



# Discrete-event simulation with SimPy

Stefan Scherfke  
OFFIS – Institute for Information Technologie  
R&D Divison Energy  
July 25, 2014



**2002:**

Started by *Klaus G. Müller*  
and *Tony Vignaux*

**2008:**

*Ontje Lünsdorf's* and my  
first contributions

**2011:**

Ontje and I became  
project maintainers

**2013:**

SimPy 3 released





Environment

Process

Event

Resource

Event loop

Task / Coroutine

Future / Promise / Deferred

Semaphore

# Generator functions for modeling processes

```
>>> def generator(x):  
...     y = yield x + 1  
...     return y + 1  
...  
>>> g = generator(1)  
>>> next(g)  
2  
>>> g.send(3)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
StopIteration: 4
```

```
>>> import simpy
>>>
>>> def clock(env, name, tick):
...     while True:
...         print(name, env.now)
...         yield env.timeout(tick)
...
>>> env = simpy.Environment()
>>>
>>> env.process(clock(env, 'fast', 0.5))
<Process(clock) object at 0x...>
>>> env.process(clock(env, 'slow', 1))
<Process(clock) object at 0x...>
>>>
>>> env.run(until=2)
fast 0
slow 0
fast 0.5
slow 1
fast 1.0
fast 1.5
```

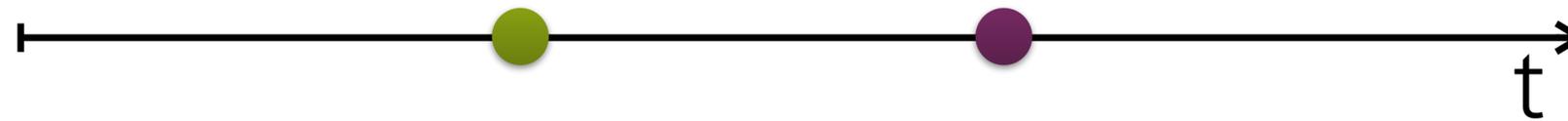
# Environment

simulate "as fast as possible"

# RealtimeEnvironment

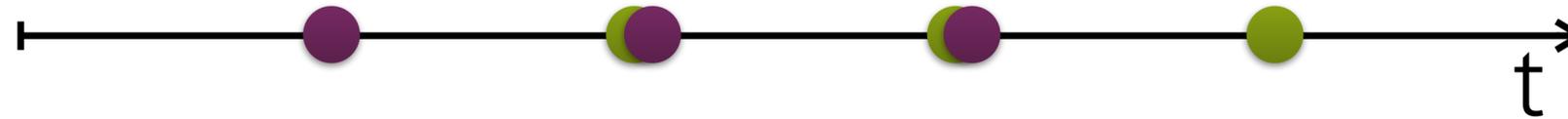
synchronized with wall-clock time

# Timeout events let time pass



```
def speaker(env, start):  
    until_start = start - env.now  
    ● yield env.timeout(until_start)  
    ● yield env.timeout(30)
```

# Processes are events, too



```
def speaker(env):  
    yield env.timeout(30)  
    return 'handout'
```

```
def moderator(env):  
    for i in range(3):  
        val = yield env.process(speaker(env))  
        print(val)
```

# Asynchronous interrupts



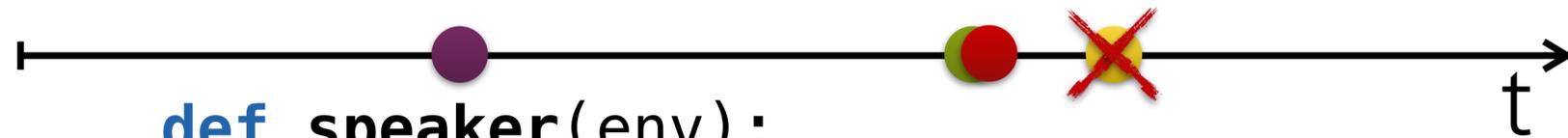
```
def speaker(env):  
    try:
```

```
        yield env.timeout(randint(25, 35))  
    except simpy.Interrupt as interrupt:  
        print(interrupt.cause)
```

```
def moderator(env):
```

```
    for i in range(3):  
        speaker_proc = env.process(speaker(env))  
        yield env.timeout(30)  
        speaker_proc.interrupt('No time left')
```

