PyCSP

The beginning of a CSP library for Python

Why PyCSP

- Internal research projects
  - Simple prototyping, especially in projects that already use Python
  - Want to use CSP from Python
- eScience
  - Python
    - Script and integration language
    - Prototyping
    - Easy to learn, readable code
    - Plenty of tools and libraries
  - CSP
    - Simpler than message passing and shared memory?
- Teaching
  - eScience
  - CS students
    - Show them CSP *and* the implementation in a few lectures
    - and let them tinker with it

Some goals

- Simple, short, and readable source code
  - Should be easy to walk students through the code
- Pure python code
  - Portable implementation that does not depend on compiling extra libraries
- Reasonable performance

Components

- Channels
  - Including the One2Any Any2One and Any2Any versions
- Parallel statements
  - And Sequential
- Alternative
- Channel Poisoning

Channels

- One2OneChannel(name)
- Any2OneChannel(name)
- One2AnyChannel(name)
- Any2AnyChannel(name)

Process

- Process(function, parameter-1,…)

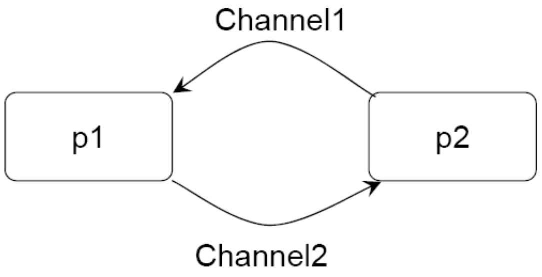
Alternative

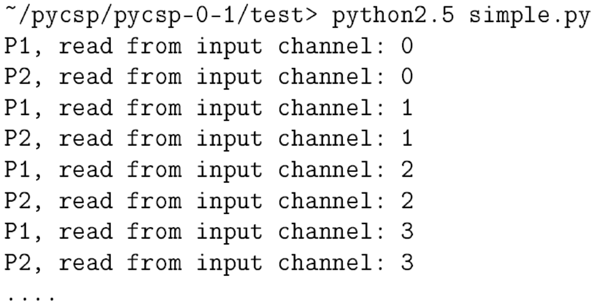
- Alt = Alternative(guard1,…)
- Ret = alt.select()

Parallel

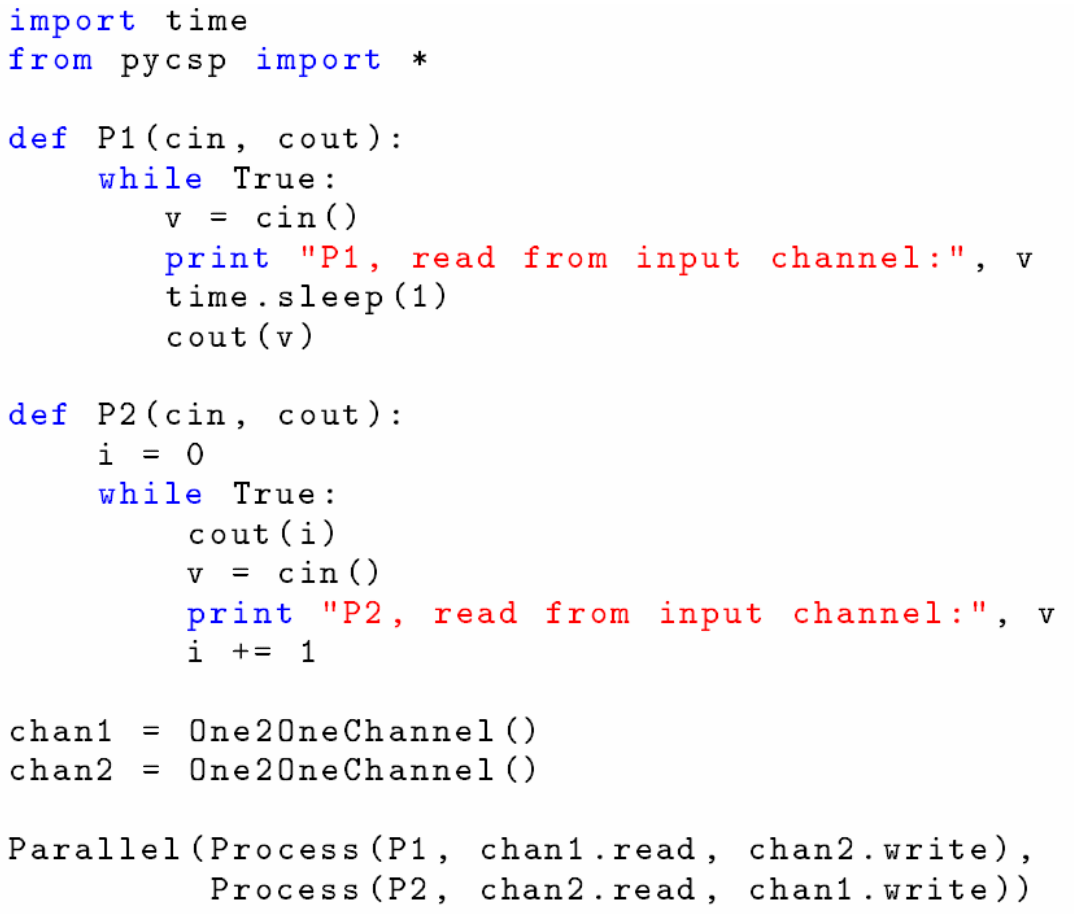
- Parallel(Process1,…)

