

# **RabbitMQ**

## **Sentimental reports from the Erlang frontline**

**Alexis Richardson**  
**Matthew Sackman**  
**Tony Garnock-Jones**

**Erlang Factory June 25th 2009**



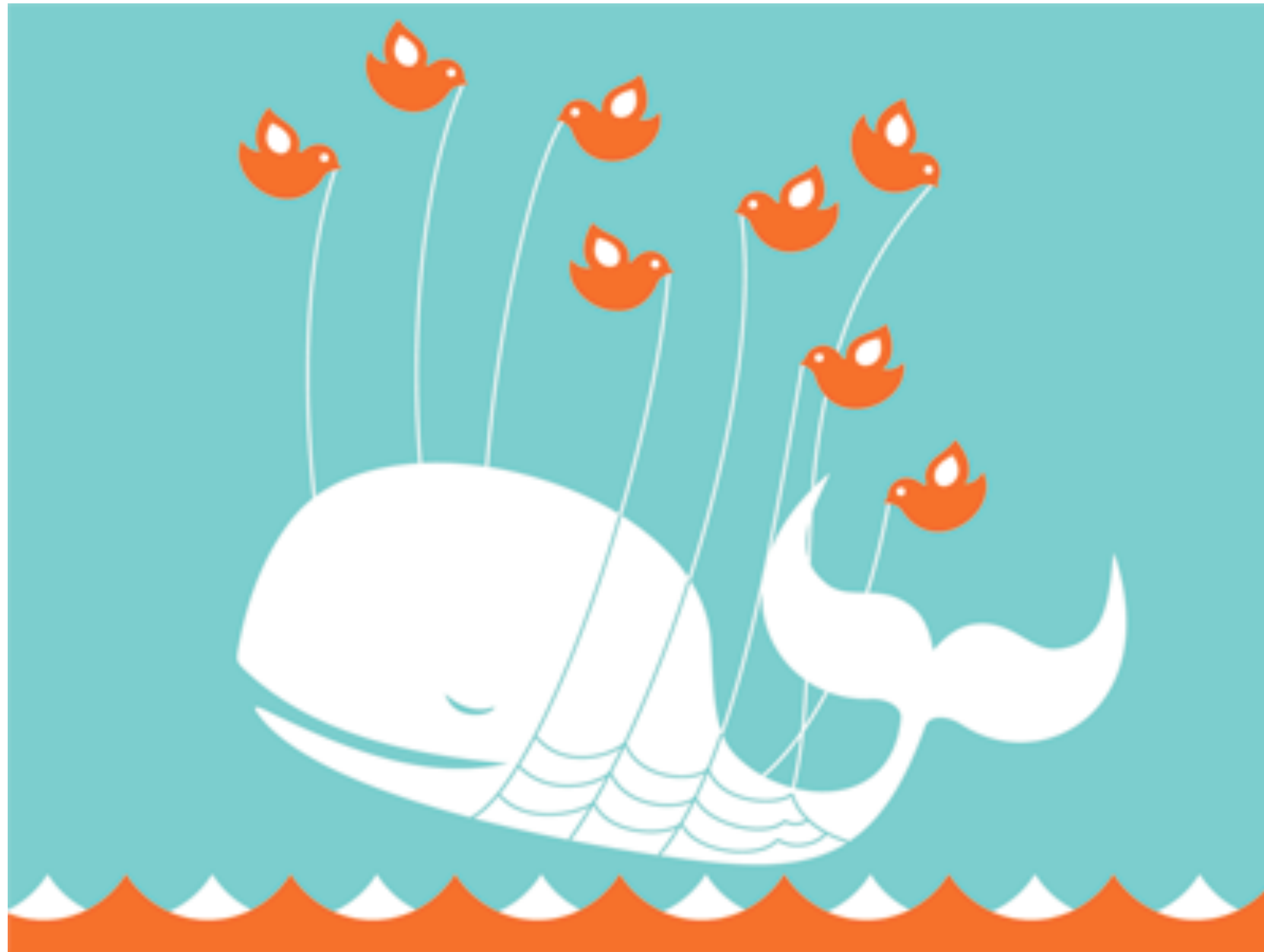
RabbitMQ is a messaging server that just works!



Im in yr serverz,  
queueing yr messagez



You might need messaging if ... you need to scale





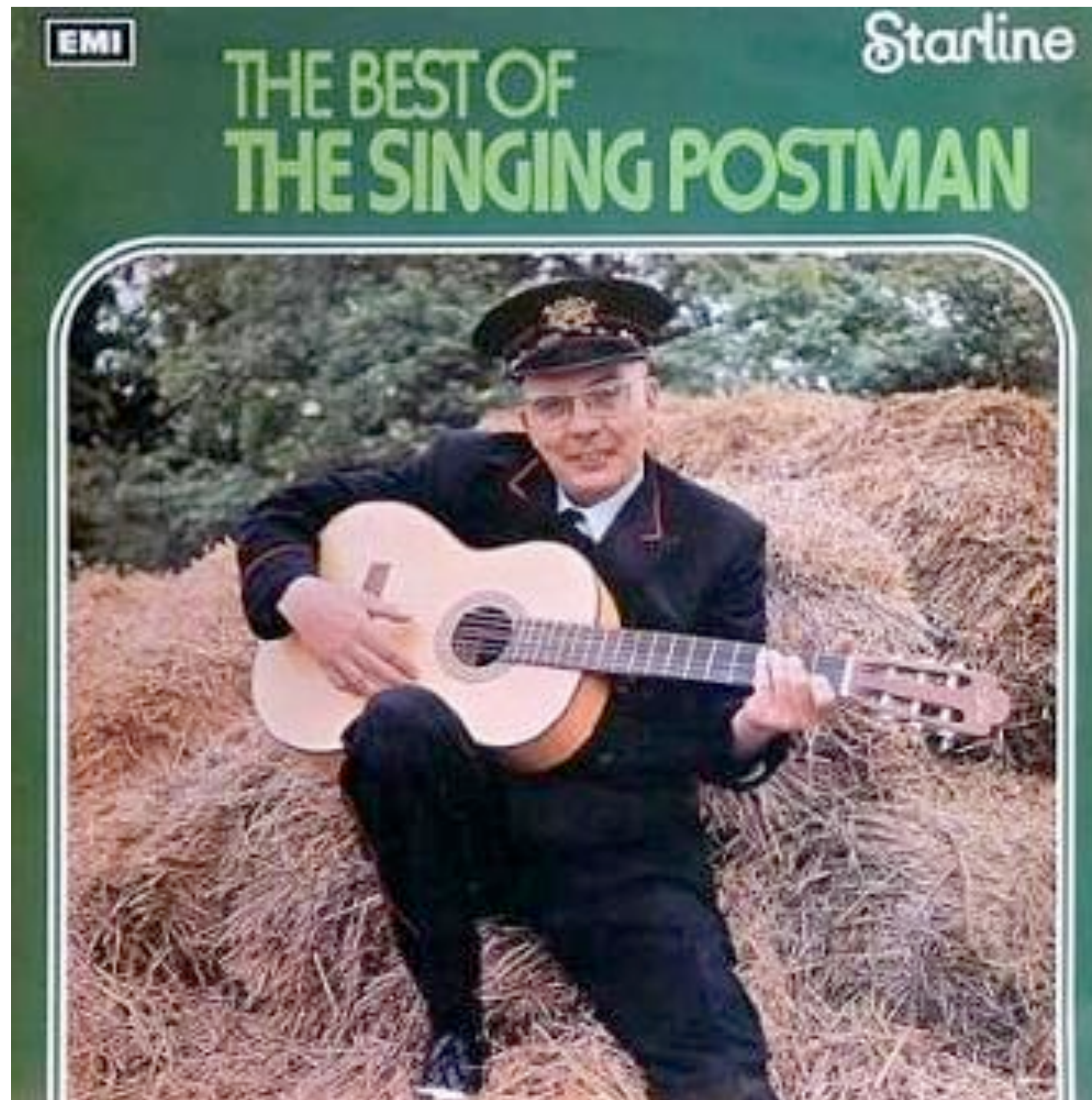
You might need messaging if ... you need to monitor data feeds



