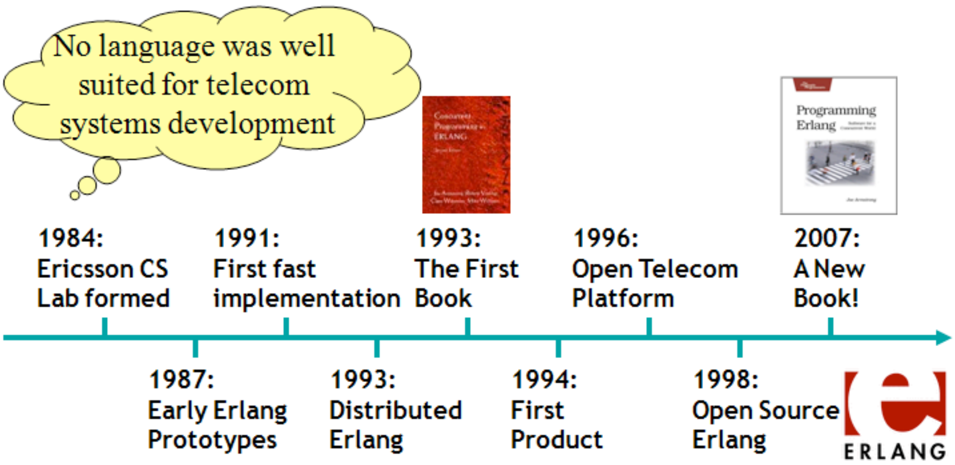
Part II

A Brief Introduction to Erlang/OTP



No language was well suited for telecom systems development

1984: Ericsson CS Lab formed

1987: Early Erlang Prototypes

1991: First fast implementation

1993: Distributed Erlang

1993: The First Book

1994: First Product

1996: Open Telecom Platform

1998: Open Source Erlang

2007: A New Book!

20 years of history

- Erlang is not some PhD student's recent pet project - it was born out of real commercial concerns 20 years ago, and has seen continuous development and commercial use over that time.
- Because for the first few years initial deployments were few and all within Ericsson, the designers of the language had the opportunity and incentive to evolve the language and correct any mistakes.
- Even now the language and platform are still evolving, but in a very controlled way - Ericsson do eat their own dogfood.

recent surge in interest

- Massive growth in community over the last couple of years
- probably because of multi-core, grid / cloud, internet

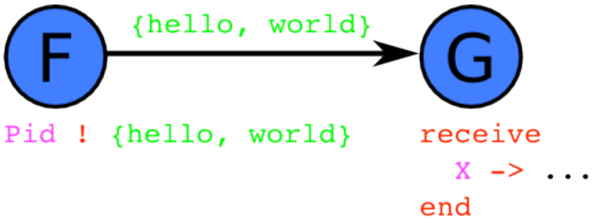
Erlang Processes

```
Pid = spawn(F)
```

- fast creation, light-weight
- shared nothing
- own gc; soft real-time
- functional - abstraction, concise code, no mutable vars

Erlang systems are built out of communicating processes. Here's how processes are created ...

Communication and Concurrency

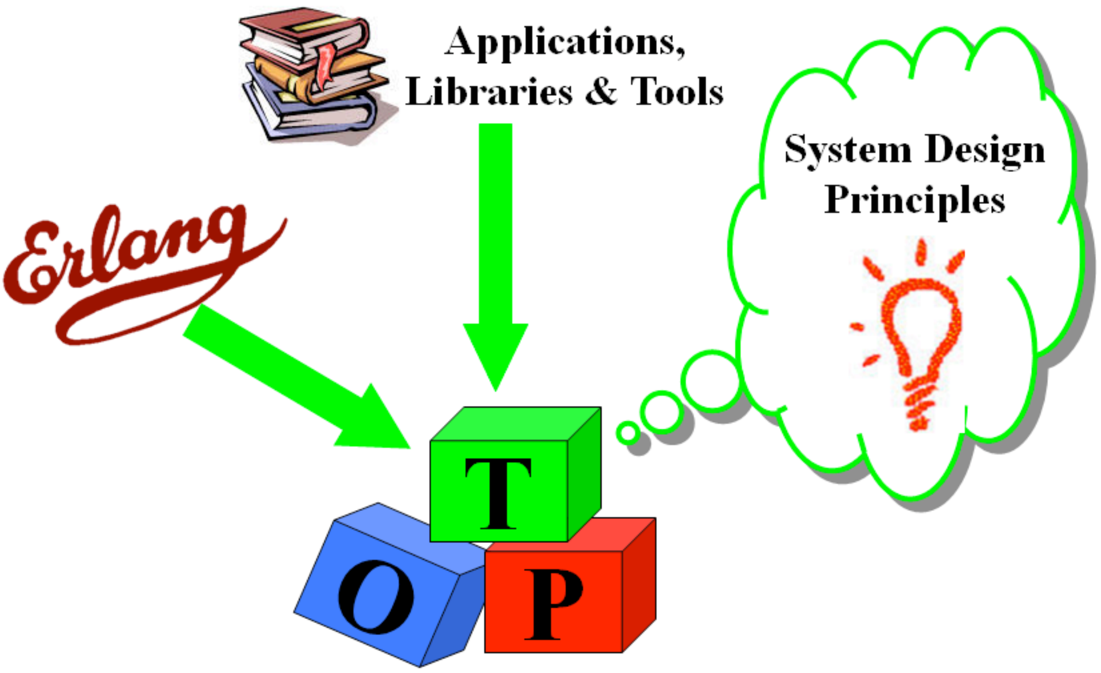


```
Pid ! {hello, world}
```

```
receive
  X -> ...
end
```

- async send, blocking receive with optional timeout
- per-process message queue
- libs for more complex communication patterns, e.g. call, multi-cast
- same code when distributed

The Open Telecom Platform



Questions?

