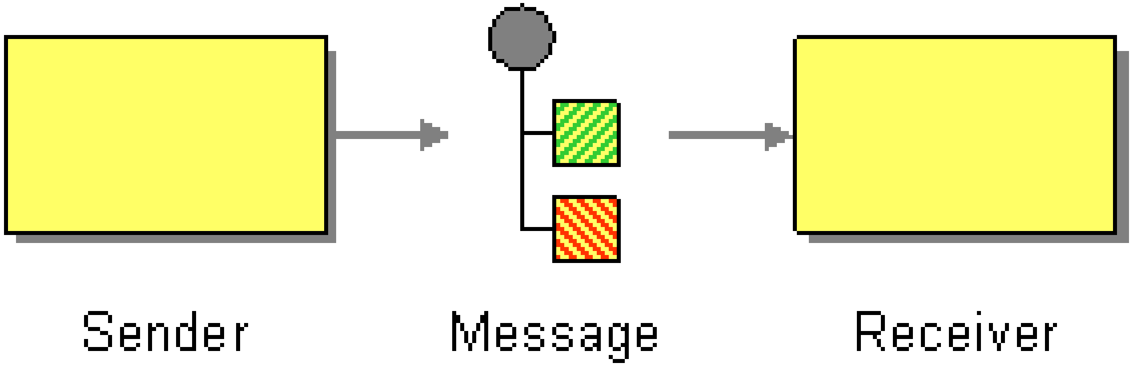
Messaging and *job queues* in Go

Gerrit Renker

Talk Structure

1. What is MOM, and why?
2. Enterprise MOMs
3. High-speed MOMs
4. Go MOMs
5. Wrap-up

Messages Decouple



Message **O**riented **M**iddleware

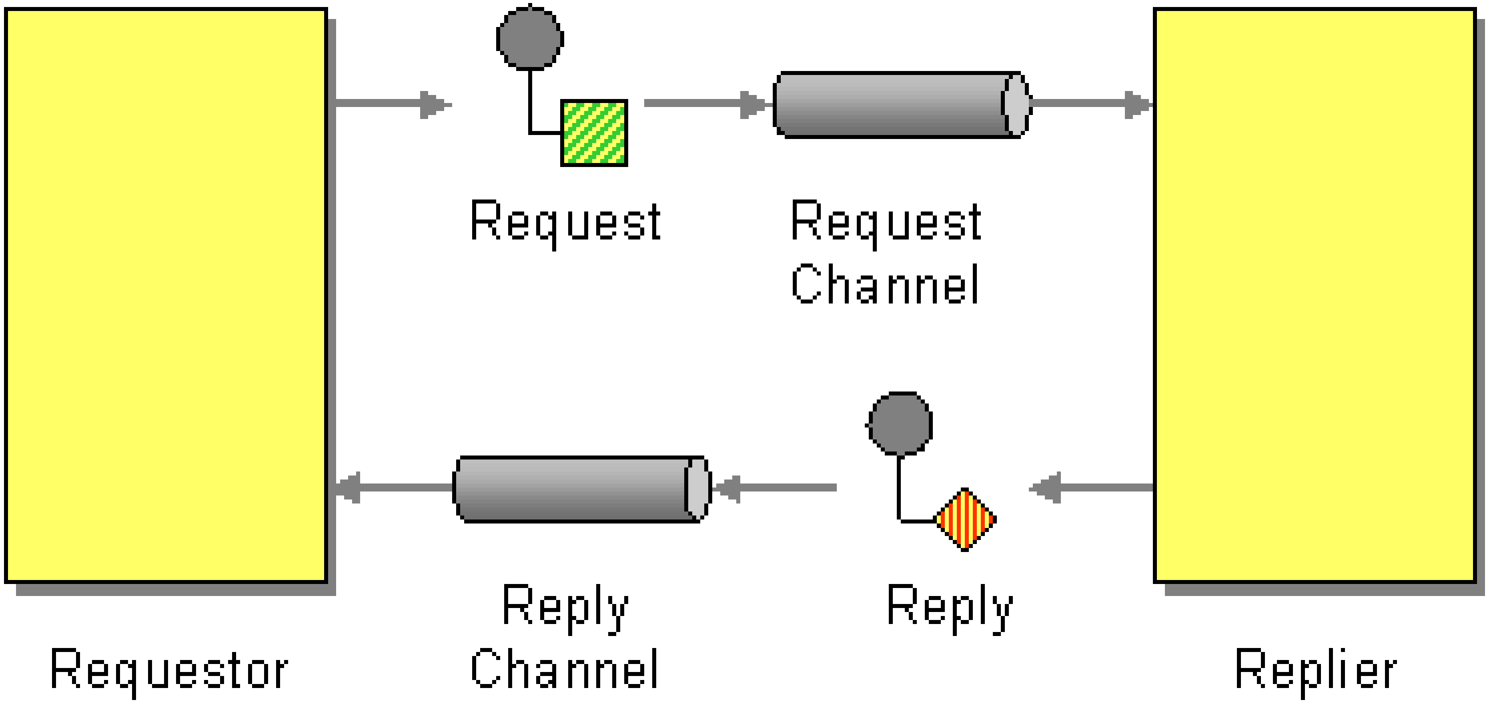
MOM= any middleware providing messaging facilities

Message **O**riented **M**iddleware

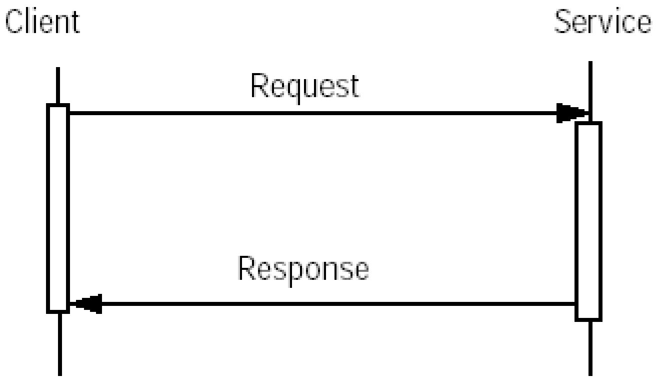
MOM= any middleware providing messaging facilities

- *asynchronous: vs “while u wait”*
- *decoupled: mailbox* functionality
- *reliable:* store & forward, guarantees
- *scalable:* distributed, load balancing

Request/Reply (RPC)



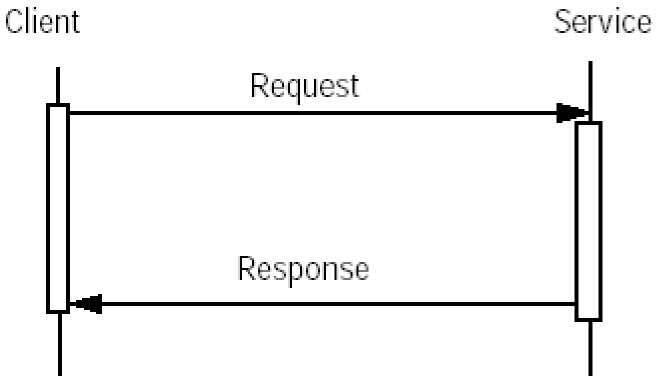
MOM vs RPC



RPC: [telephone]

- synchronous, tighly coupled processing
- caller is blocked until callee returns
- failure if one endpoint becomes unavailable

MOM vs RPC



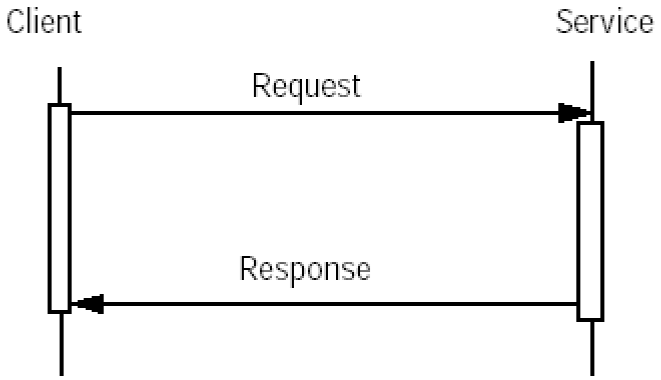
RPC: [telephone]

