embarcadero®

embarcadero®

Arquiteturas distribuídas baseadas em mensageria com RabbitMQ

Felipe Caputo

Embarcadero Conference

Quem sou eu?

Felipe Caputo

Engenheiro de Software / Softplan

felipecaputo

caputoDev

felipewcaputo

O que são sistemas distribuídos?

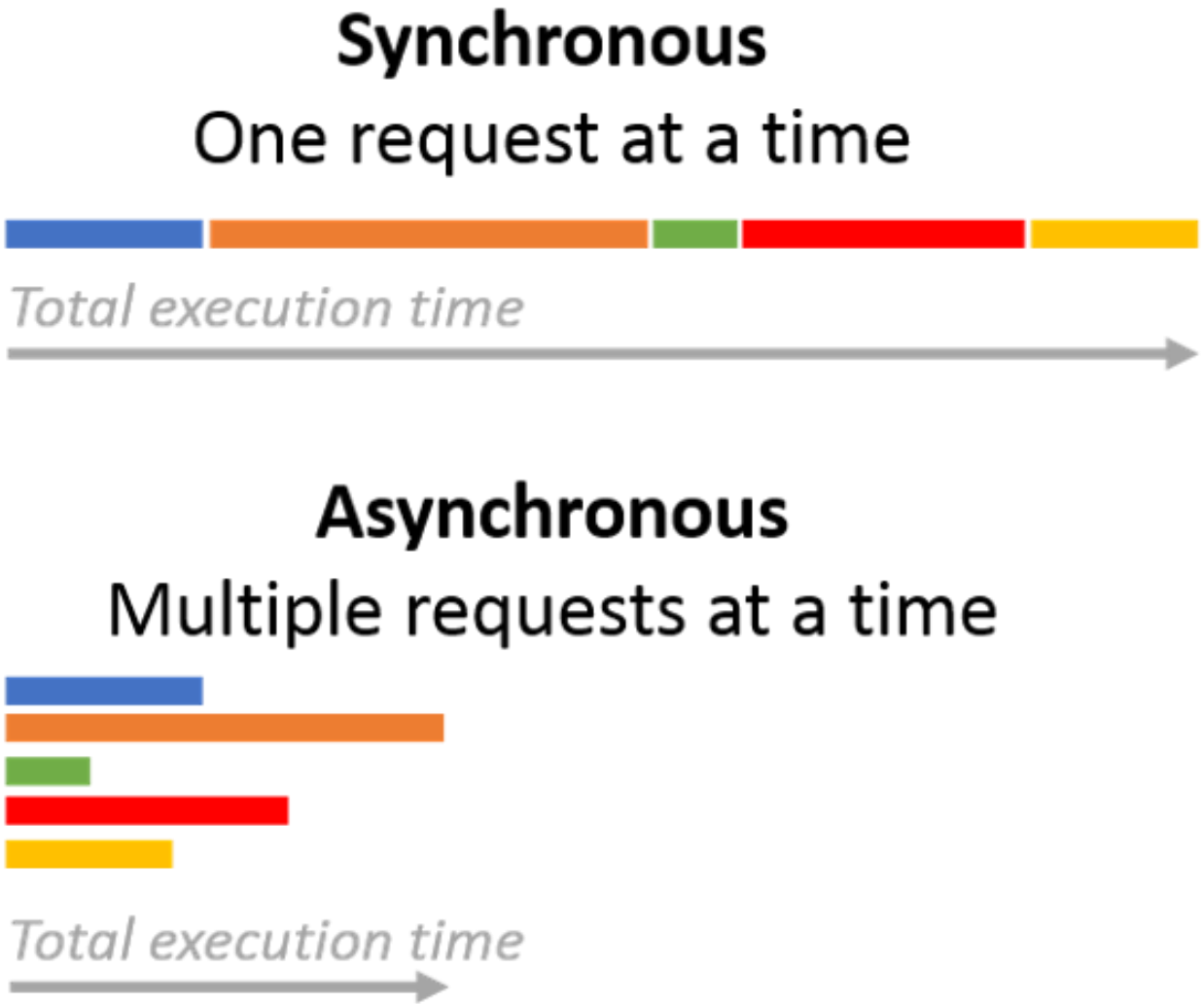
”Um sistema distribuído é aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas passando mensagens”

Tanenbaum e Van Steen,
Sistemas Distribuídos: Princípios e Paradigmas, 2007

Características dos sistemas distribuídos

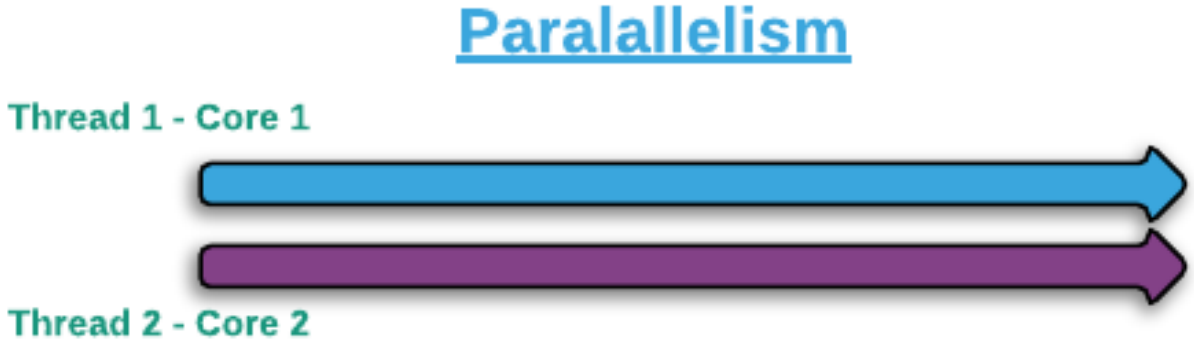
- Heterogeneidade
- Escalabilidade
- Segurança
- Tratamento de Falhas
- Concorrência
- Transparência

