Scaling Web Apps With RabbitMQ

Álvaro Videla | The NetCircle

Erlang Factory Lite 2010

Who?

About Me

- Development Manager at TheNetCircle.com
- Writing "RabbitMQ in Action" for Manning
- Blog: http://videlalvaro.github.com/
- Twitter: @old_sound

Why Do I need RabbitMQ?

The User

I don’t want to wait till your app resizes my image!

The Product Owner

Can we also notify the user friends when she uploads a new image?

Can we also notify the user friends when she uploads a new image?

I forgot to mention we need it for tomorrow…

The Sysadmin

Dumb! You're delivering full size images!

The bandwidth bill has tripled!

Dumb! You're delivering full size images!

The bandwidth bill has tripled!

We need this fixed for yesterday!

The Developer in the other team

I need to call your PHP stuff but from Python

I need to call your PHP stuff but from Python

And also Java starting next week

You

FML!

Is there a solution?

RabbitMQ & AMQP

AMQP

AMQP

- Advanced Message Queuing Protocol
- Suits Interoperability
- Completely Open Protocol
- Binary Protocol
- AMQP Model
- AMQP Wire Format

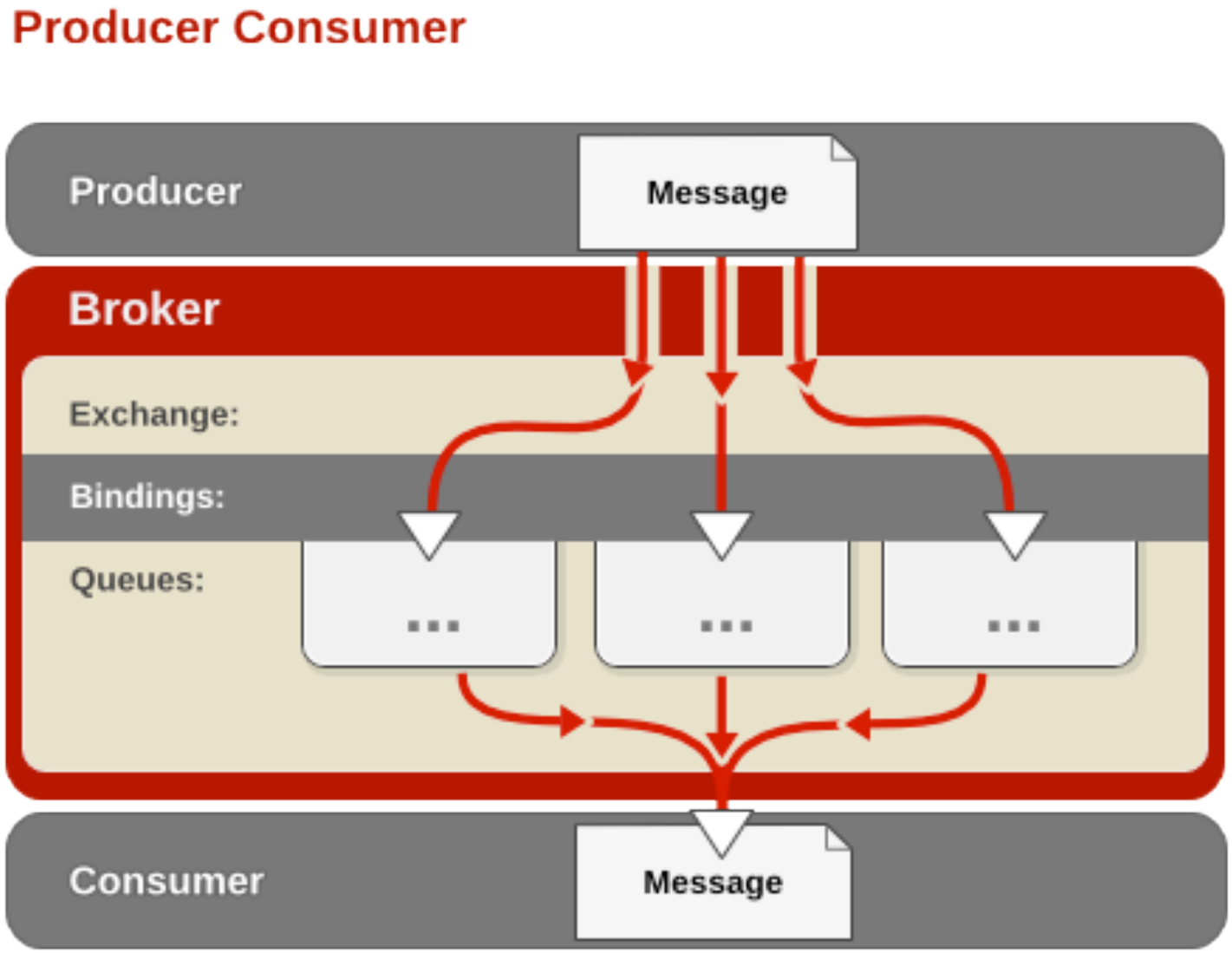
AMQP Model

- Exchanges
- Message Queues
- Bindings
- Rules for binding them

AMQP Wire Protocol

- Functional Layer
- Transport Layer

Message Flow

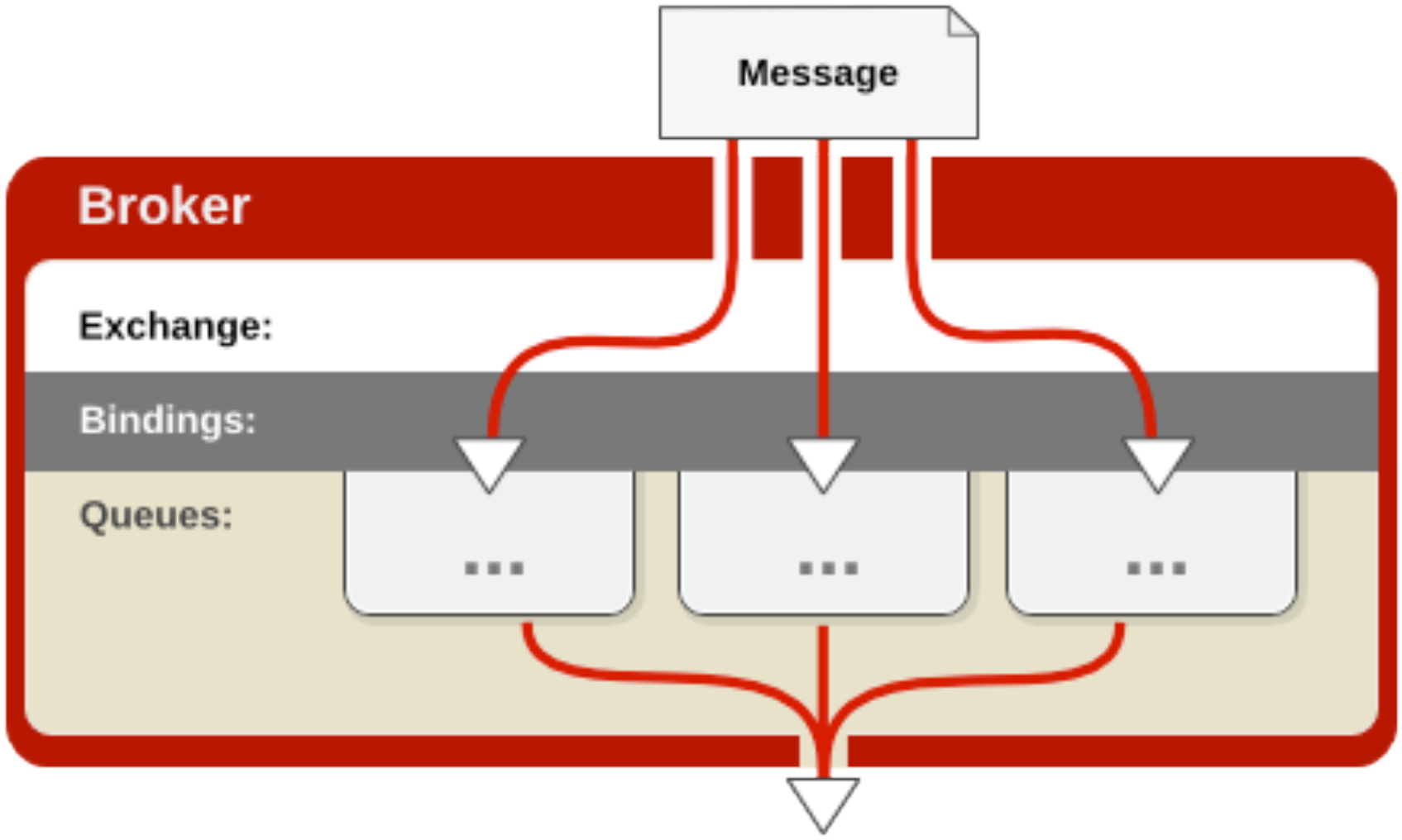


http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_MRG/1.0/html/Messaging_Tutorial/chap-Messaging_Tutorial-Initial_Concepts.html

