



Intro to Distributed Systems w/ ZeroMQ

Is it ZeroMq or 0MQ?

ØMQ

- ZeroMq, ØMQ, 0MQ, zmq
- Official site: <http://zeromq.org>

Origin Story

- Pieter Hintjens
 - CEO iMatix
 - 2007-2016
- Martin Sustrik
 - Software Architect / Lead Developer
 - 2007-2011
- JPMorganChase
 - 2004; need for new msg protocol 10K msg/sec => 100K msg/sec
 - Authored OpenAMQ (Advanced Message Queuing) protocol
- DowJones & Co
 - Hired to integrate into data distribution systems

Imatix

- Pieter felt AMQ community was 'toxic'; "don't try to fix organizations, start new ones"
- Joined iMatix focused on development of ZeroMq
- Difference of opinion between Martin & Pieter resulted in Martin leaving ZeroMq community and starting nanomsg
- Pieter emphasizing on 'community', Martin having expertise in authoring messaging libraries

What is ZeroMq?

- Broad spectrum of what it can be viewed as:
 - ‘sockets on steroids’
 - ‘mailboxes with routing’
 - Communication framework for creating robust distributed computing
 - RPC & Client/Server model
 - One-way distribution
 - Pipeline
- Multi-language
 - 52 languages (and counting)
- Multi-platform
 - Runs on ‘everything of interest’, not limited to Windows, Linux, MacOS, embedded, Android.....
- Multi-Transport Protocols
 - TCP, UDP, IPC, INPROC, PGM, EPGM

ZeroMq Core Concepts

- Born out of the financial industry
- As open-source products go, extremely well documented
 - RFC Standard
 - Comprehensive User/Developer Guide
- Library rather than suite of services
- Mechanism to build highly distributed systems
 - Library
 - Common Distributed System Recipes/Patterns

What's the Zero Stand For?

- '0' stands for:
 - Zero broker
 - Library vs broker service
 - Zero latency
 - Emphasis on speed/efficiency
 - Small framing protocol
 - Zero cost
 - Formula 1 design model; make it fast, then reliable
 - Zero waste
 - Distributed systems capitalize on available h/w, reducing specialized hw, reducing waste

Design Considerations

- How do we handle I/O?
 - Background communication engines
 - Lock-free data structures
 - No need for client application locking mechanisms
 - ZeroMQ sockets contain queues
- How do we handle dynamic components?
 - Component can come/go and ZeroMQ automatically reconnects
- How do we represent the message on the wire?
 - Len + envelope + payload
 - Does not impose format on messages, 'blobs', BYO protocol buffer
- What if we can't deliver the message immediately?
 - Message delivery can be viewed at 'atomic', all-or-nothing and asynchronous

Design Considerations (continued)

- How do we address slow consumers?
 - Dependent on messaging pattern
- How do we address lost messages?
 - All-or-Nothing, background communication engine
- What if we need to use a different network transport?
 - Singular API, no need to change code for alternative transport
- How do we route messages?
 - Routing part of the messaging pattern
- APIs for alternative languages?
 - 50+ languages supported
- How do we represent data for heterogeneous architectures?
 - Provides network marshalling
- How do we handle network errors?
 - Background tasks + retry provides robust delivery

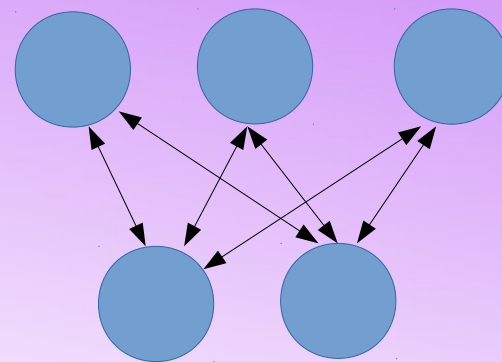
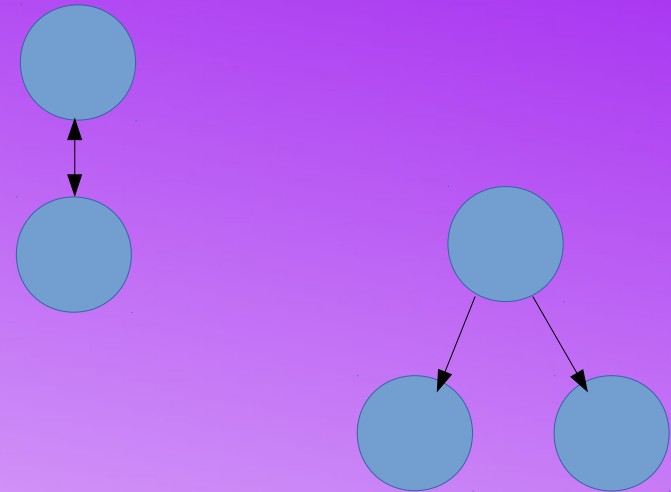
Socket Types