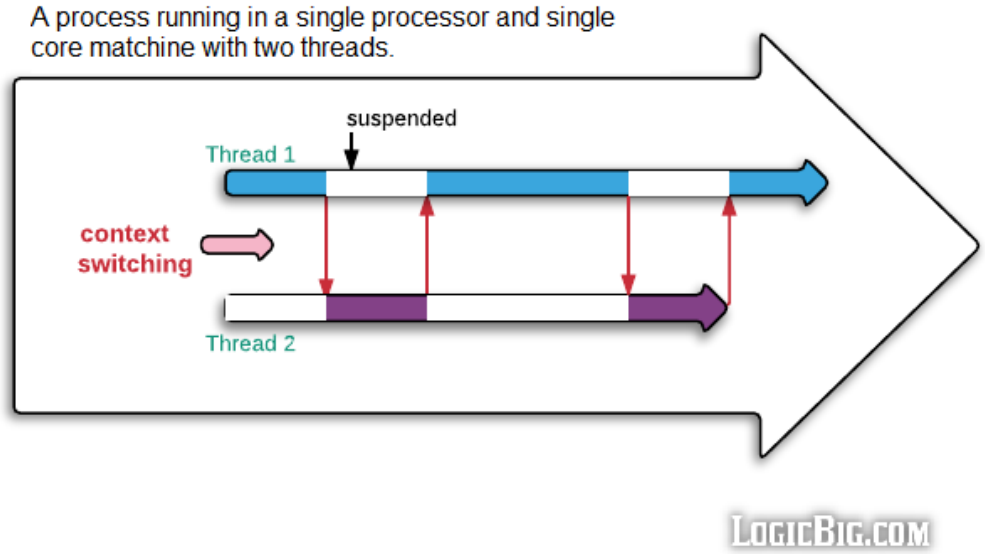
Concorrência e paralelismo

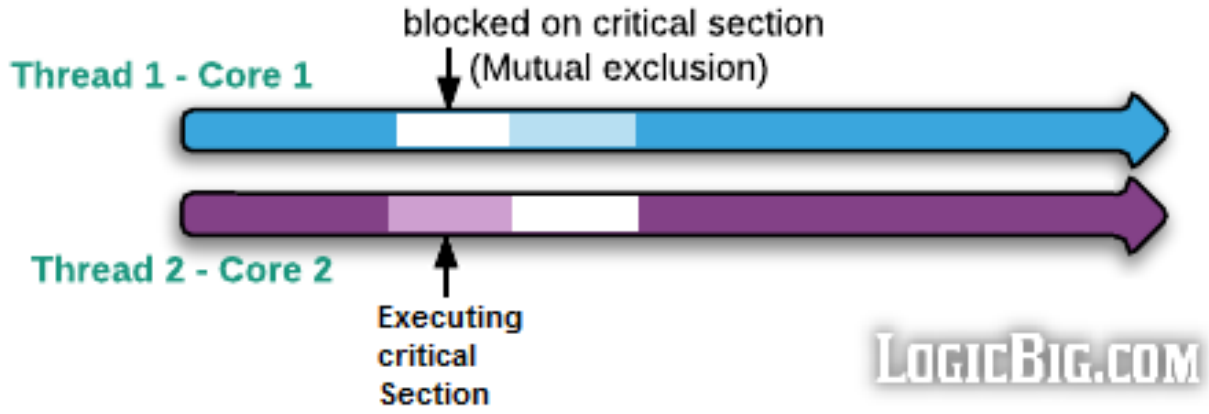


Fonte: msl-network.readthedocs.io

Concorrência e paralelismo







Fonte: https://www.logicbig.com/quick-info/programming/multi-threading.html

Sistemas distribuídos baseados em mensageria

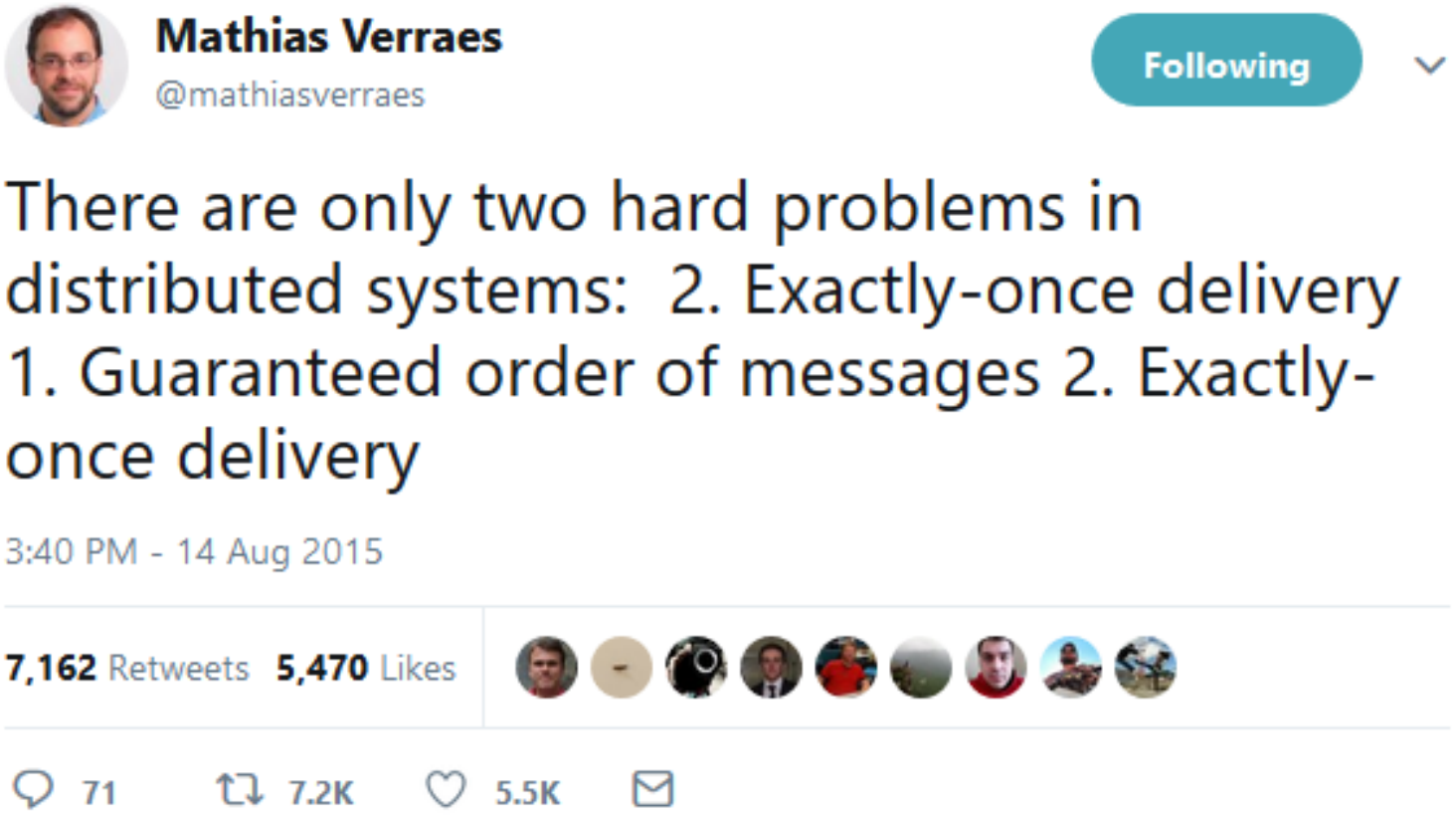
Async

Alto índice de concorrência e paralelismo

Gestão de concorrência

Idempotência

Sistemas distribuídos baseados em mensageria



Fonte: Twitter

Idempotencia

"Um método é considerado idempotente se o resultado de uma requisição realizada com sucesso é independente do número de vezes que é executada." (Fonte: InfoQ)

No REST são: GET, HEAD, PUT, DELETE

Idempotencia - Soluções

- Gerir estados de operações assíncronas para garantir que não sejam feitas em duplicidade
  - Gestão centralizada
    - Mais burocrático e lento (Ex. Servico central, banco compartilhado)
  - Gestão local
    - Gestão com uso de Ids únicos, mais simples e rápido
- Controlar estado na origem

RabbitMQ – Message Broker

RabbitMQ Features

Asynchronous Messaging

Supports multiple messaging protocols, message queuing, delivery acknowledgement, flexible routing to queues, multiple exchange type.

Developer Experience

Deploy with BOSH, Chef, Docker and Puppet. Develop cross-language messaging with favorite programming languages such as: Java, .NET, PHP, Python, JavaScript, Ruby, Go, and many others.

