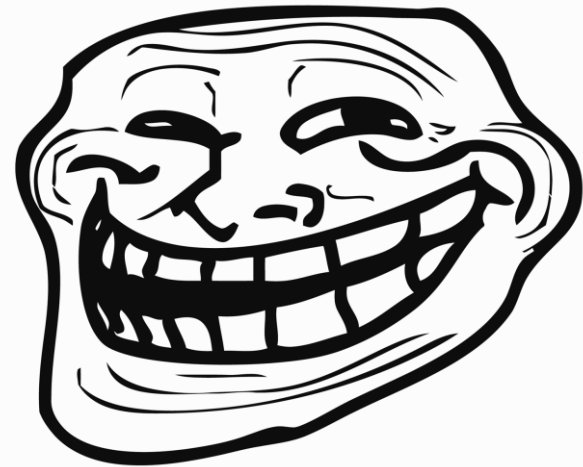


Message Broker System



Ahmedur Rahman Shovon
Codalo
shovon.sylhet@gmail.com

But I'll actually talk about RabbitMQ



“Message broker translates a message from the formal messaging protocol of the sender to the formal messaging protocol of the receiver.”

From Wikipedia, the free encyclopedia

RabbitMQ is open source message broker software that implements the Advanced Message Queuing Protocol (AMQP).

The principal idea is pretty simple: it accepts and forwards messages. You can think about it as a post office: when you send mail to the post box you're pretty sure that Mr. Postman will eventually deliver the mail to your recipient. Using this metaphor RabbitMQ is a post box, a post office and a postman!

What can RabbitMQ do for you?

- Messaging enables software applications to connect and scale. Applications can connect to each other. Messaging is asynchronous, decoupling applications by separating sending and receiving data.
- Data delivery, non-blocking operations or push notifications, publish / subscribe, asynchronous processing, or work queues.
- RabbitMQ is a messaging broker - an intermediary for messaging. It gives your applications a common platform to send and receive messages, and your messages a safe place to live until received.

Feature Highlights

- Reliability
- Flexible Routing
- Clustering and Federation
- Highly Available Queues
- Multi-protocol with Many Clients
- Plugin System

PRODUCER, QUEUE, CONSUMER

Remember these things please

A *producer* is a user application that sends messages.

A *queue* is a buffer that stores messages.

A *consumer* is a user application that receives messages.

Now let's see messaging in action!



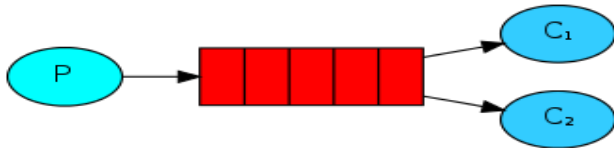
1 "Hello World!"

