Introdução a RabbitMQ e AMQP

Carlos Mattoso

Quarta, 3 de Fevereiro de 2016

Programa

- AMQP
- RabbitMQ: Exemplos de Uso em JavaScript
- *librabbitmq:* API C
- Sistemas competidores

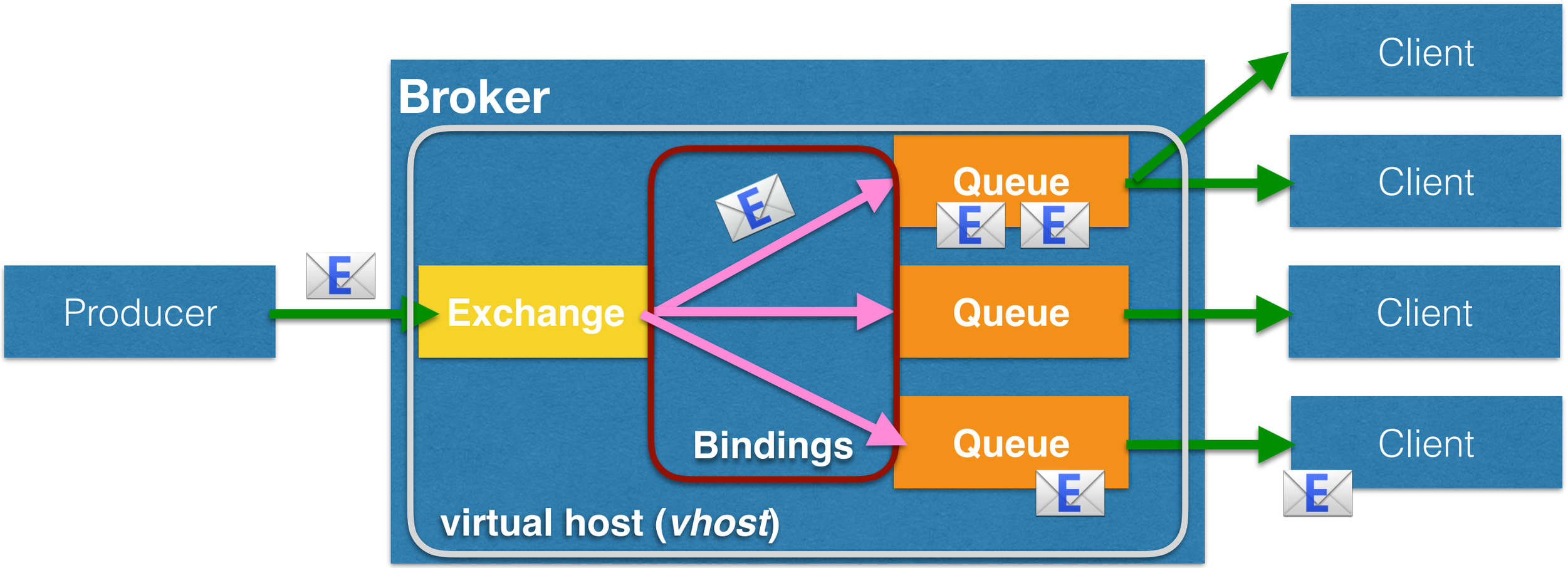
Programa

- **AMQP**
  - RabbitMQ: Exemplos de Uso em JavaScript
  - *librabbitmq:* API C
  - Sistemas competidores

AMQP

- **A**dvanced **M**essaging **Q**ueueing **P**rotocol
- Desenvolvido inicialmente para fins de interoperabilidade
- Facilitar desenvolvimentos de aplicações distribuídas e desacopladas
- Programático: *provisioning* através de *method calls*

Entidades


Broker
Producer
E
Exchange
Bindings
Queue
Queue
Queue
Client
Client
Client
Client
virtual host (*vhost*)

Mensagens

- Mensagens são compostas por: *attributes* e *payloads*
- Exemplos de *attributes: Content Type, Content Encoding, Routing key, Delivery mode (persistent or not), etc*
- *Payload* é um *byte array* opaco
- *Headers* servem para definição de atributos específicos de um broker
- *Acknowledgements*

Client/Producer Connections

- A conexão entre aplicações de usuário e o broker são feitas através de conexões TCP *long-lived*

- Em cada conexão deve haver pelo menos um *channel*, mas vários podem ser declarados

- *Channels* são multiplexados através uma única conexão TCP

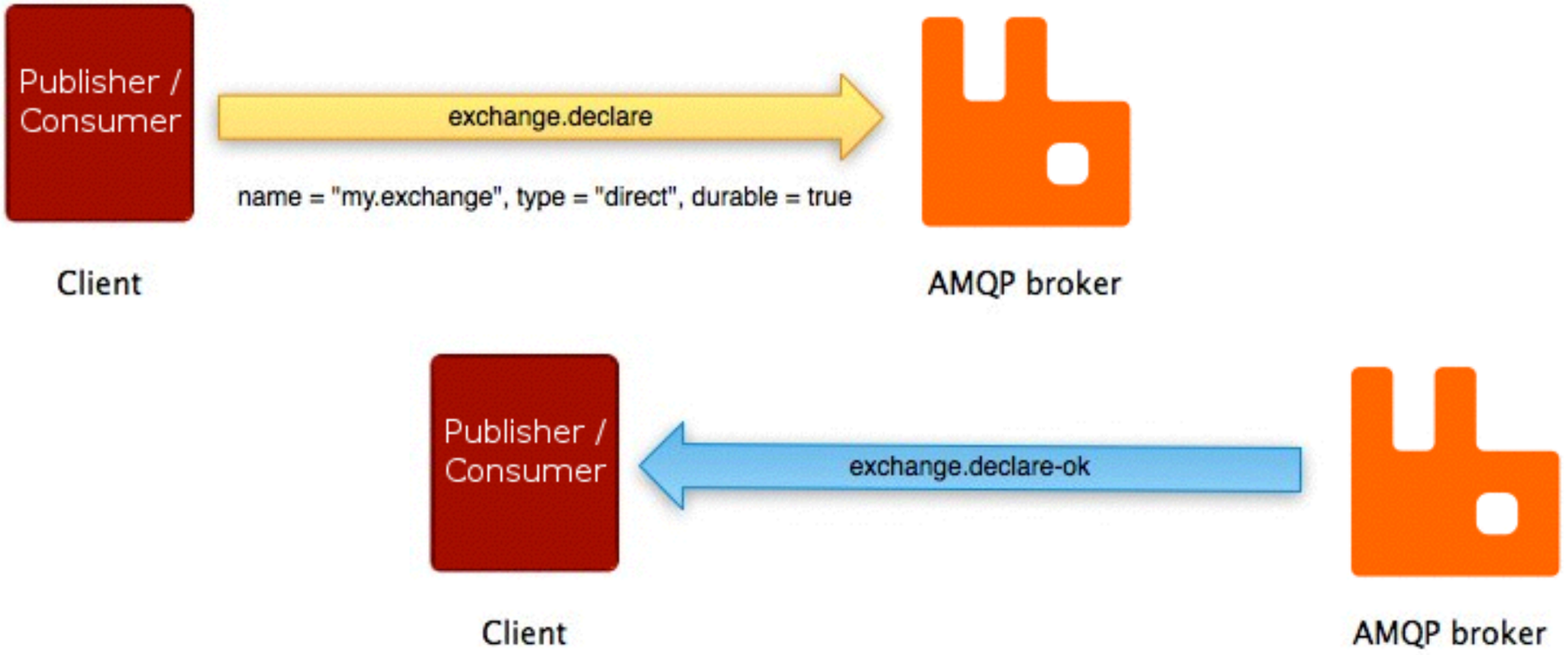
- Um *channel* por thread; uso de *channels* reduz *overhead* de sua criação/destruição

