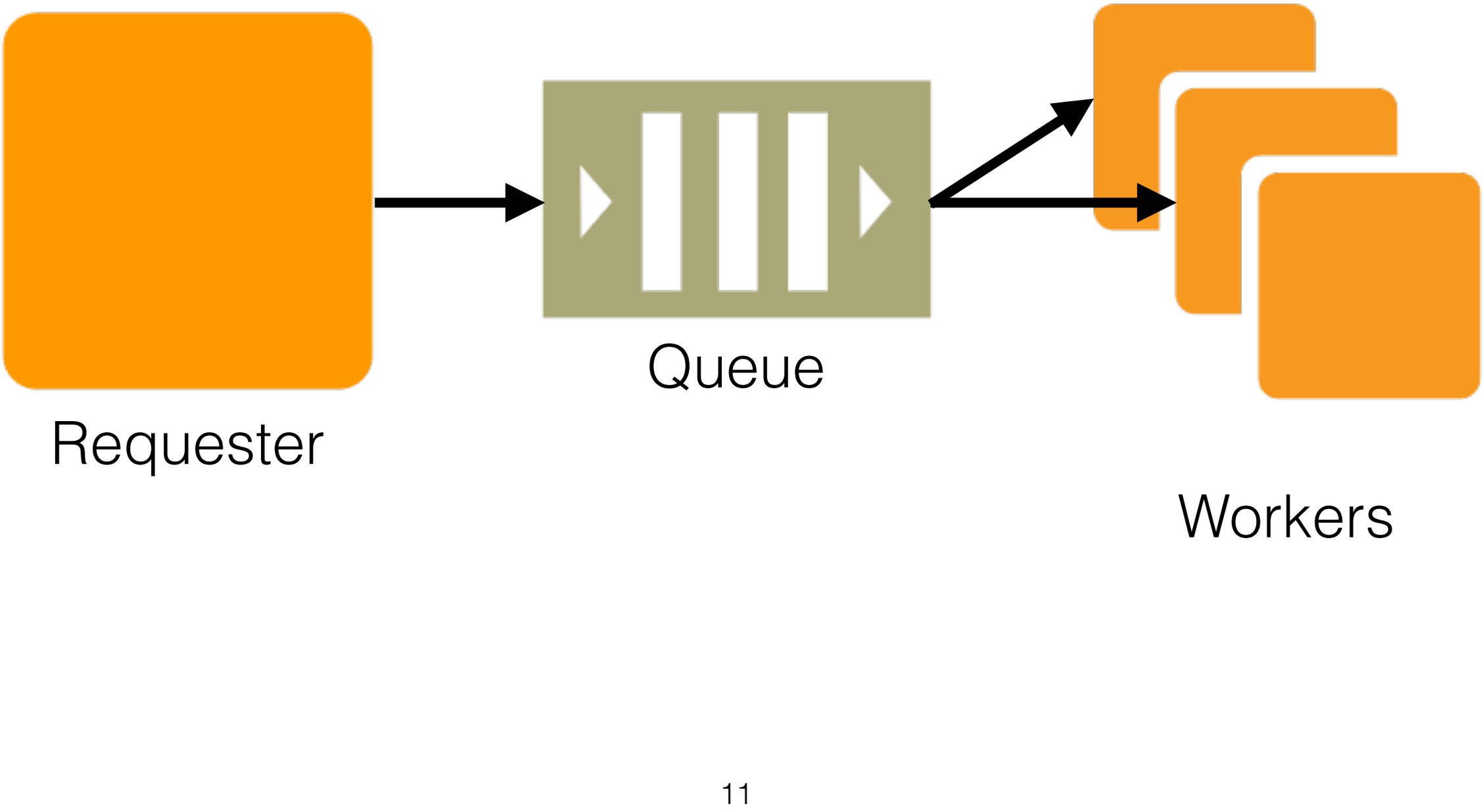



Requester
Queue
Autoscaled
Workers

Use discrete event simulation to model proposed solution

- uses 3 main object classes (Process, Resource, Monitor)
- uses Python generators as a sort of coroutine