Simple PyCSP program



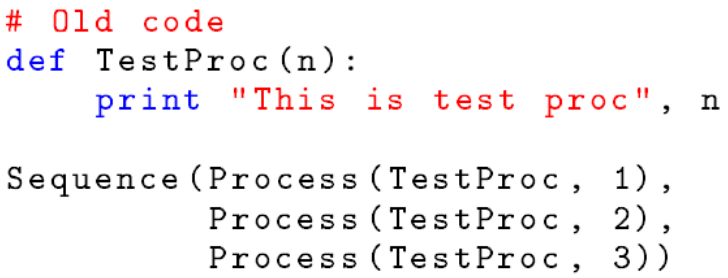


```
~/pycsp/pycsp-0-1/test> python2.5 simple.py
P1, read from input channel: 0
P2, read from input channel: 0
P1, read from input channel: 1
P2, read from input channel: 1
P1, read from input channel: 2
P2, read from input channel: 2
P1, read from input channel: 3
P2, read from input channel: 3
....
```

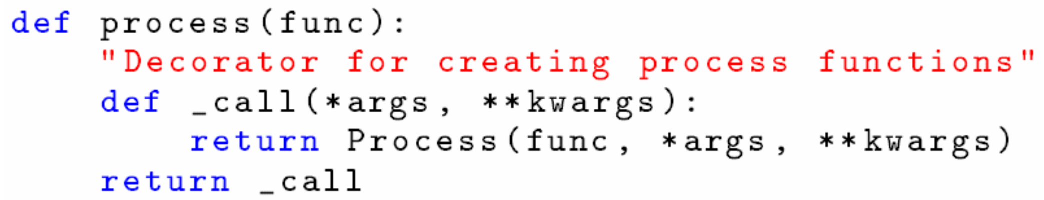


```
import time
from pycsp import *

def P1(cin, cout):
    while True:
        v = cin()
        print "P1, read from input channel:", v
        time.sleep(1)
        cout(v)

def P2(cin, cout):
    i = 0
    while True:
        cout(i)
        v = cin()
        print "P2, read from input channel:", v
        i += 1

chan1 = One2OneChannel()
chan2 = One2OneChannel()

Parallel(Process(P1, chan1.read, chan2.write),
         Process(P2, chan2.read, chan1.write))
```

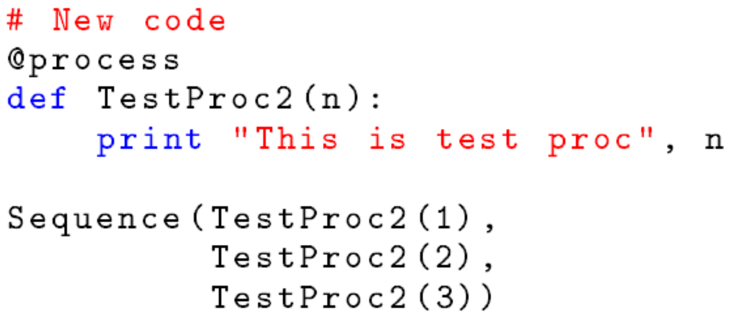
Simplifying Process Syntax using Python Descriptors



```
# Old code
def TestProc(n):
    print "This is test proc", n

Sequence(Process(TestProc, 1),
         Process(TestProc, 2),
         Process(TestProc, 3))
```



```
def process(func):
    "Decorator for creating process functions"
    def _call(*args, **kwargs):
        return Process(func, *args, **kwargs)
    return _call
```



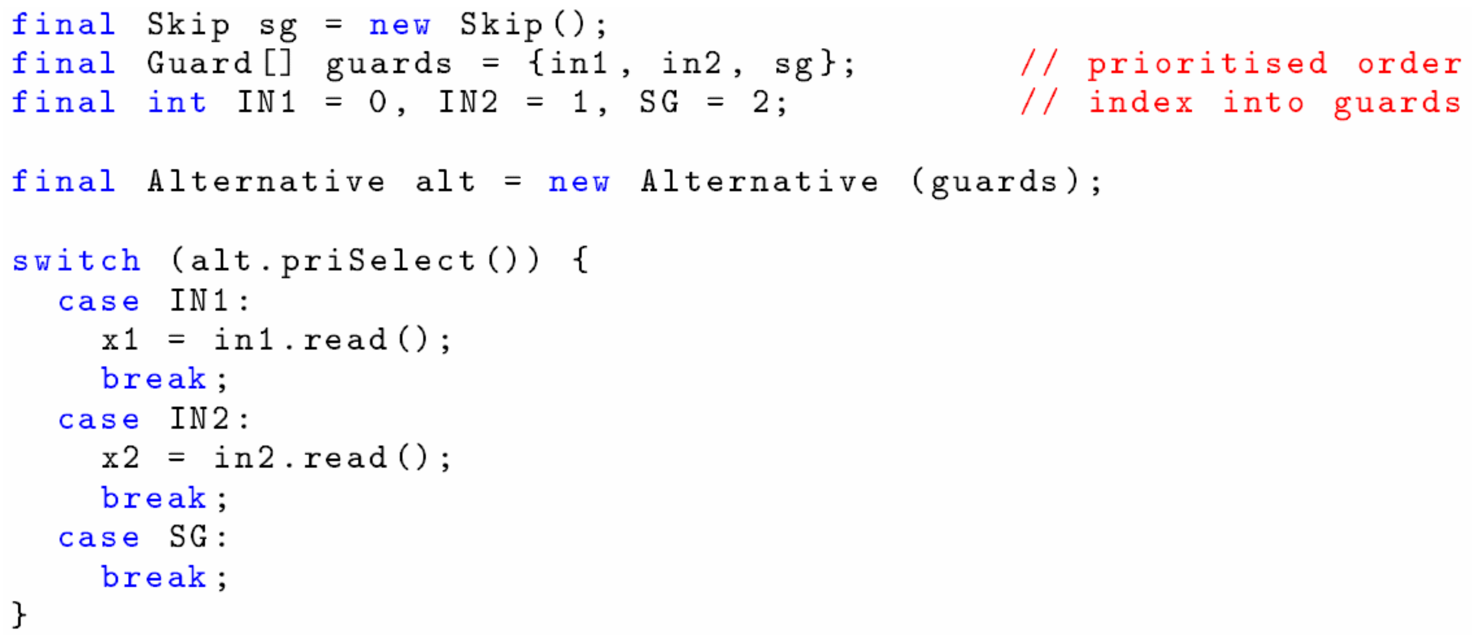
```
# New code
@process
def TestProc2(n):
    print "This is test proc", n

Sequence(TestProc2(1),
         TestProc2(2),
         TestProc2(3))
```