Distributed Deployment

Deploy as clusters for high availability and throughput; federate across multiple availability zones and regions.

Enterprise & Cloud Ready

Pluggable authentication, authorization, supports TLS and LDAP. Lightweight and easy to deploy in public and private clouds.

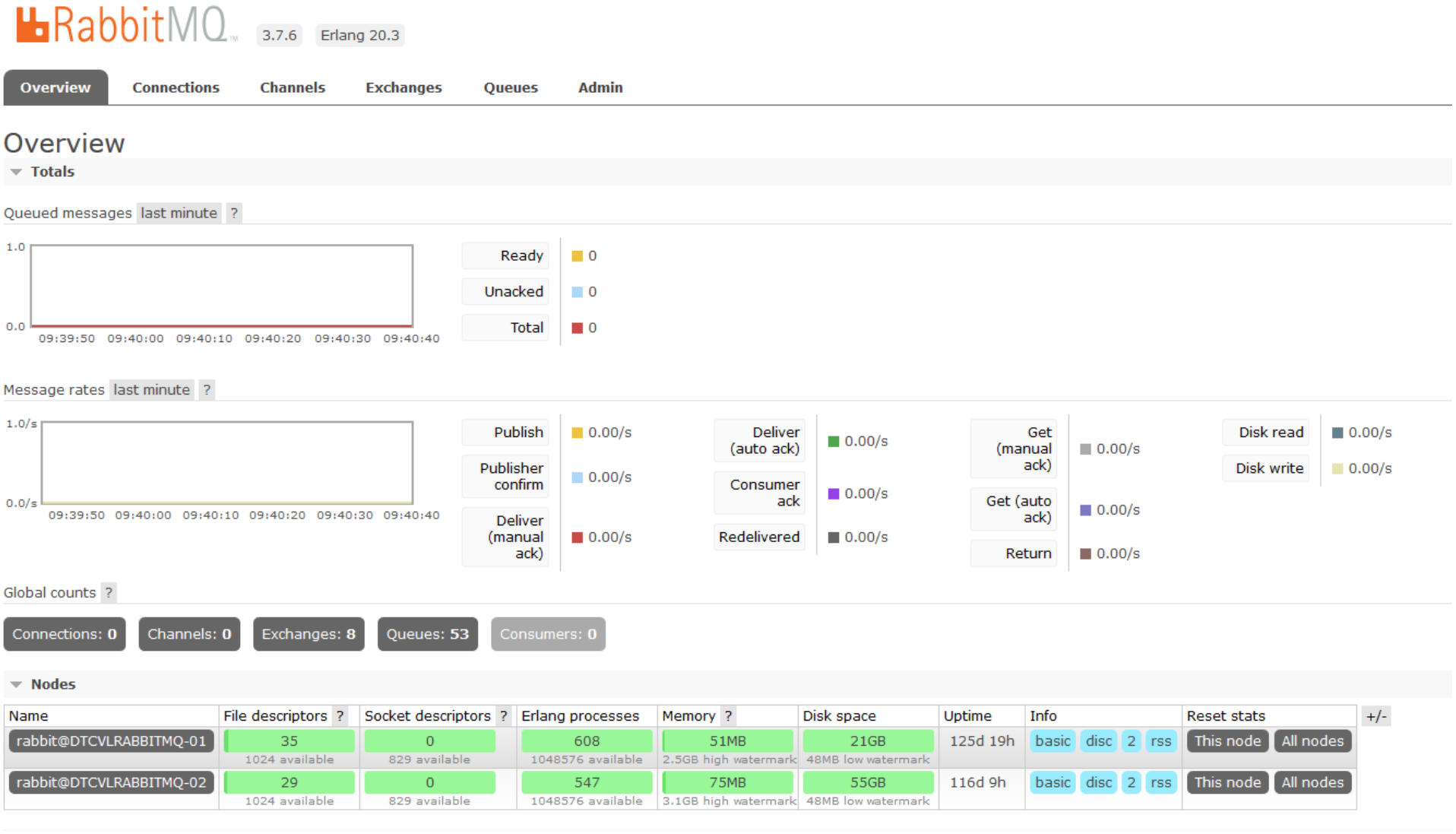
Tools & Plugins

Diverse array of tools and plugins supporting continuous integration, operational metrics, and integration to other enterprise systems. Flexible plug-in approach for extending RabbitMQ functionality.

Management & Monitoring

HTTP-API, command line tool, and UI for managing and monitoring RabbitMQ.

RabbitMQ – Message Broker



Vantagens do uso do Message Broker

- Balanceamento de carga
- Descoberta de serviços
- Garantia de Entrega
- Gestão de DeadLetters
- Simplicidade para fazer operações

E as desvantagens?

Gestão de status

Maior complexidade

Infraestrutura

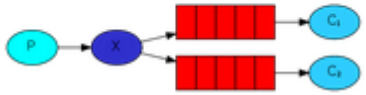
Tratamento de falhas

Utilizando RabbitMQ no Delphi

Fonte: https://www.wisys.com/2014/12/where-to-find-wisys-sdk-sample-code/cat-talk-is-cheap/

Publish/Subscribe

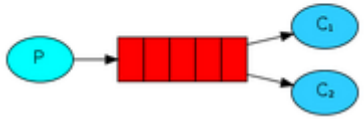
Sending messages to many consumers at once




```
procedure Example_Pub_Subscriber;
var
  StompPub, StompSubscriber: IStompClient;
  StompFrame: IStompFrame;
begin
  WriteLn('==> Example_Pub_Subscriber');
  StompSubscriber := StompUTils.StompClientAndConnect;

  StompSubscriber.Subscribe('/topic/dummy');
  StompPub := StompUTils.StompClientAndConnect;
  StompPub.Send('/topic/dummy', 'Some test message');
  repeat
    StompFrame := StompSubscriber.Receive(500);      danieleteti, 2 years ago • Refactoring
  until Assigned(StompFrame);
  WriteLn(StompFrame.Body); // Print "Some test message"
  WriteLn;
  StompSubscriber.Unsubscribe('/topic/dummy');
end;
```

Work Queues

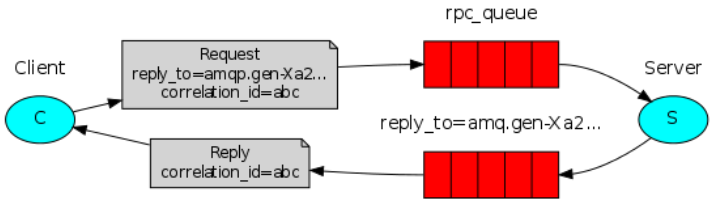
Distributing tasks among workers (the competing consumers pattern)




```
procedure Example_PointToPoint;
var
  StompPub, StompSub1, StompSub2: IStompClient;
  StompFrame: IStompFrame;
begin
  WriteLn('==> Example_PointToPoint');
  StompSub1 := StompUTils.StompClientAndConnect; // default port
  StompSub2 := StompUTils.StompClientAndConnect; // default port
  StompSub1.Subscribe('/queue/PointToPoint');
  StompSub2.Subscribe('/queue/PointToPoint');

  //
  StompPub := StompUtils.StompClientAndConnect; // default port
  StompPub.Send('/queue/PointToPoint', 'First test message on a queue');
  StompPub.Send('/queue/PointToPoint', 'Second test message on a queue');
```

```
  StompFrame := StompSub1.Receive(200);
  if Assigned(StompFrame) then
    WriteLn(StompFrame.Output);
  StompFrame := StompSub1.Receive(200);
  if Assigned(StompFrame) then
    WriteLn(StompFrame.Output);

  StompFrame := StompSub2.Receive(200);
  if Assigned(StompFrame) then
    WriteLn(StompFrame.Output);
  StompFrame := StompSub2.Receive(200);
  if Assigned(StompFrame) then
    WriteLn(StompFrame.Output);

  WriteLn;
end;
```