- synchronous, tighly coupled processing
- caller is blocked until callee returns
- failure if one endpoint becomes unavailable

MOM: [mail]

- asynchronous, simple “*fire & forget*” call
- result is queried via message (push or pull)

MOM vs RPC



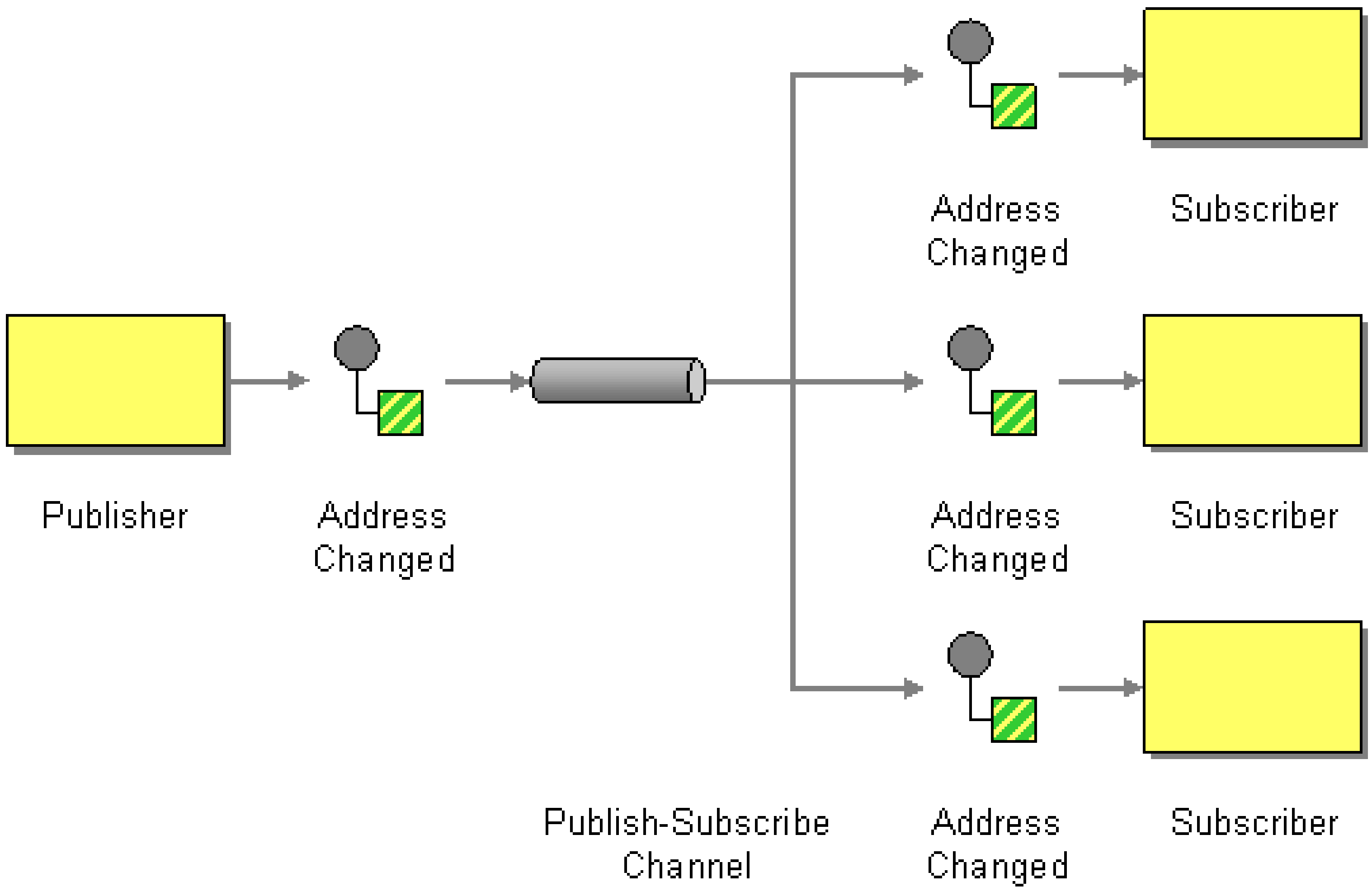
RPC: [telephone]

- synchronous, tighly coupled processing
- caller is blocked until callee returns
- failure if one endpoint becomes unavailable

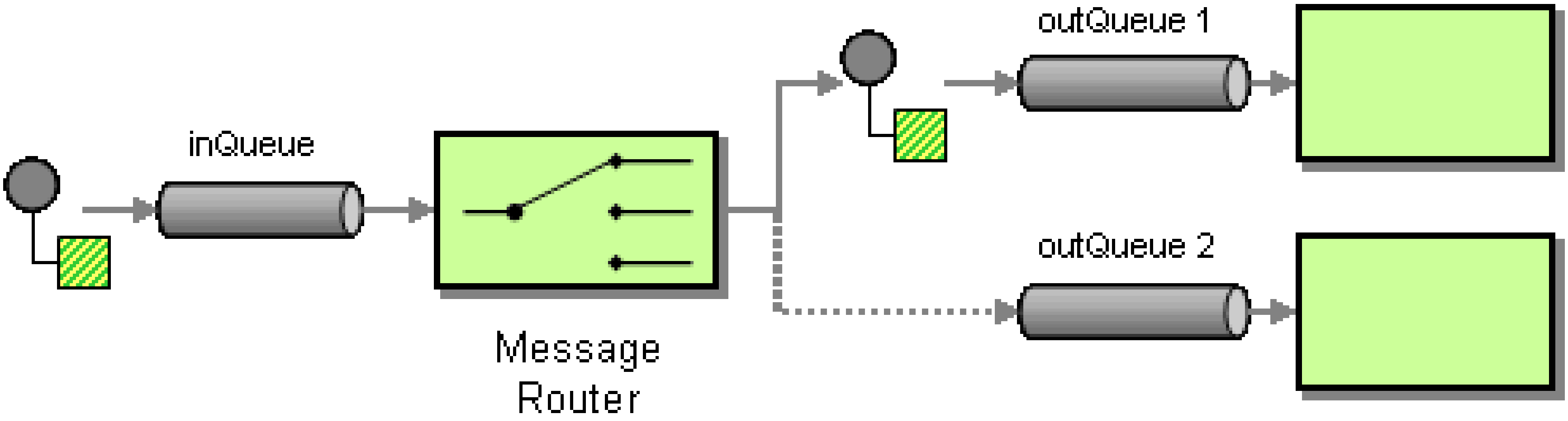
MOM: [mail]

- asynchronous, simple “*fire & forget*” call
- result is queried via message (push or pull)