- With ZeroMQ, the term 'socket' has a less primitive meaning
- Built in queuing
- Personalities
 - PUB
 - SUB
 - REQ
 - REP
 - ROUTER
 - DEALER
 - PUSH
 - PULL
 - PAIR

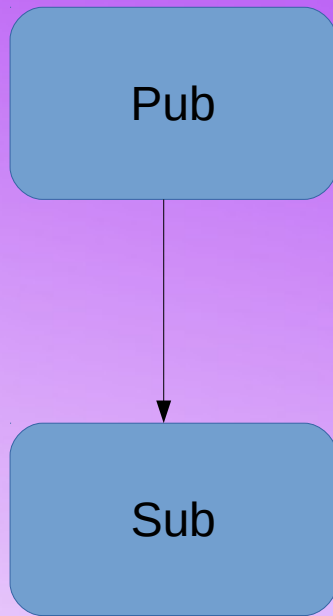
Node / Connectors

- Node Connection Types

- Connection Cardinality
- Connection Directionality



Pub / Sub 1-1



Observer pattern; 'topics'

Radio broadcast; miss everything before you begin listening

```
lipeltgm@kaylee:~/IntroToZeroMq/PubSub1-1$ cat -n pub
```

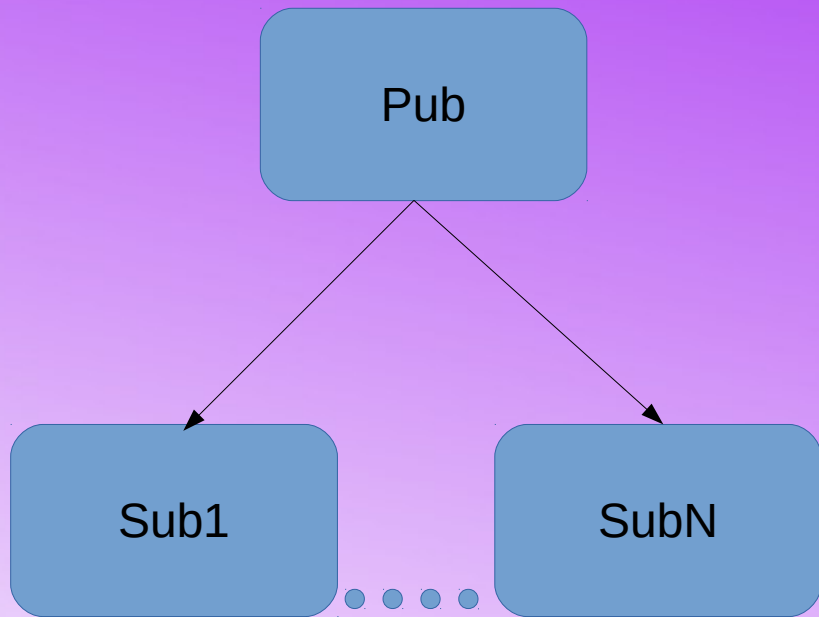
```
1  #!/usr/bin/python
2  import sys;
3  import zmq;
4  from random import randrange;
5  import time;
6
7  port=int(sys.argv[1]);
8  ctx = zmq.Context();
9  socket=ctx.socket(zmq.PUB);
10 socket.bind('tcp://*:%s'%port);
11
12 topic='TopicXX';
13 while True:
14     print "publishing";
15     socket.send('%s %s'%(topic, randrange(0,100)));
16     time.sleep(1);
```

```
lipeltgm@kaylee:~/IntroToZeroMq/PubSub1-1$ cat -n sub
```

```
1  #!/usr/bin/python
2  import sys;
3  import zmq;
4
5  port=int(sys.argv[1]);
6  ctx = zmq.Context();
7  socket=ctx.socket(zmq.SUB);
8  socket.connect('tcp://localhost:%s'%port);
9
10 topic='TopicXX';
11 socket.setsockopt_string(zmq.SUBSCRIBE, topic.decode('ascii'));
12
13 while True:
14     S=socket.recv();
15     print "S: %s"%(S);
```

