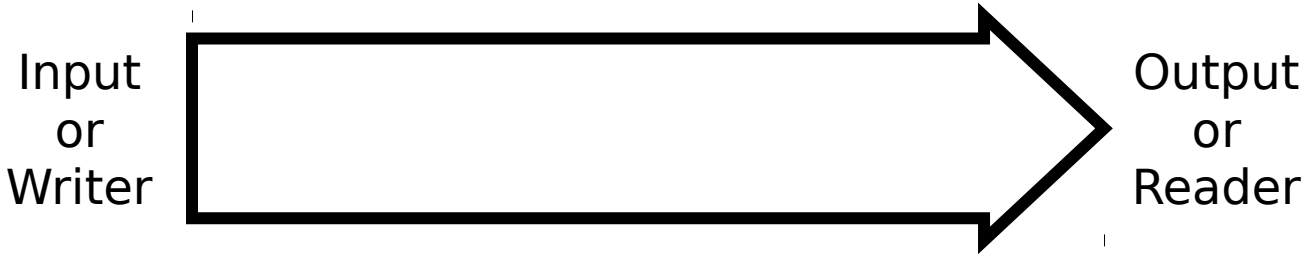
Starting PyCSP

PyCSP

- PyCSP is an implementation of CSP within Python.
- Implements the core of CSP, plus a few extras.
- https://github.com/runefriborg/pycsp.
- from pycsp.parallel import *
- Be aware that it has the strengths and weaknesses of Python.
- Runs on Python 3, except where it doesn’t.

Channels in PyCSP

- Channels have a input or writer end
- Channels have an output or reader end
- Channels are one directional



Channels in PyCSP

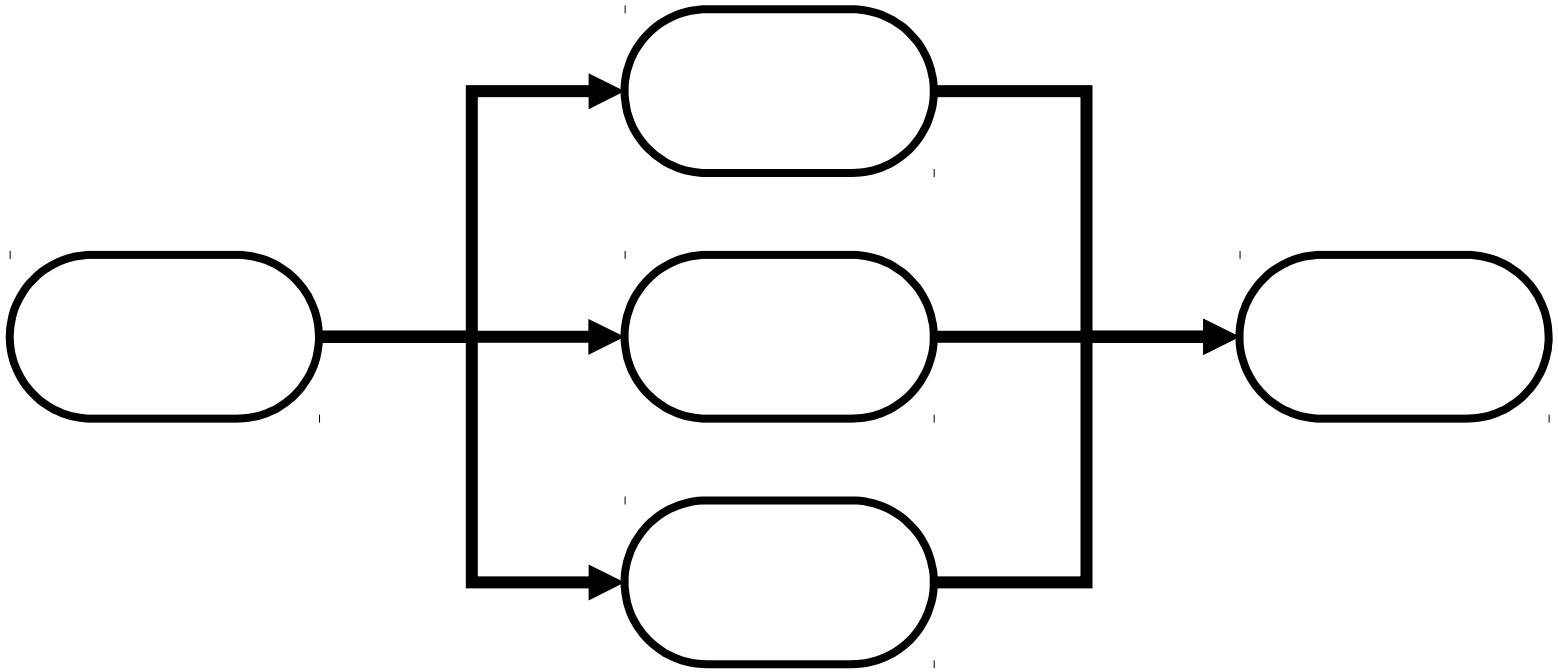
```
from pycsp.parallel import *

channel_one = Channel()
```