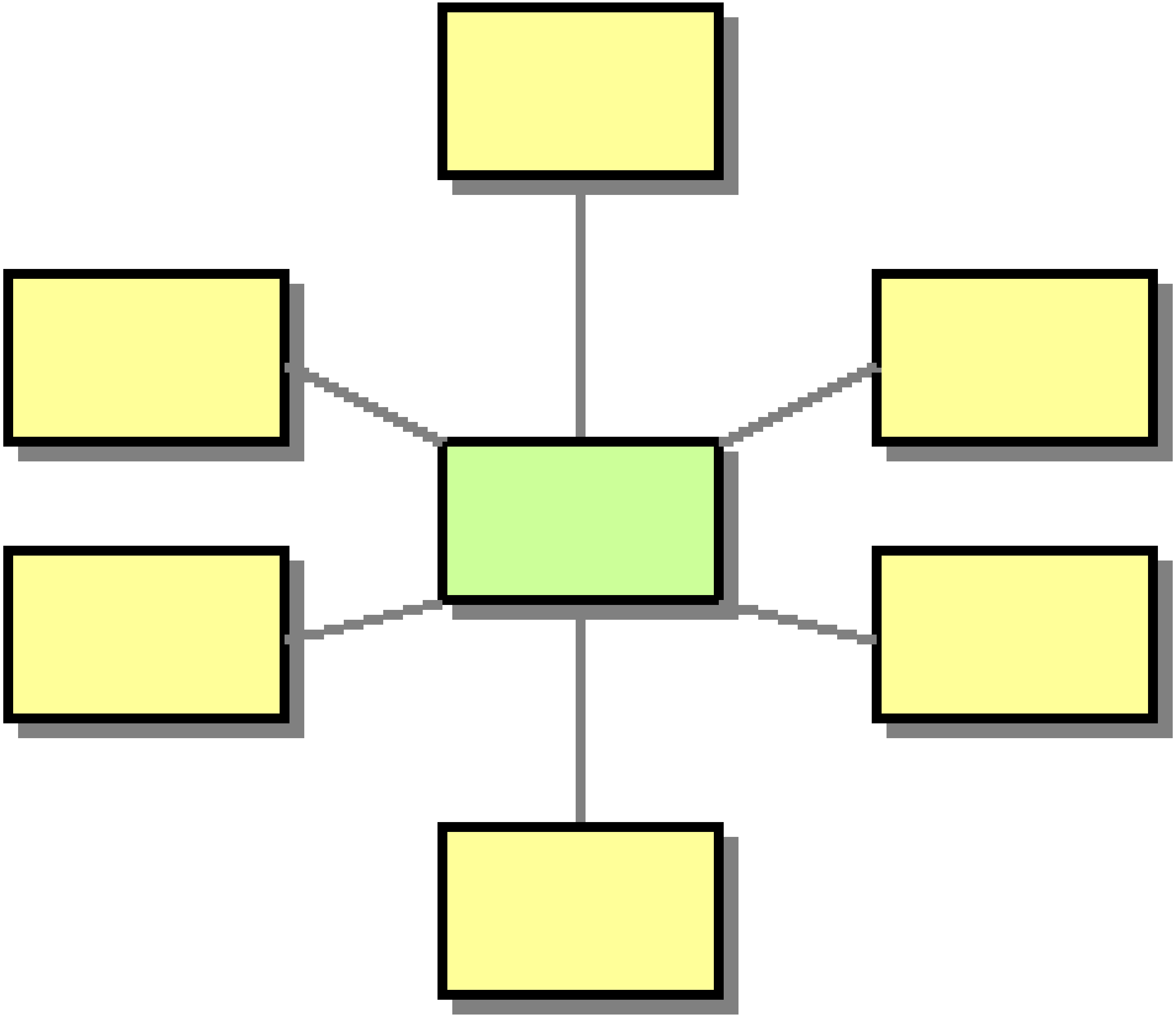
MOM service: pub/sub queue

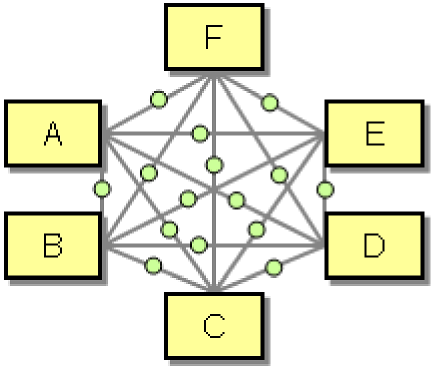


MOM service: routing



MOM service: broker





MOM characteristics

- *Queues* decouple:
  - FIFO or priority (job) queues
  - *Message Broker* turns queue into *service*

MOM characteristics

- *Queues* decouple:
  - FIFO or priority (job) queues
  - *Message Broker* turns queue into *service*
- Endpoint models:
  - point-to-point (1:1, peer-to-peer, request/reply)
  - publish/subscribe (m:n)

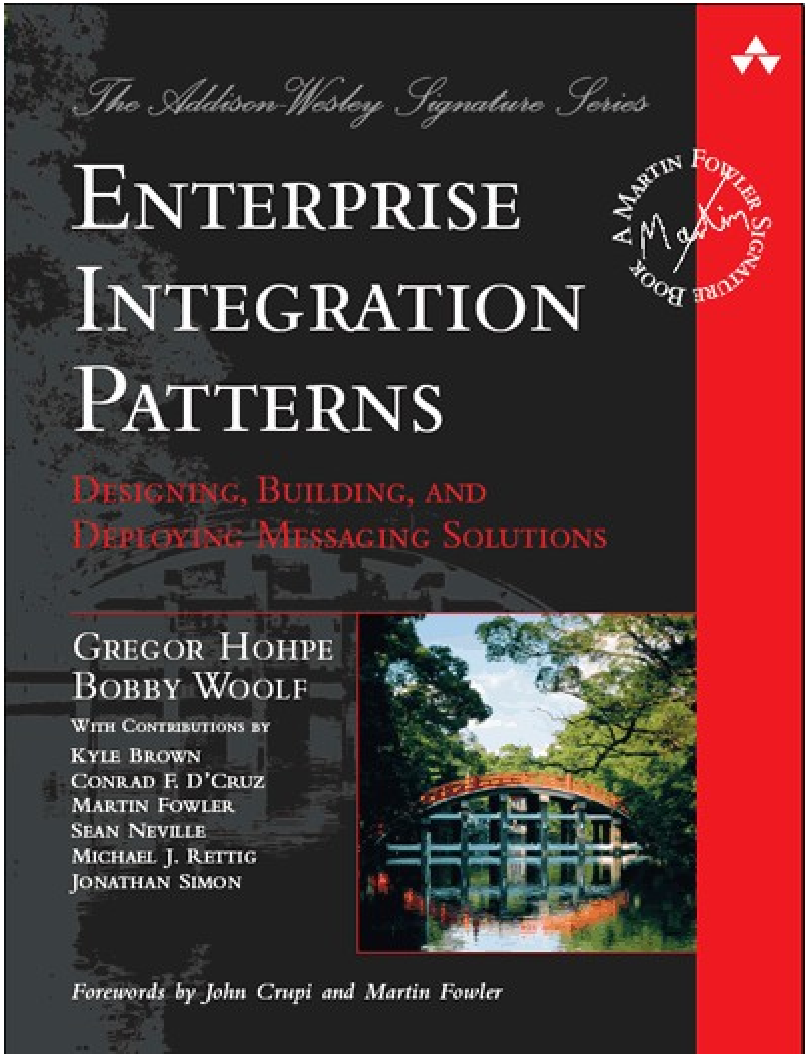
MOM characteristics

- *Queues* decouple:
  - FIFO or priority (job) queues
  - *Message Broker* turns queue into *service*
- Endpoint models:
  - point-to-point (1:1, peer-to-peer, request/reply)
  - publish/subscribe (m:n)
- QoS guarantees:
  - *at-most*-once (<2, “best effort”)
  - *at-least*-once (>0, needs idempotent receiver)
  - once-*and-once-only* (== 1)

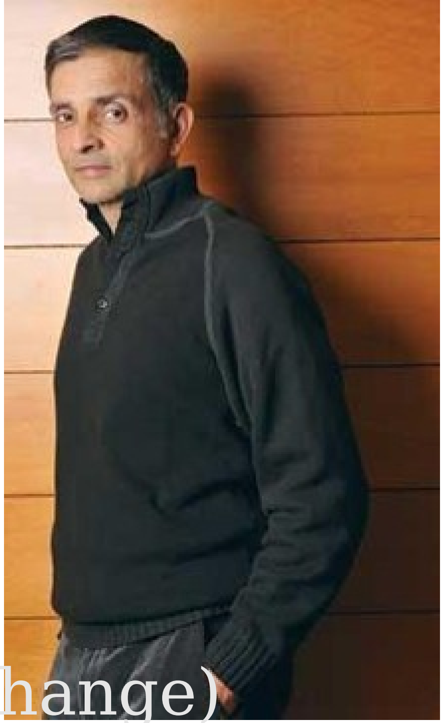
MOM uses and benefits

- *Reduce impact of change (EAI)*
  - *tight coupling*: 1 change => *n* other changes
  - *loose coupling*: 1 change in 1 place
- *Simplified application design*
  - messaging component external
  - language-agnostic