A Very Simple Example



```
#
# simulate simple M/D/C system
#
import random                                       # 1
from itertools import count                         # 2

from SimPy.Simulation import *                      # 3

class Job(Process):                                 # 4
  def execute(self, service, res, total):           # 5
    start = now()                                   # 6
    yield request, self, res                        # 7
    yield hold, self, service                       # 8
    yield release, self, res                        # 9
    total.observe(now() - start)                    # 10
```

```
class JobSource(Process):                                          # 1
  def __init__(self, name):                                        # 2
    Process.__init__(self, name)                                   # 3
    self.total = Monitor('total time')                             # 4

  def generate(self, interval, service, res):                      # 5
    for i in count():                                              # 6
      j = Job(name='Job-%d' % i)                                   # 7
      activate(j, j.execute(service, res, self.total)) # 8
      yield hold, self, random.expovariate(1.0 / interval)
```

```
if __name__ == '__main__':                                      # 1
  for cap in range(1, 7):                                       # 2
    initialize()                                                # 3
    servers = Resource(cap, monitored=True)                     # 4
    js = JobSource('JobSource')                                 # 5
    t = 80.0                                                    # 6
    activate(js, js.generate(interval=30.0, service=t,          # 7
                             res=servers))                      # 8
    simulate(until=24*60*60)                                    # 9
    print 'cap : %d  util : %3d%%  wait: %5.0fs' % \
        (cap,
         int(100 * servers.actMon.timeAverage() /
             servers.capacity),
         js.total.mean() - t)
```

Results for one run

Capacity	Utilization	Avg Wait
1	100%	27202s
2	99%	10591s
3	88%	75s
4	66%	12s
5	54%	4s
6	42%	1s

