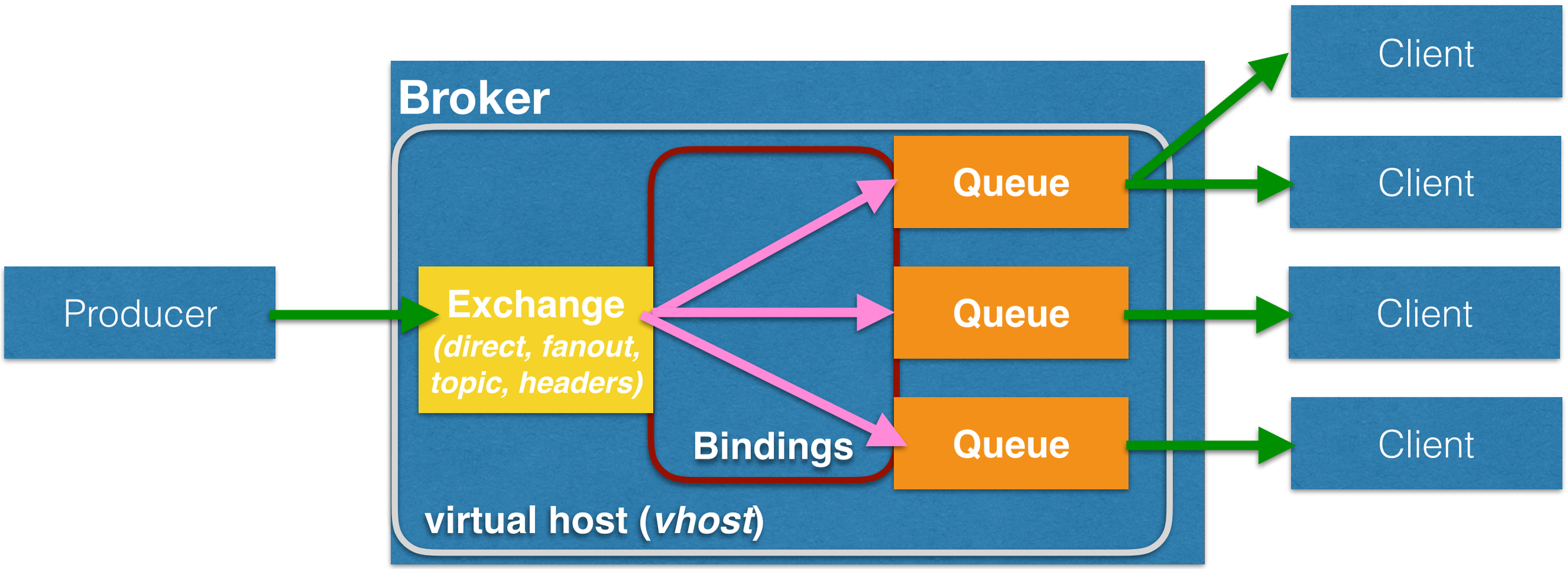
ceu-rabbitmq & *ceu-omf-ec*

Resumo de Maio e Visão Geral de omf_ec

amqp: *overview*



céu-rabbitmq

- entidades
- metodologia
- publisher/consumer

entidades

- *connection e channel*
- *exchanges*
- *queues*
- *bindings*
- ações: *publish* e *consume*

entidades

- certo *boilerplate*

```
var Exchange amq_direct with
  this.channel = &ch;
  this.name = []."amq.direct";
  this.type = []."direct";
  this.passive = false;
  this.durable = true;
  this.auto_delete = false;
  this.internal = false;
  this.arguments = _amqp_empty_table;
end;
```

- oferecer "organismos" pré-definidos
  - exemplo: *exchanges* padrões *(amq.direct, amq.topic, etc)*

metodologia

- programação estruturada
- ciclo de vida das entidades limitado ao da aplicação
- céu *over* amqp: paradigma céu deve prevalecer

exemplo

```
var Connection c with
  // setup…
end;

watching c do
 var Channel ch with
  this.conn = &c;
 end;
  watching ch do
    var Exchange amq_direct with
     // setup…
    end;

    watching amq_direct do
        do Publish with
        // setup…
        end;
       // outro código…
      end
    end
  end
end
```

programação estruturada

```
var Connection c with
   // setup…
end;

watching c do
 var Channel ch with
   this.conn = &c;
   end;

   watching ch do
     par/or do
      // função amqp
     with
      // outro código… mata acima se necessário
     end
   end
end
```

programação estruturada

connection

channel

queue 1

queue 2

. . .

