ApacheCon Europe

Sinsheim, Germany 5th-8th November 2012

Introduction to Apache Qpid Proton

Rajith Attapatu
Senior Software Engineer @ Red Hat
rajith@apache.org

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

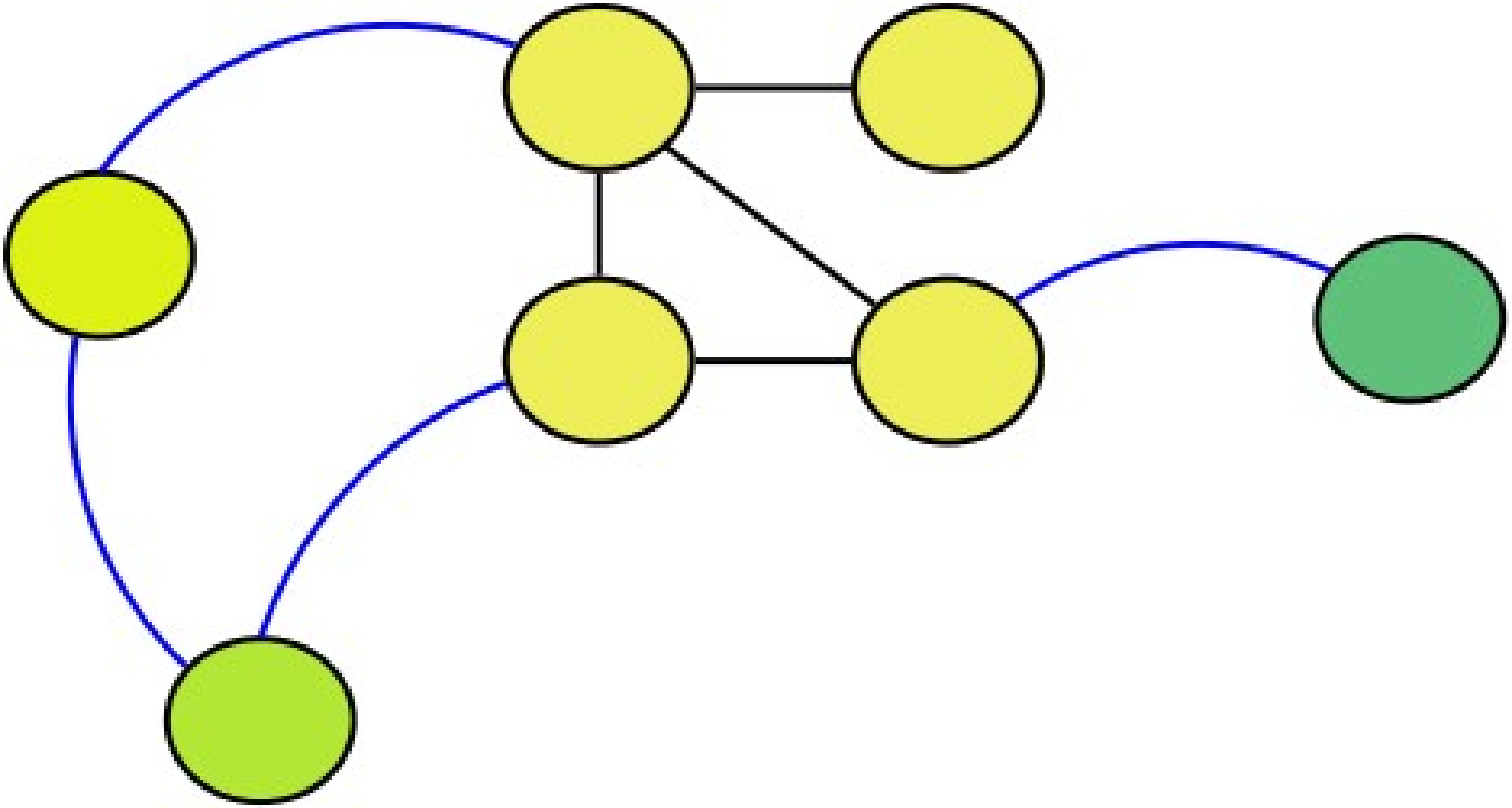
Overview