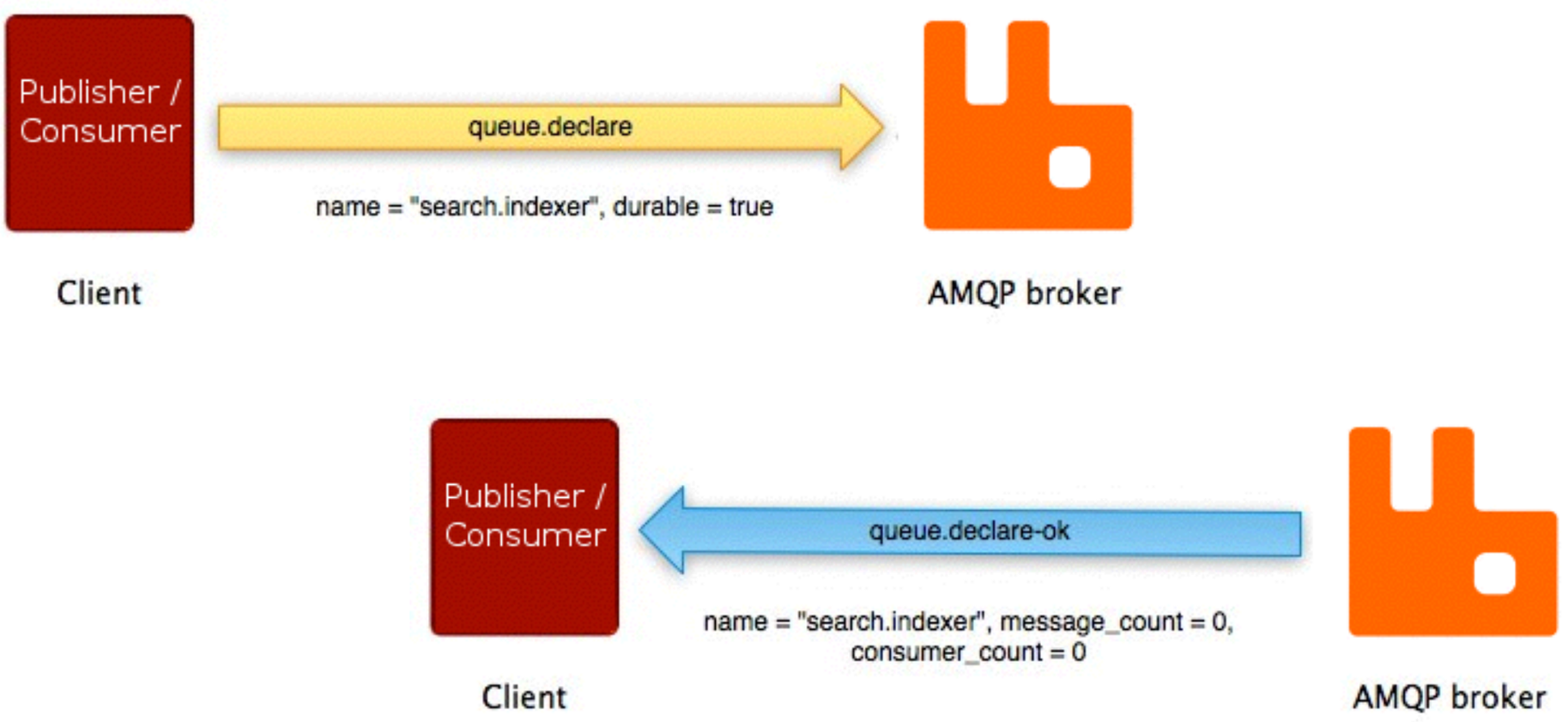
Methods

- Definem uma operação a ser executada pelo broker

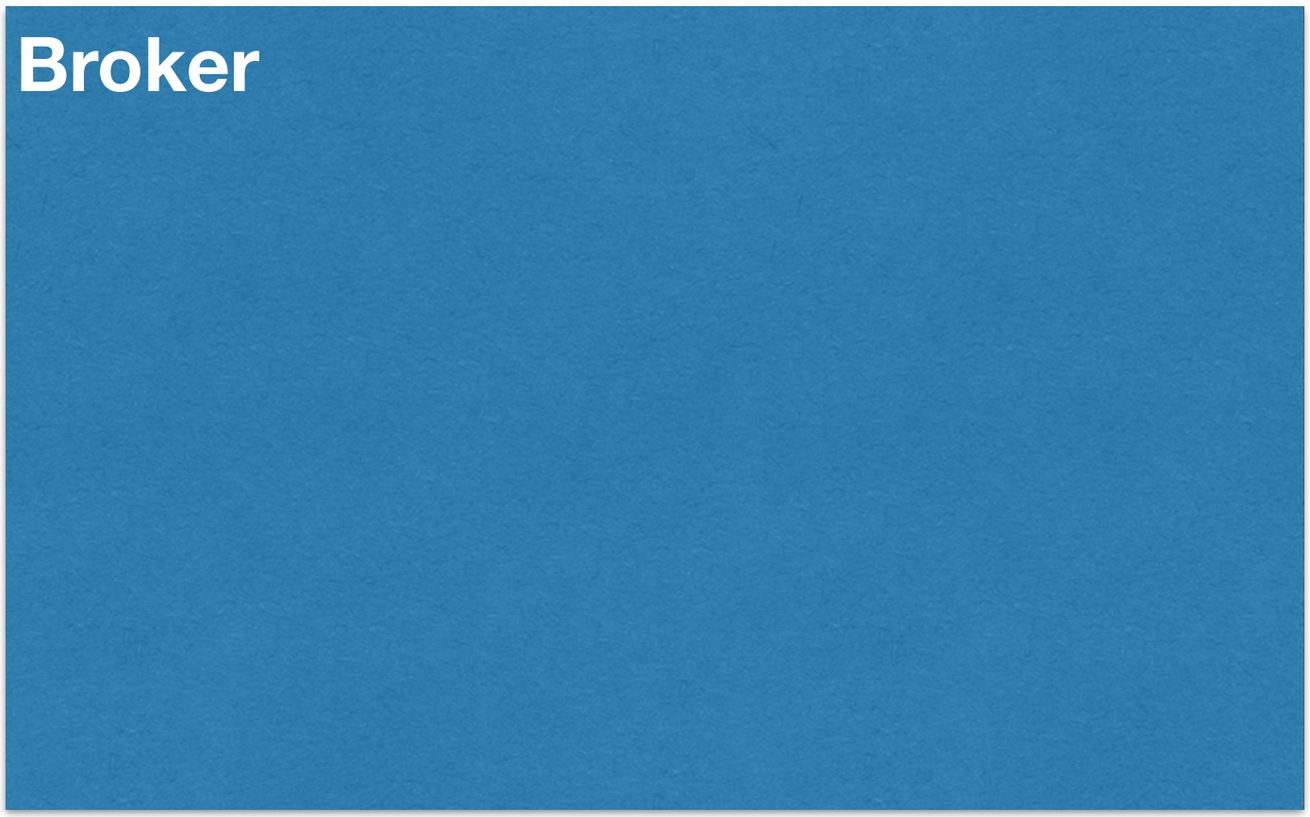
- *basic.publish*: envio de mensagem por um *producer*

- *basic.consume*: inscreve uma aplicação para recebimento (*push*) de mensagens de uma *queue*


Publisher / Consumer
exchange.declare
name = "my.exchange", type = "direct", durable = true
Client
AMQP broker
Publisher / Consumer
exchange.declare-ok
Client
AMQP broker


Publisher / Consumer
queue.declare
name = "search.indexer", durable = true
Client
AMQP broker
Publisher / Consumer
queue.declare-ok
name = "search.indexer", message_count = 0,
consumer_count = 0
Client
AMQP broker