Disconnected TCP, 1-N

Pub / Sub 1-N



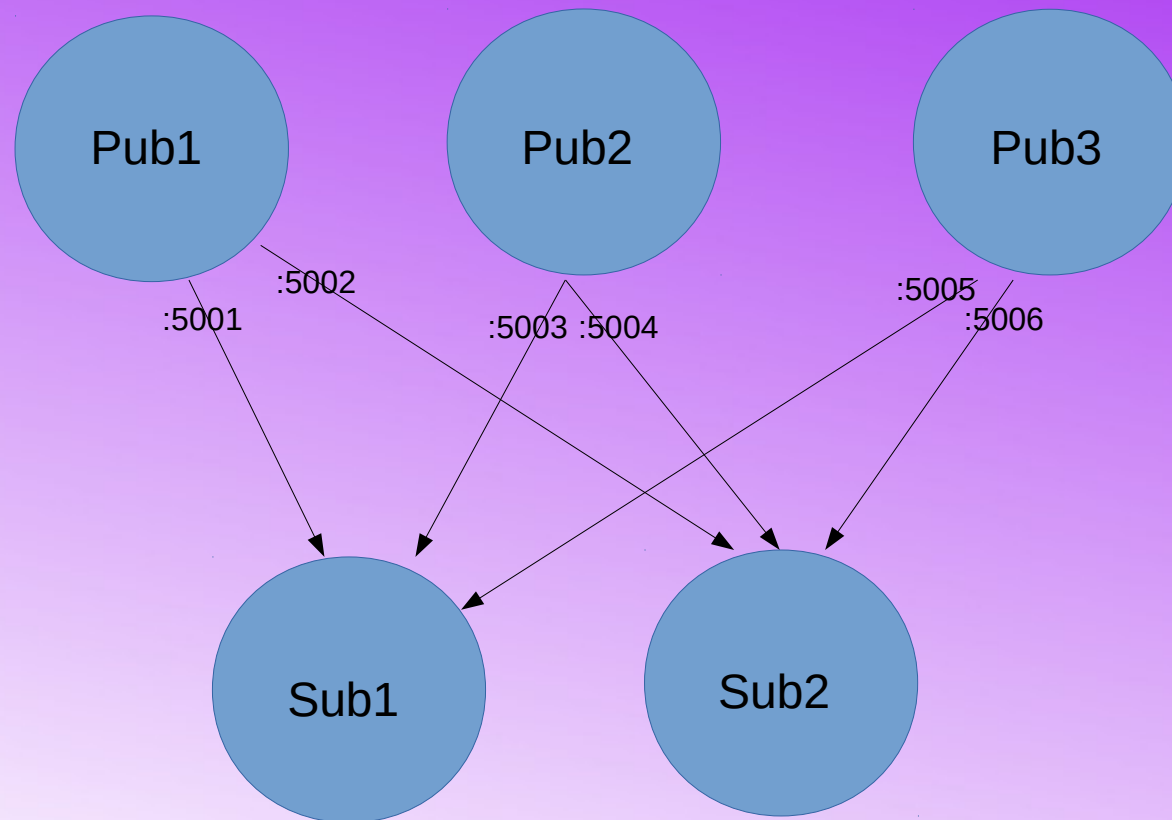
lipeltgm@kaylee:~/IntroToZeroMq/PubSub1-1\$ cat -n pub

```
1  #!/usr/bin/python
2  import sys;
3  import zmq;
4  from random import randrange;
5  import time;
6
7  port=int(sys.argv[1]);
8  ctx = zmq.Context();
9  socket=ctx.socket(zmq.PUB);
10 socket.bind('tcp://*:%s'%port);
11
12 topic='TopicXX';
13 while True:
14     print "publishing";
15     socket.send('%s %s'%(topic, randrange(0,100)));
16     time.sleep(1);
```