RPC – Remote Procedure Call




```
procedure Example_ReplyTo_Queue;
var
  lReplyTo: IStompClient;
  StompFrame: IStompFrame;
begin
  WriteLn('==> Example_ReplyTo_Queue');

  lReplyTo := StompUtils.StompClientAndConnect;
  lReplyTo.Send('/queue/dummy', 'Some test message',
    StompUtils.Headers.Add(
      StompHeaders.NewReplyToHeader('/temp-queue/temp'), 'true'));
  repeat
    StompFrame := lReplyTo.Receive(500);
  until Assigned(StompFrame);

  WriteLn(StompFrame.Body); // Print "Some test message"
  lReplyTo.Ack(StompFrame.MessageID);
  WriteLn;
end;      You, a few seconds ago • Uncommitted changes
```

Como fizemos

Usando protocolo STOMP (Simple Text Over Message Protocol)	• https://github.com/danieleteti/delphistompclient • Menos recursos que o AMQP • Desenvolvida pelo Daniele Teti
Implementação de uma biblioteca padrão interna com base no StompClient	• Padronização de comportamentos • Simplificar de desenvolvimento • Gestão de ACK/NACK e Erros
Definição de contratos padrões e versionados	• Contratos baseados em interfaces • Evolução de contratos • Classes que implementam várias interfaces
Definição de comportamentos padrões	• CustomParams • OriginIP • OperationsIDs