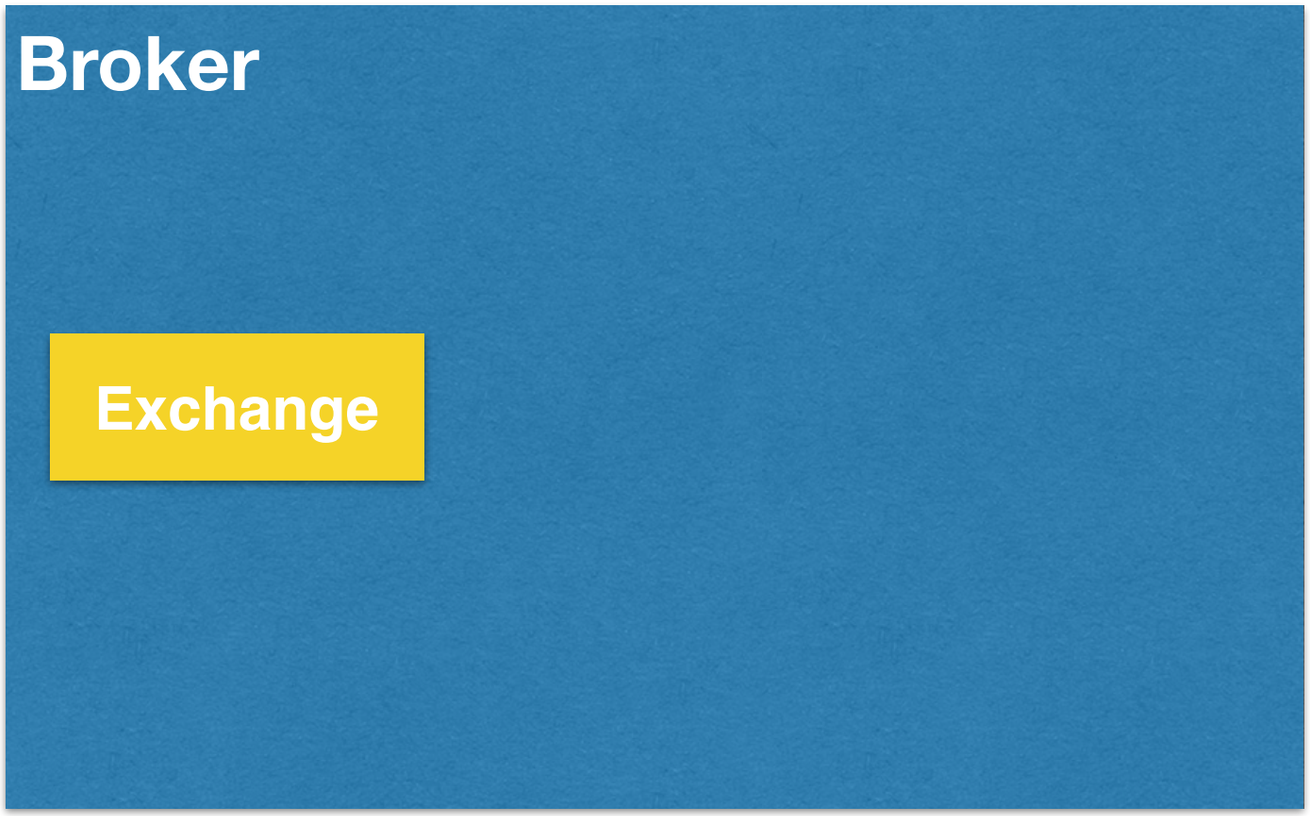
Entidades



(Message) Broker

- O *broker* é um servidor que implementa o protocolo AMQP
- RabbitMQ é um broker
- *Brokers* podem estender o protocolo

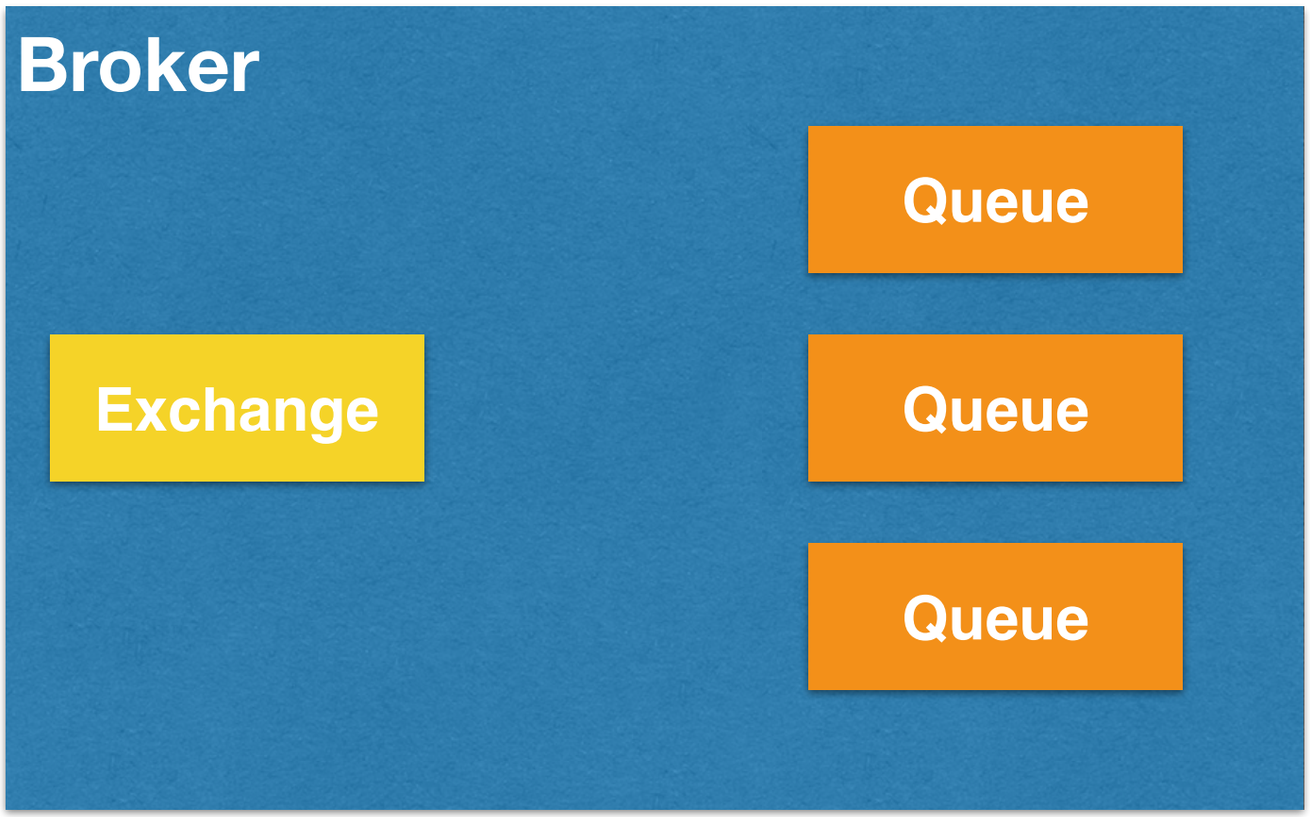
Entidades



Exchange

- Entidade que distribui mensagens para as filas
- Tem tipos: *direct, fanout, topic* e *headers*
- Mensagens sempre são roteadas por um *exchange*
- Dependendo do tipo, filtram as mensagens
- *Durability* sob controle do usuário
- O broker disponibiliza alguns *exchanges* padrão

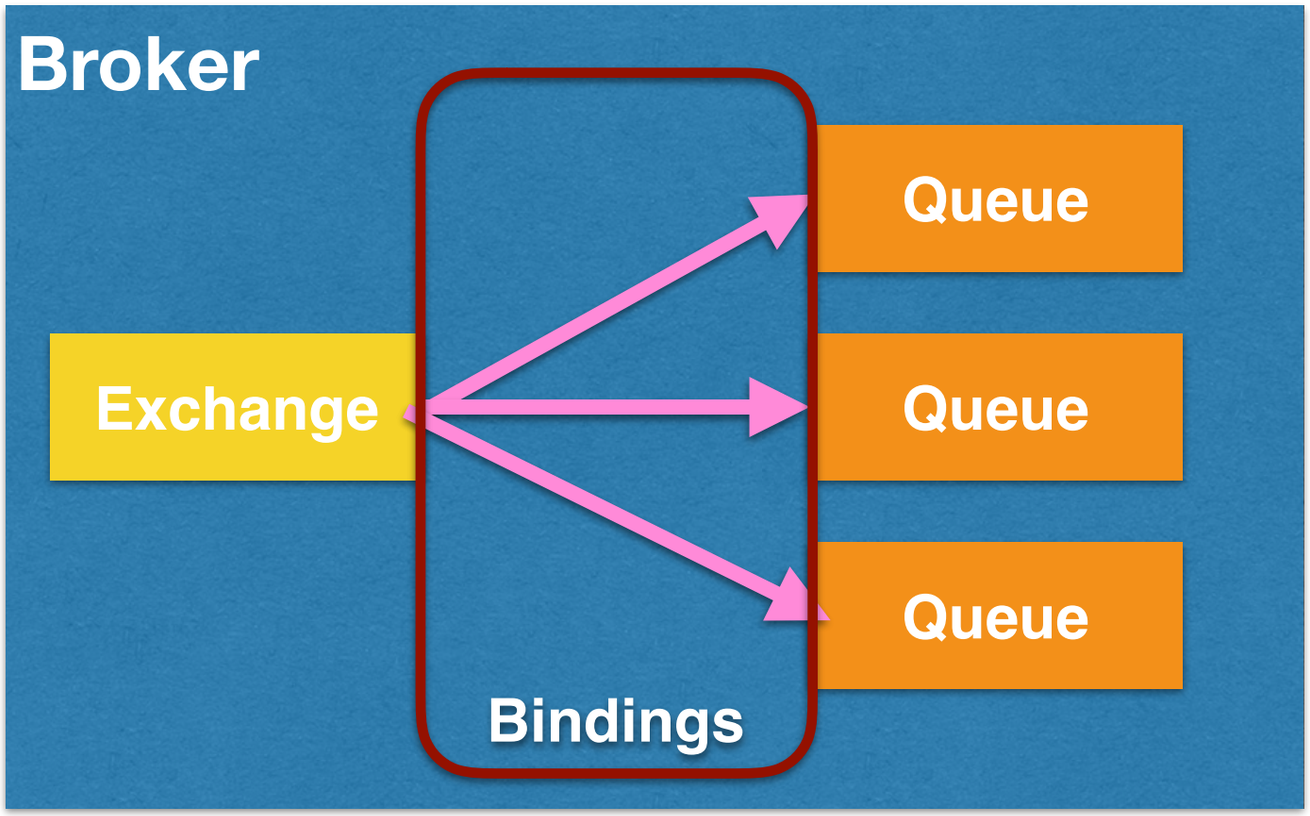
Entidades



Queue

- Servem como uma caixa de mensagens
- Sempre recebem mensagens de um *exchange*
- Um *client* recebe mensagens por *pull (basic.get)* ou *push (basic.consume)*
- *Durability* sobre controle do usuário