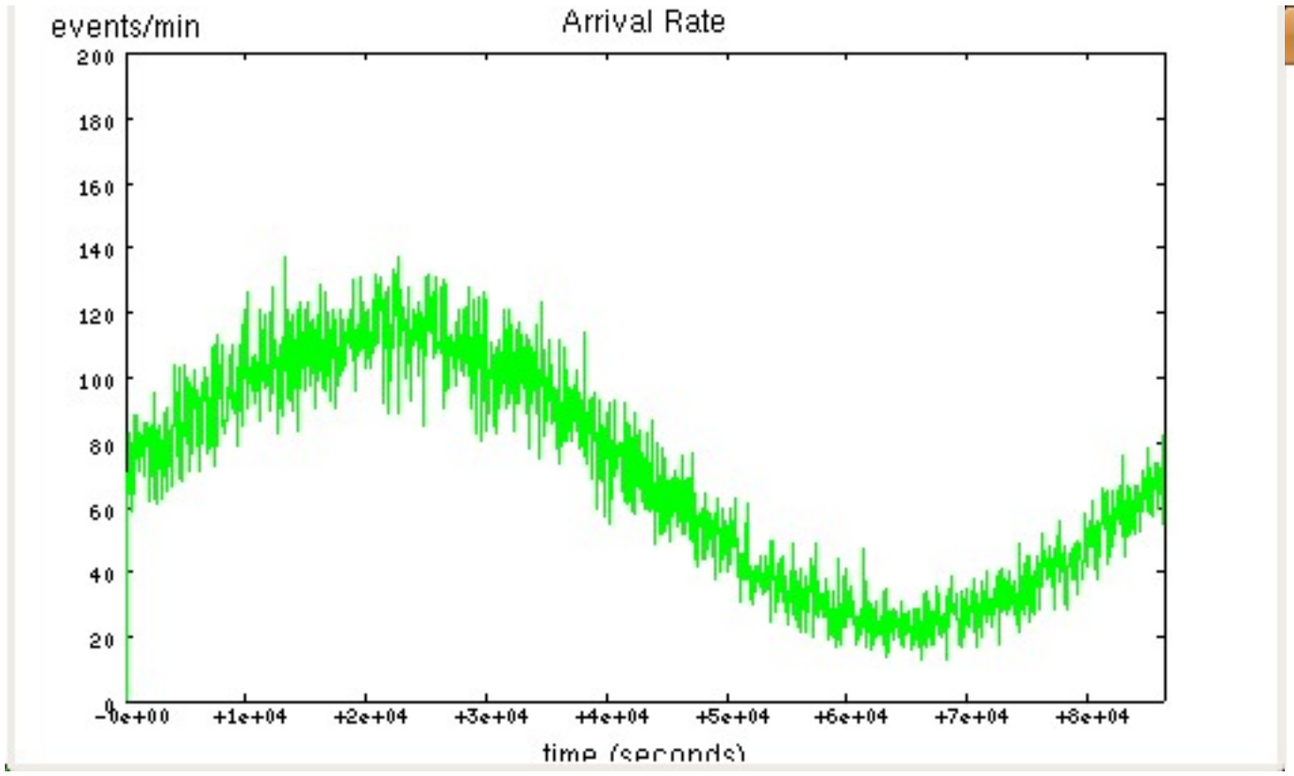
The AWS Simulation

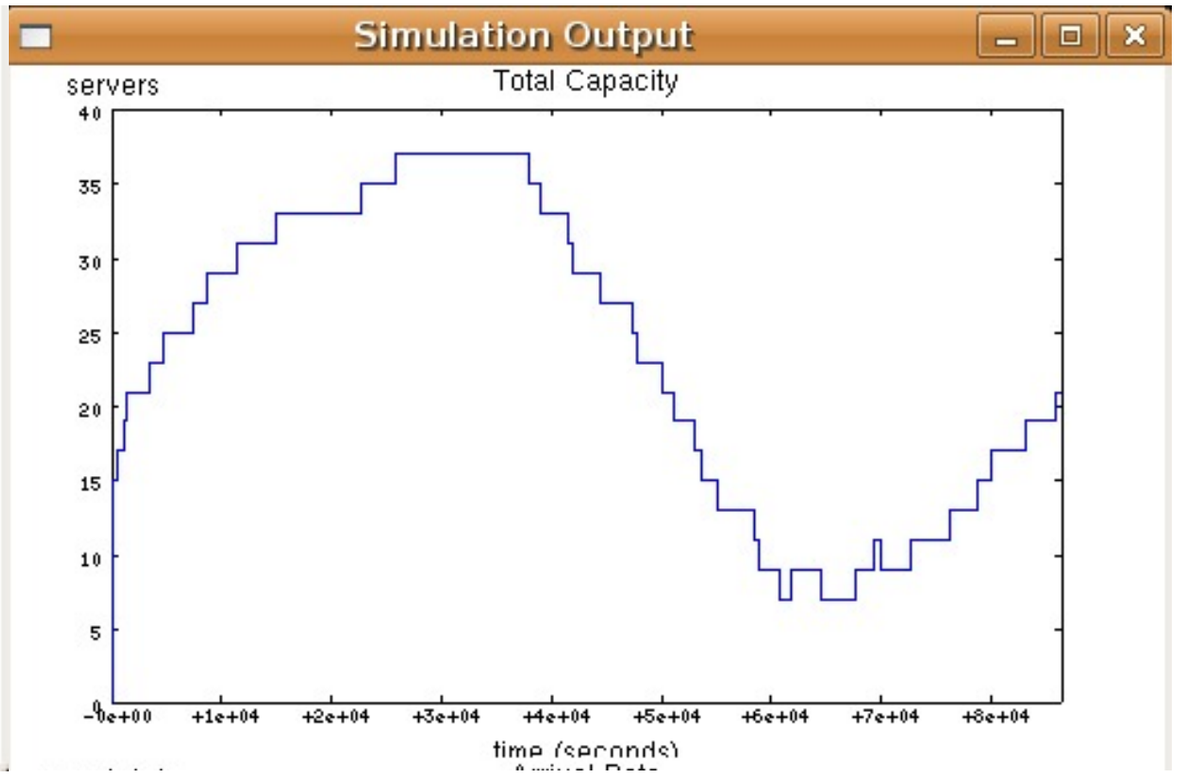
- Is the same idea with many more features
- Servers are modeled as active elements (Processes, not Resources)
- Service time distribution comes from real world data
- Have a watcher Process that implements the Autoscaling algorithms