- Introduction
- Background
- Protocol Engine
- Messenger
- Summary

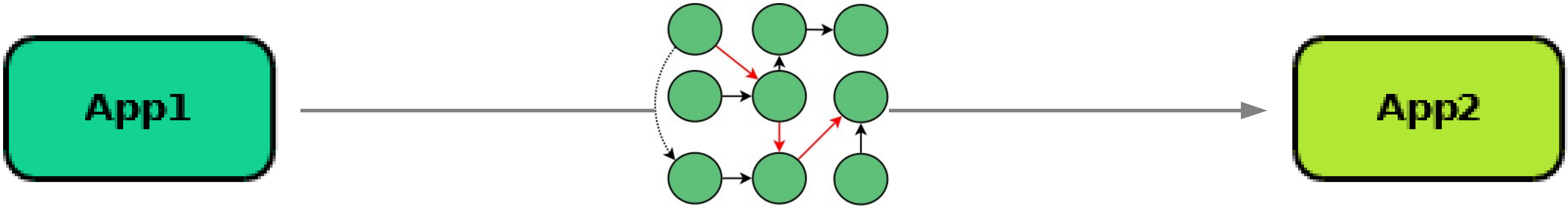
Introduction

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

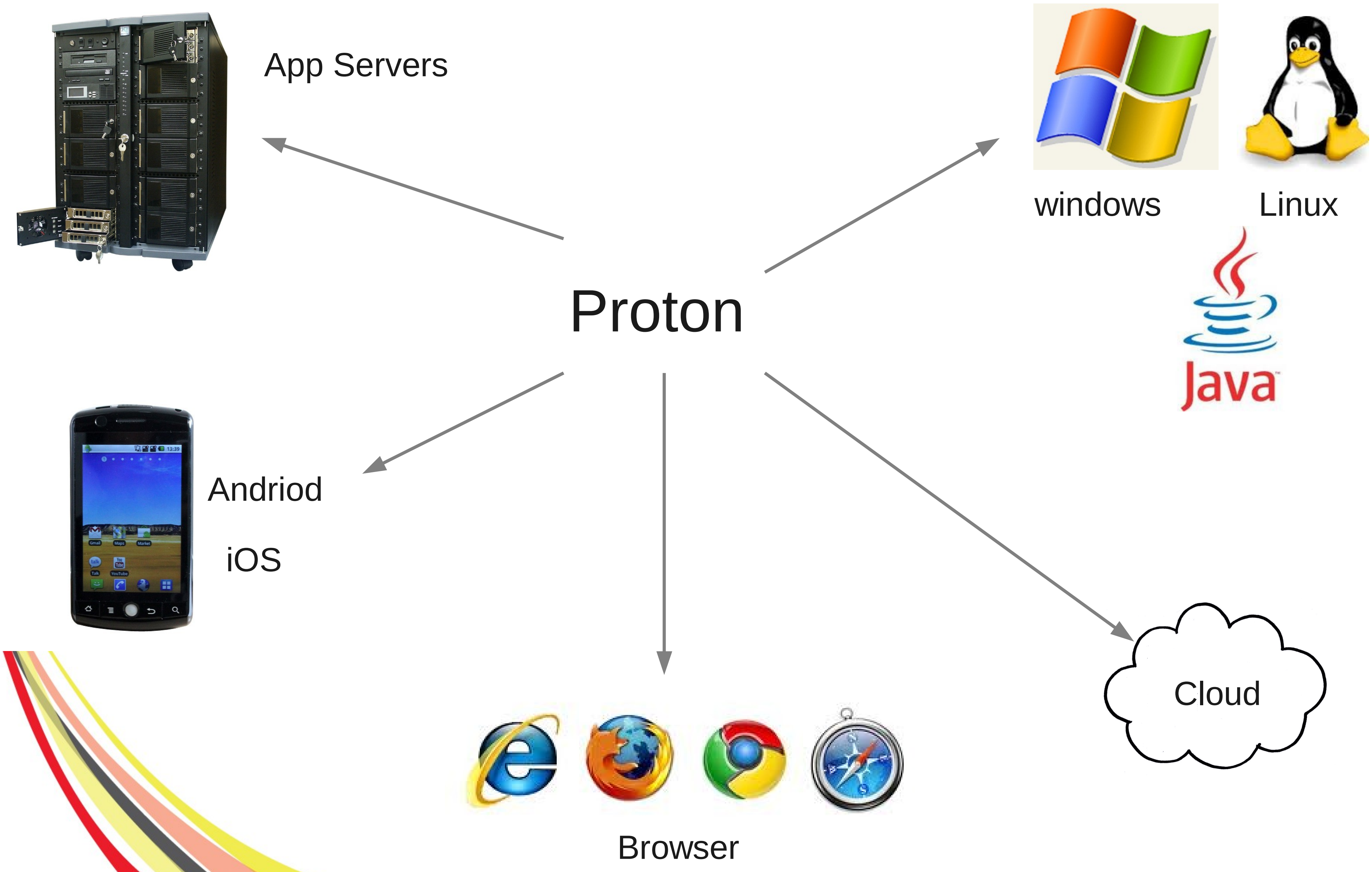
Proton is network based and decentralized