# Condition events



```
def speaker(env):  
    try:  
        yield env.timeout(randint(25, 35))  
    except simpy.Interrupt as interrupt:  
        print(interrupt.cause)
```

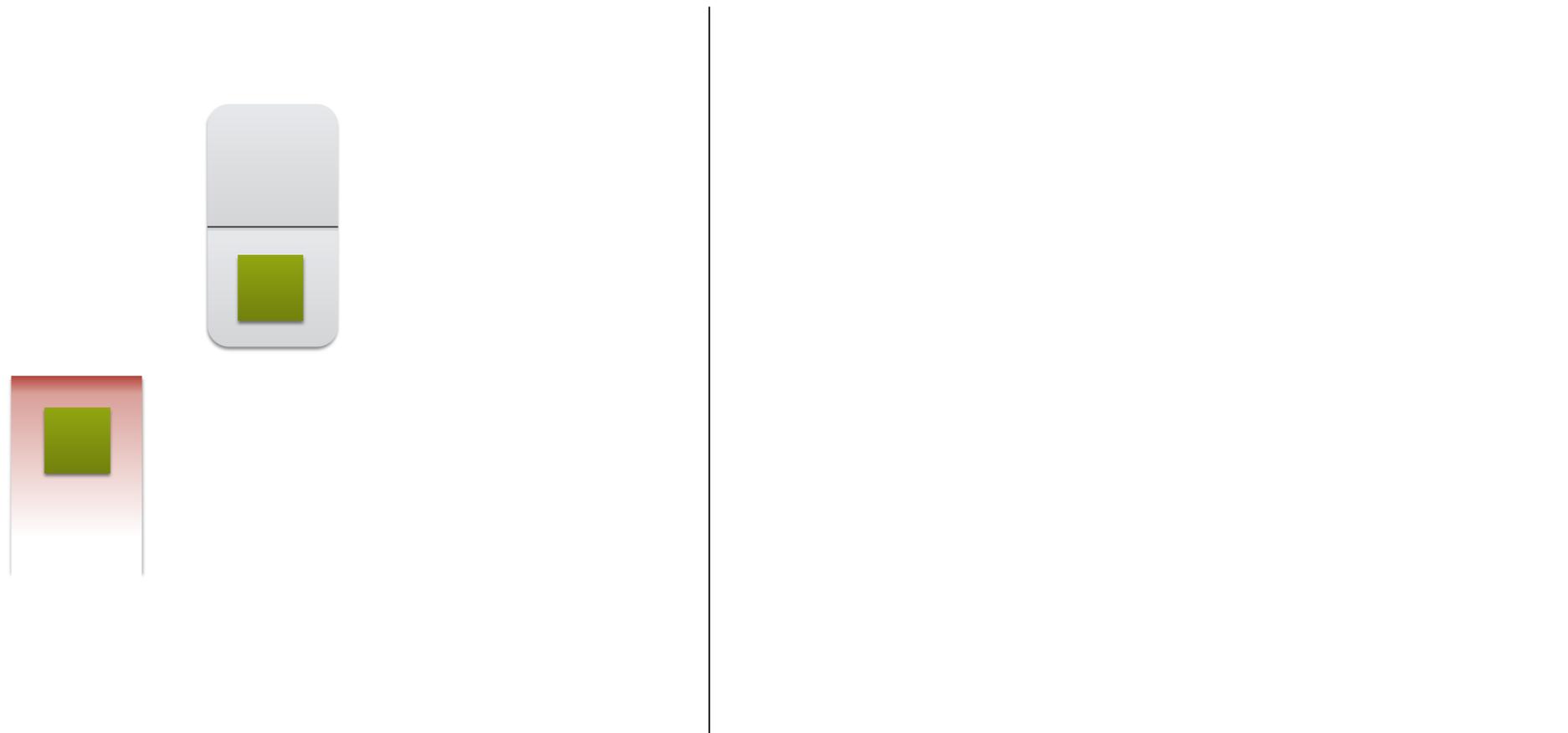
```
def moderator(env):  
    for i in range(3):  
        speaker_proc = env.process(speaker(env))  
        results = yield speaker_proc | env.timeout(30)  
  
        if speaker_proc not in results:  
            speaker_proc.interrupt('No time left')
```