queue *n*

programação estruturada

```
connection
    channel

      queue

outro código…
```

programação estruturada

- evitar *flags* de estado ou diretivas explícitas para definição de estado

- encerramento da execução por terminação de *trails*

- exemplo: colocar canal em modo consumo
  - cria-se organismo de canal apenas para consumo
  - quando não se precisa consumir, mata a trail do canal
  - definição de estado implícita

revendo entidades

- entidades tem caráter efêmero
- existem enquanto da duração de suas *trails*
- ao término das *trails*, são removidas no servidor
- preferência de prog. estruturada ao invés de flags de estado (ex: *auto_delete)*

recebimento de mensagens

- ao invés de uma fila, impera o paradigma de programação estruturada

```
var Queue q with
  // …
end

watching q do
  loop do
    _amqp_envelope_t msg = await q.receive;
    spawn Handler with
      this.message = msg;
    end
  end
end
```

céu-omf-ec

- práticas em *Ruby*
- *FRCP* em *Ruby*
- *groups* e *applications*
- e em *céu*?

práticas em *Ruby*

- basicamente, programação orientada a objetos
- módulos base com especializações(ex: *amqp* vs *xmpp)*
- implementou-se uma *TimeMachine* para se lidar com evento (vs nativo em *Céu)*
- "pontos de entrada" do ambiente: *defGroups* e *defApplication*

frcp

- não há um módulo *frcp*
- basicamente, as mensagens são criadas conforme necessário
- as mensagens simplesmente encapsulam o estado e tem algumas ações úteis (ex: *marshalling)*

frcp

- classe principal: *topic.rb*
- cria mensagens dos diferentes tipos *(create, configure, request, release, informa)*
- especialização relevante: *amqp_topic.rb*
- por herança define o código específico de envio de mensagem

frcp

mensagem

frcp

frcp

mensagem

comunicador

comunicador *amqp*

OU

comunicador *xmpp*

groups e *applications*

- no caso de *applications*, duas: *def_app_context* (para definição) e *app_context.rb* (para instanciação)

- uma mesma *app* pode ter várias instanciações distintas em um mesmo grupo -> contextos distintos

- obs: inserir um grupo em um grupo causa um *merge* dos grupos

modelo *omf_ec Ruby*

```
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
```

modelo *omf_ec Ruby*

```
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
```

modelo *omf_ec Ruby*

- nos casos acima, apenas definições, nenhuma ação

- há uma maneira melhor?

modelo *omf_ec Ruby*

```
onEvent(:ALL_UP_AND_INSTALLED) do |event|
  # Print some information message
  info "This is my first OMF experiment"
  # Start all the Applications associated to all the Groups
  allGroups.startApplications
  # After 5 sec
  after 5 do
    # Stop all the Applications associated to all the Groups
    allGroups.stopApplications
    # Tell the Experiment Controller to terminate the experiment now
    Experiment.done
  end
end
```

modelo *omf_ec Ruby*

- ação se executa sobre os estados previamente definidos -> POO

- e se fosse invertida a forma de pensar?

- por que não encapsular o controle de eventos nas próprias entidades (grupos e aplicações)? -> limitação Ruby

céu comes to save the day!

exemplo (ideia)

```
class Group g
  // interface…
with
  par do
    await ALL_UP;
    var AppA with
      // instanciação
    end;
    // <…>
  with
    await USER_DEFINED_EVENT;
    var AppB with
      // instanciação
    end;
    // <…>
  end
end
```

modelo *céu*

- organismos de aplicações e grupos definidos pelo usuário
- aplicações instanciadas dentro dos grupos no momento em que sua execução é necessária
- aplicações também reagem a eventos
- impera programação estrutura
- experimentador pode ser criativo e fazer composições interessantes
- pontos de entrada do ambiente?