Razões para adotar este modelo

Adoção incremental

Reaproveitamento do legado

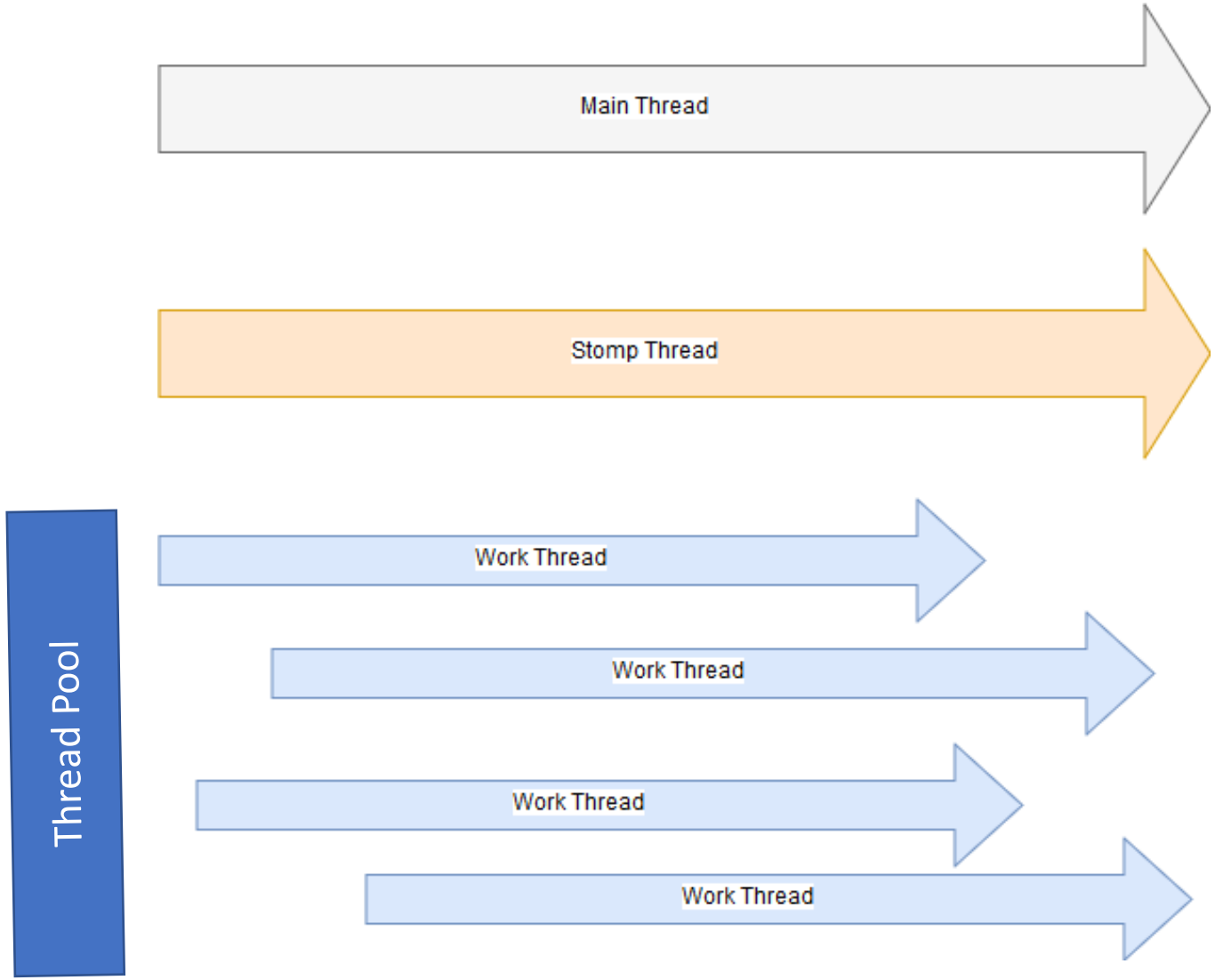
Alta interoperabilidade

Escalabilidade

Controle de falhas

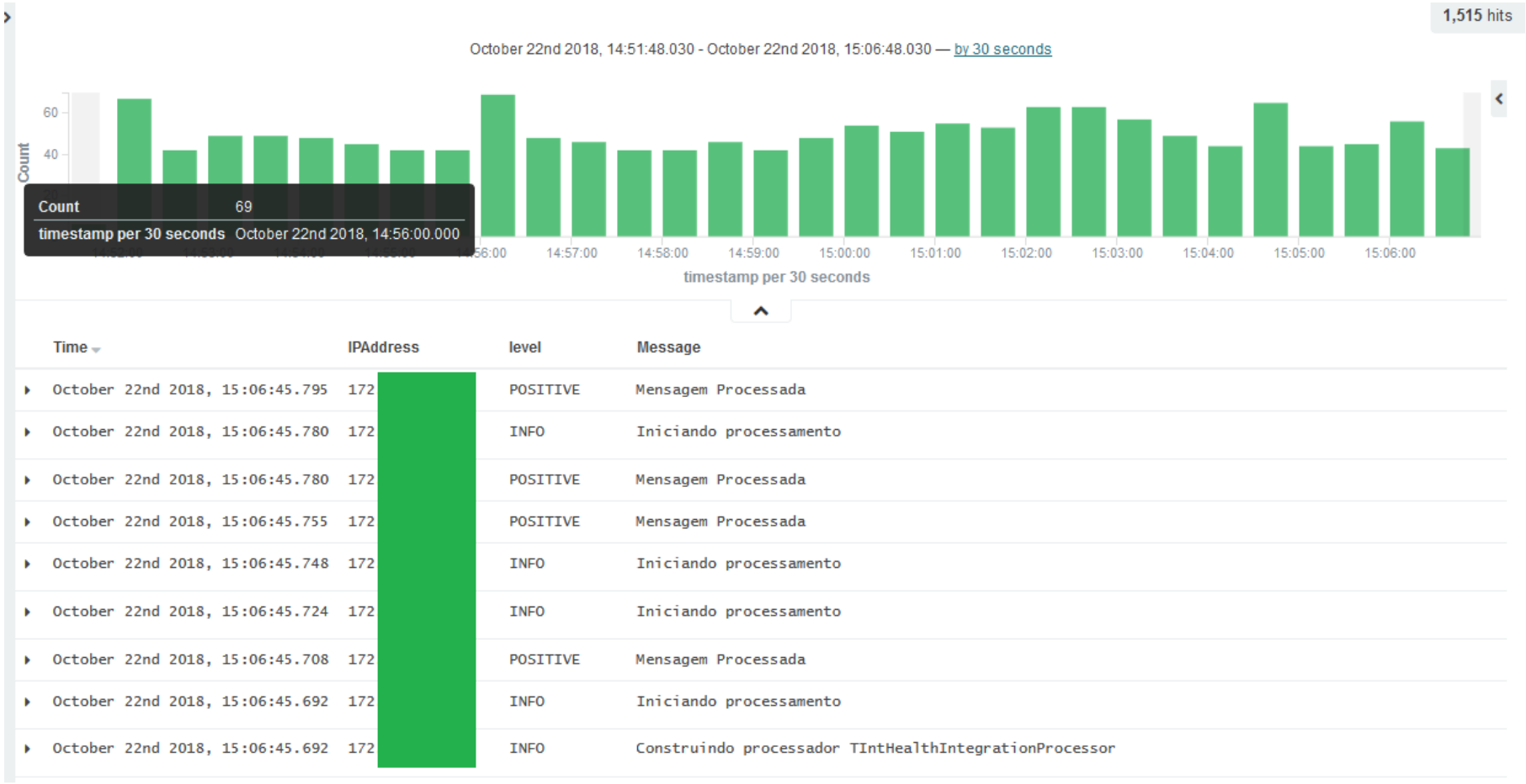
Mas e no mundo real?

Não execute o processamentos de mensagens na thread Principal do sistema

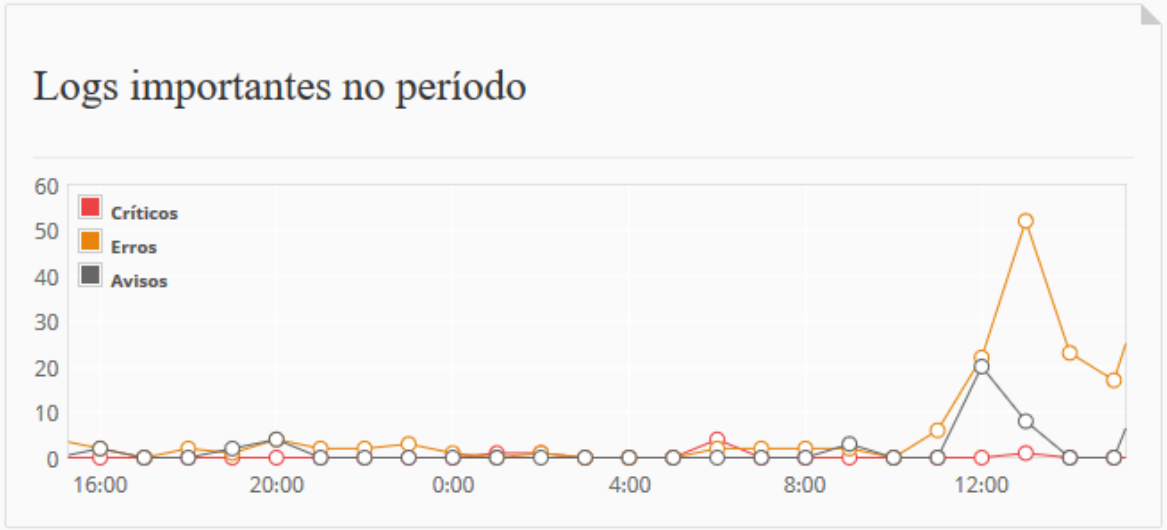


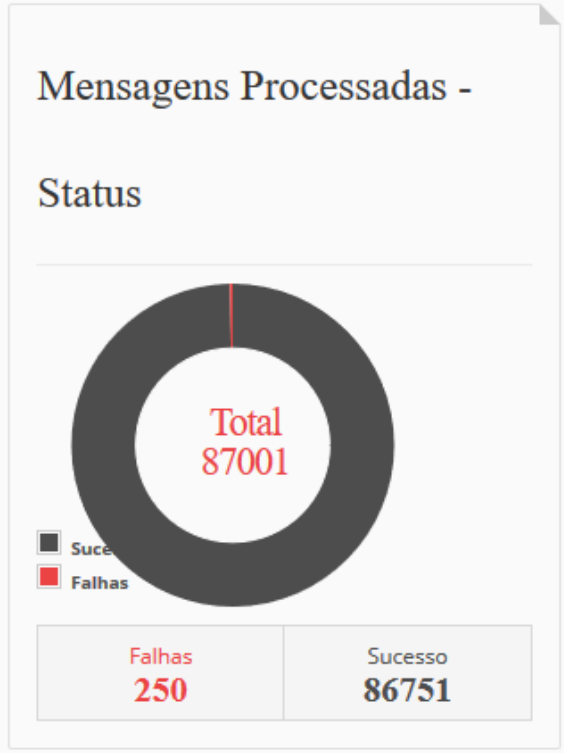
Sistema Distribuído + Log Centralizado = <3

- Não deixar os logs em cada instância da aplicação



Monitore sempre!



