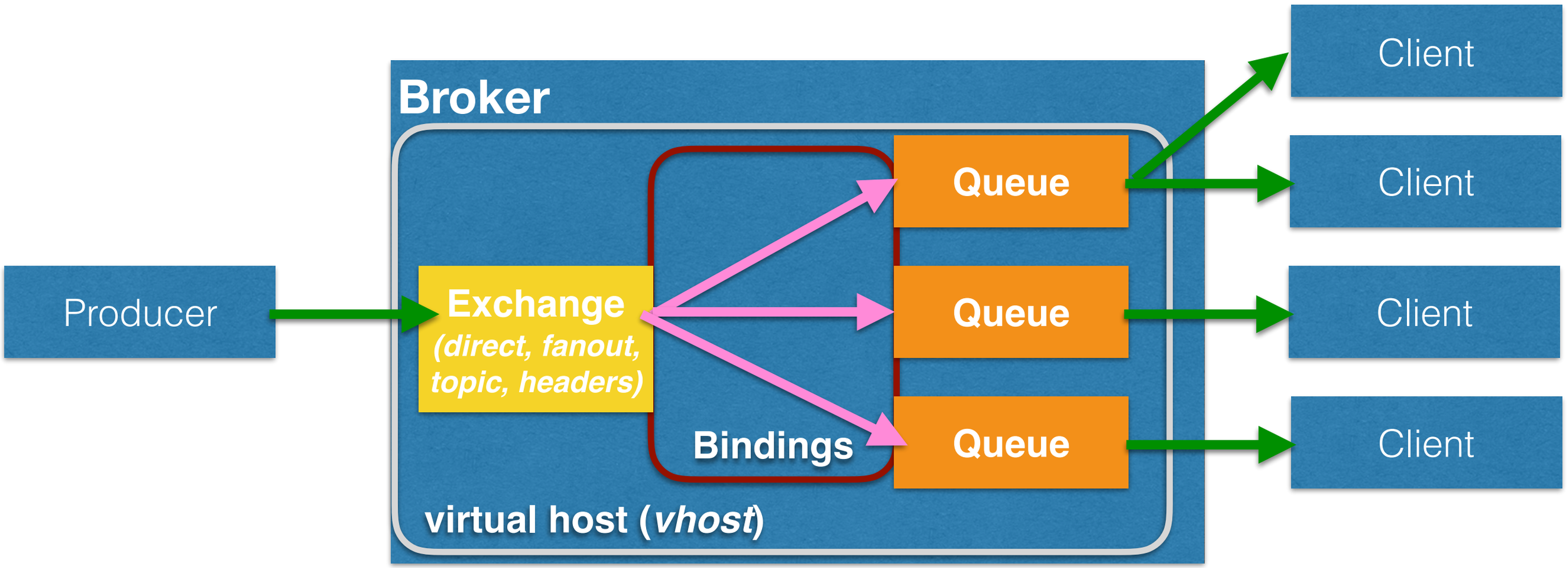
RabbitMQ/AMQP & OMF

Carlos Mattoso

25 de Fevereiro de 2016

Entidades


Broker
Queue
Client
Client
Producer
Exchange
(direct, fanout,
topic, headers)
Queue
Client
Bindings
Queue
Client
virtual host (vhost)

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!'));

    // prints 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: *Hello World*

Producer

```
#!/usr/bin/env node

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

function sleep(time, callback) {
    // …
}

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

     // Server gets killed...
    sleep(10000, function() {
      ch.assertQueue(q, {durable: false});
      ch.sendToQueue(q, new Buffer('Hello World!'));

       // Hello world is printed
      console.log(" [x] Sent 'Hello World!'");
    });
  });
});
```

C API

- Síncrona.
- Utiliza *non-blocking sockets* (nas últimas 2 versões, *mid2015*)
- Duas “classes” de métodos da API
  - *blocking* sem controle do usuário (métodos AMQP)
  - *blocking* controlado pelo usuário (e.g. conexão e recebimento de mensagens)
- Ambos limitados por algum *timeout*
- *select/poll* usados por trás dos panos

