

# python-csp

## CSP as a DSL for Python and Jython

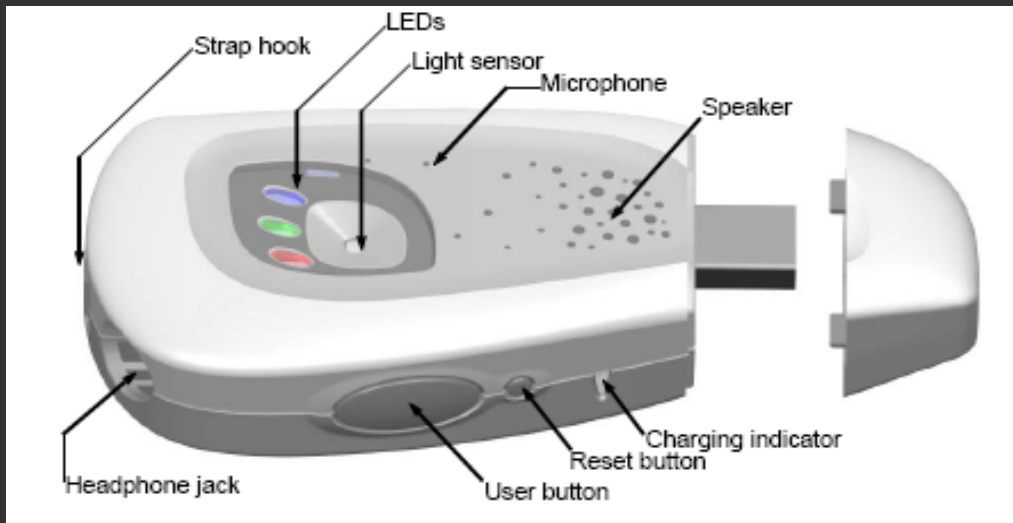
Sarah Mount, University of Wolverhampton

w:	<a href="http://www.snim2.org">http://www.snim2.org</a>
e:	<a href="mailto:s.mount@wlv.ac.uk">s.mount@wlv.ac.uk</a>
t:	@snim2