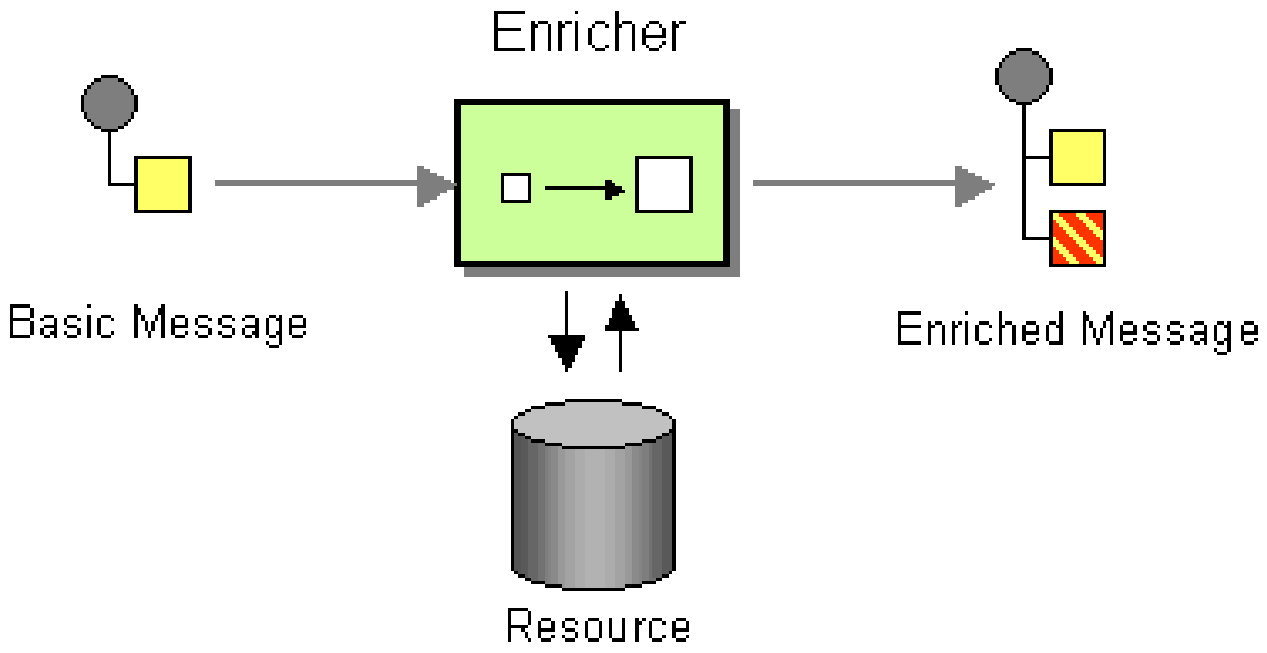


Versionamento de mensagens

```
type
  IIntMsgConsultaAvisosPendentesV1 = interface(IIntIntegrationMessage)
    ['{764B1378-CEE2-4148-8E4D-2EF233EA0C0D}']

    function GetIdRepresentado: string;
    procedure SetIdRepresentado(sIdRepresentado: string);
    function GetIdConsultante: string;
    procedure SetIdConsultante(sIdConsultante: string);
    function GetSenhaConsultante: string;
    procedure SetSenhaConsultante(sSenhaConsultante: string);
    function GetDataReferencia: string;
    procedure SetDataReferencia(sDataReferencia: string);

    property idRepresentado: string read GetIdRepresentado write SetIdRepresentado;
    property idConsultante: string read GetIdConsultante write SetIdConsultante;
    property SenhaConsultante: string read GetSenhaConsultante write SetSenhaConsultante;
    property DataReferencia: string read GetDataReferencia write SetDataReferencia;
  end;
```

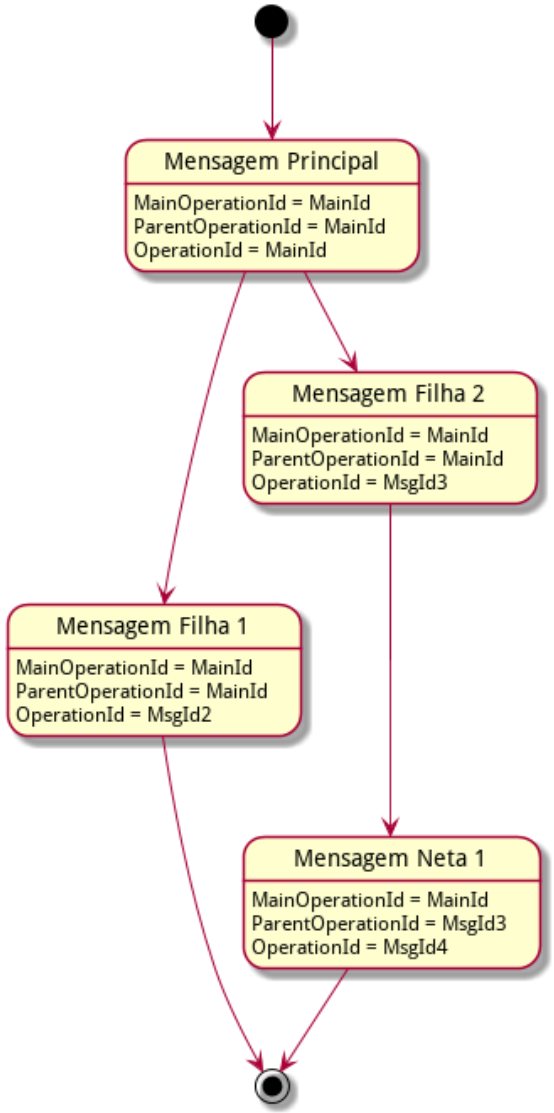
```
type
  IIntMsgConsultaAvisosPendentesV2 = interface
    ['{45A0C5E5-02E4-4FBE-915F-9B118DCF86C2}']

    function GetAgruparAvisos: Boolean;
    procedure SetAgruparAvisos(const Value: Boolean);

    procedure SetForcarConsultaProcesso(const Value: Boolean);
    function GetForcarConsultaProcesso: Boolean;

    property AgruparAvisos: Boolean read GetAgruparAvisos write SetAgruparAvisos;
    property ForcarConsultaProcesso: Boolean read GetForcarConsultaProcesso write SetForcarConsultaProcesso;
  end;
```

