ApacheCon North America
Portland, Oregon 26th - 28th February 2013

# Instant Integration into the AMQP Cloud with Apache Qpid Messenger

Rafael Schloming
Principle Software Engineer @ Red Hat
rhs@apache.org

## Overview

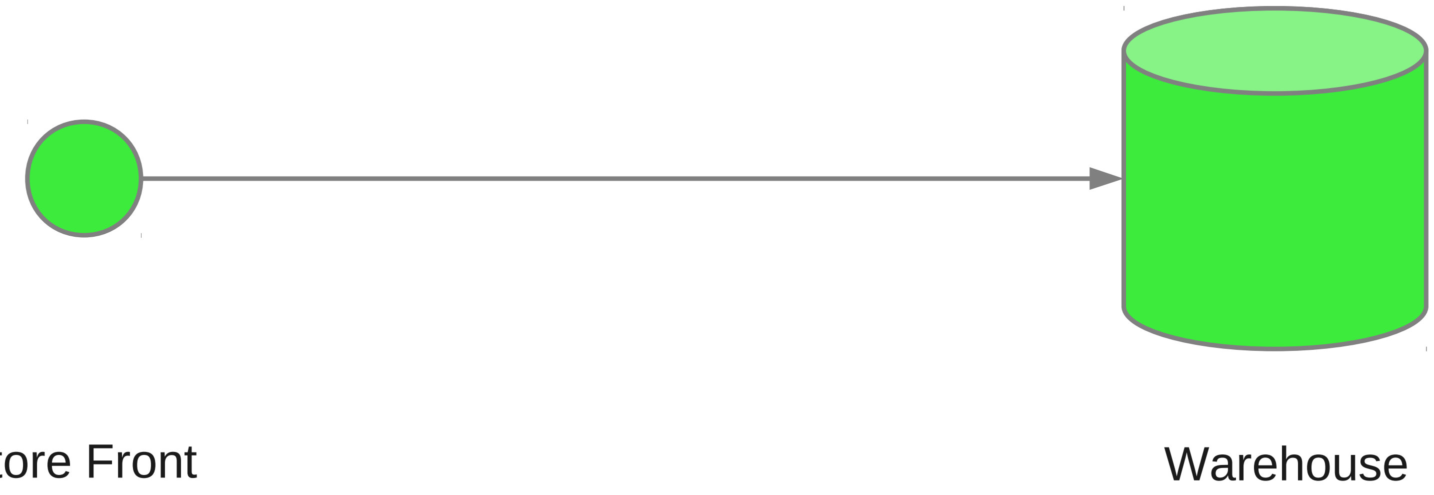
- Introduction
- Messaging
- AMQP
- Proton
- Demo
- Summary

## Introduction

- AMQP 1.0
  - OASIS Standard
  - Messaging Protocol
- Proton: A toolkit for speaking AMQP
  - The AMQP Protocol Engine API
  - The AMQP Messenger API
- Part of the Apache Qpid project
  - Qpid is the home for AMQP at Apache