Exchange Types

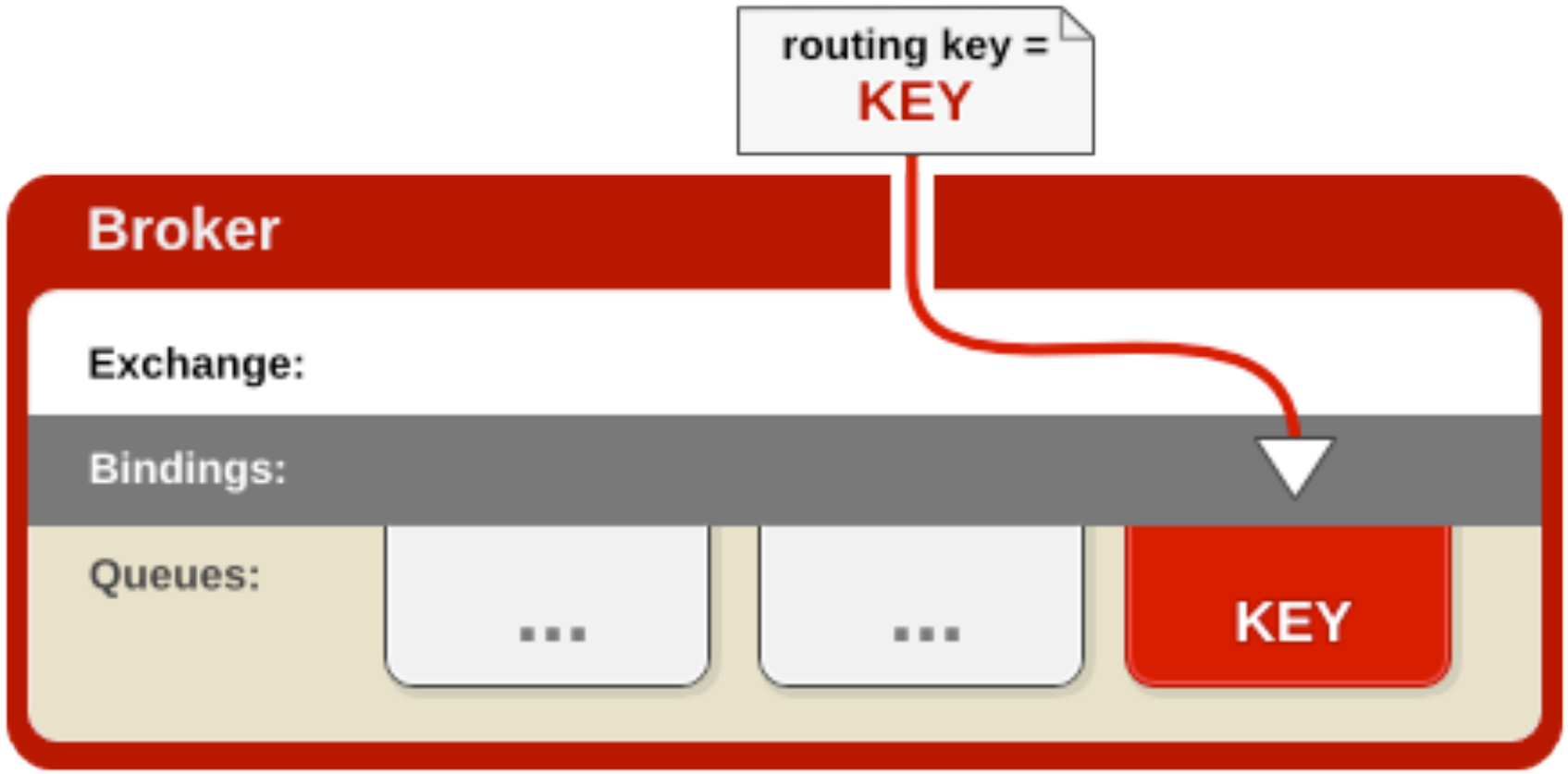
- Fanout
- Direct
- Topic

Fanout Exchange



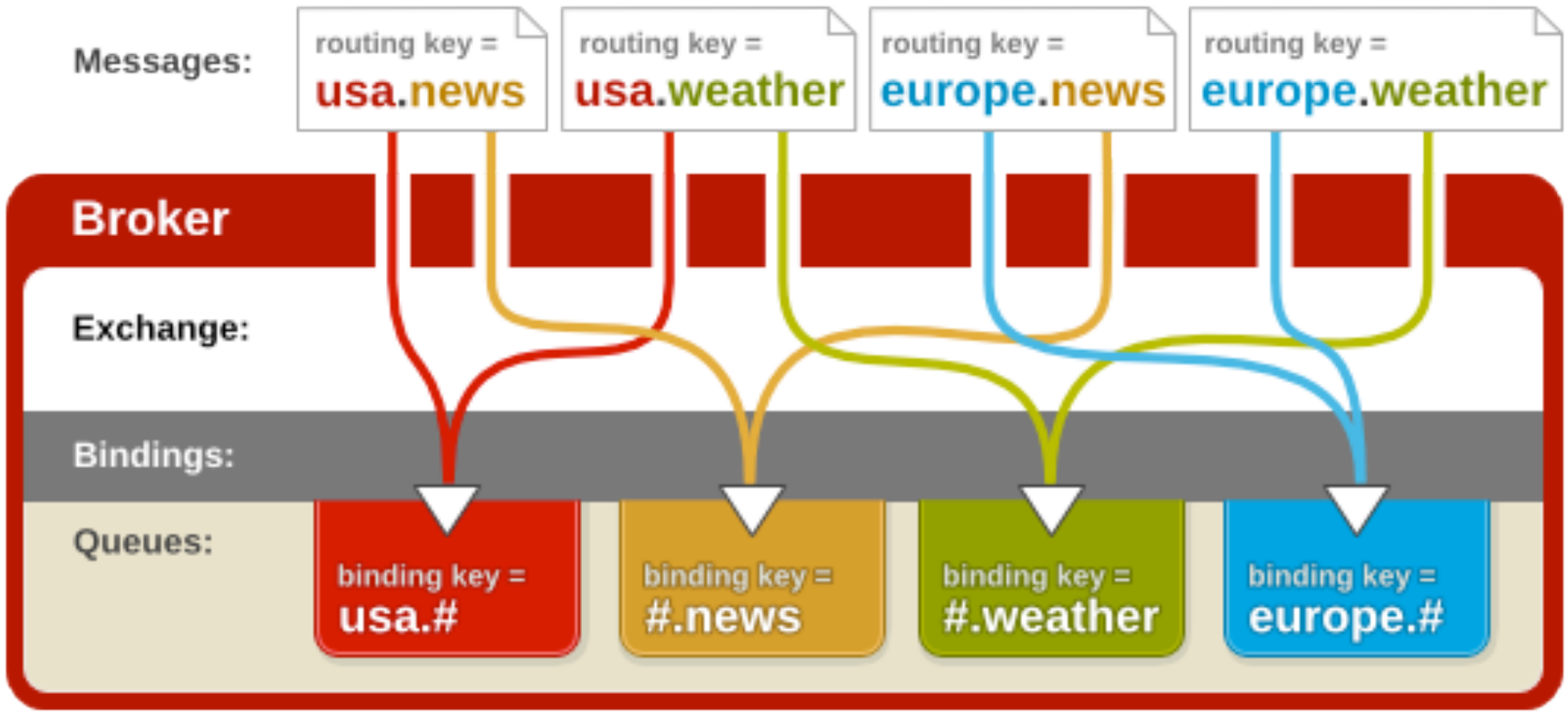
http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_MRG/1.0/html/Messaging_Tutorial/sect-Messaging_Tutorial-Initial_Concepts-Fanout_Exchange.html

Direct Exchange



http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_MRG/1.0/html/Messaging_Tutorial/sect-Messaging_Tutorial-Initial_Concepts-Direct_Exchange.html

Topic Exchange



http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_MRG/1.0/html/Messaging_Tutorial/sect-Messaging_Tutorial-Initial_Concepts-Topic_Exchange.html