(CC) Kishore Nagarigari



You might need messaging if ... you need a message delivered responsibly





You might need messaging if ... you need things done in order



(CC) David Mach

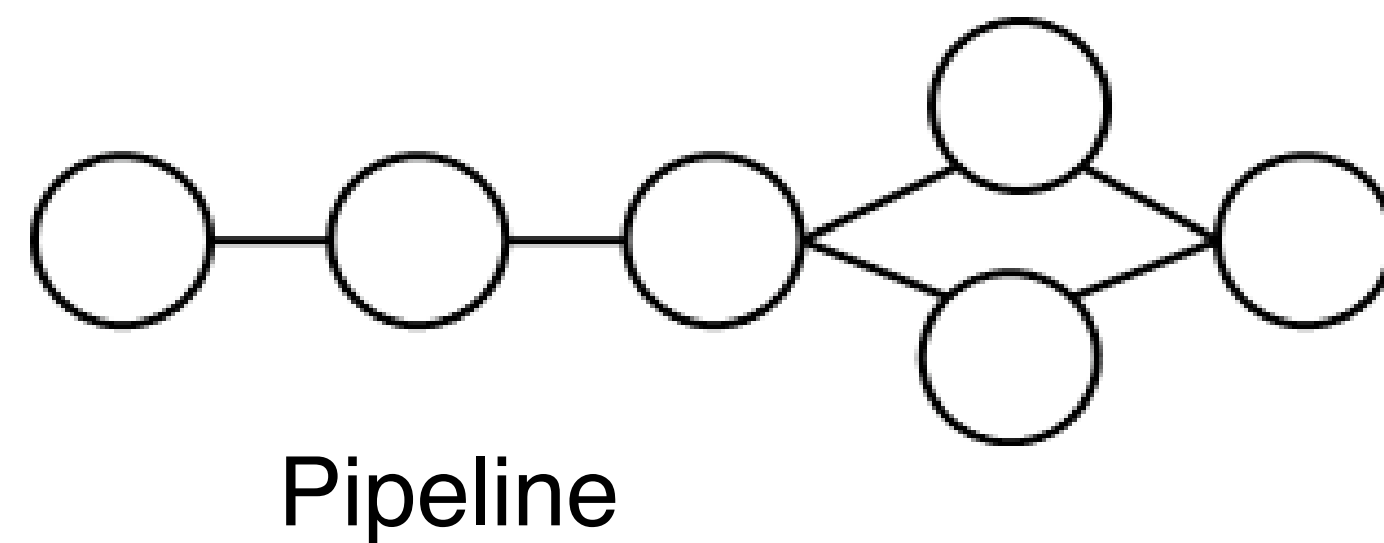
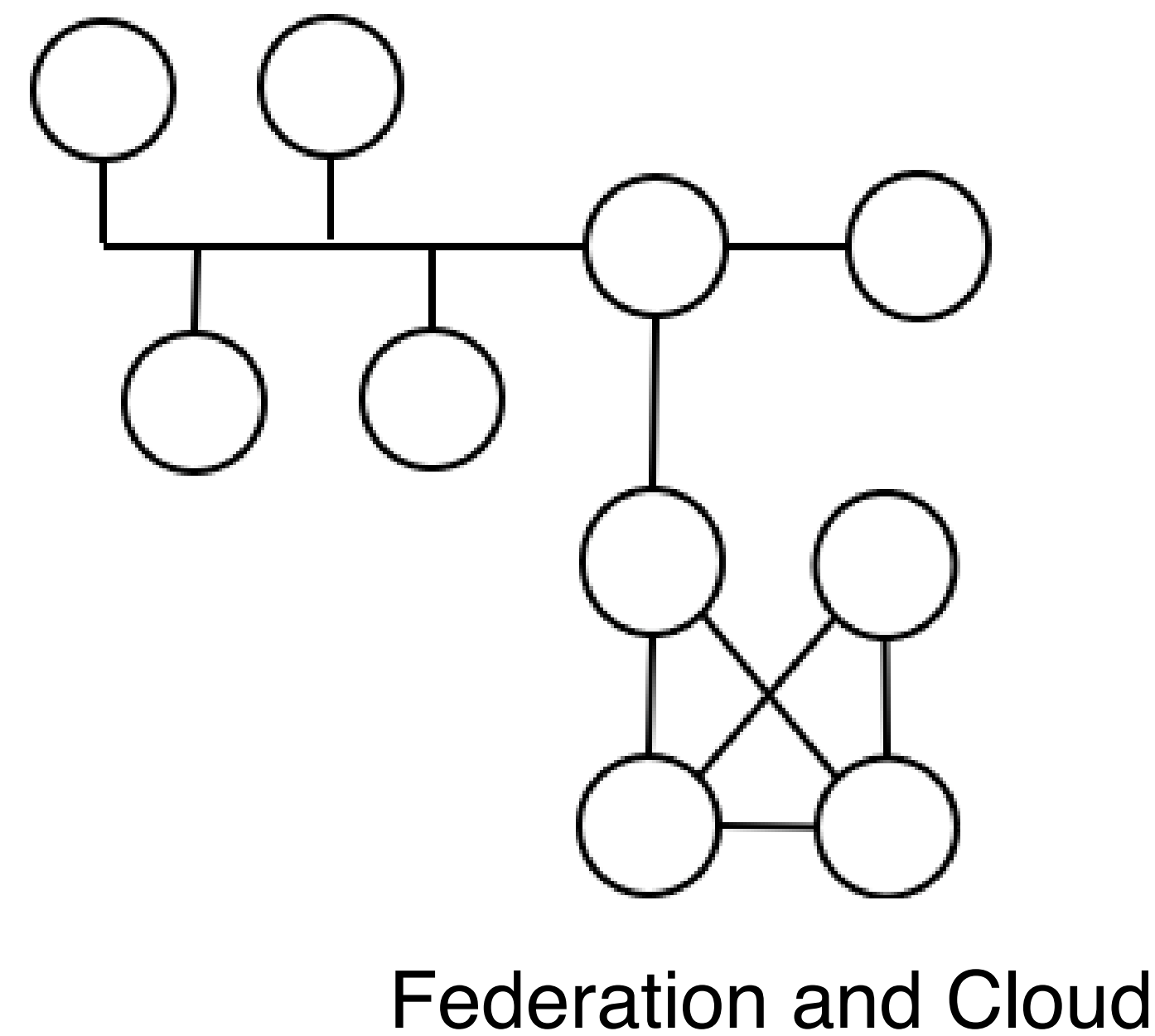
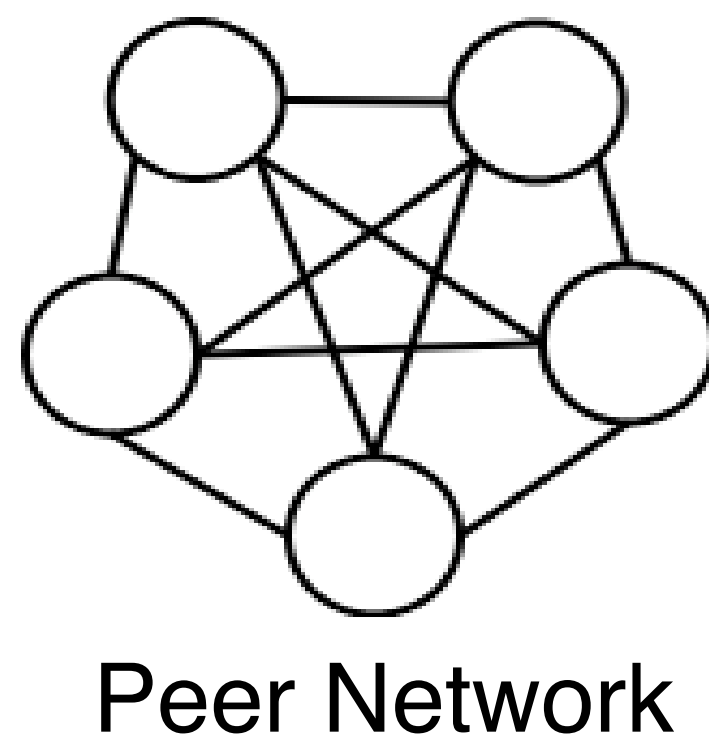
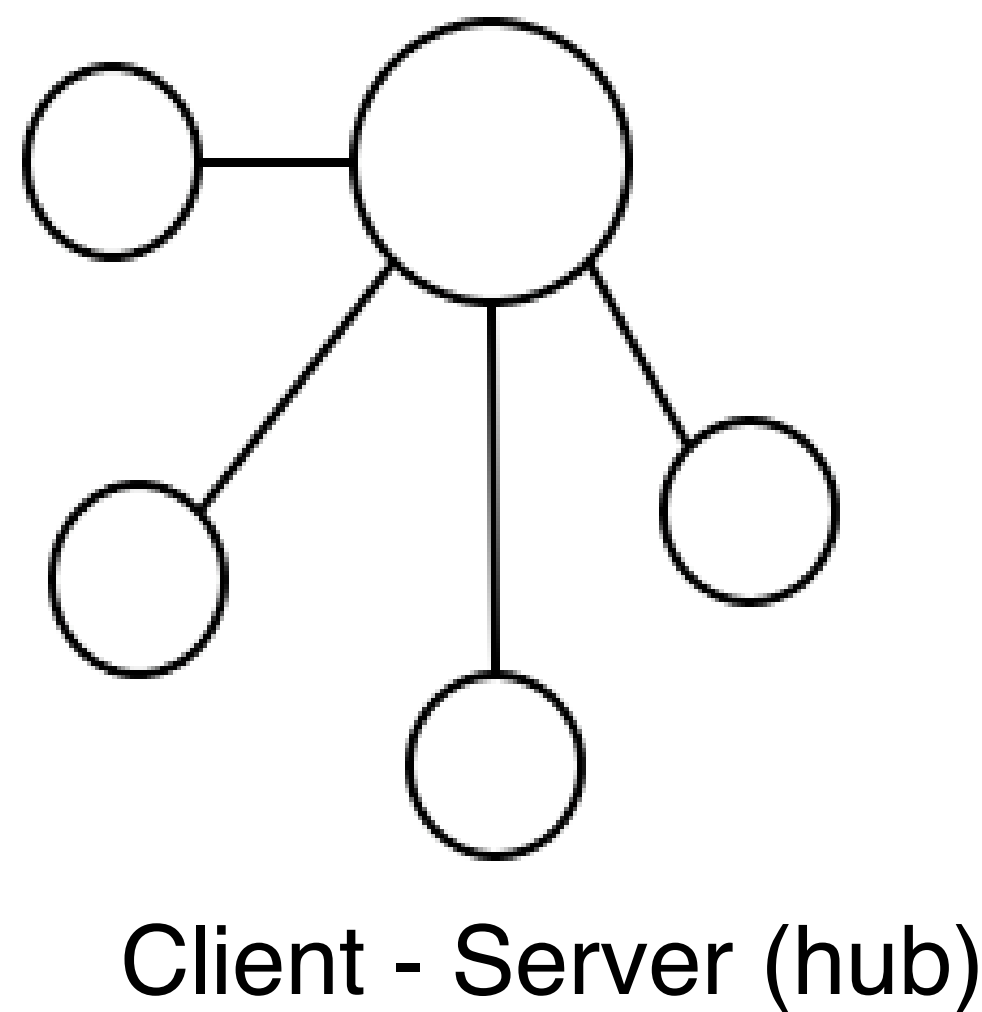
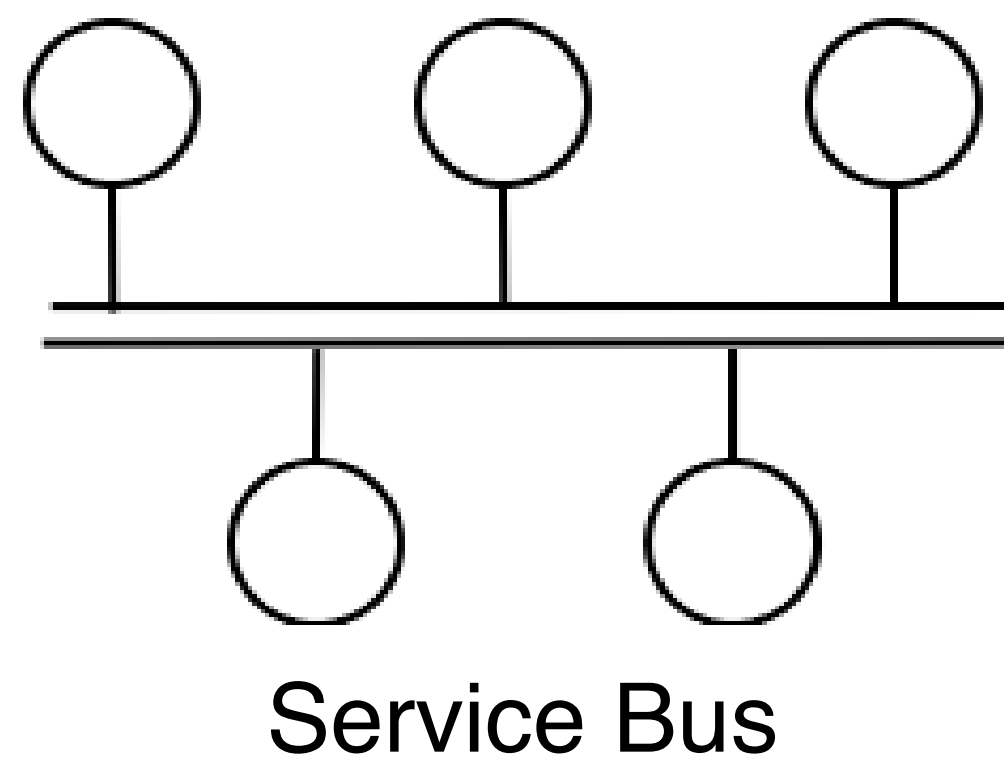


You might need messaging if ... you are using the cloud



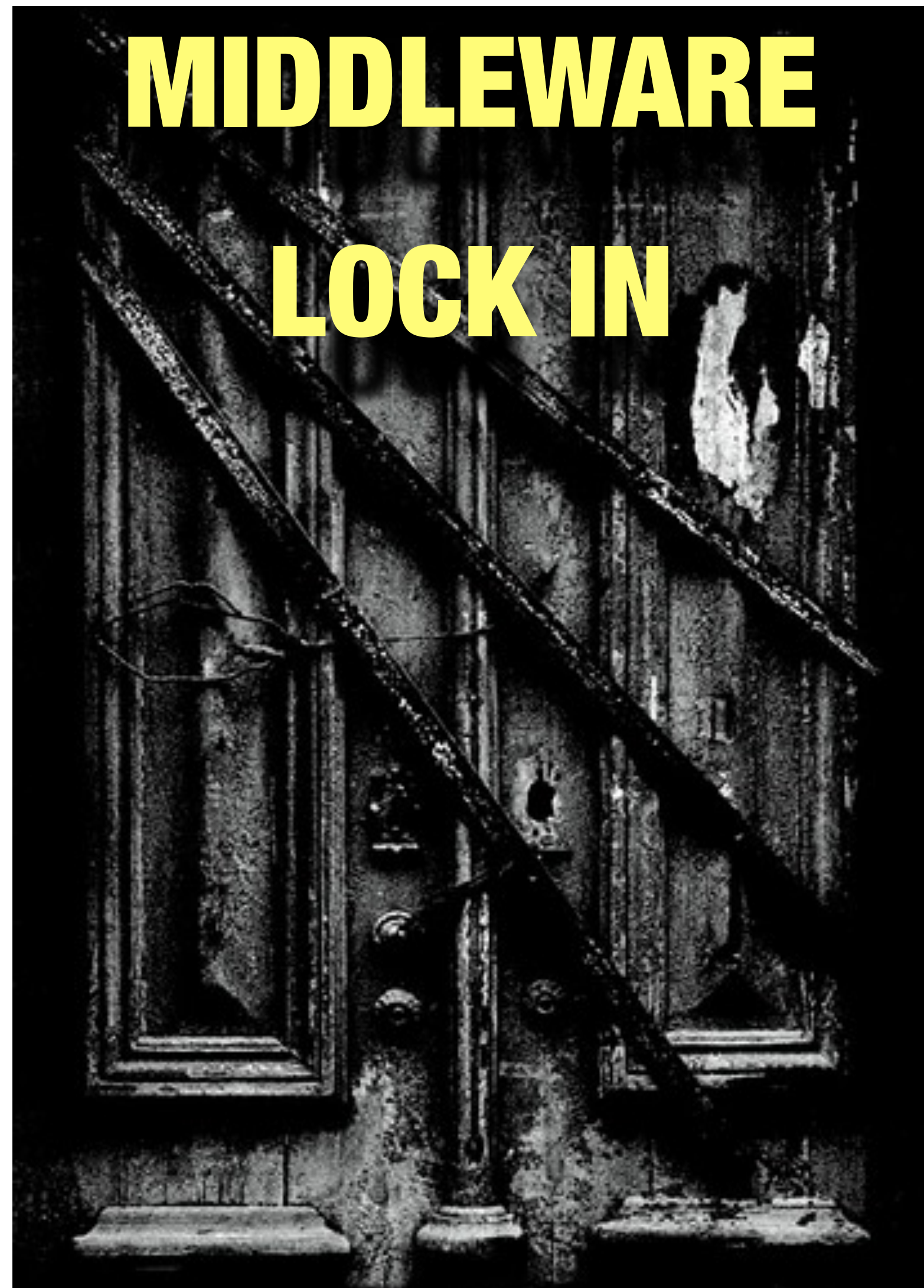


# Messaging is everywhere





Don't be evil





# When middleware goes bad

---

complex, proprietary, closed

requires installation and customisation

integration services from consultants with  
knowledge of many platforms or languages

then maintenance is done by the customer

which is then followed by system aging, bloat,  
and eventual heat death



Beware of lock in





# Meet the good guys





# OPEN INTERNET PROTOCOLS - TCP, SCTP, HTTP, SMTP - EPIC WIN

---

- 🔌 simple
- 🔌 standard
- 🔌 ubiquitous substrate
- 🔌 no customisation needed
- 🔌 no integration required from consultants
- 🔌 maintenance is done by the vendor
- 🔌 proven to outlast the lifetime of the average software company
- 🔌 (and many banks)
- 🔌 scales



# OPEN INTERNET PROTOCOLS - TCP, SCTP, HTTP, SMTP - EPIC WIN

- 🔌 simple
- 🔌 standard
- 🔌 ubiquitous substrate
- 🔌 no customization needed

## **AMQP - Advanced Message Queuing Protocol**




- 🔌 no integration required from consultants
- 🔌 maintenance is done by the vendor
- 🔌 proven to outlast the lifetime of the average software company
- 🔌 (and many banks)
- 🔌 scales





# The world is getting more open every day

---

## Then:

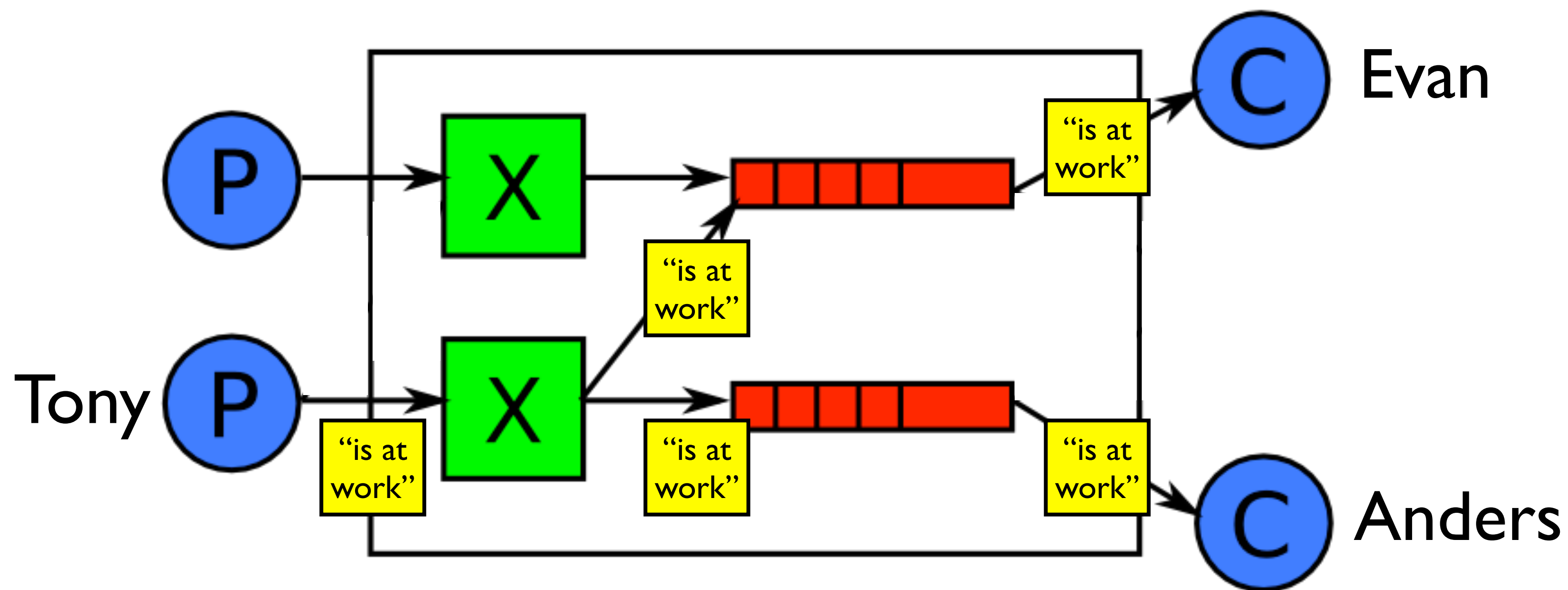
-  Imagine if we had no TCP and had to use 'IBM NetSphere'
-  Imagine if we had no HTTP and had to use 'Microsoft Home Network'
-  Imagine if we had no SMTP email and had to pay per message like SWIFT

## Now:

-  Imagine if we had no XMPP chat and had to use .. oh, wait a minute :-(
-  AMQP - business messaging - like email but you can send money over it



# AMQP combines PUBSUB with QUEUES



Evan and Anders want to follow what Tony says. They can follow Tony by binding their queues to a RabbitMQ exchange, using the pattern "tony".