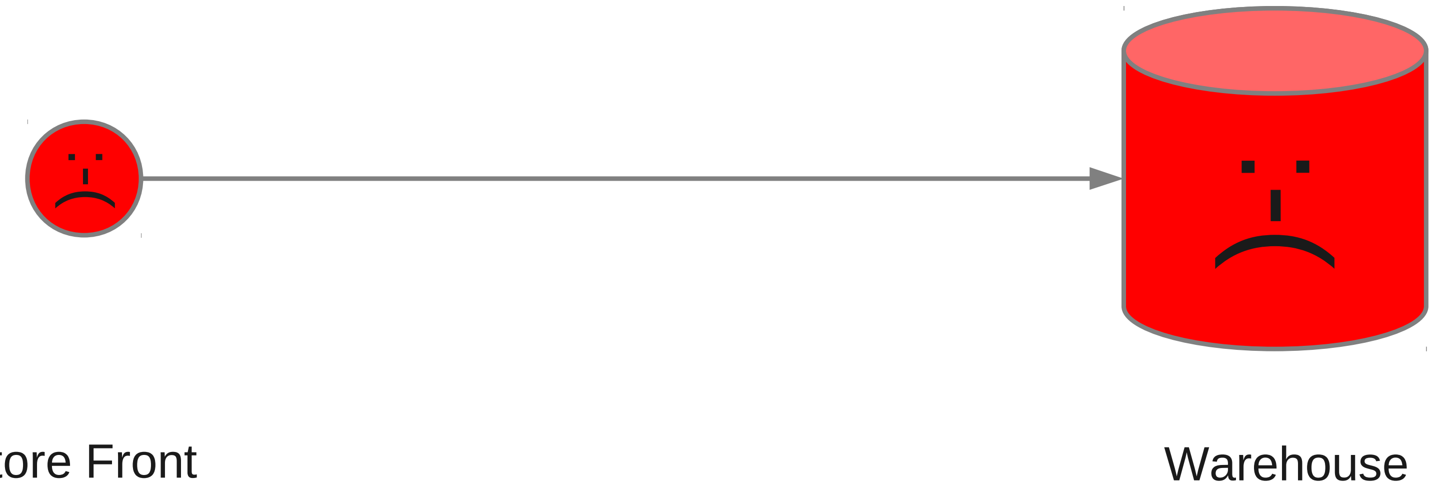
## Messaging

- Tightly Coupled



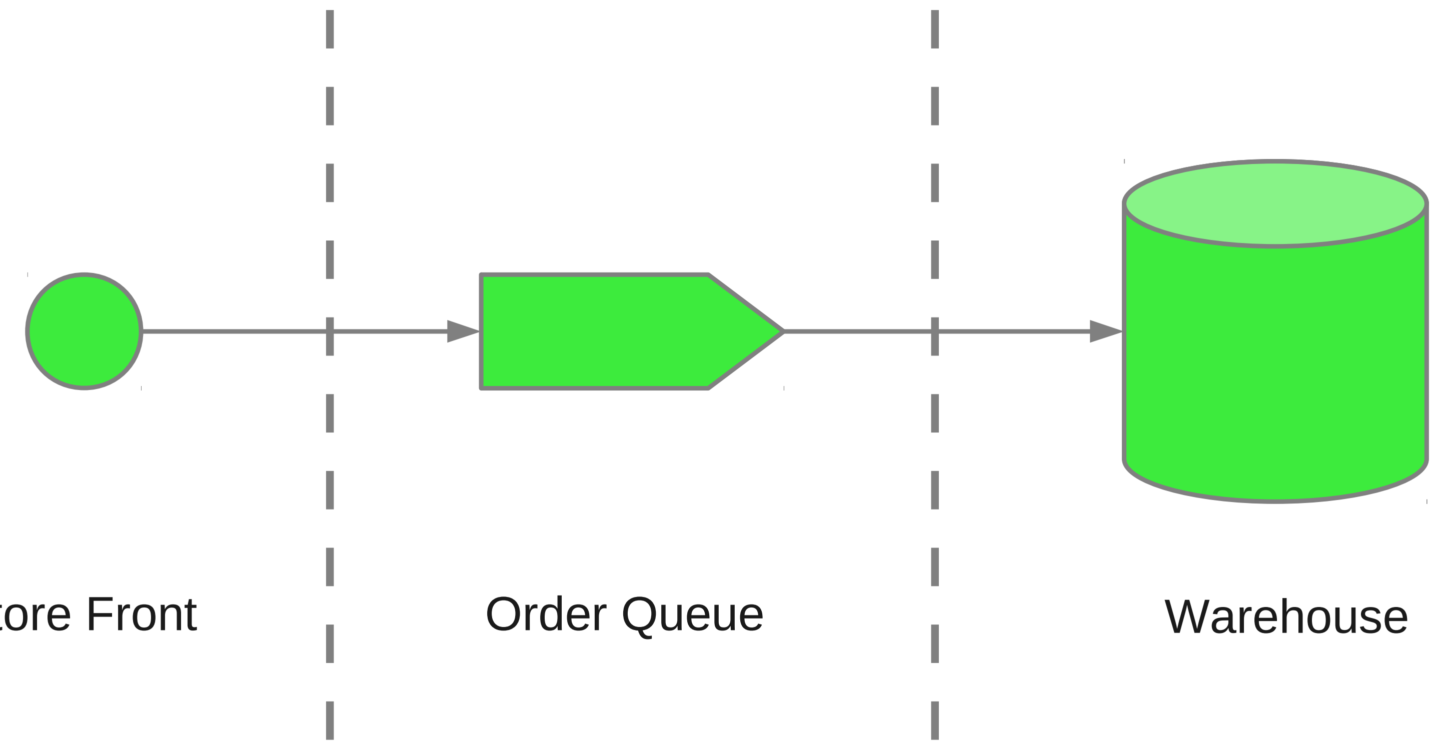
## Messaging

- Tightly Coupled



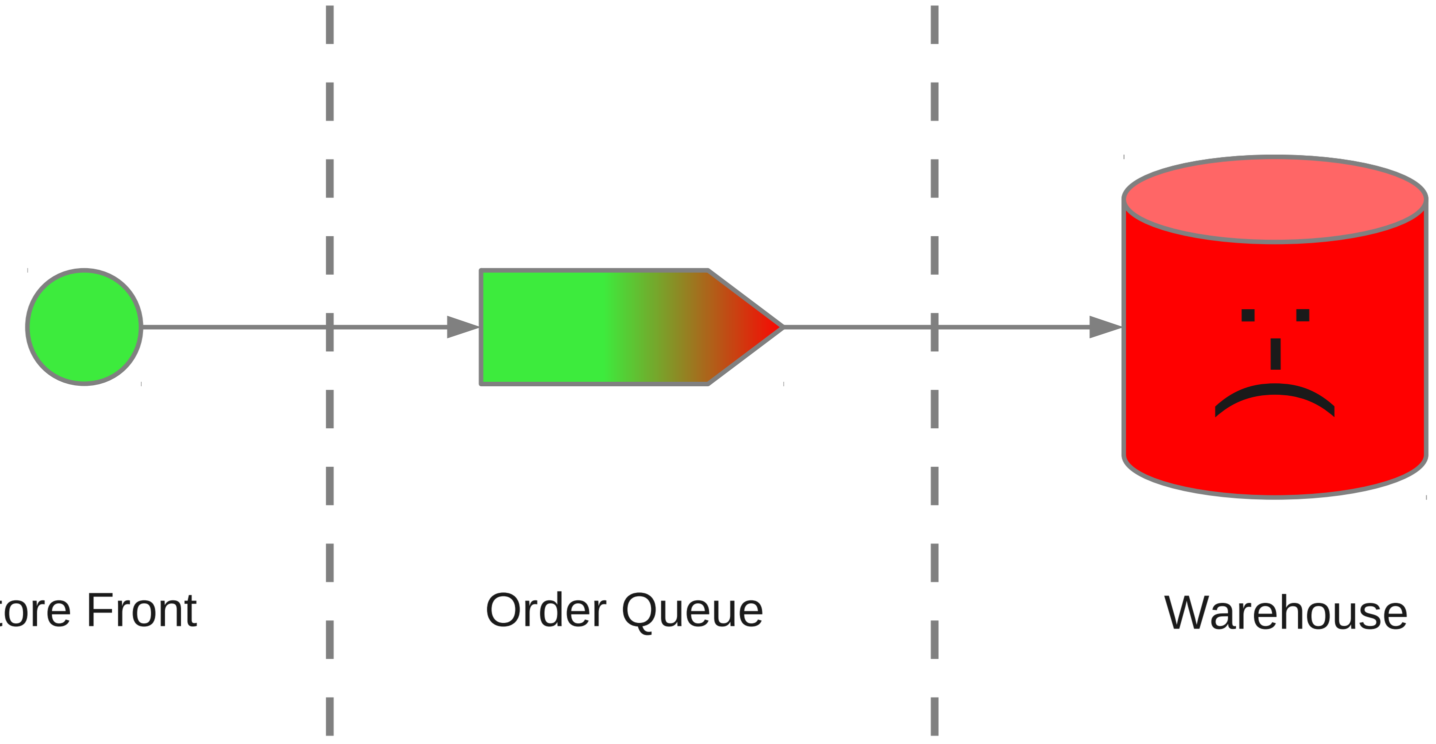
## Messaging

- Loosely Coupled



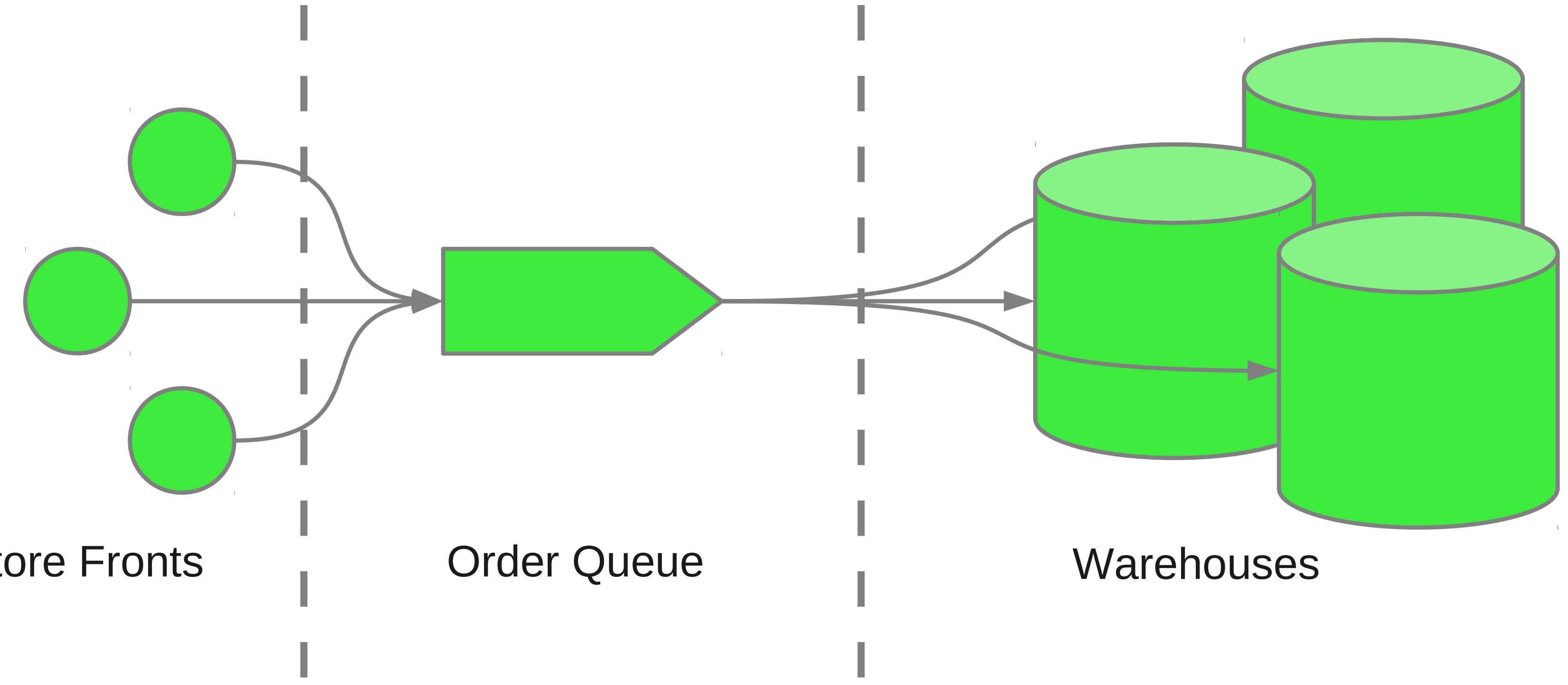
## Messaging

- Loosely Coupled



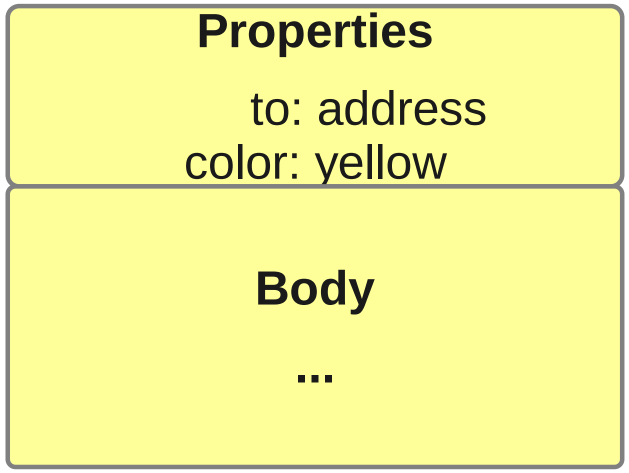
## Messaging

- Loosely Coupled



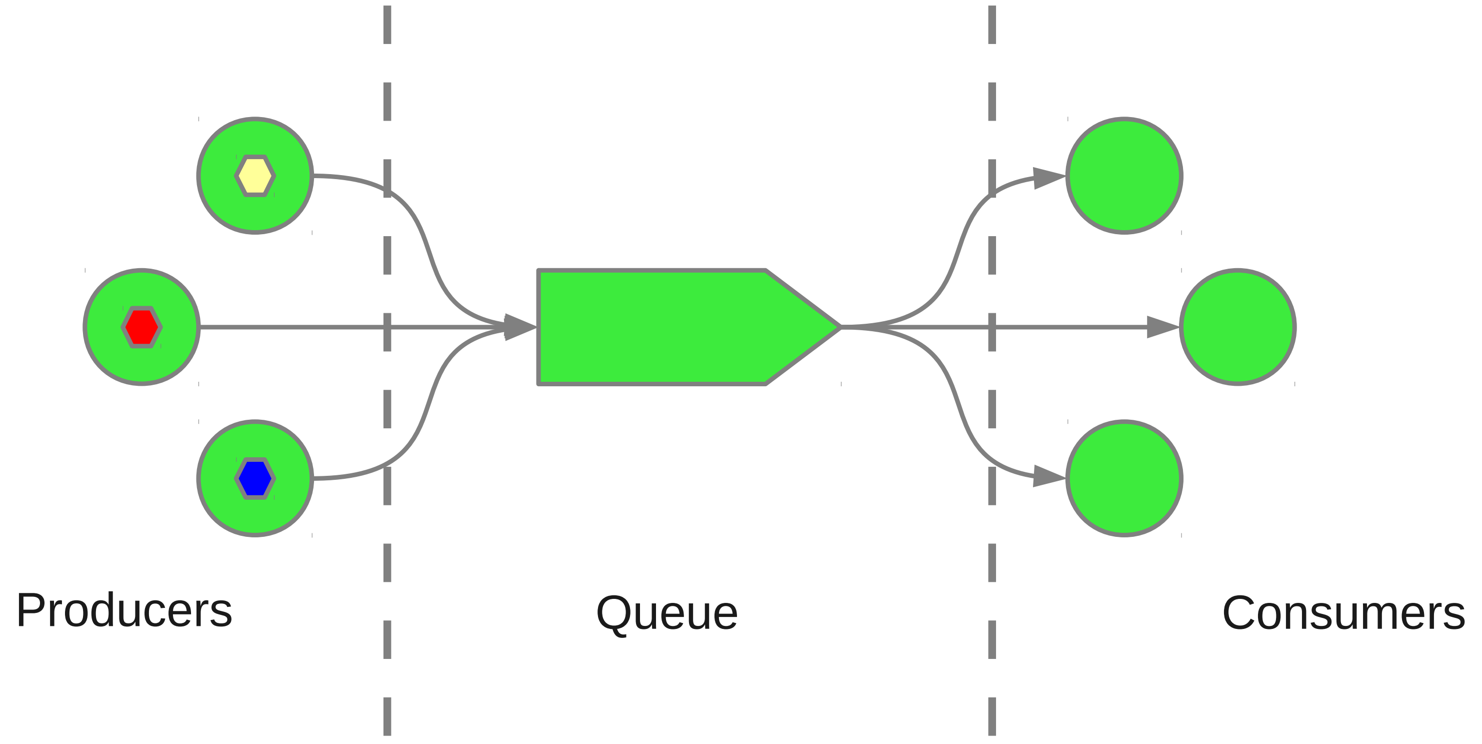
Messaging

- Message
  - addresses are rendezvous points
    - flexible n – n communication topologies
  - properties available for semantic routing