The simplest thing that does *something*



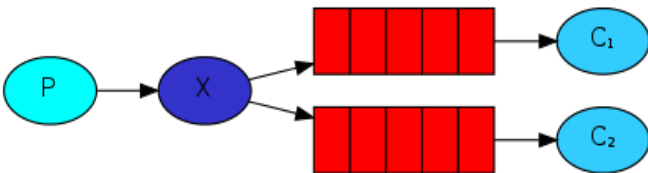
2 Work queues

Distributing tasks among workers



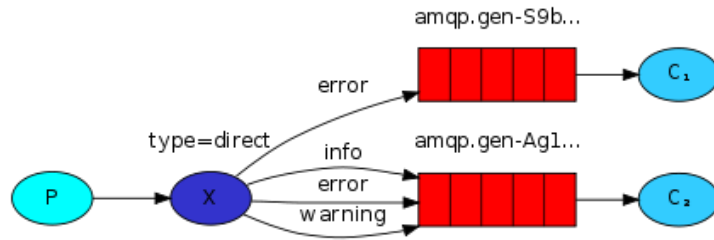
3 Publish/Subscribe

Sending messages to many consumers at once



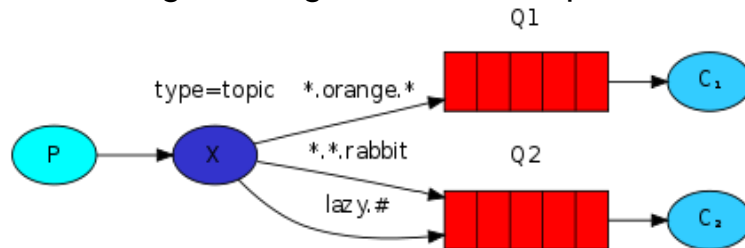
4 Routing

Receiving messages selectively



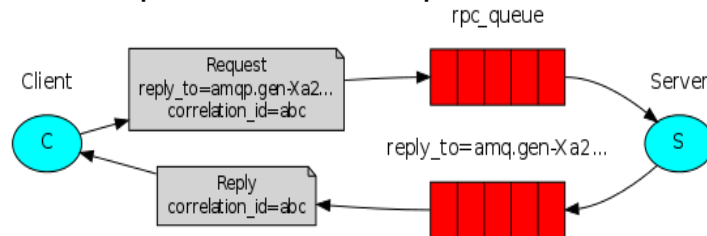
5 Topics

Receiving messages based on a pattern

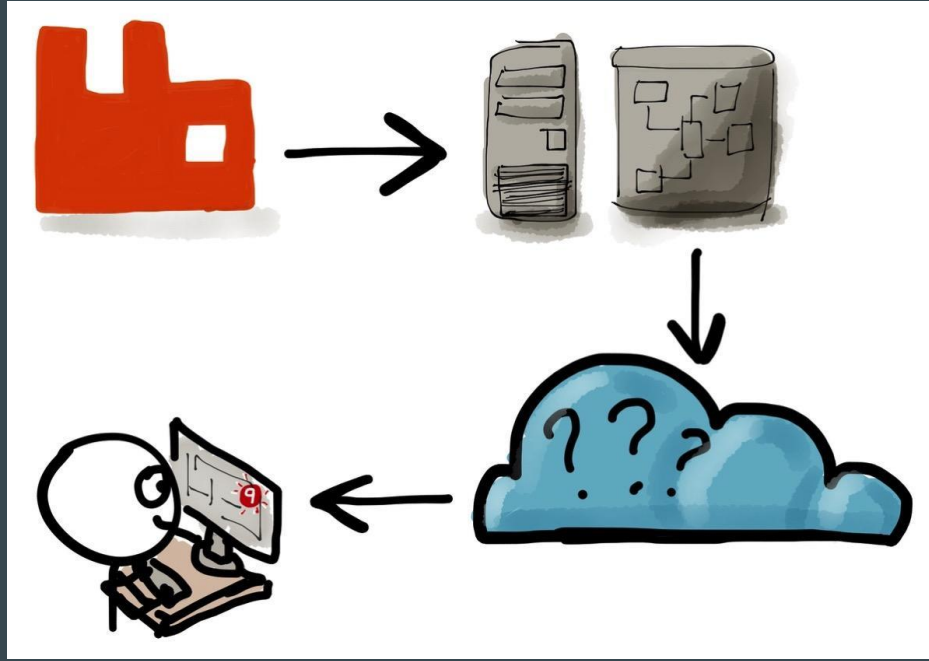


6 RPC

Remote procedure call implementation



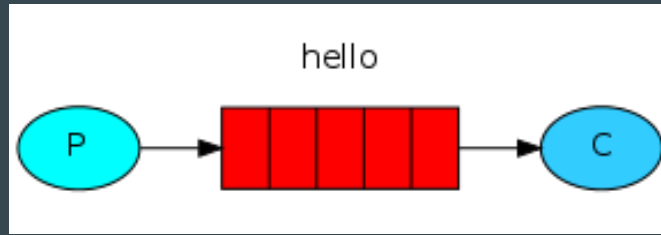
What should I do to play with Rabbits?