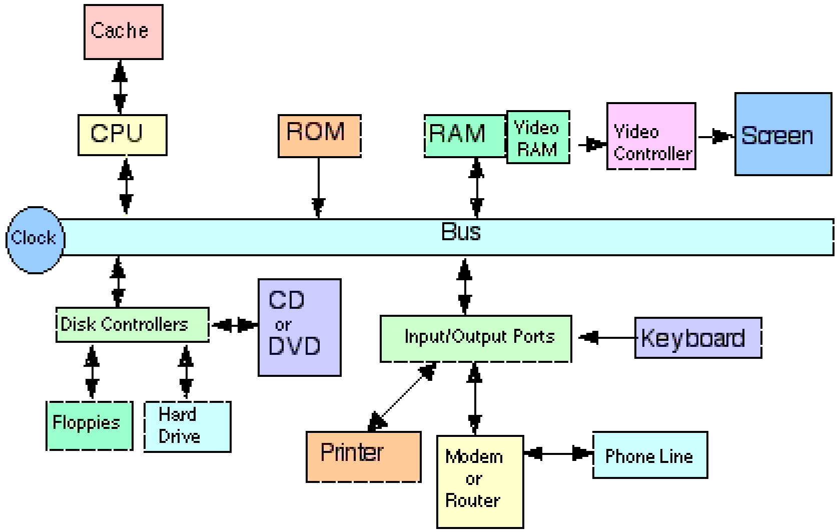
2. Enterprise MoMs

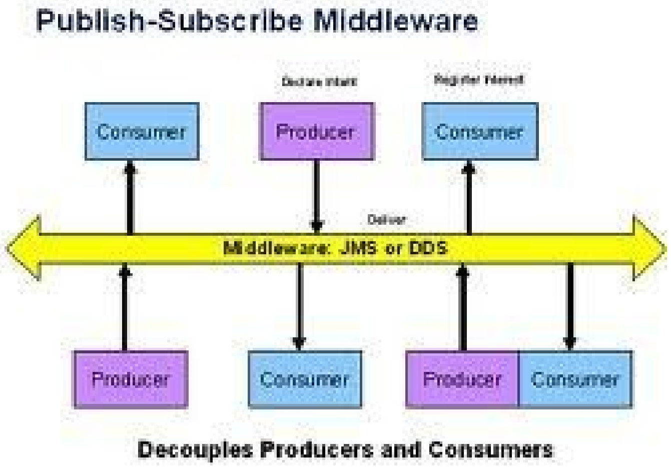

The Addison-Wesley Signature Series
Enterprise Integration Patterns
Designing, Building, and Deploying Messaging Solutions
A Martin Fowler Signature Book
Martin
Gregor Hohpe
Bobby Woolf
With Contributions by
Kyle Brown
Conrad F. D'Cruz
Martin Fowler
Sean Neville
Michael J. Rettig
Jonathan Simon
Forewords by John Crupi and Martin Fowler

The Information Bus

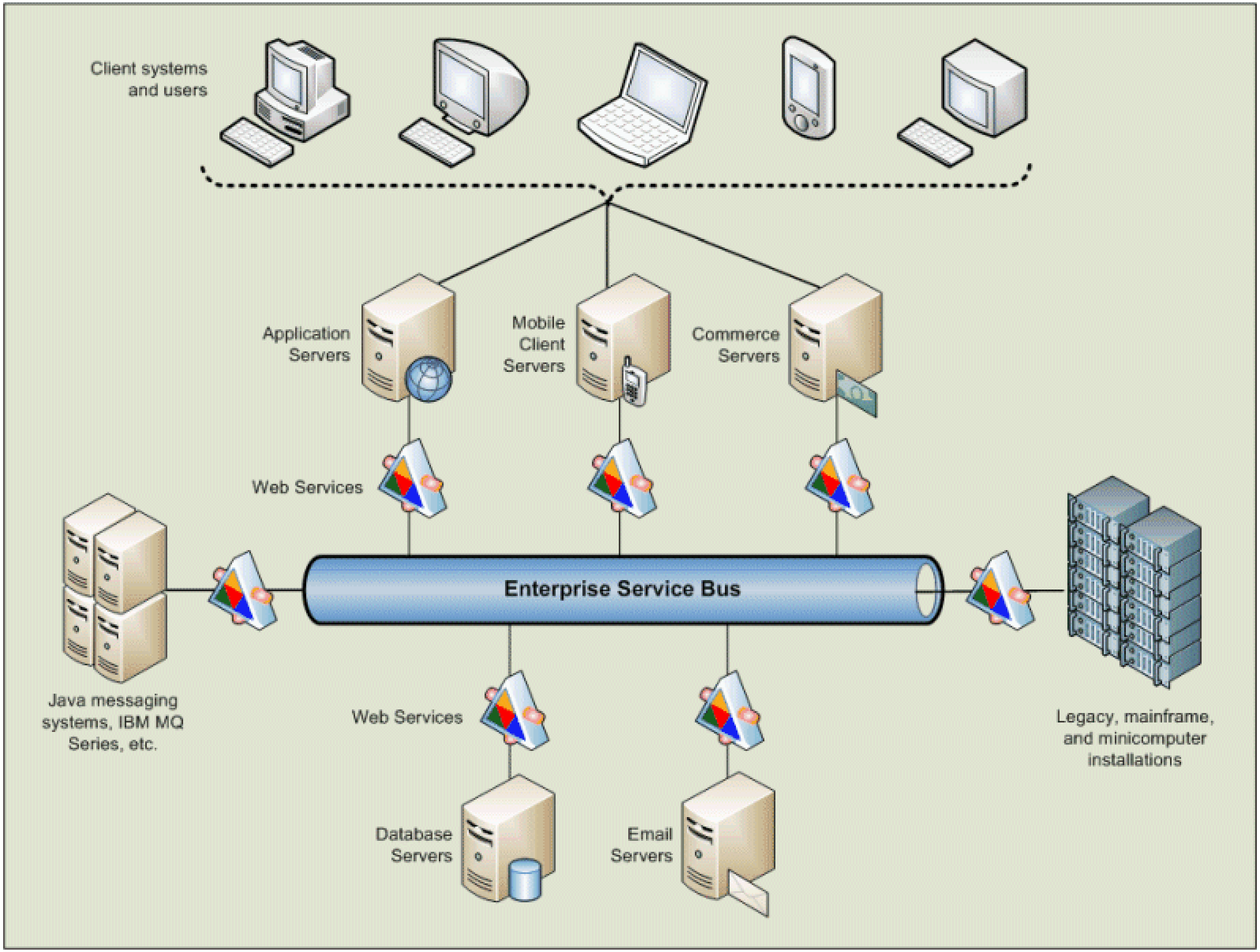


- *1983: pub/sub origin* at TIB
- Uses *software message bus* for
  - continuous operation (24/7 availability)
  - dynamic evolution (room for extension & change)
  - legacy systems (need to talk to new software)



