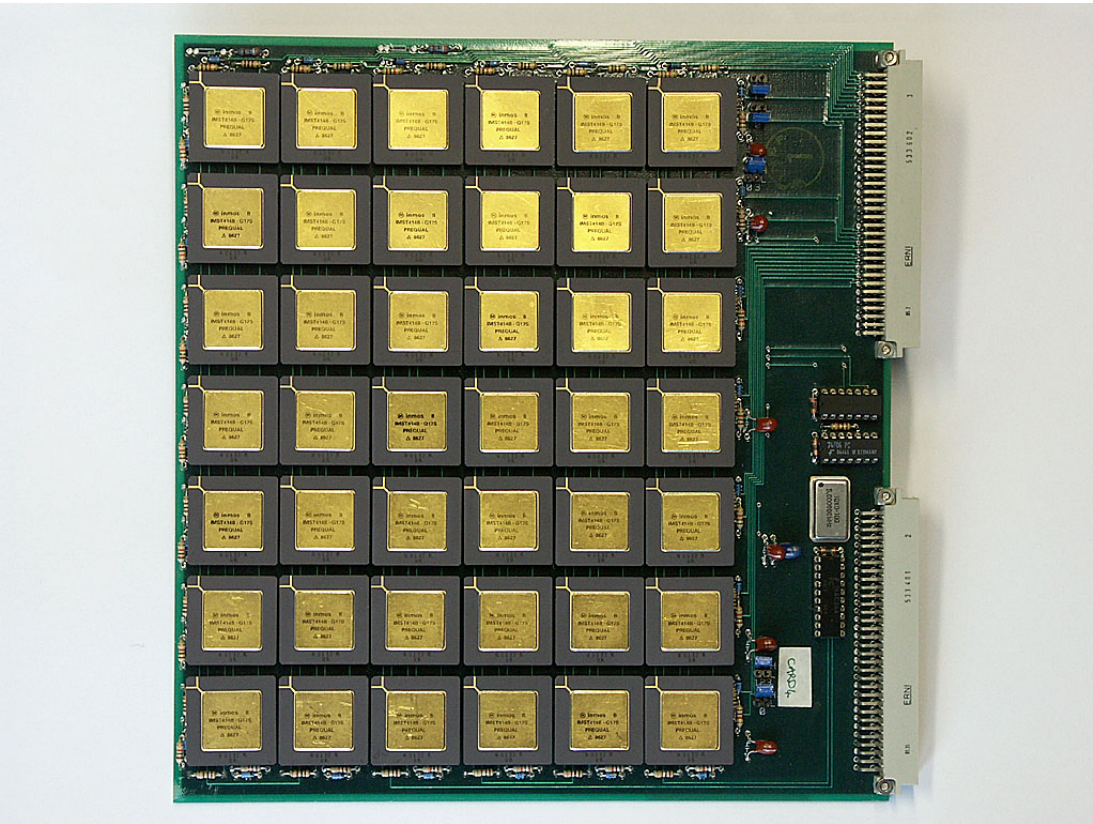
python-csp: bringing OCCAM to Python

Sarah Mount

Europython 2010

This talk...