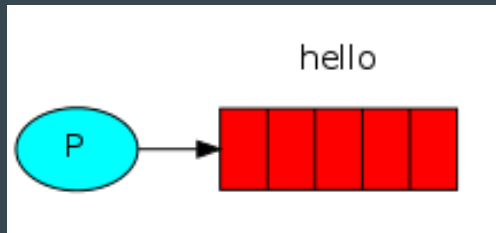
Have a Windows machine? Start installing!

- ERLANG 64 bit exe: otp_win64_19.1.exe
- MS Visual C++ redistributable 2013 64 bit
- RabbitMQ Server: rabbitmq-server-3.6.5.exe
- RabbitMQ libraries based on the language
- Python client recommended by the RabbitMQ team: Pika
- Pika Installation: pip install pika

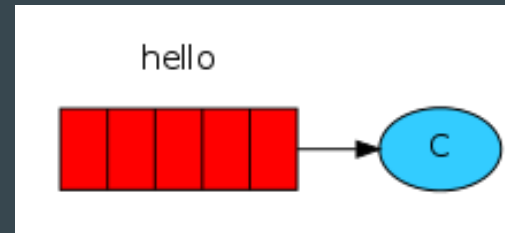
1 “Hello World!” - The simplest thing that does something



Our "Hello world" won't be too complex – let's send a message, receive it and print it on the screen. To do so we need two programs: one that sends a message and one that receives and prints it.



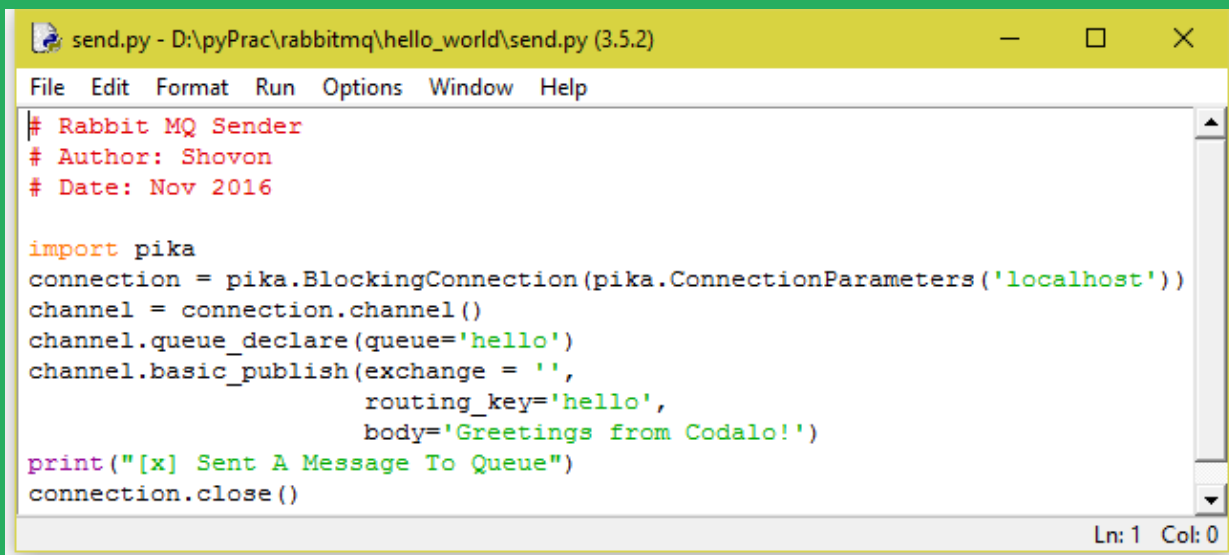
Sending message to queue



Receiving message from queue

Sending message...