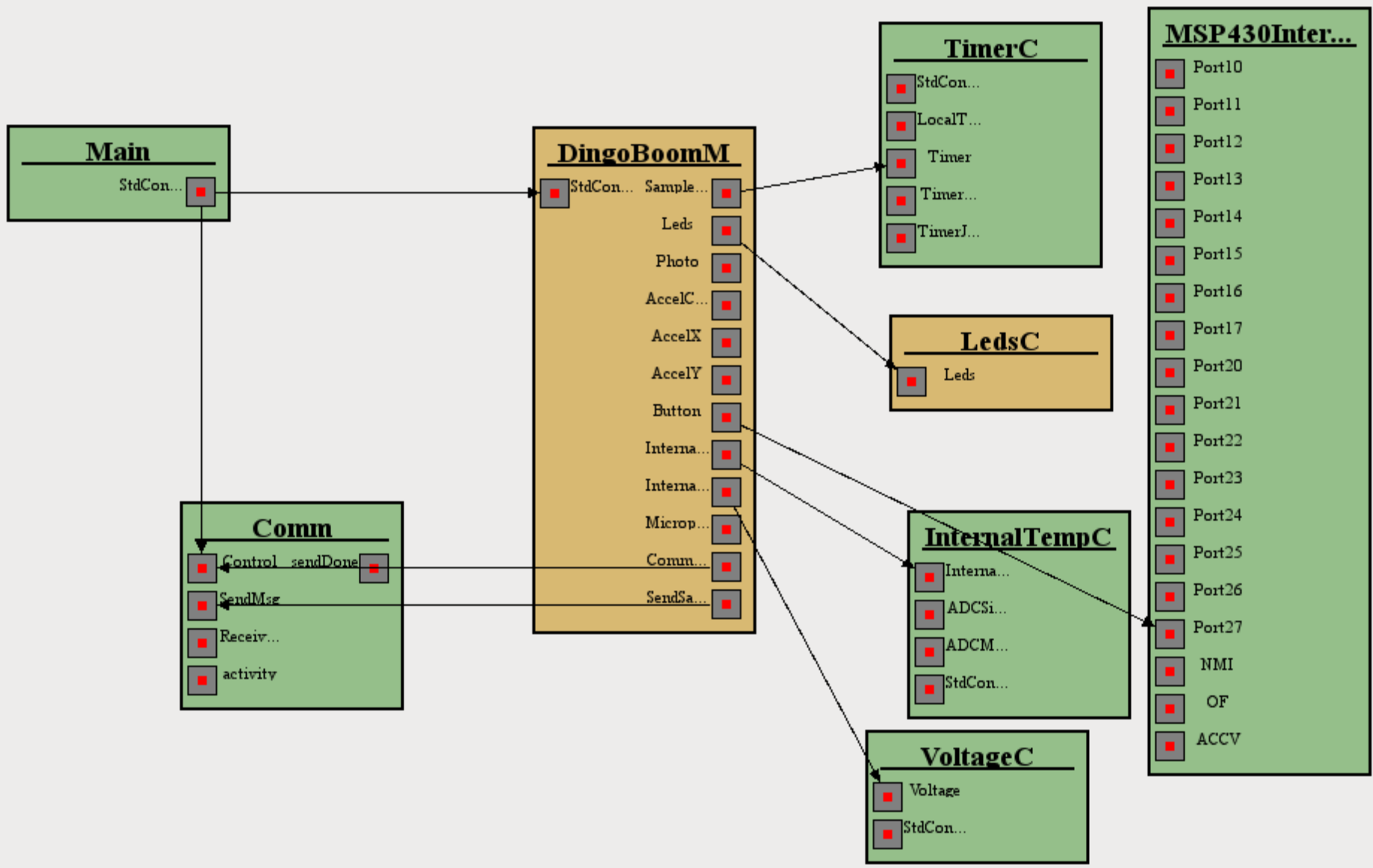
# Contents...

- Why we did this
- Advantages / disadvantages of Python for highly concurrent or process oriented work
- General theme of python-csp
- Syntax / semantics / examples
- Future directions

The story of this work ...



Tmote Invent platform from MoteIV (now Sentilla)



TinyOS code to gather raw data from Tmote Invent

```
module HL2ControllerM
{
    provides interface StdControl;
    uses { ... interface ADC as AccelX; ... }
}
implementation {
    task void getAccelXData() {
        call AccelX.getData();
    }
    async event result_t AccelX.dataReady(uint16_t
                                           data) {
        atomic am->accelX[nextX++] = data;
        post getAccelYData();
        return SUCCESS;
    }
}
```

... but what about the  
application layer?



Soil science and agronomy



-500

1000s

$$\Psi_m = \frac{R \cdot T_K}{V_w} \ln \left( \frac{\%RH}{100} \right)$$

$R = 8.314 \text{ J mol}^{-1} \text{ K}^{-1}$

$V_w = 18 \text{ mL/mol}$

water - module

et?  
var!!

EXPO L

%RH to Soil Matric Pressure