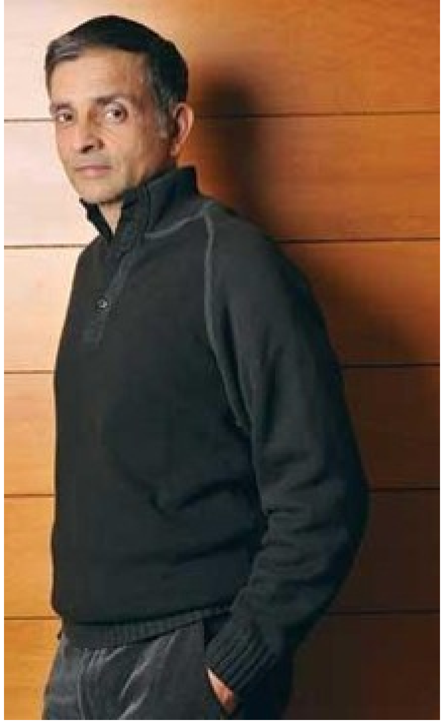


From TIB to ESB

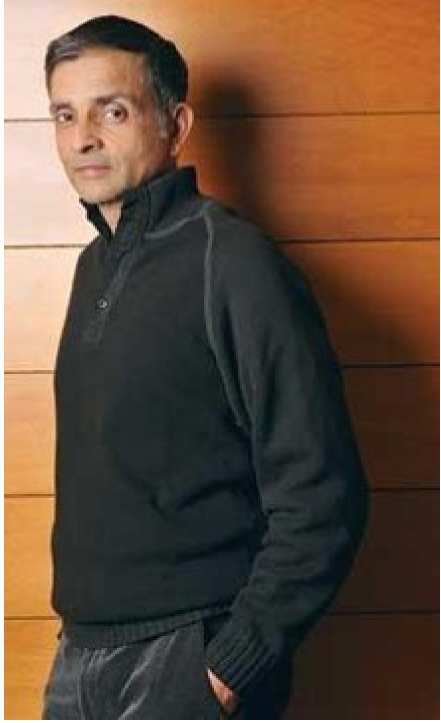


From TIB to ESB

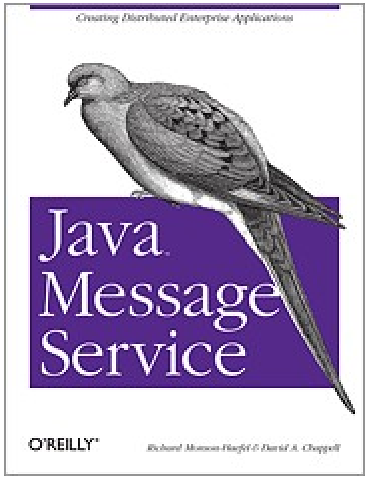
- 1985: Goldman Sachs
- 1994: Reuters buys TIBCO



From TIB to ESB

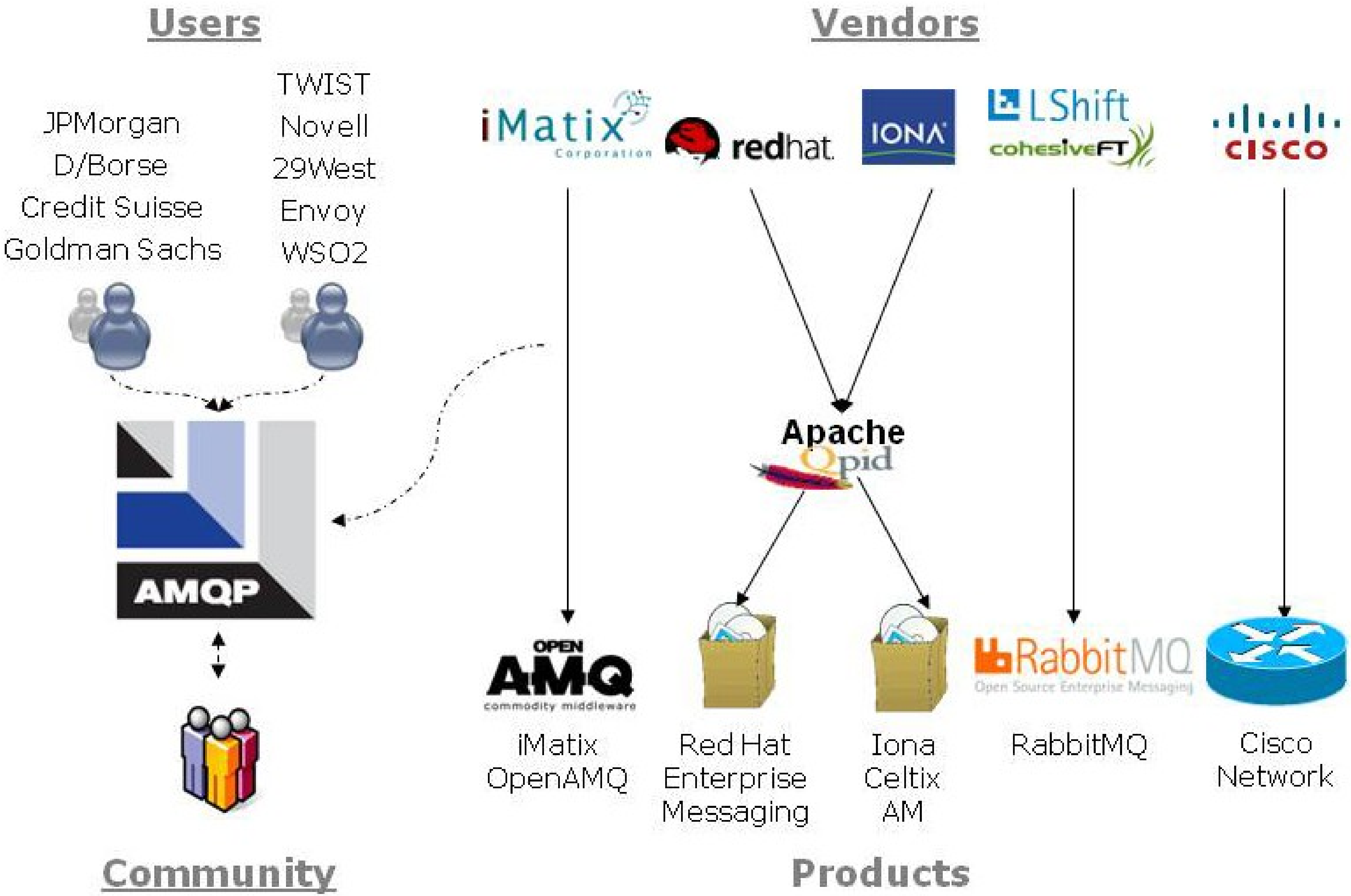


- 1985: Goldman Sachs
- 1994: Reuters buys TIBCO
- 1993: *IBM* MQseries (later Websphere)
- 1997: *Microsoft* Message Queue (MSMQ)
- 2001: *Java* Message Service
  - tries to solve vendor-lock in
  - using JDBC-like “*glue API*”
  - JSR-000914


Creating Distributed Enterprise Applications
Java Message Service
O'REILLY
Richard Monson-Haefel & David A. Chappell

MOM Standardization

AMQP Working Group



Users

JPMorgan
D/Borse
Credit Suisse
Goldman Sachs

TWIST
Novell
29West
Envoy
WSO2

Vendors

Community

iMatix
OpenAMQ

Red Hat
Enterprise
Messaging

Iona
Celtix
AM

RabbitMQ

Cisco
Network

Products

AMQP Defining Features

”enough MOM semantics [...] to meet the needs of most commercial systems”

- *extends* popular JMS semantics

AMQP Defining Features

”enough MOM semantics [...] to meet the needs of most commercial systems”

- *extends* popular JMS semantics
- unlike JMS, defines *wire-level protocol*:
  - *flow-controlled* communication
  - *delivery guarantees* (including exactly-once)
  - *authentication/encryption* based on SASL, TLS
- supports *high-speed trading*

AMQP

AMQP
Advanced Message Queuing Protocol

- 2003: JPMorgan Chase (John O'Hara)
- 2004: iMatix C broker (Pieter Hintjens)
- 2005: Apache Qpid (Java/C++)
- 2006: RabbitMQ, London (Erlang/OTP)

AMQP

- 2003: JPMorgan Chase (John O'Hara)
- 2004: iMatix C broker (Pieter Hintjens)
- 2005: Apache Qpid (Java/C++)
- 2006: RabbitMQ, London (Erlang/OTP)
- 2008: AMQP 0.9.1 (“stable”)
- 2011: AMQP 1.0 (“testing”)
- 2012: OASIS standard
- 2014: ISO/IEC 19464

