Other Useful Concepts

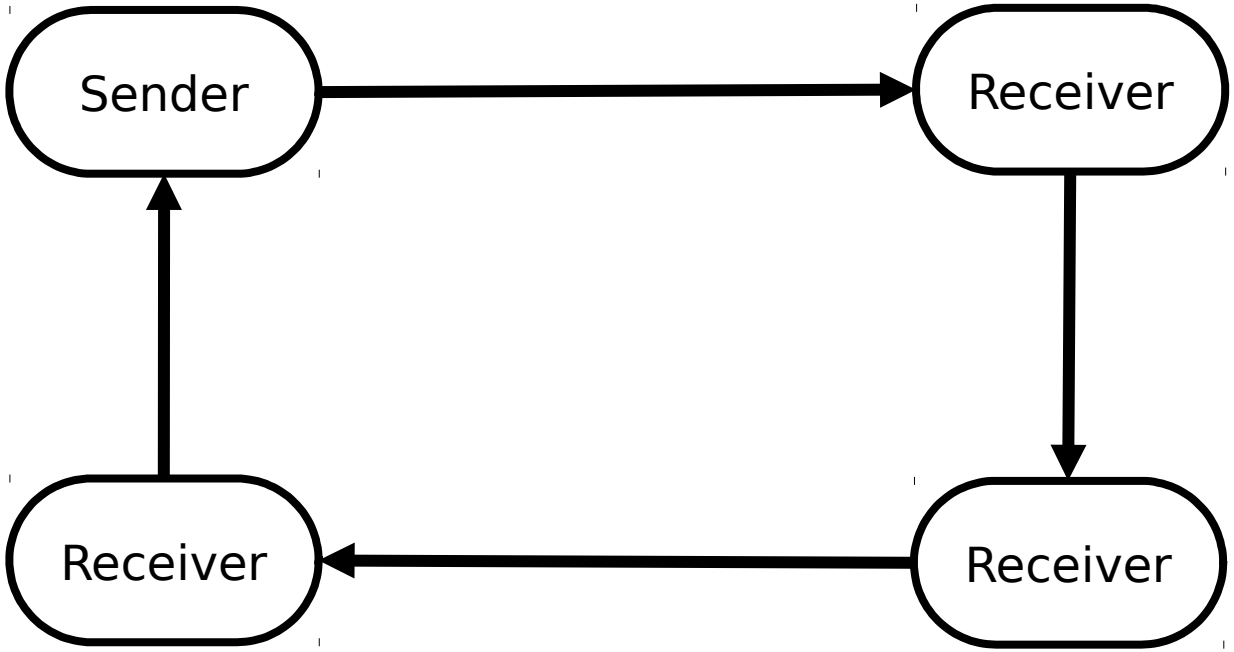
Agents, Barriers and PreCondtions

- These don’t explicitly exist in PyCSP
- But they’re useful concepts from JCSP
- They’re not essential, but they might be useful concepts to help think about concurrent systems

Agents

- Agents are special ‘processes’ that are passed between other processes
- They usually will have access to a channel to communicate back to their sender
- They may be passed over a network

Agents



Agents

```
@process
def receiver(name, out_channel, in_channel):
    while True:
        message = in_channel()
        message.run(name)
        out_channel(message)
```

Agents

```
@process
def sender(out_channel, in_channel):
    agent_to_sender = Channel()
    agent = Agent(agent_to_sender.writer())
    from_agent = agent_to_sender.reader()
    out_channel(agent)

    while True:
        input_channel, input_message = PriSelect(
            InputGuard(in_channel),
            InputGuard(from_agent))
        if input_channel == in_channel:
            print(input_message.names)
        if input_channel == from_agent:
            print(input_message)
```

Agents

```
class Agent:
    def __init__(self, to_sender):
        self.to_sender = to_sender
        self.names = list()

    def run(self, name):
        self.to_sender("hello from " + name)
        self.names.append(name)
```