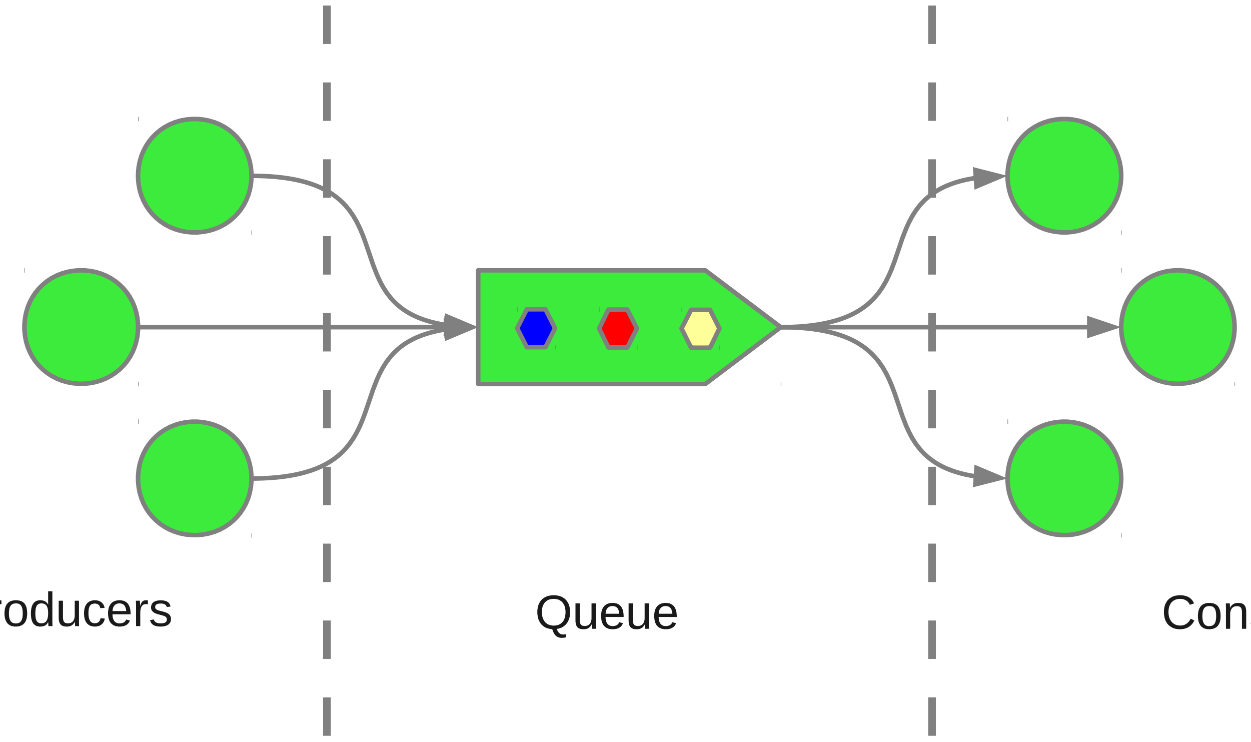
## Messaging

- Queues



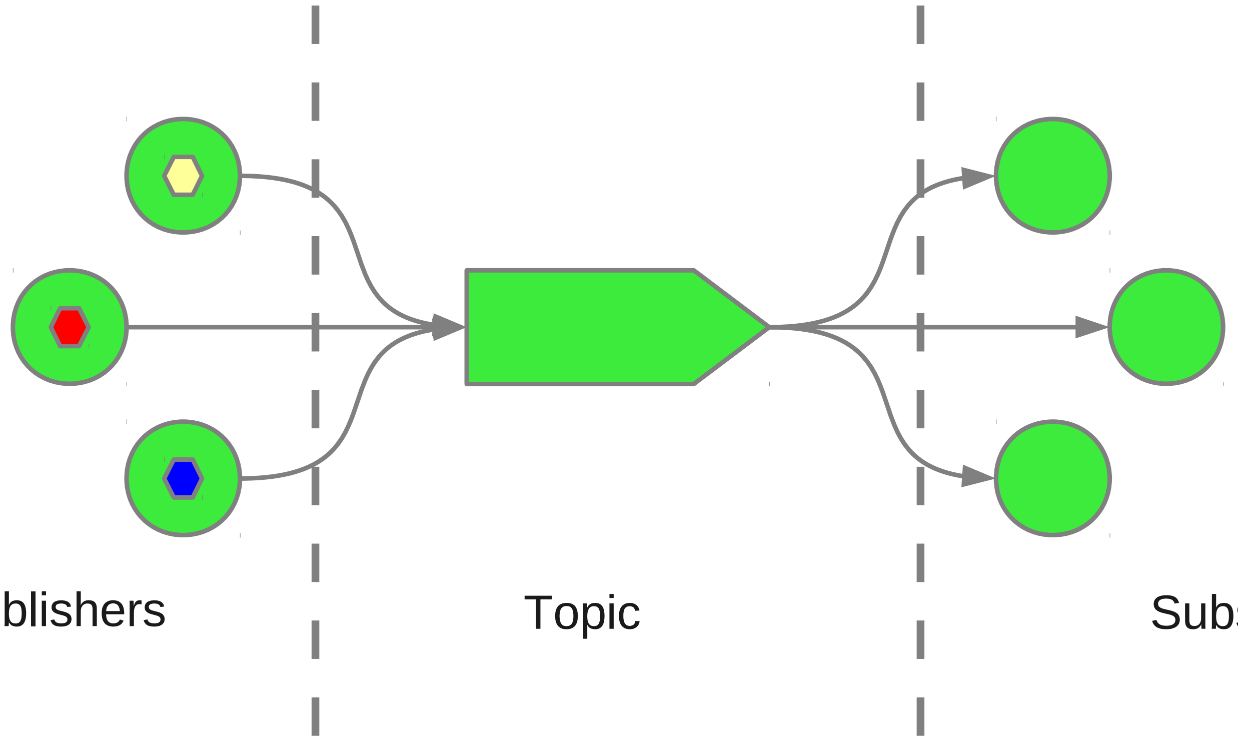
## Messaging

- Queues



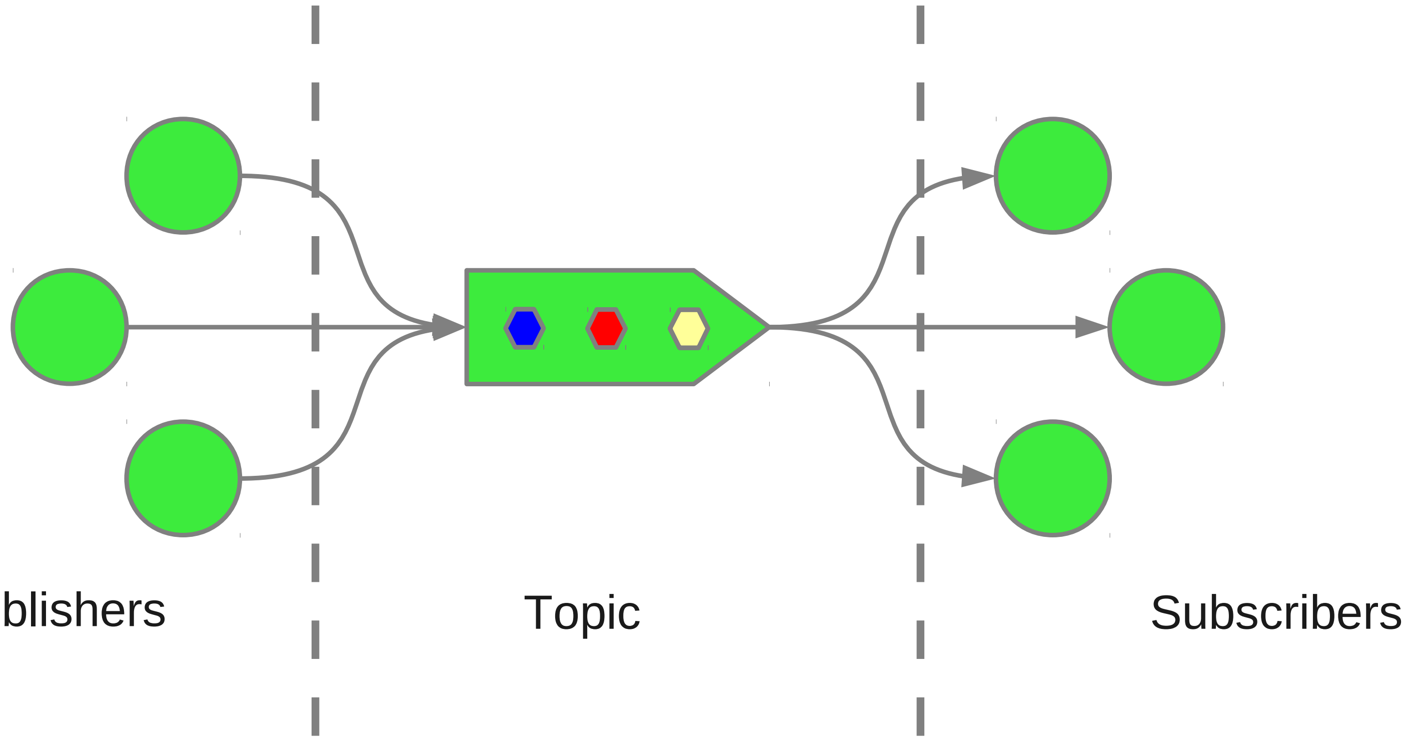
# Messaging

- Topics



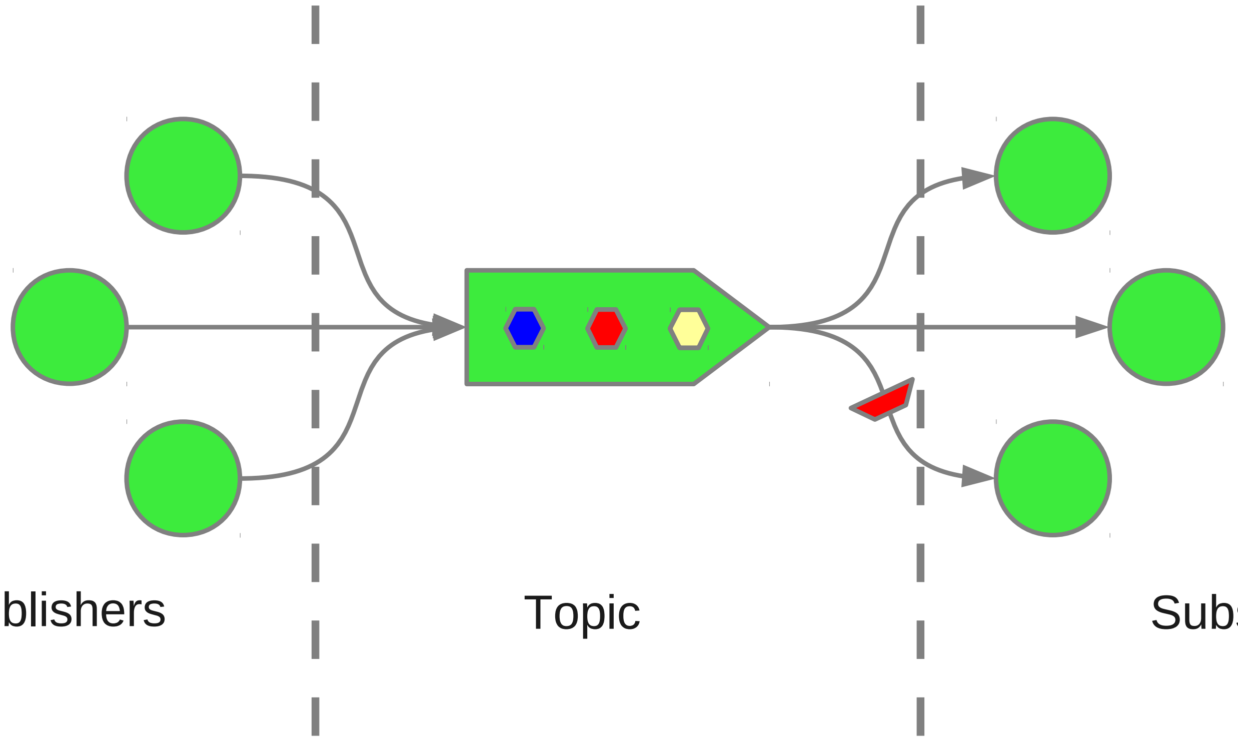
# Messaging

- Topics



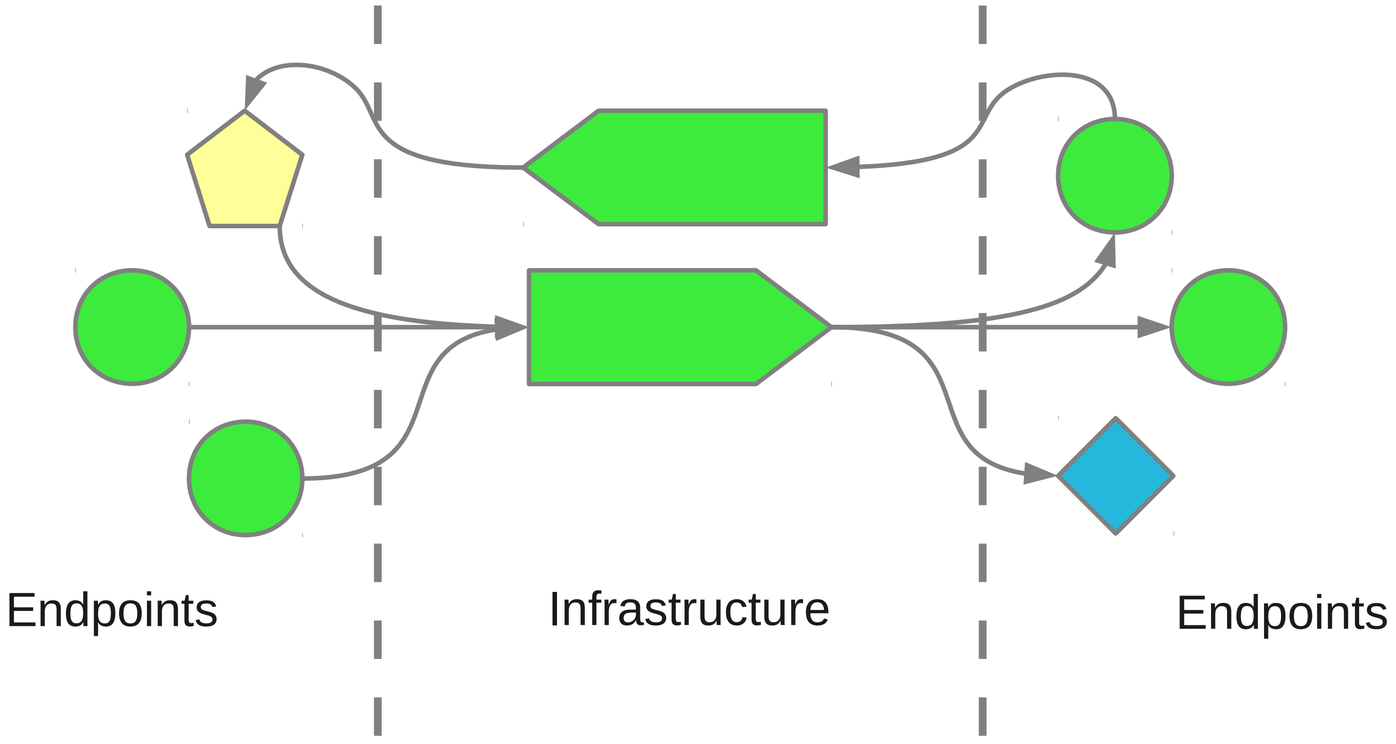
# Messaging

- Topics

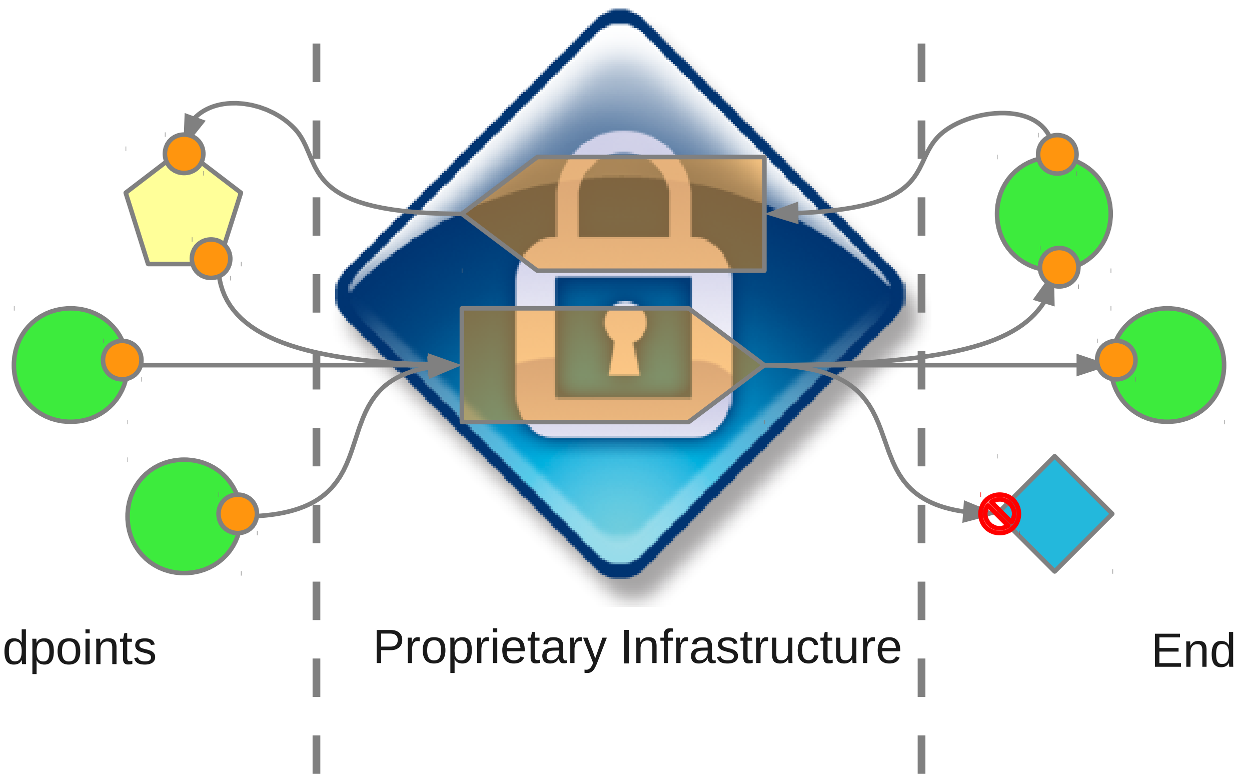


# Messaging

- Heterogeneous



# Proprietary Messaging



## Proprietary Messaging

- difficult to port
  - requires rewriting apps to different API
- difficult to integrate
  - requires app level bridging and translation