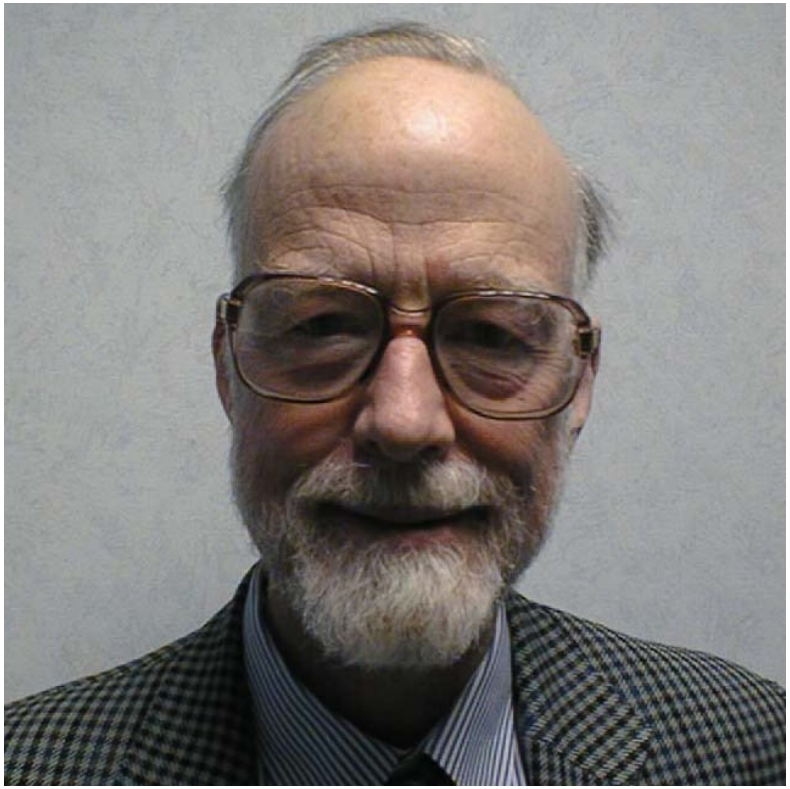
is all about the python-csp project. There is a similar project called PyCSP, these will merge over the next few months. Current code base is here: http://code.google.com/p/python-csp
Please do contribute bug fixes / issues / code / docs / rants / ...



INMOS B0042 Board © David May

The Transputer was the original "multicore" machine and was built to run the OCCAM language – a realisation of CSP.

`http://www.cs.bris.ac.uk/~dave/transputer.html`



Tony Hoare

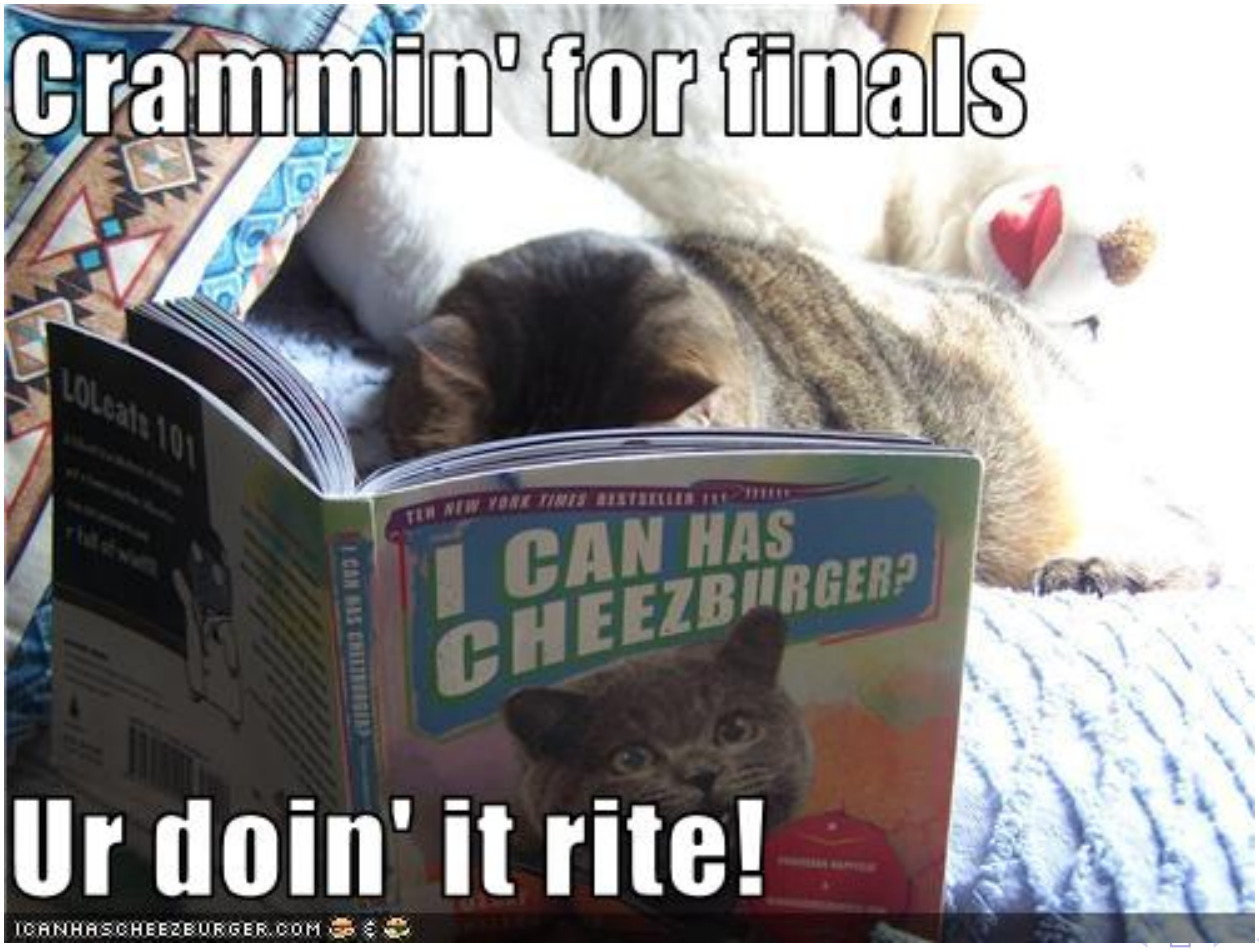
Invented CSP, OCCAM, Quicksort and other baddass stuff. You may remember him from such talks as his Europython 2009 Keynote.