Channels in PyCSP

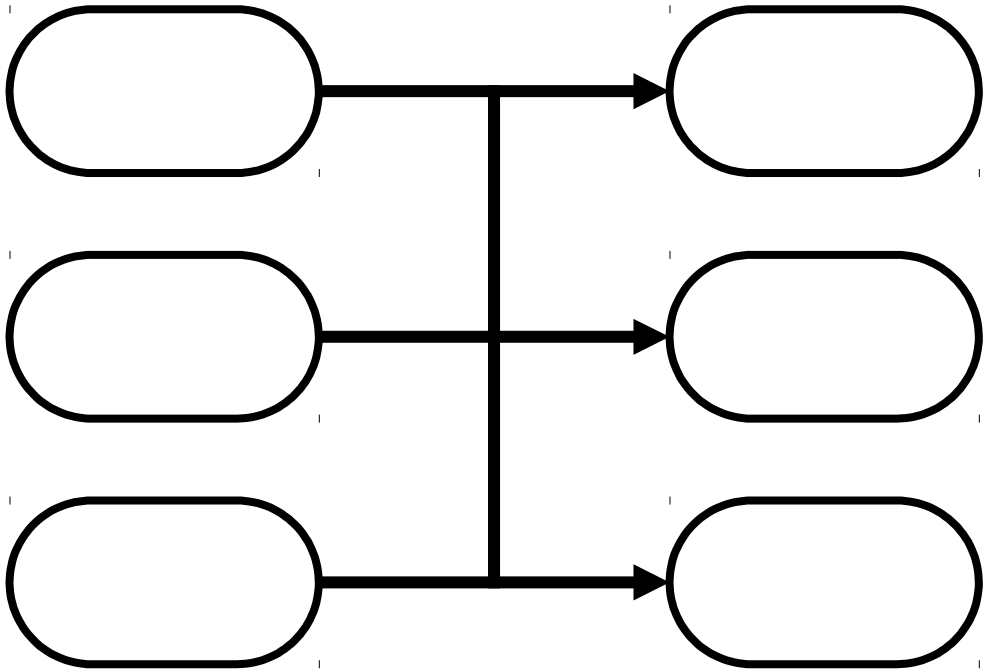
- Channels in PyCSP are Any to Any Channels
- This means that each end may be attached to multiple processes
- Messages are only sent from one process to one process however

Channels in PyCSP

Channels in PyCSP



Channels in PyCSP



Processes in PyCSP

- Process code is executed sequentially.
- Processes will only be scheduled if they can progress.
- Processes only continue for as long as you tell them. Continuous ones will need an infinite loop.

Processes in PyCSP

```
from pycsp.parallel import *


@process
def example(in, out):
    while True:
        input = in()
        # some processing.
        out(input)
```

Processes in PyCSP

- Parallel starts all processes together and blocks progression until they are finished
- Spawn starts all processes together without waiting for any to finish
- Sequence will start processes one at a time until they are all done

Processes in PyCSP

```
process_list = [
    process 1(),
    process 2()
]

Parallel(process_list)
```

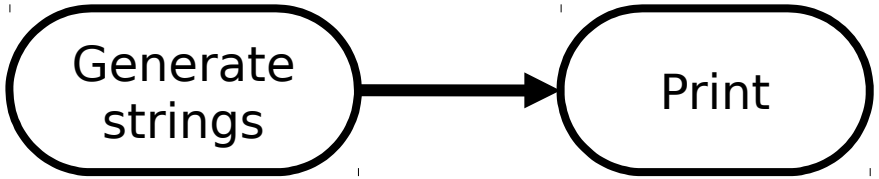
```
process_list = [
    process 1(),
    process 2()
]

Spawn(process_list)
```

```
process_list = [
    process 1(),
    process 2()
]

Sequence (process_list)
```

Building our first system in PyCSP

- Lets build a simple system in PyCSP
- It will generate strings
- These strings shall be displayed to the user

Building our first system in PyCSP



Building our first system in PyCSP

```
from pycsp.parallel import *

if __name__ == '__main__':
    string_generator_to_printer = Channel()

    process_list = [
        string_generator(string_generator_to_printer.writer()),
        printer(string_generator_to_printer.reader())
    ]

    Parallel(process_list)
```

Building our first system in PyCSP

```
from pycsp.parallel import *


@process
def string_generator(to_printer):
    a = 0
    while a < 10:
        to_printer("string_" + str(a))
        a += 1
```

Building our first system in PyCSP

```
from pycsp.parallel import *


@process
def printer(from_string_generator):
    while True:
        message = from_string_generator()
        print(message)
```

Alternatives

- Alternatives allow for non-determinism in a system
- A process can wait for input from several input channels and enact some code appropriate to the input channel used
- As well as channels, may also have timers or skips as inputs. Timers trigger after a given time and skips are always true.