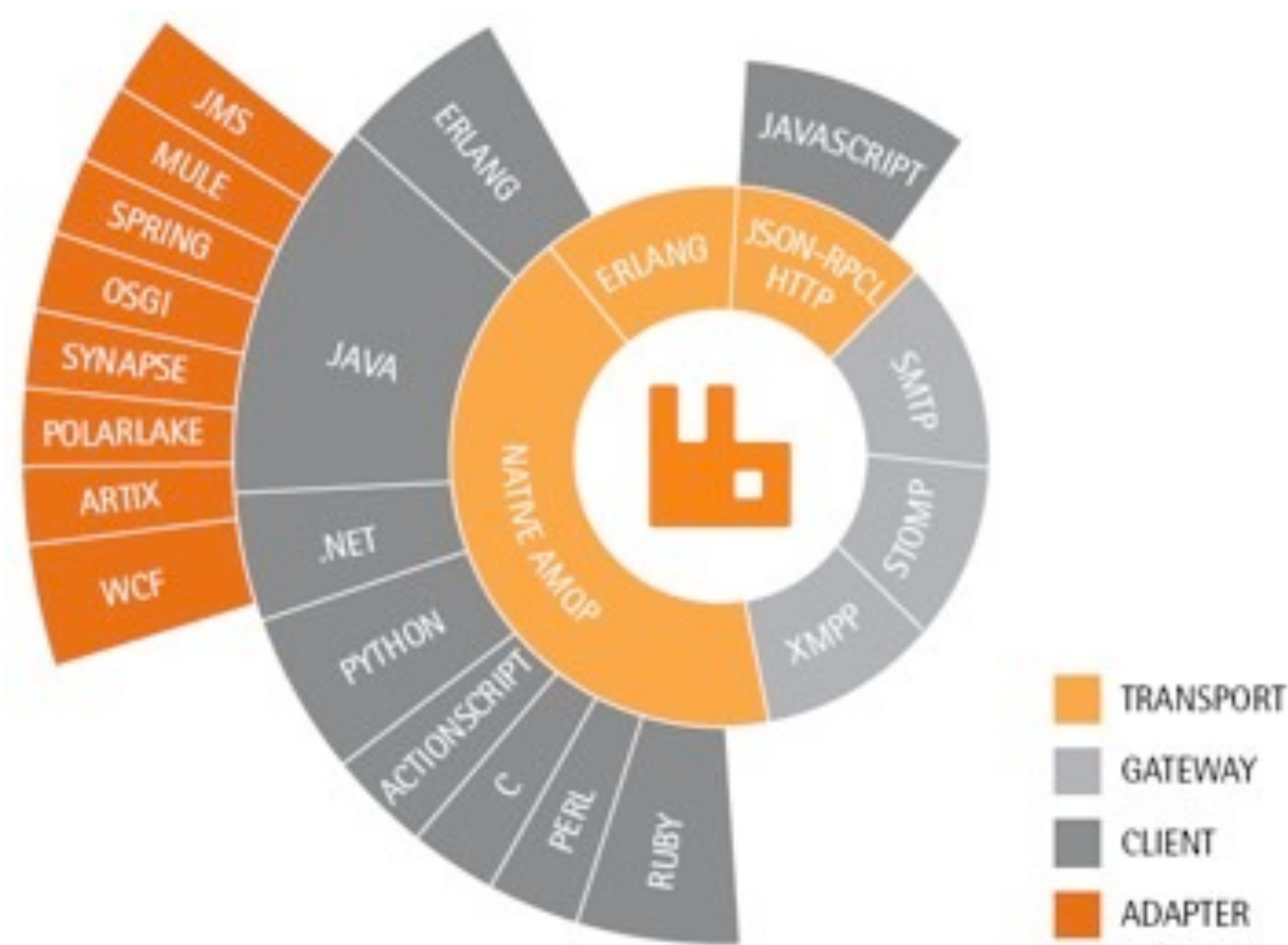
Tony publishes the message "is at work" to the same RabbitMQ exchange, using the routing key "tony".

The exchange updates Evan's and Anders' queues accordingly, for subsequent consumption by their client applications.

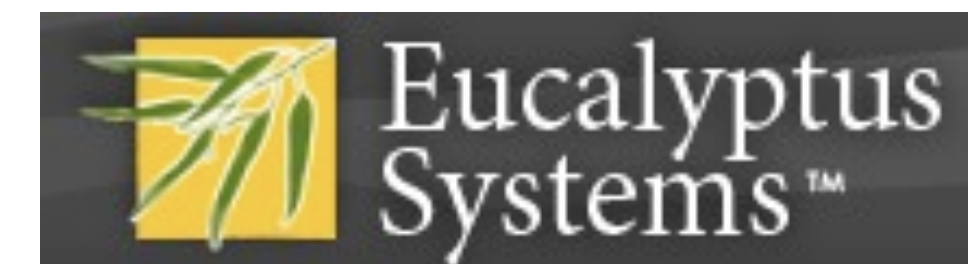
Many other patterns are possible e.g. for filtering by topic similar to this: <http://jchris.mfdz.com/posts/64>



# Erlang inside + your favourite language or protocol on the outside





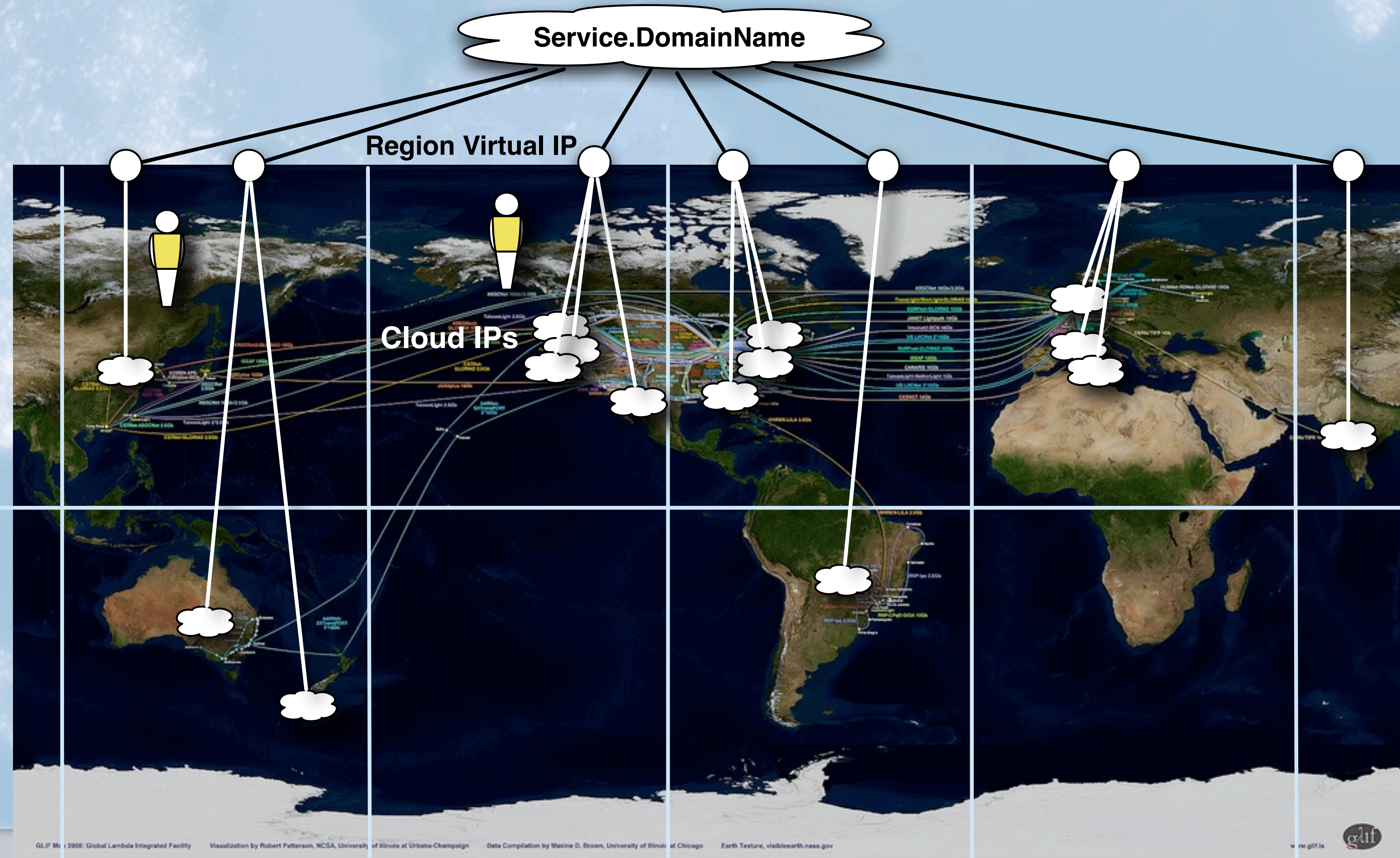


**RabbitMQ is distributed in Ubuntu**





# Use Case: Ocean Observatories “Global Twitter for Data”





# USE CASE - BBC Feeds Hub “streaming content management”

## Introducing... BBC Feeds Hub

---

Post categories: [Radio](#)

[Alan Ogilvie](#) | 09:39 UK time, Wednesday, 29 April 2009

---

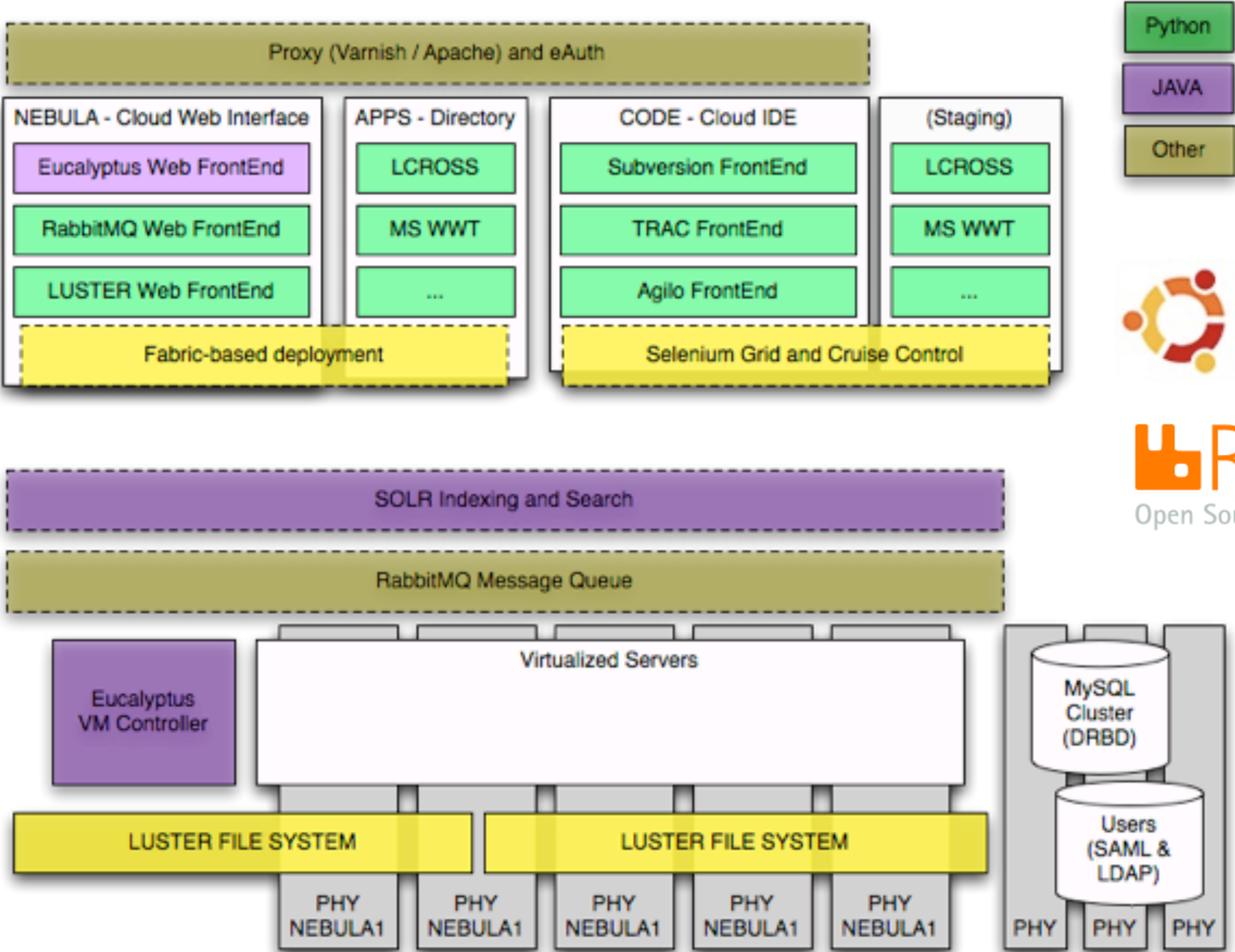
Here in the Distribution Technologies team for BBC A&M Interactive, we look at how best to distribute media and metadata across A&M for current and future platforms; we also look at how to syndicate our content to external partnerships and the public.

Feeds are a great way of reusing content more easily in an automated way. You're probably familiar with the example of RSS feeds from blogs or podcasts, which save you having to visit different sites to collect the information you want. In A&M we reuse and reversion many different feed formats, not just RSS, to save us duplicating work across different platforms.

Feeds Hub is one of our new projects focusing on registering, reusing and reversioning data feeds.