Usage Scenarios

Usage Scenarios

- Batch Processing

Usage Scenarios

- Batch Processing
- Image Uploading

Usage Scenarios

- Batch Processing
- Image Uploading
- Distributed Logging

Scenario

Batch Processing

Requirements

Requirements

- Generate XML

Requirements

- Generate XML
- Distribution Over a Cluster

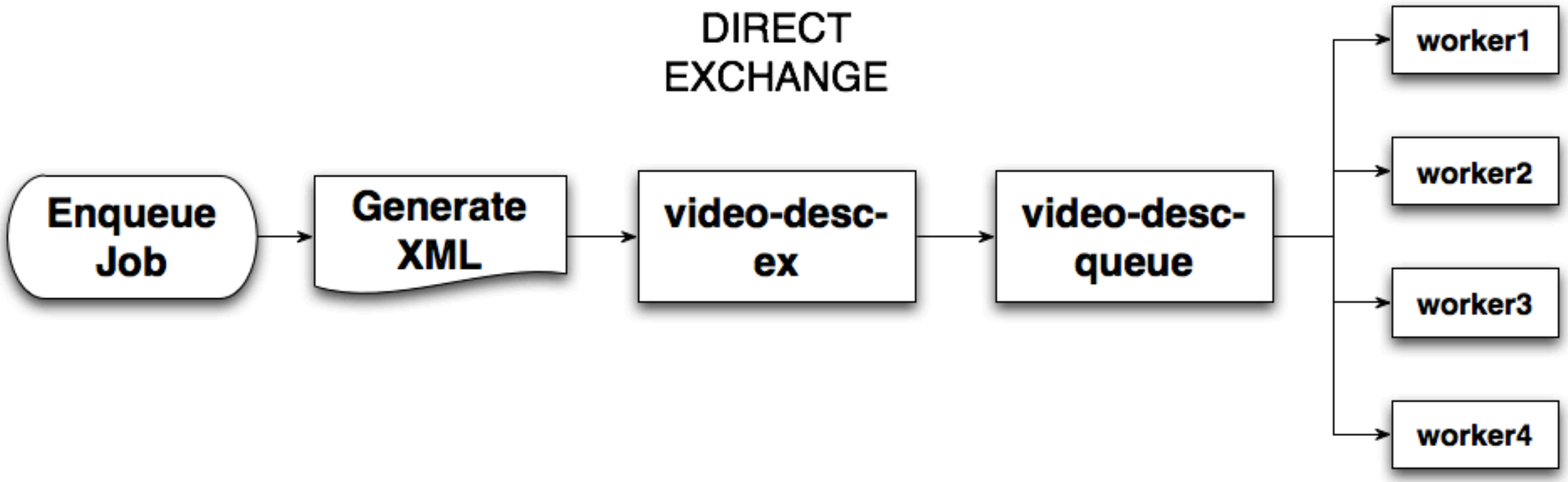
Requirements

- Generate XML
- Distribution Over a Cluster
- Elasticity - Add/Remove new workers

Requirements

- Generate XML
- Distribution Over a Cluster
- Elasticity - Add/Remove new workers
- No Code Changes

Design


DIRECT
EXCHANGE
Enqueue Job
Generate XML
video-desc-ex
video-desc-queue
worker1
worker2
worker3
worker4

Publisher Code

```
$conn = new AMQPConnection(HOST, PORT, USER, PASS, VHOST);
$channel = $conn->channel();

$channel->exchange_declare('video-desc-ex', 'direct', false,
                true, false);

$msg = new AMQPMessage($video_info,
            array('content_type' => 'text/plain',
                  'delivery_mode' => 2));

$channel->basic_publish($msg, 'video-desc-ex');

$channel->close();
$conn->close();
```

Publisher Code

```
$conn = new AMQPConnection(HOST, PORT, USER, PASS, VHOST);
$channel = $conn->channel();

$channel->exchange_declare('video-desc-ex', 'direct', false,
                true, false);

$msg = new AMQPMessage($video_info,
            array('content_type' => 'text/plain',
                  'delivery_mode' => 2));

$channel->basic_publish($msg, 'video-desc-ex');

$channel->close();
$conn->close();
```

Publisher Code

```
$conn = new AMQPConnection(HOST, PORT, USER, PASS, VHOST);
$channel = $conn->channel();

$channel->exchange_declare('video-desc-ex', 'direct', false,
                true, false);

$msg = new AMQPMessage($video_info,
            array('content_type' => 'text/plain',
                  'delivery_mode' => 2));

$channel->basic_publish($msg, 'video-desc-ex');

$channel->close();
$conn->close();
```

Publisher Code

```
$conn = new AMQPConnection(HOST, PORT, USER, PASS, VHOST);
$channel = $conn->channel();

$channel->exchange_declare('video-desc-ex', 'direct', false,
                true, false);

$msg = new AMQPMessage($video_info,
            array('content_type' => 'text/plain',
                  'delivery_mode' => 2));

$channel->basic_publish($msg, 'video-desc-ex');

$channel->close();
$conn->close();
```

Publisher Code

```
$conn = new AMQPConnection(HOST, PORT, USER, PASS, VHOST);
$channel = $conn->channel();

$channel->exchange_declare('video-desc-ex', 'direct', false,
                true, false);

$msg = new AMQPMessage($video_info,
            array('content_type' => 'text/plain',
                  'delivery_mode' => 2));

$channel->basic_publish($msg, 'video-desc-ex');

$channel->close();
$conn->close();
```

Publisher Code

```
$conn = new AMQPConnection(HOST, PORT, USER, PASS, VHOST);
$channel = $conn->channel();

$channel->exchange_declare('video-desc-ex', 'direct', false,
                true, false);

$msg = new AMQPMessage($video_info,
            array('content_type' => 'text/plain',
                  'delivery_mode' => 2));

$channel->basic_publish($msg, 'video-desc-ex');

$channel->close();
$conn->close();
```

Consumer Code

```
$conn = new AMQPConnection(HOST, PORT, USER, PASS, VHOST);
$channel = $conn->channel();

$channel->exchange_declare('video-desc-ex', 'direct', false,
                true, false);
$channel->queue_declare('video-desc-queue', false, true,
                false, false);
$channel->queue_bind('video-desc-queue', 'video-desc-ex');

$channel->basic_consume('video-desc-queue', $consumer_tag,
                false, false, false, false, $consumer);

while(count($channel->callbacks)) {
    $channel->wait();
}
```

Consumer Code

```
$conn = new AMQPConnection(HOST, PORT, USER, PASS, VHOST);
$channel = $conn->channel();

$channel->exchange_declare('video-desc-ex', 'direct', false,
                true, false);
$channel->queue_declare('video-desc-queue', false, true,
                false, false);
$channel->queue_bind('video-desc-queue', 'video-desc-ex');

$channel->basic_consume('video-desc-queue', $consumer_tag,
                false, false, false, false, $consumer);

while(count($channel->callbacks)) {
    $channel->wait();
}
```

Consumer Code

```
$conn = new AMQPConnection(HOST, PORT, USER, PASS, VHOST);
$channel = $conn->channel();

$channel->exchange_declare('video-desc-ex', 'direct', false,
                true, false);
$channel->queue_declare('video-desc-queue', false, true,
                false, false);
$channel->queue_bind('video-desc-queue', 'video-desc-ex');

$channel->basic_consume('video-desc-queue', $consumer_tag,
                false, false, false, false, $consumer);

while(count($channel->callbacks)) {
    $channel->wait();
}
```

Consumer Code

```
$conn = new AMQPConnection(HOST, PORT, USER, PASS, VHOST);
$channel = $conn->channel();

$channel->exchange_declare('video-desc-ex', 'direct', false,
                true, false);
$channel->queue_declare('video-desc-queue', false, true,
                false, false);
$channel->queue_bind('video-desc-queue', 'video-desc-ex');

$channel->basic_consume('video-desc-queue', $consumer_tag,
                false, false, false, false, $consumer);

while(count($channel->callbacks)) {
    $channel->wait();
}
```

Consumer Code

```
$conn = new AMQPConnection(HOST, PORT, USER, PASS, VHOST);
$channel = $conn->channel();

$channel->exchange_declare('video-desc-ex', 'direct', false,
                true, false);
$channel->queue_declare('video-desc-queue', false, true,
                false, false);
$channel->queue_bind('video-desc-queue', 'video-desc-ex');

$channel->basic_consume('video-desc-queue', $consumer_tag,
                false, false, false, false, $consumer);

while(count($channel->callbacks)) {
    $channel->wait();
}
```

Consumer Code

```
$conn = new AMQPConnection(HOST, PORT, USER, PASS, VHOST);
$channel = $conn->channel();

$channel->exchange_declare('video-desc-ex', 'direct', false,
                true, false);
$channel->queue_declare('video-desc-queue', false, true,
                false, false);
$channel->queue_bind('video-desc-queue', 'video-desc-ex');

$channel->basic_consume('video-desc-queue', $consumer_tag,
                false, false, false, false, $consumer);

while(count($channel->callbacks)) {
    $channel->wait();
}
```

