ØMQ

Zero Message Queue

The Intelligent Transport Layer

or "Super Sockets"

What’s a Message?

noun:
1. a verbal, written, or recorded communication sent to or left for a recipient who cannot be contacted directly.

Rails.logger.debug “This is a message”

JobQueue.do_that_thing

object.method

What's a Queue?

noun:

1. a line or sequence of people or vehicles <u>awaiting their turn to be attended</u> to or to proceed.

2. a list of data items, commands, etc., <u>stored so as to be retrievable</u> in a definite order, usually the order of insertion.

What’s a Socket?

noun:

1. The concave part of a joint that receives the end of a bone.

2. An endpoint for communicating across a network.

Applications work in “connections” which are from one socket (server) to another (client).

Other Queuing Systems

- Applications (servers) that clients communicate with, e.g.:
  - Resque (redis database)
  - ActiveMQ (apache AMQP server)
  - RabbitMQ (AMQP server)

Recap: Sockets

TCP sockets: a connected stream where the data is guaranteed to be received in the order it is sent. Server binds then client connects.

UDP sockets: a connectionless way to send datagrams (messages). No acknowledgement or ordering.

Both require higher level protocols to carry their packets.

Communications can be routed across different networks.

Zero MQ Sockets

Either can bind or connect, in any order,
to multiple sockets simultaneously.

Sends/receives only whole messages.

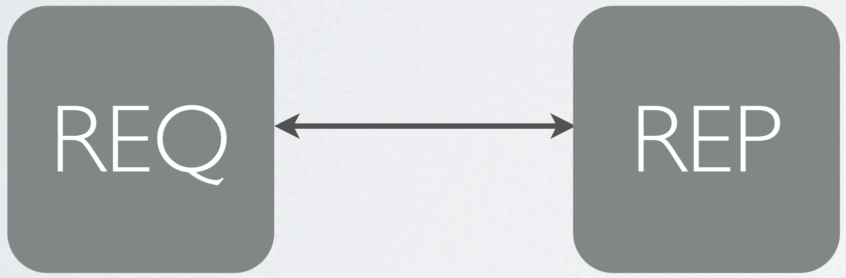
Sockets are configured with specific usage semantics,
such as:

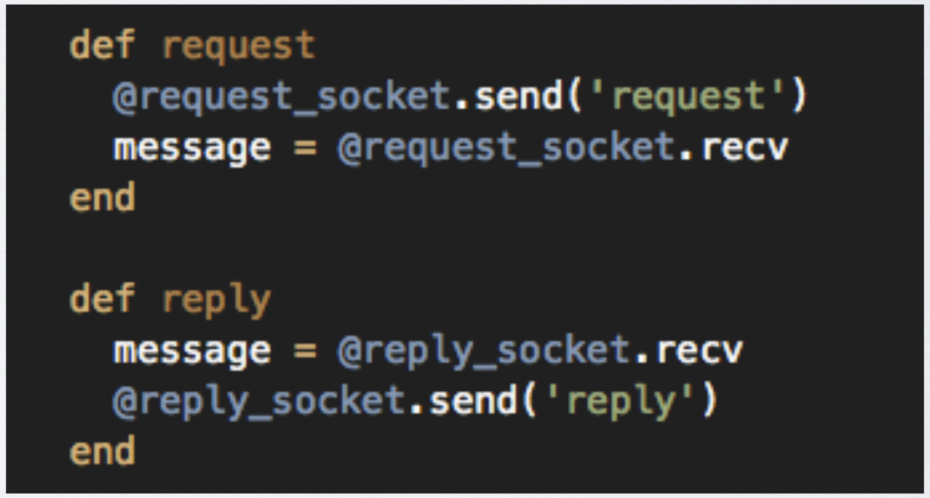
Request <=> Reply
Push => Pull
Publish <=> Subscribe

Messaging Patterns:

Request <=> Reply

Even the simplest example can handle multiple connections simultaneously.