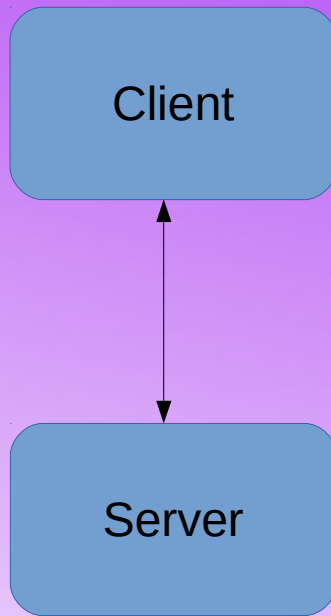
lipeltgm@kaylee:~/IntroToZeroMq/PubSub1-1\$ cat -n sub

```
1  #!/usr/bin/python
2  import sys;
3  import zmq;
4
5  port=int(sys.argv[1]);
6  ctx = zmq.Context();
7  socket=ctx.socket(zmq.SUB);
8  socket.connect('tcp://localhost:%s'%port);
9
10 topic='TopicXX';
11 socket.setsockopt_string(zmq.SUBSCRIBE,
12 topic.decode('ascii'));
13
14 while True:
15     S=socket.recv();
16     print "S: %s"%(S);
```

Pub / Sub N-M



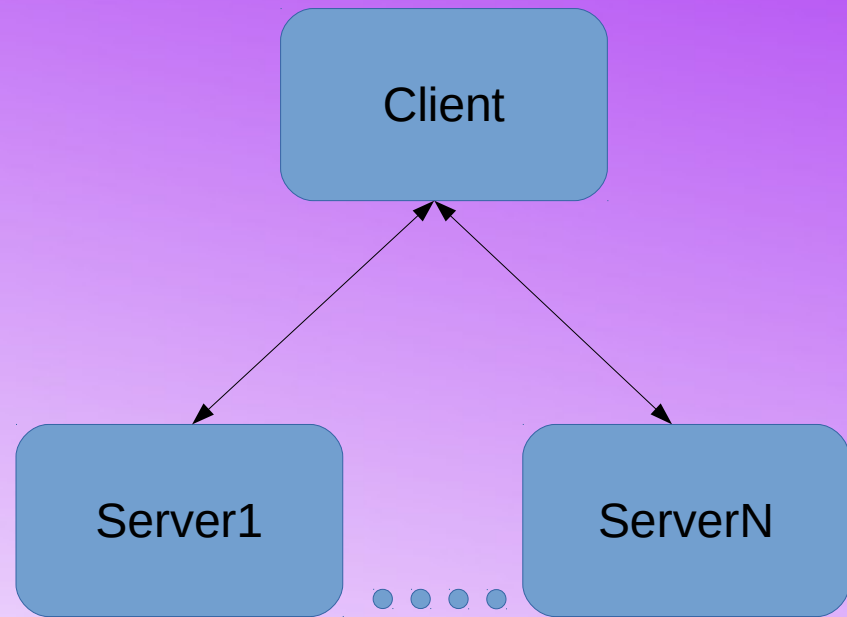
Request / Reply 1-1



Enforces ::send() / ::recv() protocol

```
• lipeltgm@kaylee:~/IntroToZeroMq/ReqRep1-1$ cat -n client
• 1      #!/usr/bin/python
• 2      import zmq;
• 3      import sys;
• 4      import time;
• 5
• 6      port=int(sys.argv[1]);
• 7      ctx=zmq.Context();
• 8      socket=ctx.socket(zmq.REQ);
• 9      socket.connect('tcp://localhost:%d'%(port));
• 10
• 11     while (True):
• 12         time.sleep(1);
• 13         socket.send('ping');
• 14         S=socket.recv();
• 15         print "got reply: %s"%(S);
•
• lipeltgm@kaylee:~/IntroToZeroMq/ReqRep1-1$ cat -n server
• 1      #!/usr/bin/python
• 2      import zmq;
• 3      import sys;
• 4      import time;
• 5
• 6      port=int(sys.argv[1]);
• 7      ctx=zmq.Context();
• 8      socket=ctx.socket(zmq.REP);
• 9      socket.bind('tcp://*:%d'%(port));
• 10
• 11     while (True):
• 12         S=socket.recv();
• 13         print 'got: %s'%(S);
• 14         socket.send('pong');
```