Producer Script

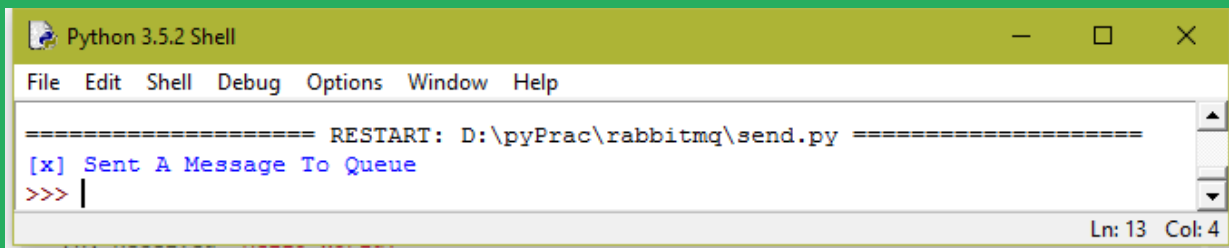


```
send.py - D:\pyPrac\rabbitmq\hello_world\send.py (3.5.2)
File Edit Format Run Options Window Help
# Rabbit MQ Sender
# Author: Shovon
# Date: Nov 2016

import pika
connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')
channel.basic_publish(exchange = '',
                    routing_key='hello',
                    body='Greetings from Codalo!')
print("[x] Sent A Message To Queue")
connection.close()

Ln: 1 Col: 0
```

Message is sent to the queue



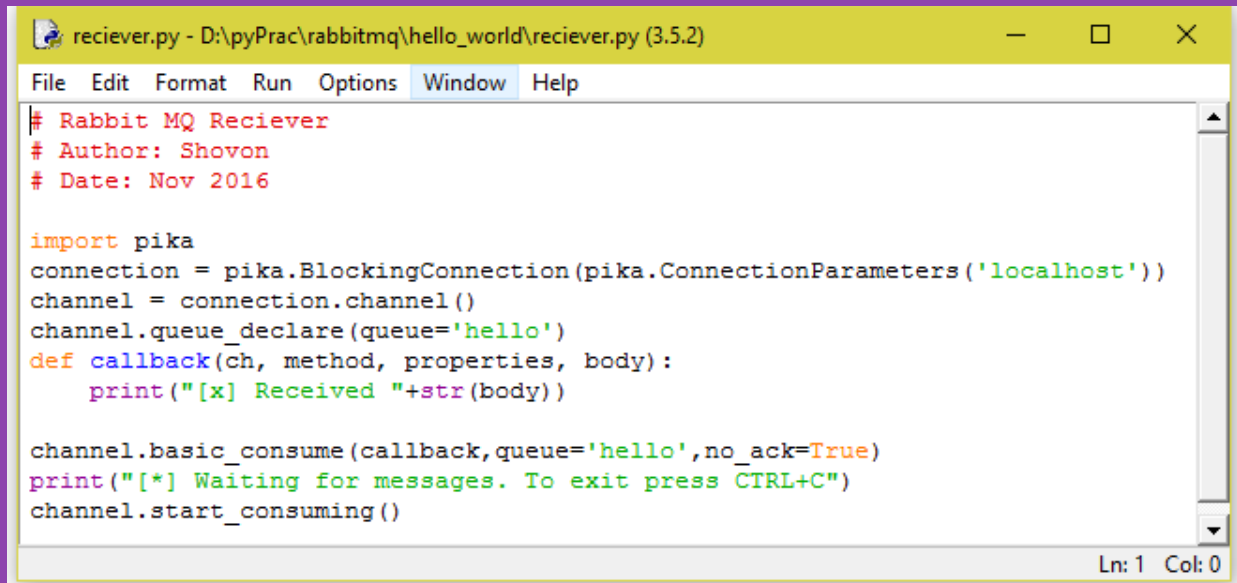
```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
===== RESTART: D:\pyPrac\rabbitmq\send.py =====
[x] Sent A Message To Queue
>>> |

Ln: 13 Col: 4
```

Receiving message...