Entidades



Bindings

- Rota de comunicação entre um *exchange* e uma *queue*
- Define uma *binding key*
- Definição de filtros para roteamento de mensagens
- *Direct exchange:* checa igualdade entre *routing key* da mensagem a *binding key*
- *Topic exchange*: a binding key tem forma hierárquica, logo o filtro pode operar sobre a hierarquia (ex: us.west.zone1); # ou *

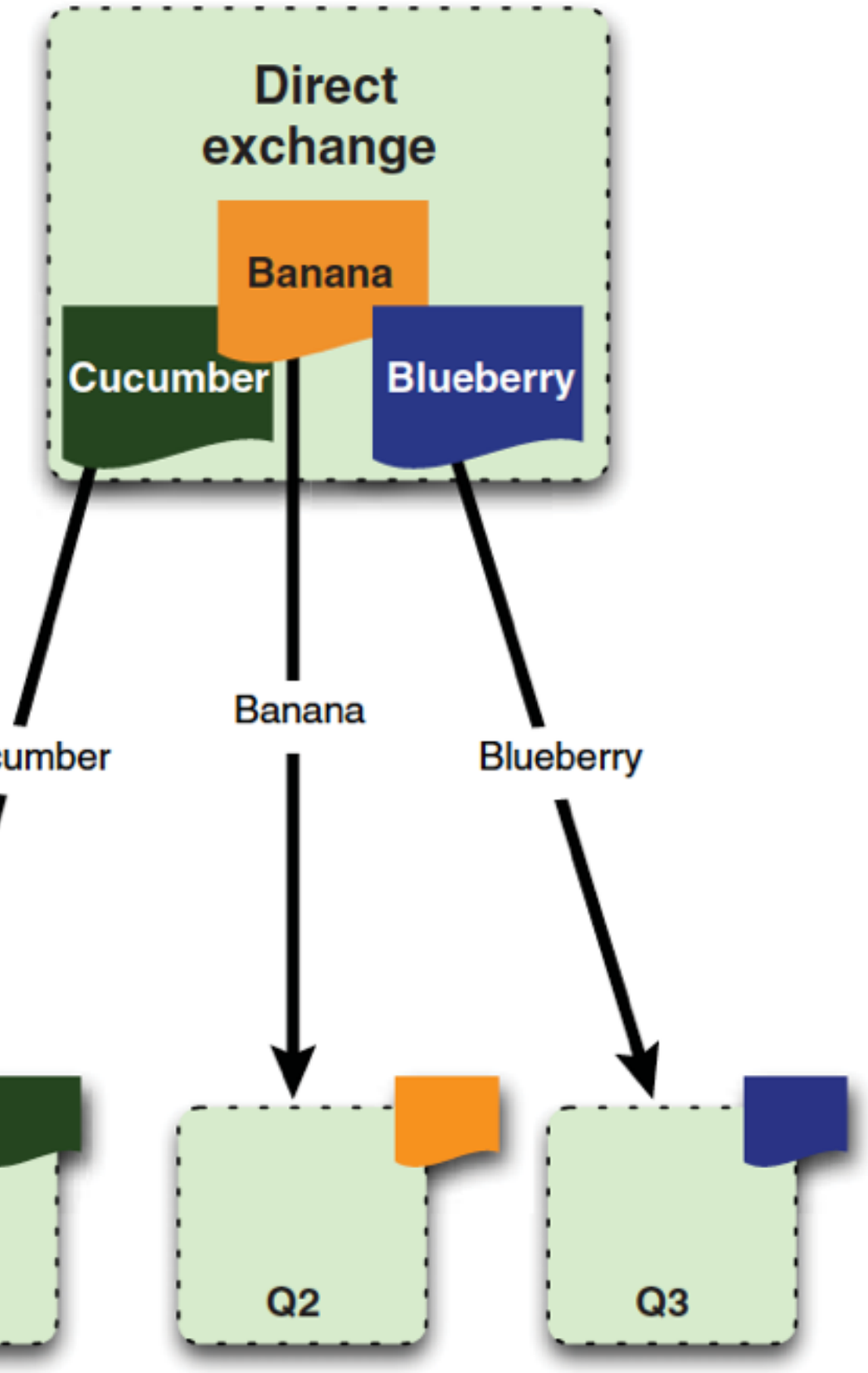


Figure 2.4 Direct exchange message flow

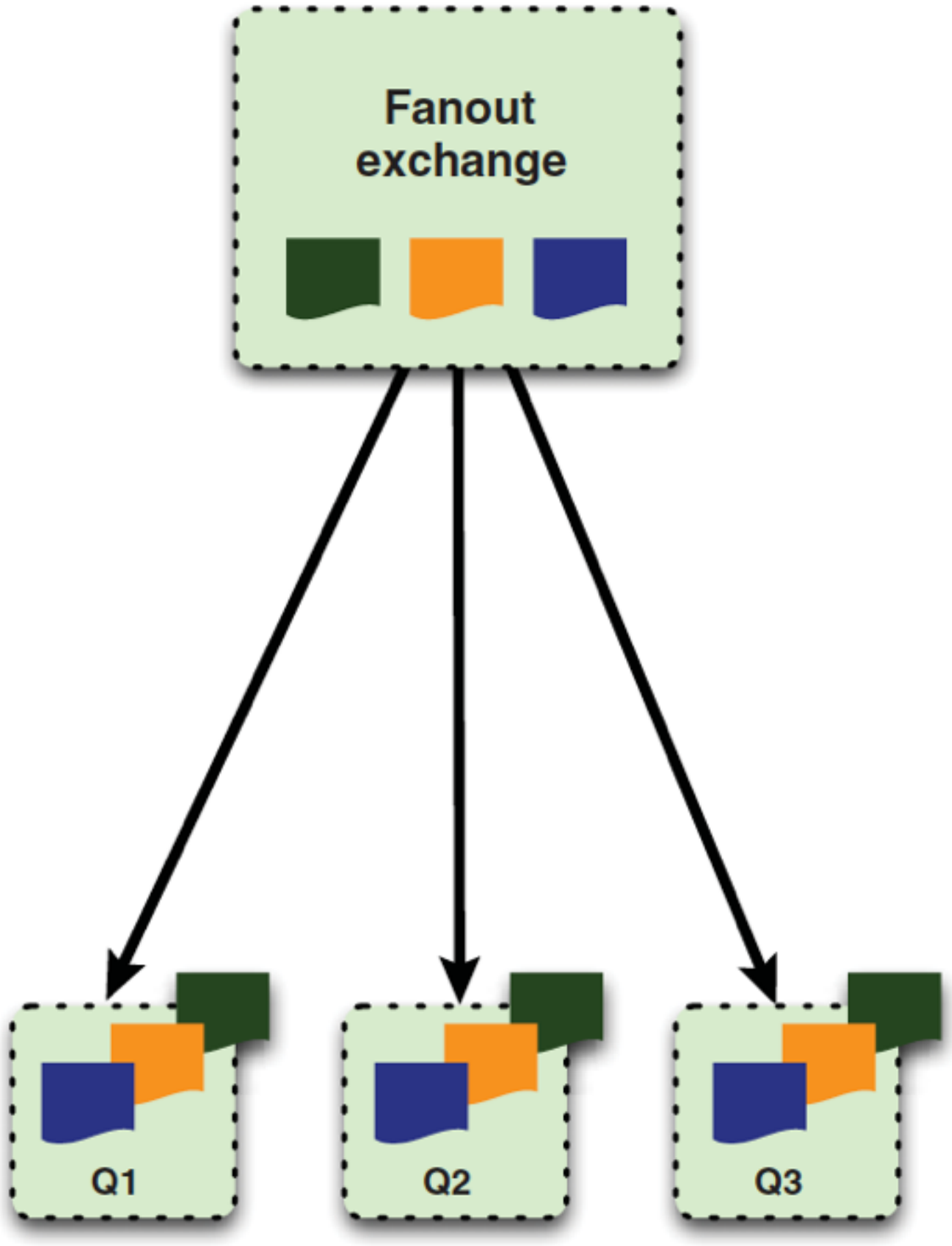


Figure 2.5 Fanout exchange message flow

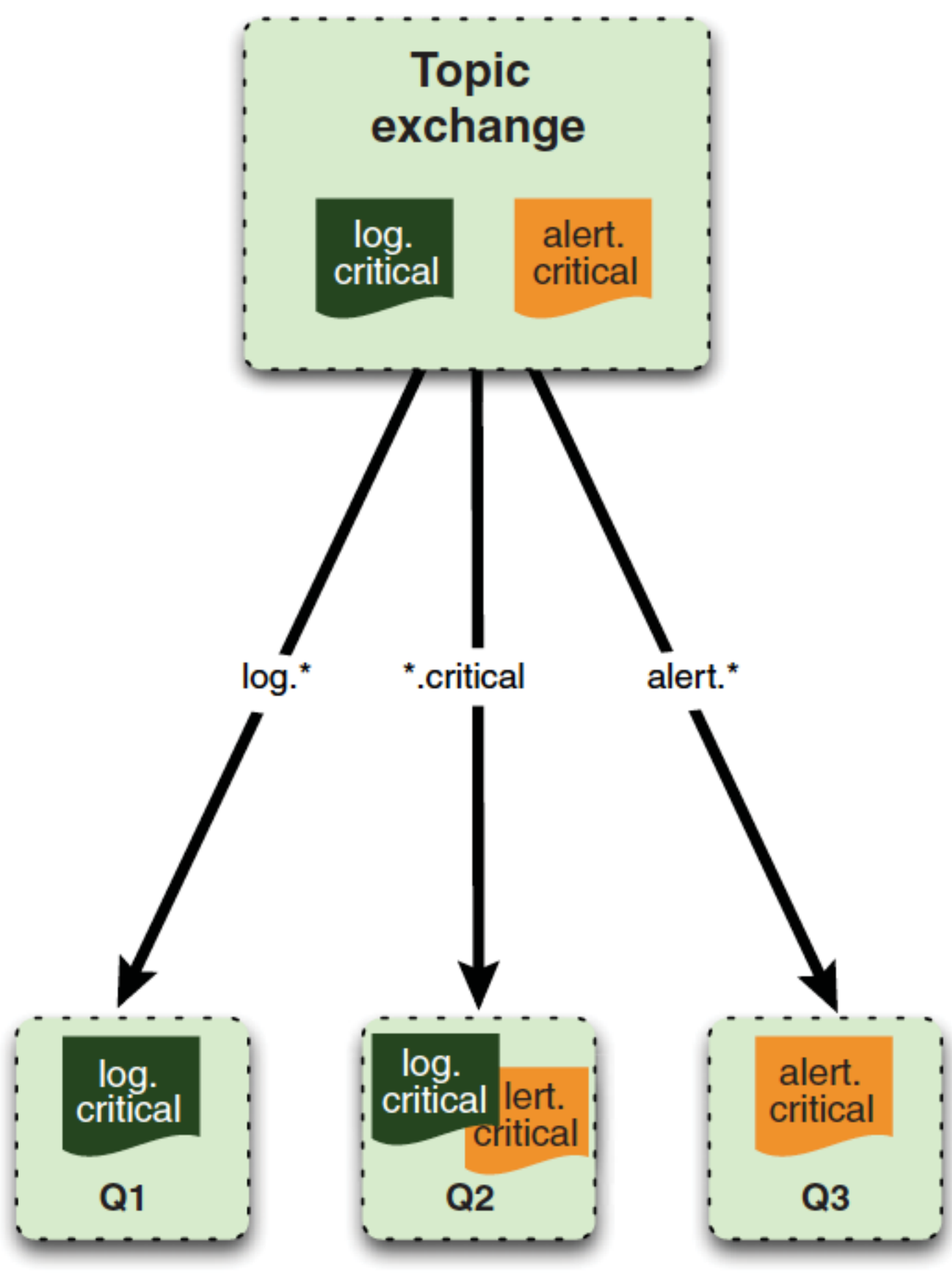
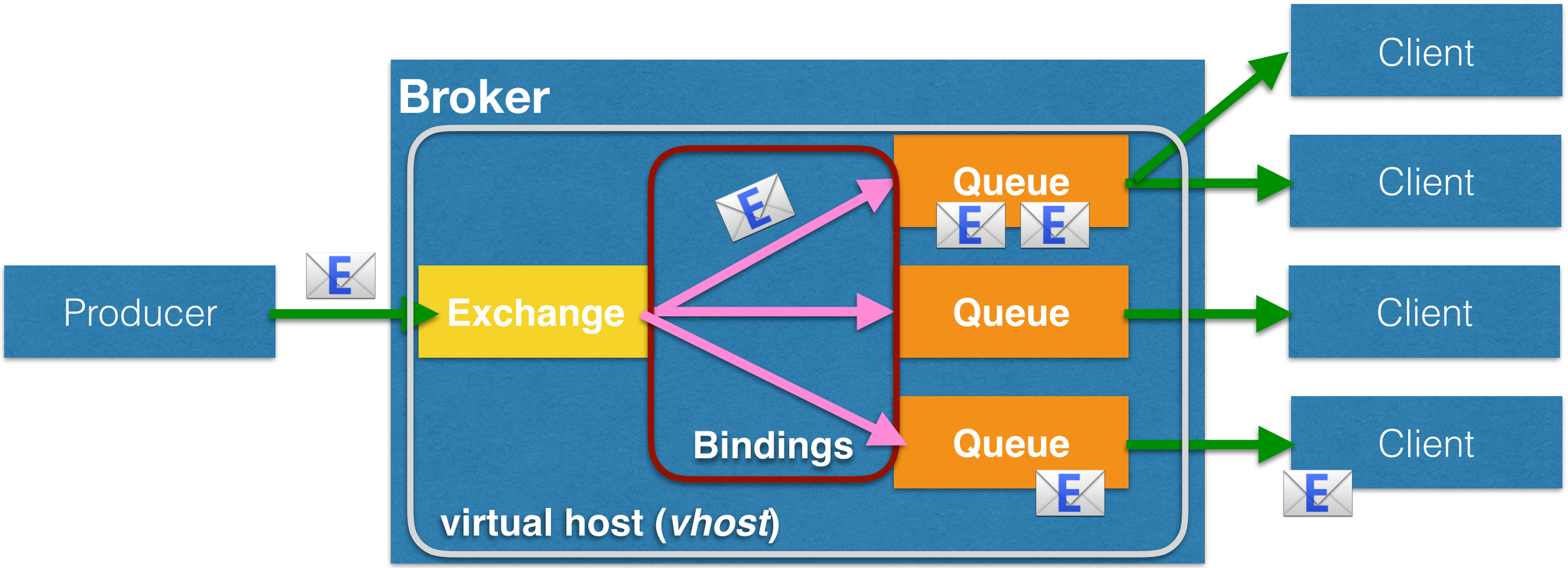


Figure 2.6 Topic exchange message flow

Entidades


Broker
Producer
E
Exchange
Bindings
Queue
Queue
Queue
Client
Client
Client
Client
virtual host (*vhost*)

vhosts

- Essencialmente, um *virtual broker*
- Isola a definição e execução de *exchanges*, *queues* e seus respectivos *bindings*
- Permite rodar diferentes grupos de funcionalidade comum em um mesmo *broker* sem riscos de colisões.
- Facilita controle de acesso por aplicações de usuário

Programa

- AMQP
- **RabbitMQ: Exemplos de Uso em JavaScript**
- *librabbitmq:* API C
- Sistemas competidores

RabbitMQ

- *RabbitMQ* é um broker AMQP
- Desenvolvido em *Erlang*
- Foco em tolerância a falhas, alto desempenho e *reliable messaging*
- Oferece algumas extensões ao protocolo (e.g. *producer confirms*, *message TTL*, *etc*)
- Outras features: *clustering, federation, highly available queues (mirroring)*