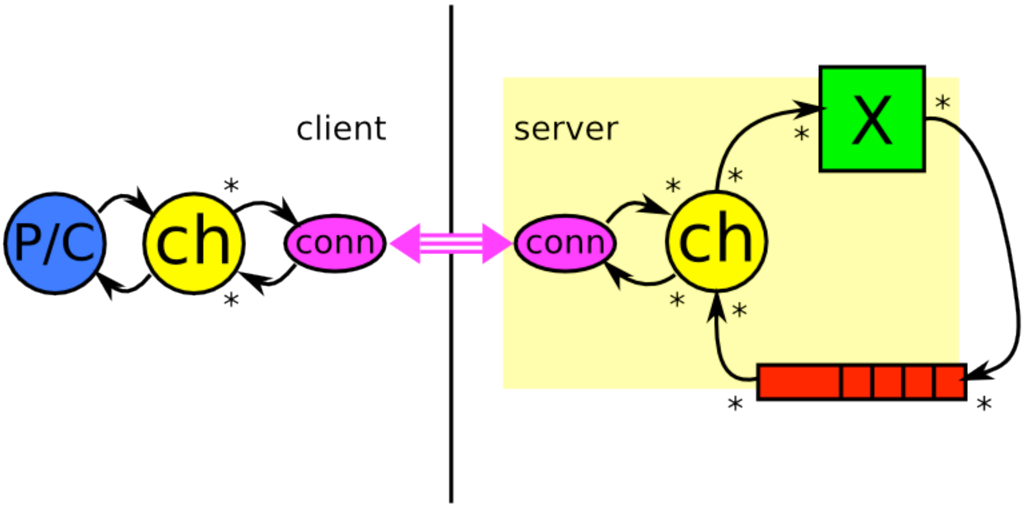
Why Erlang/OTP?

- good architectural fit for AMQP
- concise
- good performance, and it scales
- excellent management and troubleshooting support
- we can hide it

Why Erlang/OTP?

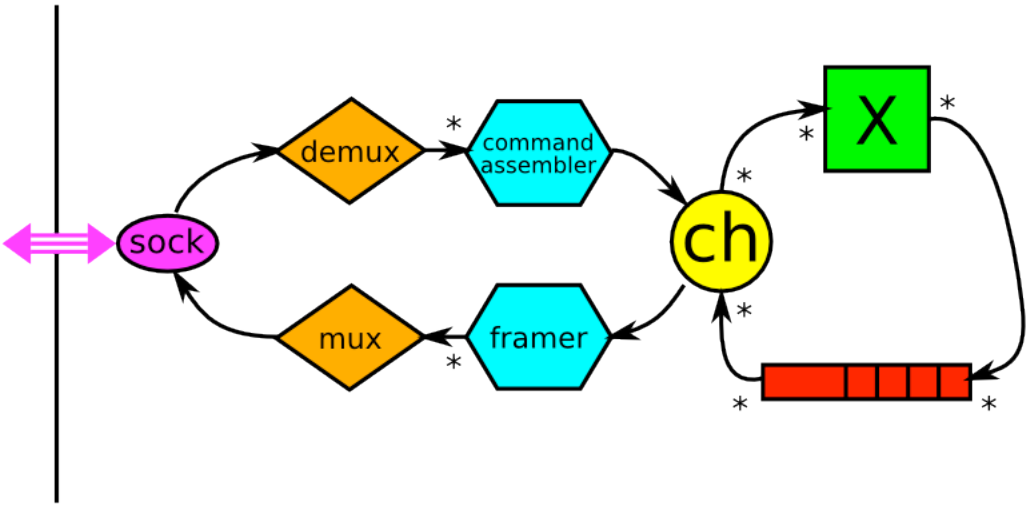
- good architectural fit for AMQP
- concise
- good performance, and it scales
- excellent management and troubleshooting support
- we can hide it

AMQP architecture (revisited)