Consumer Script

Message is received
from the queue

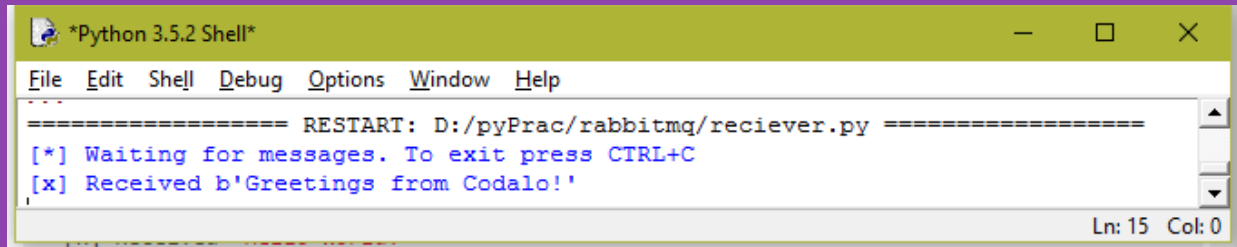


```
reciever.py - D:\pyPrac\rabbitmq\hello_world\reciever.py (3.5.2)
File Edit Format Run Options Window Help
# Rabbit MQ Reciever
# Author: Shovon
# Date: Nov 2016

import pika
connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')
def callback(ch, method, properties, body):
    print("[x] Received "+str(body))

channel.basic_consume(callback,queue='hello',no_ack=True)
print("[*] Waiting for messages. To exit press CTRL+C")
channel.start_consuming()

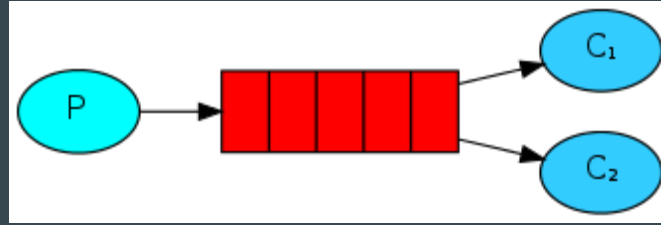
Ln: 1 Col: 0
```



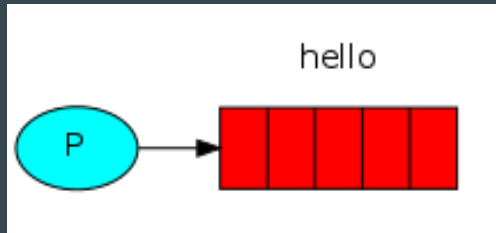
```
*Python 3.5.2 Shell*
File Edit Shell Debug Options Window Help
===== RESTART: D:/pyPrac/rabbitmq/reciever.py =====
[*] Waiting for messages. To exit press CTRL+C
[x] Received b'Greetings from Codalo!'

Ln: 15 Col: 0
```

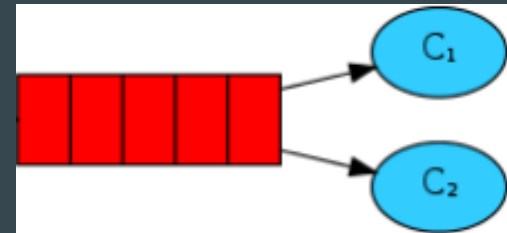
2 Work queues - Distributing tasks among workers



The main idea behind Work Queues (aka: Task Queues) is to avoid doing a resource-intensive task immediately and having to wait for it to complete. Instead we schedule the task to be done later.