C API: Métodos AMQP

- *amqp_exchange_declare_ok_t * AMQP_CALL* ***amqp_exchange_declare****(…);* <-> **exchange.declare**

- *amqp_queue_declare_ok_t * AMQP_CALL* ***amqp_queue_declare****(…);* <-> **queue.declare**

- *amqp_queue_bind_ok_t * AMQP_CALL* ***amqp_queue_bind****(…);* <-> **queue.bind**

- *amqp_basic_consume_ok_t * AMQP_CALL* ***amqp_basic_consume****(…);* <-> **basic.consume**

- e demais métodos AMQP são *blocking* invisíveis (timeout: *heartbeat)*

C API: Arquitetura RPC

- Métodos dependem da função ***amqp_simple_rpc***
- Consiste em um *send* (*non-blocking*) com *timeout == hearbeat*
  - `status = amqp_send_method(state, channel, request_id, decoded_request_method);`
  - `send_with_timeout`
- Posteriormente um *receive* com *timeout == heartbeat*
  - `status = wait_frame_inner(state, &frame, NULL);`
  - `=> amqp_try_send`

C API: Arquitetura RPC

- Algumas funções permitem ao usuário definir o *timeout*
  - *int AMQP_CALL* ***amqp_socket_open_noblock****(…, struct timeval **timeout**);*
  - *amqp_rpc_reply_t AMQP_CALL* ***amqp_consume_message****(…, struct timeval **timeout**, …);*
    - *=>* ***amqp_simple_wait_frame_noblock***

C API: Arquitetura RPC

Nos próximos slides funções da biblioteca em **azul** são *non-blocking*, em **vermelho** são *blocking* por *heartbeat* e em **laranja** aceitam um *user-defined timeout*.

C API: Consumer - Setup

```
conn = amqp_new_connection();

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

```
status = amqp_socket_open_noblock(socket, hostname, port, NULL);
// NULL timeout, então bloqueia
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");
```

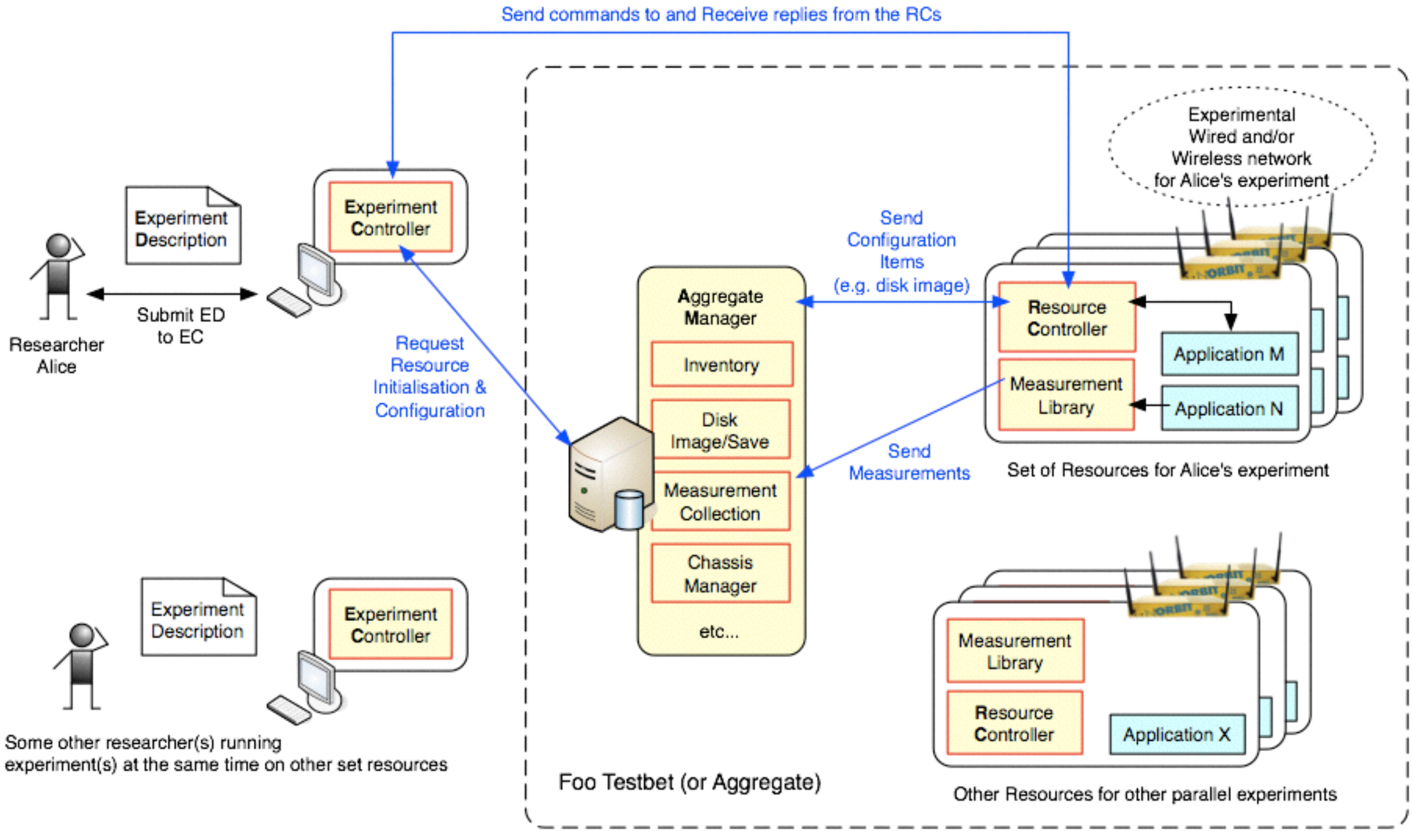
C API: Consumer - Work

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

      res = amqp_consume_message(conn, &envelope, NULL, 0); // blocking pq timeout == NULL
      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 */
```