- platforms limited to vendor provided choices

## AMQP 1.0

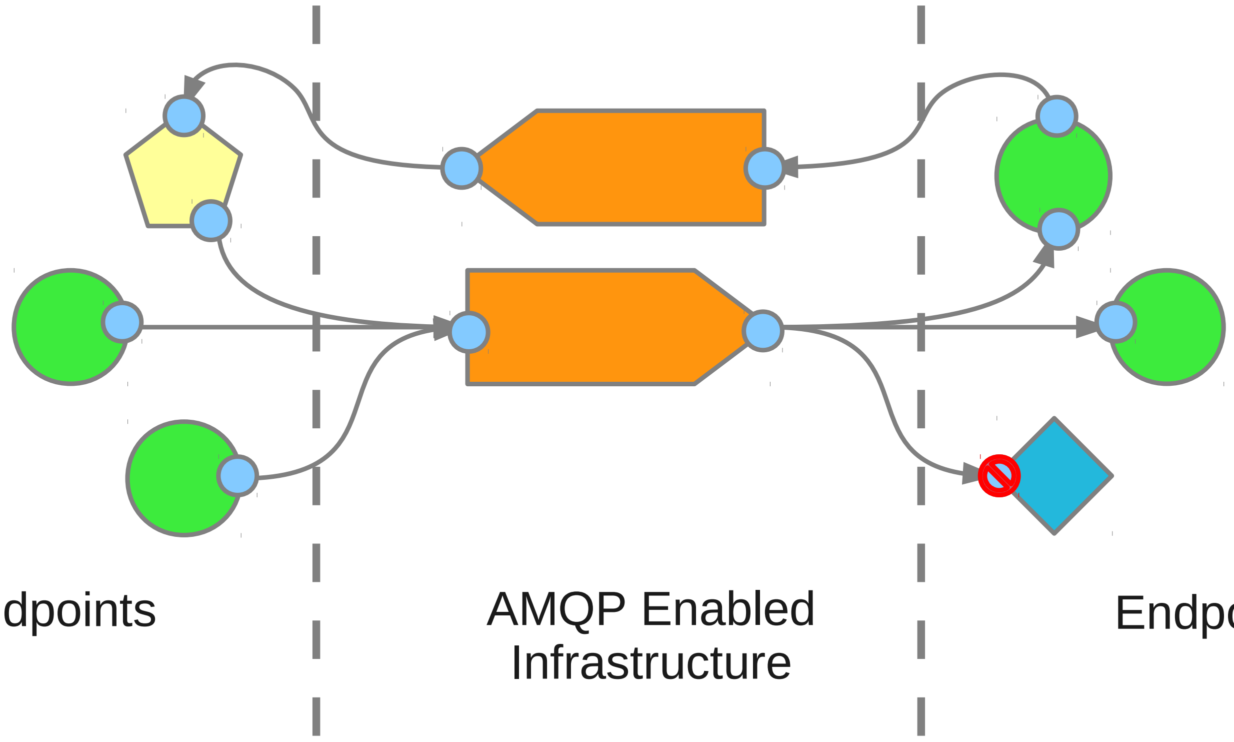
- Ratified as an OASIS standard in Oct 2012
  - Developed over several years by an industry working group including:
    - Technology vendors
      - Axway Software, Huawei Technologies, IIT Software, INETCO Systems, Kaazing, Microsoft, Mitre Corporation, Primeton Technologies, Progress Software, Red Hat, SITA, Software AG, Solace Systems, VMWare, WSO2, Zenika
    - User firms
      - Back of America, Credit Suisse, Deutsche Boerse, Goldman Sachs, JPMorgan Chase