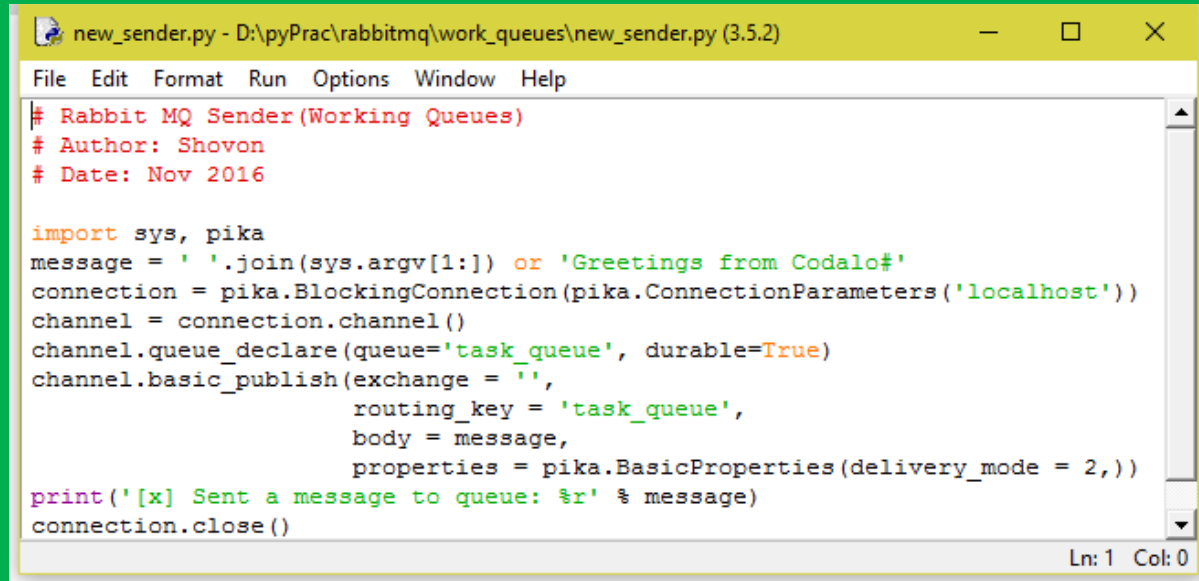
Sending message to queue



Multiple consumers receiving message

Producer's script (new_sender.py)

Producer Script is now updated to allow run time argument which contains messages

A screenshot of a Python script editor window titled 'new_sender.py - D:\pyPrac\rabbitmq\work_queues\new_sender.py (3.5.2)'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code is as follows:

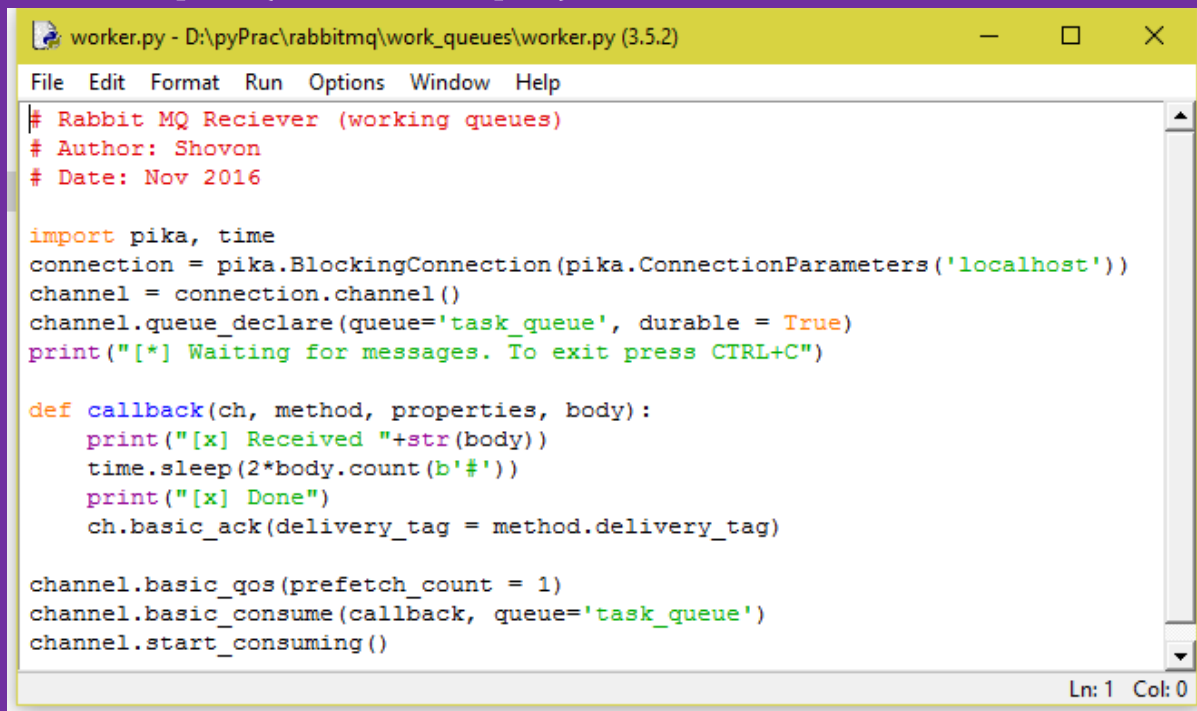
```
# Rabbit MQ Sender (Working Queues)
# Author: Shovon
# Date: Nov 2016

import sys, pika
message = ' '.join(sys.argv[1:]) or 'Greetings from Codalo#'
connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='task_queue', durable=True)
channel.basic_publish(exchange = '',
                    routing_key = 'task_queue',
                    body = message,
                    properties = pika.BasicProperties(delivery_mode = 2,))
print('[x] Sent a message to queue: %r' % message)
connection.close()
```

