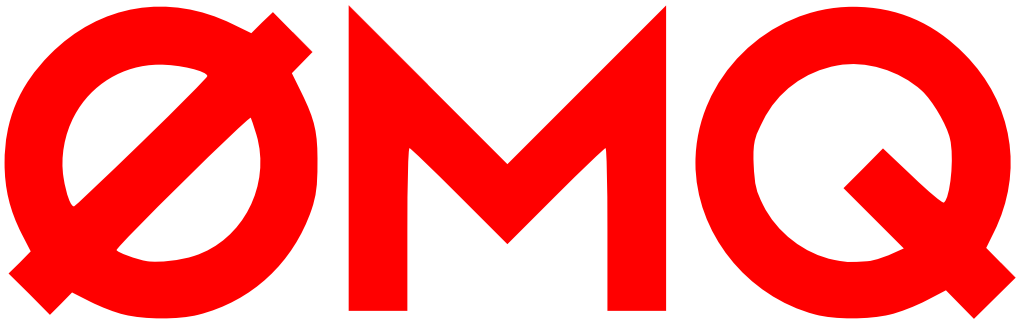
Introduction to ZeroMQ



1 December 2011

Pieter Hintjens, iMatix

Why did we need ØMQ?

Moore's Law means more moving pieces

Cost of connecting pieces was too high

- Custom TCP or UDP
- Broker-based messaging
- Clumsy RPC solutions

We needed cheaper connectivity

It had to be *really fast* and *really simple*

What is ØMQ?

Intelligent socket library for messaging

Many kinds of connection patterns

Multiplatform, multi-language (30+)

Fast (8M msg/sec, 30usec latency)

Small (20K lines of C++ code)

Open source LGPL (large community)

ØMQ Hello World

```
import org.zeromq.ZMQ;
public class hwclient {
    public static void main (String[] args){
        ZMQ.Context context = ZMQ.context (1);
        ZMQ.Socket socket = context.socket (ZMQ.REQ);
        socket.connect ("tcp://localhost:5555");
        socket.send ("Hello", 0);
        System.out.println (socket.recv(0));
    }
}
```

```
import org.zeromq.ZMQ;
public class hwserver {
    public static void main (String[] args) {
        ZMQ.Context context = ZMQ.context(1);
        ZMQ.Socket socket =
  context.socket(ZMQ.REP);
        socket.bind ("tcp://*:5555");
        while (true) {
            byte [] request = socket.recv (0);
            socket.send("World", 0);
        }
    }
}
```

