Polling Sucks

So what should we do instead?

Should we use XMPP? What about AMQP? What about plain old HTTP push? Should it be peer-to-peer? Intermediated? Disintermediated?

Messaging

The answer is banal: Use messaging. The internet is all about the two aspects of messaging: resources, and state transfer. Modern systems like XMPP and AMQP as well as older systems like SMTP, NNTP and UUCP focus on the state transfer; FTP and HTTP focus more on the resources themselves. Recent approaches like ReST are all about finding a perfect balance point between resources and communication. In this talk I'm going to highlight the split between messaging fabric and messaging applications, show a couple of ways that RabbitMQ can be used as a component of interesting distributed applications, and then see if I can convince you that there are worse places to start than HTTP for building an internet-scale messaging fabric.

Fabric

- Addressing
- Authentication & Authorisation
- Point-to-point
- Best-effort, positive acknowledgement
- Push vs. Pull
- Symmetric
- Syntax

So, to address this split between the fabric and the application. The fabric is all about getting information on a best-effort basis from point to point in the network. This might sound like messaging, but it's not: the difference is that the transfer of information at this level is stateless, with idempotent, state-transferring operations the rule, where messaging is stateful. Consider IP compared to TCP: IP simply relays packets statelessly; it's the TCP endpoints that update their own state in response to received IP packets. IP then is a fabric for TCP's application.

Application

- Trust & Responsibility
- Reliable delivery (“exactly once”)
- Relaying, Filtering, Buffering, Queueing
- Multi-hop; middleboxes
- Management + Monitoring
- Actors, Methods and Events
- Semantics

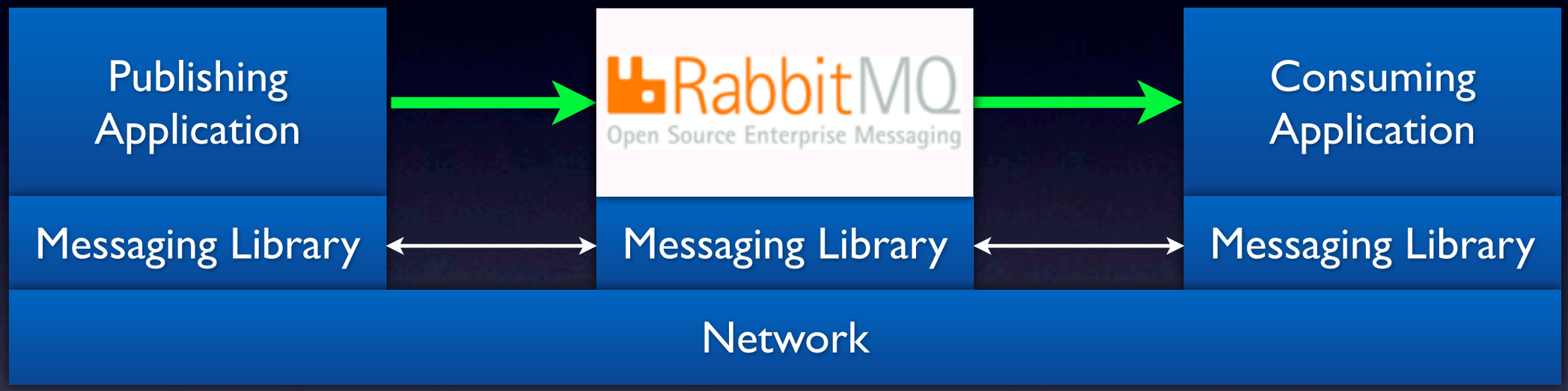
Applications, from the point of view of this split of messaging into its components, are the parts of the system that hold mutable state. Applications in a distributed system have contracts with each other: for instance, an SMTP server takes responsibility for a message when it sends back a 250 reply code. As another example, TCP endpoints take responsibility for received segments by acknowledging them. Applications are also where domain-specific logic starts to become visible: for example, within the mailing-list manager Mailman, delivery of a received message to the whole subscriber list is a domain concern that the underlying SMTP network is unaware of.

Traditional View

The traditional view of MOM, Middleware Oriented Messaging, has the queues and relays as services that are subordinate to the producers and consumers of messages. This is fine up to a point; it starts to look a bit strange when you realise that you want to be able to trust the middleman as a peer, allowing it to make its own decisions about the messages you send to it as a first-class part of your distributed application.

Revised View

Trust.