Exemplo: *Hello World*

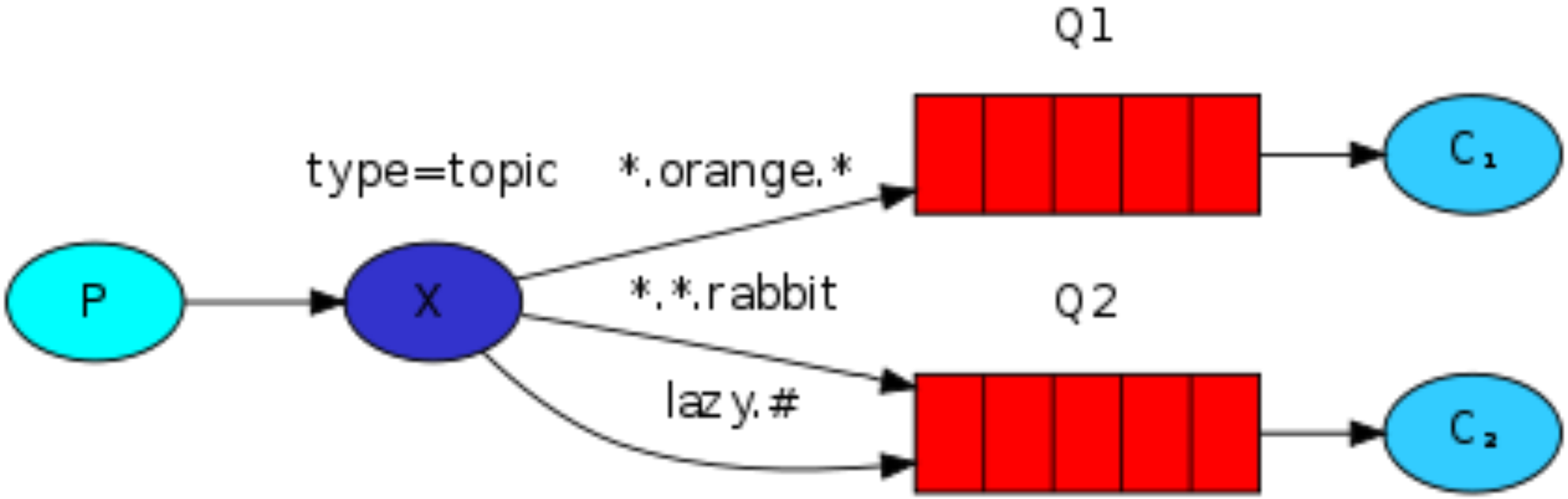
Producer

```
var amqp = require('amqplib/callback_api');

amqp.connect('amqp://localhost', function(err, conn) {
  conn.createChannel(function(err, ch) {
    var q = 'hello';

    ch.assertQueue(q, {durable: false});
    ch.sendToQueue(q, new Buffer('Hello World!'));

    console.log(" [x] Sent 'Hello World!'");
  });

  setTimeout(function() {
    conn.close();
    process.exit(0)
    }, 500);
});
```

Consumer

```
var amqp = require('amqplib/callback_api');

amqp.connect('amqp://localhost', function(err, conn) {
  conn.createChannel(function(err, ch) {
    var q = 'hello';

    ch.assertQueue(q, {durable: false});

    console.log(" [*] Waiting for messages in %s.", q);

    ch.consume(q, function(msg) {
      console.log(" [x] Received %s",
                  msg.content.toString());
    }, {noAck: true});
  });
});
```

Exemplo: *Topic Exchange*



Topic Exchange: Producer

```
var amqp = require('amqplib/callback_api');

amqp.connect('amqp://localhost', function(err, conn) {
  conn.createChannel(function(err, ch) {
    var exchange_name = 'topic_logs';
    var args = process.argv.slice(2);
    var key = (args.length > 0) ? args[0] : 'anonymous.info';
    var msg = args.slice(1).join(' ') || 'Hello World!';

    ch.assertExchange(exchange_name, 'topic', {durable: false});
    ch.publish(exchange_name, key, new Buffer(msg));

    console.log(" [x] Sent %s: '%s'", key, msg);
  });

  setTimeout(function() { conn.close(); process.exit(0) }, 500);

});
```

Topic Exchange: Consumer

```
/*. . .*/
amqp.connect('amqp://localhost', function(err, conn) {
  conn.createChannel(function(err, ch) {
    var ex = 'topic_logs';

    ch.assertExchange(ex, 'topic', {durable: false});
    ch.assertQueue('', {exclusive: true}, function(err, q) {
    console.log(' [*] Waiting for logs. To exit press CTRL+C');

    args.forEach(function(key) {
        ch.bindQueue(q.queue, ex, key);
    });

    ch.consume(q.queue, function(msg) {
        console.log(" [x] %s:'%s'", msg.fields.routingKey,
                                    msg.content.toString());
      }, {noAck: true});
    });
  });
});
```

Modelo para *Producer*

- Estabelecer uma conexão ao *broker (e.g. RabbitMQ)*
- Estabelecer um canal sob esta conexão
- Declarar um *exchange*
- Criar uma mensagem
- Publicar a mensagem (possivelmente com uma *routing key)*
- Por fim, fechar o canal e a conexão

Modelo para *Consumer*

- Estabelecer uma conexão ao *broker (e.g. RabbitMQ)*
- Estabelecer um canal sob esta conexão
- Declarar um *exchange*
- Definir um binding entre uma *queue* e um *exchange*
- Consumir as mensagens da *queue*
- Por fim, fechar o canal e a conexão

Programa

- AMQP
- RabbitMQ: Exemplos de Uso em JavaScript
- ***librabbitmq:*** **API C**
- Sistemas competidores

C API

- *It works!*

```
Carloss-MacBook-Pro:RabbitMQ calmattoso$ node send.js
  [x] Sent 'Hello World!'
```

```
Carloss-MacBook-Pro:build calmattoso$ ./examples/amqp_listen localhost 5672 amq.
direct test
Delivery 1, exchange  routingkey hello
----
00000000: 48 65 6C 6C 6F 20 57 6F : 72 6C 64 21              Hello World!
0000000C:
Delivery 2, exchange  routingkey hello
----
00000000: 48 65 6C 6C 6F 20 57 6F : 72 6C 64 21              Hello World!
0000000C:
        Delivery 3, exchange  routingkey hello
----
00000000: 48 65 6C 6C 6F 20 57 6F : 72 6C 64 21              Hello World!
0000000C:
```

C API: Consumer - Setup

```
conn = amqp_new_connection();

socket = amqp_tcp_socket_new(conn);
if (!socket) {
  die("creating TCP socket");
}
```

```
status = amqp_socket_open(socket, hostname, port);
if (status) {
  die("opening TCP socket");
}

die_on_amqp_error(amqp_login(conn, "/", 0, 131072, 0,
                             AMQP_SASL_METHOD_PLAIN,
                             "guest", “guest"), "Logging in");
amqp_channel_open(conn, 1);
die_on_amqp_error(amqp_get_rpc_reply(conn), "Opening channel");
```

C API: Consumer - Queue Setup

```
{
  amqp_queue_declare_ok_t *r = amqp_queue_declare(
                                  conn, 1, amqp_empty_bytes,
                                  0, 0, 0, 1, amqp_empty_table);
  die_on_amqp_error(amqp_get_rpc_reply(conn), "Declaring queue");
  queuename = amqp_bytes_malloc_dup(r->queue);
  if (queuename.bytes == NULL) {
    fprintf(stderr, "Out of memory while copying queue name");
    return 1;
  }
}

amqp_queue_bind(conn, 1, queuename, amqp_cstring_bytes(exchange),
                amqp_cstring_bytes(bindingkey), amqp_empty_table);
die_on_amqp_error(amqp_get_rpc_reply(conn), "Binding queue");

amqp_basic_consume(conn, 1, queuename,
                   amqp_empty_bytes, 0, 1, 0,
                   amqp_empty_table);
die_on_amqp_error(amqp_get_rpc_reply(conn), "Consuming");
```

C API: Consumer - Work

```
{
    while (1) {
      amqp_rpc_reply_t res;
      amqp_envelope_t envelope;
      amqp_maybe_release_buffers(conn);

      res = amqp_consume_message(conn, &envelope, NULL, 0);
      if (AMQP_RESPONSE_NORMAL != res.reply_type) {
        break;
      }

      printf("Delivery %u, exchange %.*s routingkey %.*s\n",
             (unsigned) envelope.delivery_tag,
             (int) envelope.exchange.len, (char *) envelope.exchange.bytes,
             (int) envelope.routing_key.len, (char *)envelope.routing_key.bytes);
      if (envelope.message.properties._flags & AMQP_BASIC_CONTENT_TYPE_FLAG) {
          printf("Content-type: %.*s\n",
                 (int) envelope.message.properties.content_type.len,
                 (char *) envelope.message.properties.content_type.bytes);
      }
      printf("----\n");

      amqp_dump(envelope.message.body.bytes, envelope.message.body.len);
      amqp_destroy_envelope(&envelope);
    }
}
  /* continua */
```