Request-Reply Pattern

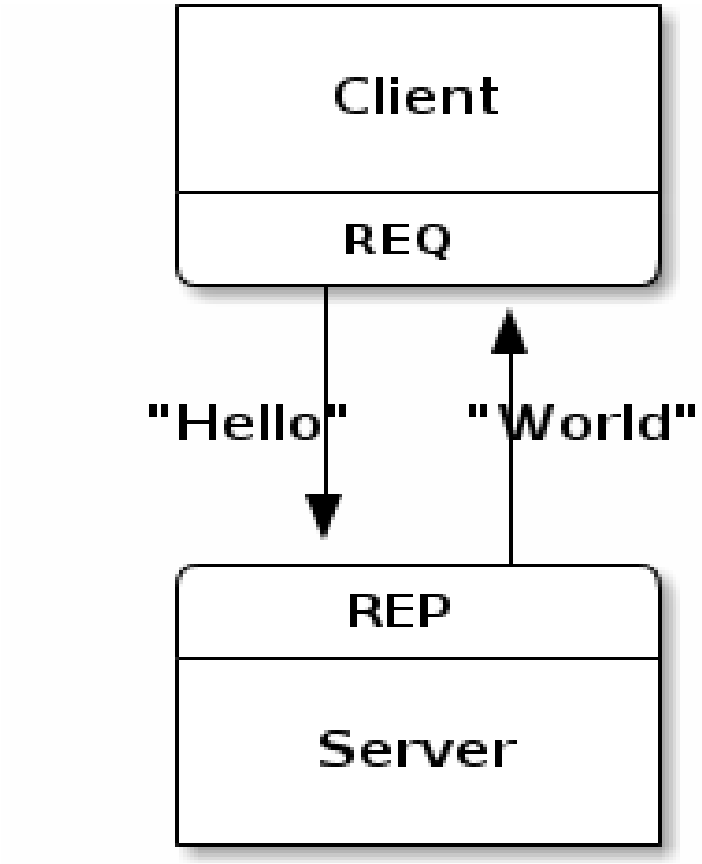


Figure 1 – Request-Reply

Publish-Subscribe Pattern

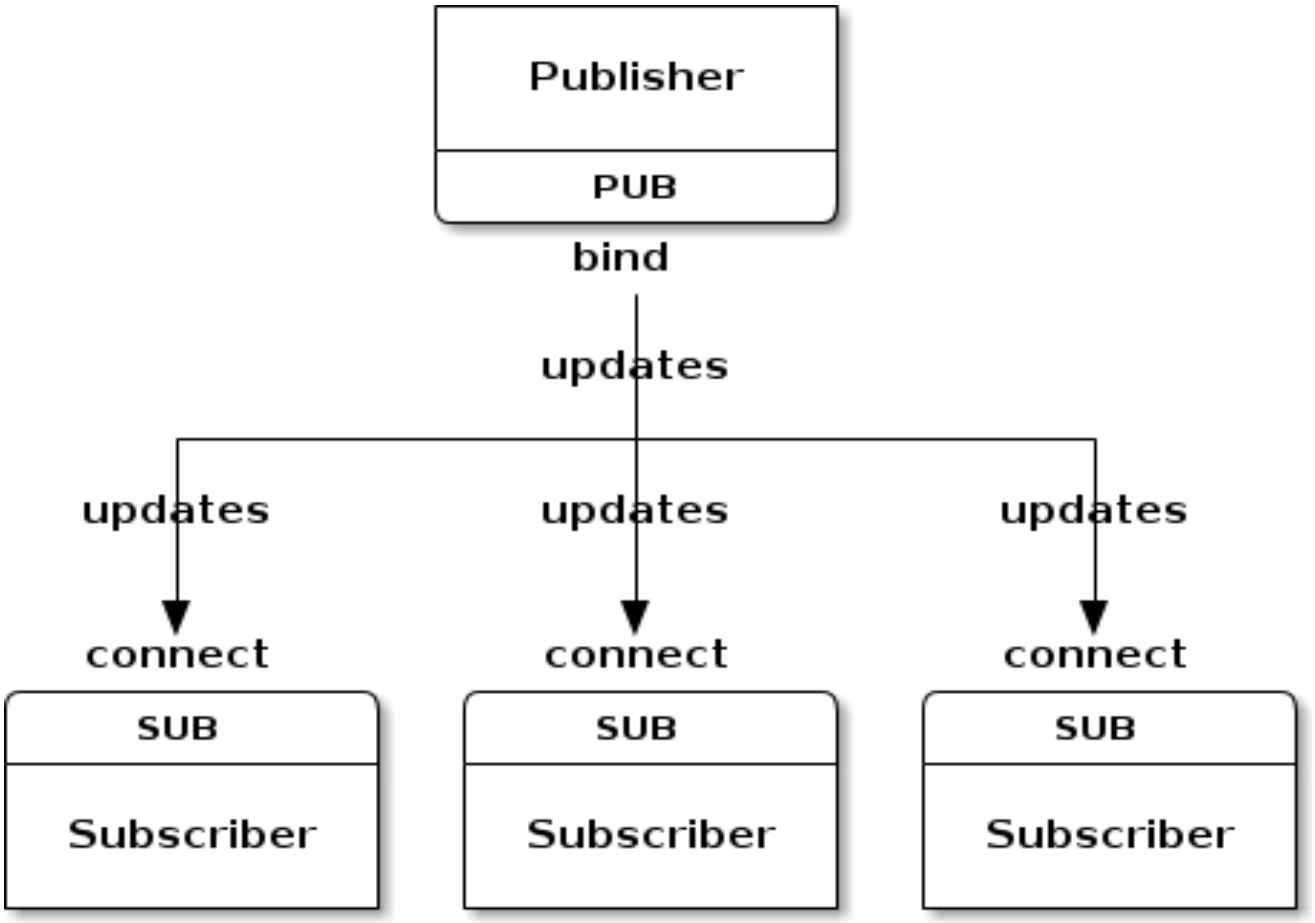


Figure 4 – Publish-Subscribe