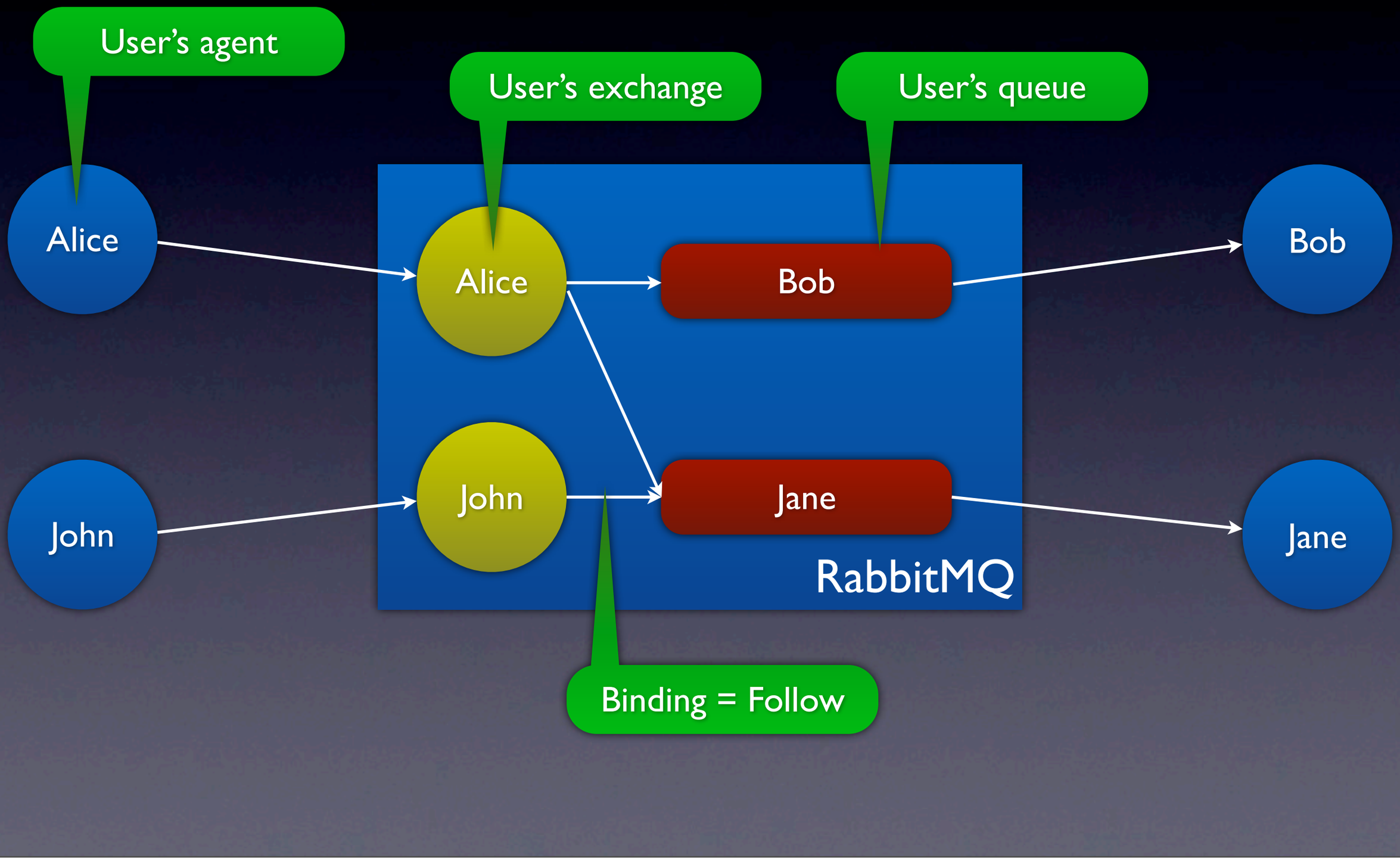
So the way we're starting to look at things has the broker -- the relays, exchanges, buffers and queues, as well as their management and monitoring -- as a first-class peer in each distributed application. RabbitMQ's purpose is to be a first-rate implementation of a broker, no matter what transport you use to communicate with it.