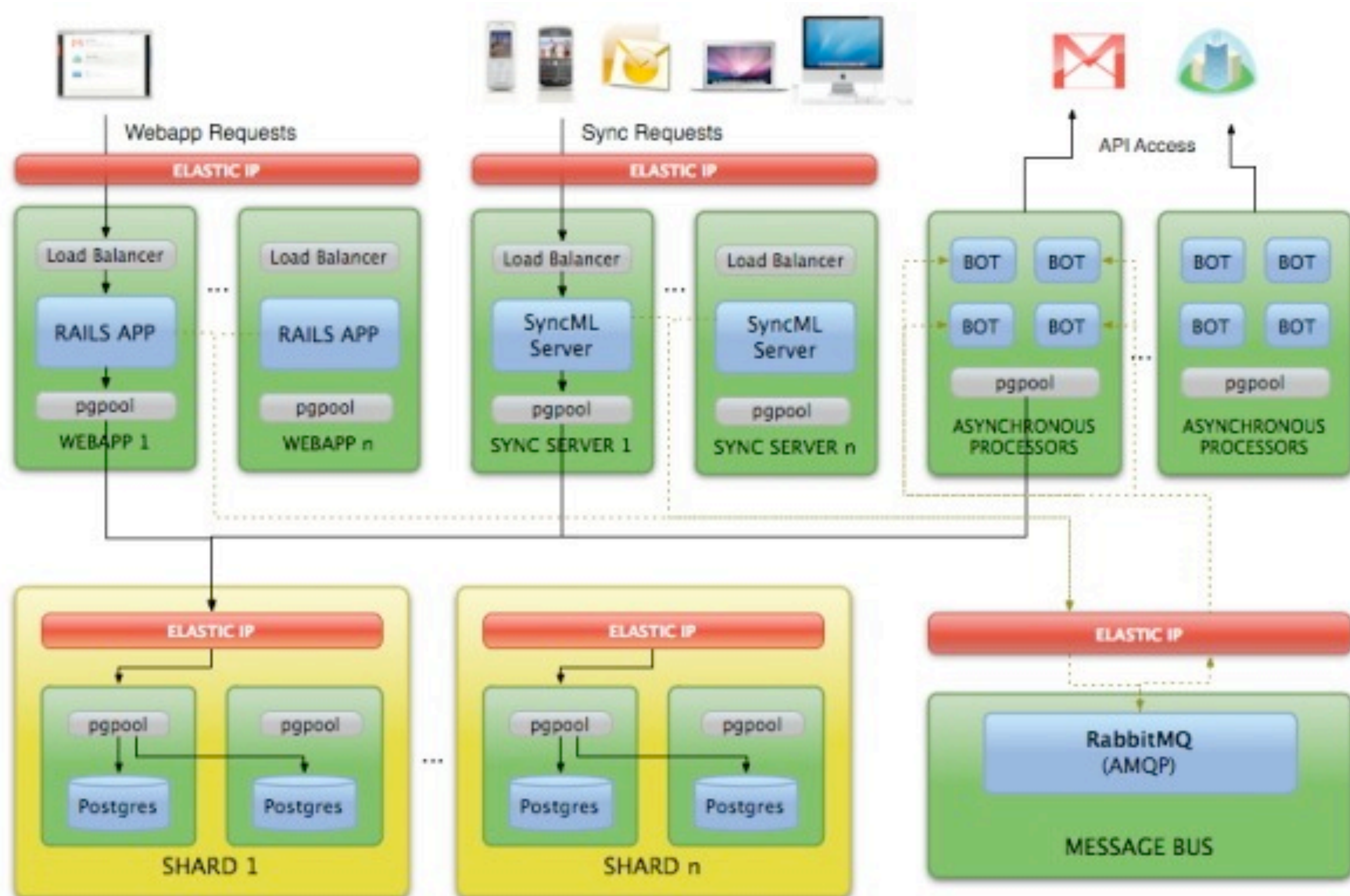


# USE CASE - NASA Cloud (“Nebula”)





# USE CASE - Soocial.com is 100% on Amazon





## USE CASE - Second Life





Get stuck in!





# SERVING RABBIT ON A (DISK) PLATTER

Matthew Sackman<sup>1</sup>

<sup>1</sup>LShift



# NORMAL ACADEMIC TALK

$$\begin{array}{l}
 (sr, (rr, rc)) = h \downarrow_4 (p, c, k) \quad q = h \downarrow_3 (c, dual(h, c, k)) \\
 sr \notin \text{dom}(q) \quad n = |h \downarrow_2 (c, r)| \quad q' = q[sr \mapsto (n, \{0 \mapsto v\})] \\
 k \in \text{channels}(h \downarrow_1 (c), r) \quad h' \downarrow_3 = h \downarrow_3 [(c, dual(h, c, k)) \mapsto q'] \\
 h' \downarrow_4 = h \downarrow_4 [(p, c, k) \mapsto (sr + 1, (rr, rc))] \quad h' \downarrow_j = h \downarrow_j, j \in \{1, 2, 5, 6\} \\
 \hline
 h, p[E\langle \text{send } c \ k \ v \rangle] \longrightarrow h', p[E\langle () \rangle]
 \end{array}
 \quad \text{SEND-0}$$
  

$$\begin{array}{l}
 (sr, (rr, rc)) = h \downarrow_4 (p, c, k) \quad q = h \downarrow_3 (c, dual(h, c, k)) \\
 (n, qr) = q(sr) \quad q' = q[sr \mapsto (n - 1, qr \triangleleft v)] \\
 k \in \text{channels}(h \downarrow_1 (c), r) \quad h' \downarrow_3 = h \downarrow_3 [(c, dual(h, c, k)) \mapsto q'] \\
 h' \downarrow_4 = h \downarrow_4 [(p, c, k) \mapsto (sr + 1, (rr, rc))] \quad h' \downarrow_j = h \downarrow_j, j \in \{1, 2, 5, 6\} \\
 \hline
 h, p[E\langle \text{send } c \ k \ v \rangle] \longrightarrow h', p[E\langle () \rangle]
 \end{array}
 \quad \text{SEND-N}$$

# NORMAL ACADEMIC TALK - ADJUSTED FOR HACKERS

onePlusOne (1, 1) ->

$$\begin{array}{l}
 \text{\% \%} \quad (sr, (rr, rc)) = h \downarrow_4 (p, c, k) \quad q = h \downarrow_3 (c, dual(h, c, k)) \\
 \text{\% \%} \quad sr \notin \text{dom}(q) \quad n = |h \downarrow_2 (c, r)| \quad q' = q[sr \mapsto (n, \{0 \mapsto v\})] \\
 \text{\% \%} \quad k \in \text{channels}(h \downarrow_1 (c), r) \quad h' \downarrow_3 = h \downarrow_3 [(c, dual(h, c, k)) \mapsto q'] \\
 \text{\% \%} \quad h' \downarrow_4 = h \downarrow_4 [(p, c, k) \mapsto (sr + 1, (rr, rc))] \quad h' \downarrow_j = h \downarrow_j, j \in \{1, 2, 5, 6\}
 \end{array}
 \quad \text{SEND-0}$$

$$h, p[E\langle \text{send } c \ k \ v \rangle] \longrightarrow h', p[E\langle () \rangle]$$

$$\begin{array}{l}
 \text{\% \%} \quad (sr, (rr, rc)) = h \downarrow_4 (p, c, k) \quad q = h \downarrow_3 (c, dual(h, c, k)) \\
 \text{\% \%} \quad (n, qr) = q(sr) \quad q' = q[sr \mapsto (n - 1, qr \triangleleft v)] \\
 \text{\% \%} \quad k \in \text{channels}(h \downarrow_1 (c), r) \quad h' \downarrow_3 = h \downarrow_3 [(c, dual(h, c, k)) \mapsto q'] \\
 \text{\% \%} \quad h' \downarrow_4 = h \downarrow_4 [(p, c, k) \mapsto (sr + 1, (rr, rc))] \quad h' \downarrow_j = h \downarrow_j, j \in \{1, 2, 5, 6\}
 \end{array}
 \quad \text{SEND-N}$$

$$h, p[E\langle \text{send } c \ k \ v \rangle] \longrightarrow h', p[E\langle () \rangle]$$

2.



## WHAT IS RABBITMQ?

- An AMQP *broker*
- Written entirely in Erlang
- Only about 14k lines of Erlang

## WHAT IS RABBITMQ?

- An AMQP *broker*
- Written entirely in Erlang
- Only about 14k lines of Erlang

## WHAT IS AMQP?



## WHAT IS RABBITMQ?

- An AMQP *broker*
- Written entirely in Erlang
- Only about 14k lines of Erlang

## WHAT IS AMQP?

- Protocol for dynamically configurable message routing platform

## WHAT IS RABBITMQ?

- An AMQP *broker*
- Written entirely in Erlang
- Only about 14k lines of Erlang

## WHAT IS AMQP?

- Protocol for dynamically configurable message routing platform
- Written entirely by committee



## WHAT IS RABBITMQ?

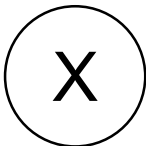
- An AMQP *broker*
- Written entirely in Erlang
- Only about 14k lines of Erlang

## WHAT IS AMQP?

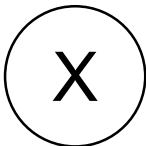
- Protocol for dynamically configurable message routing platform
- Written entirely by committee
- Only about 14k lines of text

# AMQP KEY CONCEPTS



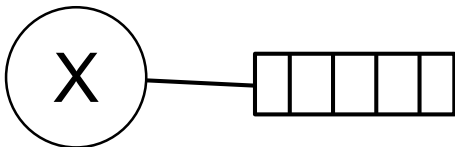


# AMQP KEY CONCEPTS

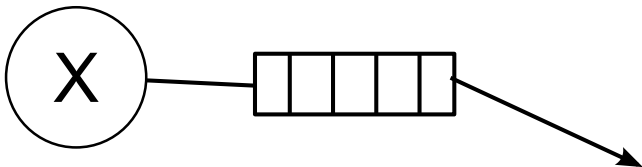




# AMQP KEY CONCEPTS

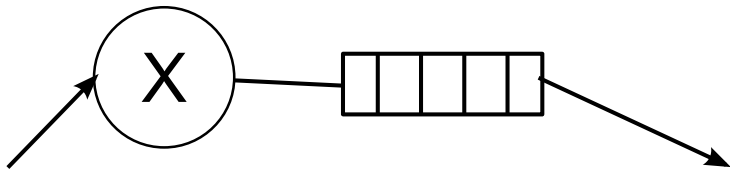


# AMQP KEY CONCEPTS

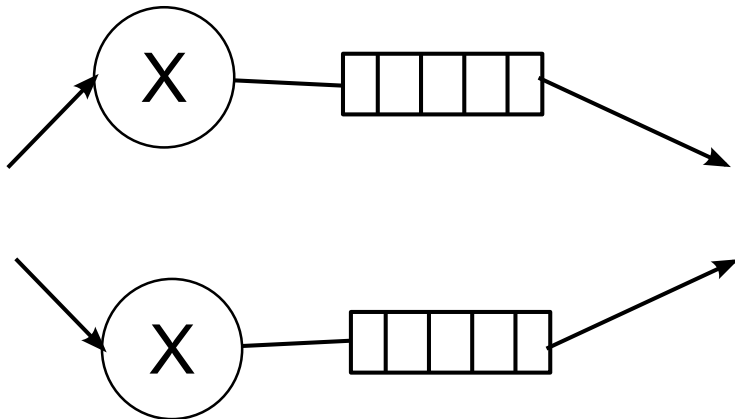




# AMQP KEY CONCEPTS

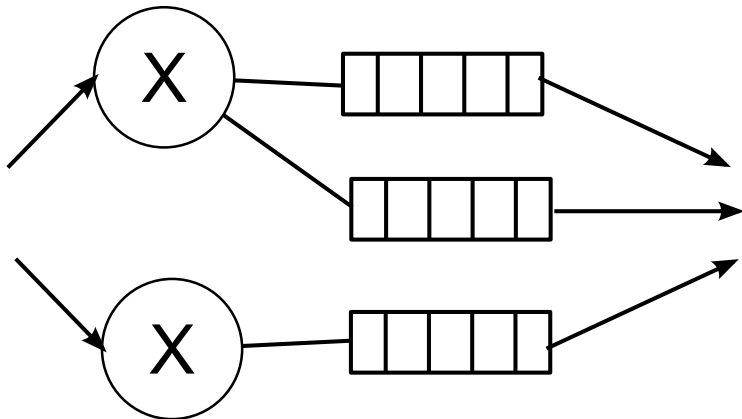


# AMQP KEY CONCEPTS

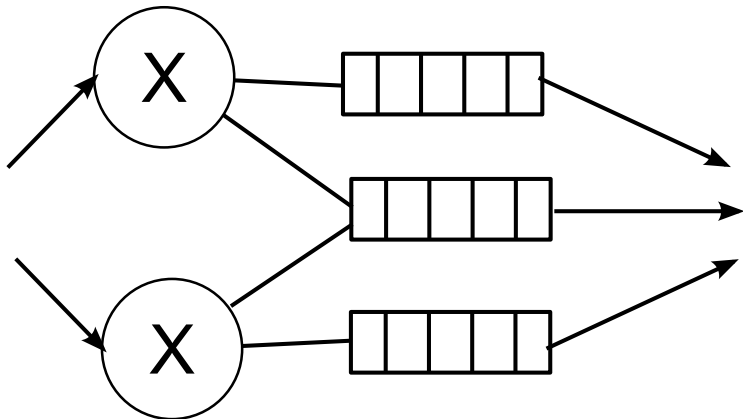




# AMQP KEY CONCEPTS

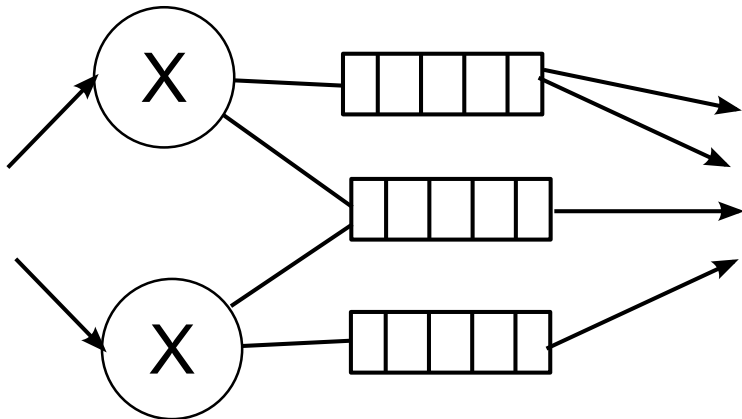


# AMQP KEY CONCEPTS

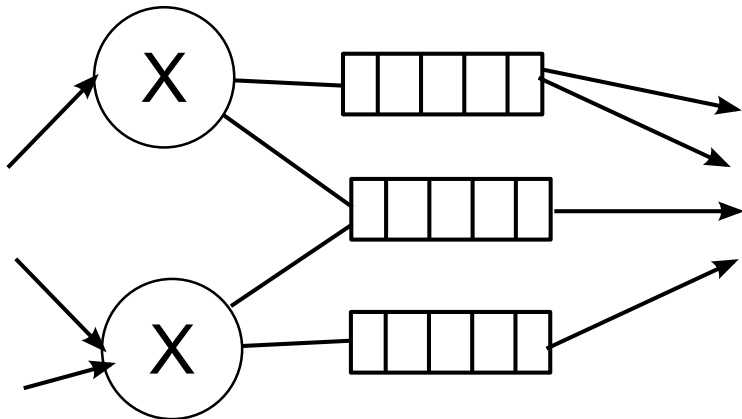




# AMQP KEY CONCEPTS



# AMQP KEY CONCEPTS





## SENDING MESSAGES TO DISK

- If a queue is *durable* and it receives a message that is *persistent* then the message will be sent to disk
- Durable queues magically survive broker shutdowns and reappear with their content they had when they died
- Hard disks are rather slow, so writing to disk as optimally as possible is a good idea