## AMQP 1.0

- Concisely expresses core messaging semantics
  - flow control
  - settlement
  - transactions
  - data binding
- Suitable as a wire protocol for a wide range of message oriented applications

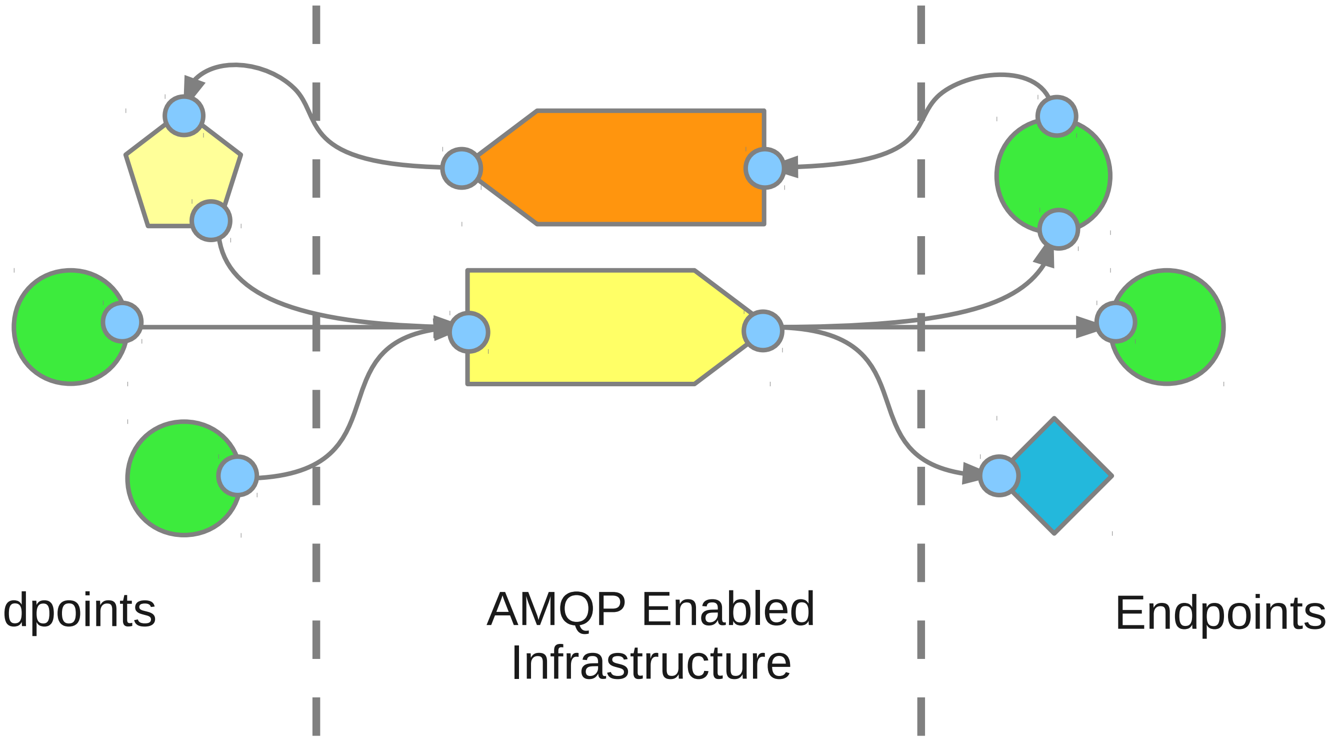
## AMQP 1.0

- AMQP Enabled Infrastucture



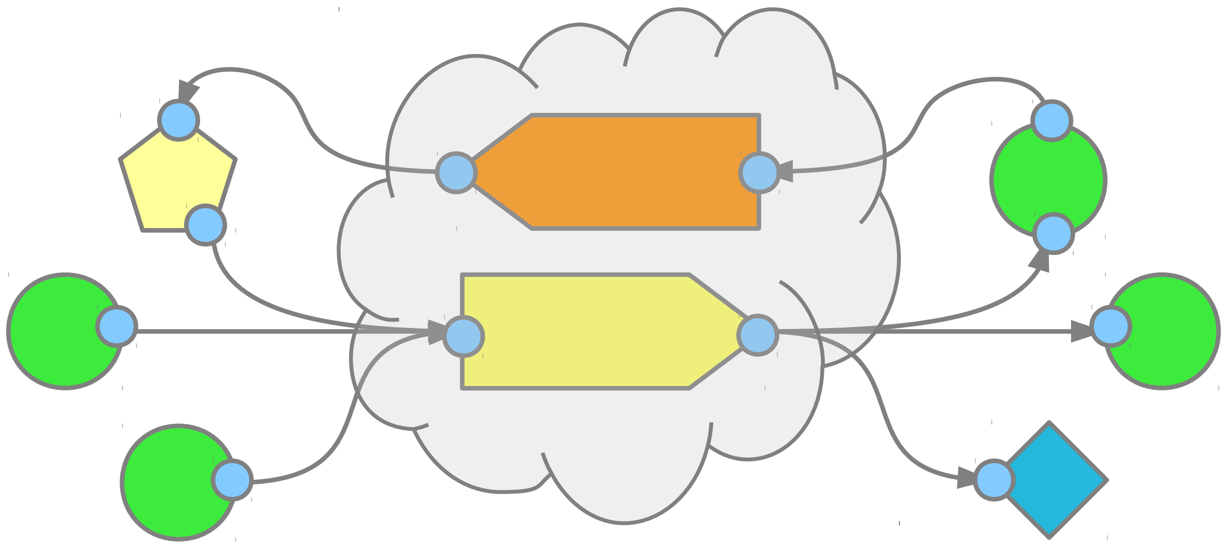
# AMQP 1.0

- Heterogeneous Infrastructure



# AMQP 1.0

- Standard + Secure = Open Deployments



## AMQP 1.0 Implementations

- Apache Qpid
  - C++ and Java brokers
- Apache ActiveMQ
  - multiprotocol message broker
- Azure Service Bus
  - PaaS
- Swift MQ
  - JMS broker + client
- Rabbit MQ

## Apache Qpid Proton

- Proton is a ***protocol*** implementation
  - Previous attempts to standardize messaging have been client/server based, i.e. RPC
  - AMQP 1.0 is a protocol specification
    - Network oriented: Symmetric, Decentralized
    - Provides intermediated messaging semantics, but does not restrict to hub and spoke topology
    - Not just a standard way to talk to a traditional broker
  - AMQP 1.0 makes a protocol implementation possible

## Apache Qpid Proton

- Traditional MOM transformed
  - Traditional MOMs conflate both
    - store and forward *infrastructure*
    - specialized *application* behaviors
      - special queues: last value, ring queues
      - message transformation
  - Driven by Scalability and Standardization
- With AMQP 1.0, these features can be
  - distributed, scalable, heterogeneous

## Apache Qpid Proton

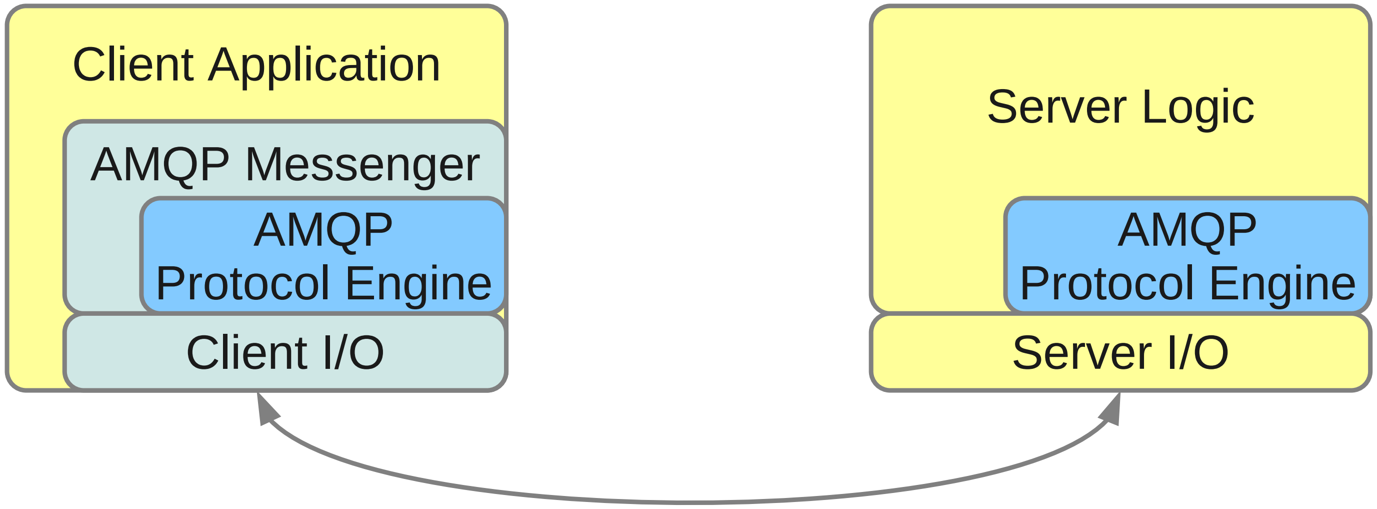
- Many things benefit from speaking AMQP
  - A concise expression of a very general set of messaging semantics
    - Flow control
    - Settlement
    - Transactions
    - Data binding
  - Not everyone wants to implement all this down to the wire

## Apache Qpid Proton

- Goals
  - Make it easy to speak AMQP
    - minimal dependencies
    - minimal threading assumptions
    - multilingual
      - C, Java, Javascript
      - C Bindings in python, ruby, php, perl, ...
    - multi-platform
      - Linux/unix, windows, android, iOS