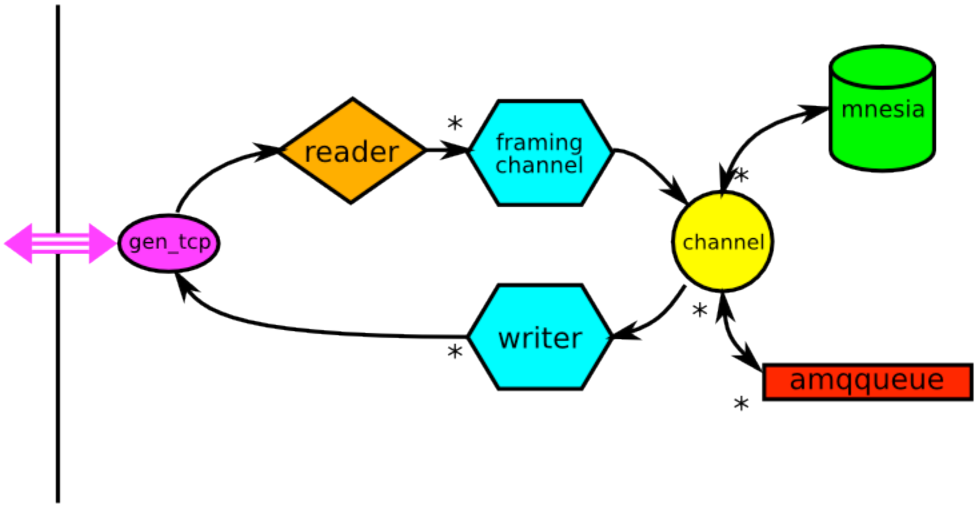
- concentrate on server for rest of the talk
- similarities between structure of client and server, in particular when it comes to protocol handling

AMQP architecture (refined)



- refine relationship between connection and the channels it carries
- introduction of framing - AMQP commands (create queue, delete queue, publish message, etc) are chopped up into frames for interleaving on transport - useful for e.g. large messages

RabbitMQ server design



- domain of discourse of AMQP is very similar to Erlang/OTP
- actual implementation is identical to architecture
- each of the nodes corresponds to a separate Erlang module, and process (taking into account the multiplicities)
- only two minor exceptions - mux, which is handled as part of gen_tcp, and mnesia, which is a collection of processes

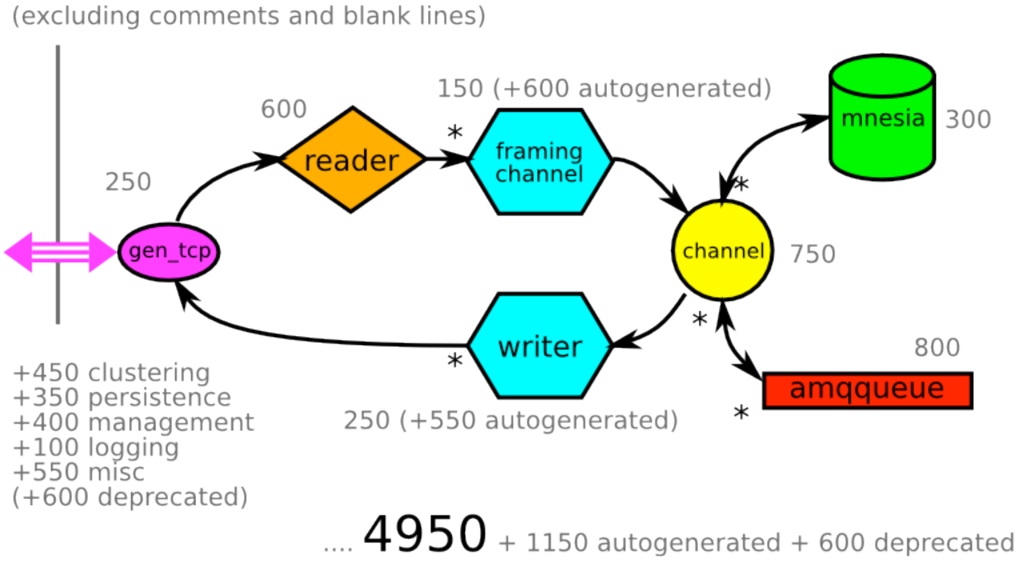
Why Erlang/OTP?

- good architectural fit for AMQP
- concise
- good performance, and it scales
- excellent management and troubleshooting support
- we can hide it

- naturalness of the from the AMQP logical architecture to the Erlang-based design of RabbitMQ mapping results in small, readable codebase where there is a direct correspondence between AMQP features and code fragments
- this makes the code easy to understand, modify and extend - keep up with evolution of spec, try experimental features, build extensions (recall that this is a major focus for RabbitMQ)

So how small is the code base, really? Here are some stats ...

Look, it is tiny!



from left to right:

- networking stack on top of gen_tcp - mostly generic, i.e. not RabbitMQ-specific
- reader - dealing with AMQP connection management, demux, error handling
- framing channel and writer - codec; mostly auto-generated from spec
- channel, amqqueue, exchanges/routing (on top of mnesia) - the AMQP model; bulk of the code

also

- clustering - more about that later
- persistence - messages can be stored durably on disc and survive broker restarts
- management - CLI for administering RabbitMQ
- deprecated code is realms and tickets; will disappear from spec soon

summary - ~5000 LOC
but this is not just down to the naturalness of the mapping between the AMQP architecture and RabbitMQ design. Erlang language features play a big part too. Some examples of that next ...

Binaries

```
decode_method('basic.publish',
              <<F0:16/unsigned,
                F1Len:8/unsigned, F1:F1Len/binary,
                F2Len:8/unsigned, F2:F2Len/binary,
                F3Bits:8>>) ->
    F3 = (F3Bits band 1) /= 0,
    F4 = (F3Bits band 2) /= 0,
    #'basic.publish'{ticket      = F0,
                     exchange    = F1,
                     routing_key = F2,
                     mandatory   = F3,
                     immediate   = F4};
```

This is some of the auto-generated code; decoding an AMQP method frame into an Erlang data structure

- length prefixed strings

It is hard to see how this could be any more concise.