Consumer Code

```
$conn = new AMQPConnection(HOST, PORT, USER, PASS, VHOST);
$channel = $conn->channel();

$channel->exchange_declare('video-desc-ex', 'direct', false,
                true, false);
$channel->queue_declare('video-desc-queue', false, true,
                false, false);
$channel->queue_bind('video-desc-queue', 'video-desc-ex');

$channel->basic_consume('video-desc-queue', $consumer_tag,
                false, false, false, false, $consumer);

while(count($channel->callbacks)) {
    $channel->wait();
}
```

Scenario

Upload Pictures

Requirements

Requirements

- Upload Picture

Requirements

- Upload Picture
- Reward User

Requirements

- Upload Picture
- Reward User
- Notify User Friends

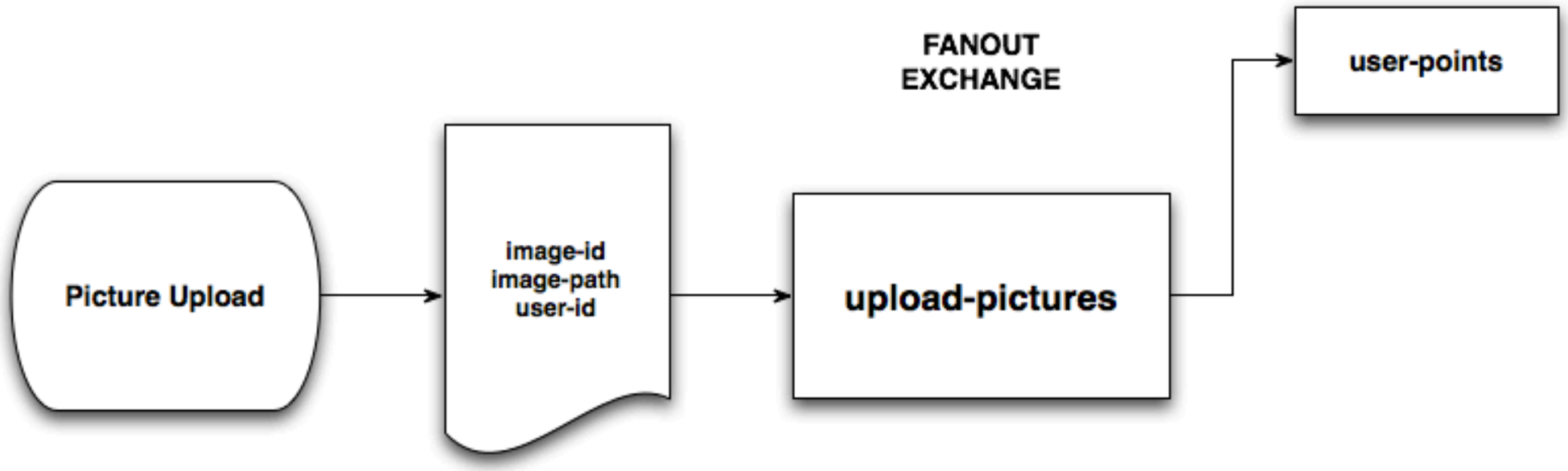
Requirements

- Upload Picture
- Reward User
- Notify User Friends
- Resize Picture

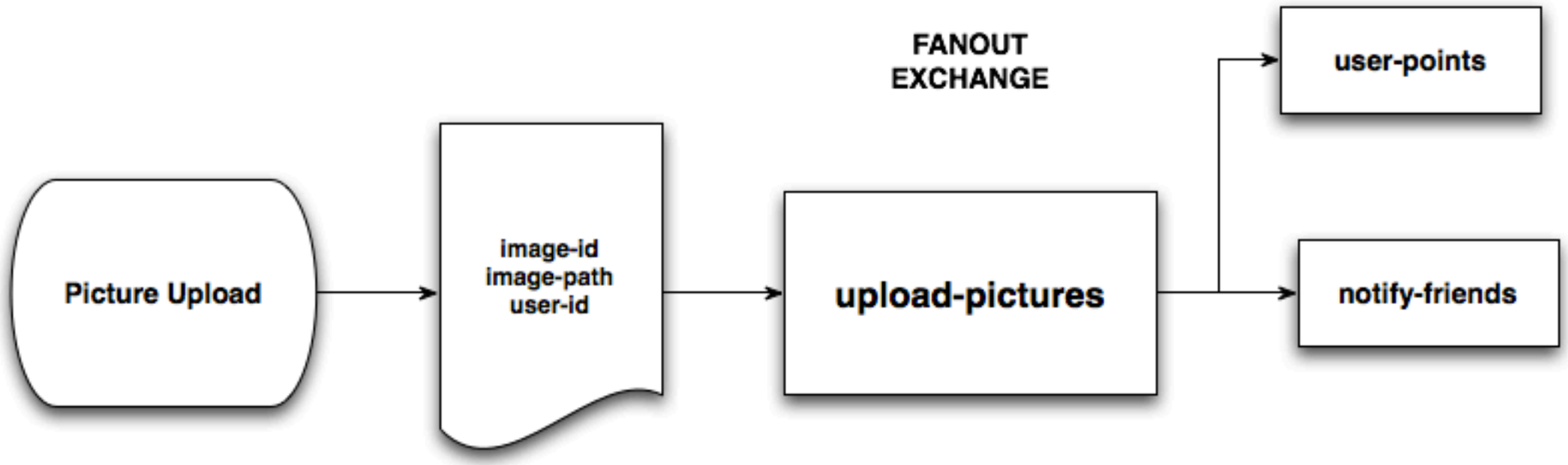
Requirements

- Upload Picture
- Reward User
- Notify User Friends
- Resize Picture
- No Code Changes

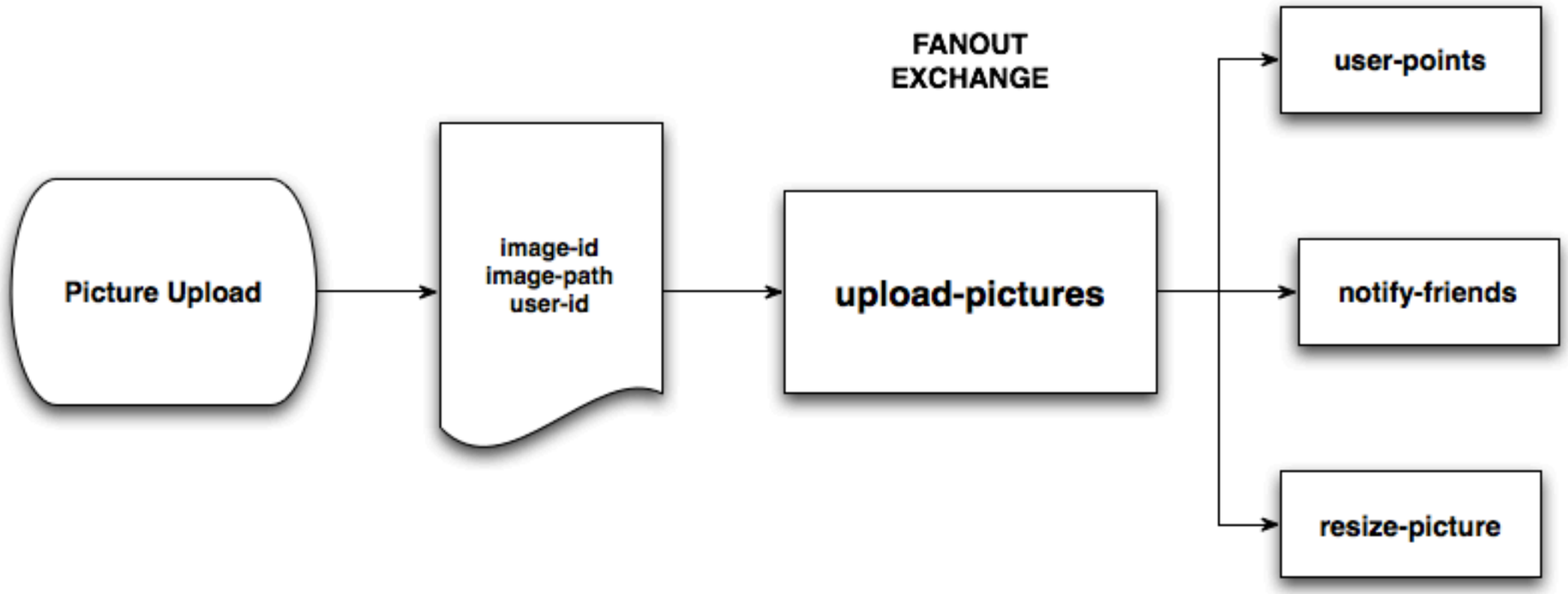
Design


FANOUT
EXCHANGE
user-points
Picture Upload
image-id
image-path
user-id
upload-pictures