-Tags the function as a PyCSP process

Parallel and Sequence

```python
class Parallel:
    def __init__(self, *processes):
        self.procs = processes
        # run, then sync with them.
        for p in self.procs:
            p.start()
        for p in self.procs:
            p.join()

class Sequence:
    def __init__(self, *processes):
        self.procs = processes
        for p in self.procs:
            p.run()
```

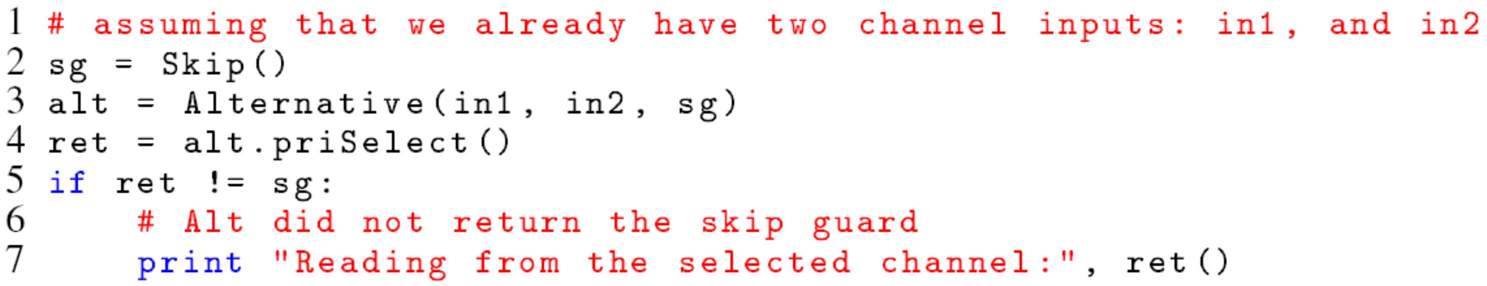
Alternative Example



```
final Skip sg = new Skip();
final Guard[] guards = {in1, in2, sg};        // prioritised order
final int IN1 = 0, IN2 = 1, SG = 2;           // index into guards

final Alternative alt = new Alternative (guards);

switch (alt.priSelect()) {
  case IN1:
    x1 = in1.read();
    break;
  case IN2:
    x2 = in2.read();
    break;
  case SG:
    break;
}
```