The status bar at the bottom right shows 'Ln: 1 Col: 0'.

Consumer's / Worker's script (worker.py)

Receiver Script is now updated to allow pause to process the queue element. Lets think each consumer as a worker. We ensure the fair dispatch of queues using *prefetch_count=1*. Each “#” in message takes 2 seconds to process.

A screenshot of a Python IDE window titled "worker.py - D:\pyPrac\rabbitmq\work_queues\worker.py (3.5.2)". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The code is as follows:

```
# Rabbit MQ Reciever (working queues)
# Author: Shovon
# Date: Nov 2016

import pika, time
connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='task_queue', durable = True)
print("[*] Waiting for messages. To exit press CTRL+C")

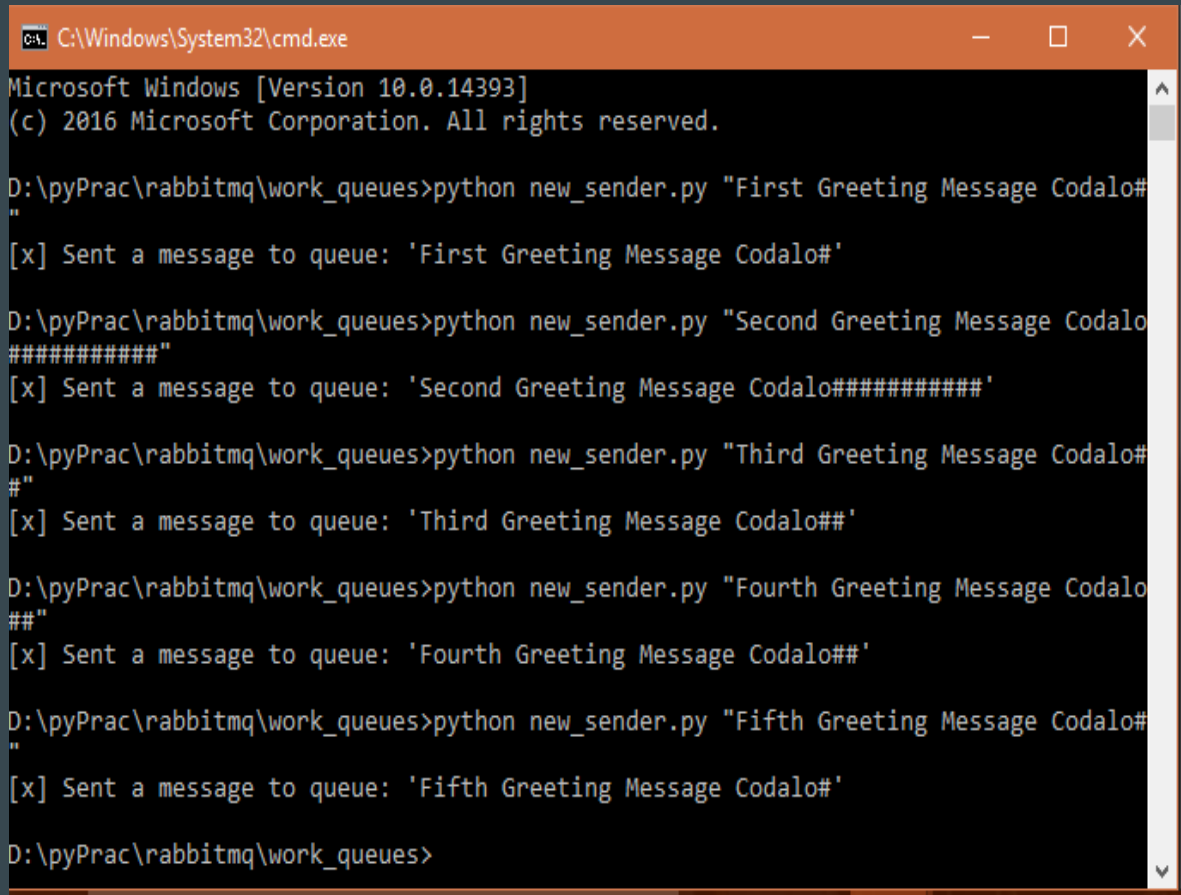
def callback(ch, method, properties, body):
    print("[x] Received "+str(body))
    time.sleep(2*body.count(b'#'))
    print("[x] Done")
    ch.basic_ack(delivery_tag = method.delivery_tag)

channel.basic_qos(prefetch_count = 1)
channel.basic_consume(callback, queue='task_queue')
channel.start_consuming()
```