OCCAM

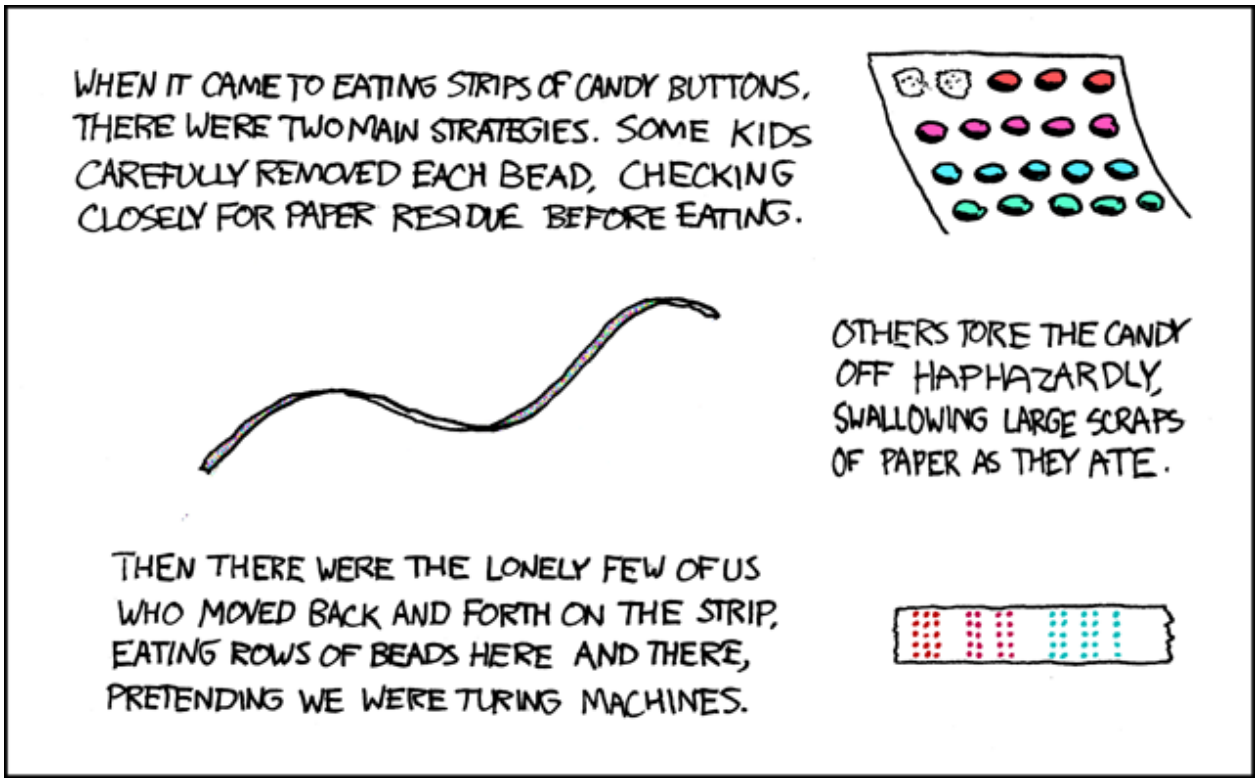
OCCAM lives on in it's current implementation as OCCAM-π
`http://www.cs.kent.ac.uk/projects/ofa/kroc/`
It can run on Lego bricks, Arduino boards and other things. Like Python it uses indentation to demark scope. Unlike Python OCCAM is MOSTLY IN UPPERCASE.