AMQP Basics

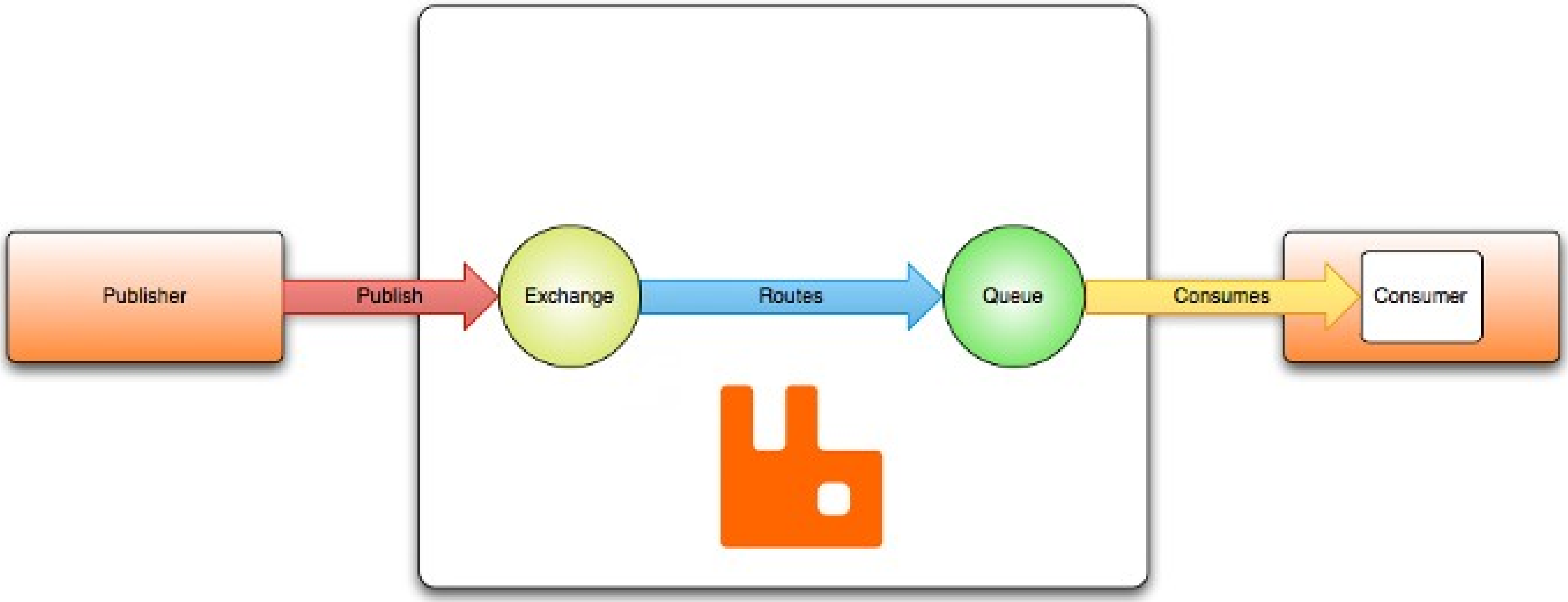
- defines *messaging capabilities*
  - *stores* messages if receiver offline
  - message *tags/topics* used for routing

AMQP Basics

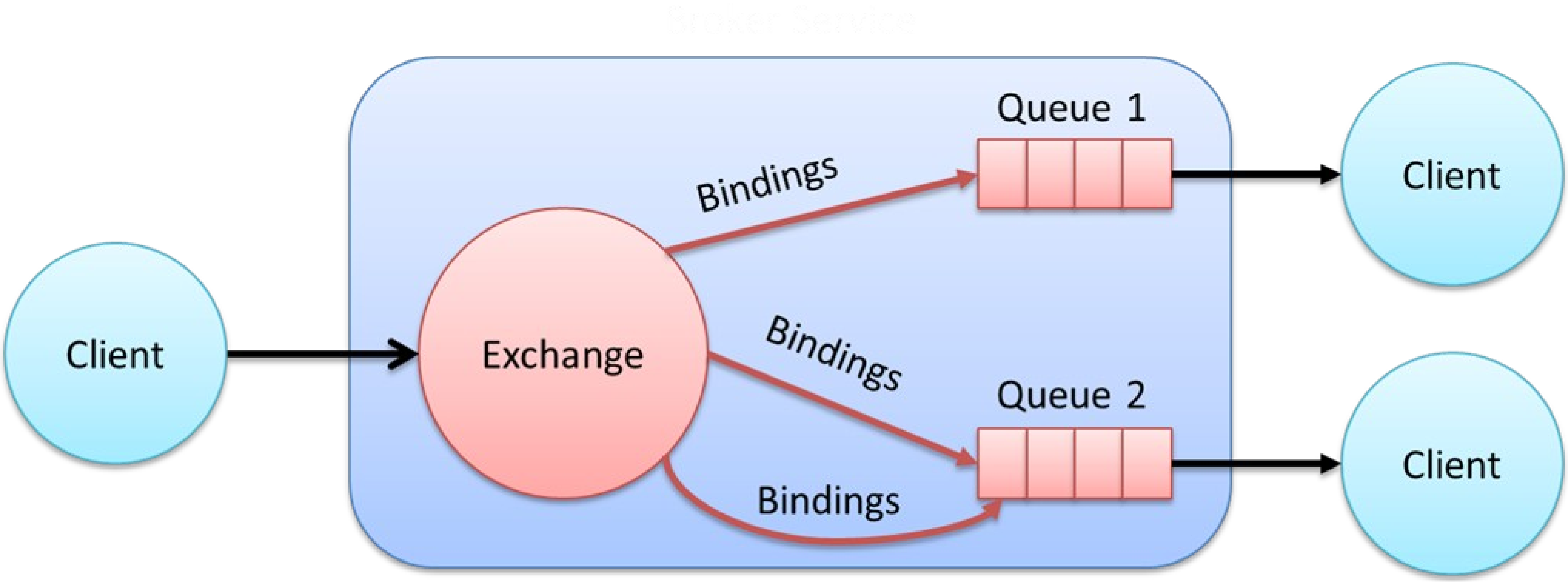
- defines *messaging capabilities*
  - *stores* messages if receiver offline
  - message *tags/topics* used for routing
- components:
  - exchange: *producer* publishes message to
  - queue: stores message for *consumer*
  - binding: governs message *routing* from exchange to particular queue(s)

AMQP Basics

"Hello, world" example routing



AMQP Basics


Broker Service
Queue 1
Bindings
Client
Client
Exchange
Bindings
Queue 2
Bindings
Client

Im in yr serverz,
queueing yr messagez

RabbitMQ

- implements AMQP 0.9.1
- rock-solid, stable
- based on Erlang/OTP
- *see talk:*

  “Evolution of AMQP at Soundcloud”
  Sebastian Ohm, Tomás Senart
  Erlang User Conference 2013

AMQP to Go

1. Soundcloud RabbitMQ 0.9.1
   https://github.com/streadway/amqp
2. RabbitMQ Go examples:
   https://github.com/rabbitmq/rabbitmq-tutorials/tree/master/go
3. Via RabbitMQ HTTP mgmt API:
   https://github.com/michaelklishin/rabbit-hole
4. Relay AMQP wrapper:
   https://github.com/armon/relay
5. Apache Qpid Go examples:
   https://github.com/apache/qpid-proton

Standards after AMQP ...