Going the other way - packing terms into a binary blob, rather than unpacking - uses exactly the same syntax, and is just as smooth.

List Comprehension and HOF

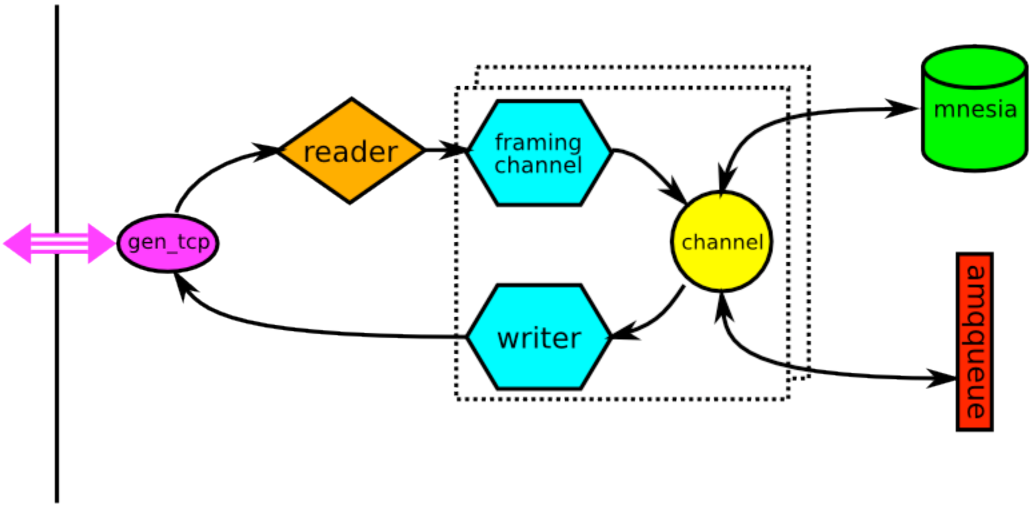
```
upmap(F, L) ->
    Parent = self(),
    Ref = make_ref(),
    [receive {Ref, Result} -> Result end
     || _ <- [spawn(fun() -> Parent ! {Ref, F(X)} end)
              || X <- L]].
```

- Functional programming at work
- While a lot of code for a presentation, it is not a lot for what it does!
- upmap - unordered parallel map

Why Erlang/OTP?

- good architectural fit for AMQP
- concise
- good performance, and it scales
- excellent management and troubleshooting support
- we can hide it

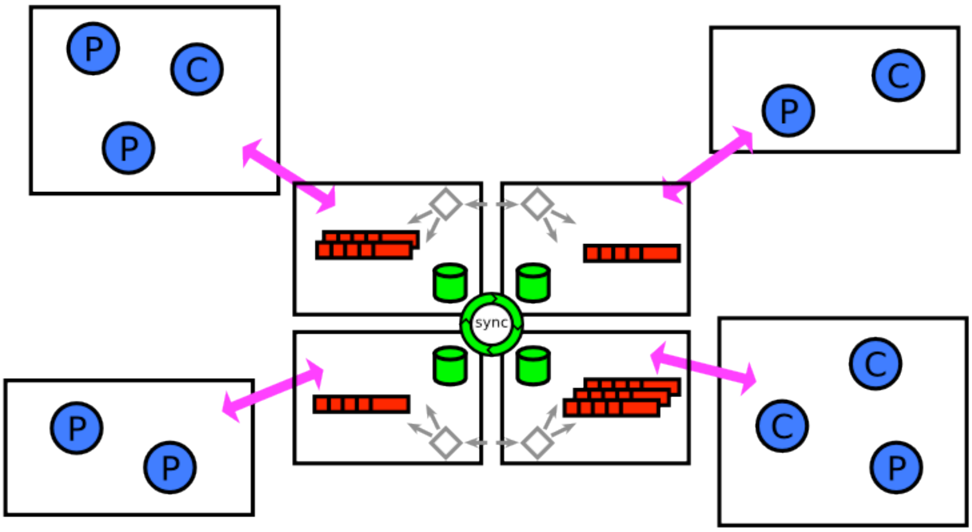
Exploiting Parallelism



three different areas of parallelism that arise naturally from AMQP and that RabbitMQ is exploiting, thanks to Erlang's fine-grained parallelism

- 8-stage pipeline end-to-end between producer and consumer
- parallelism across connections and channels - with queues acting as synchronisation points (queues are about the only stateful part of the core AMQP model)
- queues as active processes

Clustering



logical view remains unchanged, but under the covers

- multiple nodes share routing information
- queues are distributed
- messages are routed efficiently between nodes

Why Erlang/OTP?

- good architectural fit for AMQP
- concise
- good performance, and it scales
- excellent management and troubleshooting support
- we can hide it

- for free, almost
- the Erlang shell in conjunction with the myriad of OTP tools to inspect, trace, profile, debug, modify a running system
- augmented by our own command line tools

...on to some examples of both...