Design


FANOUT
EXCHANGE
user-points
Picture Upload
image-id
image-path
user-id
upload-pictures
notify-friends

Design


FANOUT
EXCHANGE
Picture Upload
image-id
image-path
user-id
upload-pictures
user-points
notify-friends
resize-picture

Publisher Code

```
$channel->exchange_declare('upload-pictures', 'fanout', false,
true, false);

$metadata = json_encode(array(
   'image_id' => $image_id,
   'user_id' => $user_id,
   ‘image_path' => $image_path));

$msg = new AMQPMessage($metadata, array('content_type' =>
'application/json',   'delivery_mode' => 2));

$channel->basic_publish($msg, 'upload-pictures');
```

Publisher Code

```
$channel->exchange_declare('upload-pictures', 'fanout', false,
true, false);

$metadata = json_encode(array(
   'image_id' => $image_id,
   'user_id' => $user_id,
   ‘image_path' => $image_path));

$msg = new AMQPMessage($metadata, array('content_type' =>
'application/json',   'delivery_mode' => 2));

$channel->basic_publish($msg, 'upload-pictures');
```

Publisher Code

```
$channel->exchange_declare('upload-pictures', 'fanout', false,
true, false);

$metadata = json_encode(array(
   'image_id' => $image_id,
   'user_id' => $user_id,
   ‘image_path' => $image_path));

$msg = new AMQPMessage($metadata, array('content_type' =>
'application/json',   'delivery_mode' => 2));

$channel->basic_publish($msg, 'upload-pictures');
```

Publisher Code

```
$channel->exchange_declare('upload-pictures', 'fanout', false,
true, false);

$metadata = json_encode(array(
   'image_id' => $image_id,
   'user_id' => $user_id,
   ‘image_path' => $image_path));

$msg = new AMQPMessage($metadata, array('content_type' =>
'application/json',   'delivery_mode' => 2));

$channel->basic_publish($msg, 'upload-pictures');
```

Publisher Code

```
$channel->exchange_declare('upload-pictures', 'fanout', false,
true, false);

$metadata = json_encode(array(
   'image_id' => $image_id,
   'user_id' => $user_id,
   ‘image_path' => $image_path));

$msg = new AMQPMessage($metadata, array('content_type' =>
'application/json',   'delivery_mode' => 2));

$channel->basic_publish($msg, 'upload-pictures');
```

Consumer Code

```
$channel->exchange_declare('upload-pictures', 'fanout',
                false, true, false);

$channel->queue_declare('resize-picture', false, true,
                false, false);

$channel->queue_bind('resize-picture', 'upload-pictures');

$channel->basic_consume('resize-picture', $consumer_tag,
                false, false, false, false, $consumer);

while(count($channel->callbacks)) {
  $channel->wait();
}
```

Consumer Code

```
$channel->exchange_declare('upload-pictures', 'fanout',
                false, true, false);

$channel->queue_declare('resize-picture', false, true,
                false, false);

$channel->queue_bind('resize-picture', 'upload-pictures');

$channel->basic_consume('resize-picture', $consumer_tag,
                false, false, false, false, $consumer);

while(count($channel->callbacks)) {
  $channel->wait();
}
```

Consumer Code

```
$channel->exchange_declare('upload-pictures', 'fanout',
                false, true, false);

$channel->queue_declare('resize-picture', false, true,
                false, false);

$channel->queue_bind('resize-picture', 'upload-pictures');

$channel->basic_consume('resize-picture', $consumer_tag,
                false, false, false, false, $consumer);

while(count($channel->callbacks)) {
  $channel->wait();
}
```

Consumer Code

```
$channel->exchange_declare('upload-pictures', 'fanout',
                false, true, false);

$channel->queue_declare('resize-picture', false, true,
                false, false);

$channel->queue_bind('resize-picture', 'upload-pictures');

$channel->basic_consume('resize-picture', $consumer_tag,
                false, false, false, false, $consumer);

while(count($channel->callbacks)) {
  $channel->wait();
}
```

Consumer Code

```
$channel->exchange_declare('upload-pictures', 'fanout',
                false, true, false);

$channel->queue_declare('resize-picture', false, true,
                false, false);

$channel->queue_bind('resize-picture', 'upload-pictures');

$channel->basic_consume('resize-picture', $consumer_tag,
                false, false, false, false, $consumer);

while(count($channel->callbacks)) {
  $channel->wait();
}
```

Consumer Code

```
$channel->exchange_declare('upload-pictures', 'fanout',
                false, true, false);

$channel->queue_declare('resize-picture', false, true,
                false, false);

$channel->queue_bind('resize-picture', 'upload-pictures');

$channel->basic_consume('resize-picture', $consumer_tag,
                false, false, false, false, $consumer);

while(count($channel->callbacks)) {
  $channel->wait();
}
```

Consumer Code

```
$consumer = function($msg){

  $meta = json_decode($msg->body, true);

  resize_picture($meta['image_id'], $meta['image_path']);

  $msg->delivery_info['channel']->
    basic_ack($msg->delivery_info['delivery_tag']);
};
```

Consumer Code

```
$consumer = function($msg){

  $meta = json_decode($msg->body, true);

  resize_picture($meta['image_id'], $meta['image_path']);

  $msg->delivery_info['channel']->
    basic_ack($msg->delivery_info['delivery_tag']);
};
```

Consumer Code

```
$consumer = function($msg){

  $meta = json_decode($msg->body, true);

  resize_picture($meta['image_id'], $meta['image_path']);

  $msg->delivery_info['channel']->
    basic_ack($msg->delivery_info['delivery_tag']);
};
```

Consumer Code

```
$consumer = function($msg){

  $meta = json_decode($msg->body, true);

  resize_picture($meta['image_id'], $meta['image_path']);

  $msg->delivery_info['channel']->
    basic_ack($msg->delivery_info['delivery_tag']);
};
```

Consumer Code

```
$consumer = function($msg){

  $meta = json_decode($msg->body, true);

  resize_picture($meta['image_id'], $meta['image_path']);

  $msg->delivery_info['channel']->
    basic_ack($msg->delivery_info['delivery_tag']);
};
```

Scenario

Distributed Logging

Requirements

Requirements

- Several Web Servers

Requirements

- Several Web Servers
- Logic Separated by Module/Action

Requirements

- Several Web Servers
- Logic Separated by Module/Action
- Several Log Levels:

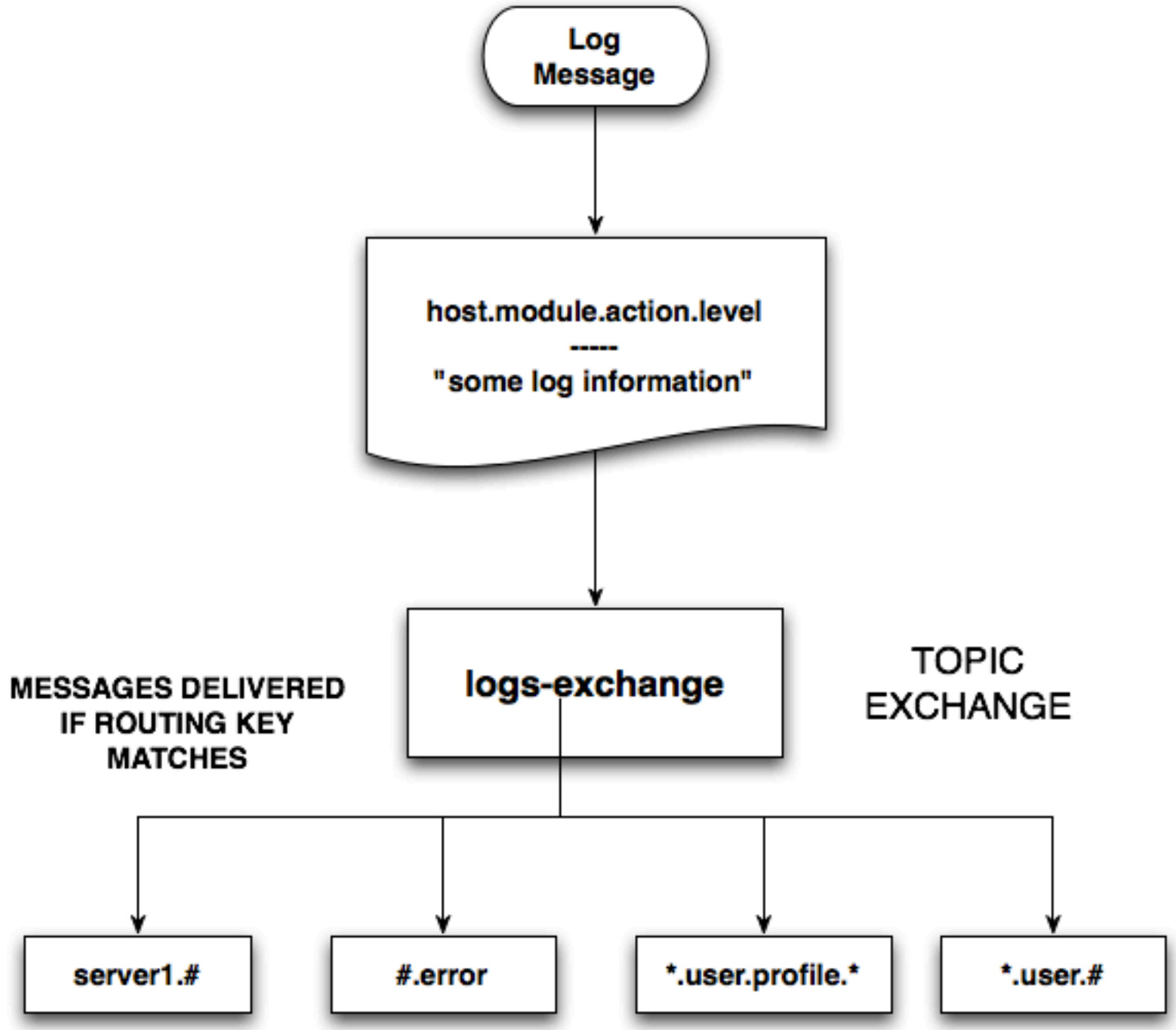
Requirements

- Several Web Servers
- Logic Separated by Module/Action
- Several Log Levels:
  - Info, Warning, Error

Requirements

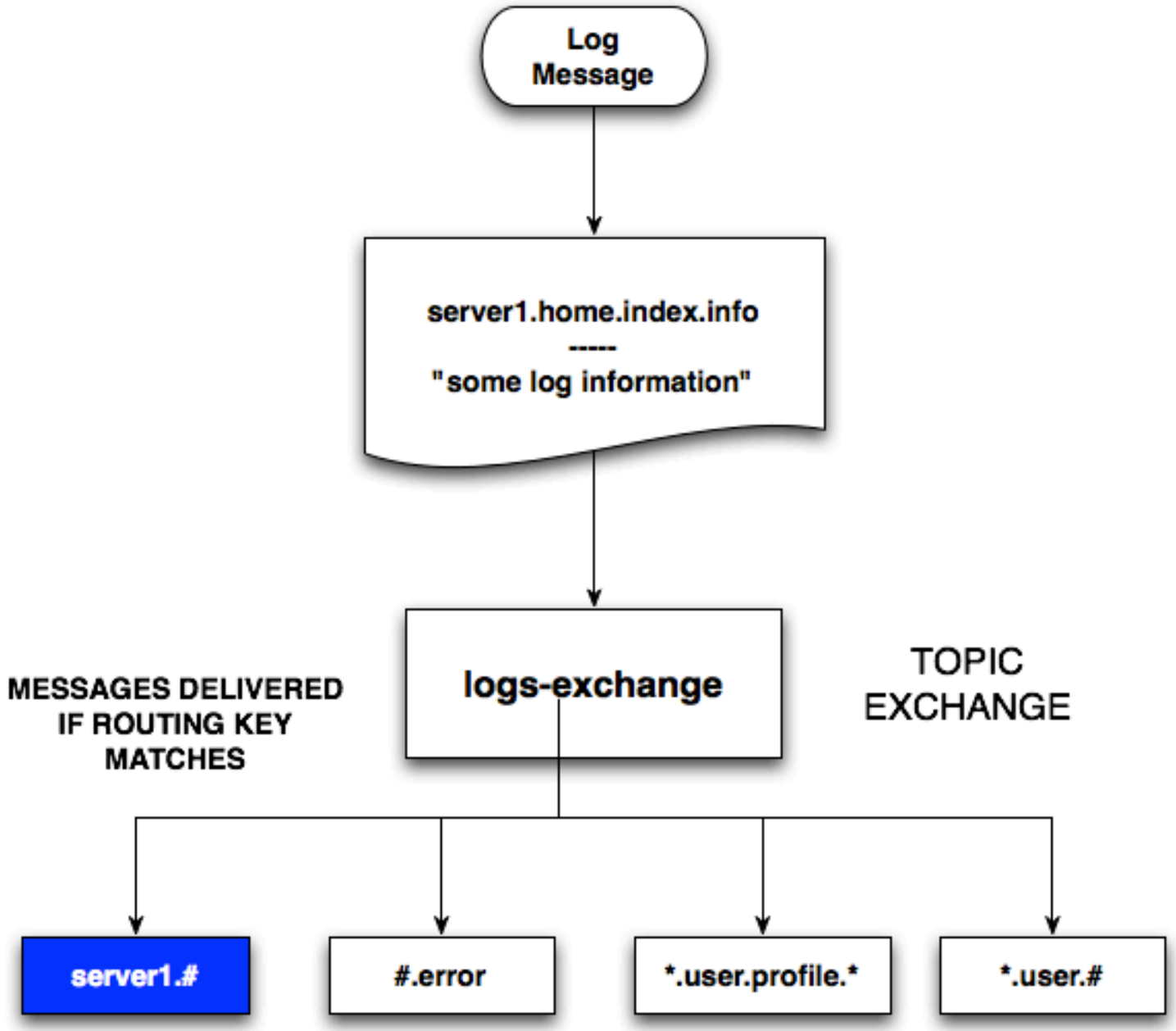
- Several Web Servers
- Logic Separated by Module/Action
- Several Log Levels:
  - Info, Warning, Error
- Add/Remove log listeners at will

Design


Log
Message
host.module.action.level

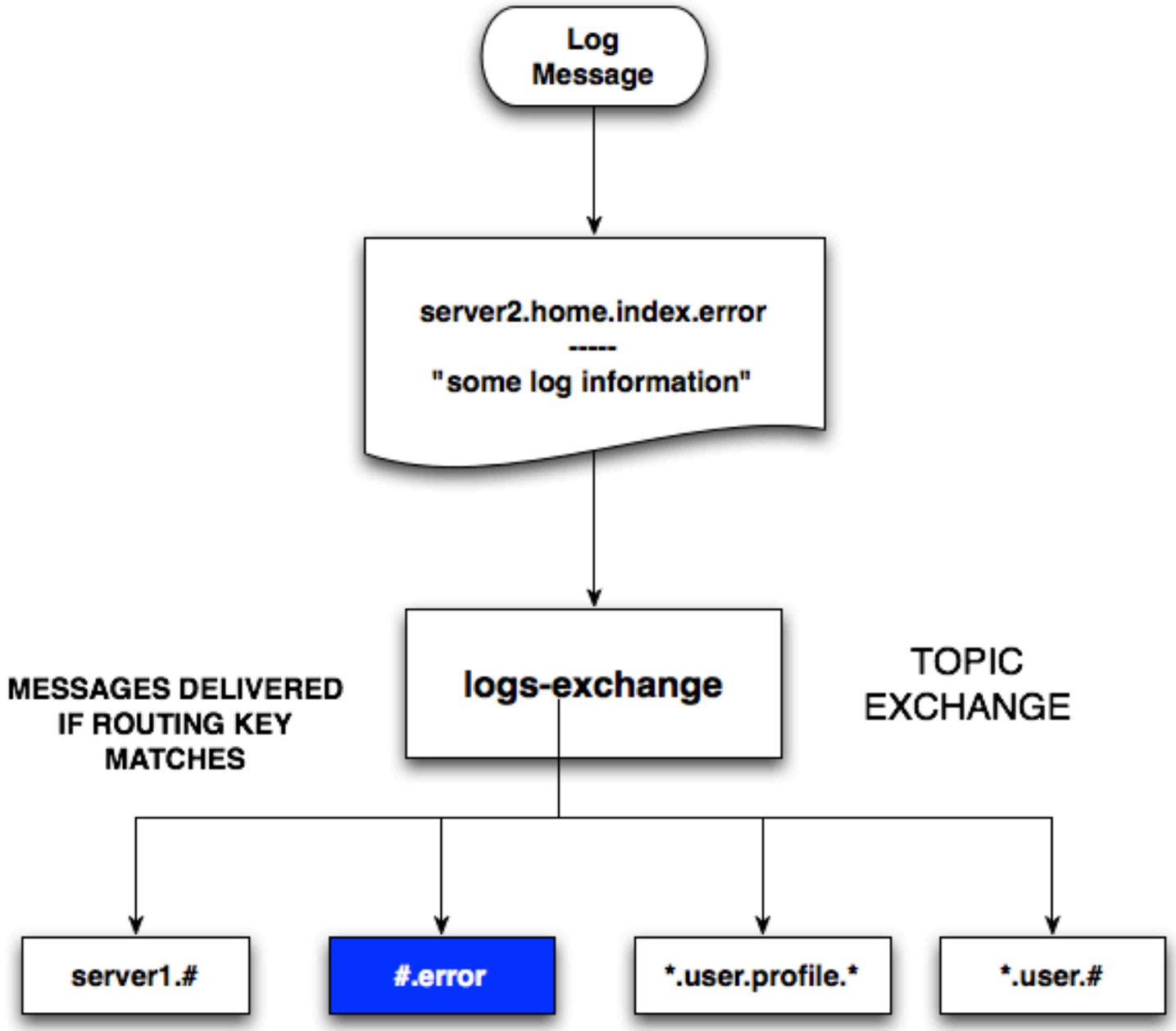
"some log information"
logs-exchange
MESSAGES DELIVERED
IF ROUTING KEY
MATCHES
TOPIC
EXCHANGE
server1.#
#.error
.user.profile.
*.user.#