Request / Reply 1-N

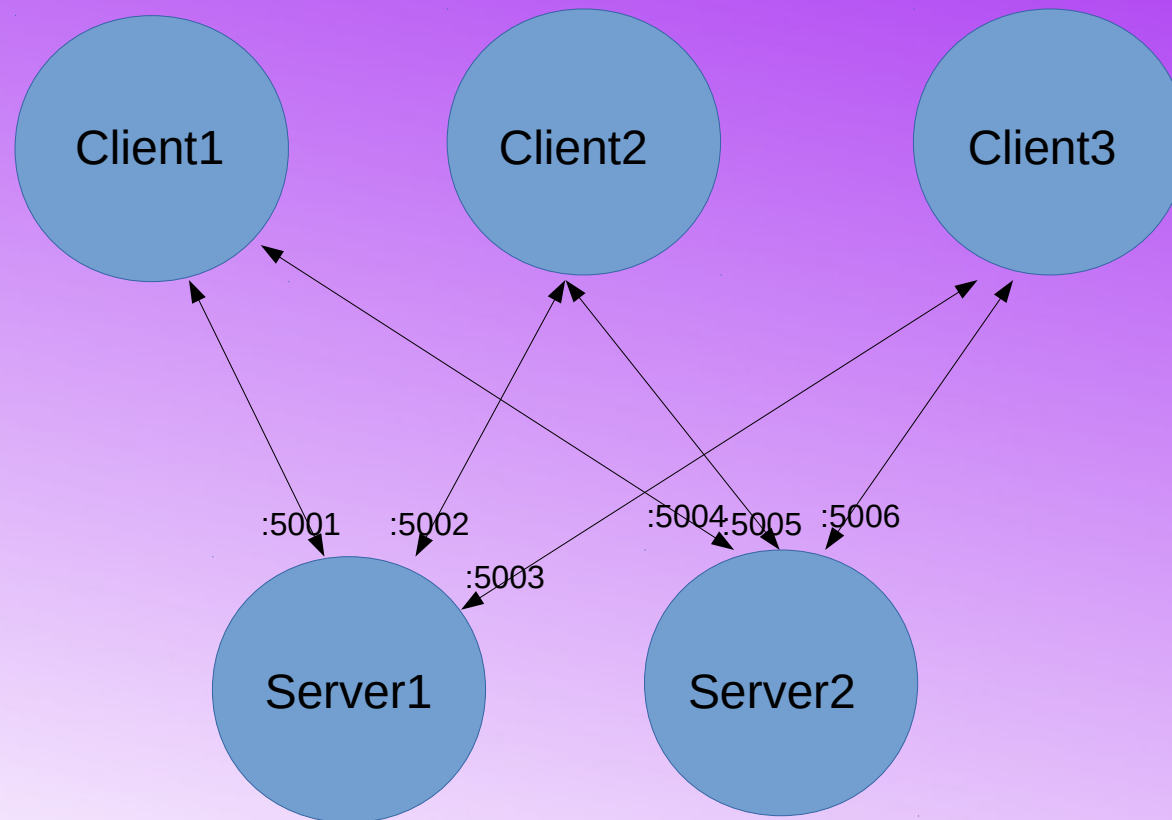


Enforces 'fair queuing' protocol

```
lipeltgm@kaylee:~/IntroToZeroMq/ReqRep1-N$ cat -n client
1  #!/usr/bin/python
2  import zmq;
3  import sys;
4  import time;
5
6  port1=int(sys.argv[1]);
7  port2=int(sys.argv[2]);
8  ctx=zmq.Context();
9  socket=ctx.socket(zmq.REQ);
10 socket.connect('tcp://localhost:%d'%(port1));
11 socket.connect('tcp://localhost:%d'%(port2));
12
13 while (True):
14     time.sleep(1);
15     socket.send('ping');
16     S=socket.recv();
17     print "got reply: %s"%(S);
```