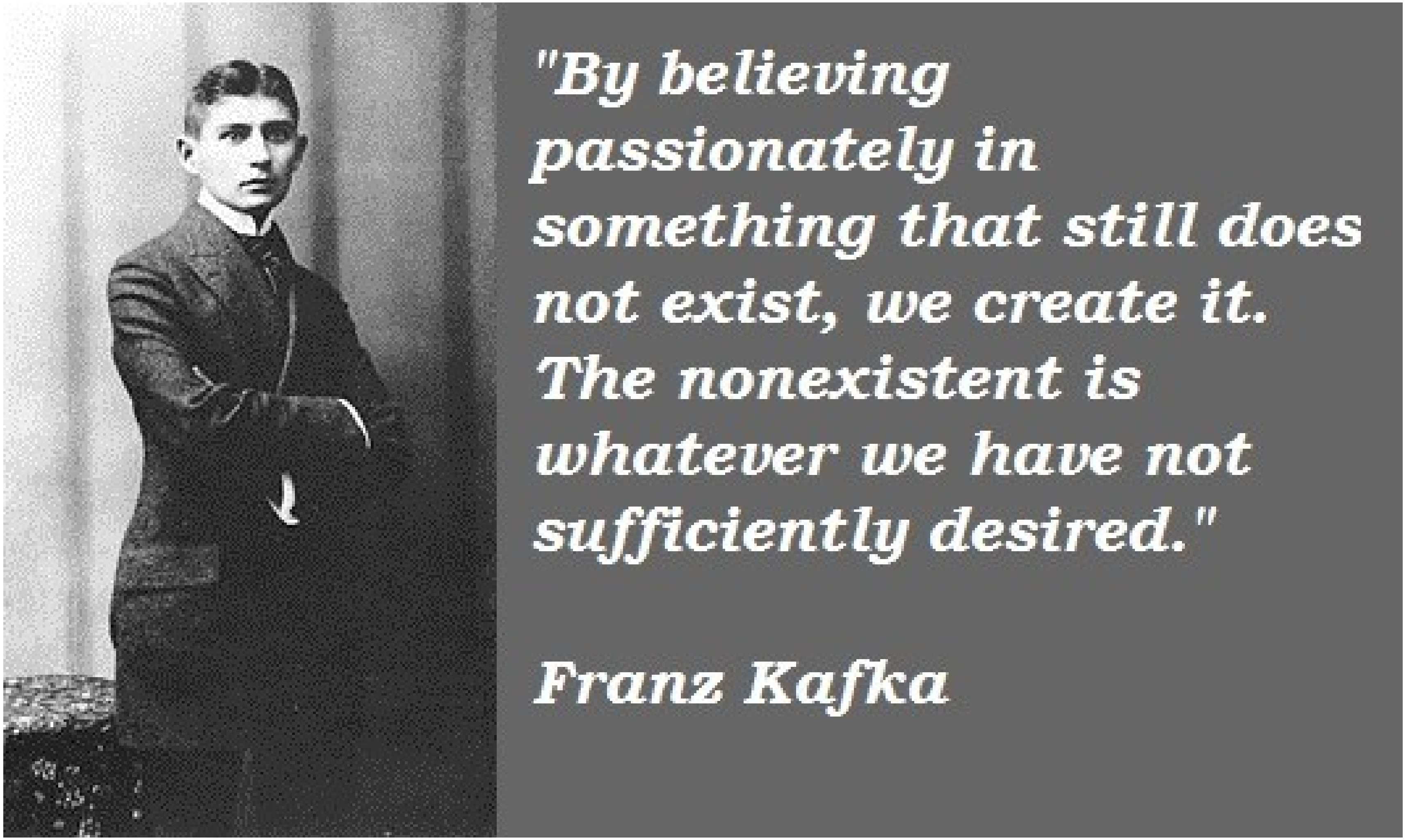
- AMQP 1.0 spec *drops broker details*

- SOA demand for *reactive systems*:
  - willing to *trade off* speed for consistency and/or reliability
  - 0MQ, LinkedIn, Twitter, Facebook ...

ØMQ

- AMQP is a protocol
- ØMQ is a messaging library:
  - no message queues
  - does not define a broker
  - low-level socket/messaging primitives
  - abstractions for messaging patterns
  - focus on high-performance
- Go clients:
  - https://github.com/zeromq/goczmq ZMQ v3
  - https://github.com/pebbe/zmq4 ZMQ v4
- nanomsg/libmill (go-like concurrency)

Kafka



"By believing passionately in something that still does not exist, we create it. The nonexistent is whatever we have not sufficiently desired."

Franz Kafka

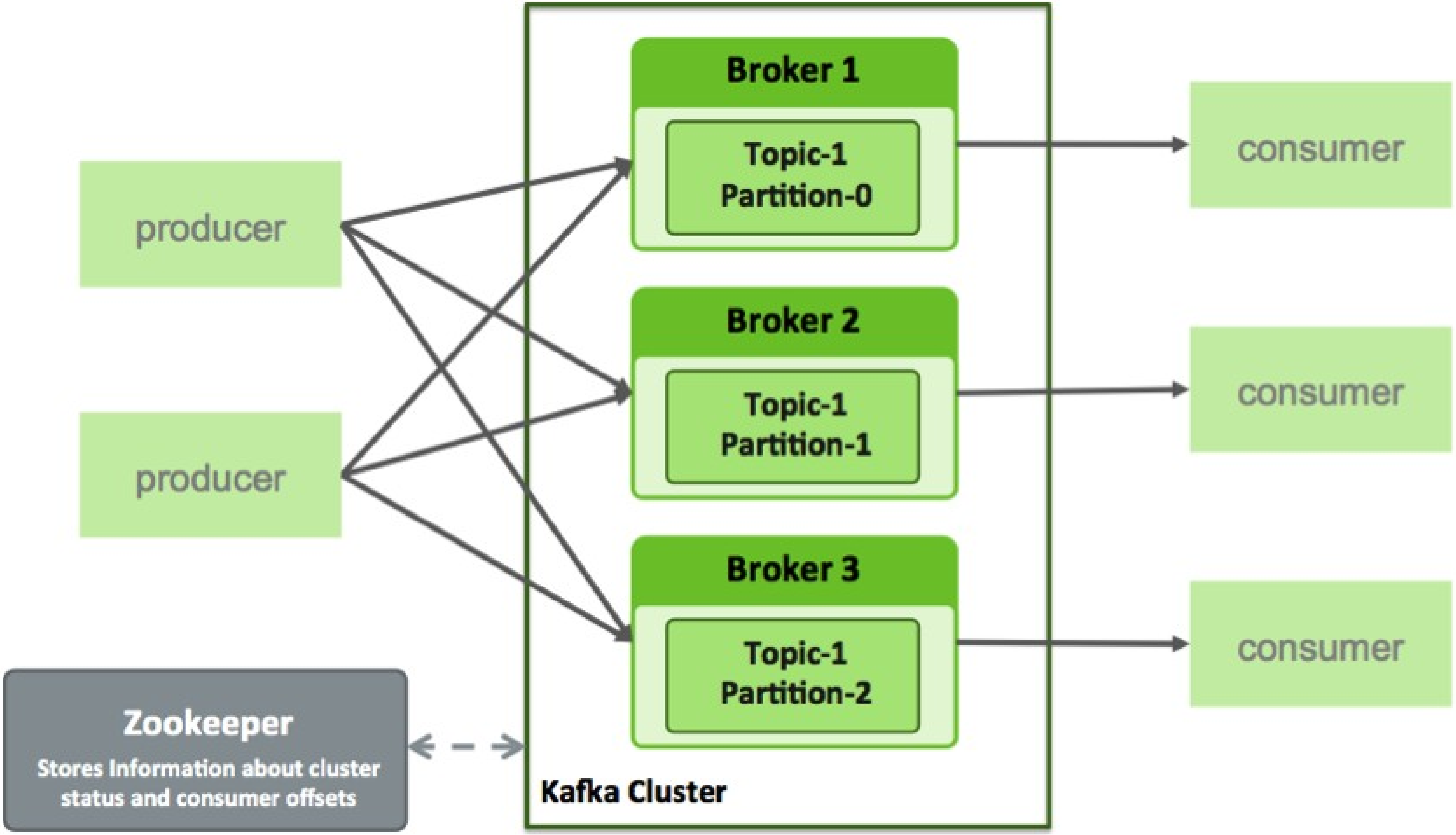
Apache Kafka

- *broker developed at LinkedIn*:
  - to help with high-speed Hadoop ingest
  - sustain *“firehose throughput”*

Apache Kafka

- *broker developed at LinkedIn*:
  - to help with high-speed Hadoop ingest
  - sustain *“firehose throughput”*
- *unique features:*
  - each *topic* is treated as a *commit log*
  - data partitioning
  - scalable & *low latency*
  - messages are *persisted* on disk and
  - replicated within the cluster

Kafka Architecture



Sarama: Go 4 Kafka

- developed at Shopify
- *Repos:*
  https://github.com/Shopify/sarama
  https://github.com/eapache
- *see talk:*
  "Complex Concurrency Patterns in Go"
  Evan Huus
  Golang UK 2015