Where has all the memory gone?

```
(rabbit@t)1> [{_, Pid} | _] =
               lists:reverse(
                 lists:sort(
                   [{process_info(P, memory), P}
                    || P <- processes()])).
[{{memory,16434364},<0.160.0>}, ...]
(rabbit@t)2> process_info(Pid, dictionary).
{dictionary,
    [{'$ancestors',
      [rabbit_amqqueue_sup,rabbit_sup,<0.106.0>]},
     {'$initial_call',
      {gen,init_it,
       [gen_server,<0.138.0>,<0.138.0>,
        rabbit_amqqueue_process,
        {amqqueue,
         {resource,<<"/">>,queue,<<"test queue">>},
         false,false,[],[],none},
        []]}}]}
```

just tell the story

Setting up a RabbitMQ cluster

```
rabbit2$ rabbitmqctl stop_app
Stopping node rabbit@rabbit2 ...done.
rabbit2$ rabbitmqctl reset
Resetting node rabbit@rabbit2 ...done.
rabbit2$ rabbitmqctl cluster rabbit@rabbit1
Clustering node rabbit@rabbit2 with [rabbit@rabbit1] ...
done.
rabbit2$ rabbitmqctl start_app
Starting node rabbit@rabbit2 ...done.

rabbit2$ rabbitmqctl status
Status of node rabbit@rabbit2 ...
[...,
 {nodes,[rabbit@rabbit2,rabbit@rabbit1]},
 {running_nodes,[rabbit@rabbit1,rabbit@rabbit2]}]
done.
```

don't talk through it

Why Erlang/OTP?

- good architectural fit for AMQP
- concise
- good performance, and it scales
- excellent management and troubleshooting support
- we can hide it

You do not need to know anything about Erlang to use RabbitMQ

Making Erlang disappear

- AMQP is a *protocol*
- AMQP client libraries exist for many languages and are *broker neutral*
- RabbitMQ is packaged for various platforms
- admin scripts, log files

- two aspects - one to do with the protocol, the other with the operational side
- users write client code, typically using one the available broker-neutral libs; they don't need to care what the server is written in
- one still needs to get a server up and running, configure it and look after it. Need to hide Erlang there too. Hence ...

Why Erlang/OTP? - Summary

- good architectural fit for AMQP
- concise
- good performance, and it scales
- excellent management and troubleshooting support
- we can hide it

protocol handling is an Erlang/OTP sweet spot

Web messaging & the future

Messaging and queueing are ubiquitous. Everybody's doing it.

- Email (SMTP, IMAP/POP3, Mailman, ...)
- Feeds (RSS/Atom, AtomPub)
- Instant Messaging (XMPP, XEP-60 aka. pubsub)
- HTTP, REST, Comet
- Web services, CORBA, RPC
- Databases and File Systems (SQL, FTP, WebDAV, NFS)
- etc. (syslog, ...)

Everybody's doing it; but noone's got it right -- yet.

The same basic ideas of messaging and queueing keep coming up over and over and over again. There's the problem of idempotency to deal with. There's the problem of ensuring your request is heard to deal with. There're all of the issues of network failure to deal with. Lots of variations on the theme are already in wide use -- not just the email infrastructure, but all the tools listed above and more have been pressed into service as ways of getting messages or lists of work-items delivered between separate parties.

Building our XMPP-AMQP gateway taught us a lot. We're starting to be able to see the shape of the essence of messaging.

Messaging at internet scale

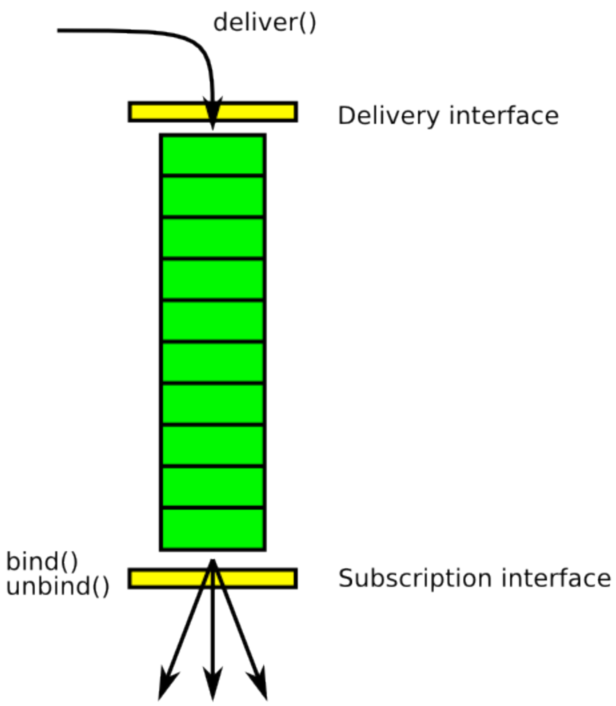
- Encapsulating state (messaging is fundamentally stateless)
    - stateless messages
    - stateful applications
- Responsibility transfer
    - contracts about who is to blame for a lost message; application-level delegation
    - plan for failure!
    - QoS and SLAs appear here
- State synchronisation
    - a very common *application* of messaging
    - idempotent state updates, or responsibility-transferred non-idempotent updates
    - metacircular: use it to build efficient responsibility transfer!