CSP and message passing

This next bit will be revision for many of you!



Most of us think of programs like this



http://xkcd.com/205/

This is the old "standard" model!

- Multicore will soon take over the world (if it hasn't already).
- Shared memory concurrency / parallelism is hard:
  - Race hazards
  - Deadlock
  - Livelock
  - Tracking which lock goes with which data and what order locks should be used.
  - Operating Systems are old news!

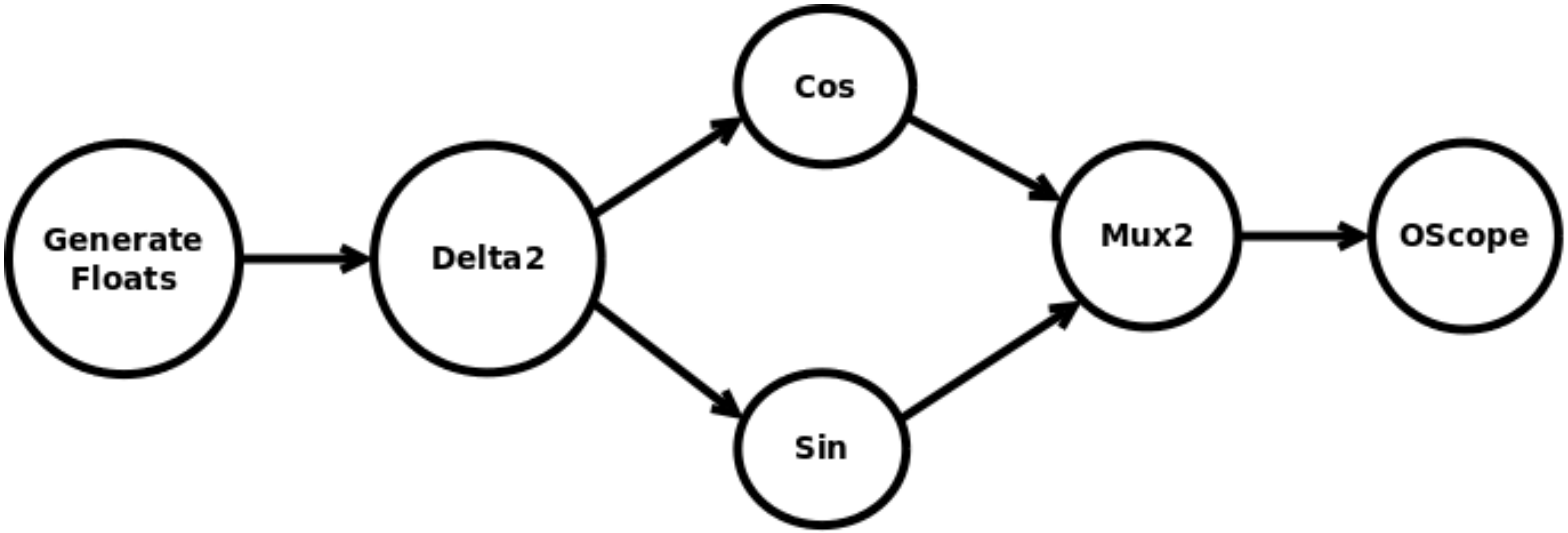
Message passing

Definition

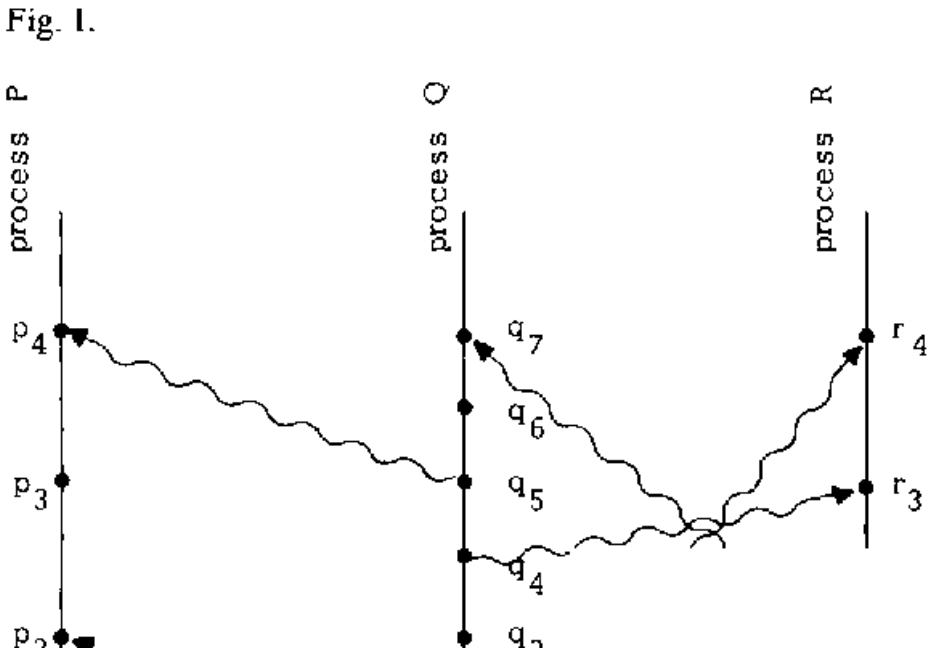
In *message passing* concurrency (or parallelism) "everything" is a process and no data is shared between processes. Instead, data is "passed" between *channels* which connect processes together. Think: OS processes and UNIX pipes.

- The *actor* model (similar to Kamaelia) uses asynchronous channels. Processes write data to a channel and continue execution.
- CSP (and π-calculus) use synchronous channels, or *rendezvous*, where a process writing to a channel has to wait until the value has been read by another process before proceeding.

Process diagram



Why synchronous channels?



Leslie Lamport (1978). Time, clocks, and the ordering of events in a distributed system". Communications of the ACM 21 (7)

Hello World!

```
1 from csp.cspprocess import *
2
3 @process
4 def hello():
5     print 'Hello CSP world!'
6
7 hello().start()
```

Example of process creation: web server

```
1  @process
2  def server(host, port):
3      sock = socket.socket(...)
4      # Set up sockets
5      while True:
6          conn, addr = sock.accept()
7          request = conn.recv(4096).strip()