Simulation Run Results

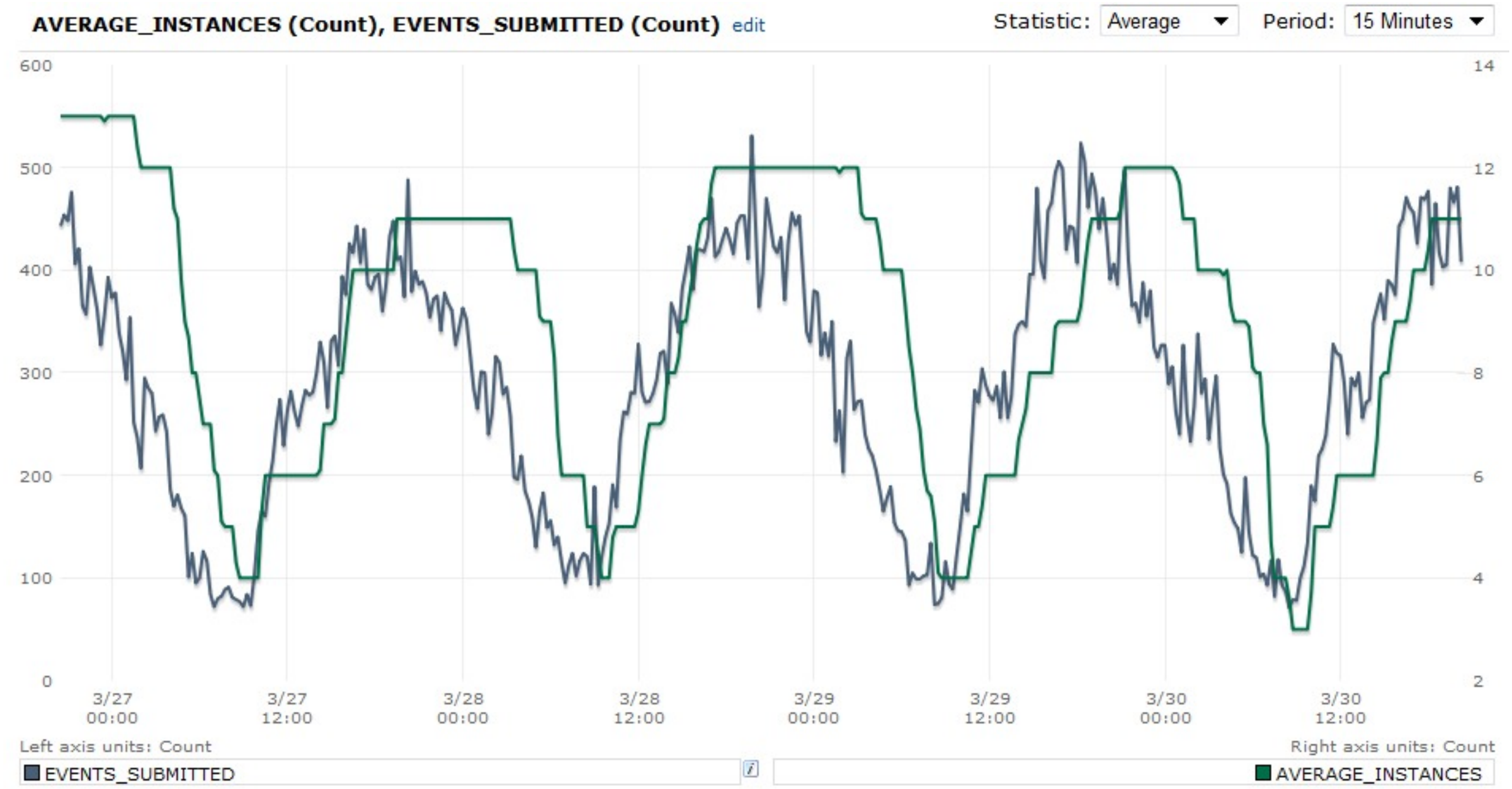
```
$ ./verifyd-simulation.py \
      --plot \
      --encoders 5000 \
      --capacity 15 \
      --si_amplitude 0.65 \
      --as_lower_threshold 40 \
      --as_upper_threshold 60 \
      --as_breach_duration 300 \
      --as_upper_breach_scale_increment=2 \
      --as_lower_breach_scale_increment=-2

utilization                  : 43.7%  (goal: > 50%)
requests served in < 30s : 95.7%  (goal: > 95%)
requests timed out (180s):  0.11% (goal: < 0.1%)
```

Simulated Autoscaling





Autoscaling works as simulated



Things not done

- Inferring distribution of results from repeated runs
- Using this program as input to a parametric optimization program