Exploring RabbitMQ

Toy Twitter

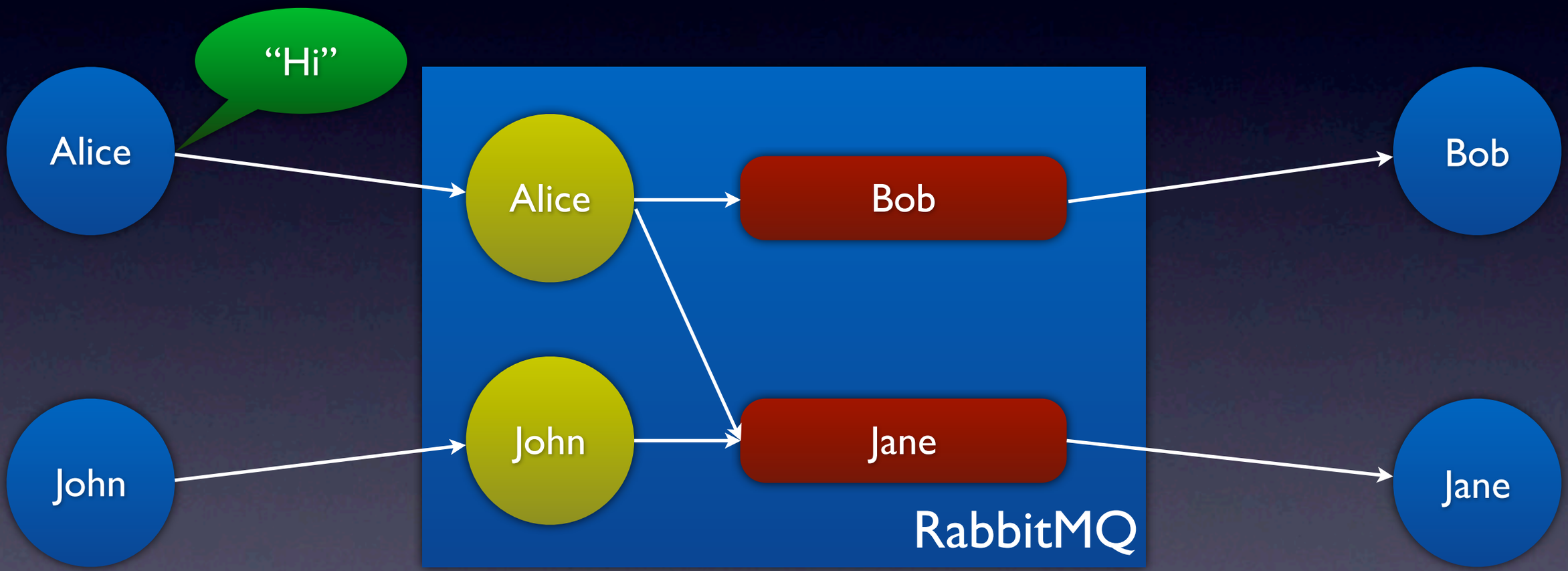
- Running example: A twitter-a-like using AMQP
- AMQP's asymmetric bindings great for asymmetric follow!
- One exchange for things you send
- One queue for things to read
- One binding per follow

Toy Twitter

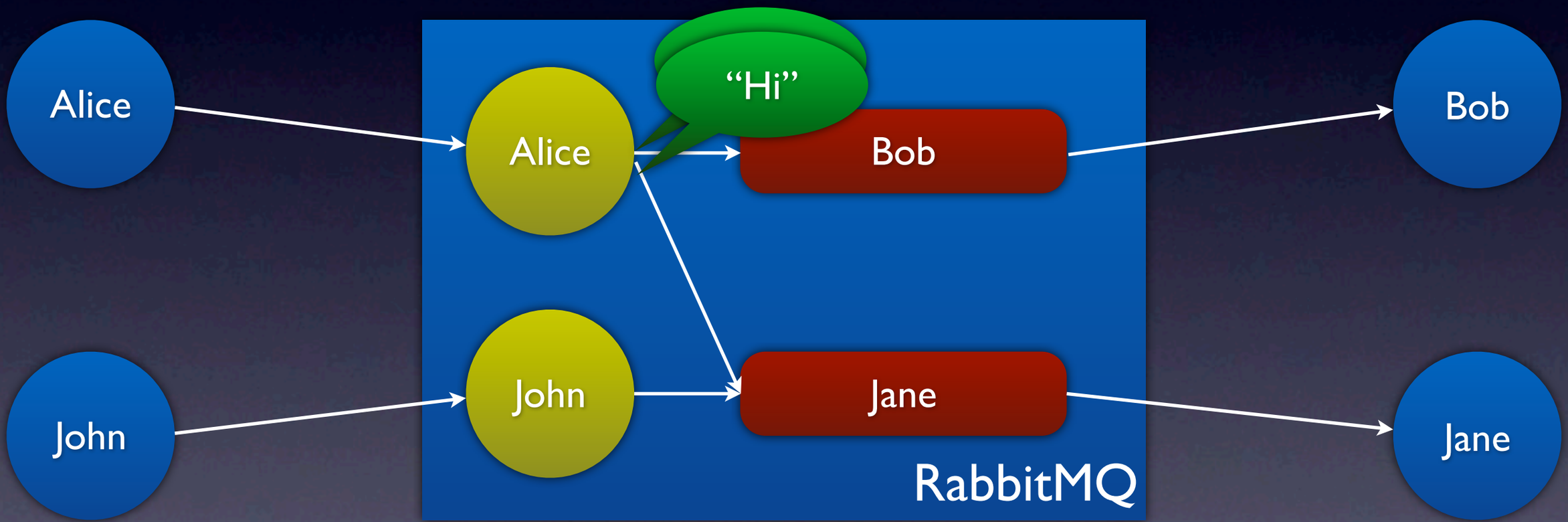


Introduce AMQP concepts implemented by RabbitMQ. Map these to the toy twitter example.