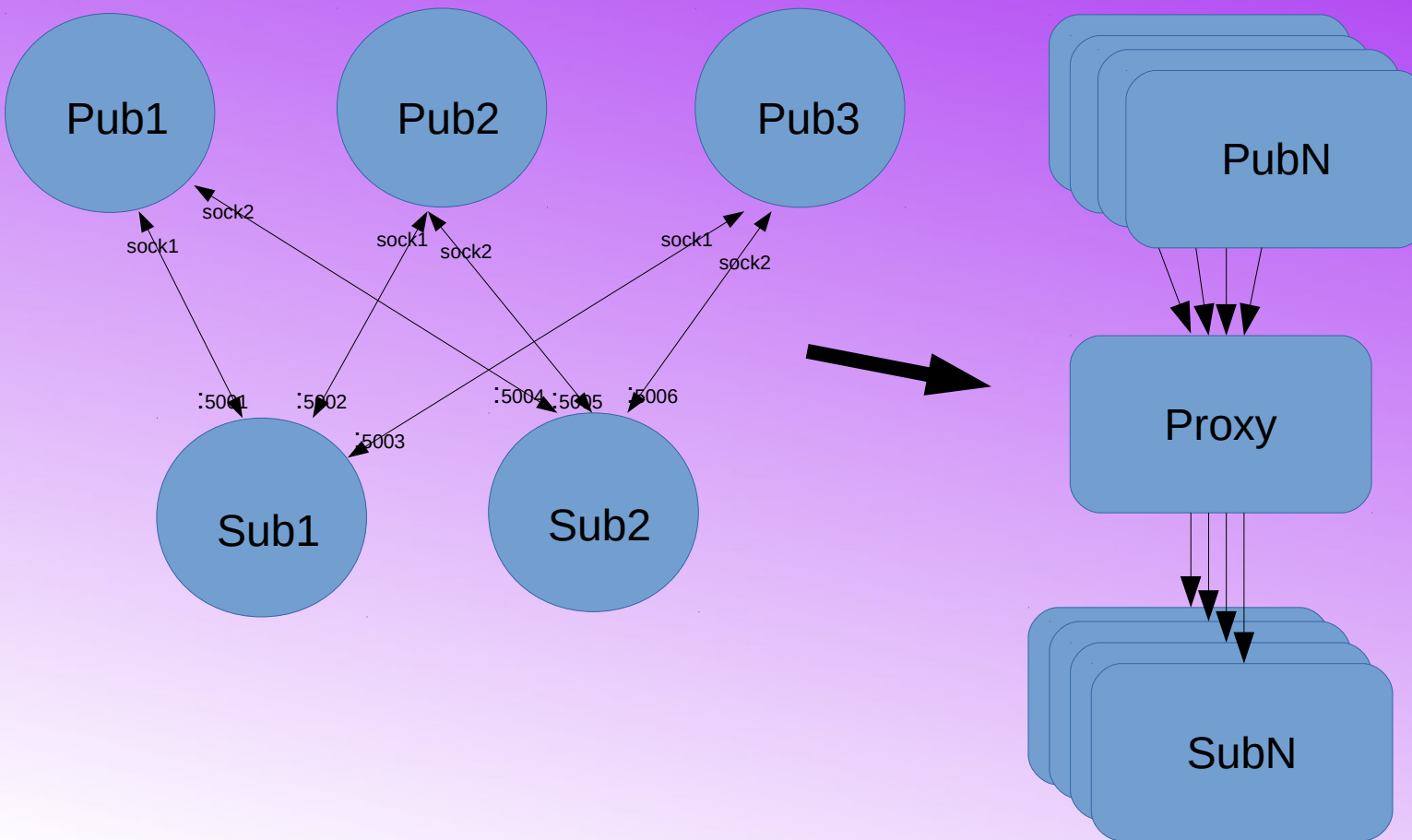
```
lipeltgm@kaylee:~/IntroToZeroMq/ReqRep1-1$ cat -n server
1  #!/usr/bin/python
2  import zmq;
3  import sys;
4  import time;
5
6  port=int(sys.argv[1]);
7  ctx=zmq.Context();
8  socket=ctx.socket(zmq.REP);
9  socket.bind('tcp://*:%d'%(port));
10
11 while (True):
12     S=socket.recv();
13     print 'got: %s'%(S);
14     socket.send('pong');
```