8          if request.startswith('GET'):
9              ok(request,conn).start()
10         else:
11             not_found(request,conn).start()
```

Example of process creation: web server

```
1  @process
2  def not_found(request, conn):
3      page = '<h1>File not found</h1>'
4      conn.send(response(404,
5                         'Not Found',
6                         page))
7      conn.shutdown(socket.SHUT_RDWR)
8      conn.close()
9      return
```

Combining processes: in parallel

```
1  # With a Par object:
2  Par(p1, p2, p3, ... pn).start()
3  # With syntactic sugar:
4  p1 // (p2, p3, ... pn)
5  # RHS must be a sequence type.
```

Combining processes: repeating a process sequentially

```
1 @process
2 def hello():
3     print 'Hello CSP world!'
4 if __name__ == '__main__':
5     hello() * 3
6     2 * hello()
```

Channels

```
1  @process
2  def client(chan):
3      print chan.read()
4  @process
5  def server(chan):
6      chan.write('Hello CSP world!')
7
8  if __name__ == '__main__':
9      ch = Channel()
10     Par(client(ch), server(ch)).start()
```

What you need to know about channels

- Channels are currently UNIX pipes
- This will likely change
- Channel rendezvous is "slow" compared with JCSP: $200\mu s$ *vs* $16\mu s$
- This will be fixed by having channels written in C
- Because channels are, at the OS level, open "files", there can only be a limited number of them (around 300 on my machine)
- This makes me sad :-(

Message passing an responsiveness

- This isn't about speed.
- Sometimes, you have several channels and reading from them all round-robin may reduce responsiveness if one channel read blocks for a long time.