Maleabilidade nos contratos



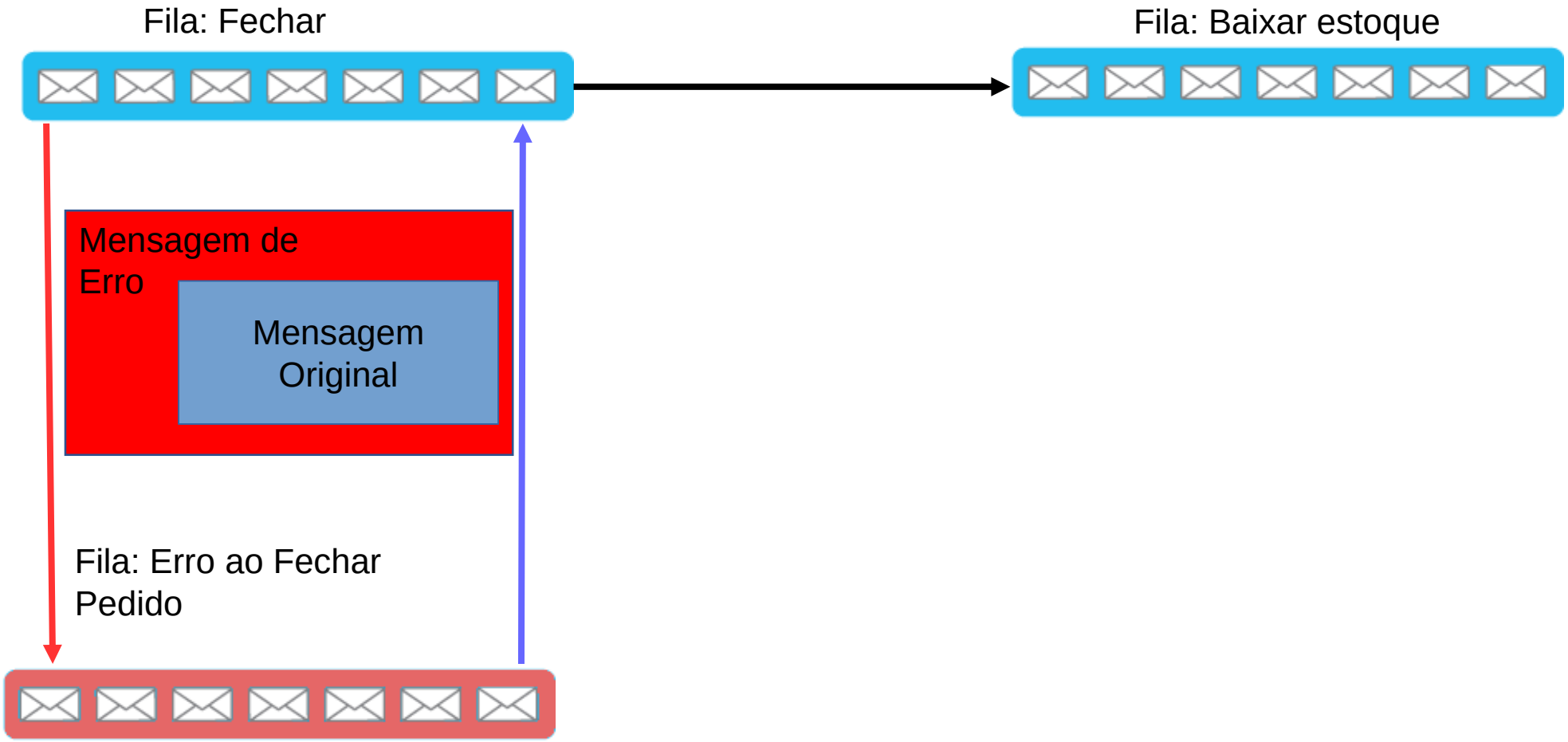
Fonte: http://camel.apache.org/content-enricher.html

Rastreamento de operações



- Rastreamento de operações End2End
  - Operação principal e ramificações
- Adicione dados de identificação da operação
  - Codigo da Venda, Número da NFe, etc
- Não adicione dados sensíveis
  - Senhas, dados pessoais e etc
- Adicione dados que sejam úteis para identificação e correção de erros.
  - Exceções, classes da exceção, contexto de operações