C API: Consumer - Cleanup

```
  die_on_amqp_error(amqp_channel_close(conn, 1, AMQP_REPLY_SUCCESS),
                    "Closing channel");
  die_on_amqp_error(amqp_connection_close(conn, AMQP_REPLY_SUCCESS),
                    "Closing connection");

  die_on_error(amqp_destroy_connection(conn), "Ending connection");

  return 0;

}
```

Programa

- AMQP
- RabbitMQ: Exemplos de Uso em JavaScript
- *librabbitmq:* API C
- **Sistemas competidores**

SQS

- **S**imple **Q**ueueing **S**ystem: sistema distribuído de filas gerenciado com promessas de *“fast, reliable, scalable"*

- É realmente simples: *message queueing* sem *brokers*, *exchanges*, *bindings*. Apenas *queues.*

- Essencialmente *send message, get message* e *delete message.*

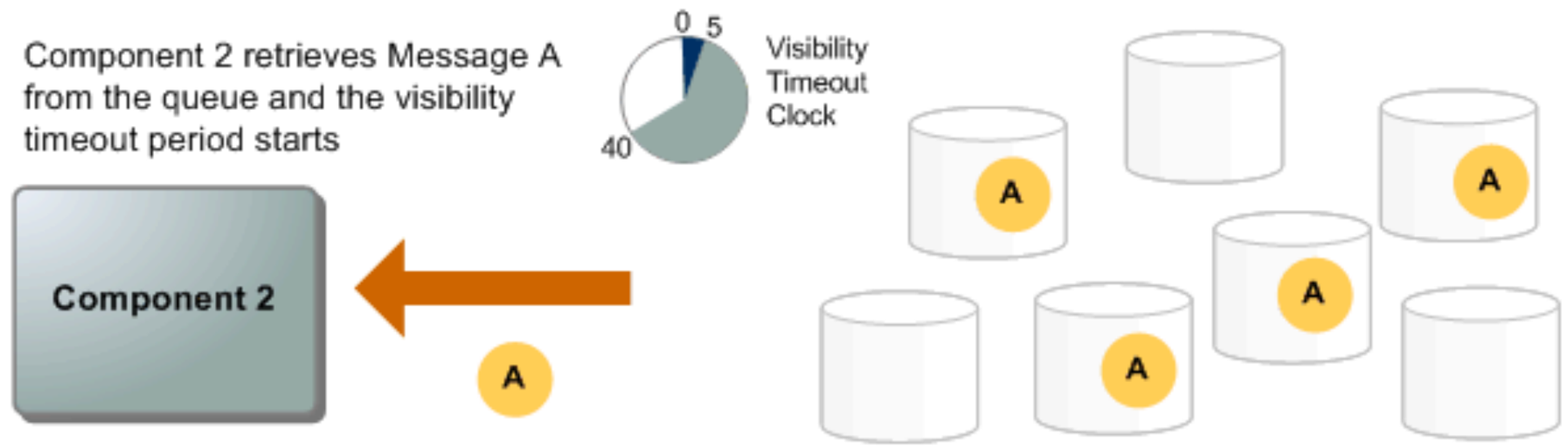
- Não apresenta funcionalidades como *topic filtering* (SNS pode ser usado para isto)

1

Component 1 sends Message A to the queue

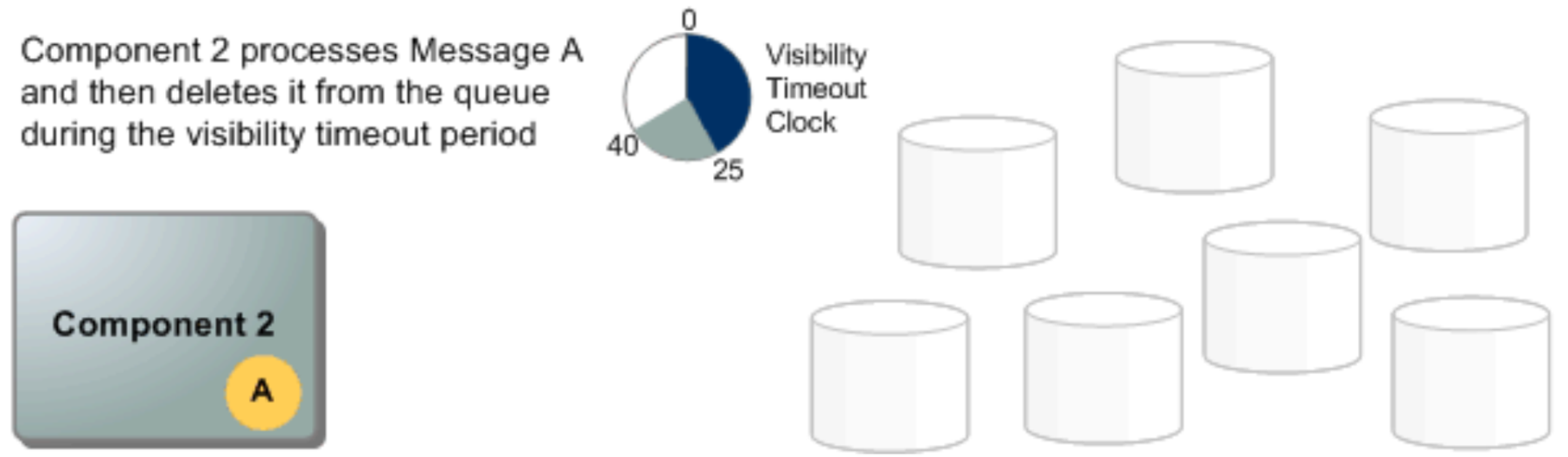
2

Component 2 retrieves Message A from the queue and the visibility timeout period starts



3

Component 2 processes Message A and then deletes it from the queue during the visibility timeout period



SQS

Send message

```
System.out.println("Sending a message to MyQueue.\n");
sqs.sendMessage(new SendMessageRequest()
    .withQueueUrl(myQueueUrl)
    .withMessageBody("This is my message text.”))
```