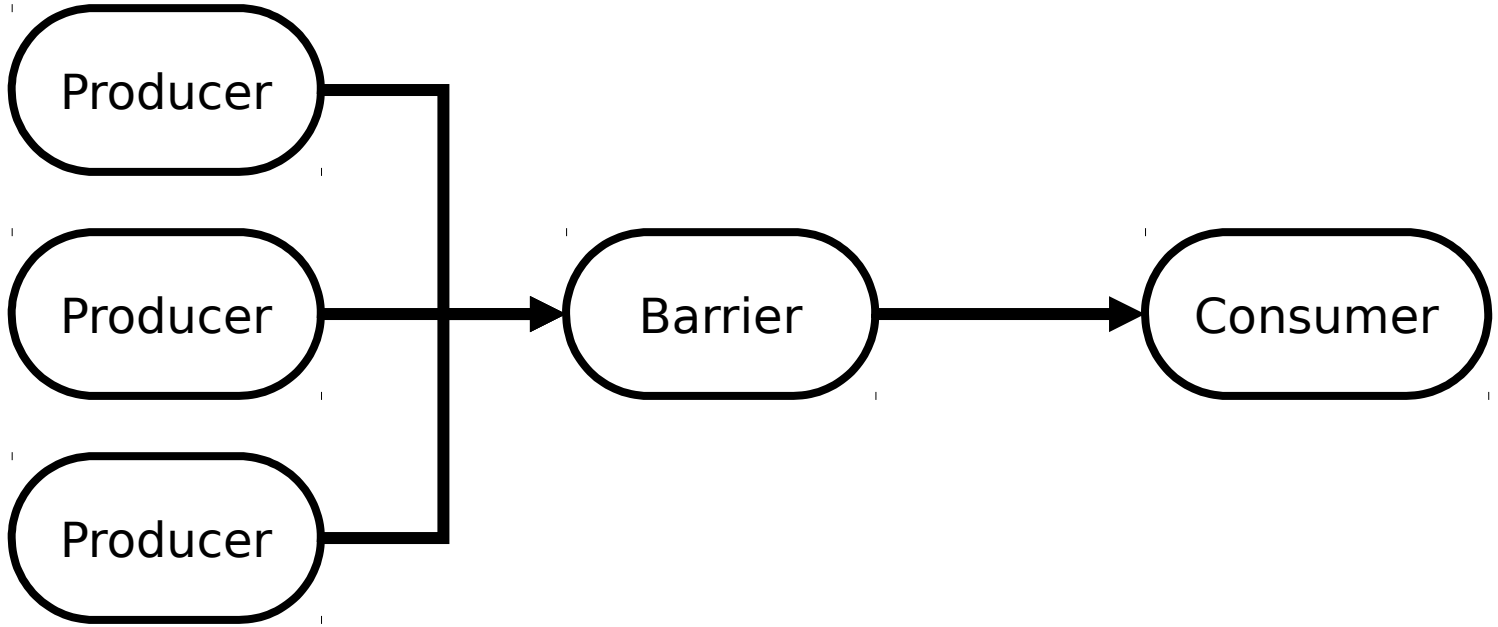
Agents

```
if __name__ == '__main__':
    channel_1 = Channel()
    channel_2 = Channel()
    channel_3 = Channel()
    channel_4 = Channel()

    process_list = [
        sender(channel_1.writer(), channel_4.reader()),
        receiver("Sam”, channel_2.writer(), channel_1.reader()),
        receiver("Gerty”, channel_3.writer(), channel_2.reader()),
        receiver("Tess”, channel_4.writer(), channel_3.reader())
    ]

    Parallel(process_list)
```

Barriers

- A barrier is a construct within a process
- Similar to an Alternative, but rather than picking one of many inputs, will wait till *n* of many inputs are ready
- Used to synchronise processes

Barriers



Barriers

```
@process
def producer(pro_num, to_barrier):
    sleep_for = random.randint(2, 10)
    print(str(pro_num) + " sleeping for " + str(sleep_for))
    time.sleep(sleep_for)
    print(str(pro_num) + " waking up")
    to_barrier(0)
```

Barriers

```
@process
def consumer(from_barrier):
    while True:
        from_barrier()
        print("Consumer received note from barrier")
```

Barriers

```
@process
def barrier(to_consumer, from_producers):
    barrier = [False] * len(from_producers)
    while True:
    input_channel, input_message = PriSelect(
        [InputGuard(from_producers[i].reader()) for i in range(len(from_producers))]
    )
    for i in range(len(from_producers)):
        if str(from_producers[i].reader()) == str(input_channel):
            barrier[i] = True
    if all(barrier):
        to_consumer(0)
        barrier = [False] * len(from_producers)
```