Proton Can Scale Transparently.



Proton is Highly Embeddable



App Servers

windows

Linux

Java

Proton

Andriod

iOS

Cloud

Browser

Designed For Maximum Embeddability

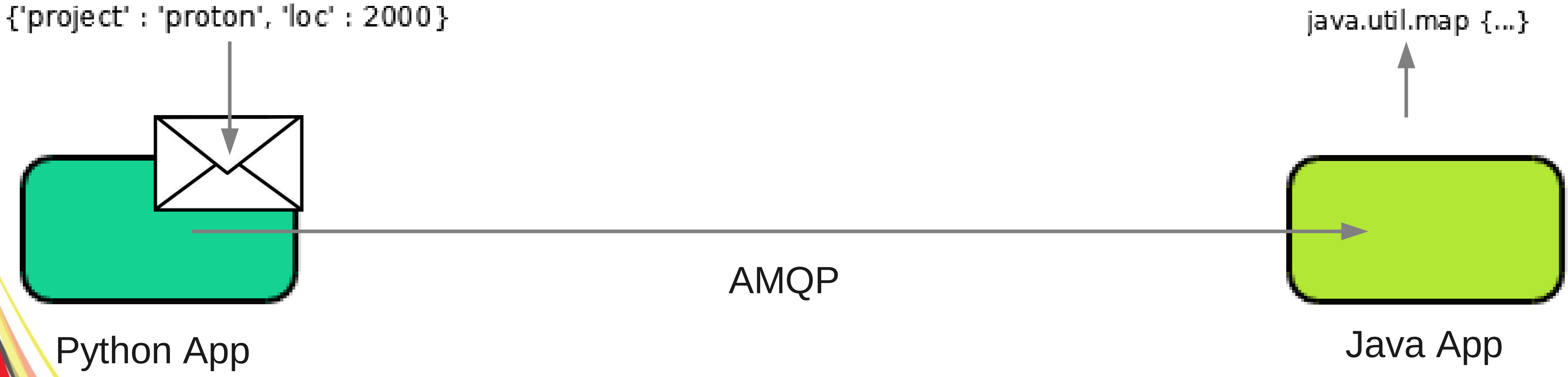
- Minimal assumptions about the host environment.
- Minimal assumptions about the application threading model.
- Minimal dependencies.

Proton Design Goals.