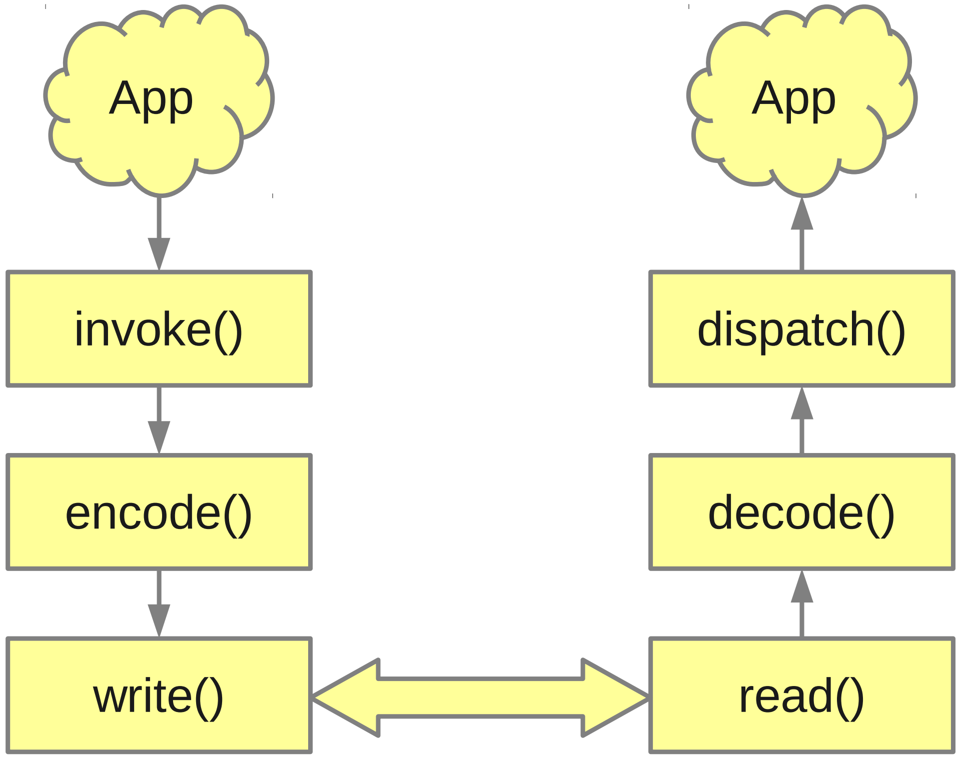
# Apache Qpid Proton

- Protocol Engine



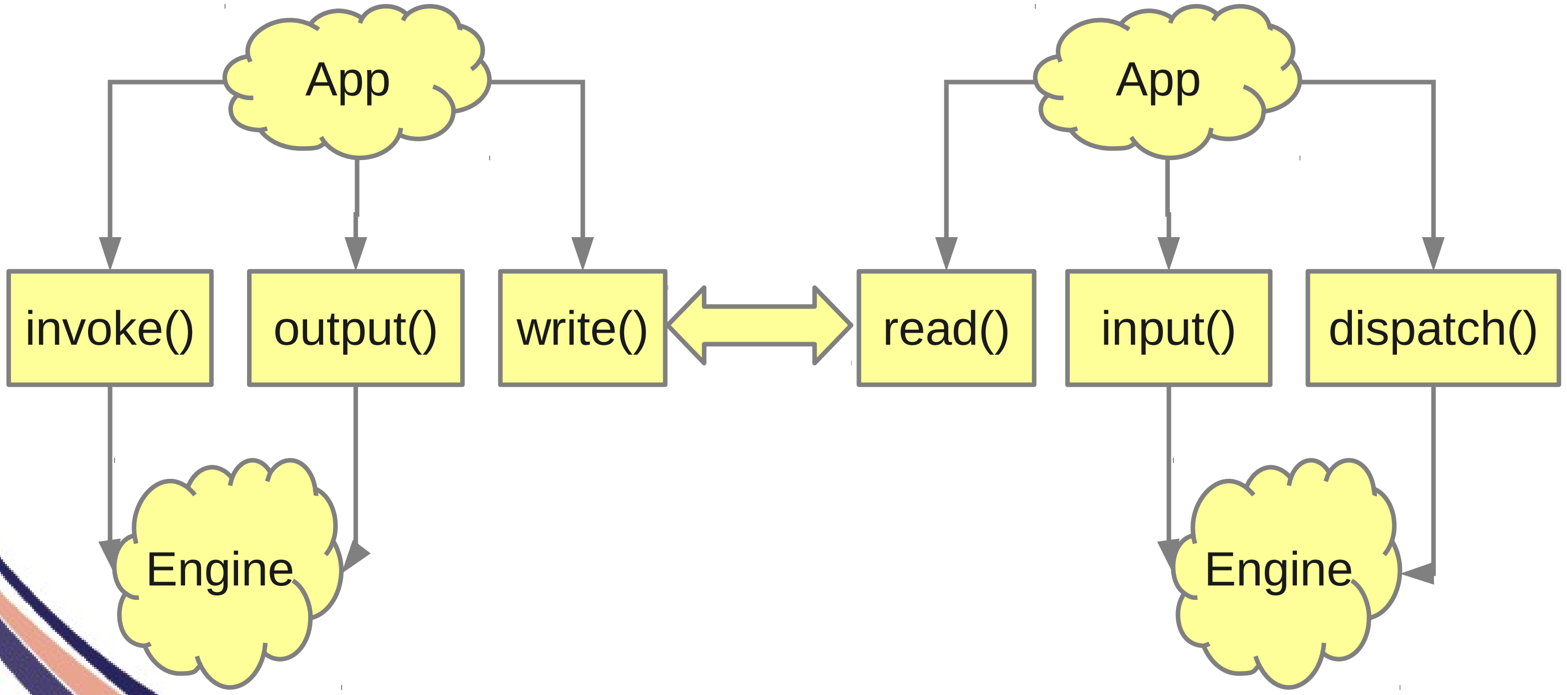
## Protocol Engine

- NOT a traditional “RPC-like” pattern:
  - protocol implementation does I/O
    - Coupled to OS interfaces, I/O strategy, threading model



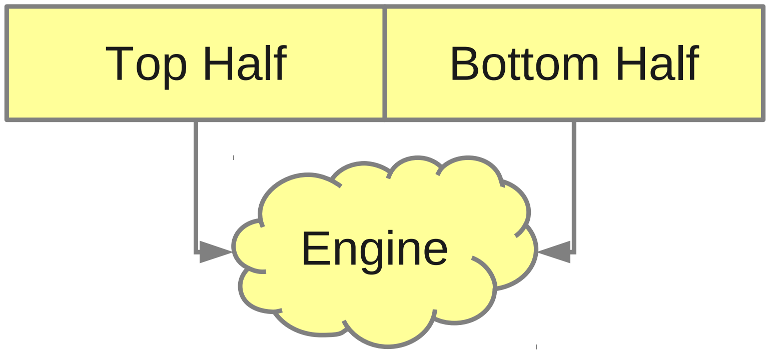
## Protocol Engine

- Engine pattern:
  - application does I/O
  - engine encapsulates protocol state
    - pure state machine, no dependencies, no callbacks



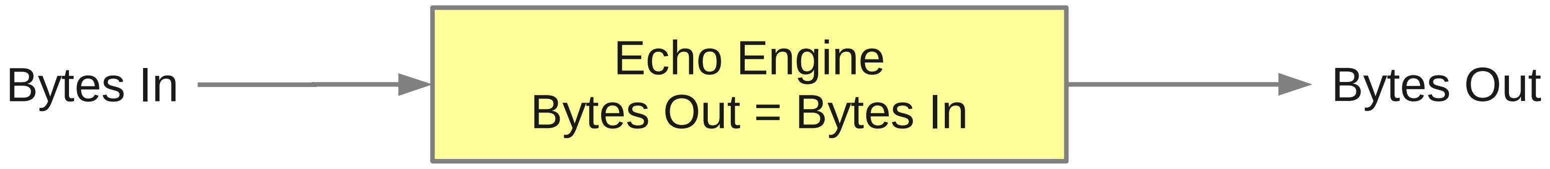
## Protocol Engine

- Engine interface: “top” and “bottom” half
  - Top half
    - traditional protocol interface in non blocking form
      - establish senders and receivers, send/recv message data
  - Bottom half
    - transport interface, inverted
      - normal transport pushes bytes to a socket
      - inverted transport pulls bytes from the engine