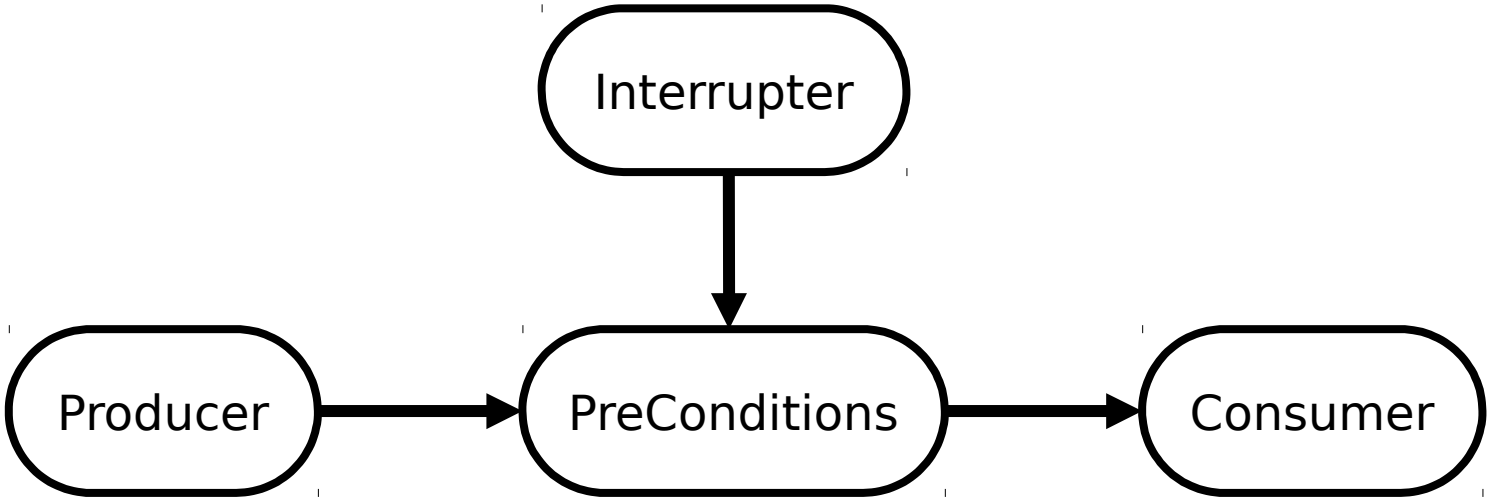
Barriers

```
if __name__ == "__main__":
    num = 5
    barrier_to_consumer = Channel()
    prod_to_barrier = Channel() * num

    Parallel(
        [producer(i, prod_to_barrier[i].writer()) for i in range(num)],
        barrier(barrier_to_consumer.writer(), prod_to_barrier),
        consumer(barrier_to_consumer.reader())
    )
```

PreConditions

- PreConditions are a bit of extra Alternative functionality
- Inputs are only considered if a Boolean is true
- Useful for being extra selective on what inputs we consider, and retaining greater control over the system

PreConditions



PreConditions

```
@process
def producer(to_preconditions):
    num = 0
    while True:
        to_preconditions("string_" + str(num))
        time.sleep(1)
        num += 1
```

PreConditions

```
@process
def consumer(from_preconditions):
    while True:
        message = from_preconditions()
        print(message)
```

PreConditions

```
@process
def interrupter(to_preconditions):
    while True:
        time.sleep(5)
        to_preconditions("stop")
        time.sleep(5)
        to_preconditions("start")
```

PreConditions

```
if __name__ == "__main__":
    pro_to_pre = Channel()
    int_to_pre = Channel()
    pre_to_con = Channel()

    Parallel(
        producer(pro_to_pre.writer()),
        interrupter(int_to_pre.writer()),
        preconditions(pro_to_pre.reader(), int_to_pre.reader(), pre_to_con.writer()),
        consumer(pre_to_con.reader())
    )
```

PreConditions

```
@process
def preconditions(from_prod, from_inter, to_con):
    PRODUCER, INTERRUPTER = 0, 1
    preconditions = [True] * 2
    guards = [None] * 2
    guards[PRODUCER] = InputGuard(from_prod)
    guards[INTERRUPTER] = InputGuard(from_inter)
```

PreConditions

```
while True:
    preconditioned = []
    for index, guard in enumerate(guards):
        if preconditions[index]:
            preconditioned.append(guard)

    input_channel, input_message = PriSelect(
        preconditioned
    )
```

PreConditions

```
    if input_channel == from_prod:
        to_con(input_message)
    if input_channel == from_inter:
        if input_message == "start":
            preconditions[PRODUCER] = True
        if input_message == "stop":
            preconditions[PRODUCER] = False
# End of While True
```

Plug ‘n’ Play

- Honourable mention must be made of the Plug ‘n’ Play processes
- These are simple processes bundled with Java
- Very simple processes
- But they’re prebuilt