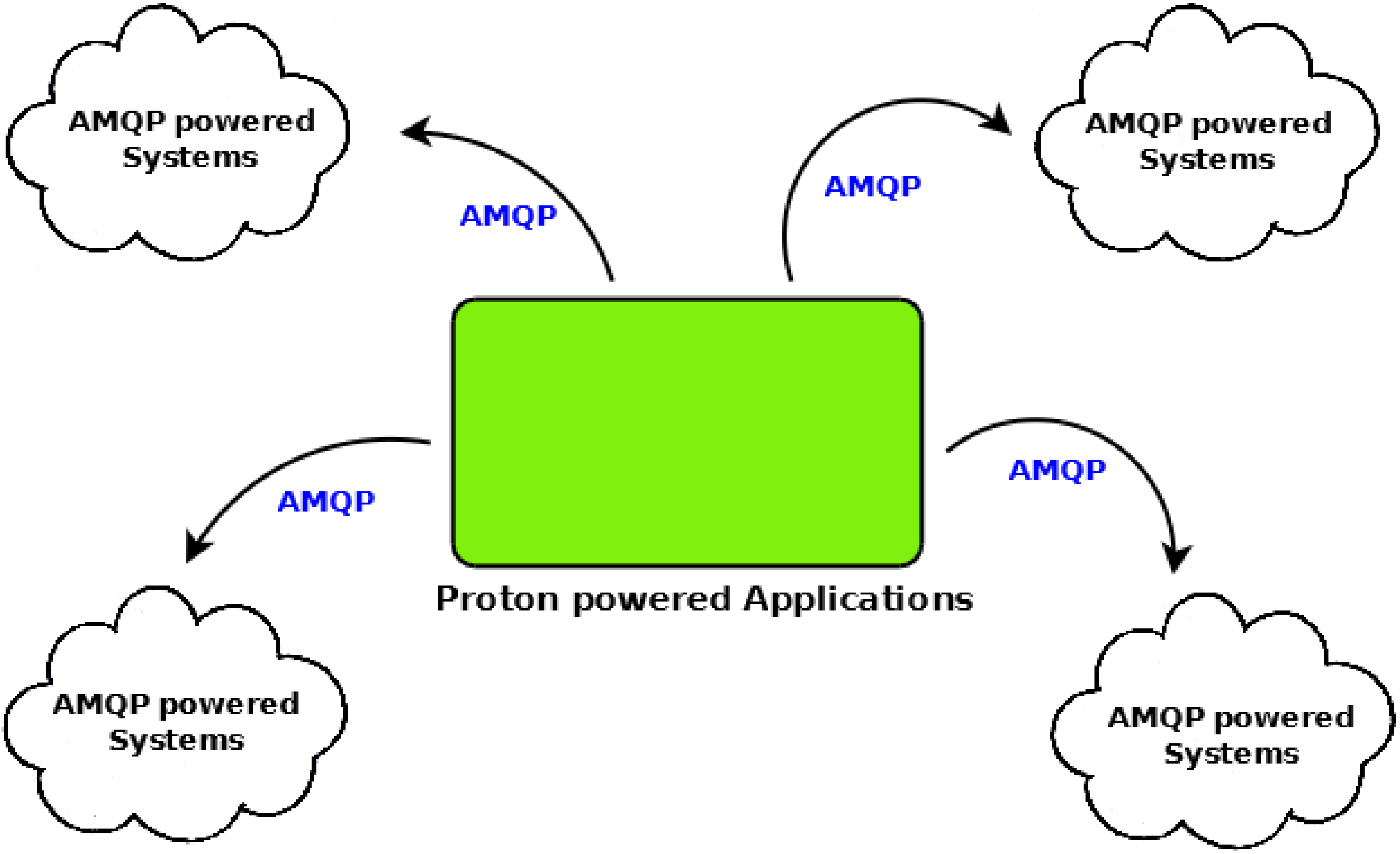
- Multi-language support.
  - Pure Java and pure C stacks.
  - Java Script will be added shortly.
  - Common design across the language implementations.
  - Common API across the language implementations.
  - Designed for easy language bindings. Using swig
    - Python
    - Ruby
    - PHP