This is BAD:

```
1 @process
2 def foo(channels):
3     for chan in channels:
4         print chan.read()
```

ALTing and choice

Definition

ALTing, or non-deterministic choice, *selects* a channel to read from from those channels which are ready to read from.

You can do this with (only) two channels:

```
1 @process
2 def foo(c1, c2):
3     print c1 | c2
4     print c1 | c2
```

ALTing proper (many channels)

```
1  @process
2  def foo(c1, c2, c3, c4, c5):
3      alt = Alt(c1, c2, c3, c4, c5)
4      # Choose random available offer
5      print alt.select()
6      # Avoid last selected channel
7      print alt.fair_select()
8      # Choose channel with lowest index
9      print alt.priselect()
```

Syntactic sugar for ALTing

```
1  @process
2  def foo(c1, c2, c3, c4, c5):
3      gen = Alt(c1, c2, c3, c4, c5)
4      print gen.next()
5      print gen.next()
6      print gen.next()
7      print gen.next()
8      print gen.next()
```

ALTing: when you really, really must return right now

```
1  @process
2  def foo(c1, c2, c3):
3      # Skip() will /always/ complete
4      # a read immediately
5      gen = Alt(c1, c2, c3, Skip())
6      print gen.next()
7      ...
```

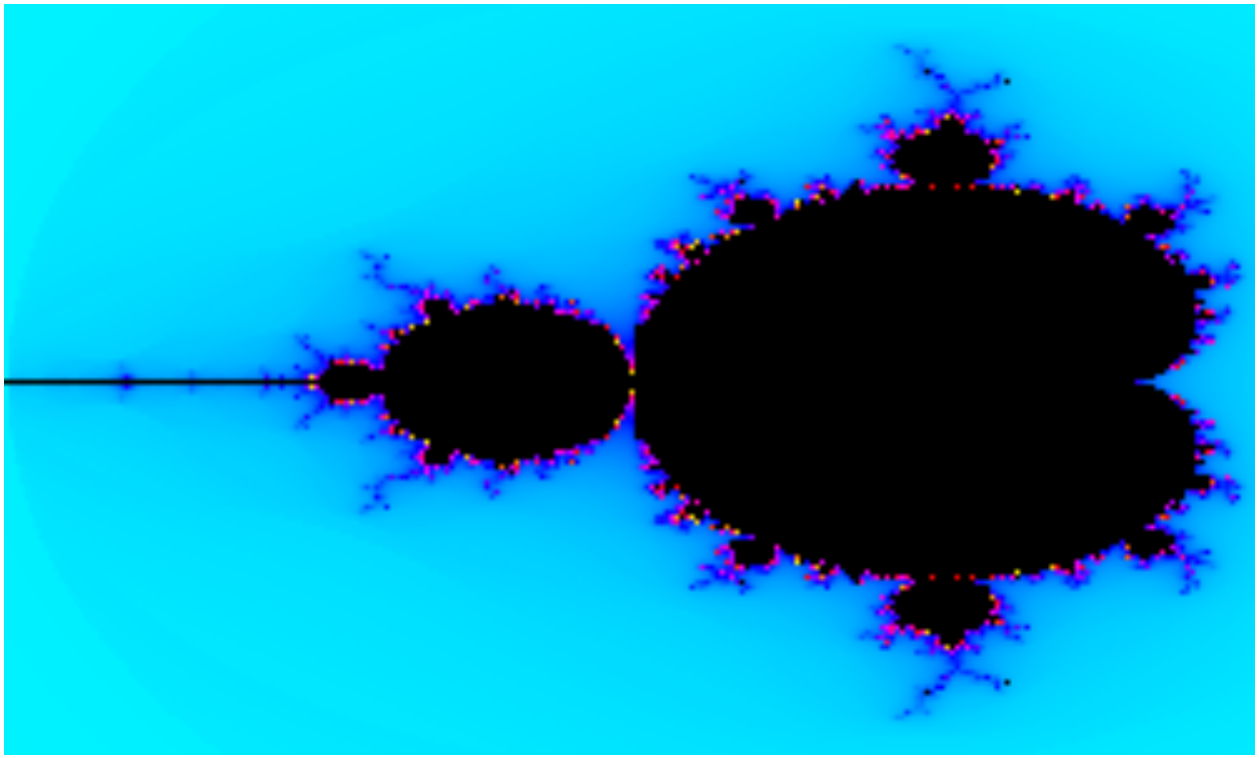
Channel poisoning

Definition