Delegation at the application level, not the transport level. This is what SMTP got wrong: relaying is transport-level delegation. Lifting responsibility-transfer to the application level fixes the issue: the application (not the transport) makes smart decisions about whether to delegate responsibility or not.

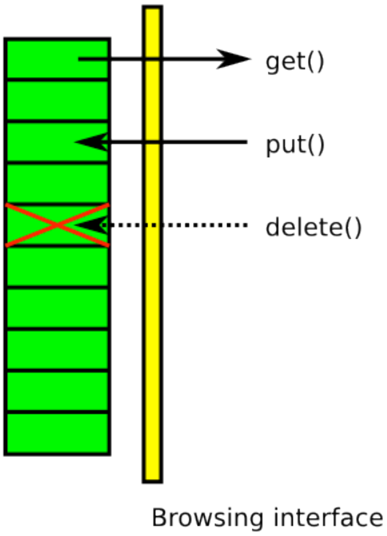
Threefold nature of shared queues

- Transfer: "from the ends"; messages in flight ("messaging")
- Browsing: "from the side"; messages at rest ("queueing")
- Synchronisation makes use of both: uses browsing to build efficient responsible transfer of messages

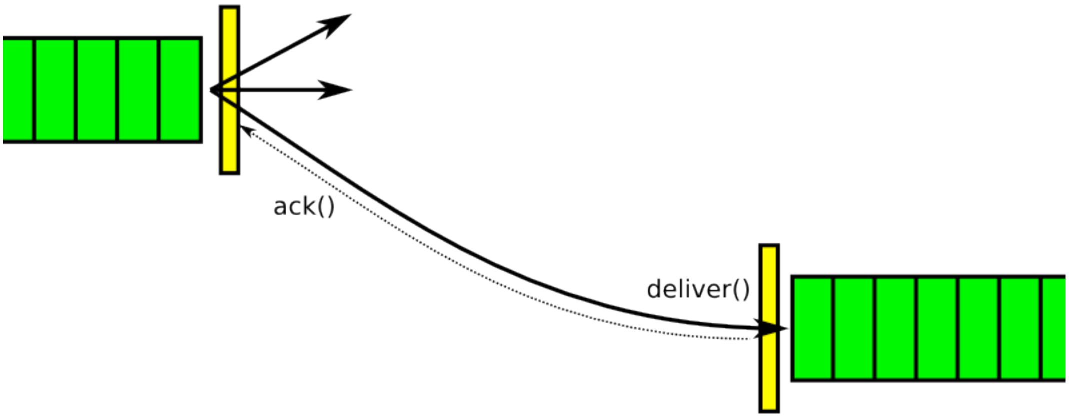
Transfer from the ends



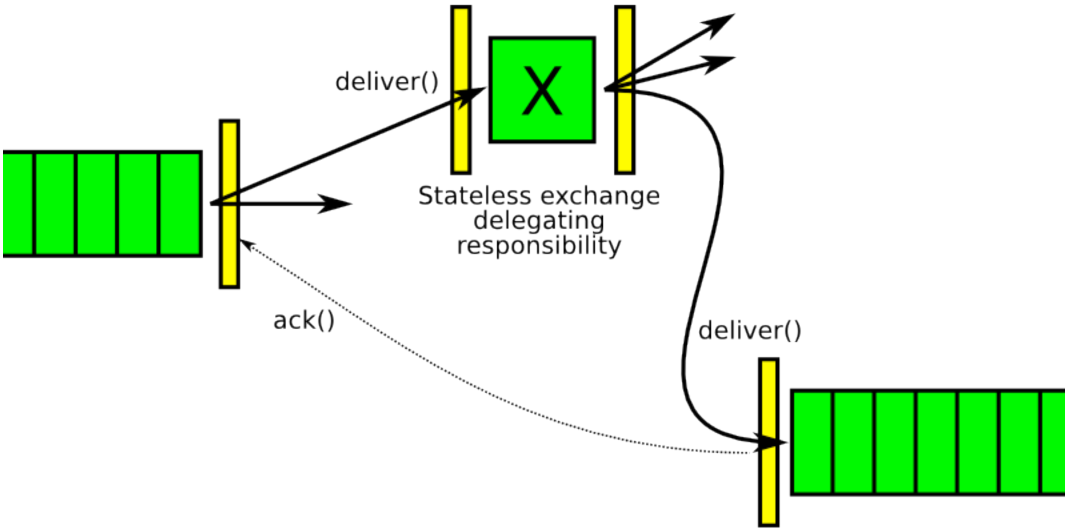
Browsing from the side



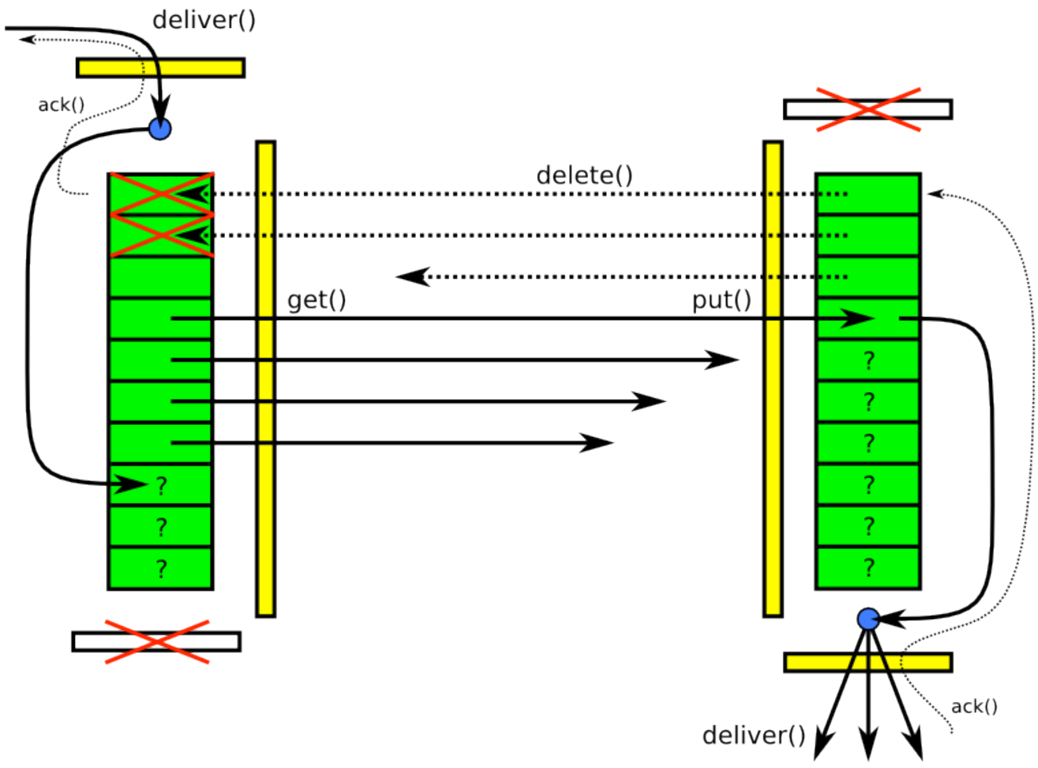
Responsibility transfer (1/2)



Responsibility transfer (2/2)



Synchronisation



Layering is vital here.

- The simplest choice of operations at the low levels assembles to support the next layer
- Proper layering makes implementation easy, as well as pinning down corner-cases in the semantics

The holy grail (1/3)

- AMQP already gets us part-way there:
  - Transfer (delivery, subscription, responsibility)
  - Local transactions (grouping, exactly-/atleast-/atmost-once)
  - Routing (exchanges, filtering, topics)
  - Buffering (queues, relays)
- We want to cleanly integrate the rest:
  - Browsing (list, get, put, delete)
  - Internet-sized (global addressing, federation)
  - Trustable identity (addressing, PKI)
  - Peer to peer