Out of the box support for common data structures

- Strings
- Lists
- Maps



Proton is based on a Standard - AMQP



AMQP powered Systems

AMQP

AMQP

AMQP powered Systems

Proton powered Applications

AMQP

AMQP

AMQP powered Systems

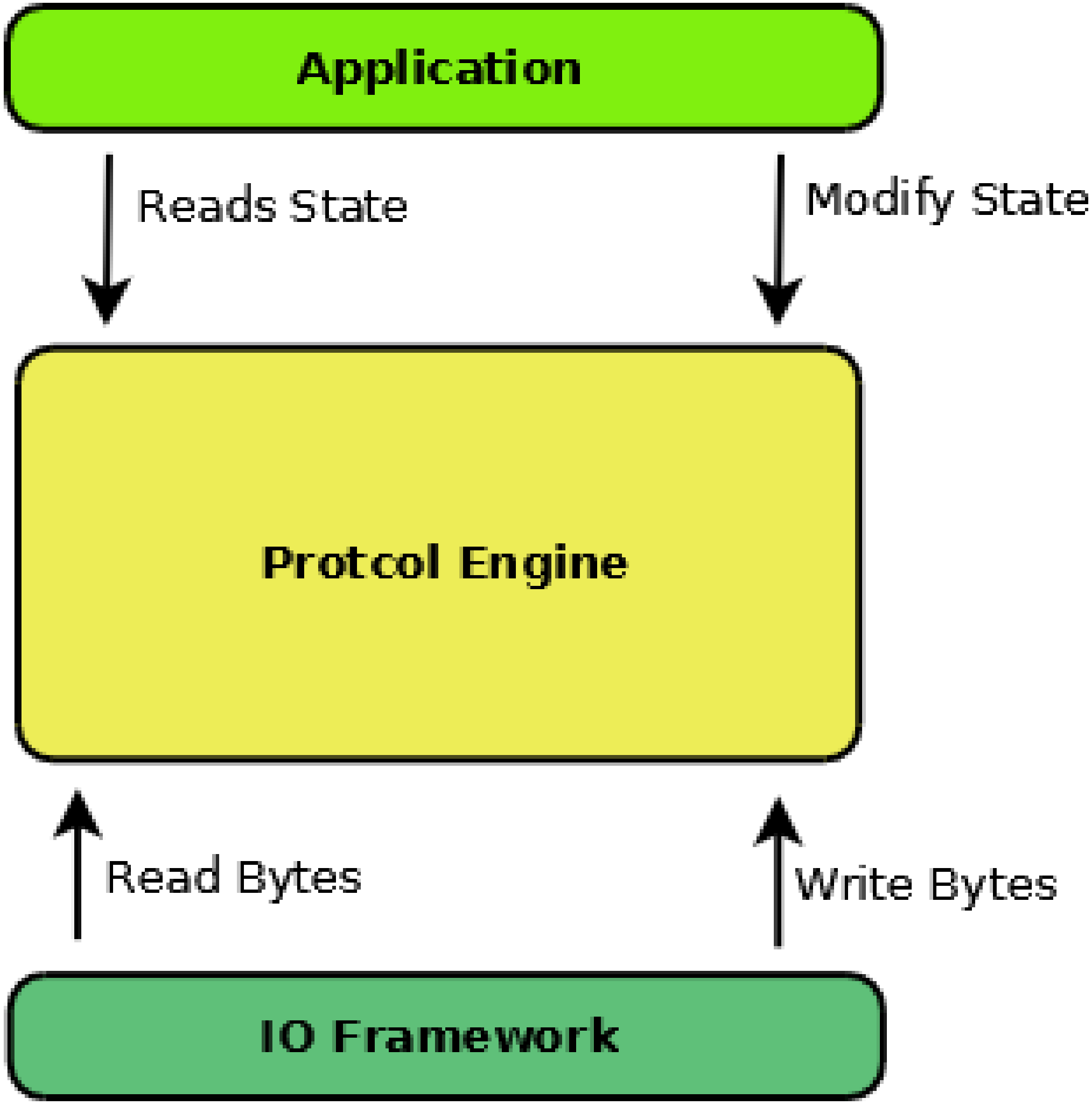
AMQP powered Systems

Proton Provides You With Two Options

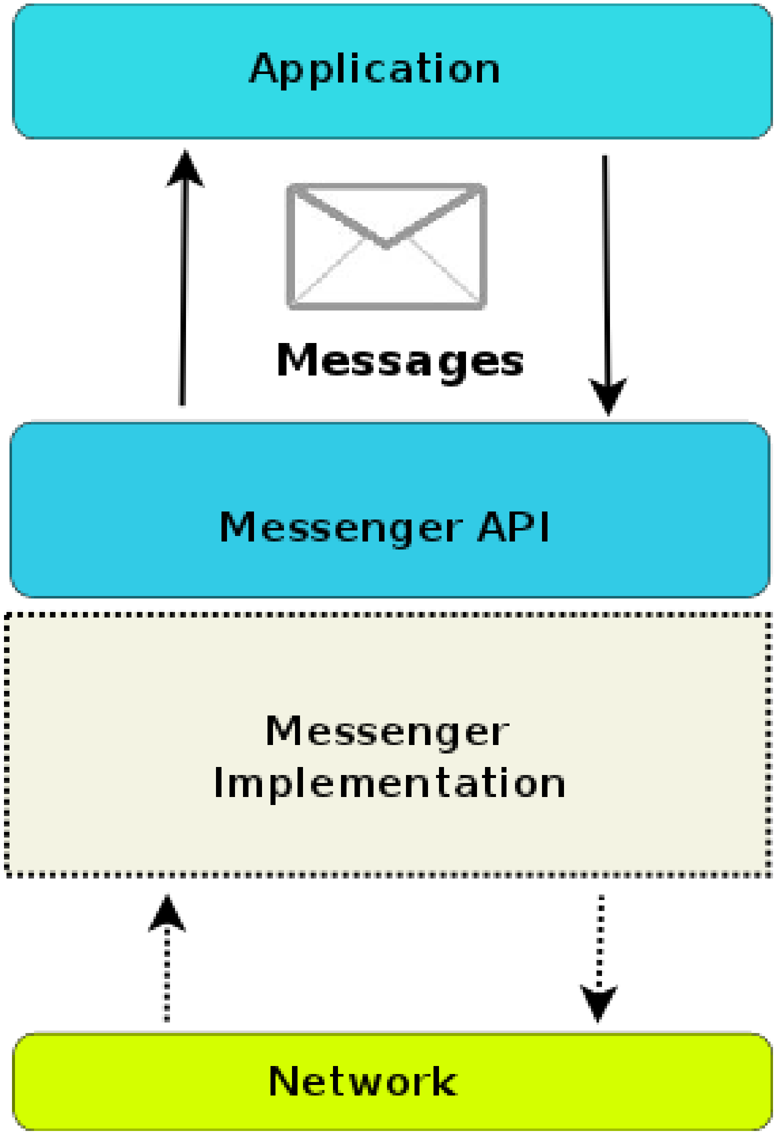
- The AMQP Messenger API, a simple but powerful interface to send and receive message over AMQP.