Idiomatically in this style of programming most processes contain infinite loops. *Poisoning* a channel asks all processes connected to that channel to shut down automatically. Each process which has a poisoned channel will automatically poison any other channels it is connected to. This way, a whole graph of connected processes can be terminated, avoiding race conditions.

```
1 @process
2 def foo(channel):
3     ...
4     channel.poison()
```

A bigger example: Mandelbrot generator

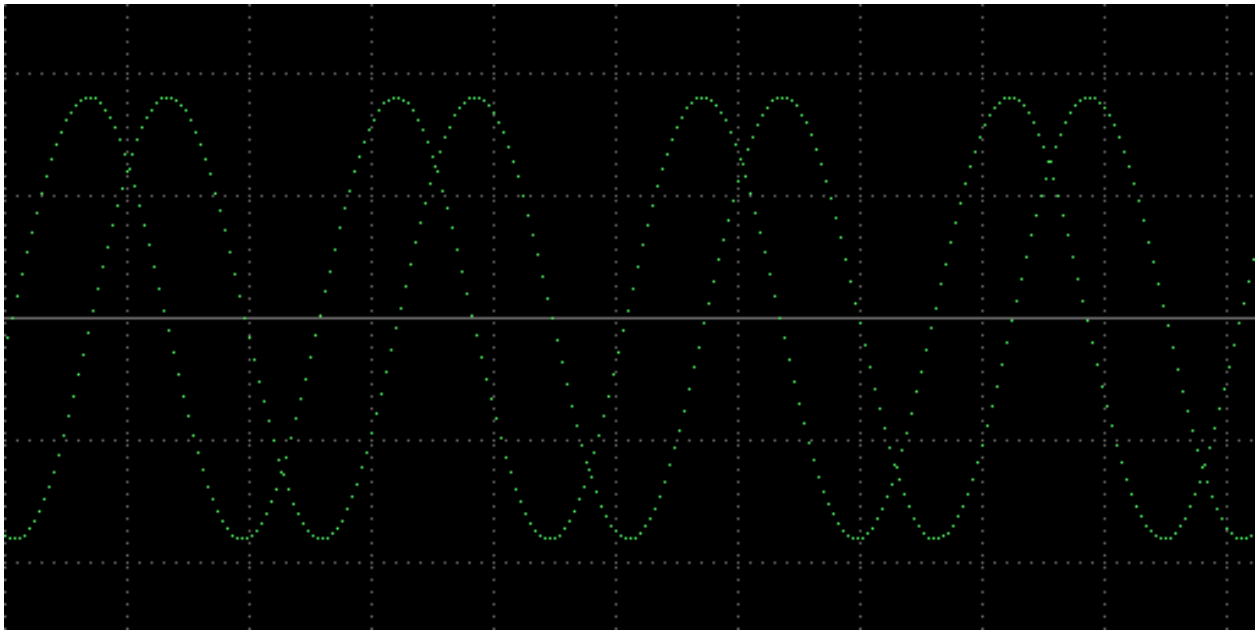


```
1  @process
2  def main(IMSIZE):
3      channels = []
4      # One producer + channel for each image colum
5      for x in range(IMSIZE[0]):
6          channels.append(Channel())
7          processes.append(mandelbrot(x, ... channe
8      processes.insert(0, consume(IMSIZE, channels)
9      mandel = Par(*processes)
10     mandel.start()
```

```
1 @process
2 def mandelbrot(xcoord, cout):
3     # Do some maths here
4     cout.write(xcoord, column_data)
```

```
1  @process
2  def consumer(cins):
3      # Initialise PyGame...
4      gen = len(cins) * Alt(*cins)
5      for i in range(len(cins)):
6          xcoord, column = gen.next()
7          # Blit that column to screen
8      for event in pygame.event.get():
9          if event.type == pygame.QUIT:
10             for channel in cins:
11                 channel.poison()
12             pygame.quit()
```

