Inter Process Communication using Conditions

- We use to use flag and lock together for synchronisation in mydata but here instead of using flag and lock together we use **conditions** variable for inter process communication.

- **Working of condition variable :**

- Producer will **wait** to acquire the lock upon the data

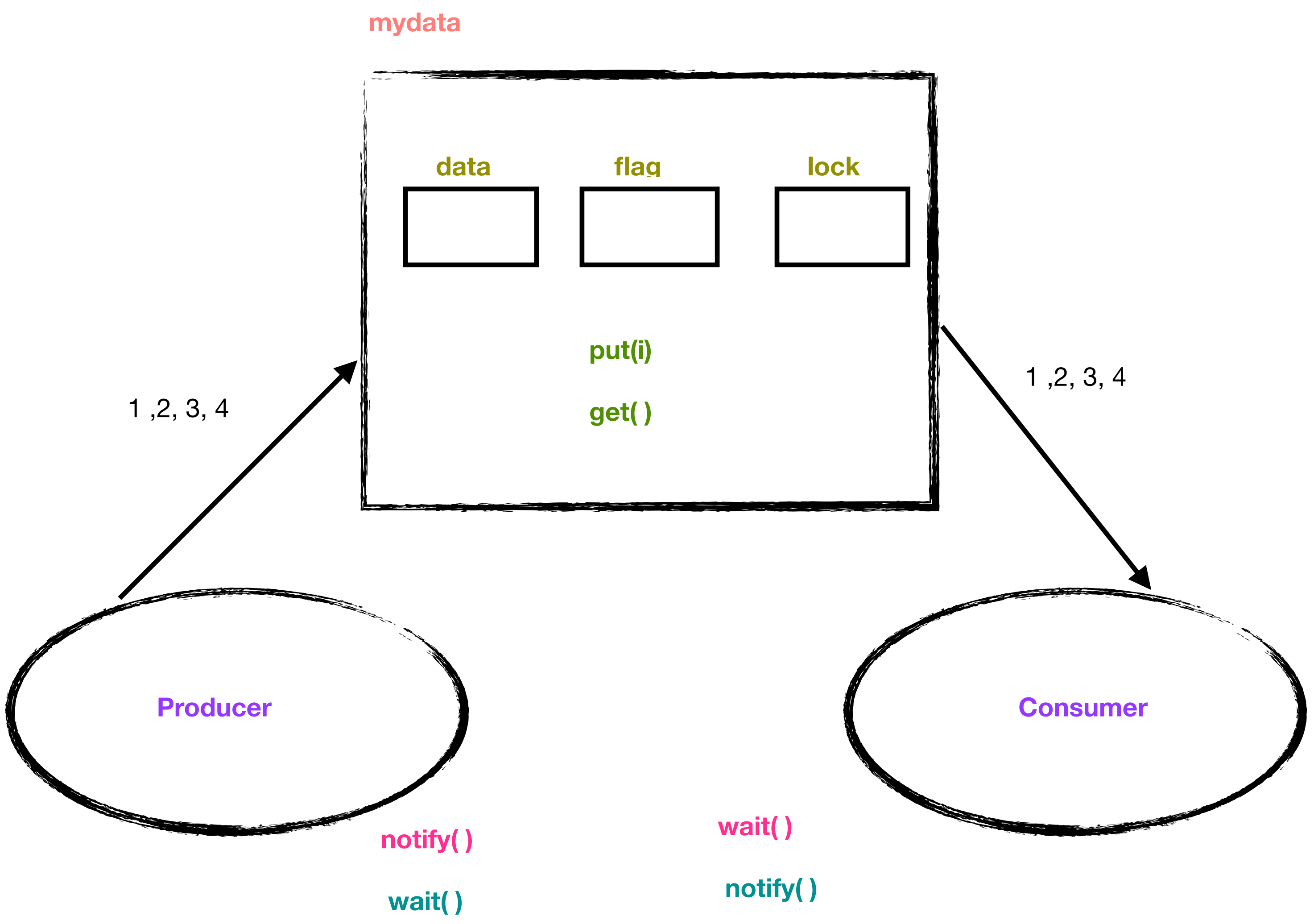
- Once producer gets the lock it'll lock the data object and write down data inside data object by using **put(i)**