  - Management (statistics, resource limits)

```
WebDAV: and by extension, other (networked) file systems
Some of these things -- duplicate detection, for instance -- are application-level issues; others include the deci
http://wiki.secondlife.com/wiki/Chttp
http://www.mnot.net/drafts/draft-nottingham-http-poe-00.txt
```

The holy grail (2/3)

Protocol	Injection	Browsing	Subscription	Responsibility Transfer	Duplicate detection	Federated / Global	Peer-to-peer / Symmetric (vs. Client-Server)
IDEAL	Y	Y	Y	Optional	Optional	Y	Y
AMQP	Y	N [6]	Y	Y [7]	N	N	Partial [8]
Twitter	Y	Y	Y	N	?	N	N
TCP	Y	N	Y	Y	Y	N	Y
XMPP	Y	N	N	N [1]	N	Y	Y
Atompub	Y	Y	N	Y [3]	N	Y	N
Pubsub	Y	N	Y	Partial [4]	N	Y	N
Atom/RSS	N	Y	N	N	--	Y	Y
POE, CHTTP	Y	N	N	Y	Y	Y	N
WebDAV	Y	Y	N	Y	--	Y	N
SMTP	Y	N	N	Y	N	Y	N
IMAP	Partial	Y	Y	Partial [3]	--	N	N
Mailman	N	Y	Y	N	--	N	N
POP3	N	Y	N	Partial [3]	--	N	N
Bittorrent	N	Y	N	N	Y	N	N
HTTP	N	N	N	Y	N	Y	N
Comet	N	N	N	N	N	N	N

Talk about: Resp xfer; federation; peer-to-peer

1 There are XEPs that touch on this area
2 The "offline storage" modules often provided act as a kind of buffer
3 Deletion can act as an acceptance of responsibility
4 There's an acknowledgement for publication only, not delivery
5 Aggregators sort of do this
6 Queue browsing is being worked on in the AMQP WG
7 AMQP 0-10 introduces a form of responsibility transfer
8 0-10's message transfers are quasi-symmetric; the wire-protocol is symmetric; the model is not
9 Management and monitoring is being worked on in the AMQP WG

The holy grail (3/3)