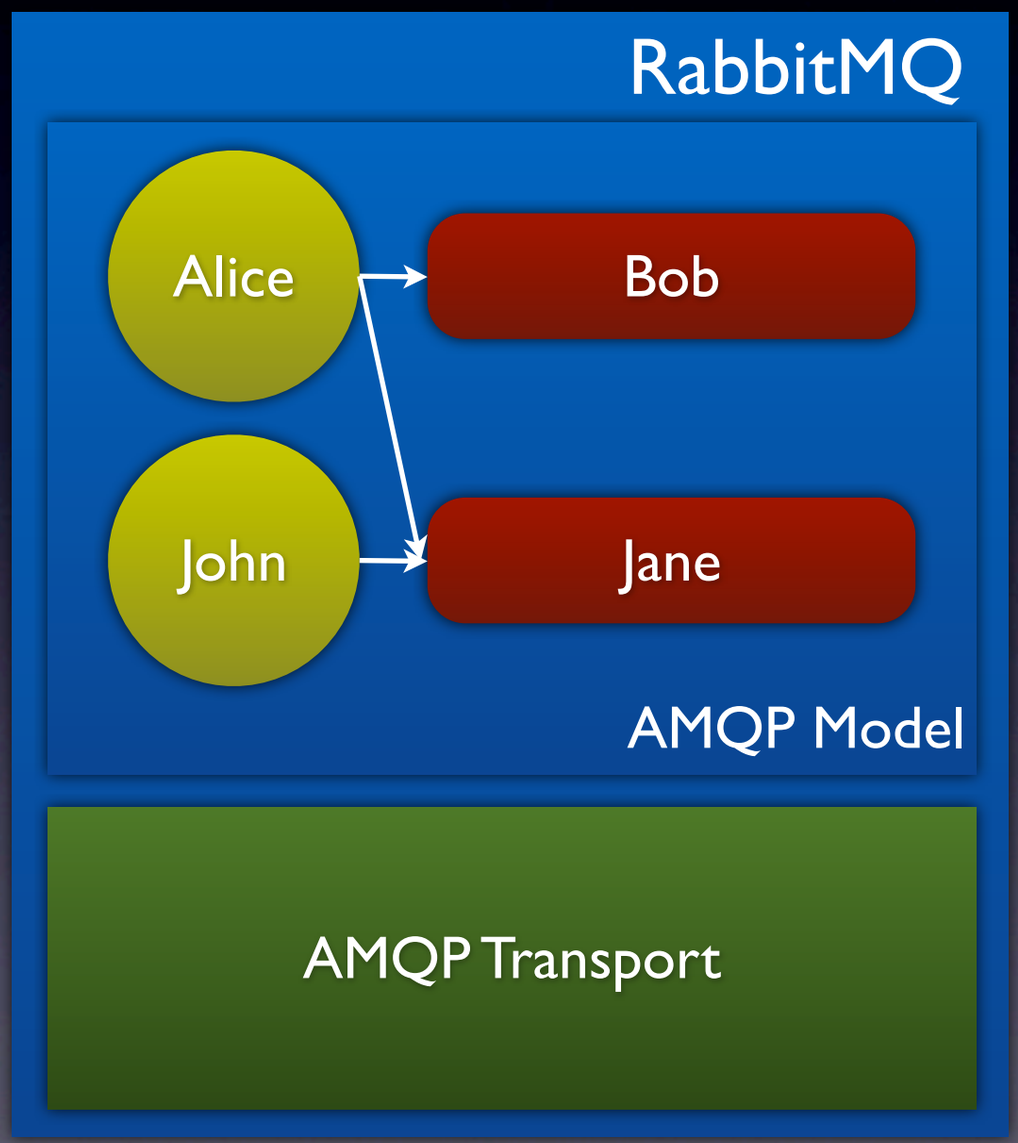
Toy Twitter



Toy Twitter

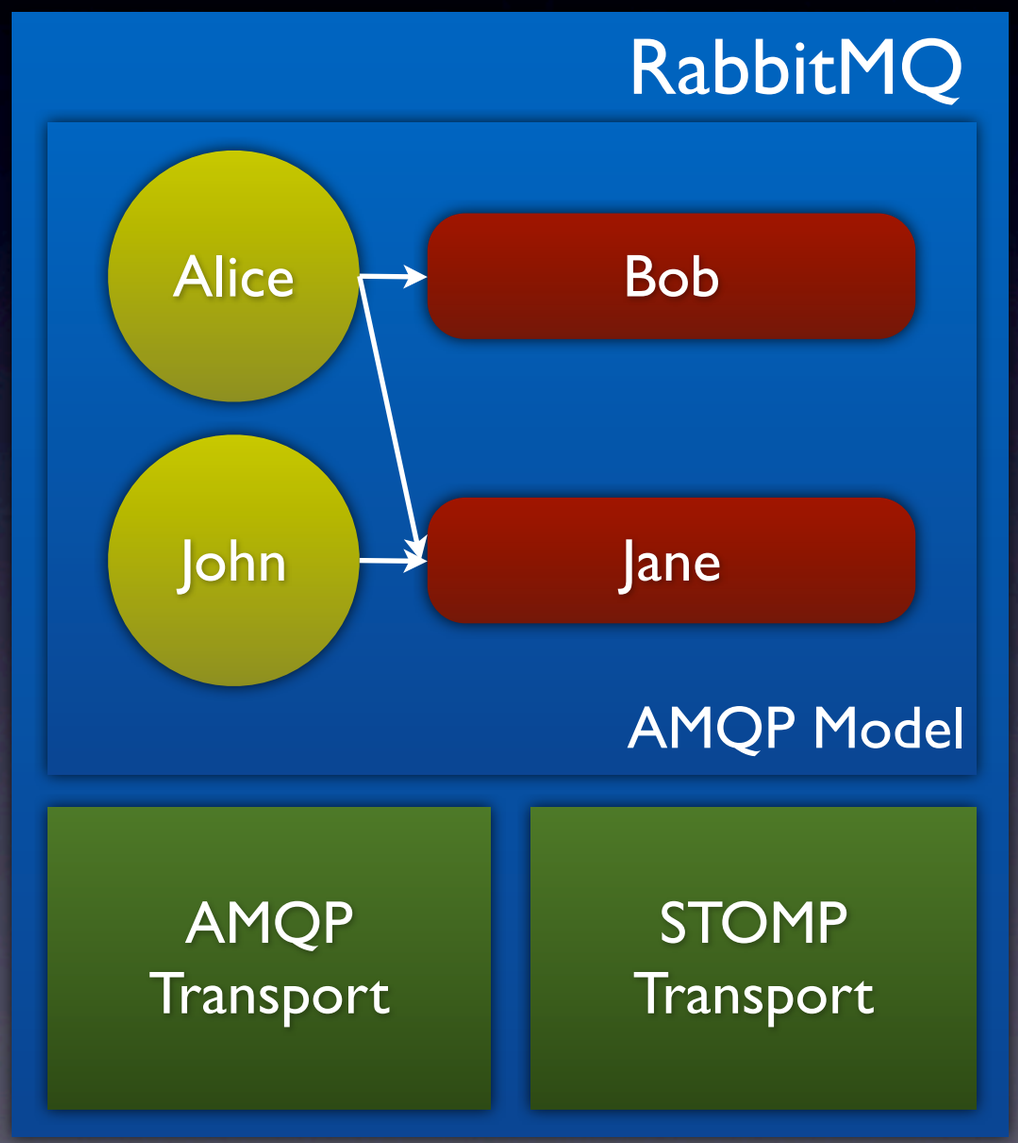


In Detail



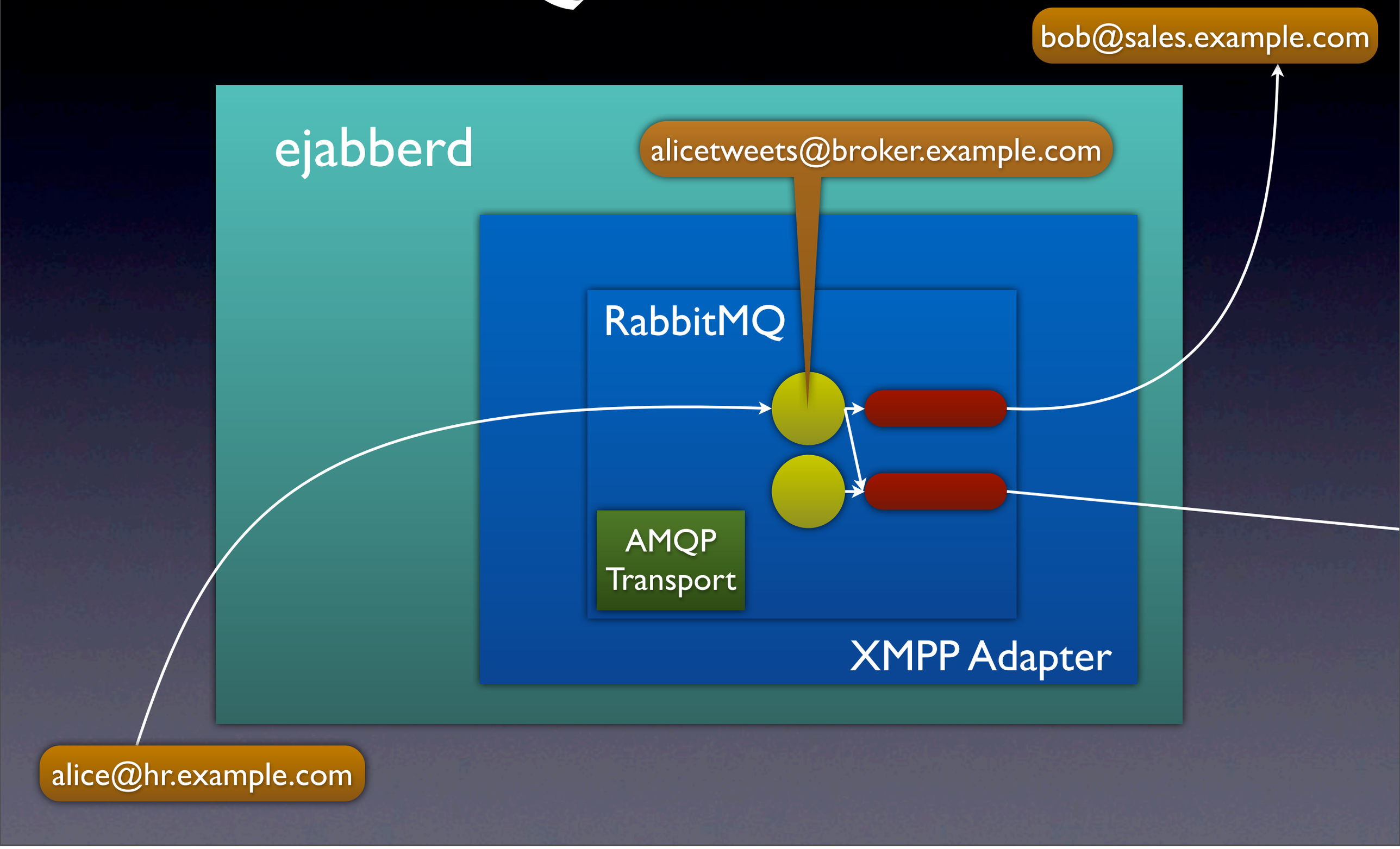