- Consumer will wait until it get its turn , when it get its turn it'll lock the data object and get the data by using **get( )** method

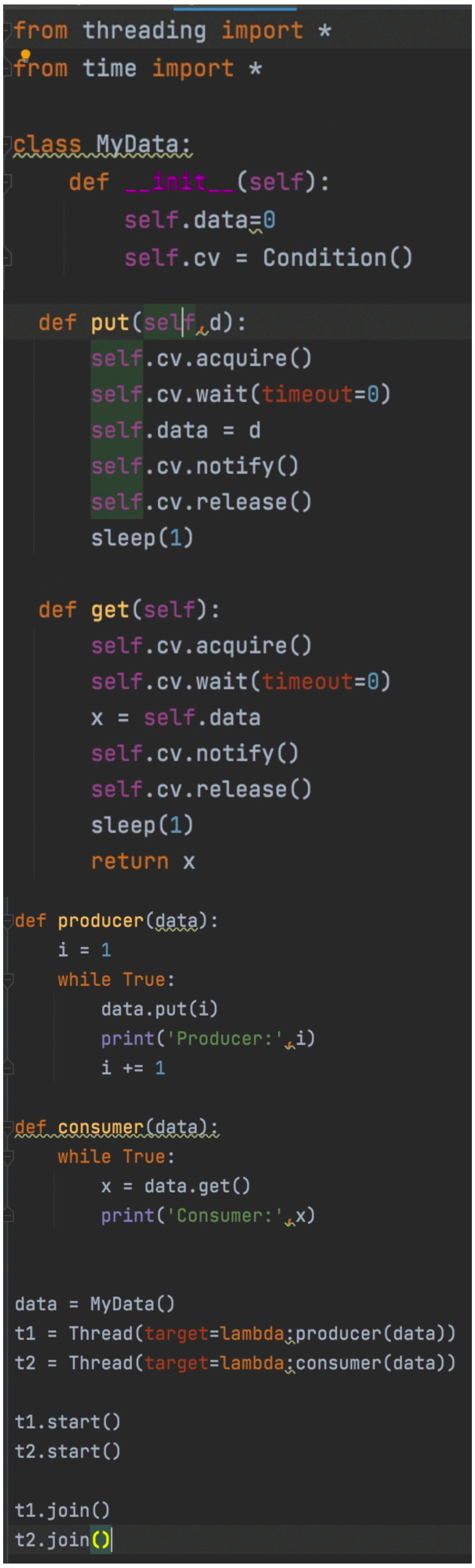
- Producer and consumer exchange their turn one by one

- If producer is producing **consumer** will **wait** when producer is done producing it'll **notify( )** waiting threads ,while consumer is consuming data **producer** will **wait** , once consumer finishes it work it'll **notify producer** saying it's your turn now.

Illustrated using a diagram , here flag and lock will be replaced by conditions



Program :



```python
from threading import *
from time import *

class MyData:
    def __init__(self):
        self.data=0
        self.cv = Condition()

    def put(self,d):
        self.cv.acquire()
        self.cv.wait(timeout=0)
        self.data = d
        self.cv.notify()
        self.cv.release()
        sleep(1)

    def get(self):
        self.cv.acquire()
        self.cv.wait(timeout=0)
        x = self.data
        self.cv.notify()
        self.cv.release()
        sleep(1)
        return x

def producer(data):
    i = 1
    while True:
        data.put(i)
        print('Producer:',i)
        i += 1

def consumer(data):
    while True:
        x = data.get()
        print('Consumer:',x)


data = MyData()
t1 = Thread(target=lambda:producer(data))
t2 = Thread(target=lambda:consumer(data))

t1.start()
t2.start()

t1.join()
t2.join()
```

# importing required module and creating a class

# creating put and get method so, that if producer is producing consumer should wait and vice versa

# initiating threads and returning the result