The image shows two overlapping windows of the 'Dingo development tool for wireless sensor networks'. The top window displays a network topology with nodes and connections, and a log of simulation events. The bottom window shows a more detailed network topology with nodes labeled S1 through S17 and a log of radio broadcast events.

**Top Window Log:**

```

SIM: tn value: 0.111111111111
SIM: random number value 0.537540527422
SIM: tn value: 0.111111111111
SIM: random number value 0.676084833671
SIM: tn value: 0.111111111111
SIM: random number value 0.600800820662
SIM: tn value: 0.111111111111
SIM: random number value 0.537540527422
SIM: tn value: 0.111111111111
SIM: random number value 0.676084833671
SIM: tn value: 0.111111111111
SIM: random number value 0.600800820662
SIM: tn value: 0.111111111111
SIM: random number value 0.537540527422
SIM: tn value: 0.111111111111
SIM: random number value 0.676084833671
SIM: tn value: 0.111111111111
SIM: random number value 0.600800820662
RADIO: Sensor[40]
RADIO: Sensor[20]
RADIO: Sensor[32]
RADIO: Sensor[42]
RADIO: Sensor[25]
RADIO: Sensor[17]
RADIO: Sensor[29]
RADIO: Sensor[7]
RADIO: Sensor[39]
RADIO: Sensor[11]
RADIO: Sensor[5]
RADIO: Sensor[13]
RADIO: Sensor[37]
RADIO: Sensor[45]