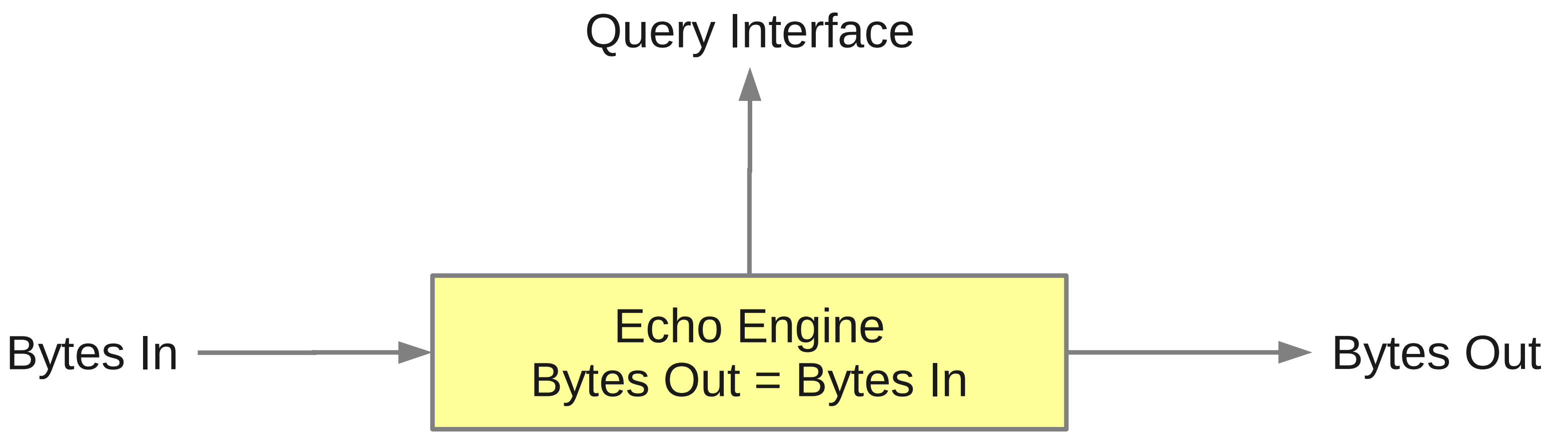
## Protocol Engine

- Echo Server



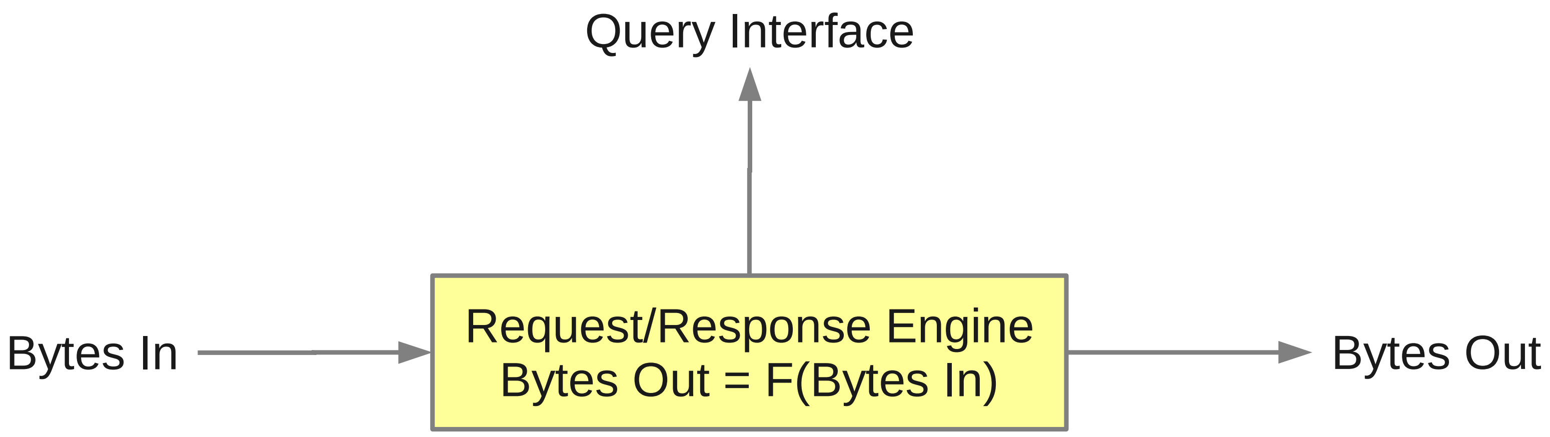
# Protocol Engine

- Echo Server



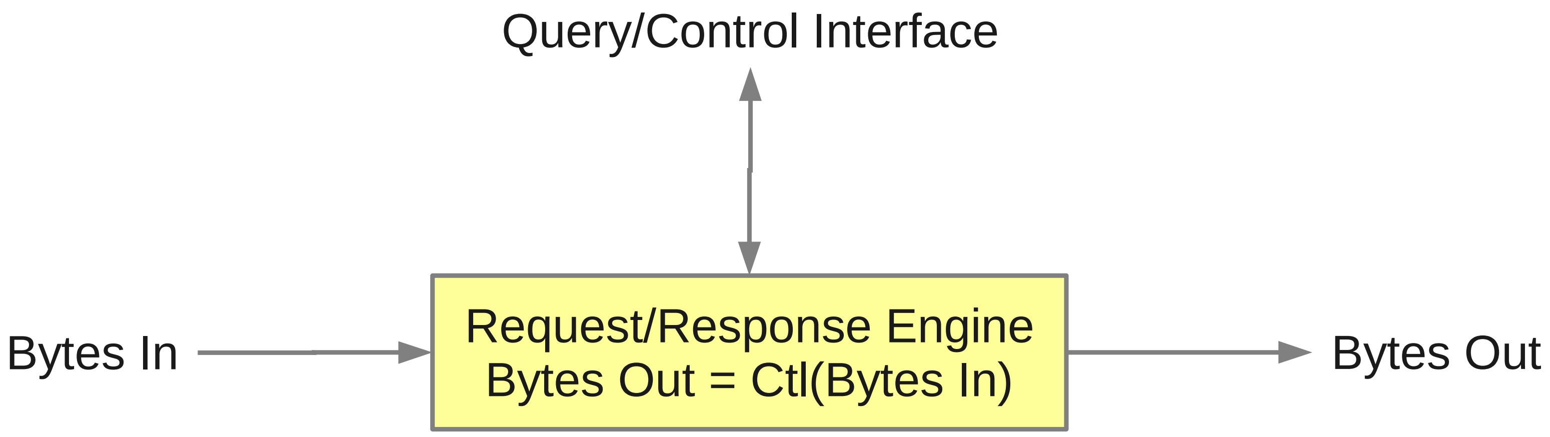
# Protocol Engine

- Request/Response Server



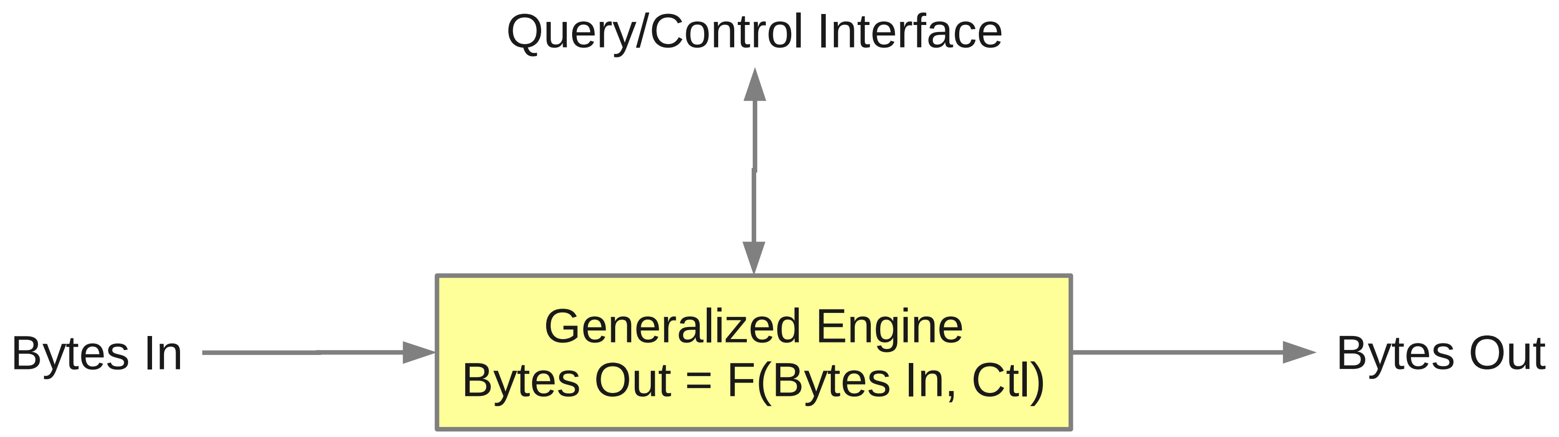
## Protocol Engine

- Request/Response Server



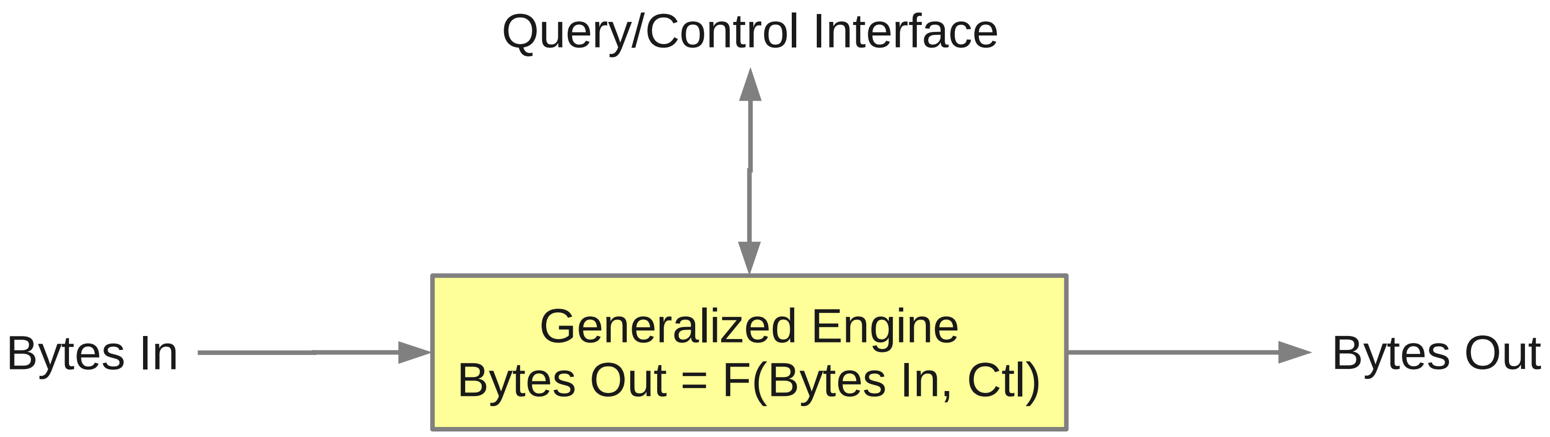
# Protocol Engine

- Generalized Server



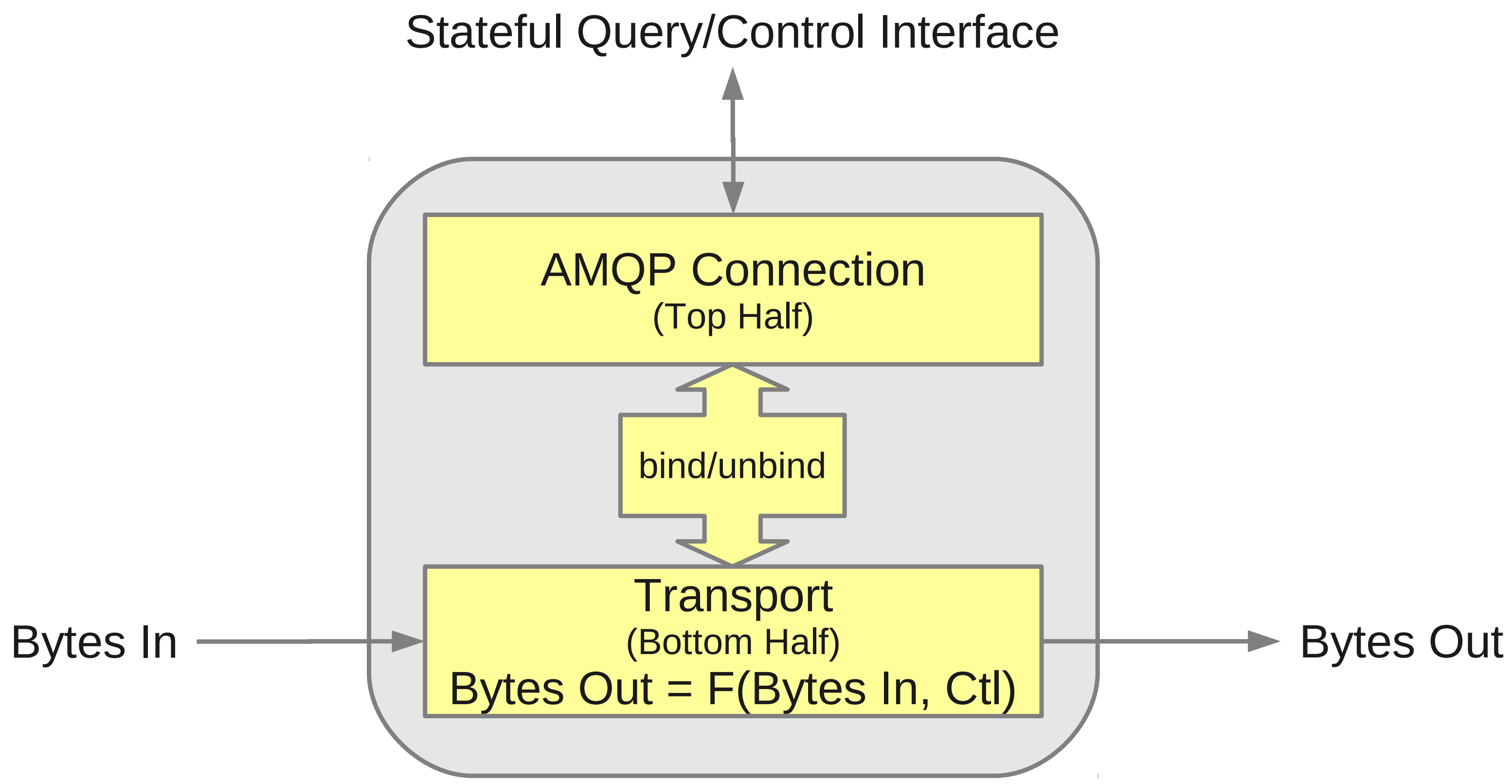
## Protocol Engine

- Generalized Endpoint



# Protocol Engine

- Generalized AMQP Engine

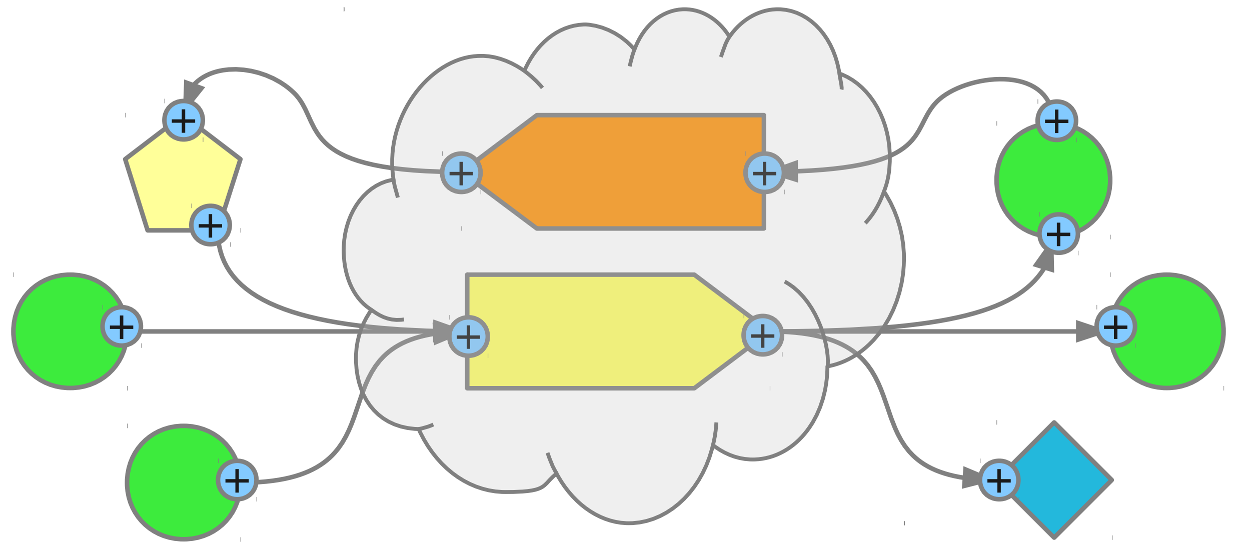


## Protocol Engine

- Benefit: flexibility
  - Single protocol implementation can be shared
    - Used in a simple client
    - Easy to embed into existing servers
  - Thread agnostic
    - works with single threaded and multithreaded servers of any architecture
  - Easy to swig
    - pure data structure
    - no callbacks
    - simple interface

## Protocol Engine

- Multiplatform + Multilingual = Protons Everywhere



Endpoints AMQP Cloud Endpoints

# Messenger

Sending

```
messenger = Messenger()

messenger.start()

msg = Message()
msg.address = "0.0.0.0"
msg.body = u"Hello World!"

messenger.put(msg)
messenger.send()


messenger.stop()
```

Receiving

```
messenger = Messenger()
messenger.subscribe("~0.0.0.0")
messenger.start()

msg = Message()

while True:
  messenger.recv()
  while messenger.incoming:
    messenger.get(msg)
    print msg.body

messenger.stop()
```

## Messenger

- Message oriented, not connection oriented