- AMQP Protocol Engine, a succinct encapsulation of the full AMQP protocol machinery.

Protocol Engine



Messenger API



Background

- 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

Background

- 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

Background

- 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

Background

- 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

Messenger

Sending

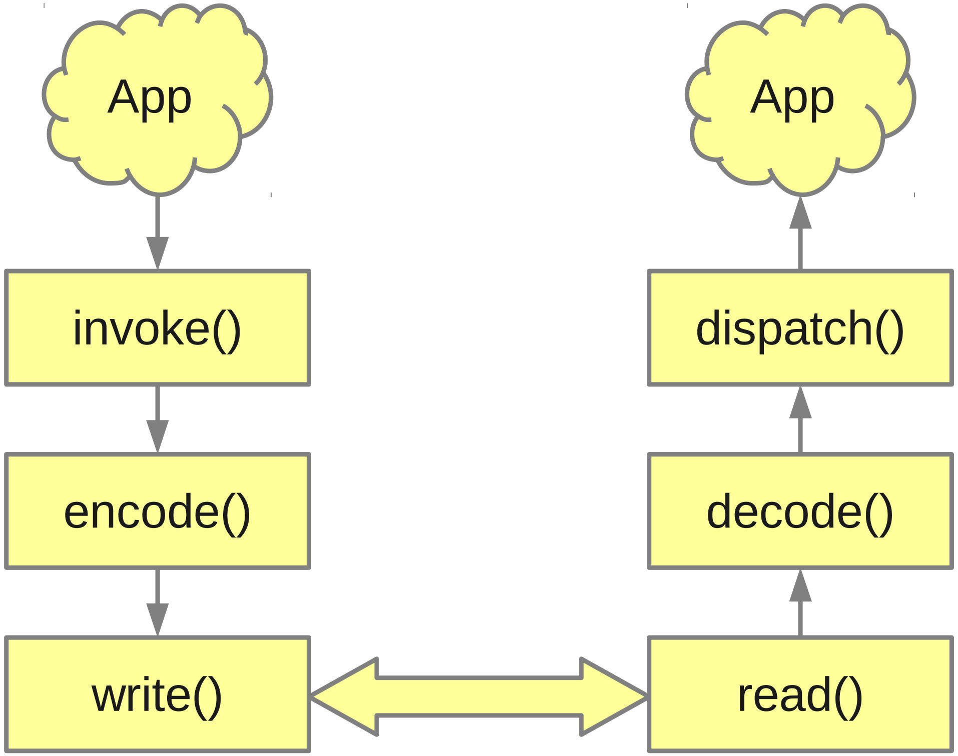
```
messenger = Messenger()

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(10)
  while messenger.incoming:
    messenger.get(msg)
    print msg.body

messenger.stop()
```