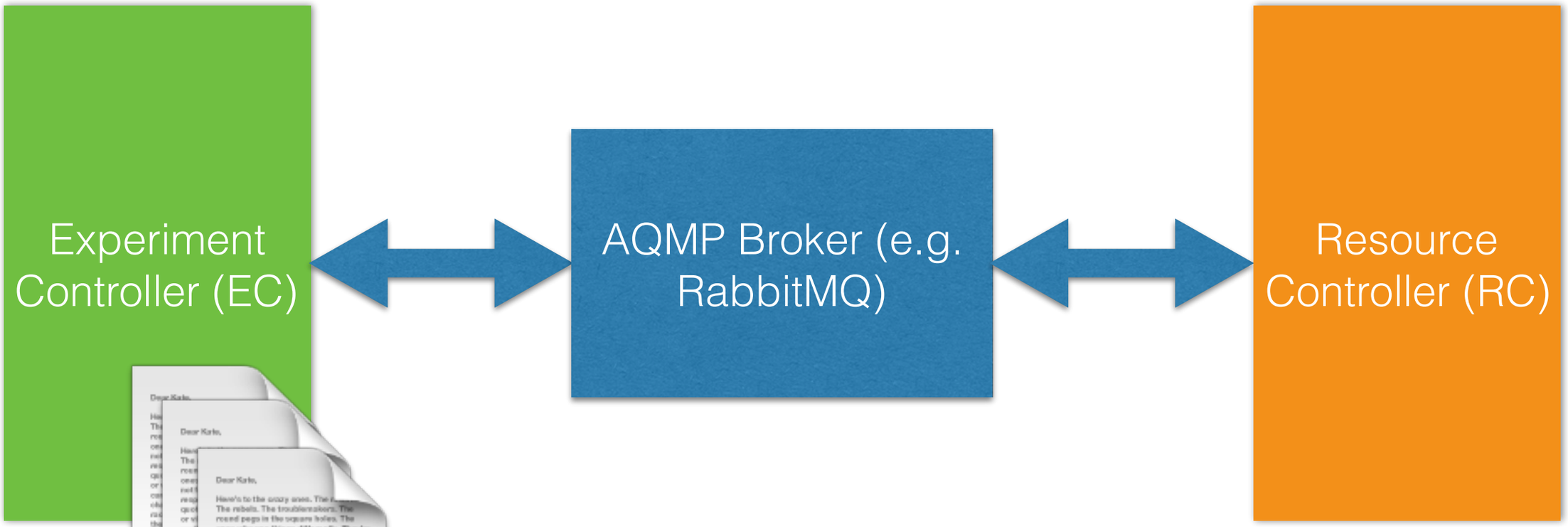
C API: Consumer - Cleanup

```
  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;

}
```