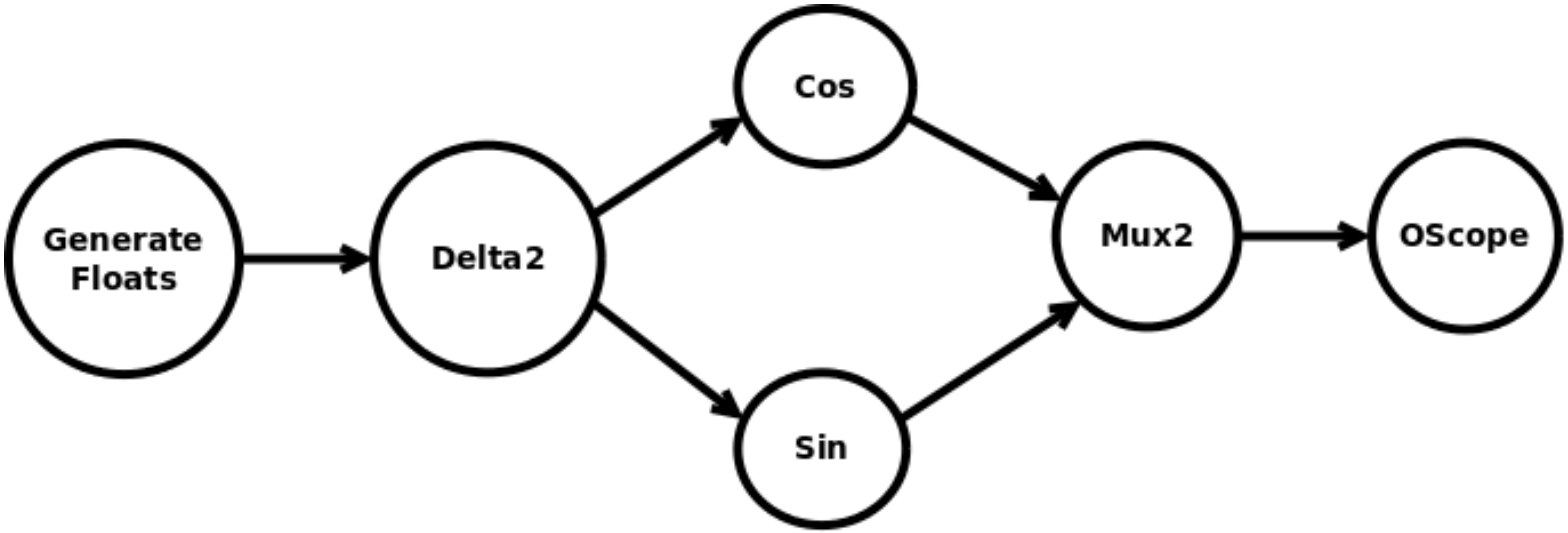
Oscilloscope



```
1  @process
2  def Oscilloscope(inchan):
3      # Initialise PyGame
4      ydata = []
5      while not QUIT:
6          ydata.append(inchan.read())
7          ydata.pop(0)
8          # Blit data to screen...
9      # Deal with events
```

```
1 from csp.builtins import Sin
2 def trace_sin():
3     chans = Channel(), Channel()
4     Par(GenerateFloats(chans[0]),
5         Sin(chans[0], chans[1]),
6         Oscilloscope(chans[1])).start()
7     return
```

Process diagram



```
1  def trace_mux():
2      chans = [Channel() for i in range(6)]
3      par = Par(GenerateFloats(chans[0]),
4                Delta2(chans[0], chans[1],
5                       chans[2]),
6                Cos(chans[1], chans[3]),
7                Sin(chans[2], chans[4]),
8                Mux2(chans[3], chans[4],
9                     chans[5]),
10               Oscilloscope(chans[5]))
11     par.start()
```

Wiring


Switchboard operator kitteh
is agravated with your wire shinanagins
ICANHASCHEEZBURGER.COM