Design


Log
Message
server1.home.index.info

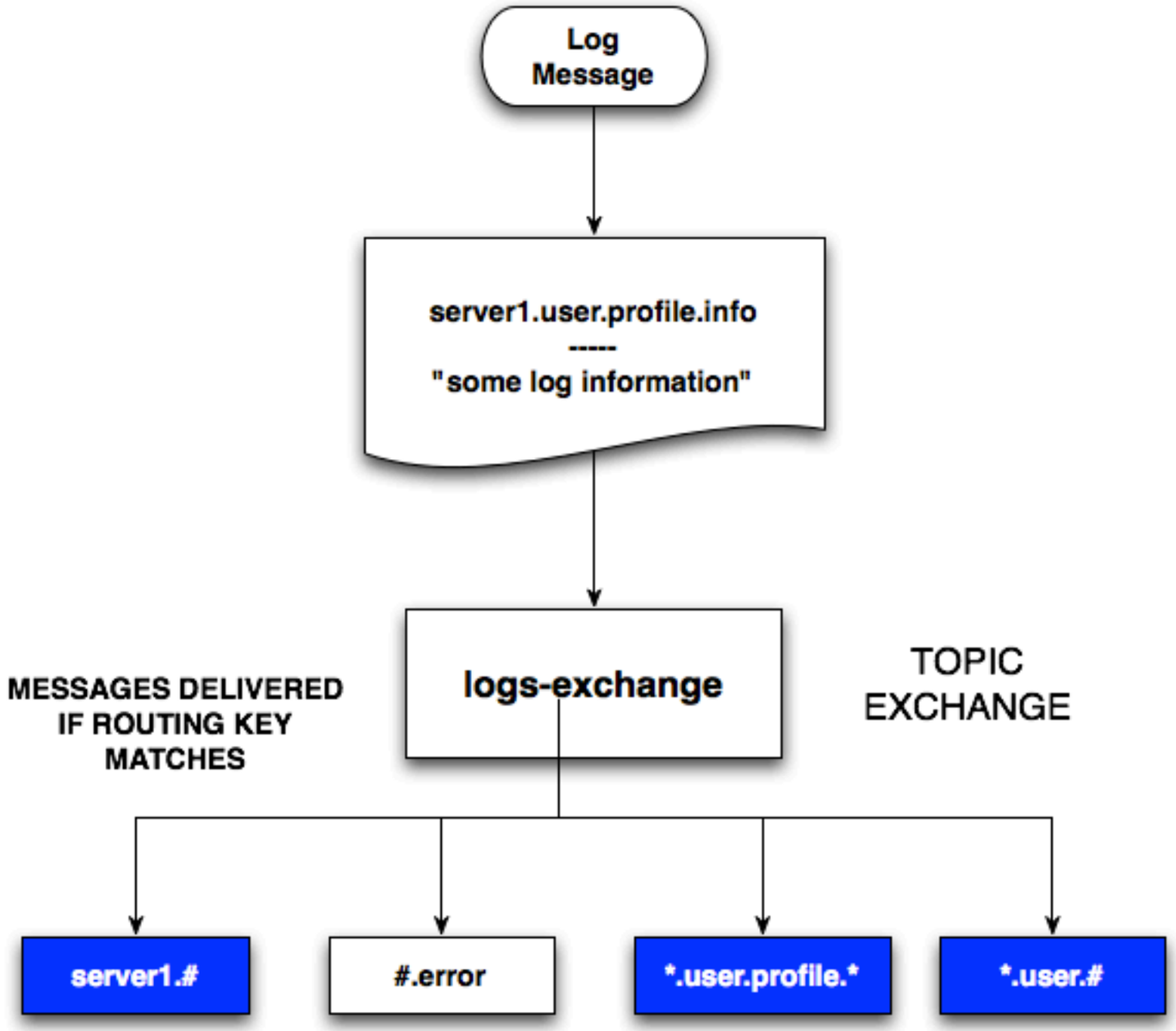
"some log information"
logs-exchange
MESSAGES DELIVERED
IF ROUTING KEY
MATCHES
TOPIC
EXCHANGE
server1.#
#.error
.user.profile.
*.user.#

Design


Log
Message
server2.home.index.error

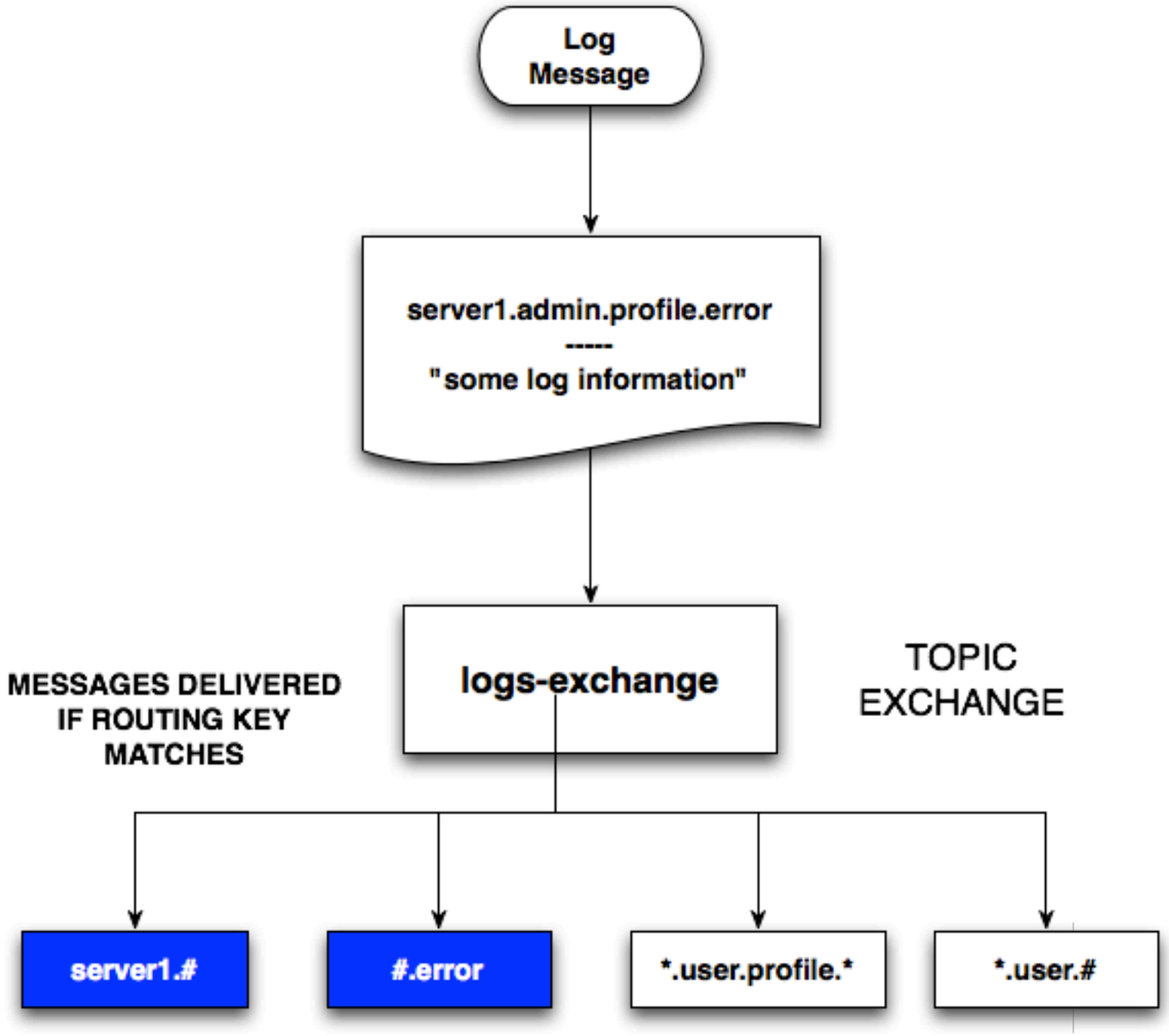
"some log information"
logs-exchange
MESSAGES DELIVERED
IF ROUTING KEY
MATCHES
TOPIC
EXCHANGE
server1.#
#.error
.user.profile.
*.user.#