# Shared resources



# Shared resources

Resource

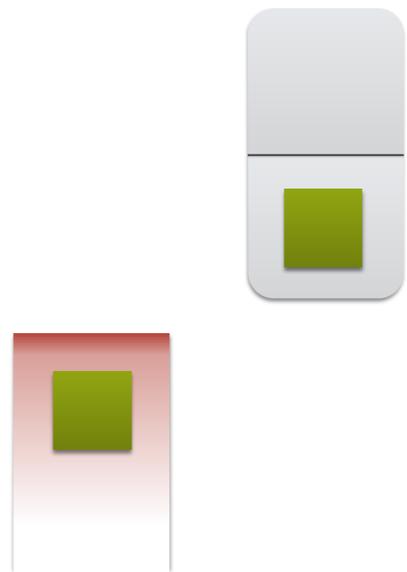


 Queue

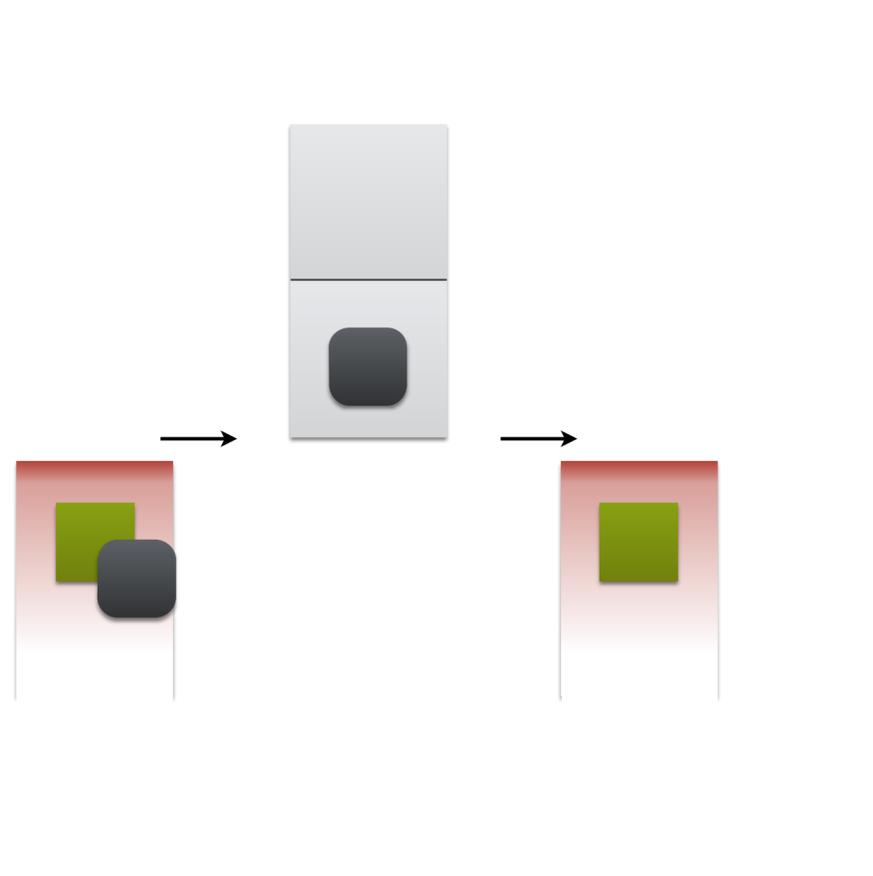
 Process

# Shared resources

Resource



Store



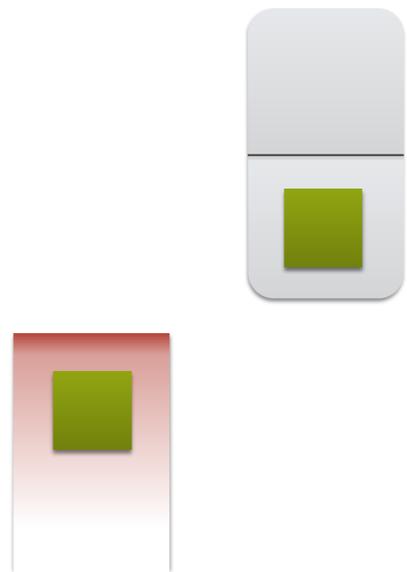
 Queue

 Process

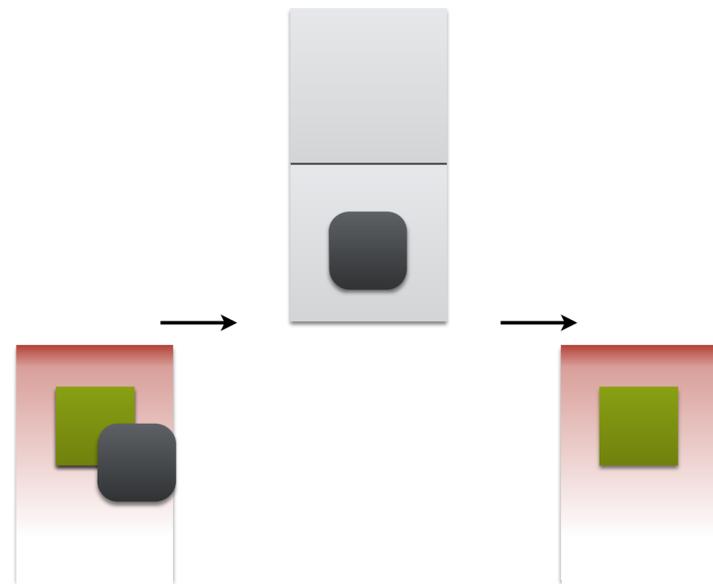
 Object

# Shared resources

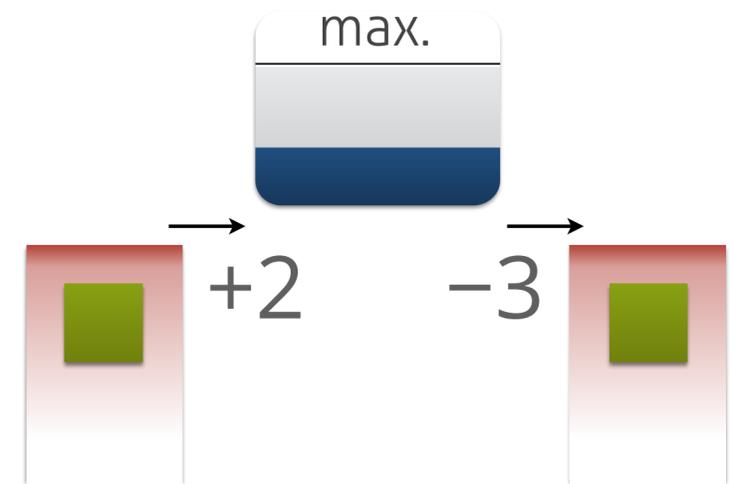
Resource



Store



Container



 Queue     Process     Object

# Example: Conference Attendee



imports

```
from random import randint
import simpy
```

config

```
TALKS_PER_SESSION = 3
TALK_LENGTH = 30
BREAK_LENGTH = 15
```

repeat sessions

```
def attendee(env, name, knowledge=0, hunger=0):
    while True:
        # Visit talks
        for i in range(TALKS_PER_SESSION):
            knowledge += randint(0, 3) / (1 + hunger)
            hunger += randint(1, 4)

            yield env.timeout(TALK_LENGTH)

        print('Attendee %s finished talks with knowledge %.2f and hunger '
              '%.2f.' % (name, knowledge, hunger))

        # Go to buffet
        food = randint(3, 12)
        hunger -= min(food, hunger)

        yield env.timeout(BREAK_LENGTH)

        print('Attendee %s finished eating with hunger %.2f' % (name, hunger))
```

visit talks