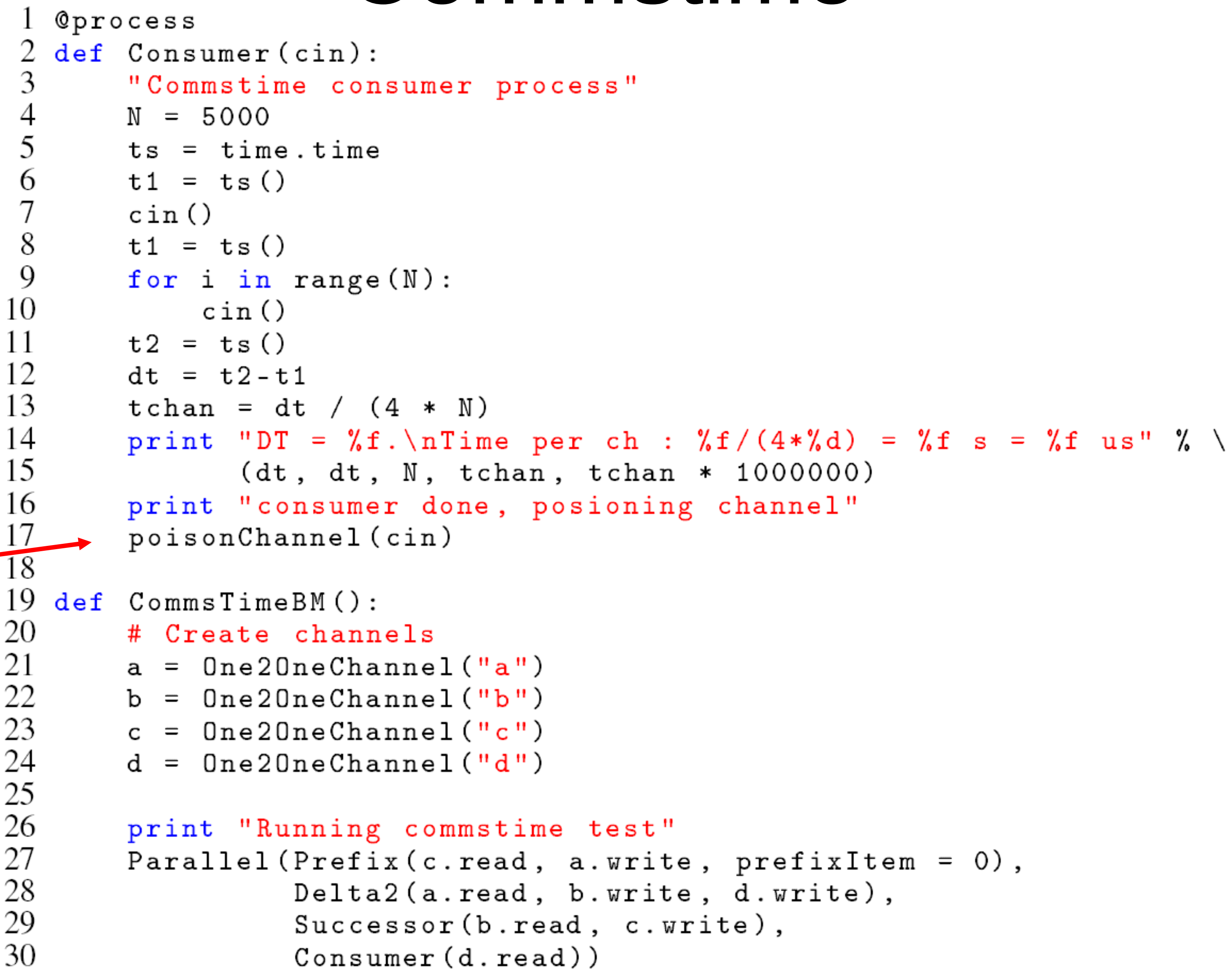
JCSP

PyCSP – returns the guard, not the guard index



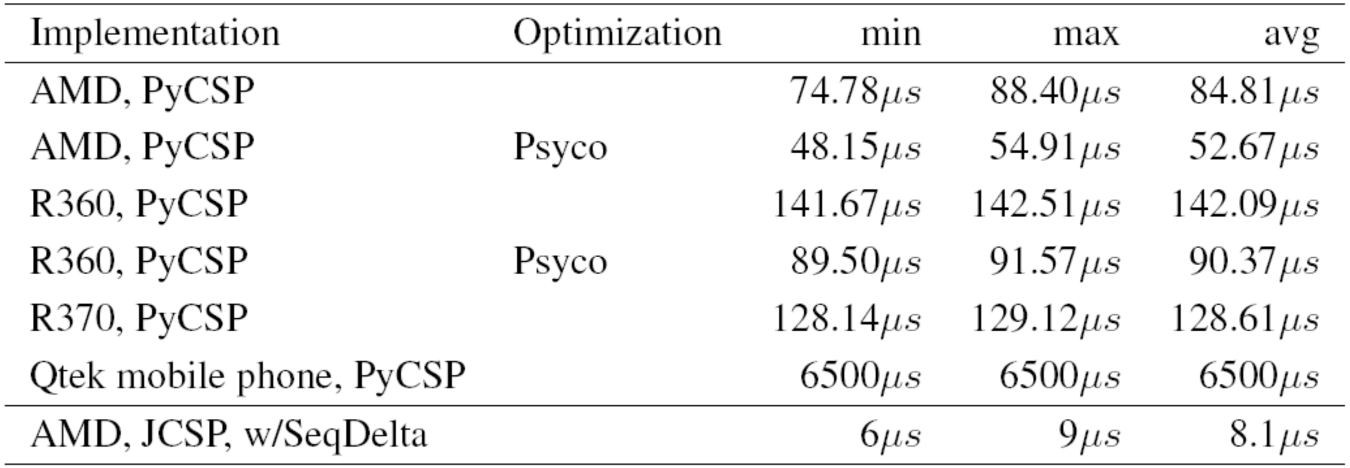
```
# assuming that we already have two channel inputs: in1, and in2
sg = Skip()
alt = Alternative(in1, in2, sg)
ret = alt.priSelect()
if ret != sg:
    # Alt did not return the skip guard
    print "Reading from the selected channel:", ret()
```

Commstime



```
@process
def Consumer(cin):
    "Commstime consumer process"
    N = 5000
    ts = time.time
    t1 = ts()
    cin()
    t1 = ts()
    for i in range(N):
        cin()
    t2 = ts()
    dt = t2-t1
    tchan = dt / (4 * N)
    print "DT = %f.\nTime per ch : %f/(4*%d) = %f s = %f us" % \
          (dt, dt, N, tchan, tchan * 1000000)
    print "consumer done, posioning channel"
    poisonChannel(cin)

def CommsTimeBM():
    # Create channels
    a = One2OneChannel("a")
    b = One2OneChannel("b")
    c = One2OneChannel("c")
    d = One2OneChannel("d")

    print "Running commstime test"
    Parallel(Prefix(c.read, a.write, prefixItem = 0),
             Delta2(a.read, b.write, d.write),
             Successor(b.read, c.write),
             Consumer(d.read))
```

Performance evaluation



Implementation	Optimization	min	max	avg
AMD, PyCSP		$74.78\mu s$	$88.40\mu s$	$84.81\mu s$
AMD, PyCSP	Psyco	$48.15\mu s$	$54.91\mu s$	$52.67\mu s$
R360, PyCSP		$141.67\mu s$	$142.51\mu s$	$142.09\mu s$
R360, PyCSP	Psyco	$89.50\mu s$	$91.57\mu s$	$90.37\mu s$
R370, PyCSP		$128.14\mu s$	$129.12\mu s$	$128.61\mu s$
Qtek mobile phone, PyCSP		$6500\mu s$	$6500\mu s$	$6500\mu s$
AMD, JCSP, w/SeqDelta		$6\mu s$	$9\mu s$	$8.1\mu s$