## SENDING MESSAGES TO DISK

- If a queue is *durable* and it receives a message that is *persistent* then the message *may* be sent to disk
- Durable queues magically survive broker shutdowns and reappear with their content they had when they died
- Hard disks are rather slow, so writing to disk as optimally as possible is a good idea



## SENDING MESSAGES TO DISK

- If a queue is *durable* and it receives a message that is *persistent* then the message *may* be sent to disk
- Durable queues magically survive broker shutdowns and reappear with their content they had when they died
- Hard disks are rather slow, so writing to disk as optimally as possible is a good idea
- If the queue grows *really* big, then it's not a good idea to hold messages in RAM
- ...and for maximum scalability, we don't want any per-message data structures

## OPERATION

- Use `disk_log`
- Write out the current contents of the queue
- Then write out the changes to that queue: publishes, delivers, acks, etc
- Maintain the content of the queue in RAM
- From time to time, write a new log, with a new snapshot of the queue

## OPERATION

- Use `disk_log`
- Write out the current contents of the queue
- Then write out the changes to that queue: publishes, delivers, acks, etc
- Maintain the content of the queue in RAM
- From time to time, write a new log, with a new snapshot of the queue

## PROPERTIES

- Really fast for simple use cases - one in, one out (optimal)
- Performance degrades as queues get bigger
- Messages held in RAM causes some scaling issues



## OPERATION

- Use `disk_log`
- Write out the current contents of the queue
- Then write out the changes to that queue: publishes, delivers, acks, etc
- Maintain the content of the queue in RAM
- From time to time, write a new log, with a new snapshot of the queue

## PROPERTIES

- Really fast for simple use cases - one in, one out (optimal)
- Performance degrades as queues get bigger
- Messages held in RAM causes *massive* scaling issues

## OPERATION

- Messages get appended to a file
- When the current file gets *full* we start a new file
- Message delivery does not alter any file at all
- When two neighbouring files get empty enough we garbage collect and combine them

## OPERATION

- Messages get appended to a file
- When the current file gets *full* we start a new file
- Message delivery does not alter any file at all
- When two neighbouring files get empty enough we garbage collect and combine them

## PROPERTIES

- Additional accounting needed, so not optimal in simplest case
- In more complex cases, performance does not degrade
- Accounting done in `ets` and `mnesia` so can switch to `dets` and `disc_only_copies` when RAM gets tight



```
dets_ets_insert(Obj, #state {mode = Mod, table = Tab}) ->
  Mod:insert(Tab, Obj).
```

```
...
```

```
ok = dets_ets_insert({a,b,c}, State),
```

```
...
```

```
dets_ets_insert(Obj, #state {mode = Mod, table = Tab}) ->
  Mod:insert(Tab, Obj).
```

```
...
```

```
ok = dets_ets_insert({a,b,c}, State),
```

```
...
```

```
dets_ets_insert(Obj, #state {mode = dets, table = Tab}) ->
  ok = dets:insert(Tab, Obj);
```

```
dets_ets_insert(Obj, #state {mode = ets, table = Tab}) ->
  true = ets:insert(Tab, Obj),
  ok.
```

```
...
```

```
ok = dets_ets_insert({a,b,c}, State),
```

```
...
```

# SOME INTERESTING GOTCHAS

```
...  
WriteHdl = file:open(Name, [write, raw, binary,  
                           delayed_write]),  
...
```



## SOME INTERESTING GOTCHAS

```
...  
WriteHdl = file:open(Name, [write, raw, binary,  
                           delayed_write]),  
...  
  
...  
ReadHdl = file:open(Name, [read, raw, binary,  
                           read_ahead]),  
...
```

## SOME INTERESTING GOTCHAS

```
...  
WriteHdl = file:open(Name, [write, raw, binary,  
                           delayed_write]),  
...
```

```
...  
ReadHdl = file:open(Name, [read, raw, binary,  
                           read_ahead]),  
...
```

If the message is in the same file as is being written to, make sure we `file:sync` before attempting the read!

## ONLY SYNC WHEN NECESSARY

- We only need to sync when we're closing a file (maybe?) or...
- ...when reading from the same file as we're writing to, or...
- ...on `tx_commit` (must guarantee data's gone to disk)



## ONLY SYNC WHEN NECESSARY

- We only need to sync when we're closing a file (maybe?) or...
- ...when reading from the same file as we're writing to, or...
- ...on `tx_commit` (must guarantee data's gone to disk)
- So just hold an `is_dirty` flag and sync if it's set on any of the above + reset the flag.

## ONLY SYNC WHEN NECESSARY

- We only need to sync when we're closing a file (maybe?) or...
- ...when reading from the same file as we're writing to, or...
- ...on `tx_commit` (must guarantee data's gone to disk)
- So just hold an `is_dirty` flag and sync if it's set on any of the above + reset the flag.

## `Tx_commit` COALESCING

- Lots of tiny transactions hurts - too many `file:syncs`
- Better to delay sending the reply to `tx_commit`, bunch them together and deal with them all in one sync

## ONLY SYNC WHEN NECESSARY

- We only need to sync when we're closing a file (maybe?) or...
- ...when reading from the same file as we're writing to, or...
- ...on `tx_commit` (must guarantee data's gone to disk)
- So just hold an `is_dirty` flag and sync if it's set on any of the above + reset the flag.

## `Tx_commit` COALESCING

- Lots of tiny transactions hurts - too many `file:syncs`
- Better to delay sending the reply to `tx_commit`, bunch them together and deal with them all in one sync
- Thus if `is_dirty`, and there are outstanding commits, and either a timer fires (timer), or we have no work to do (`{reply, Result, State, 0}` or `{noreply, State, 0}`) then sync.



## MAKING THINGS GO FAST

- Reducing calls to the OS is a good idea, especially when you're not IO bound
- In one case, literally halving the number of calls to `file:read` doubled performance, even though reading the same volume of data

## MAKING THINGS GO FAST

- Reducing calls to the OS is a good idea, especially when you're not IO bound
- In one case, literally halving the number of calls to `file:read` doubled performance, even though reading the same volume of data
- Please give me `mmap`

## MAKING THINGS GO FAST

- Reducing calls to the OS is a good idea, especially when you're not IO bound
- In one case, literally halving the number of calls to `file:read` doubled performance, even though reading the same volume of data
- **Please give me `mmap`**
- Then please rewrite `dets` to use `mmap`

```
...  
F = Obj #myrecord.myfield,  
...
```



```
...
F = Obj #myrecord.myfield,
...

...
Accessor = fun #myrecord.field/1,
F = Accessor(Obj),
...
```

### ON THE WAY IN

- A message can end up going to several queues
- But a message's payload is a binary blob, so if it's > 64 bytes then only one copy will exist
- We identify duplicates in the disk queue and reference count

### ON THE WAY IN

- A message can end up going to several queues
- But a message's payload is a binary blob, so if it's > 64 bytes then only one copy will exist
- We identify duplicates in the disk queue and reference count

### ON THE WAY OUT

- A message can end up going to several queues
- But a message's payload is a binary blob, so if it's > 64 bytes then only one copy will exist

### ON THE WAY IN

- A message can end up going to several queues
- But a message's payload is a binary blob, so if it's > 64 bytes then only one copy will exist
- We identify duplicates in the disk queue and reference count

### ON THE WAY OUT

- A message can end up going to several queues
- But a message's payload is a binary blob, so if it's > 64 bytes then only one copy will exist *every time it's read off disk...*



### ON THE WAY IN

- A message can end up going to several queues
- But a message's payload is a binary blob, so if it's > 64 bytes then only one copy will exist
- We identify duplicates in the disk queue and reference count

### ON THE WAY OUT

- A message can end up going to several queues
- But a message's payload is a binary blob, so if it's > 64 bytes then only one copy will exist *every time it's read off disk...*
- ...*which is once per queue*

### ON THE WAY IN

- A message can end up going to several queues
- But a message's payload is a binary blob, so if it's > 64 bytes then only one copy will exist
- We identify duplicates in the disk queue and reference count

### ON THE WAY OUT

- A message can end up going to several queues
- But a message's payload is a binary blob, so if it's > 64 bytes then only one copy will exist *every time it's read off disk...*
- ...*which is once per queue*
- Add caching layer to detect queues reading messages that have already been read and still remain in RAM

## MEMORY MANAGEMENT

- How do you decide when to convert a ram queue to a disk-only queue?
- How do you decide when to convert a disk-only queue to a ram queue?
- Given several candidates for each, how do you order them?

## WHERE CAN I GET THESE MAGIC BEANS?

- Not yet released
- Probably still buggy
- But is in our public Hg repository
- Some features still to be implemented
- Targetting the 1.7 release of RabbitMQ



END

... WAKE UP, IT'S OVER.

Thank you.

Questions?



# Demo URL

`http://www.reversehttp.net/demos/  
standalone/rabbitlog.zip`

- Unzip the single HTML file
- Open it from your local disk
- If you use Firefox, you will be asked for permission to access the network

# RabbitHub

RabbitMQ + Mochiweb = PubSubHubBub

Tony Garnock-Jones <[tonyg@lshift.net](mailto:tonyg@lshift.net)>

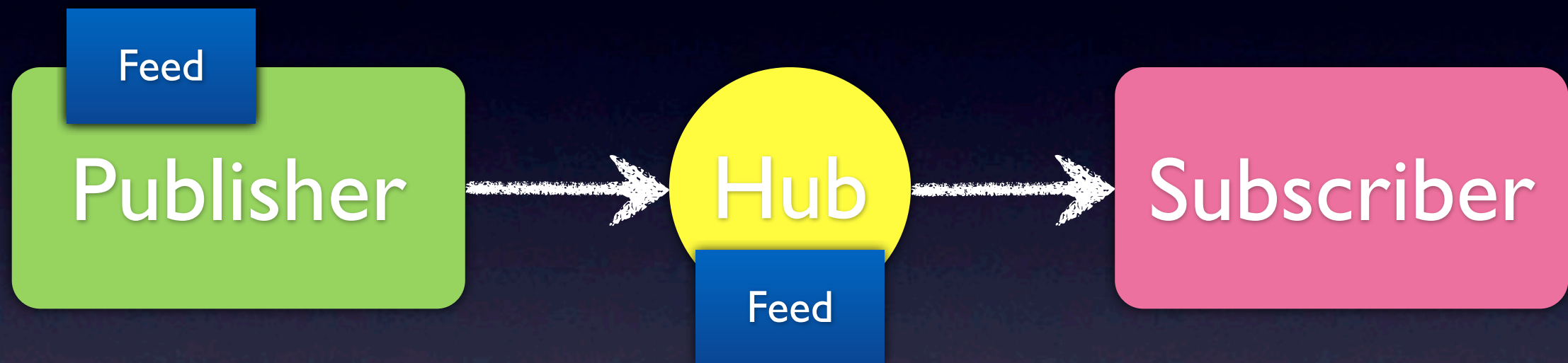


# PubSubHubBub Basics



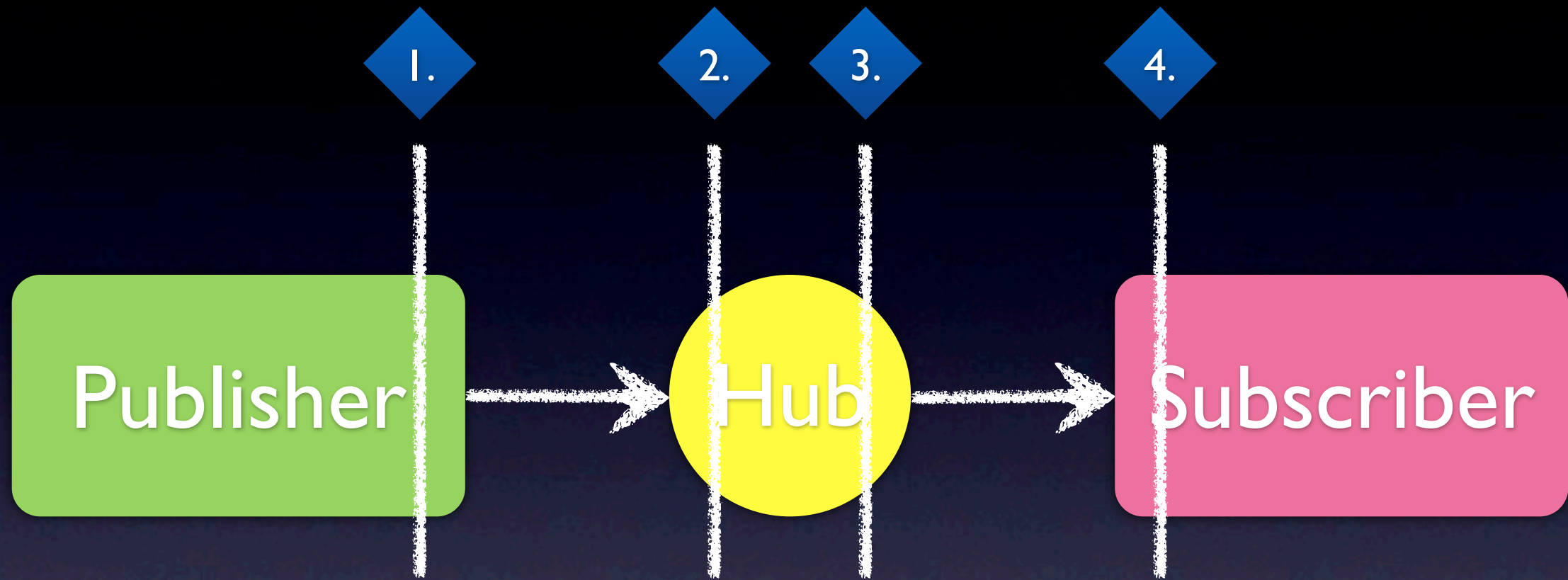
- Polls on subscribers' behalf
- Uses HTTP **push** to deliver messages
- <http://code.google.com/p/pubsubhubbub/>

# PubSubHubBub Basics



- Publisher produces an Atom feed
- Someone pings the Hub, pointing at the feed
- The Hub (re)fetches the feed, and ...
- ... digests it, republishing articles on to Subscribers.