take a break

run simulation

```
env = simpy.Environment()
for i in range(5):
    env.process(attendee(env, i))
env.run(until=220)
```

Attendee 0 finished talks with knowledge 3.48 and hunger 9.00.  
Attendee 1 finished talks with knowledge 2.67 and hunger 5.00.  
Attendee 2 finished talks with knowledge 4.08 and hunger 4.00.  
Attendee 3 finished talks with knowledge 2.67 and hunger 5.00.  
Attendee 4 finished talks with knowledge 0.33 and hunger 5.00.  
Attendee 0 finished eating with hunger 0.00  
Attendee 1 finished eating with hunger 1.00  
Attendee 2 finished eating with hunger 0.00  
Attendee 3 finished eating with hunger 0.00  
Attendee 4 finished eating with hunger 0.00  
Attendee 0 finished talks with knowledge 4.38 and hunger 7.00.  
Attendee 1 finished talks with knowledge 4.29 and hunger 10.00.  
Attendee 2 finished talks with knowledge 7.62 and hunger 10.00.  
Attendee 3 finished talks with knowledge 6.20 and hunger 10.00.  
Attendee 4 finished talks with knowledge 3.67 and hunger 7.00.  
Attendee 0 finished eating with hunger 0.00  
Attendee 1 finished eating with hunger 0.00  
Attendee 2 finished eating with hunger 7.00  
Attendee 3 finished eating with hunger 6.00  
Attendee 4 finished eating with hunger 0.00