By default, RabbitMQ is configured with just the AMQP transport available.

AMQP + STOMP



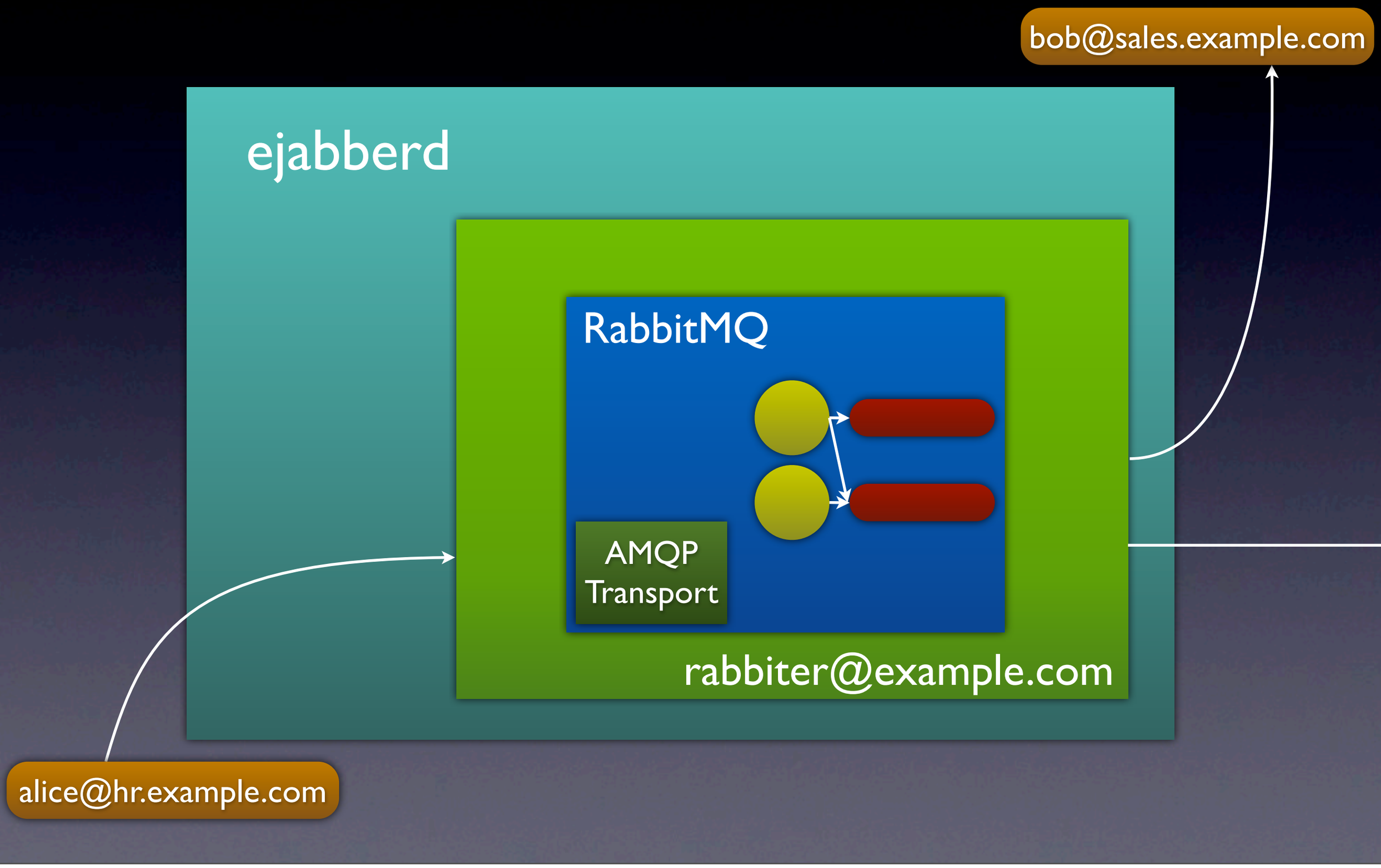
Adding in a STOMP adapter is a straightforward process. Once this is done, clients can interact with the AMQP model -- the exchanges and queues -- using either protocol. STOMP doesn't let you configure the broker, though; we're considering defining an extension, but for now use AMQP.