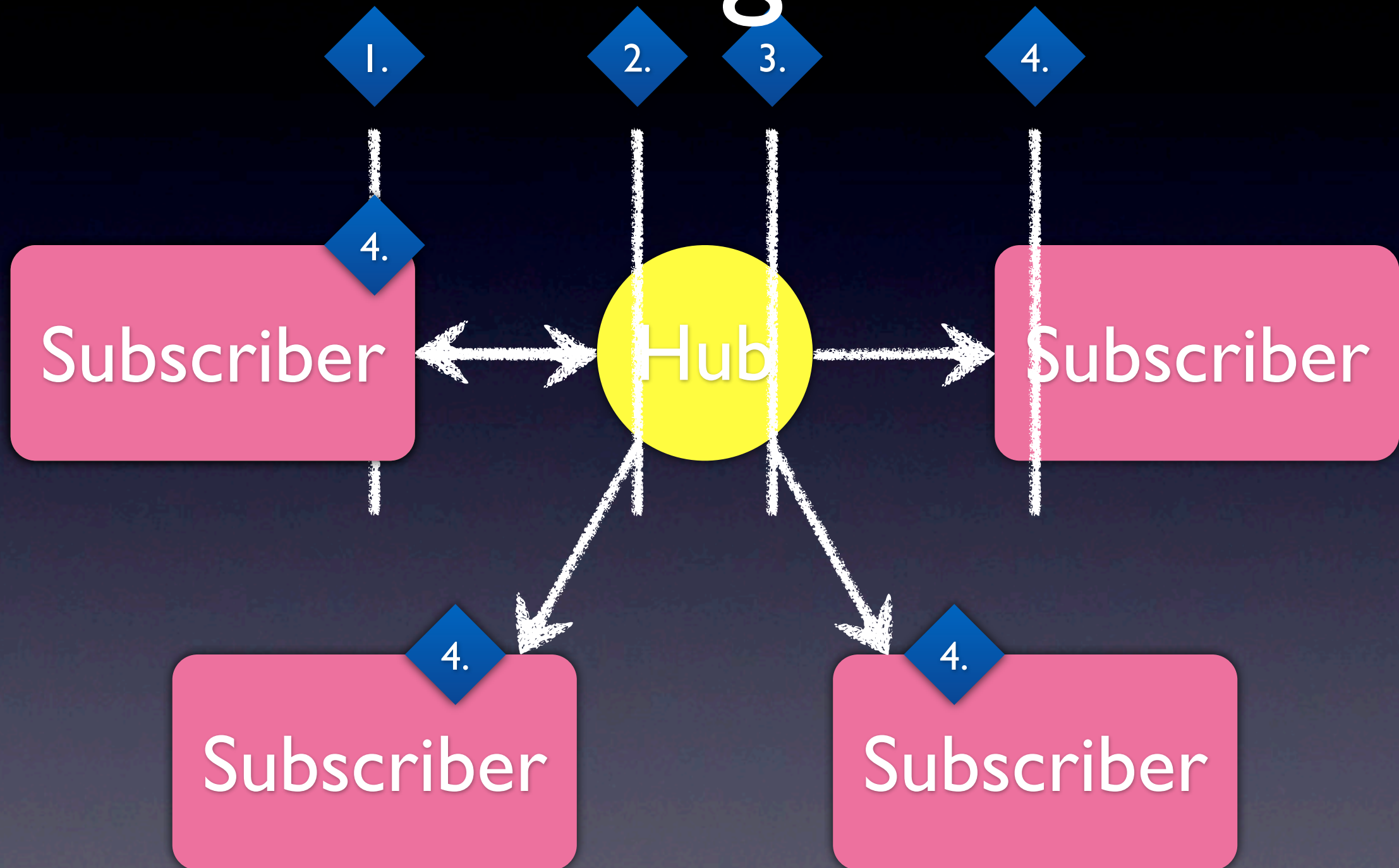
# Four Roles



1. publishing content
2. being notified of new content
3. pushing content to subscribers
4. receiving content from a hub

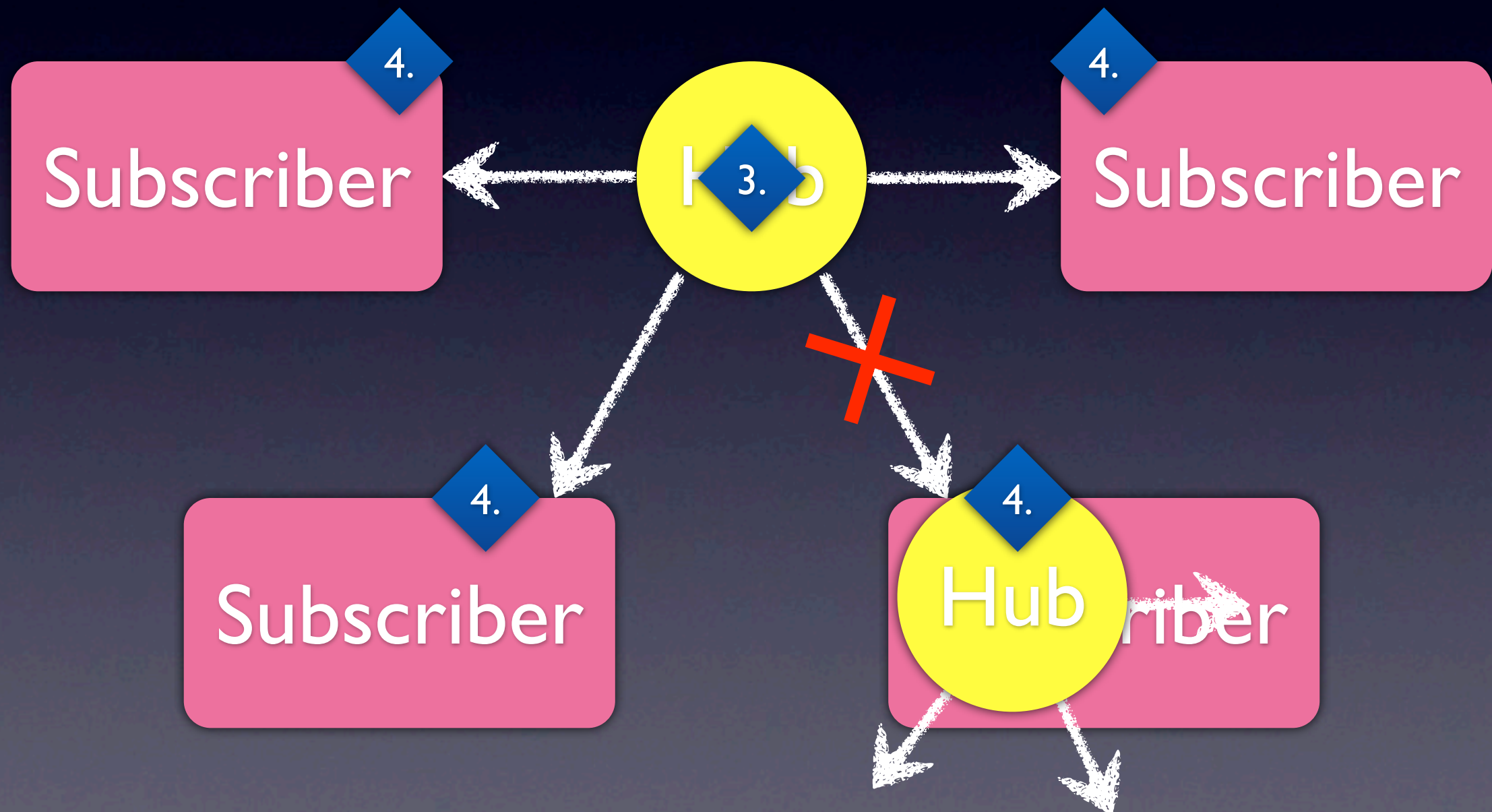


# Chaining Hubs

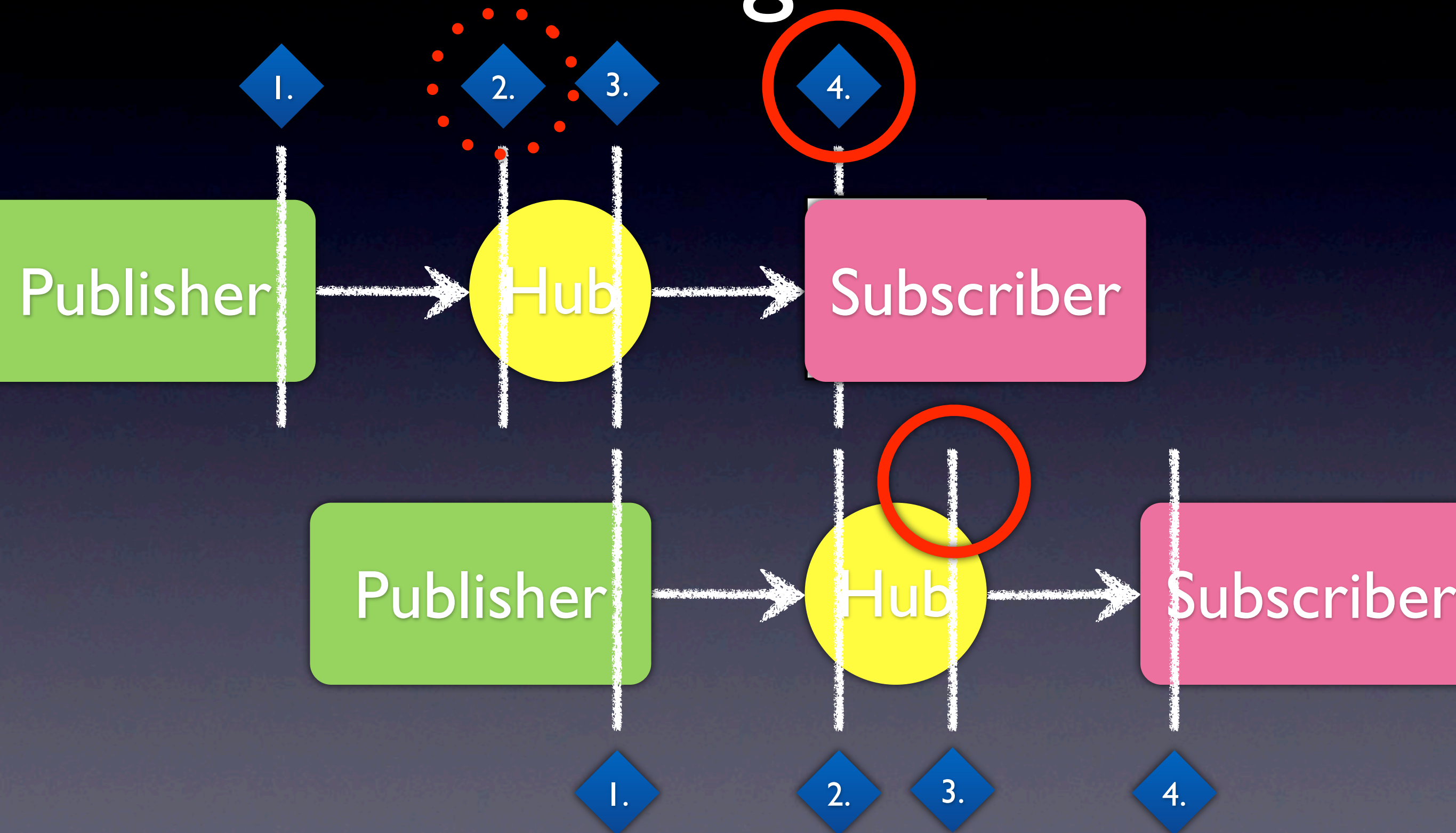




# Chaining Hubs



# Chaining Hubs



# From Roles to URLs

## 3. Subscribable source

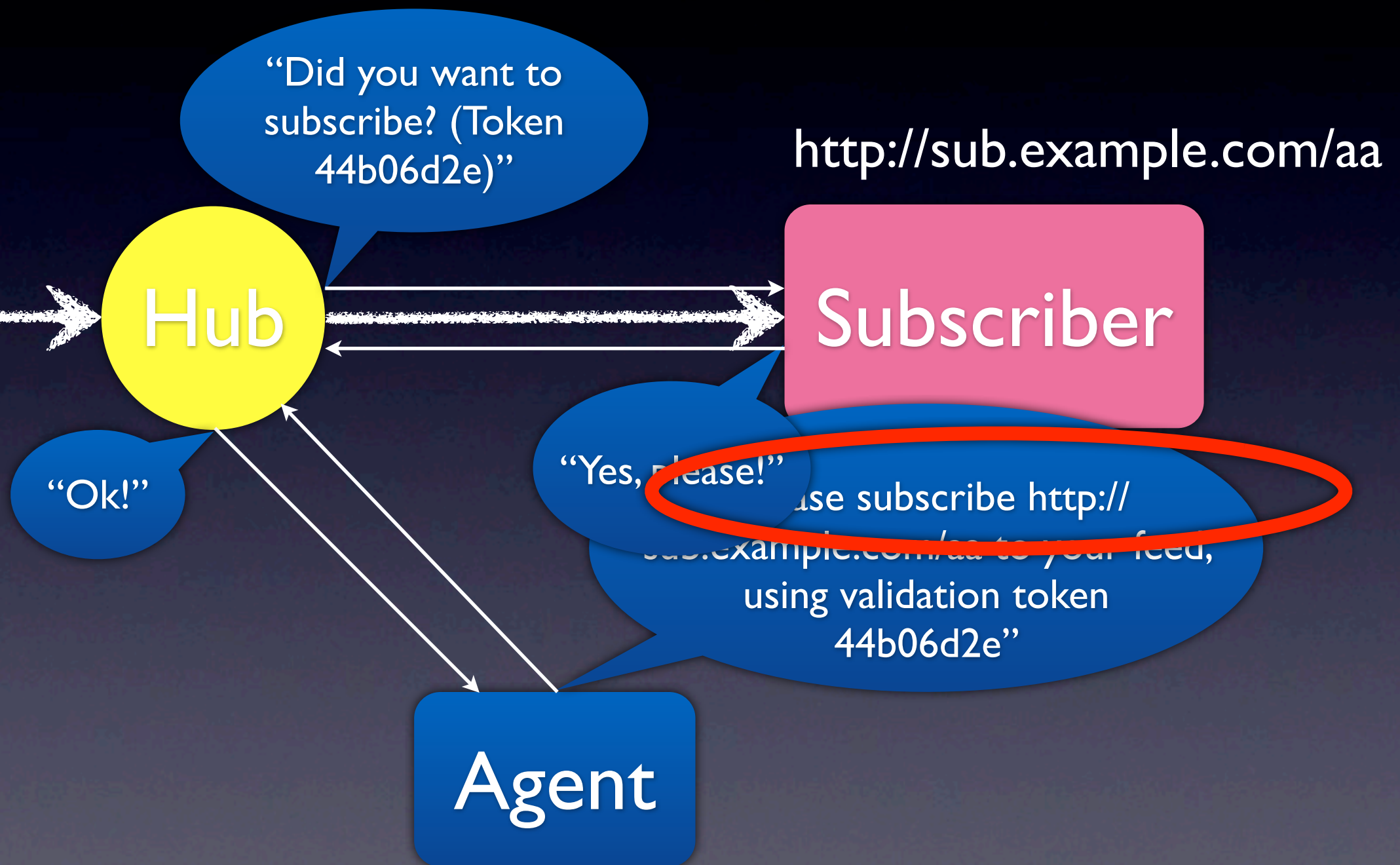
- `http://example.com/subscribe/...`
- `.../x/exchangename, .../q/queueName`