A buffet table is shown with various dishes. In the foreground, there are several small glass cups filled with a green salad, a plate of bread topped with sliced tomatoes and cheese, and a glass of water. In the background, a white sign is visible with the text "Example: Conference Attendee at the Buffet". To the right, there are more dishes, including a large stack of breads and pastries wrapped in plastic. The table is set with blue placemats and silverware holders.

Example:  
Conference Attendee  
at the Buffet

more config

```
[...]  
DURATION_EAT = 3  
BUFFET_SLOTS = 1
```

same as before

```
def attendee(env, name, buffet, knowledge=0, hunger=0):  
    while True:  
        # Visit talks  
        [...]
```

try to get to the buffet in time

```
        # Go to buffet  
        start = env.now  
        with buffet.request() as req:  
            yield req | env.timeout(BREAK_LENGTH - DURATION_EAT)  
            time_left = LEN_BREAK - (env.now - start)
```

got to the buffet

```
            if req.triggered:  
                food = min(randint(3, 12), time_left) # Less time -> less food  
                yield env.timeout(DURATION_EAT)  
                hunger -= min(food, hunger)  
                time_left -= DURATION_EAT  
                print('Attendee %s finished eating with hunger %.2f' %  
                      (name, hunger))
```

onooes!

```
            else:  
                hunger += 1 # Penalty for only taking a look at all the food.  
                print('Attendee %s didn't make it to the buffet, hunger is now '  
                      'at %.2f.' % (name, hunger))
```

wait

```
            yield env.timeout(time_left)
```

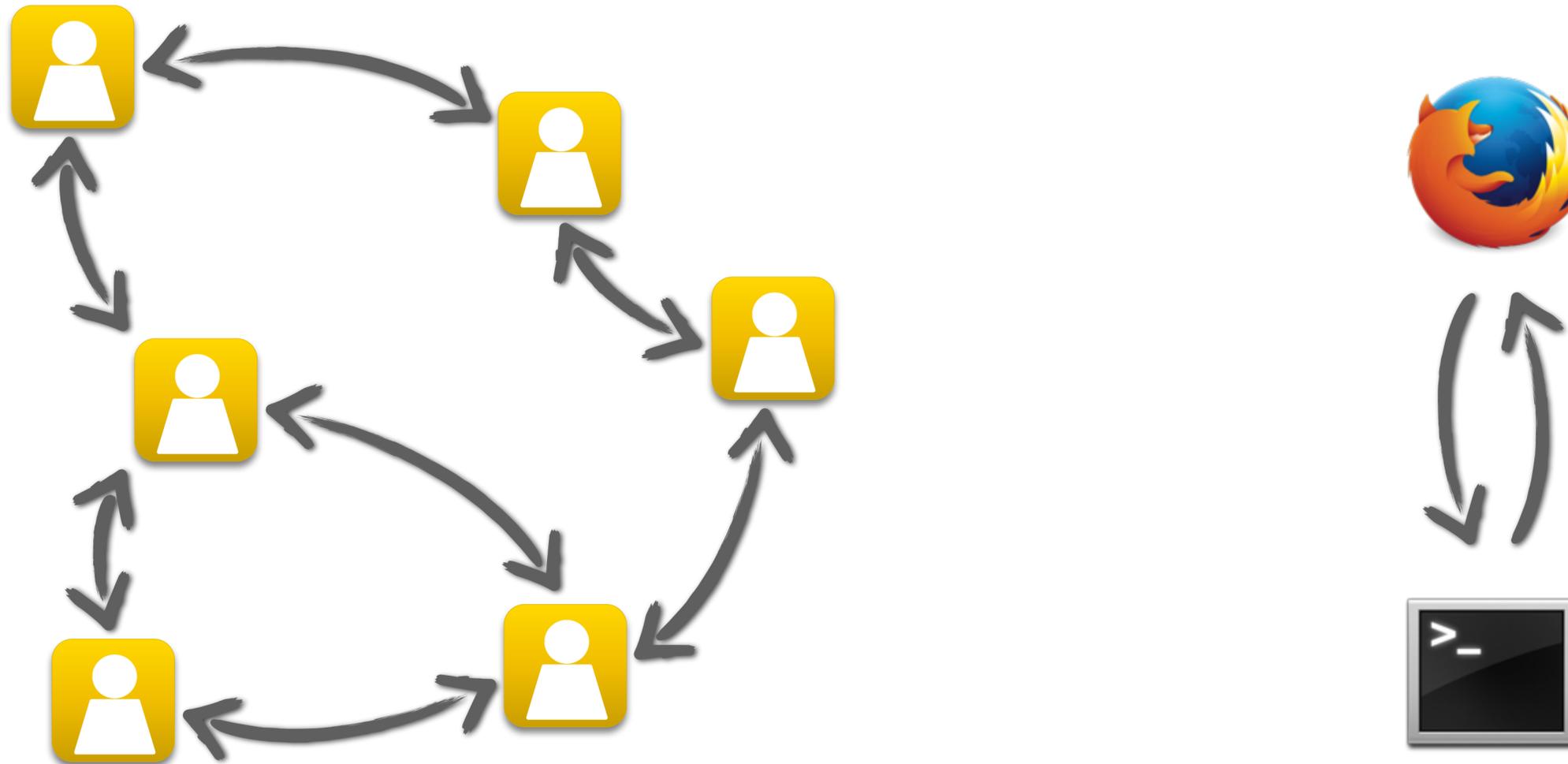
run simulation

```
env = simpy.Environment()  
buffet = simpy.Resource(env, capacity=BUFFET_SLOTS)  
for i in range(5):  
    env.process(attendee(env, i, buffet))  
env.run(until=220)
```