AMQP + XMPP



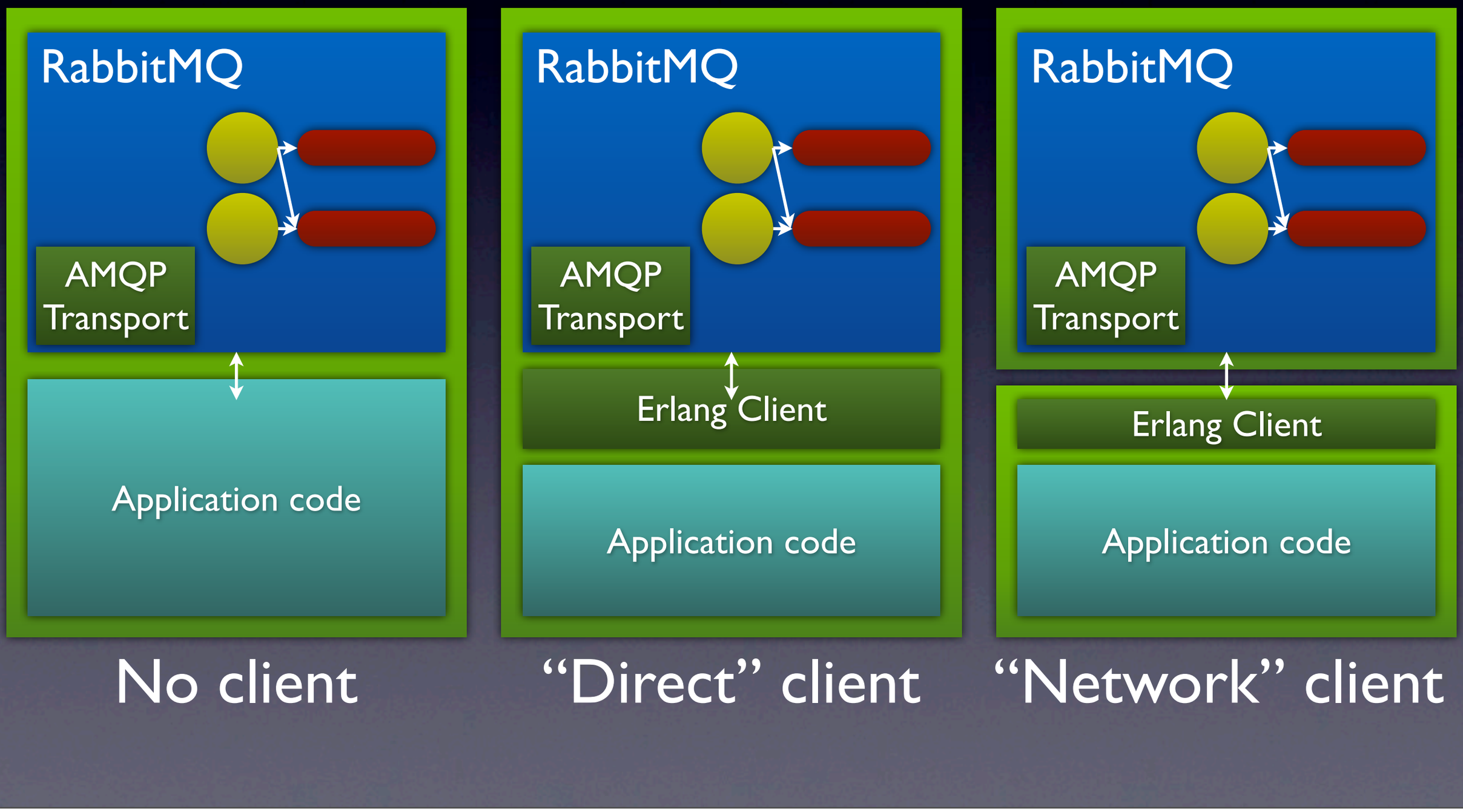
Rabbit has an experimental XMPP gateway, where RabbitMQ is embedded in ejabberd. Each exchange within the broker is given an XMPP JID. When XMPP users in the wider XMPP network add a RabbitMQ exchange as a contact, a private queue is created for the remote user (unless one already exists!) and a binding between the exchange and the private queue is set up. The “resource” part of the JID is interpreted as the AMQP routing key and binding pattern. When remote users indicate that they are online via presence, a consumer is started, which sends them the messages from their private queue. When they’re offline, the queue buffers messages for them. Now, this is quite a raw interface to exchange and queue functionality: it doesn’t feel very twitter-like!