  - (re) creates and pools the minimal number of connections behind the scenes
    - simplifies failover
  - topology is invisible to application
- Simple, but not a toy
  - batch oriented interface
    - high performance

# Messenger

Sending Reliably

```
messenger = Messenger()

messenger.outgoing_window = 100
messenger.start()

msg = Message()
msg.address = "0.0.0.0"
msg.body = u"Hello World!"

tracker = messenger.put(msg)
messenger.send()
print messenger.status(tracker)

messenger.stop()
```

Receiving Reliably

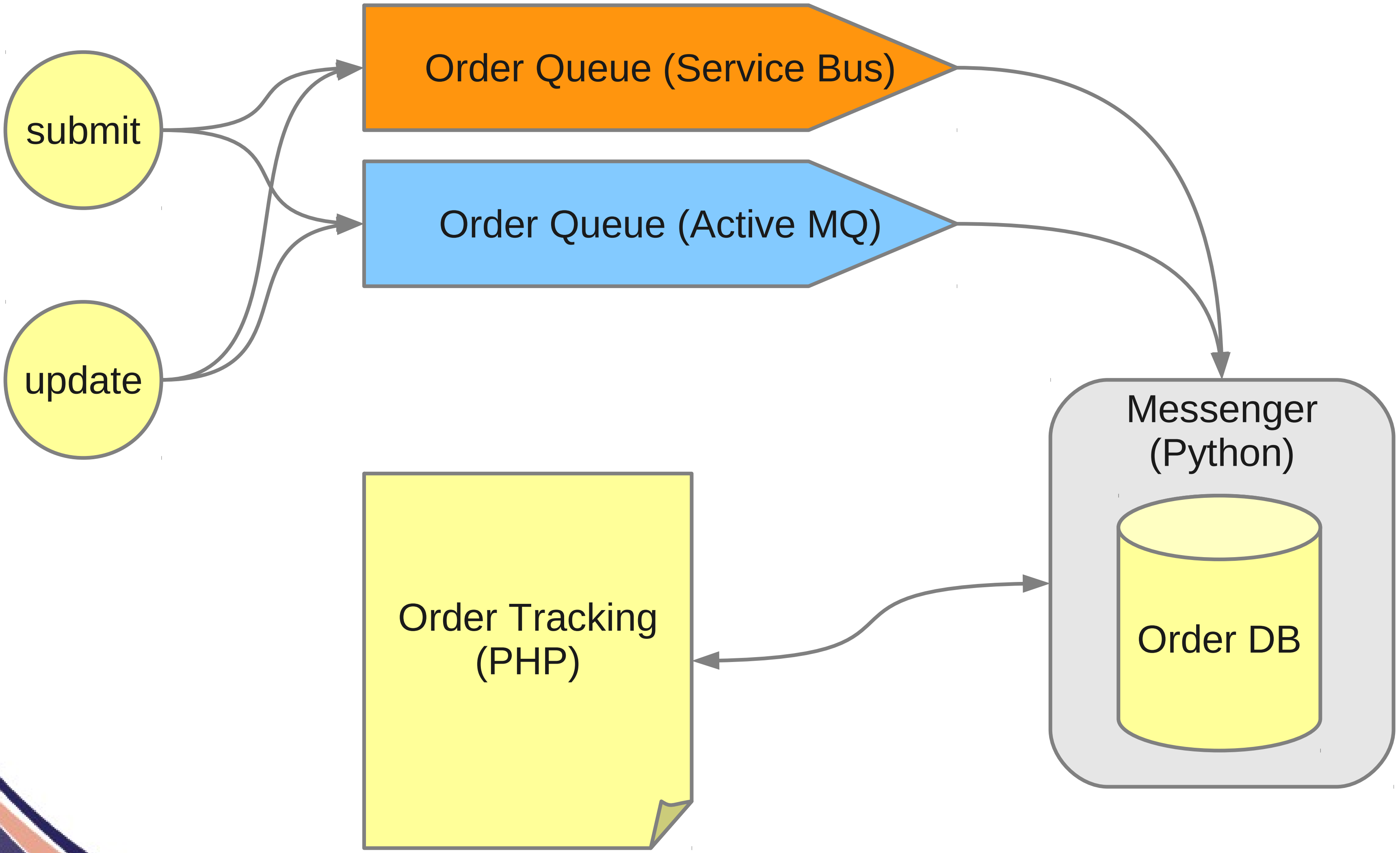
```
messenger = Messenger()
messenger.subscribe("~0.0.0.0")
messenger.start()

msg = Message()

while True:
  messenger.recv()
  while messenger.incoming:
    messenger.get(msg)
    print msg.body
    messenger.accept()

messenger.stop()
```

## Message

- mutable and reusable holder of content
  - works with batch send
    - more performance
  - doesn't conflate delivery with message
    - flexible: modify a received message and resend it
- data binding from AMQP to native types
- usable with Messenger or Engine

Portland, Oregon 26th - 28th February 2013

## Demo



## Summary

- AMQP 1.0 is a new kind of messaging
  - brings messaging to the masses
- Proton
  - The AMQP Protocol Engine
    - advanced architecture
    - based on years of enterprise experience
  - The AMQP Messenger API
    - simple but powerful programming API
- This is the basis of next gen applications

## More Information

- http://qpid.apache.org/proton
- proton@qpid.apache.org
- http://www.amqp.org