Request / Reply N-M



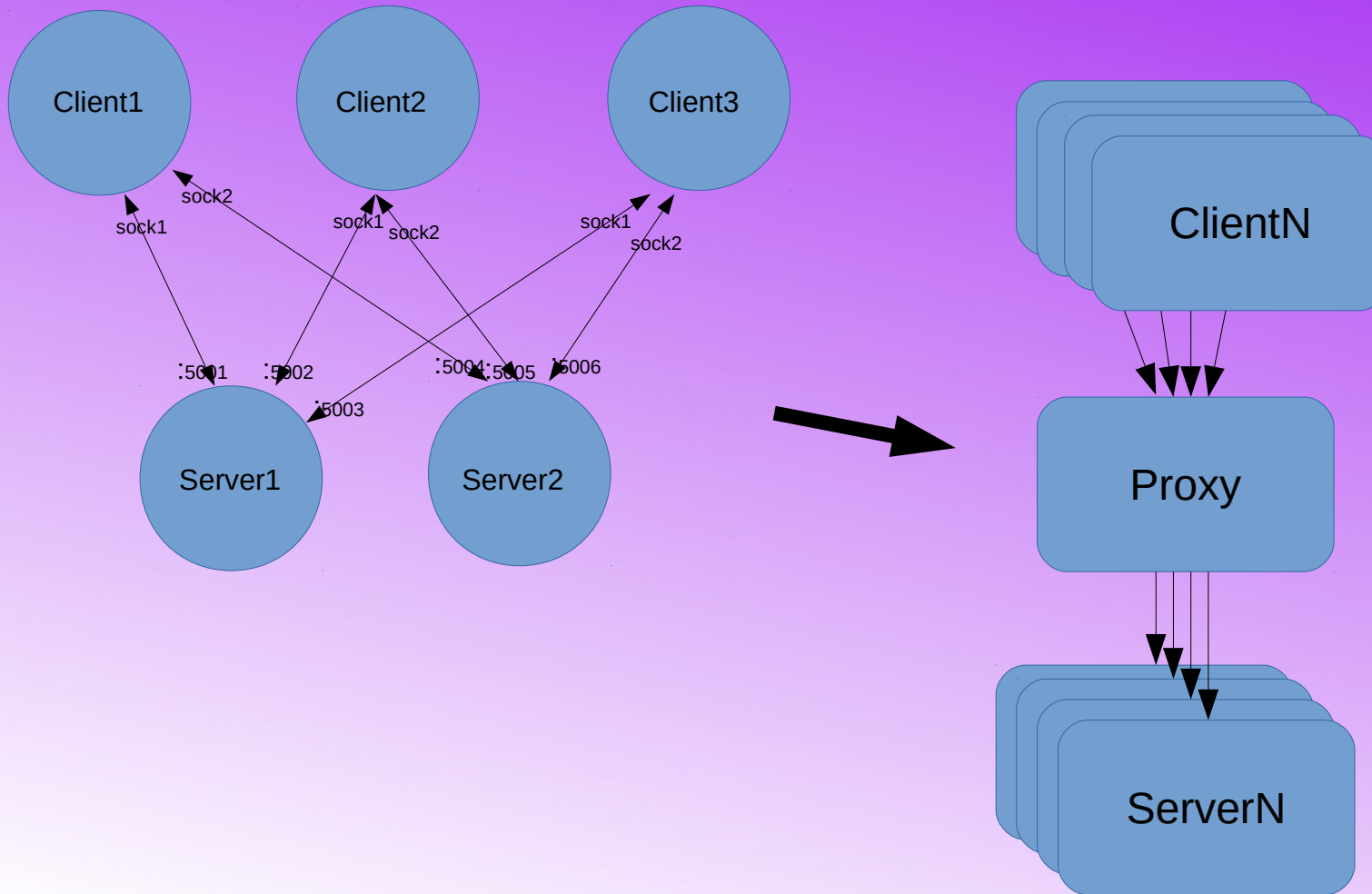
Data Distribution w/Intermediary

Pub/Sub



- XPUB/XSUB Sockets provide proxy mechanism
- XPUB/XSUB expose subscriptions as special messages,
- Forward subscription messages from subscriber to publisher