## 4. Deliverable sink

- `http://example.com/endpoint/...`
- `.../x/exchangename, .../q/queueName`



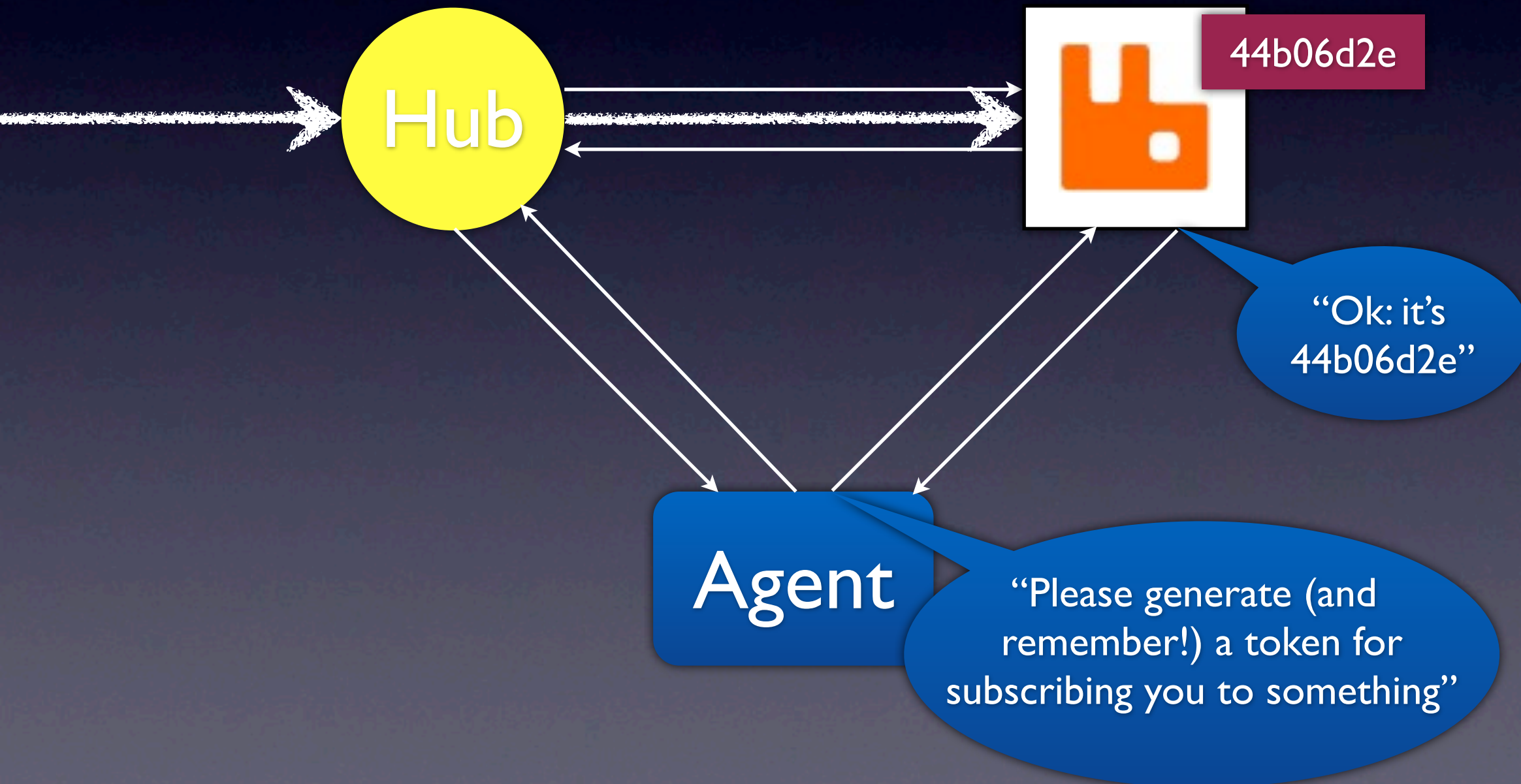
# Subscription





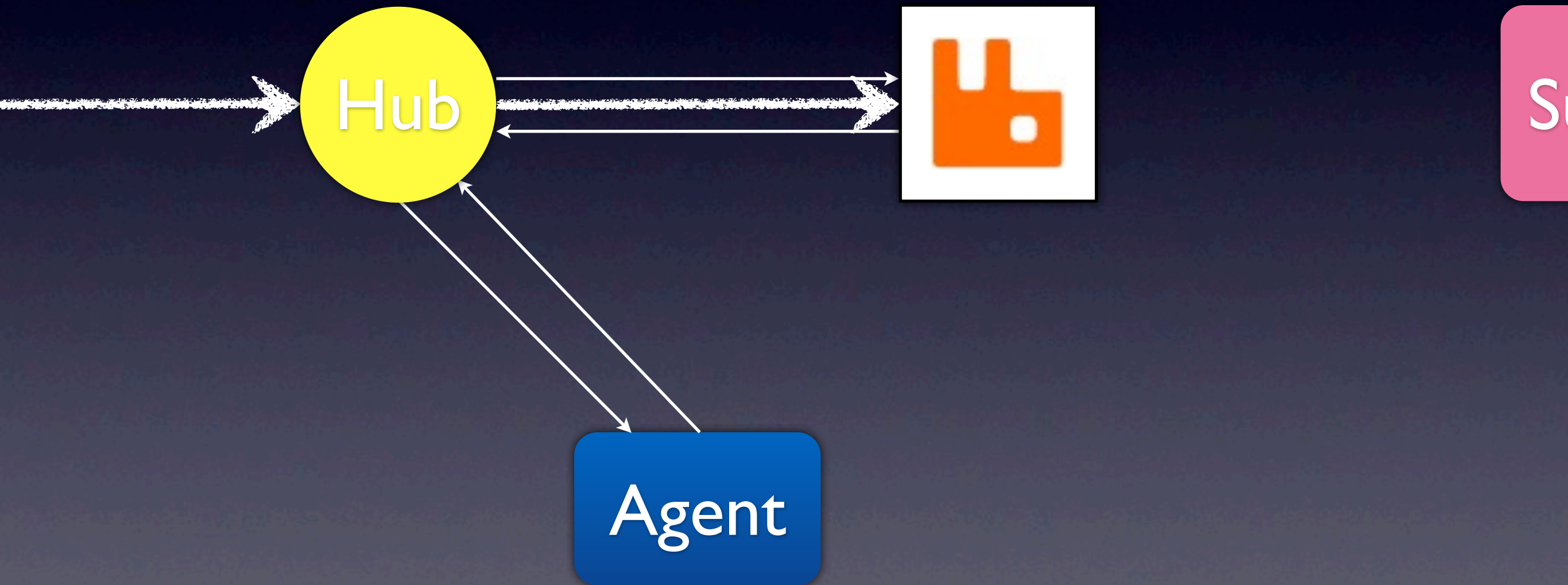
# RabbitHub Sink

<http://dev.rabbitmq.com/rabbithub/endpoint/x/amq.fanout>



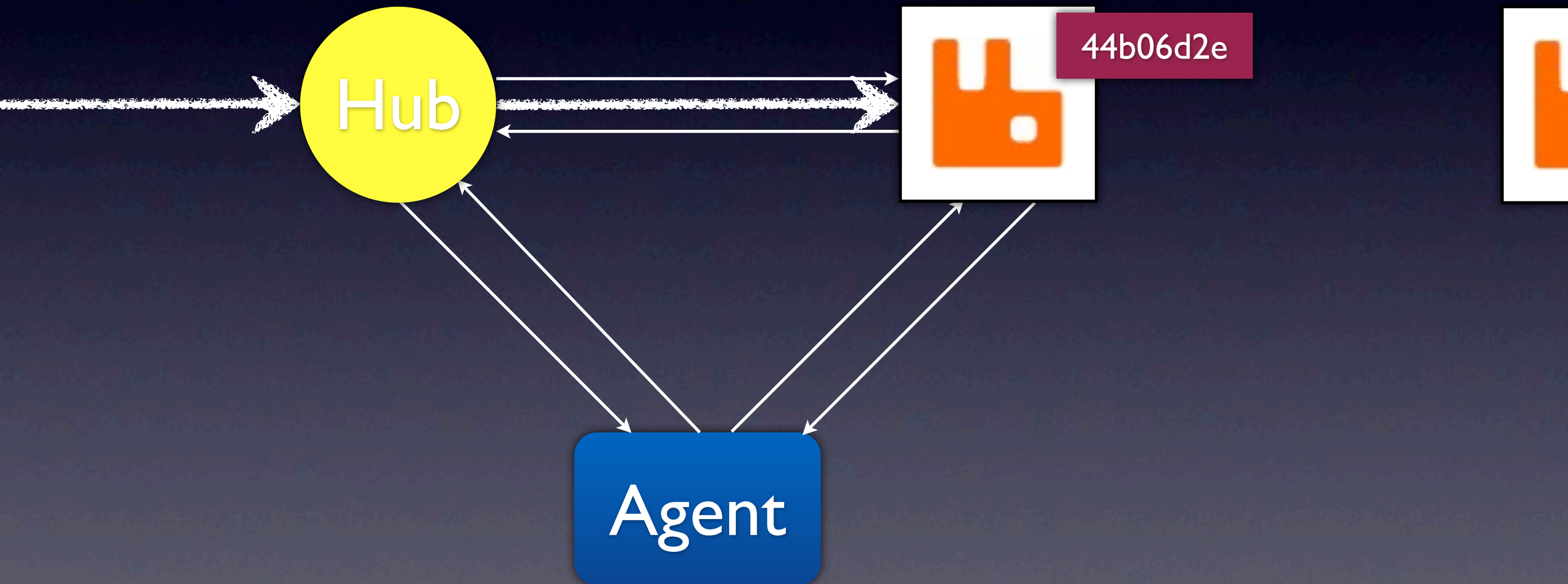
# RabbitHub Source

<http://dev.rabbitmq.com/rabbithub/endpoint/x/amq.fanout>



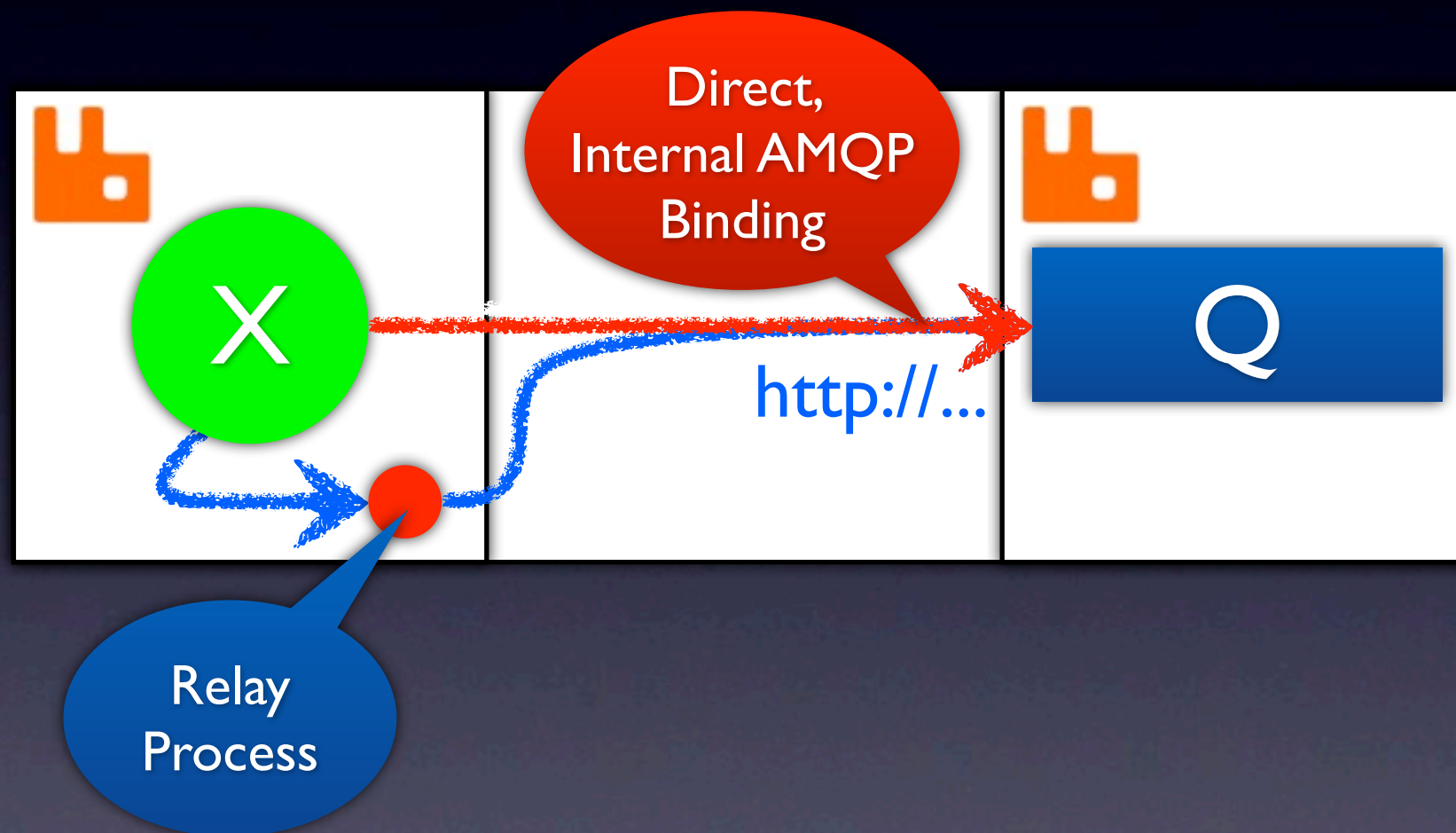
# Both Pieces Together

<http://dev.rabbitmq.com/rabbithub/endpoint/x/amq.fanout>





# Shortcut Binding

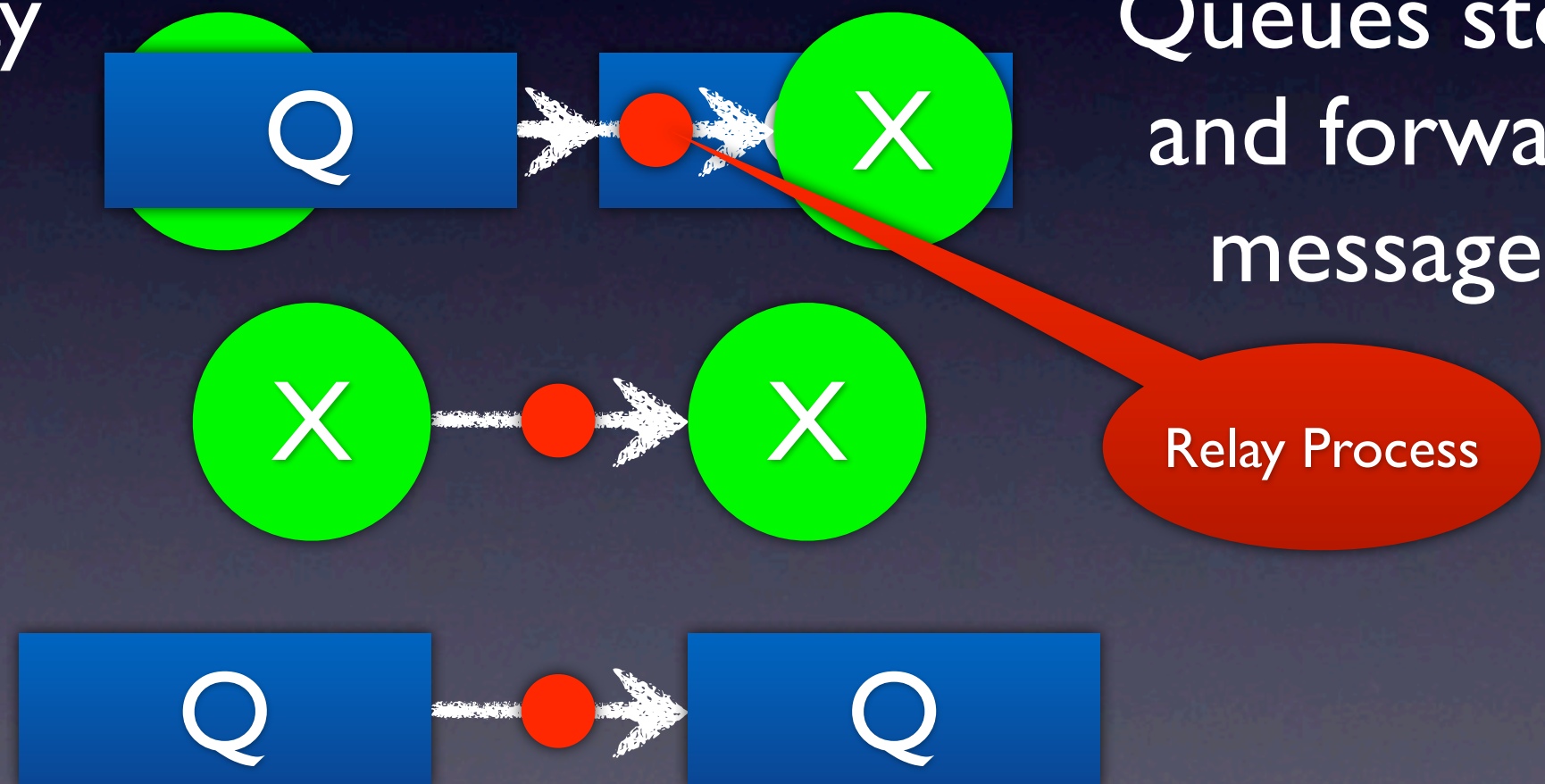




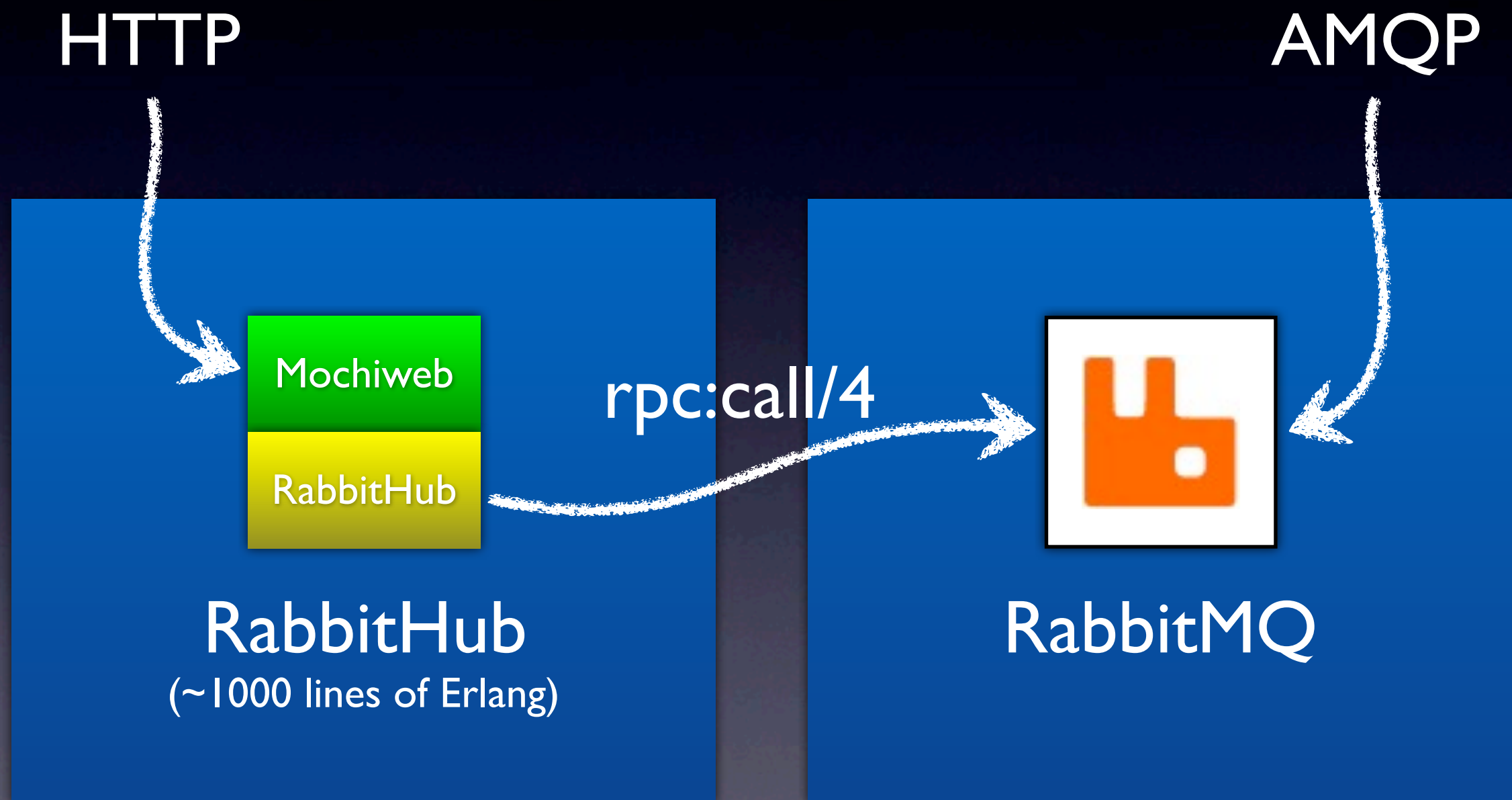
# Full Connectivity

Exchanges relay  
and filter  
messages

Queues store  
and forward  
messages



# Implementation



# Browser-based Pubsub



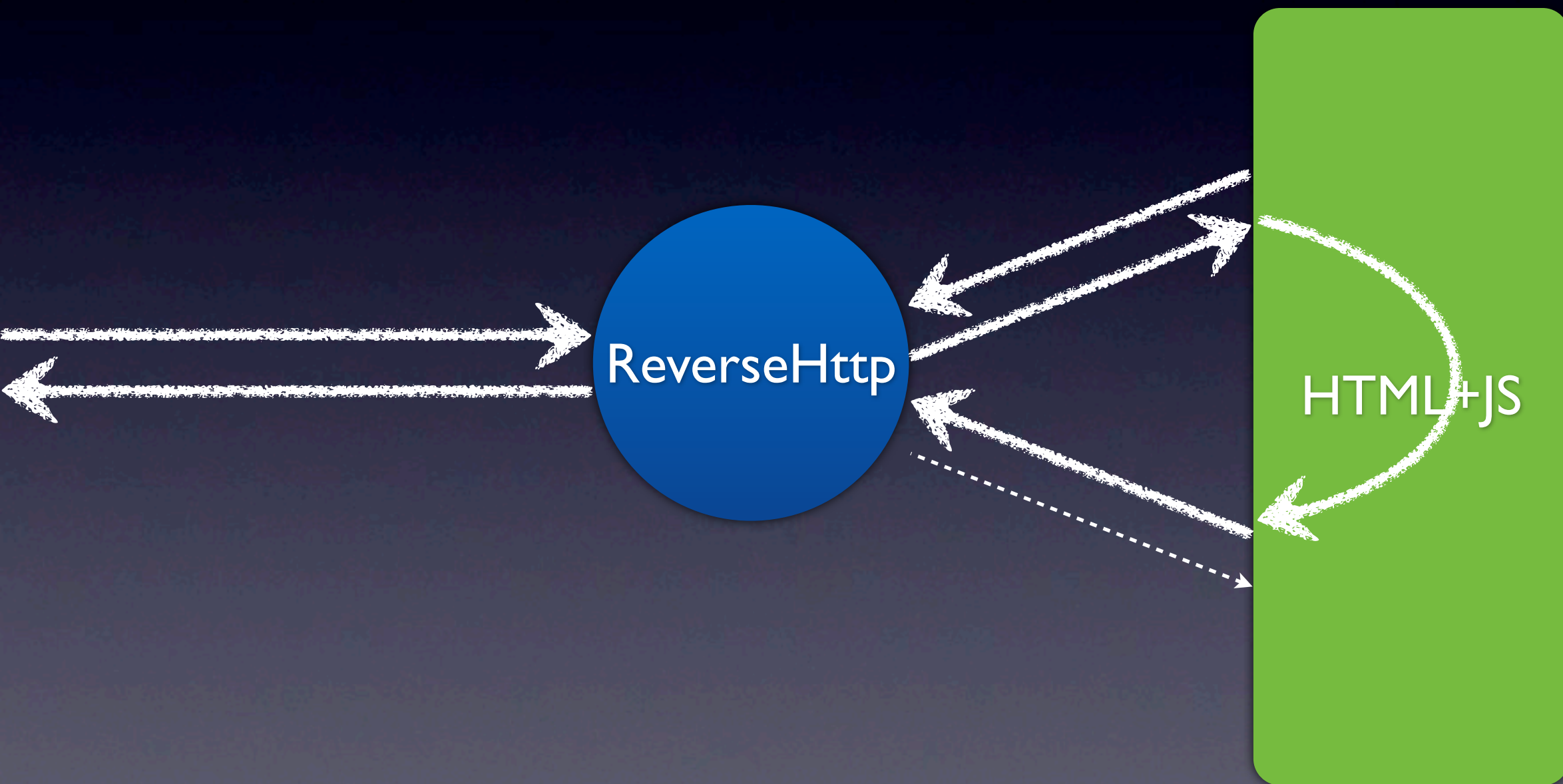


# ReverseHttp

- Tunnels HTTP over HTTP, in both directions
- Lets HTML+JS be full participants in the web - **httpd in a webpage** (Opera Unite is like this)