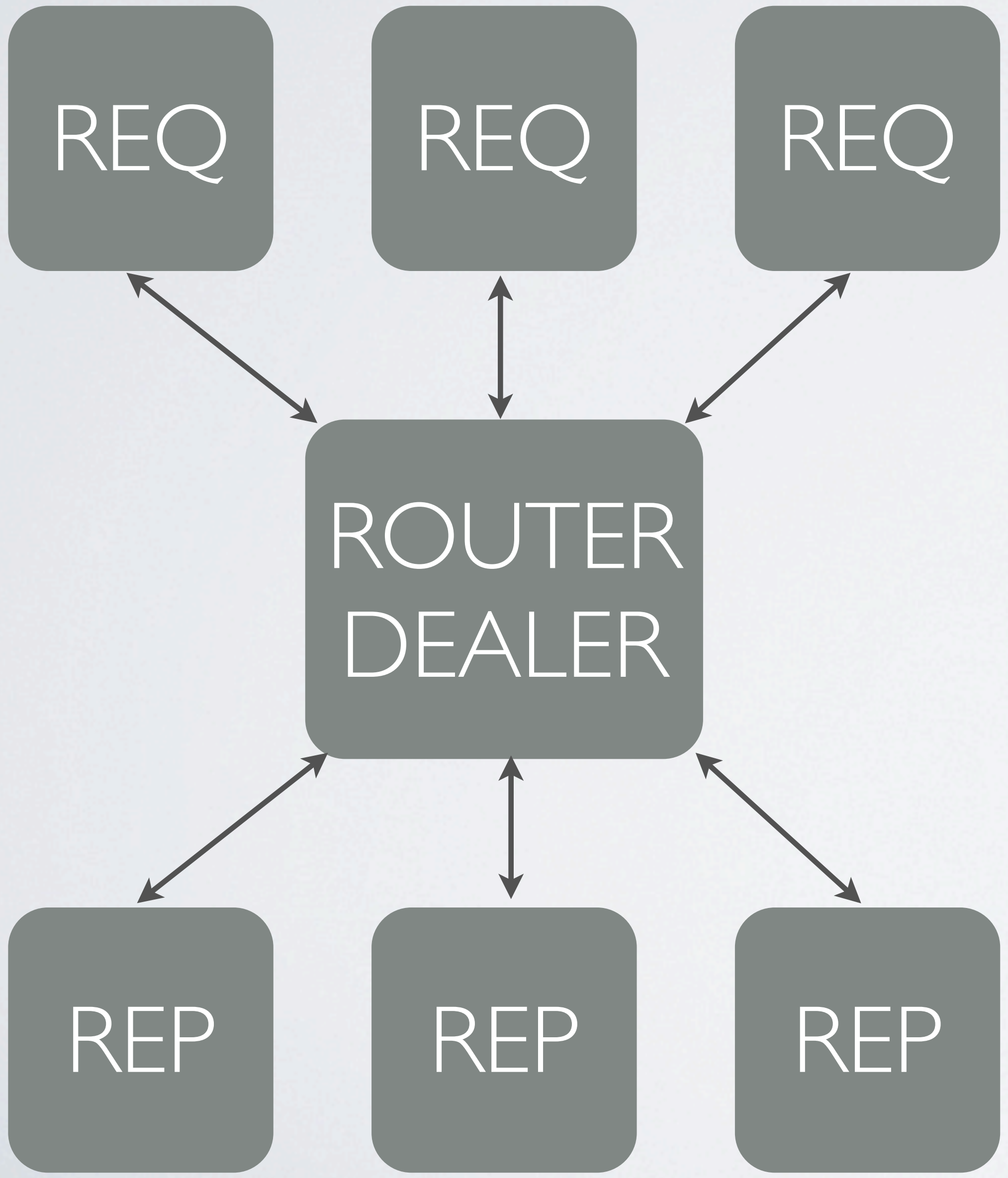




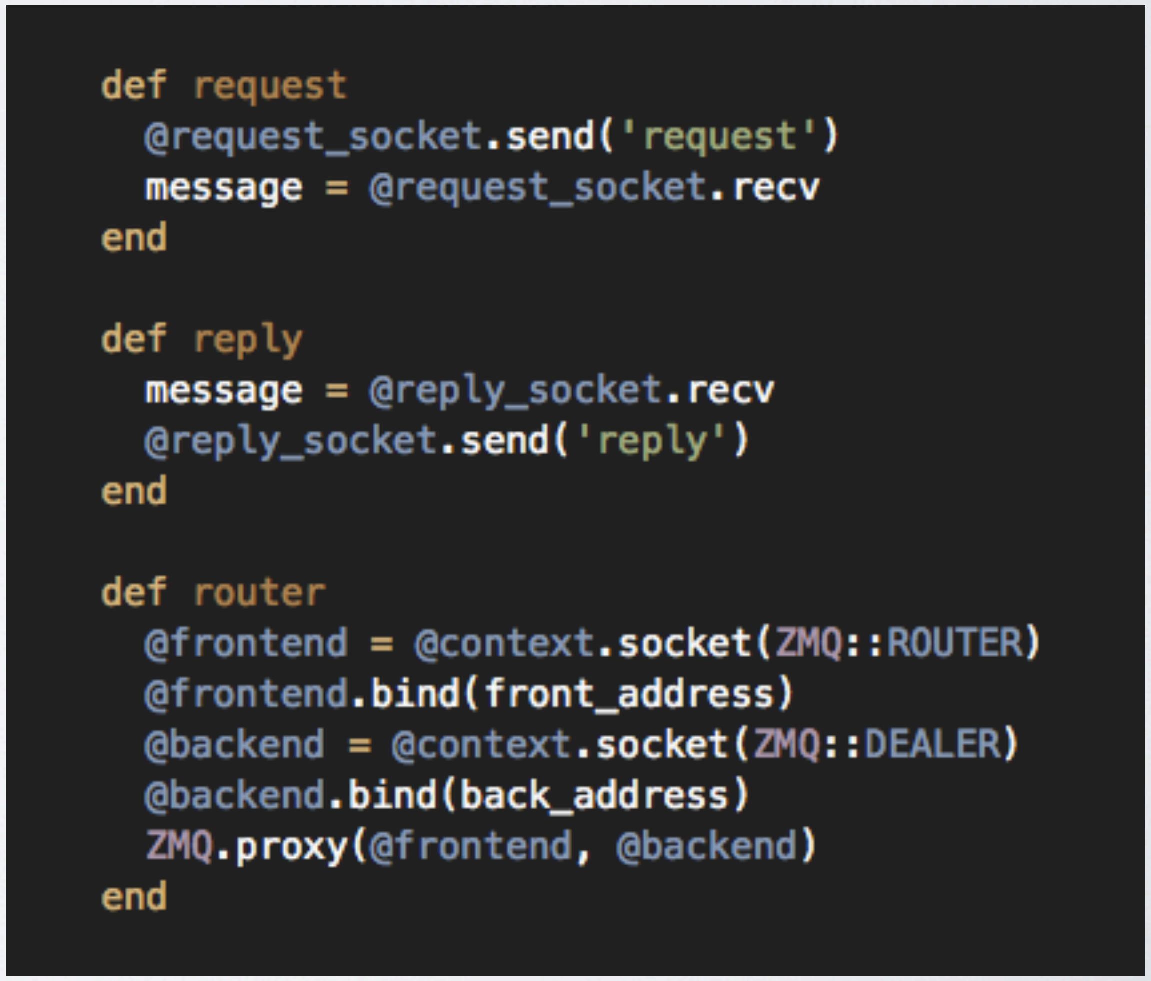
```
def request
  @request_socket.send('request')
  message = @request_socket.recv
end

def reply
  message = @reply_socket.recv
  @reply_socket.send('reply')
end
```

Messaging Patterns:

Request <=> Reply



Routed example:

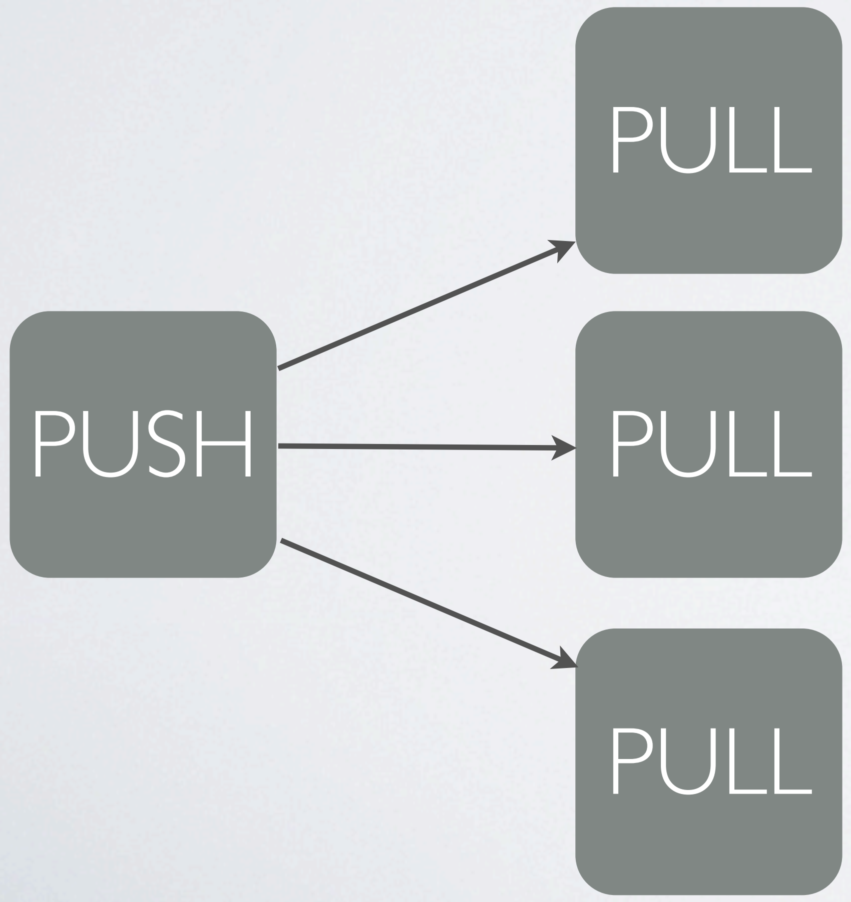


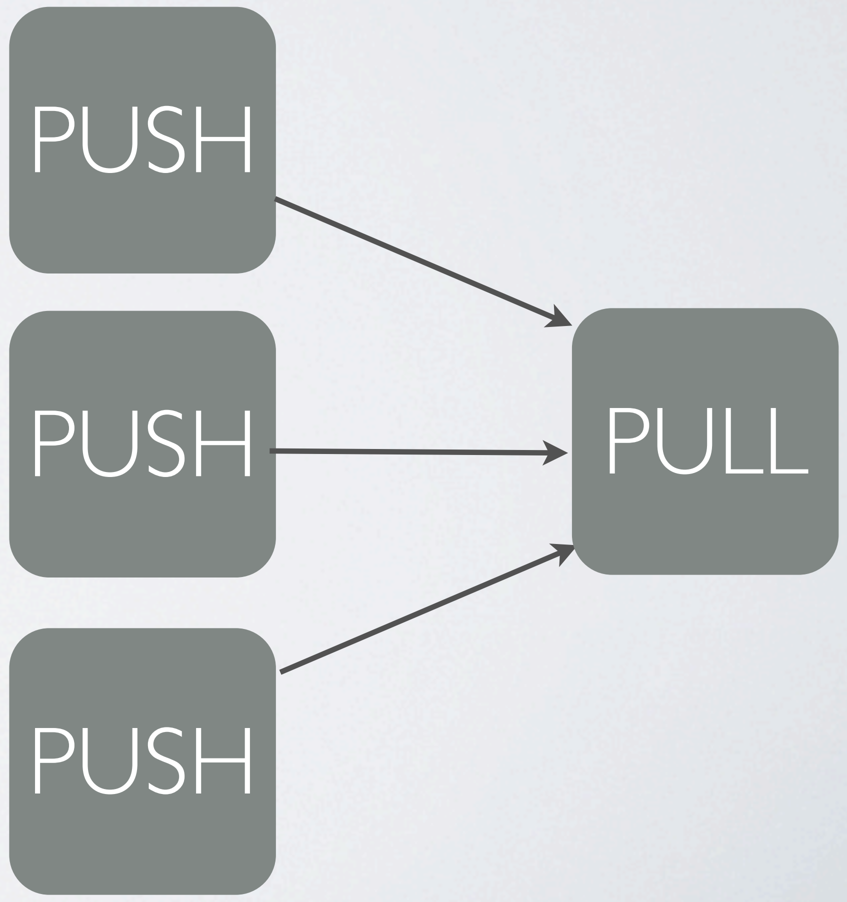
```
def request
  @request_socket.send('request')
  message = @request_socket.recv
end

def reply
  message = @reply_socket.recv
  @reply_socket.send('reply')
end

def router
  @frontend = @context.socket(ZMQ::ROUTER)
  @frontend.bind(front_address)
  @backend = @context.socket(ZMQ::DEALER)
  @backend.bind(back_address)
  ZMQ.proxy(@frontend, @backend)
end
```