Pipeline Pattern

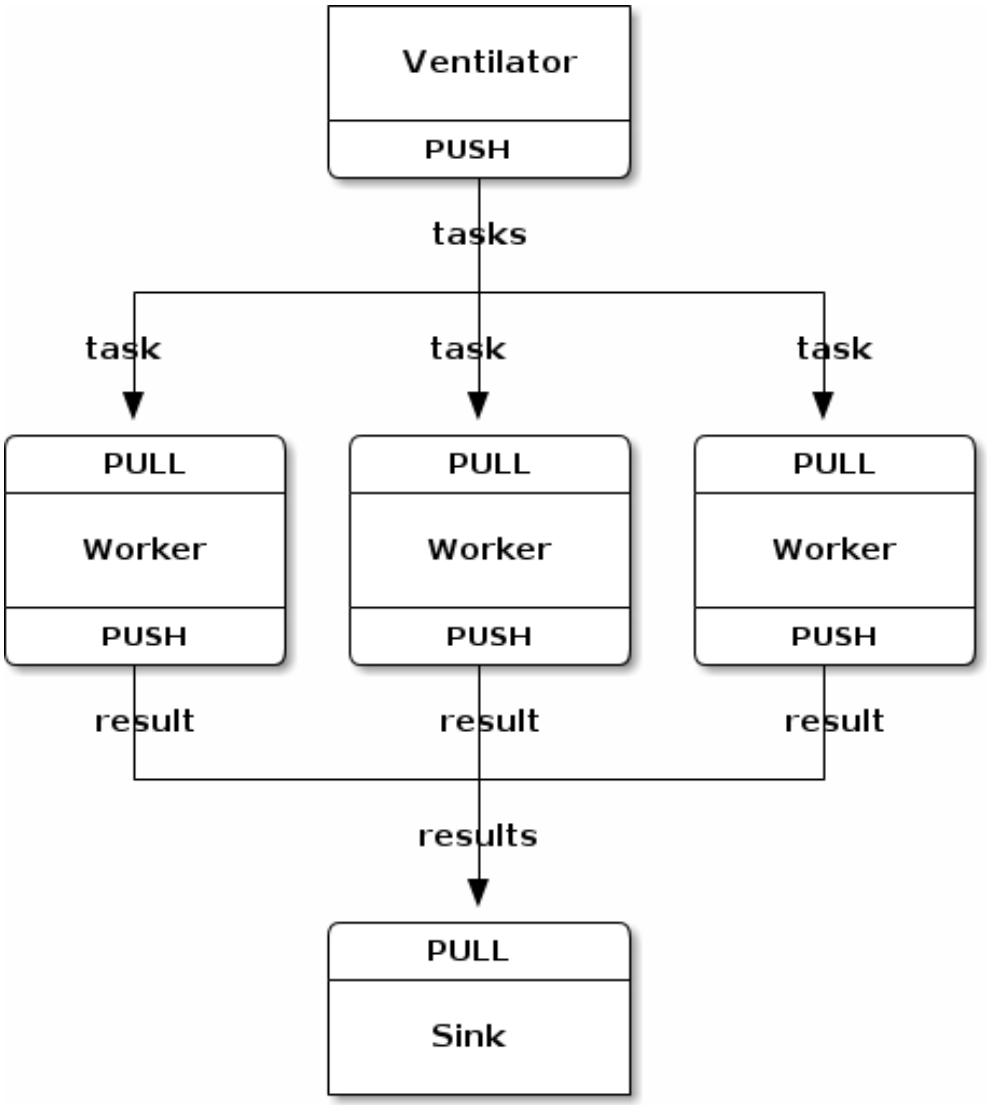


Figure 5 – Parallel Pipeline

Simple ØMQ Application

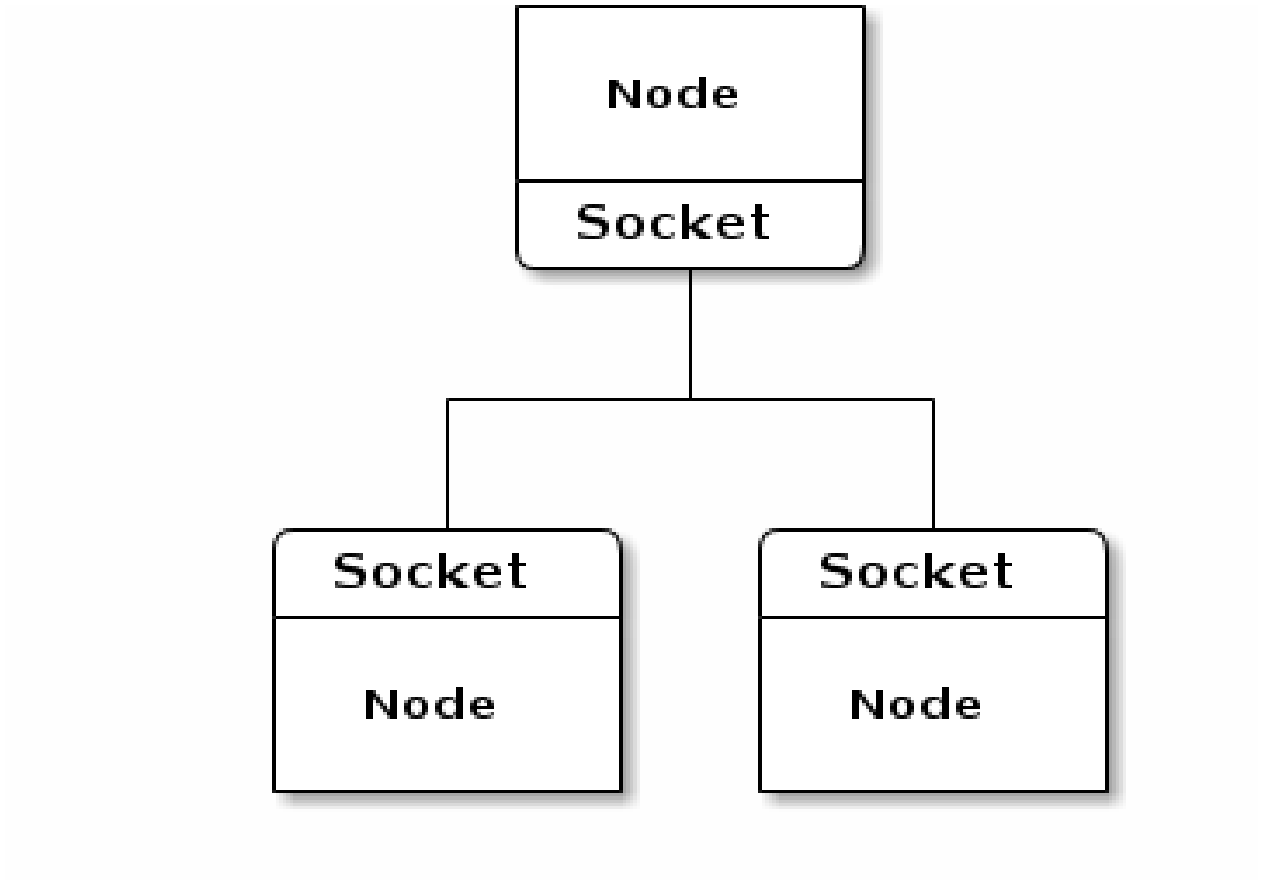


Figure 15 – Small scale ØMQ application

ØMQ Transports

Threads in one process (inproc://)

Processes on one box (ipc://)

Processes on one network (tcp://)

Multicast group (pgm://)