Table 1 Commstime results

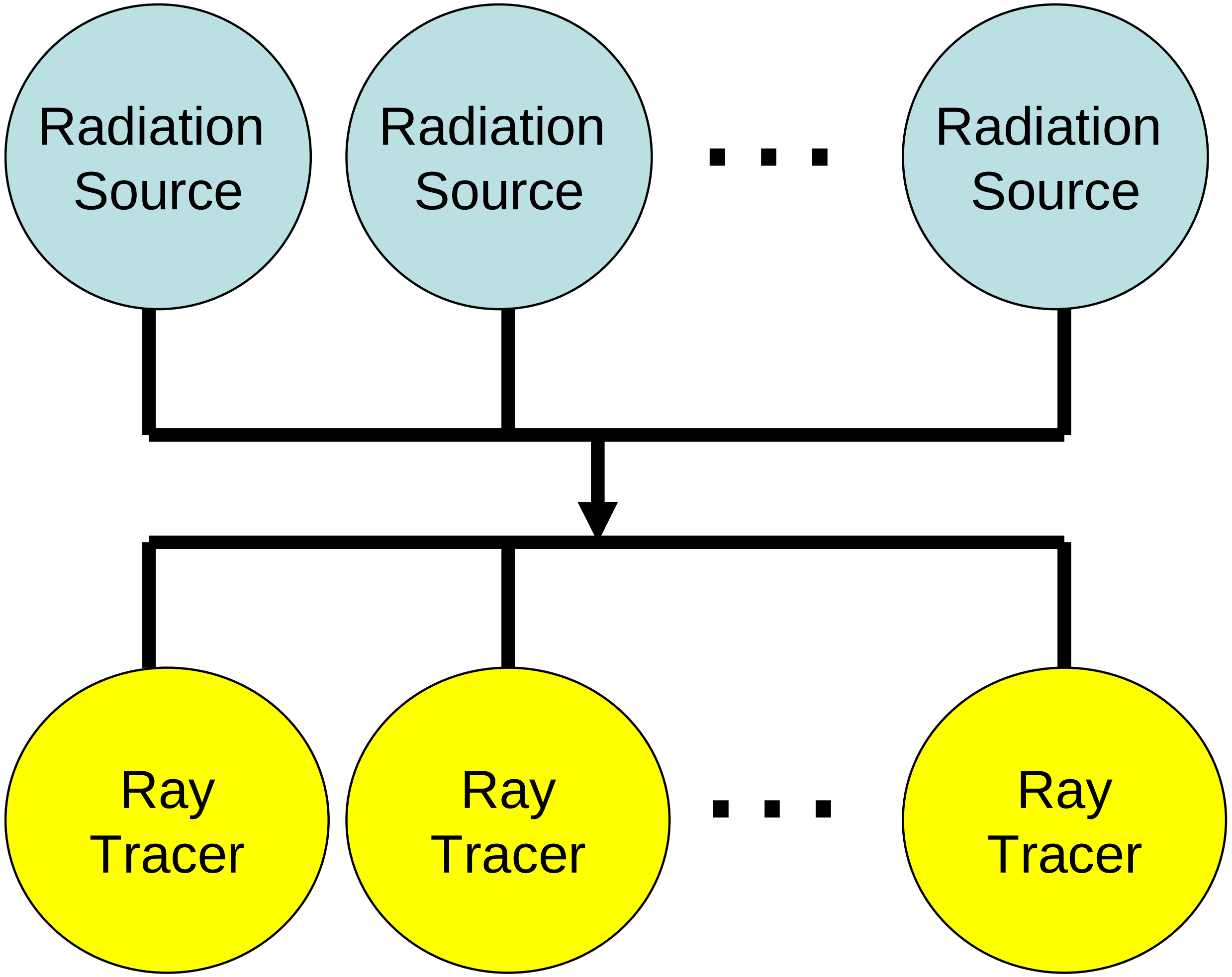
Example

- Radiation therapy planning
- Multiple radiation sources
- Embarrassingly parallel execution

Parallelism

- Have each source as a process in its own right
- Have each source as process that produces output for ray-tracing processes
  - This has more parallelims