The status bar at the bottom right shows "Ln: 1 Col: 0".

Sending messages...

- Open two cmd and run worker.py in both prompt.
- Open another cmd and run new_sender.py.
- The messages with lot of # will take much time to process in worker



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

D:\pyPrac\rabbitmq\work_queues>python new_sender.py "First Greeting Message Codalo#"
[x] Sent a message to queue: 'First Greeting Message Codalo#'

D:\pyPrac\rabbitmq\work_queues>python new_sender.py "Second Greeting Message Codalo#####"
[x] Sent a message to queue: 'Second Greeting Message Codalo#####'

D:\pyPrac\rabbitmq\work_queues>python new_sender.py "Third Greeting Message Codalo##"
[x] Sent a message to queue: 'Third Greeting Message Codalo##'

D:\pyPrac\rabbitmq\work_queues>python new_sender.py "Fourth Greeting Message Codalo###"
[x] Sent a message to queue: 'Fourth Greeting Message Codalo###'

D:\pyPrac\rabbitmq\work_queues>python new_sender.py "Fifth Greeting Message Codalo#"
[x] Sent a message to queue: 'Fifth Greeting Message Codalo#'

D:\pyPrac\rabbitmq\work_queues>
```

Receiving messages...

First worker received and processed messages faster because the fifth worker takes a lot of time to process only one message which takes time to process.

```
C:\Windows\System32\cmd.exe - python worker.py
D:\pyPrac\rabbitmq\work_queues>python worker.py
[*] Waiting for messages. To exit press CTRL+C
[x] Received b'First Greeting Message Codalo#'
[x] Done
[x] Received b'Third Greeting Message Codalo##'
[x] Done
[x] Received b'Fourth Greeting Message Codalo##'
[x] Done
[x] Received b'Fifth Greeting Message Codalo#'
[x] Done
```

```
C:\Windows\System32\cmd.exe - python worker.py
(c) 2016 Microsoft Corporation. All rights reserved.
D:\pyPrac\rabbitmq\work_queues>python worker.py
[*] Waiting for messages. To exit press CTRL+C
[x] Received b'Second Greeting Message Codalo#
#####'
[x] Done
```

Real World Uses

- Fast logging solution
- Sending emails
- Sending SMSs
- Background processing (data analysis)

Want to learn more?

- Go to the official site of RabbitMQ: <https://www.rabbitmq.com/>
- How about Wiki? : <https://en.wikipedia.org/wiki/RabbitMQ>

And pass the messages... :D

Thank you.