OMF


Send commands to and Receive replies from the RCs
Experiment Description
Experiment Controller
Researcher Alice
Submit ED to EC
Request Resource Initialisation & Configuration
Send Configuration Items (e.g. disk image)
Aggregate Manager
Inventory
Disk Image/Save
Measurement Collection
Chassis Manager
etc...
Send Measurements
Experimental Wired and/or Wireless network for Alice's experiment
Resource Controller
Application M
Measurement Library
Application N
Set of Resources for Alice's experiment
Experiment Description
Experiment Controller
Some other researcher(s) running experiment(s) at the same time on other set resources
Measurement Library
Resource Controller
Application X
Other Resources for other parallel experiments
Foo Testbet (or Aggregate)

OMF: Mais simples

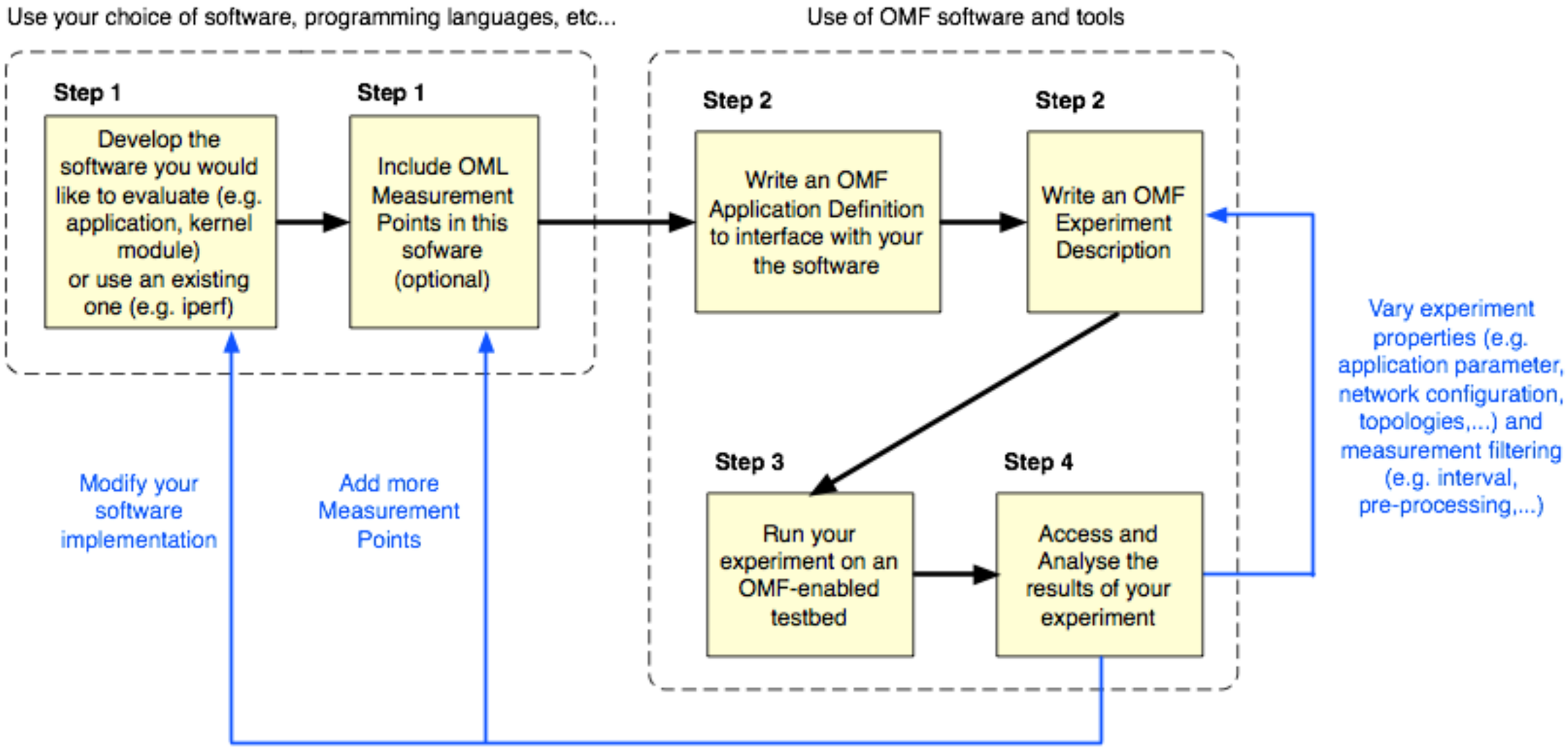


Experiment Description (em OEDL)

OMF: Desenvolvimento

1. Desenvolvimento da aplicação
2. Desenvolvimento do experimento (ED)
3. Execução do experimento (EC)
4. Avaliação de resultados (OML)

OMF


Use your choice of software, programming languages, etc...
Use of OMF software and tools
Step 1
Develop the software you would like to evaluate (e.g. application, kernel module) or use an existing one (e.g. iperf)
Step 1
Include OML Measurement Points in this sofware (optional)
Step 2
Write an OMF Application Definition to interface with your the software
Step 2
Write an OMF Experiment Description
Step 3
Run your experiment on an OMF-enabled testbed
Step 4
Access and Analyse the results of your experiment
Vary experiment properties (e.g. application parameter, network configuration, topologies,...) and measurement filtering (e.g. interval, pre-processing,...)
Modify your software implementation
Add more Measurement Points

OEDL

- Linguagem baseada em *Ruby* com métodos próprios para se orquestrar experimentos

- Permite descrever e configurar os **recursos** de um experimento