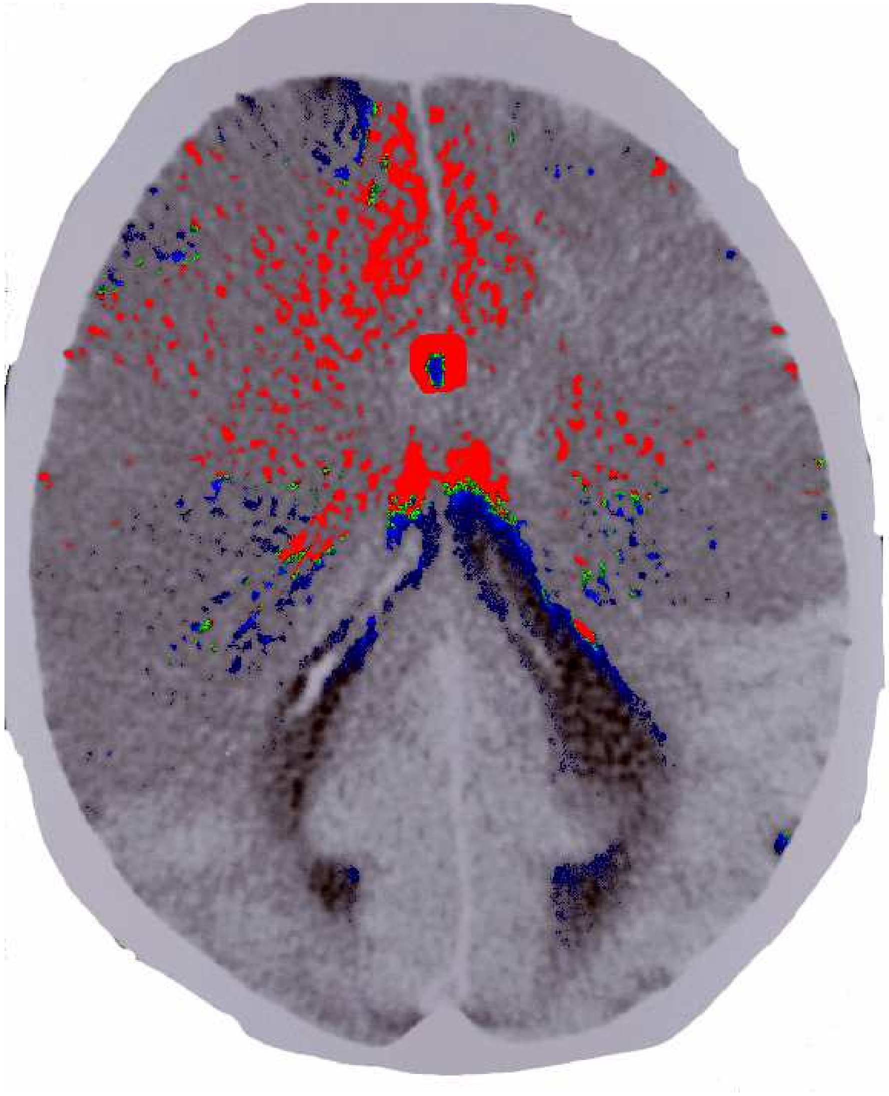
Architecture



Code

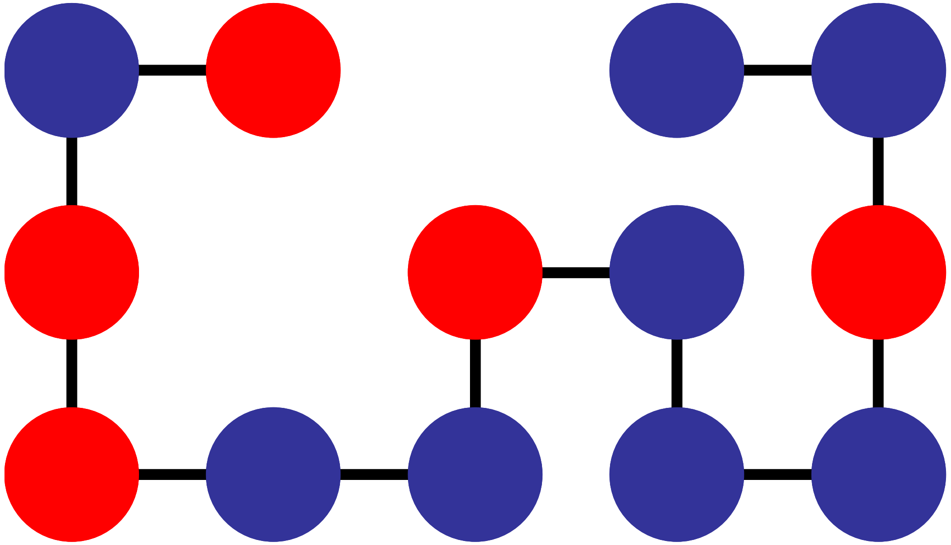
```
c = Any2AnyChannel()
ec = Any2OneChannel()

Parallel(Process(kanon,  (85.0,75.0),  (1.0,0.8),  50000, c.write, ec.write),
        Process(kanon, (10.0,230.0),  (1.0,0.0),  50000, c.write, ec.write),
        Process(kanon,(550.0,230.0), (-1.0,0.0),  50000, c.write, ec.write),
        Process(kanon, (475.0,90.0), (-1.0,.75),  50000, c.write, ec.write),
        Process(kanon,  (280.0,0.0),  (0.0,1.0),  50000, c.write, ec.write),
        Process(barrier, 5, ec.read, c.write),
        Process(trace_particles, c.read),
        Process(trace_particles, c.read),
        Process(trace_particles, c.read),
        Process(trace_particles, c.read),
        Process(trace_particles, c.read),
        Process(trace_particles, c.read))
```