Rabbiter



Which brings us on to a thing called Rabbiter, which we built about a year ago as an experiment. It's our toy twitter-a-like, embedded in ejabberd just like the generic XMPP RabbitMQ gateway, but with a single JID as the user interface. Backing the JID is a bot that automates the resource management for you, and relays messages both to and from the RabbitMQ broker. The internal exchanges and queues are not exposed as JIDs.

Embedded RabbitMQ



No client "Direct" client "Network" client

Explain the three styles of embedding, and talk about the Erlang client and what it can do for you. OTP's applications structure an Erlang VM instance.

Internet Messaging

Pushing polling to the edges

So having briefly explored some of the ways RabbitMQ can take part in a distributed application, I'd like to talk a bit about designing an Internet-scale messaging system, with a special focus on HTTP. Questions about the previous segment?

Architecture

- Fabric
  - Point-to-point transport protocol
  - Distributed objects/actors
- Applications: Messaging system
  - send()
  - subscribe(), unsubscribe()

The architecture of the system should reflect the split between the fabric and the application. The way I see it is as a three-layered setup: stateless point-to-point messaging; a distributed object or actor system; and finally, messaging primitives like relays and queues as applications on top of the other two layers.

Point-to-Point

- Fidelity
  - checksums
  - binary content
  - message delimiters
- Best-effort, with acknowledgements
- Asynchronous

Best-effort: at most once, at least once, but not exactly once.
Acknowledgements: acceptance, rejection, error, ...
Asynchronous because we can build any other interaction pattern on top of that.

Distributed Objects

- Addressing & Symmetry
- Authentication & Authorization
- Operations & Events

Operations *and* events. Events are a missing piece in descriptions of object-oriented systems, but were there almost from the start as a library feature.

Messaging System

- Responsibility transfer (acknowledgement) and Retransmission
- Subscription and Filtering
- Sources
- Sinks
- Relays and Buffers (both source and sink)

HTTP as Fabric?

- Checksums: from TCP
- Binary content: entity bodies are binary
- Message delimiters: content-length or chunked
- Best-effort: from TCP; acks: HTTP responses
- Asynchronous: *hmmm*
- Addressing: URLs
- Symmetry: *rHTTP*
- Authentication & Authorization: yes; also capabilities!
- Operations & Events: requesting and responding role for HTTP POST, respectively

Posting links instead of posting content: matches common patterns like link sharing via Twitter, Delicious etc. HTTP effortlessly switches between push and pull.

Other Possibilities

	Binary	Acks	Async	Addr	Auth	Ops, Events
XMPP	No	No	Yes	Yes	Maybe	Yes
AMQP	Yes	Yes	Yes	No	Local	Yes
SMTP	No	Yes	Hmm	Yes	No	No

Compensating: XMPP has presence...

Capabilities - CReST

- No global authority
  - HATEOAS
- Secure names
  - unguessable URLs
- Revokable, delegatable
  - Object cap, meta cap

Reverse HTTP

- "Remote CGI"
- Talk to a Point-Of-Attachment (POA)
- Claim a piece of URL-space
- Receive requests from POA
- Send replies to POA

On the wire

- HTTP "Upgrade" to rHTTP ("PTTH")
  - http://www.ietf.org/internet-drafts/draft-lentczner-rhttp-00.txt
- Long-poll for requests; POST replies
  - http://www.reversehttp.net/
  - http://wiki.secondlife.com/wiki/Reverse_HTTP#COMET_Fallback

Reverse HTTP Demo

The End.

http://www.rabbitmq.com/how