- Permite descrever as **aplicações** e métricas a serem coletadas durante sua execução

- Permite reagir a **eventos**, executando alguma tarefa

OEDL: aplicações

```
defApplication('vstream') do |app|
  app.description = “An app definition for a video stream"
  app.binary_path = “/usb/local/bin/vstream”

  # Definição de parâmetros que o app pode/deve receber
  app.defProperty('target', 'Address to stream from', nil, {
    :type => :string,
    :mandatory => true,
    :default => ‘localhost'
  })
  app.defProperty('fps', 'Frames per second', '-fps',
                  {:type => :integer})
  # Continua

  app.defMeasurement('probe_statistic') do |m|
    m.defMetric('dest_addr', :string)
    m.defMetric('ttl', :uint32)
    m.defMetric('rtt', :double)
    m.defMetric('rtt_unit', :string)
  end

  app.defMeasurement('video_stream_statistic') do |m|
    m.defMetric('frame_number', :uint64)
    m.defMetric('drop_rate', :uint32)
    m.defMetric('codec_name', :string)
    m.defMetric('bitrate', :unit32)
  end
end
```

OEDL: aplicações

```
# Métricas disponibilizadas através de OML
app.defMeasurement('probe_statistic') do |m|
  m.defMetric('dest_addr', :string)
  m.defMetric('ttl', :uint32)
  m.defMetric('rtt', :double)
  m.defMetric('rtt_unit', :string)
end

app.defMeasurement('video_stream_statistic') do |m|
  m.defMetric('frame_number', :uint64)
  m.defMetric('drop_rate', :uint32)
  m.defMetric('codec_name', :string)
  m.defMetric('bitrate', :unit32)
end
end
```

OEDL: grupos

```
defGroup('Sender', "omf.nicta.node2") do |node|
  node.addApplication("test:app:otg2") do |app|
    app.setProperty('udp:local_host', '192.168.0.2')
    app.setProperty('udp:dst_host', '192.168.0.3')
    app.setProperty('udp:dst_port', 3000)

    # Obter métrica em intervalos de 3s
    app.measure('udp_out', :interval => 3)
  end
  node.net.w0.mode = "adhoc"
  node.net.w0.type = 'g'
  node.net.w0.channel = "6"
  node.net.w0.essid = "helloworld"
  node.net.w0.ip = "192.168.0.2"
end
```