Result

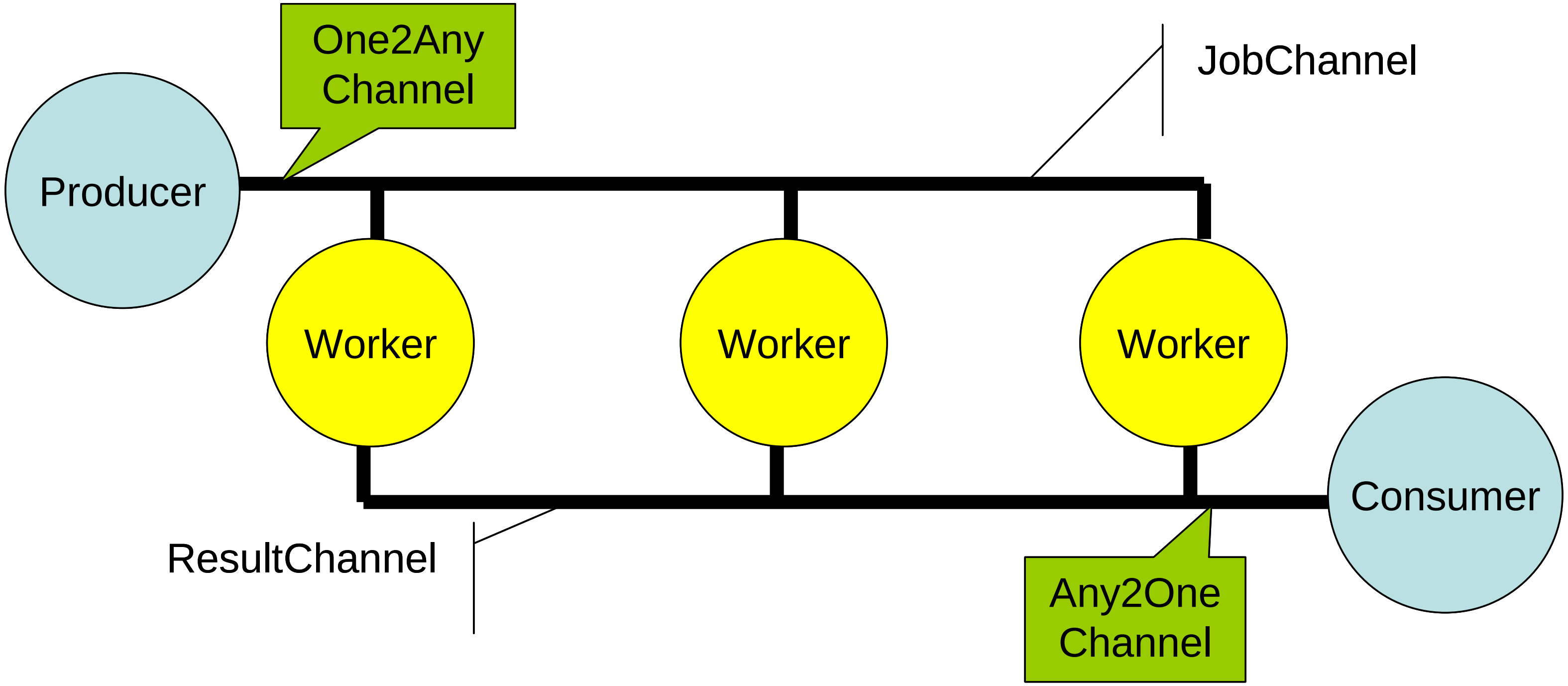


Protein Folding

- Not folding real proteins
- But rather prototeins
  - They are simpler to understand and code but complexity of the calculations is the same



Design


One2Any Channel
JobChannel
Producer
Worker
Worker
Worker
Consumer
ResultChannel
Any2One Channel

Master Code

```
feeder = One2AnyChannel()
collector = Any2OneChannel()
done = Any2OneChannel()

Parallel( Process(producer, protein, map, place, feeder.write),
          Process(worker, feeder.read, collector.write, done.write),
          Process(worker, feeder.read, collector.write, done.write),
          Process(worker, feeder.read, collector.write, done.write),
          Process(worker, feeder.read, collector.write, done.write),
          Process(worker, feeder.read, collector.write, done.write),
          Process(barrier, 5, done.read, collector.write),
          Process(sink, collector.read) )
```

Producer

```
def producer(protein, map, place, cout):
    res=sfold(protein, map, place, cout)
    for f in res:
      cout((protein, f.map, f.place))
    poisonChannel(cout)
```

Worker

```
def worker(cin, cout, done):
    run = True
    while run:
        try:
            protein, map, pplace = cin()
            res = fold(protein, map, place)
            cout(res)
        except:
            run = False
    done('done')
```

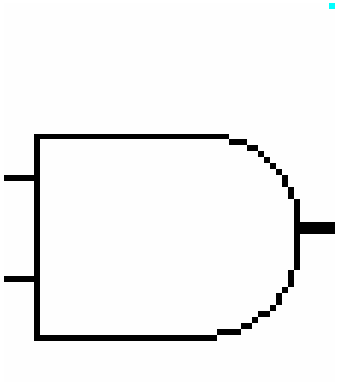
Terminator

```
def barrier(n, cin, cout):
    for i in range(n): cin()
    print 'Now the pill'
    poisonChannel(cout)
```

Digital circuit simulator

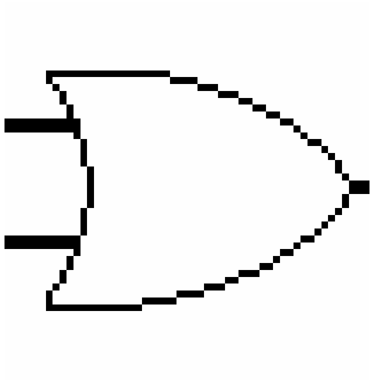
- An example of utilizing CSP for modeling functions rather than concurrency
- We build a library of basic digital circuitry and use them to build more complex circuits
  - Which may then be used for even more circuitry etc..