```

**Bottom Window Log:**

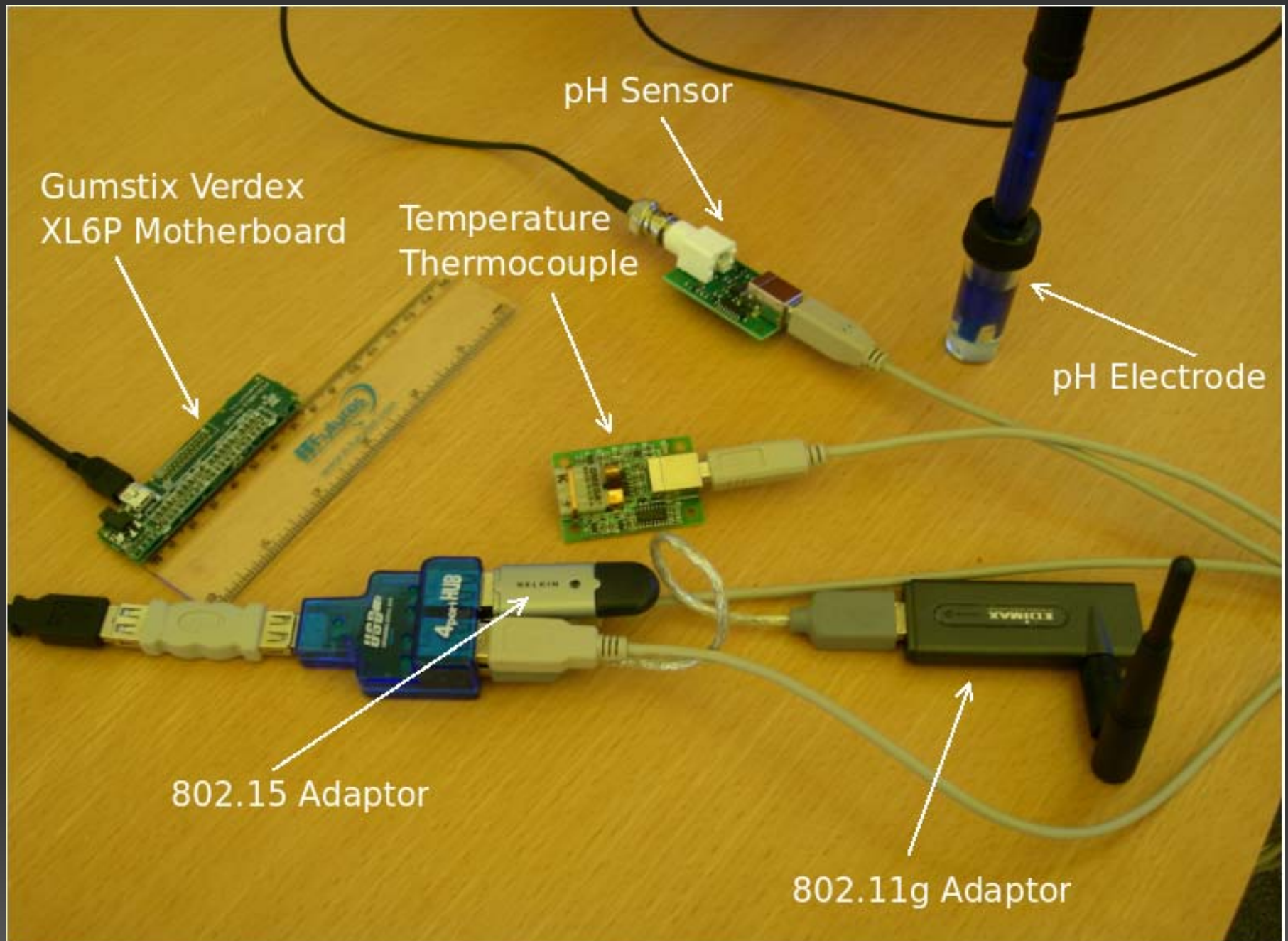
```

RADIO: Tried to send to unreachable sensorsS13
RADIO: Tried to send to unreachable sensorsS16
RADIO: Tried to send to unreachable sensorsS17
RADIO: Tried to send to unreachable sensorsS13
RADIO: Packet broadcast by S11 at time 1.660 has RSSI = 0.010
RADIO: Packet broadcast by S16 at time 1.660 has RSSI = 0.006
RADIO: Packet broadcast by S0 at time 1.640 has RSSI = 0.007
RADIO: Packet broadcast by S7 at time 1.650 has RSSI = 0.006
RADIO: Packet broadcast by S6 at time 1.650 has RSSI = 0.005
RADIO: Packet broadcast by S5 at time 1.660 has RSSI = 0.010
RADIO: Packet broadcast by S13 at time 1.660 has RSSI = 0.010
RADIO: Packet broadcast by S16 at time 1.660 has RSSI = 0.007
RADIO: Packet broadcast by S15 at time 1.670 has RSSI = 0.006