OEDL: eventos

```
onEvent(:ALL_UP_AND_INSTALLED) do |event|
  info "This is my first OMF experiment"
  wait 10
  allGroups.startApplications
  info "All my Applications are started now..."
  wait 30
  allGroups.stopApplications
  info "All my Applications are stopped now."
  Experiment.done
end
```

Eventos básicos:

:ALL_NODES_UP*, *:ALL_UP_AND_INSTALLED*, *:ALL_APPS_DONE*, *:ALL_INTERFACE_UP*, *:ALL_UP

OEDL: eventos

```
defEvent :APP_EXITED do |state|
  triggered = false
  state.each do |resource|
      triggered = true if
        (resource.type == 'application') &&
          (resource.state == 'stopped')
  end
  triggered
end

onEvent :APP_EXITED, consume_event= false do
    info "An application has just finished... should
we do something about it?"
end
```

OEDL: ED Simples (Bruno/Miguel)

```
# Simple OEDL Experiment for OMF
# displays the hostname and date/time of the remote RC

defProperty('res1', “node-name“, "ID of a node")
defGroup('Actor', property.res1)

onEvent(:ALL_UP) do |event|
  # 'after' is not blocking. This executes 3 seconds after :ALL_UP fired.
  after 3 do
    info "TEST - allGroups"
    allGroups.exec("/bin/date")
  end
  # 'after' is not blocking. This executes 6 seconds after :ALL_UP fired.
  after 6 do
    info "TEST - group"
    group("Actor").exec("/bin/hostname -A")
  end
  # 'after' is not blocking. This executes 9 seconds after :ALL_UP fired.
  after 9 do
    Experiment.done
  end
end
```

Troca de mensagens (*FRCP*)