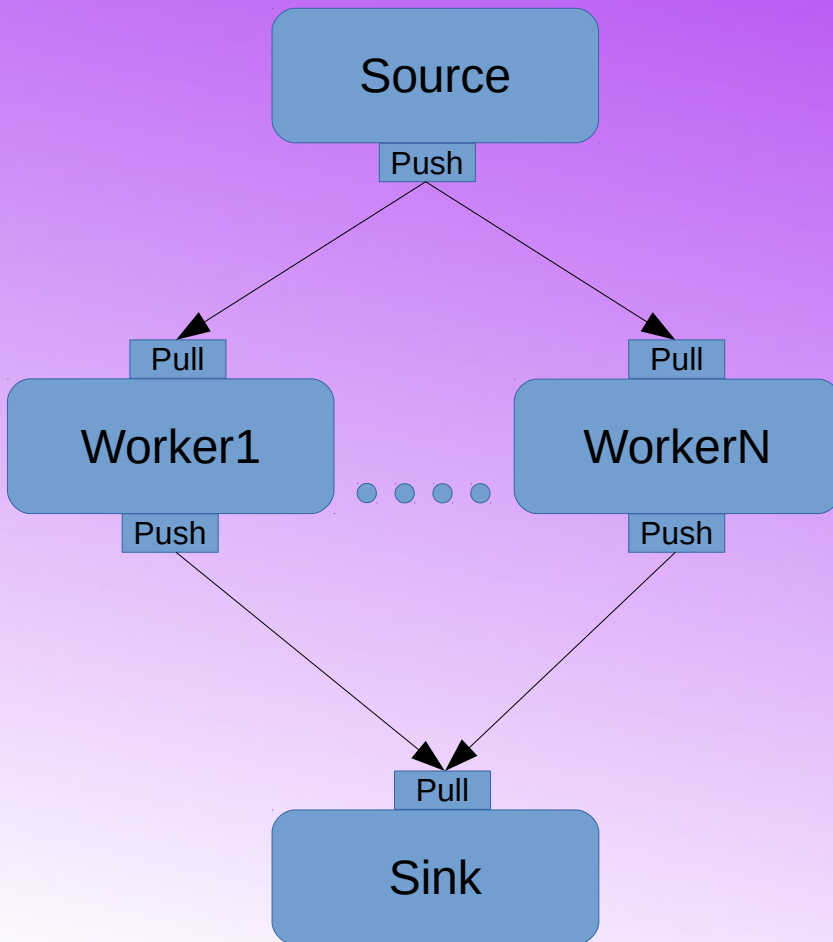
Data Distribution w/Intermediary Req/Rep



- ROUTER/DEALER socket pairing performs the magic

Parallel / Pipeline 1-N

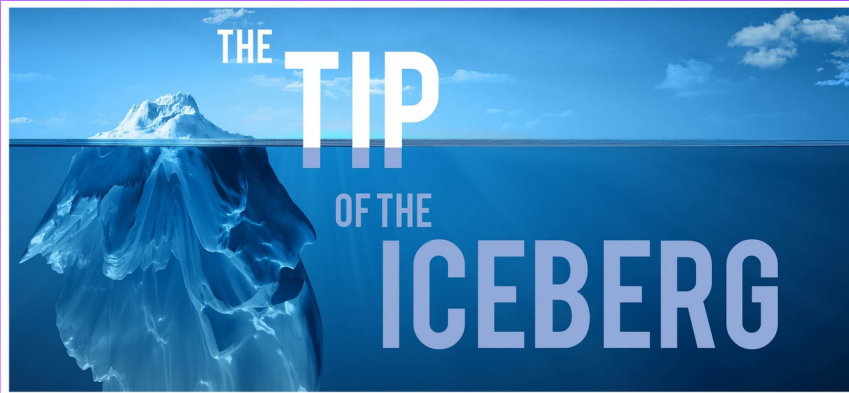
- Distributed / Parallel Processing Architecture



Advanced Messaging Patterns

- The Lazy Pirate pattern: reliable request/reply from the client side
- The Simple Pirate pattern: request/reply using load balancing
- The Parnoid Pirate pattern: reliable request/reply with heartbeating
- The Majordomo pattern: service-oriented reliable queuing
- The Titanic pattern: disk-based/disconnected reliable queuing
- The Binary Star pattern: primary-backup server failover
- The Freelance pattern: brokerless reliable request/reply

Quick Summary



- ZeroMQ was authored as a library-based high-performance messaging framework
- Multi-language, multi-platform & multi-transport
- Message contents are considered 'blobs', bring your own protocol buffer or simply use string
- Queuing/Asynchronous deliver via background delivery engines
- Flexible design componentry to support quick/simple peer-to-peer to advanced message broker, heartbeating, dynamic heterogeneous nodes
- Advanced patterns to common distributed system challenges
- Substantial documentation and support community, ZeroMQ Guide is detailed document with light humor, worth the read
- Now, go build something cool.

Contact Info

- Slides:
 - <https://github.com/fsk-software/pub/IntroToZeroMq/>
- FSK Consulting Inc.
 - <http://fsksoftware.com>
- Blog: <http://dragonquest64.blogspot.com>
- Slack: pymntos.slack.com lipeltgm