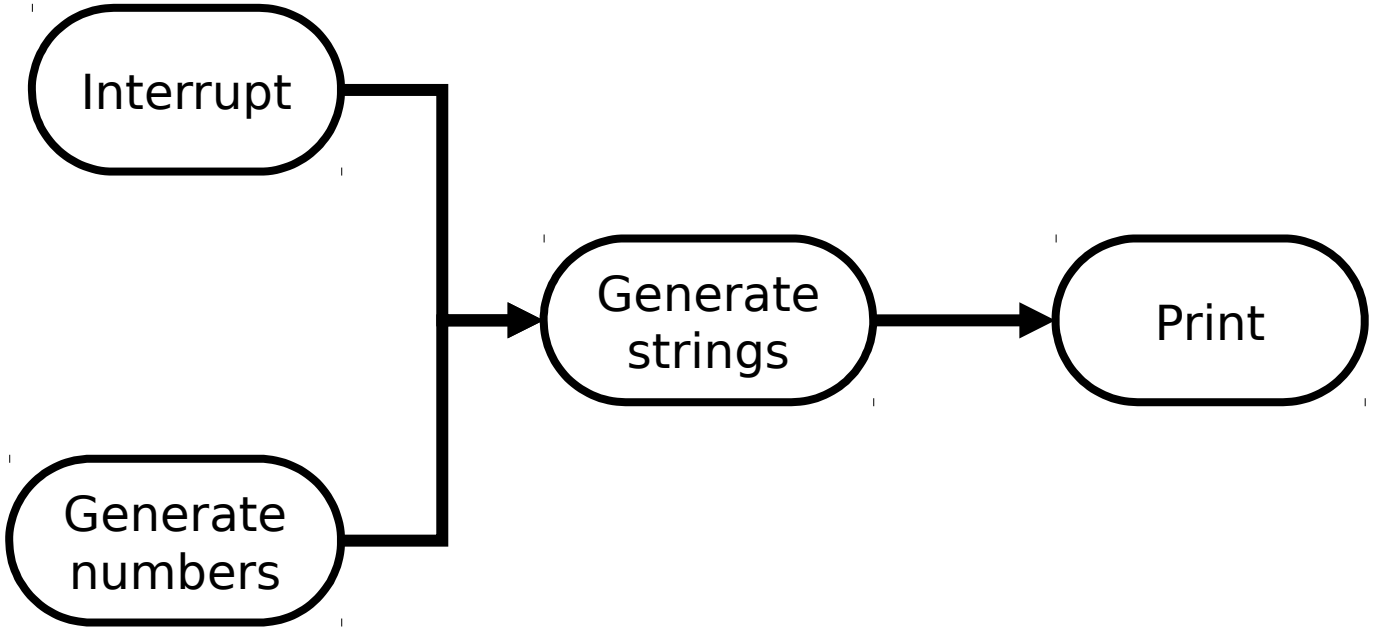
Alternatives

- PriSelect – select input based on the order given, with preference given to inputs declared first
- FairSelect – select input based on previous inputs. Defaults to PriSelect but will reorder inputs as they are used
- AltSelect – selects the fastest and defaults to PriSelect but may make a different choice based on network latency (?)

Building a slightly bigger system in PyCSP

- Lets build a slightly bigger system in PyCSP
- It will generate incrementing numbers
- These numbers shall be displayed to the user as strings
- Periodically this process will be interrupted and should let the user know this has happened

Building a slightly bigger system in PyCSP



Building a slightly bigger system in PyCSP

```
if __name__ == '__main__':
    number_generator_to_string_generator = Channel()
    interrupter_to_string_generator = Channel()
    string_generator_to_printer = Channel()

    process_list = [
    number_generator(number_generator_to_string_generator.writer()),
    interrupter(interrupter_to_string_generator.writer()),
    string_generator(number_generator_to_string_generator.reader(),
        interrupter_to_string_generator.reader(),
        string_generator_to_printer.writer()),
    printer(string_generator_to_printer.reader())]
    Parallel(process_list)
```

Building a slightly bigger system in PyCSP

```
@process
def number_generator(to_string_generator):
    number = 0
    while True:
        to_string_generator(number)
        number += 1
        time.sleep(0.5)
```

Building a slightly bigger system in PyCSP

```
@process
def interrupter(to_string_generator):
    while True:
        time.sleep(3)
        to_string_generator(0)
```

Building a slightly bigger system in PyCSP

```
@process
def printer(from_string_generator):
    while True:
        message = from_string_generator()
        print(message)
```

Building a slightly bigger system in PyCSP

```
@process
def string_generator(from_number_generator, from_interrupter, to_printer):
    while True:
        input_channel, input_message = PriSelect(
            InputGuard(from_number_generator),
            InputGuard(from_interrupter))
        if input_channel == from_number_generator:
            output_message = "string_" + str(input_message)
        if input_channel == from_interrupter:
           output_message = "INTERRUPTED"
        to_printer(output_message)
```