- <http://www.reversehttp.net/> has draft specification and downloadable implementation (Mochiweb again! ~700 LOC)
- Language-neutral: JS, Java, Python, ...



# ReverseHttp



# Demo URL

`http://www.reversehttp.net/demos/  
standalone/rabbitlog.zip`

- Unzip the single HTML file
- Open it from your local disk
- If you use Firefox, you will be asked for permission to access the network

# Demo

- `rabbitlog.html` – HTML+JS Pubsub App
- `www.reversehttp.net` – ReverseHttp
- `dev.rabbitmq.com/rabbithub` – RabbitHub
- `dev.rabbitmq.com` – AMQP



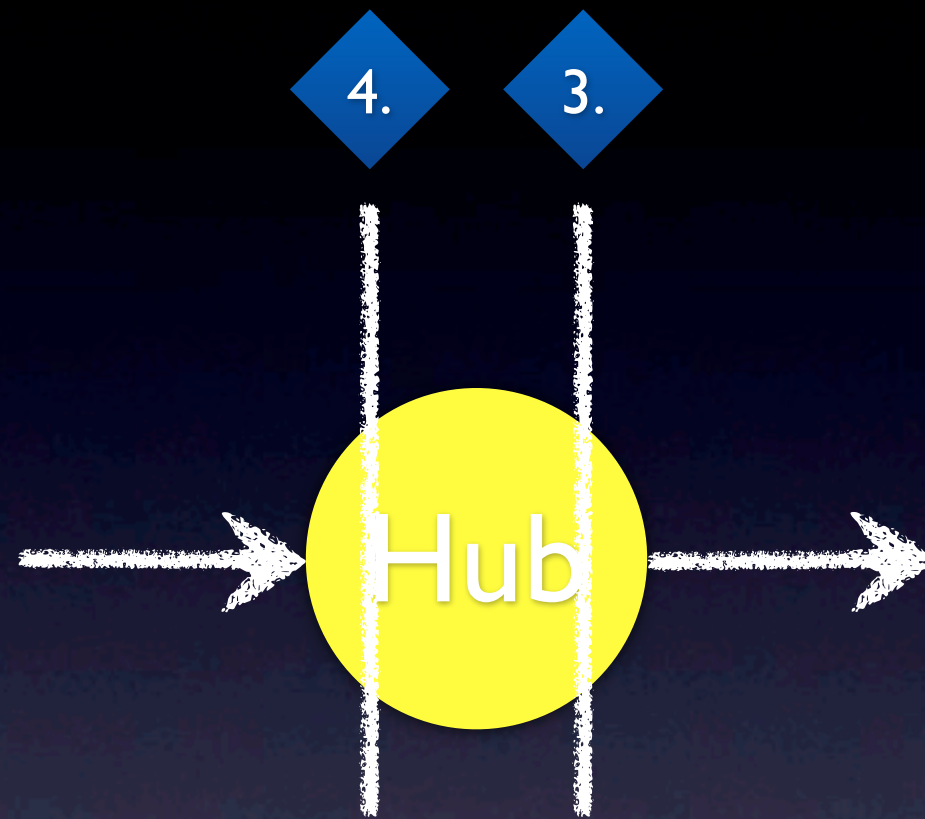
<http://www.rabbitmq.com/>  
<http://github.com/tonyg/rabbithub>  
<http://github.com/tonyg/reversehttp>

Questions?





# From Roles to URLs



3. pushing content to subscribers
  - <http://.../subscribe/x/exchangename>
  - <http://.../subscribe/q/queueuname>
4. receiving content
  - <http://.../endpoint/x/exchangename>
  - <http://.../endpoint/q/queueuname>