AND



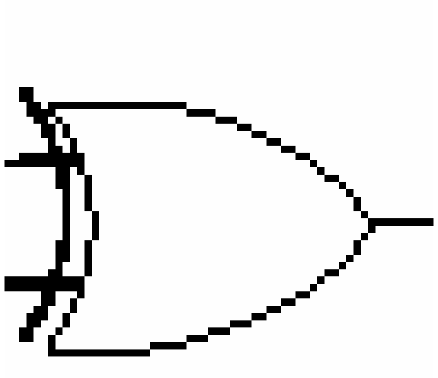
```
def AND(cin1, cin2, cout):
    x1=x2=0
    alt = Alternative([cin1, cin2])
    while True:
        cout(x1 and x2)
        ret = alt.select()
        if ret == cin1:
            x1 = ret()
        else:
            x2 = ret()
```

OR



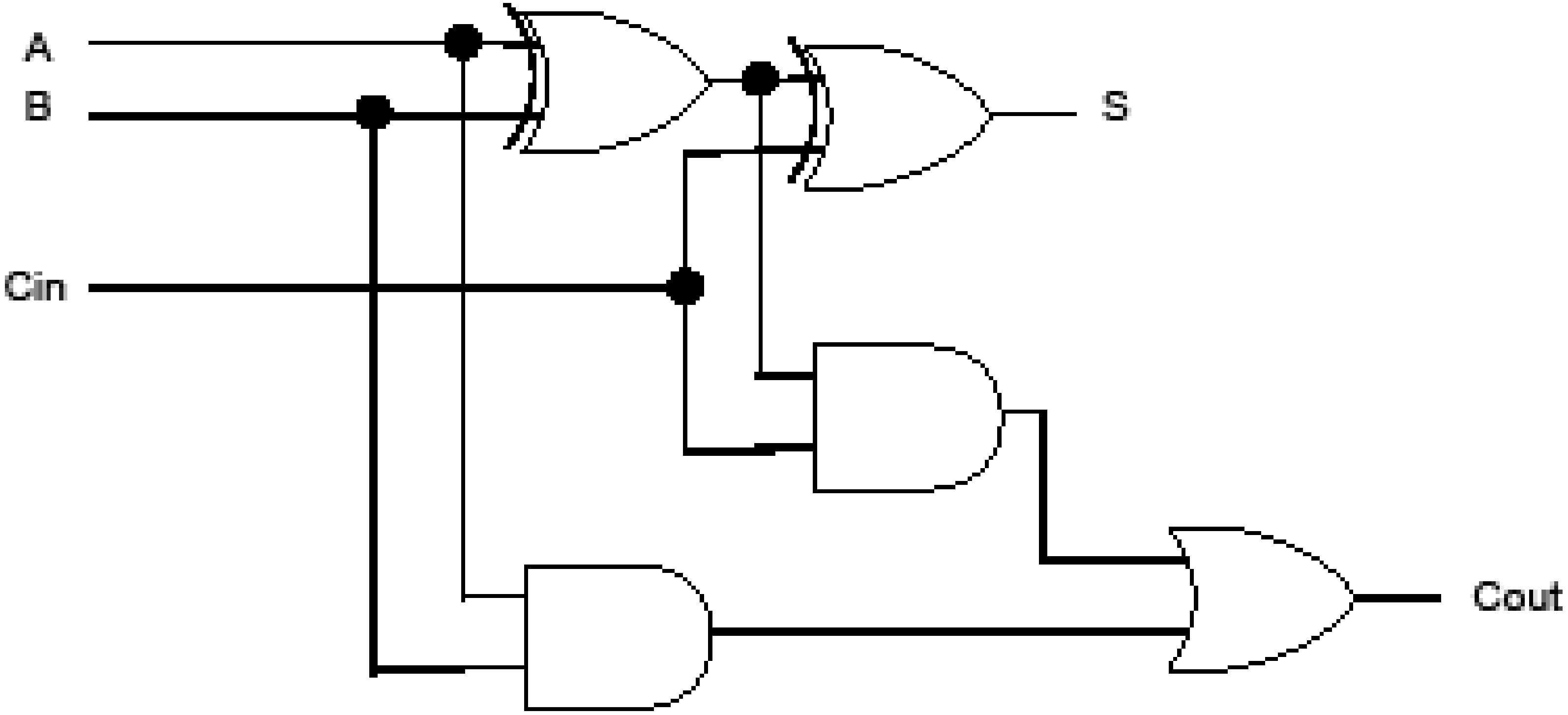
```
def OR(cin1, cin2, cout):
    x1=x2=0
    alt = Alternative([cin1, cin2])
    while True:
        cout(x1 or x2)
        ret = alt.select()
        if ret == cin1:
            x1 = ret()
        else:
            x2 = ret()
```

XOR



```
def XOR(cin1, cin2, cout):
    x1=x2=0
    alt = Alternative([cin1, cin2])
    while True:
        cout((x1 or x2) and not (x1 and x2))
        ret = alt.select()
        if ret == cin1:
            x1 = ret()
        else:
            x2 = ret()
```

Full-adder



Full-adder

```
def Adder(A, B, Cin, S, Cout):
    Aa = One2OneChannel("Aa")
    Ab = One2OneChannel("Ab")
    Ba = One2OneChannel("Ba")
    Bb = One2OneChannel("Bb")
    Ca = One2OneChannel("Ca")
    Cb = One2OneChannel("Cb")
    i1 = One2OneChannel("i1")
    i1a = One2OneChannel("i1a")
    i1b = One2OneChannel("i1b")
    i2 = One2OneChannel("i2")
    i3 = One2OneChannel("i3")
```

```
Parallel(Process(delta, A.read, Aa.write, Ab.write),
         Process(delta, B.read, Ba.write, Bb.write),
         Process(delta, Cin.read, Ca.write, Cb.write),
         Process(delta, i1.read, i1a.write, i1b.write),
         Process(XOR, Aa.read, Ba.read, i1.write),
         Process(XOR, i1a.read, Ca.read, S.write),
         Process(AND, Ab.read, Bb.read, i2.write),
         Process(AND, i1b.read, Cb.read, i3.write),
         Process(OR, i2.read, i3.read, Cout.write))
```

Download

- Get PyCSP at
  - www.cs.uit.no/~johnm/code/PyCSP/