Multihop ØMQ Application

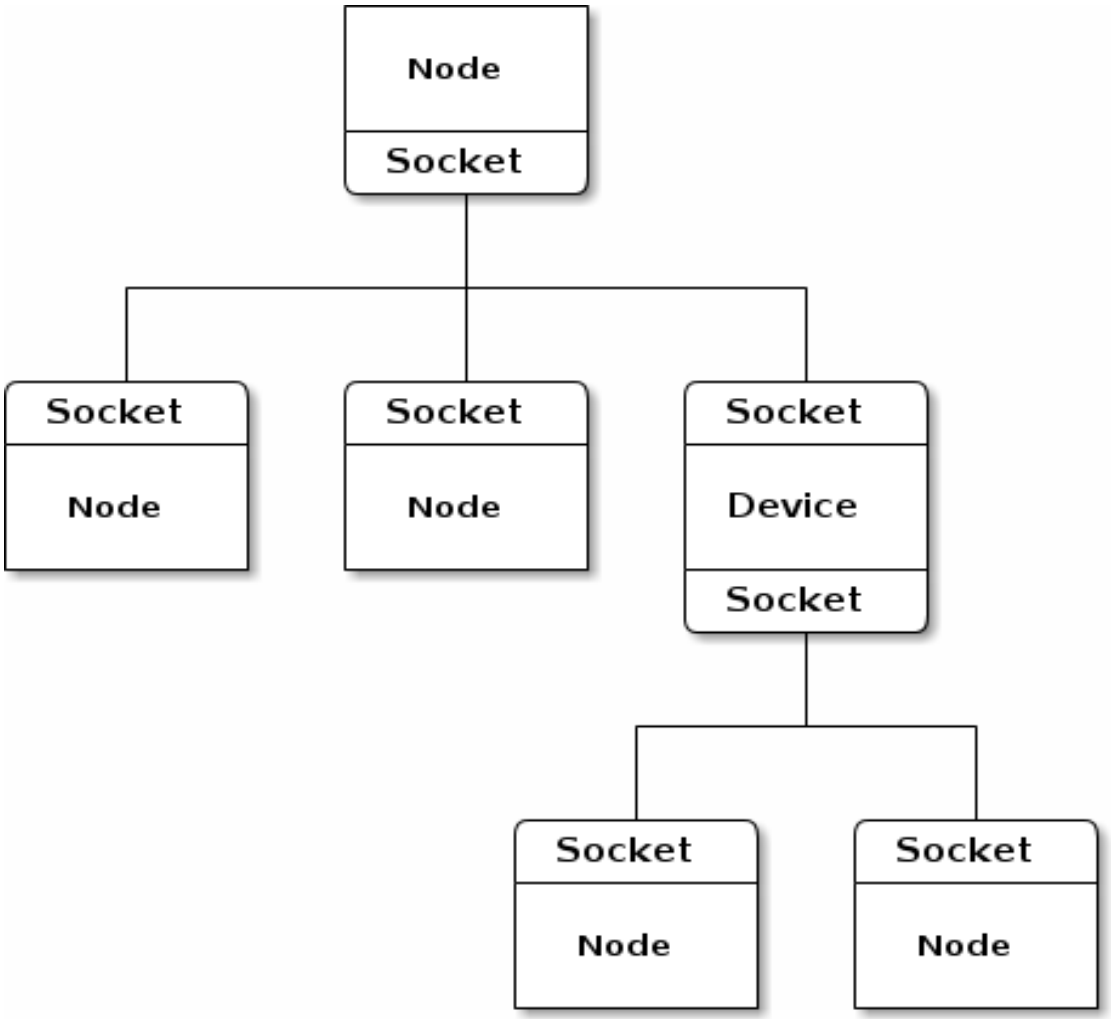


Figure 16 – Larger scale ØMQ application

Typical ØMQ Design

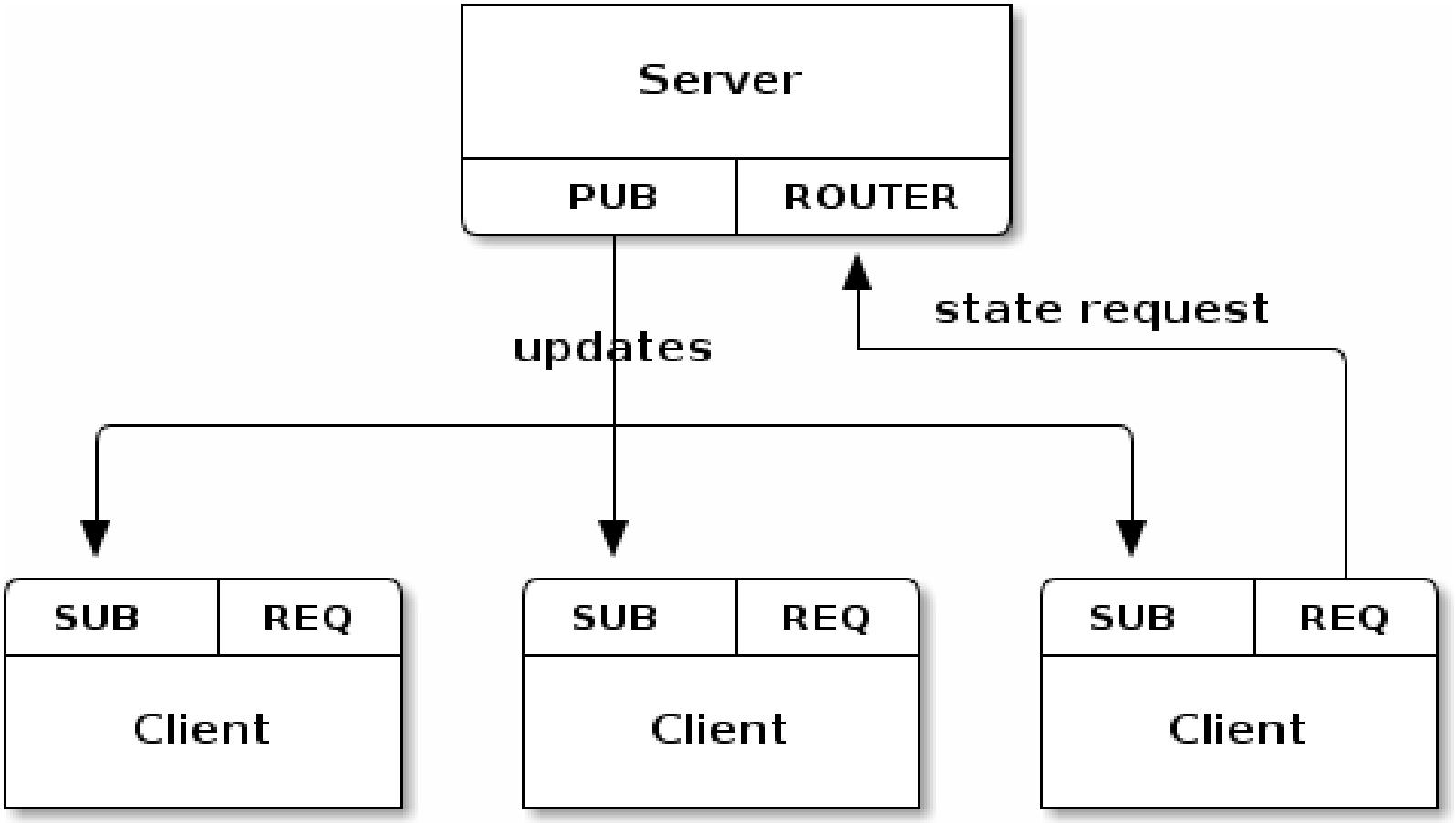


Figure 68 — State Replication

ØMQ Routing

Round-robin (REQ, PUSH, DEALER)

Multicast (PUB)

Fair-queuing (REP, SUB, PULL, DEALER)

Explicit addressing (ROUTER)

Unicast (PAIR)

ØMQ Features

Message blobs of 0 to N bytes

One socket connect to many sockets

Queuing at sender and receiver

Automatic TCP (re)connect

Zero-copy for large messages

ØMQ for Multithreading

Don't use locks, semaphores, mutexes

Design app as message-driven tasks

Each task reads from 1..n sockets

Tasks can talk over inproc://

Tasks can be split into processes over tcp://

No wait states, no locks, full CPU use

Scalable to any number of cores

ØMQ Benefits

Start with simple / fast language (Python)

Move to faster language where needed (C)

Run on arbitrary platforms (Windows, Android)

Scale to arbitrary sizes (2 cores, 16 cores...)

No per-core or per-seat licensing

Easy to experiment and learn

Working with ØMQ

Rapid prototyping of main components

Small protocols for main flows

ØMQ patterns for main flows

Break components up for performance

Profile and test

Improve incrementally over many cycles

ØMQ Origins

iMatix history of enterprise middleware

2004 – AMQP standard for JPMorganChase

2005 – OpenAMQ message broker/client

2008 – ØMQ/0.x for the avant-garde

2009 – ØMQ/1.x for pioneers (finance)

2010 – ØMQ/2.x for early adopters (foss)

2011 – ØMQ/3.x for mass market (cloud)

ØMQ Community

Large, 24/7 community of experts

1,000 people on dev list, 120 on IRC

Responsible for everything:

- Core development & packaging
- Language bindings (35 or more)
- Events and presentations

ØMQ is 100% owned by the community

- Which iMatix is a happy part of

ØMQ Resources

www.zeromq.org – main web site

zero.mq – community wiki

#zeromq – IRC channel on Freenode

zeromq-dev – email list on zeromq.org

github.com/zeromq – git repositories

zguide.zeromq.org – user guide

api.zeromq.org – reference manual