Attendee 0 finished talks with knowledge 1.33 and hunger 9.00.  
Attendee 1 finished talks with knowledge 2.00 and hunger 5.00.  
Attendee 2 finished talks with knowledge 3.20 and hunger 11.00.  
Attendee 3 finished talks with knowledge 2.50 and hunger 5.00.  
Attendee 4 finished talks with knowledge 1.50 and hunger 6.00.  
Attendee 0 finished eating with hunger 1.00  
Attendee 1 finished eating with hunger 0.00  
Attendee 2 finished eating with hunger 2.00  
Attendee 3 finished eating with hunger 0.00  
Attendee 4 didn't make it to the buffet, hunger is now at 7.00.  
Attendee 0 finished talks with knowledge 5.42 and hunger 7.00.  
Attendee 1 finished talks with knowledge 6.33 and hunger 5.00.  
Attendee 2 finished talks with knowledge 7.68 and hunger 8.00.  
Attendee 3 finished talks with knowledge 5.93 and hunger 8.00.  
Attendee 4 finished talks with knowledge 2.14 and hunger 15.00.  
Attendee 0 finished eating with hunger 0.00  
Attendee 1 finished eating with hunger 2.00  
Attendee 2 finished eating with hunger 0.00  
Attendee 3 finished eating with hunger 2.00  
Attendee 4 finished eating with hunger 10.00

# simpy.io

Event-driven IO with  
real and simulated TCP sockets



# simpy.io

```
def client(env, client_sock):  
    message = Message(env, client_sock)  
    reply = yield message.send('ohai')  
    print(reply)  
  
def server(env, server_sock):  
    # Accept new connection  
    sock = yield server_sock.accept()  
    message = Message(env, PacketUTF8(sock))  
  
    # Get message and send reply  
    request = yield message.recv()  
    print(request.content)  
    yield request.succeed('cya')
```

# Plans for SimPy 3.x

- Keep the community happy :-)
- Minor optimizations
- Documentation improvements
- Helper functions for monitoring?

# Final notes

- Easy to use and flexible
- Documentation, mailing list, tests
- Pure Python (2.7, 3.2+), no dependencies
  - PyPy supported
  - `simpy-cython` [github.com/chaosmail/simpy-cython](https://github.com/chaosmail/simpy-cython)
  - SimSharp (SimPy in C#) [github.com/abeham/SimSharp](https://github.com/abeham/SimSharp)
- `simpy.io` for socket communication



[bitbucket.org/simpy](http://bitbucket.org/simpy)  
[simpy.rtfid.org](http://simpy.rtfid.org)

Stefan Scherfke  
@sscherfke