RADIO: Packet broadcast by S17 at time 1.640 has RSSI = 0.006
RADIO: Packet broadcast by S3 at time 1.640 has RSSI = 0.007
RADIO: Packet broadcast by S6 at time 1.650 has RSSI = 0.006
RADIO: Packet broadcast by S0 at time 1.640 has RSSI = 0.010
RADIO: Packet broadcast by S11 at time 1.660 has RSSI = 0.010
RADIO: Packet broadcast by S5 at time 1.660 has RSSI = 0.007
RADIO: Packet broadcast by S17 at time 1.640 has RSSI = 0.010
RADIO: Packet broadcast by S13 at time 1.660 has RSSI = 0.007
RADIO: Packet broadcast by S3 at time 1.640 has RSSI = 0.005
RADIO: Packet broadcast by S14 at time 1.660 has RSSI = 0.007
RADIO: Packet broadcast by S16 at time 1.660 has RSSI = 0.006
RADIO: Packet broadcast by S2 at time 1.640 has RSSI = 0.006

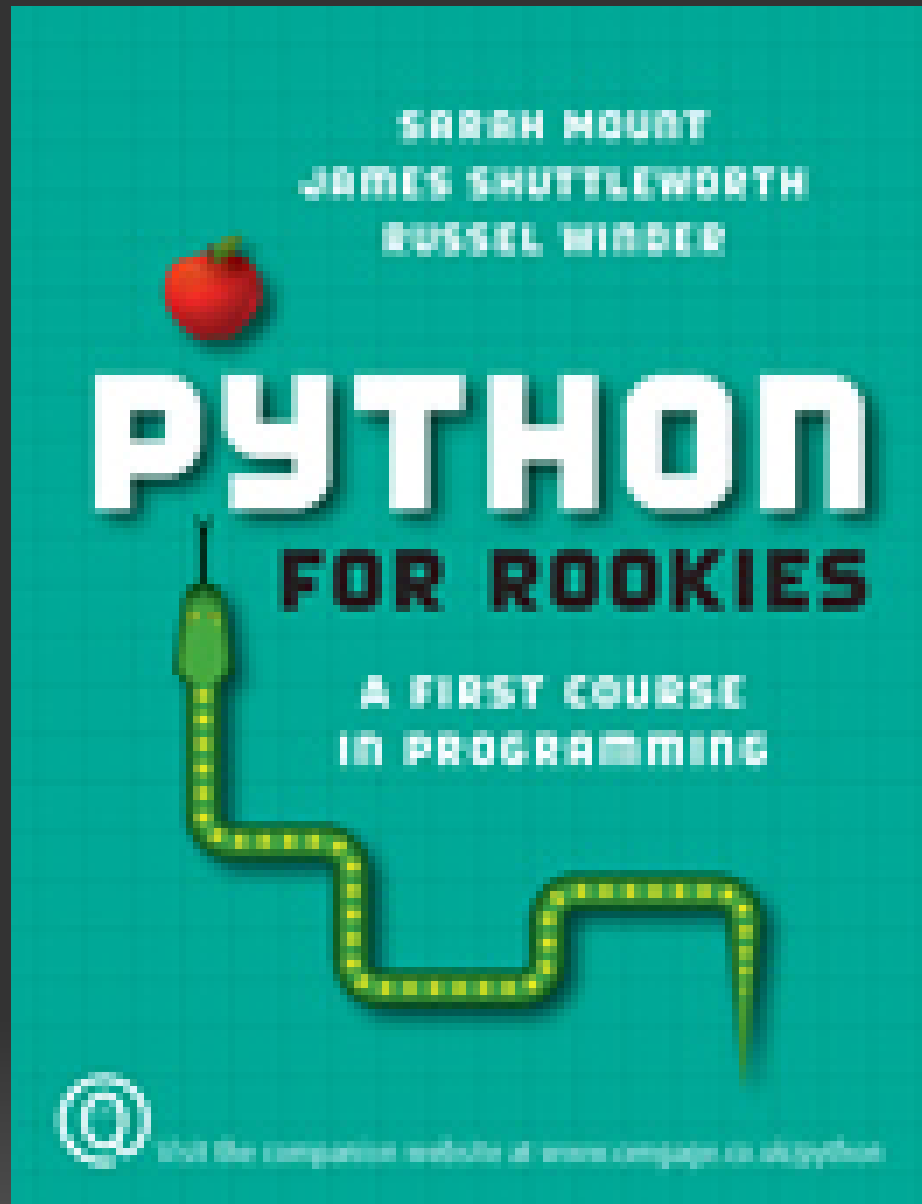
```