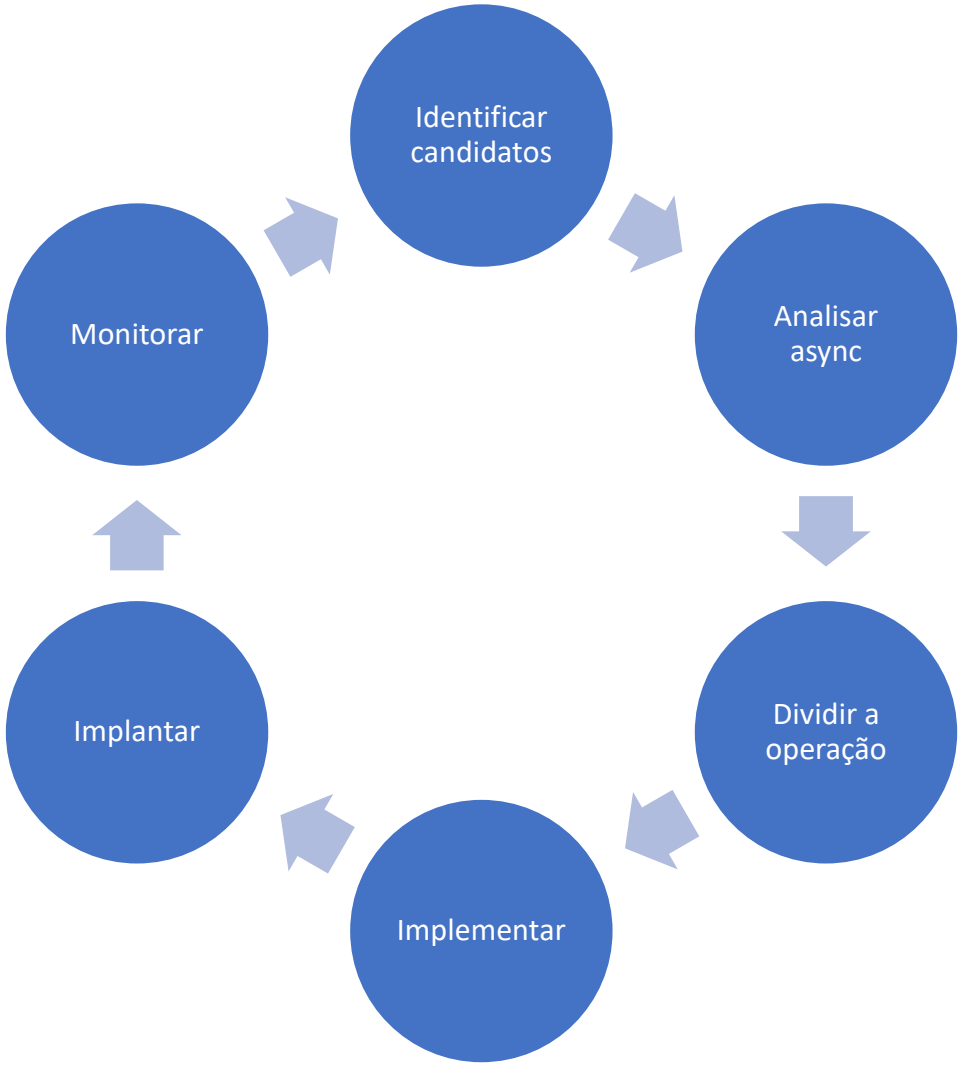
Tratamento de erros



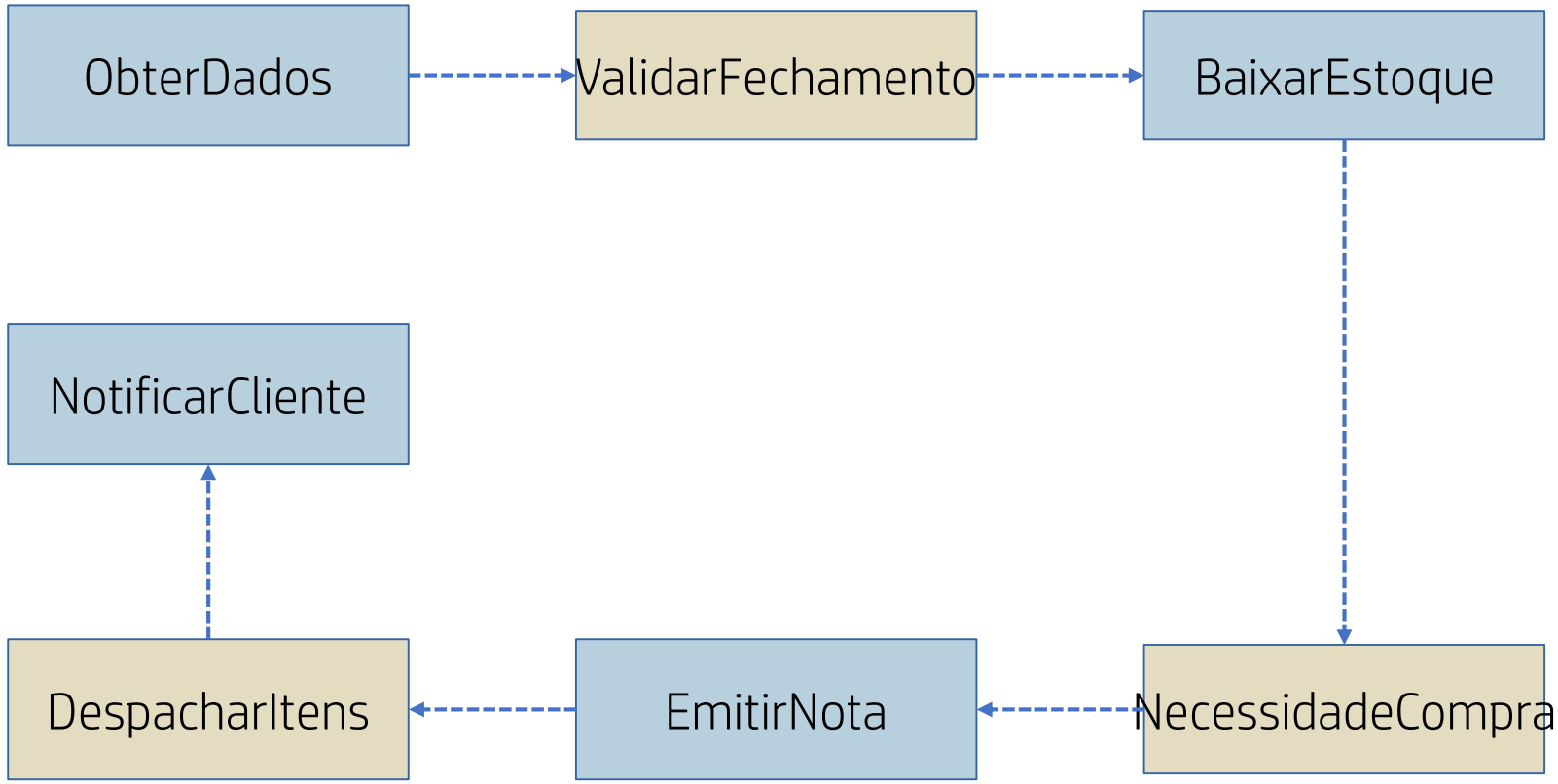
O que estamos usando?

- Tecnologia agnóstica
- Facilmente adaptável
- Escalável
- Operações atômicas

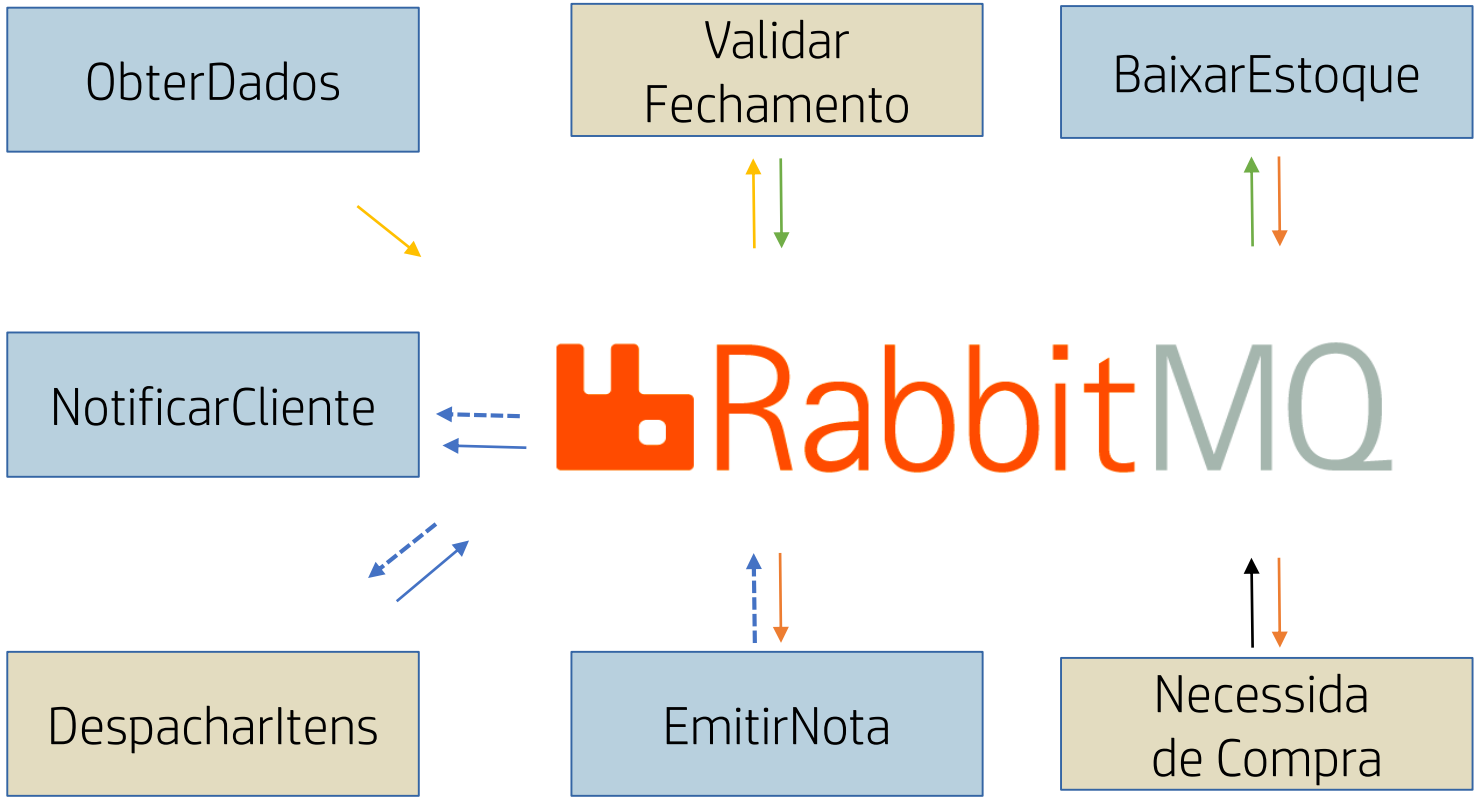
Como começar a adotar?



Exemplo de aplicação



Divindo em processos independentes



Quando usar processamento distribuído

Fonte: https://pixr8.com/inspirational/know-how-this-startup-founder-beat-his-depression-in-a-weird-but-amazing-way/attachment/with-great-power-comes-great-responsibility-spiderman/

Quando utilizar?

- Muitas suboperações que podem ser paralelizadas
- Operações que não dependem de interação com o usuário
- Operações com sistemas terceiros
- Operações que podem ser assíncronas

E quando não?

- Operações que rodam somente no cliente
- Muitas interações com o usuário
- Necessidade de operações síncronas
- Obrigatoriamente transacional

Quer fazer a diferença na vida das pessoas?

Estamos contratando!!

embarcadero

OBRIGADO

Embarcadero Conference

embarcadero®