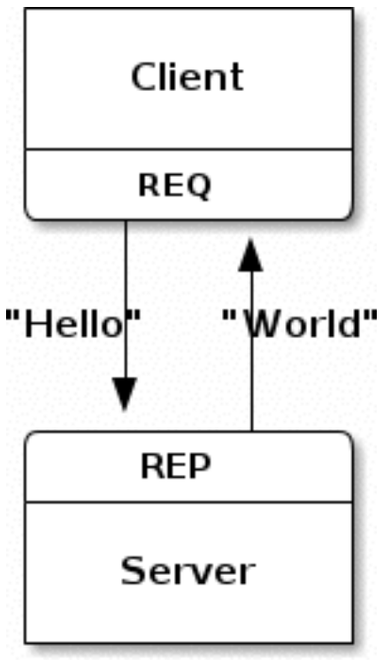
Receive message

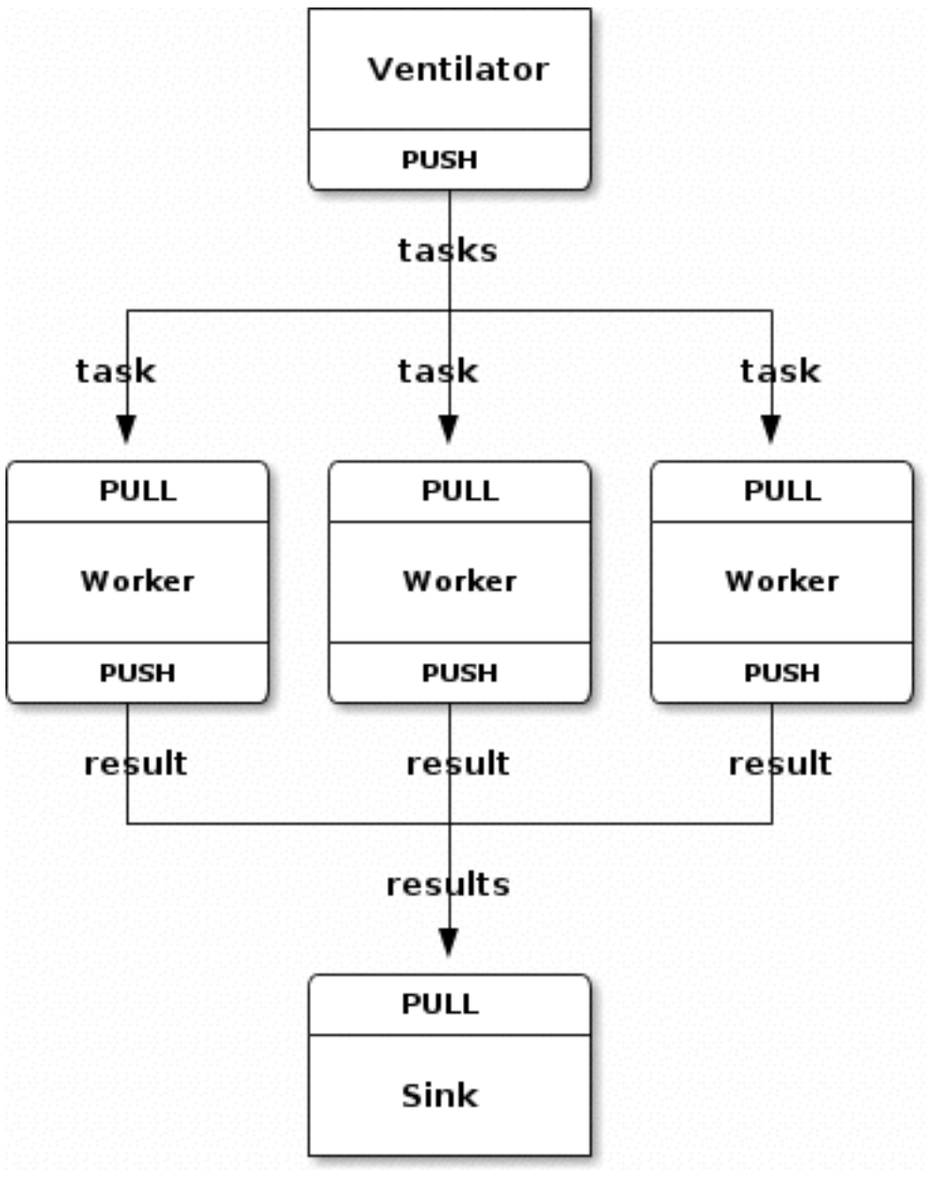
```
System.out.println("Receiving messages from MyQueue.\n");
ReceiveMessageRequest receiveMessageRequest = new ReceiveMessageRequest(myQueueUrl);
List<Message> messages = sqs.receiveMessage(receiveMessageRequest).getMessages();
for (Message message : messages) {
    System.out.println("  Message");
    System.out.println("    MessageId:     " + message.getMessageId());
    System.out.println("    ReceiptHandle: " + message.getReceiptHandle());
    System.out.println("    MD5OfBody:     " + message.getMD5OfBody());
    System.out.println("    Body:          " + message.getBody());
    for (Entry<String, String> entry : message.getAttributes().entrySet()) {
        System.out.println("  Attribute");
        System.out.println("    Name:  " + entry.getKey());
        System.out.println("    Value: " + entry.getValue());
    }
}
System.out.println();
```

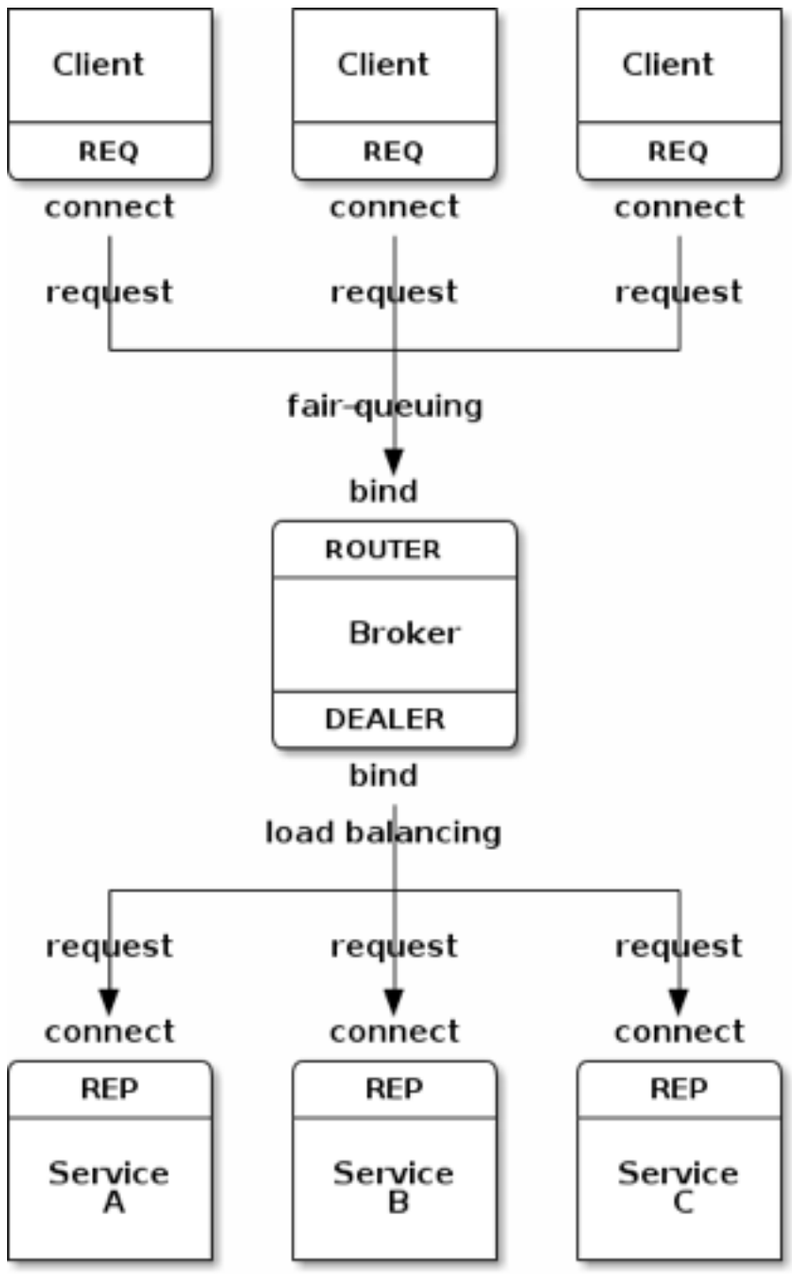
0mq & nanomsg

- Ambos atuam como *messaging middleware*
- *nanomsg* é visto como um *mini-0mq*
- Ambos são “apenas *sockets”!* Sem *brokers* ou *exchanges.*
- Suas implementações de socket *"present an abstraction of an asynchronous message queue”* e dependem do padrão de comunicação adotado
- Operam sobre mais tipos de transporte: IPC, Inproc, TCP, …

0mq & nanomsg


Client
REQ
"Hello"
"World"
REP
Server


Ventilator
PUSH
tasks
task
task
task
PULL
Worker
PUSH
PULL
Worker
PUSH
PULL
Worker
PUSH
result
result
result
results
PULL
Sink


Client
REQ
Client
REQ
Client
REQ
connect
connect
connect
request
request
request
fair-queuing
bind
ROUTER
Broker
DEALER
bind
load balancing
request
request
request
connect
connect
connect
REP
Service A
REP
Service B
REP
Service C

0mq: client <-> server

Server

```
# ZeroMQ Context
context = zmq.Context()

# Define the socket using the "Context"
sock = context.socket(zmq.REP)
sock.bind("tcp://127.0.0.1:5678")

# Run a simple "Echo" server
while True:
    message = sock.recv()
    sock.send("Echo: " + message)
    print "Echo: " + message
```

Client

```
# ZeroMQ Context
context = zmq.Context()

# Define the socket using the "Context"
sock = context.socket(zmq.REQ)
sock.connect("tcp://127.0.0.1:5678")

# Send a "message" using the socket
sock.send(" ".join(sys.argv[1:]))
print sock.recv()
```

Próximos Passos

- Estudo de OMF
- API de RabbitMQ em Céu
- Documento preliminar de propostas

Referências

- RabbitMQ e AMQP: http://www.rabbitmq.com/
- RabbitMQ in Action: https://www.manning.com/books/rabbitmq-in-action
- C API: https://github.com/alanxz/rabbitmq-c
- AMQP Spec: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf
- SQS: https://aws.amazon.com/sqs/
- ØMQ: http://ruudud.github.io/presentations/zeromq/ & http://zguide.zeromq.org/page:all
- nanomsg: http://nanomsg.org/documentation.html