**Status:** 51 threads still active (top window), 20 threads still active (bottom window).

Then I made SenSor and Dan Goldsmith made SensorPlus



Laboratory hardware running Dingo



... so we wrote a book about it all ...

# Why python-csp

- Keep all the increased productivity and fun of Python
- Add scalable, mobile concurrency
- Profit.



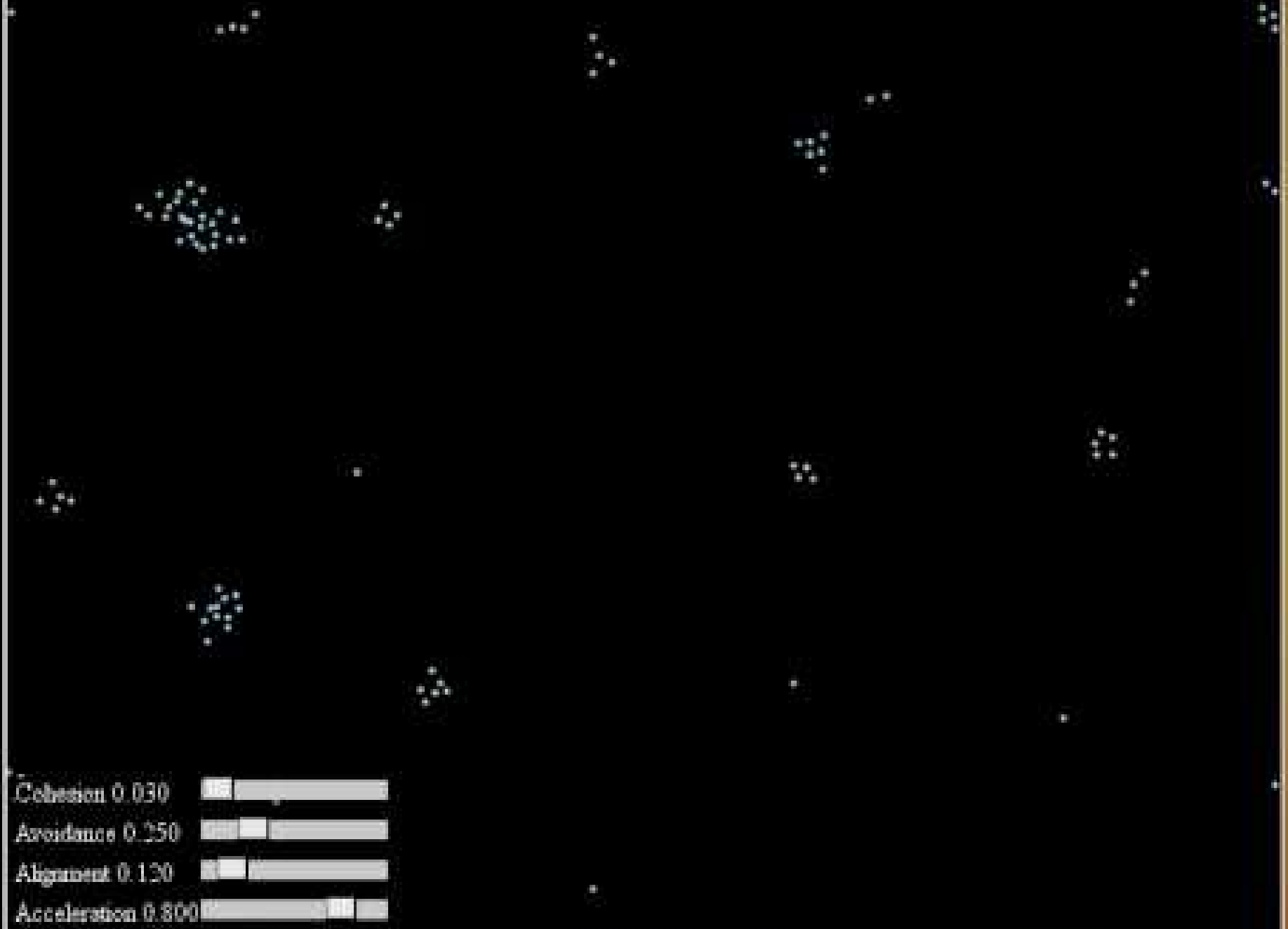
# Commstime results

	Mean (micro s)	s.d.
JCSP (Java threads)	23.8	4.29
PyCSP (Processes)	394.97	75.82
PyCSP (Threads)	292.2	47.21
PyCSP (Greenlets)	24.41	0.36
python-csp (Processes)	116.75	35.53
python-csp (Threads)	225.77	17.51
jython-csp (Java threads)	157.8	30.78

# Python oddities

- The Beazley effect
  - A multi-threaded algorithm can be slower than a single-threaded algorithm
  - GIL preempts every  $\$X$  OPCODES
- The state of Python's low-level threading libraries
  - Implement POSIX threads
  - Locking facilities (condition variables, locks, mutexes, semaphores) usually implemented in natively in Python, not provided by the OS

python-csp example: Boids



Cohesion 0.030

Avoidance 0.250

Alignment 0.130

Acceleration 0.800

# Morals of this story...

- Not every language has nice, high-level concurrency features
- It is still worth porting CSP etc. to your favourite language
  - If you don't like Python, try Actionscript ;-)
- The JVM is not the answer to every ill
- Sometimes waiting is a good idea ...
  - Google will finish Unladen Swallow
  - Jython will get faster (but will it get jythonc back?!)

# Future directions

- Mobility
- Performance issues
  - Can we do better?
    - Coroutines, protothreads, ...
    - Unladen Swallow (LLVM -> ???)
- Using the underlying thread / process libraries
  - Brings an overhead
  - Doesn't directly implement POSIX anything
  - May prove useful to replace
- Pythonic issues
  - Get high level concurrency into the standard library ;-)
- Pervasive computing -- bigraphs?