Design


Log
Message
server1.user.profile.info

"some log information"
logs-exchange
MESSAGES DELIVERED
IF ROUTING KEY
MATCHES
TOPIC
EXCHANGE
server1.#
#.error
.user.profile.
*.user.#

Design


Log
Message
server1.admin.profile.error

"some log information"
logs-exchange
MESSAGES DELIVERED
IF ROUTING KEY
MATCHES
TOPIC
EXCHANGE
server1.#
#.error
.user.profile.
*.user.#

Publisher Code

```
$channel->exchange_declare('logs', 'topic', false,
                           true, false);

$msg = new AMQPMessage('some log message',
                       array('content_type' => 'text/plain'));

$channel->basic_publish($msg, 'logs',
                        'server1.user.profile.info');
```

Publisher Code

```
$channel->exchange_declare('logs', 'topic', false,
                           true, false);

$msg = new AMQPMessage('some log message',
                       array('content_type' => 'text/plain'));

$channel->basic_publish($msg, 'logs',
                       server1.user.profile.info');
```

Publisher Code

```
$channel->exchange_declare('logs', 'topic', false,
                           true, false);

$msg = new AMQPMessage('some log message',
                       array('content_type' => 'text/plain'));

$channel->basic_publish($msg, 'logs',
                        server1.user.profile.info');
```

Publisher Code

```
$channel->exchange_declare('logs', 'topic', false,
                           true, false);

$msg = new AMQPMessage('some log message',
                       array('content_type' => 'text/plain'));

$channel->basic_publish($msg, 'logs',
                        server1.user.profile.info');
```

Consumer Code

Get messages sent by host:

```
server1
```

Consumer Code

```
$channel->exchange_declare('logs', 'topic', false,
                           true, false);

$channel->queue_declare('server1-logs', false, true,
                        false, false);

$channel->queue_bind('server1-logs', 'logs', 'server1.#');
```

Consumer Code

```
$channel->exchange_declare('logs', 'topic', false,
                           true, false);

$channel->queue_declare('server1-logs', false, true,
                        false, false);

$channel->queue_bind('server1-logs', 'logs', 'server1.#');
```

Consumer Code

```
$channel->exchange_declare('logs', 'topic', false,
                           true, false);

$channel->queue_declare('server1-logs', false, true,
                        false, false);

$channel->queue_bind('server1-logs', 'logs', 'server1.#');
```

Consumer Code

```
$channel->exchange_declare('logs', 'topic', false,
                           true, false);

$channel->queue_declare('server1-logs', false, true,
                        false, false);

$channel->queue_bind('server1-logs', 'logs', 'server1.#');
```

Consumer Code

Get all `error` messages

Consumer Code

```
$channel->exchange_declare('logs', 'topic', false,
                           true, false);

$channel->queue_declare('error-logs', false, true,
                        false, false);

$channel->queue_bind('error-logs', 'logs', '#.error');
```

Consumer Code

```
$channel->exchange_declare('logs', 'topic', false,
                           true, false);

$channel->queue_declare('error-logs', false, true,
                        false, false);

$channel->queue_bind('error-logs', 'logs', '#.error');
```

Consumer Code

```
$channel->exchange_declare('logs', 'topic', false,
                           true, false);

$channel->queue_declare('error-logs', false, true,
                        false, false);

$channel->queue_bind('error-logs', 'logs', '#.error');
```

Consumer Code

```
$channel->exchange_declare('logs', 'topic', false,
                           true, false);

$channel->queue_declare('error-logs', false, true,
                        false, false);

$channel->queue_bind('error-logs', 'logs', '#.error');
```

Why RabbitMQ?

RabbitMQ

- Enterprise Messaging System
- Open Source MPL
- Written in Erlang/OTP
- Commercial Support

Features

- Reliable and High Scalable
- Easy To install
- Easy To Cluster
- Runs on: Windows, Solaris, Linux, OSX
- AMQP 0.8 - 0.9.1

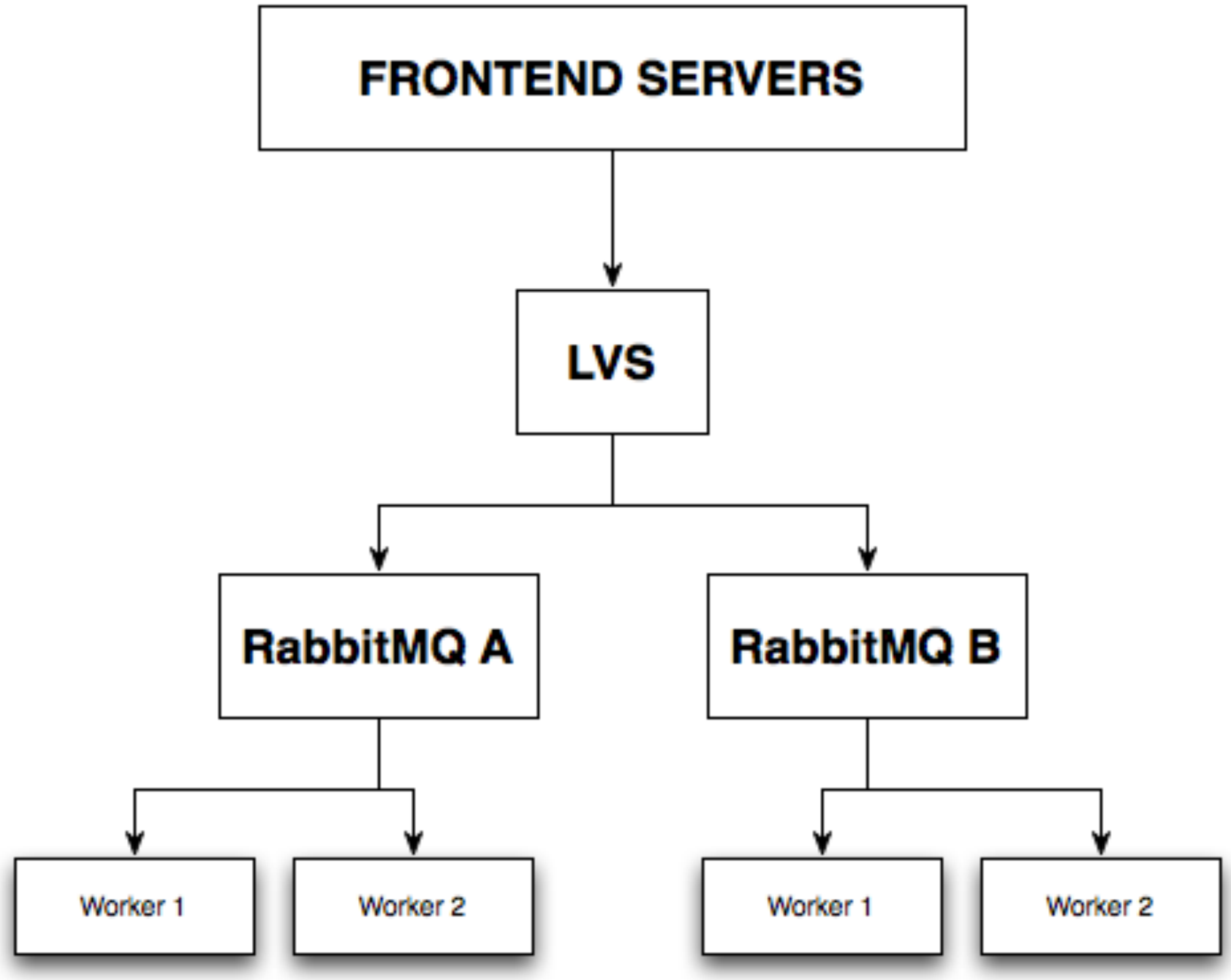
Client Libraries

- Java
- .NET/C#
- Erlang
- Ruby, Python, PHP, Perl, AS3, Lisp, Scala, Clojure, Haskell

Docs/Support

- http://www.rabbitmq.com/documentation.html
- http://dev.rabbitmq.com/wiki/
- #rabbitmq at irc.freenode.net
- http://www.rabbitmq.com/email-archive.html

One Setup for HA



Conclusion

Conclusion

- Flexibility

Conclusion

- Flexibility
- Scalability

Conclusion

- Flexibility
- Scalability
- Interoperability

Conclusion

- Flexibility
- Scalability
- Interoperability
- Reduce Ops

Questions?

Thanks!

Álvaro Videla

http://twitter.com/old_sound

http://github.com/videlalvaro

http://github.com/tnc

http://www.slideshare.net/old_sound