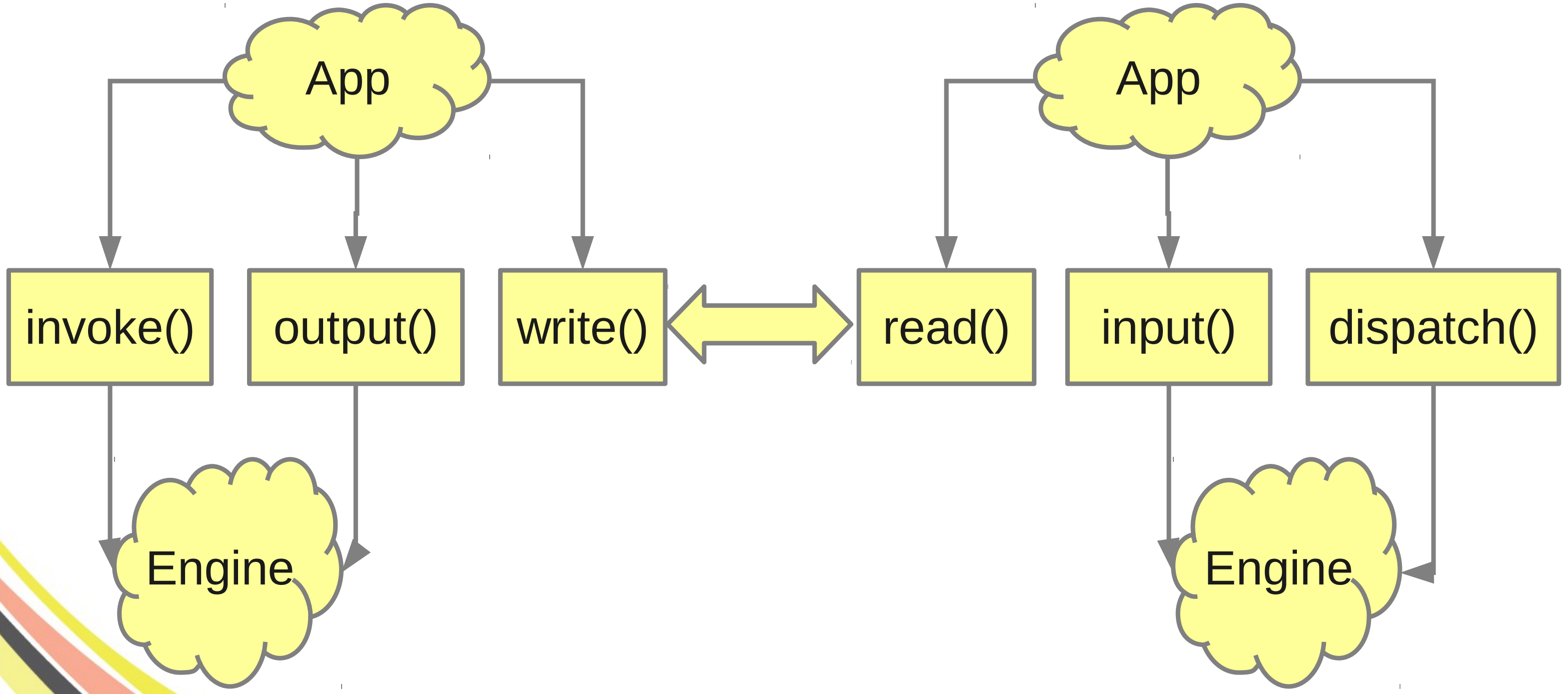
Protocol Engine

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



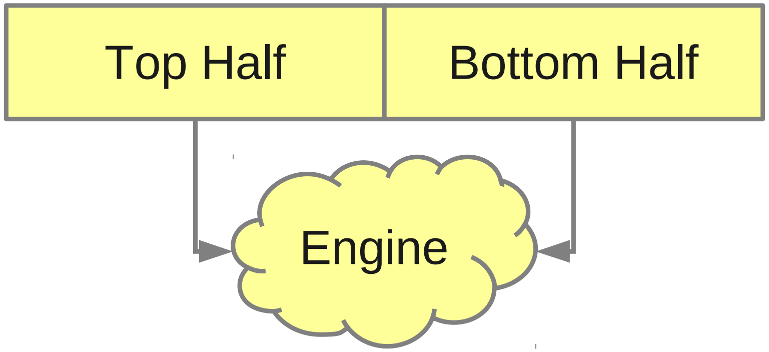
Protocol Engine

- 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

- 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
    - no callbacks
    - simple interface

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(10)
  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.incoming = 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(10)
  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

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