Protocol	Unicast Routing	Multicast / Complex Routing	Relaying	Queueing / Buffering	Presence / Lifecycle Events	Management / Monitoring	Trust model
IDEAL	Y	Y	N	Y	Y	Y	???
AMQP	Y	Y	N	Y	N	Y [9]	--
Twitter	Y	Y	N	Y	N	--	--
TCP	N	N	N	Y	N	Some	--
XMPP	Y	N	N	N [2]	Y	N	DNS
Atompub	N	N	N	Y	N	--	--
Pubsub	N	Y	N	N	N	N	XMPP
Atom/RSS	N	N [5]	N [5]	Y	N	--	--
POE, CHTTP	N	N	N	N	N	--	--
WebDAV	N	N	N	N	N	N	--
SMTP	Y	N	Implied	N	N	N	complicated
IMAP	N	N	N	Y	N	N	--
Mailman	N	Y	N	N	N	N	--
POP3	N	N	N	Y	N	N	--
Bittorrent	N	N	N	N	N	N	--
HTTP	N	N	N	N	N	N	--
Comet	N	N	N	N	N	N	--

Talk about: relaying; presence

1 There are XEPs that touch on this area
2 The "offline storage" modules often provided act as a kind of buffer
3 Deletion can act as an acceptance of responsibility
4 There's an acknowledgement for publication only, not delivery
5 Aggregators sort of do this
6 Queue browsing is being worked on in the AMQP WG
7 AMQP 0-10 introduces a form of responsibility transfer
8 0-10's message transfers are quasi-symmetric; the wire-protocol is symmetric; the model is not
9 Management and monitoring is being worked on in the AMQP WG

Data languages (1/3)

AMQP's current wire protocols and data languages perform well, but are missing a few features that will be required as AMQP grows to internet scale.

The *ideal* data language:

- Schemas, type definitions
- Abstract syntax essential: different concrete syntaxes useful in different contexts
  - for optimisation
  - for scaling down, scaling up, and scaling to the web
  - for interoperation with "legacy" protocols
- Extensibility
- Friendly to crypto (canonical form for signability)
- Widely-supported protocol already in use: ideal!

Human-readable concrete syntax is a debuggable representation

Data languages (2/3)

Language	Schemas	Multiple concrete syntaxes	Parseable without schema	Metacircular	Extensible	Unbounded integers (bignums)	Good support for binary data
IDEAL	Y	Optional	Y	Y	Y	Y	Y
ASN.1	Y	Y	Y	N	Y	Y	Y
Protocol Buffers	Y	N	Y	N [5]	Y	Y	Y
XML	Y	Y	Y	Y	Y	Y	N
JSON	N	N	Y	N	Y	Y	N
SPKI	N	N	N [2]	N	Y	Y	Y
AMQP	Y	N	N	N	N	N	Y

[1] Ordering of key-value pairs within tables is missing; otherwise, it's more-or-less canonical as it stands

[2] The "display type" attached to a byte-string compensates partially for this, but numbers are not distinct

[3] SPKI doesn't care what interpretation you place on the bytes it packages

[4] XML schemas don't line up with common programming-language data types very well (not even RelaxNG)

[5] But it could be!

[6] There are good tools, but they're commercial. C and C++ are well served; Java less so; other languages sti

Data languages (3/3)

Language	Canonical form defined	Multiple programming-language support	In wide use already	Unicode support	"Efficient" concrete syntax	Good tools widely available already	Simple to use, understand, implement
IDEAL	Y	Y	Y	Y	Y	Y	Y
ASN.1	Y	Y	Y	Y	Y	N [6]	N
Protocol Buffers	N	Y	Y	Y	Y	Y	Y
XML	Y	Y	Y	Y	N	Y	N [4]
JSON	N	Y	Y	Y	N	Y	Y
SPKI	Y	Y	N	N [3]	Y	N	Y
AMQP	N [1]	Y	N	Y	Y	N	N

[1] Ordering of key-value pairs within tables is missing; otherwise, it's more-or-less canonical as it stands

[2] The "display type" attached to a byte-string compensates partially for this, but numbers are not distinct

[3] SPKI doesn't care what interpretation you place on the bytes it packages

[4] XML schemas don't line up with common programming-language data types very well (not even RelaxNG)

[5] But it could be!

[6] There are good tools, but they're commercial. C and C++ are well served; Java less so; other languages sti

Learning from XMPP and AMQP

Implementing our XMPP-AMQP gateway provided a good foundation for the preceding analysis. XMPP and AMQP complement each other: the areas where they differ give strong design and layering clues.

- XMPP has internet-scale addressing and federation
- XMPP has a presence model
- AMQP mandates a store-and-forward capability
- AMQP has programmable routing and filtering (using exchanges and bindings)
- AMQP has local transactions and responsibility transfer

What next for AMQP?

- community - wiki, mailing lists, expert involvement
- responsibility transfer
- management and monitoring
- federation and global addressing
- encryption, signing, trust and identity

(Conclusion of this segment) We've been looking at the ways messaging is in use on the internet today, and incorporating those patterns and techniques into the development of RabbitMQ and of AMQP itself.

What next for RabbitMQ?

- community - wiki, hg, dev list, AMQP
- AMQP test suite and interop
- modularity - packages and plugins
- more clients, adapters, gateways and transports
- features - distributed queues, federation, management
- features - tracking the AMQP standard
- performance improvements and bug fixes

www.rabbitmq.com

RabbitMQ is available; it's in use by customers; grab a copy and try it out! We also have a public server running; join the mailing list to report any problems or suggestions.