NATS

- *High-performance system* used at Apcera, Baidu, Siemens, vmWare
  https://github.com/nats-io/gnatsd
- *Trade-offs:*
  - lightweight (server in Go, ruby/Go clients)
  - at-most-once delivery
  - no persistence or transactions
- *see talk:*
  “High Performance Systems in Go”
  Derek Collison (Apcera)
  GopherCon 2014

NSQ

bitly

- Trade-Offs:
  - for *real-time* distributed messaging
  - horizontal scaling *without brokers*
  - primarily an *in-memory queue*
    - messages not durable
    - *message order not guaranteed*
    - *at-least-once* delivery (idempotent receiver)

https://github.com/nsqio/nsq

- *see talk:*

  “Spray Some NSQ On It”
  Matt Reiferson
  GopherCon 2014

NSQ
BILLIONS AND BILLIONS
OF MESSAGES SERVED

Redis & Go

- in-memory store (like memcached)
  - with persistence to disk, more datatypes
  - interesting for realtime, but not reliability

Redis & Go

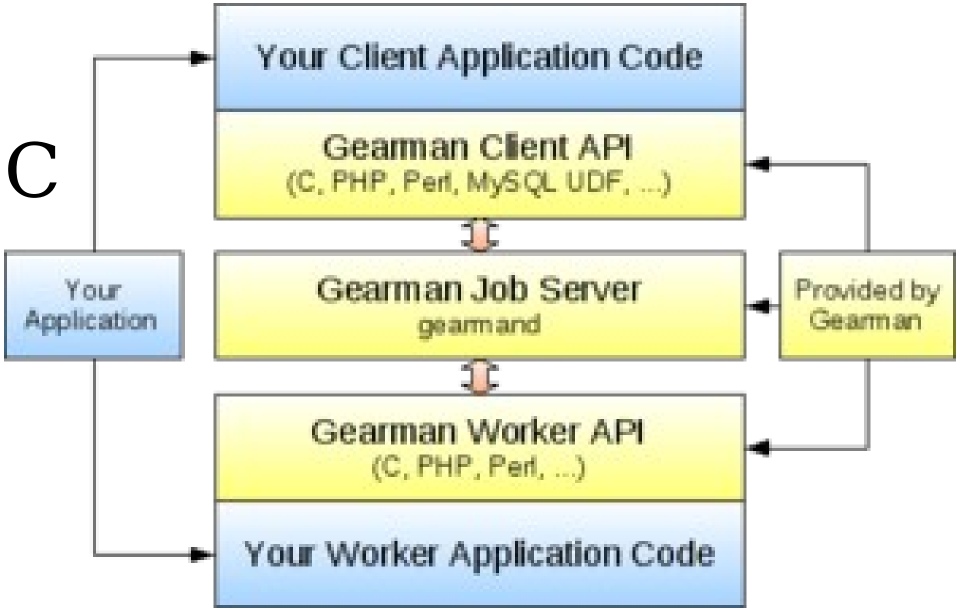
- in-memory store (like memcached)
  - with persistence to disk, more datatypes
  - interesting for realtime, but not reliability
- *Redis Message Queue in Go:*
  https://github.com/adjust/rmq

Redis & Go

- in-memory store (like memcached)
  - with persistence to disk, more datatypes
  - interesting for realtime, but not reliability
- *Redis Message Queue in Go:*
  https://github.com/adjust/rmq
- *Github's resque for background jobs:*
  http://github.com/resque/resque
  https://github.com/kavu/go-resque
  - resque compatible Goworker:
    - 10..1000 times faster than ruby-based workers
    - https://www.goworker.org/

Gearman Job Server

- *Load-balance jobs across workers*
  - conceptually related to map-reduce
  - binary protocol
  - original in Perl,now in C
  - Fitzpatrick: manaGer



- Go Gearman API (client/worker): https://github.com/mikespook/gearman-go

beanstalkd

- fast, simple, in-memory *work queue*
  - simple, text-based protocol a la memcached
  - 3 *job states* (ready, reserved, buried)
  - transactional
- *Go beanstalk*:
  https://github.com/kr/beanstalk
  https://github.com/manveru/gostalk
  https://github.com/iwanbk/gobeanstalk
  https://github.com/99designs/cmdstalk
  https://github.com/nutrun/lentil

Machinery - Go

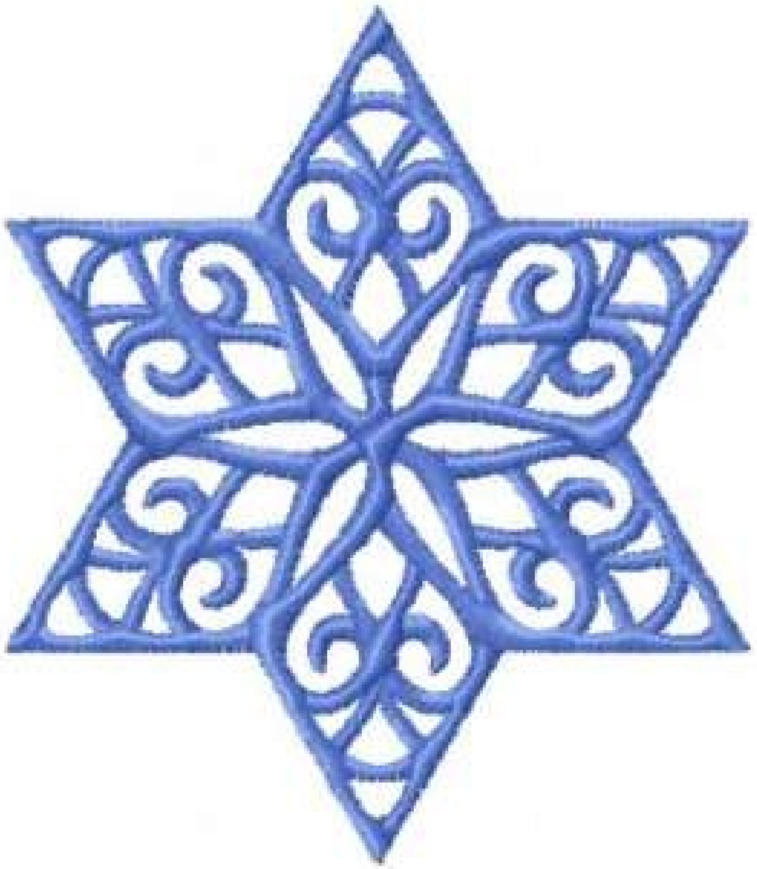
- asynchronous *task queue*/job queue
  - to execute distributed jobs in parallel
  - inspired by Python celery task queue
- with choice of *broker*:
  - RabbitMQ or
  - Redis

https://github.com/RichardKnop/machinery

STOMP

- ***S****imple* ***T****ext* ***O****rientated* ***M****essaging* ***P****rotocol*
  - Http-like commands (SEND, SUBSCRIBE, COMMIT, ACK, ABORT, ACK, NACK, ...)
  - https://stomp.github.io/
- supported by RabbitMQ, ActiveMQ plugins
- *Go Stomp:*
  https://github.com/go-stomp/stomp
  https://github.com/gmallard/stompngo

In search of Patterns



In search of Patterns

- Go-kit patterns:
  - rate-limiter
  - circuit breaker
  - load-balancer
  - tracing

https://github.com/go-kit/kit

- *Talk*:
  “Go kit: a toolkit for microservices”
  Peter Bourgon
  GopherCon 2015, Golang UK 2015

In search of Patterns

- Evan Huus *resiliency patterns*:
  - circuit-breaker
  - deadline/timeout
  - batching
  - retrier

https://github.com/eapache/go-resiliency

Talk:

"Complex Concurrency Patterns in Go"
Evan Huus
Golang UK, August 2015

Wrap-up

- *breadth of MOM implementations*
- *what about depth?*

Wrap-up

- *breadth of MOM implementations*
- *what about depth?*
- *example: RabbitMQ in Erlang*
  - *complex spec in 5000 lines of code*
  - *solves hard concurrency problems*

Wrap-up

- *breadth of MOM implementations*
- *what about depth?*
- *example: RabbitMQ in Erlang*
  - *complex spec in 5000 lines of code*
  - *solves hard concurrency problems*
- *what are the strengths of Go here?*