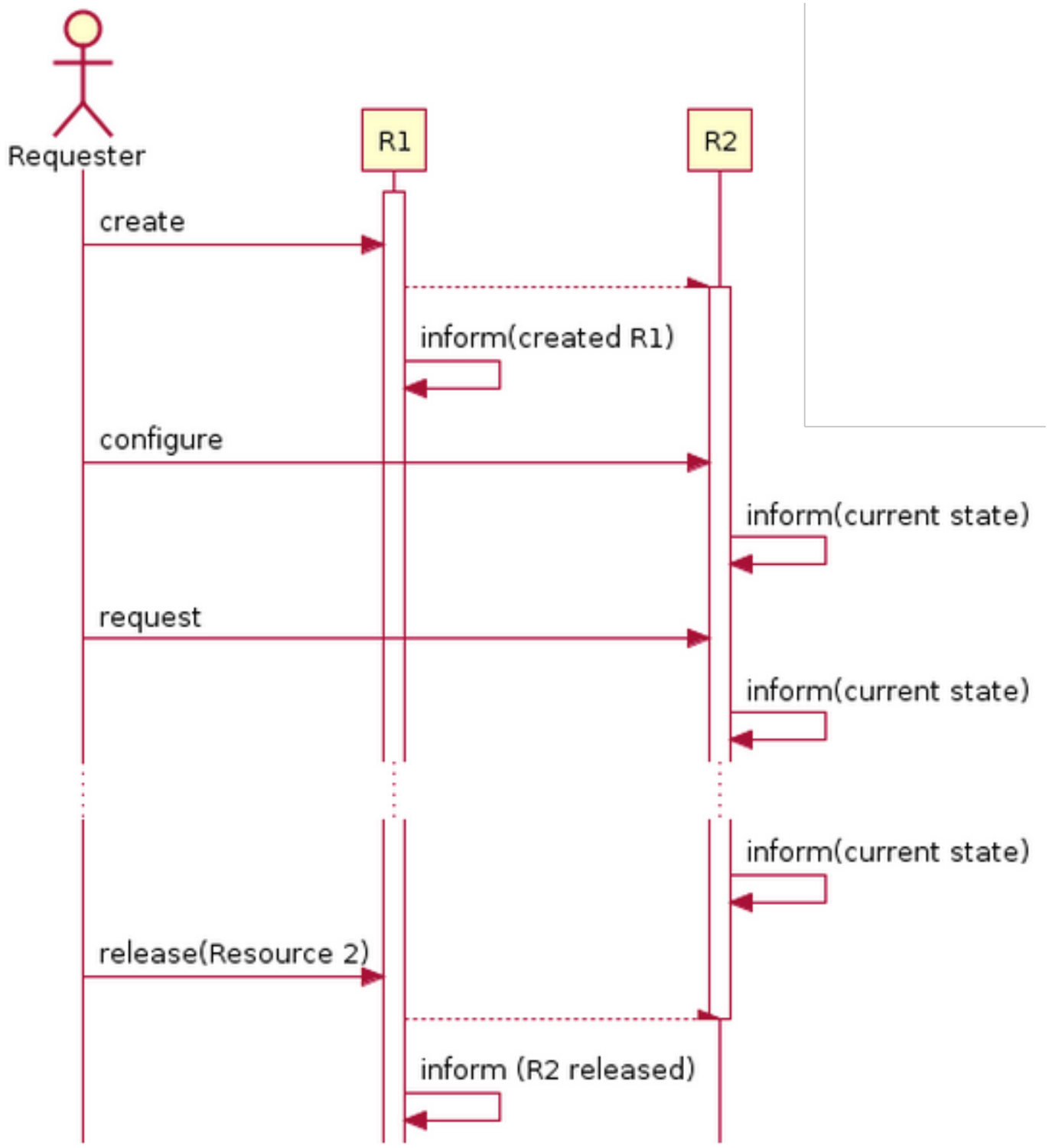
create

inform

configure

request

release



OEDL: ED Simples (Bruno/Miguel)

Setup

1. Cria um exchange para o *RC*

OEDL: ED Simples (Bruno/Miguel)

```
# Simple OEDL Experiment for OMF
# displays the hostname and date/time of the remote RC

defProperty('res1', “node-name“, "ID of a node")
defGroup('Actor', property.res1)

onEvent(:ALL_UP) do |event|
  # 'after' is not blocking. This executes 3 seconds after :ALL_UP fired.
  after 3 do
    info "TEST - allGroups"
    allGroups.exec("/bin/date")
  end
  # 'after' is not blocking. This executes 6 seconds after :ALL_UP fired.
  after 6 do
    info "TEST - group"
    group("Actor").exec("/bin/hostname -A")
  end
  # 'after' is not blocking. This executes 9 seconds after :ALL_UP fired.
  after 9 do
    Experiment.done
  end
end
```

OEDL: ED Simples (Bruno/Miguel)

Criação do experimento

1. Recebe uma mensagem do tipo *configure*
2. Inscreve-se no *exchange* (OMF usa o termo *topic*) do experimento
3. Notifica (mensagem *inform*) a inscrição no tópico do experimento e do *RC*

OEDL: ED Simples (Bruno/Miguel)

Criação da aplicação

1. Recebe uma mensagem do tipo *create*

2. Inscreve-se no *tópico* da aplicação (definido pelo EC na mensagem)

3. Notifica (*inform)* a criação através do tópico do experimento e do RC

4. Notifica (*inform)* a inscrição no tópico da aplicação e do RC

OEDL: ED Simples (Bruno/Miguel)

Execução da aplicação

1. Recebe uma mensagem do tipo *configure*

2. Publica no tópico da aplicação que ela foi criada e iniciada (*inform)*

3. A cada evento, notifica o EC através do tópico da aplicação (e.g. *APP_EVENT::STDOUT* quando a aplicação exibe alguma informação)

4. Notifica o término da aplicação (*APP_EVENT::EXIT)*

OEDL: ED Simples (Bruno/Miguel)

Fim do experimento

1. Recebe uma mensagem do tipo *configure*
2. Indica no tópico do RC o fim da inscrição neste
3. Finaliza todos os tópicos

Produtos

- API de *AMQP* em Céu (usando *rabbitmq-c*)
- *Experiment Controller* (*omf_ec.rb)* em Céu?
  - Aceitaria *EDs* em Céu (?)
- Extensão ao *testbed*?