Messaging Patterns:

Push => Pull

Asynchronous, one way messaging (fan out / fan in). Messages are distributed by "load-balancing" or "fair-queueing".

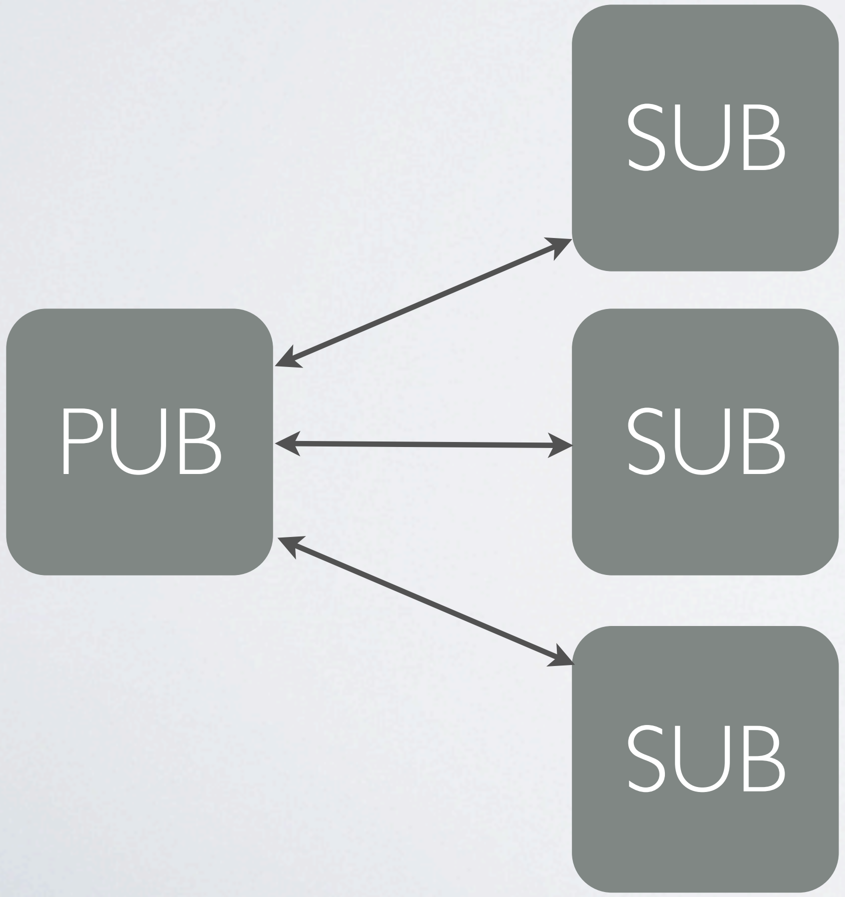


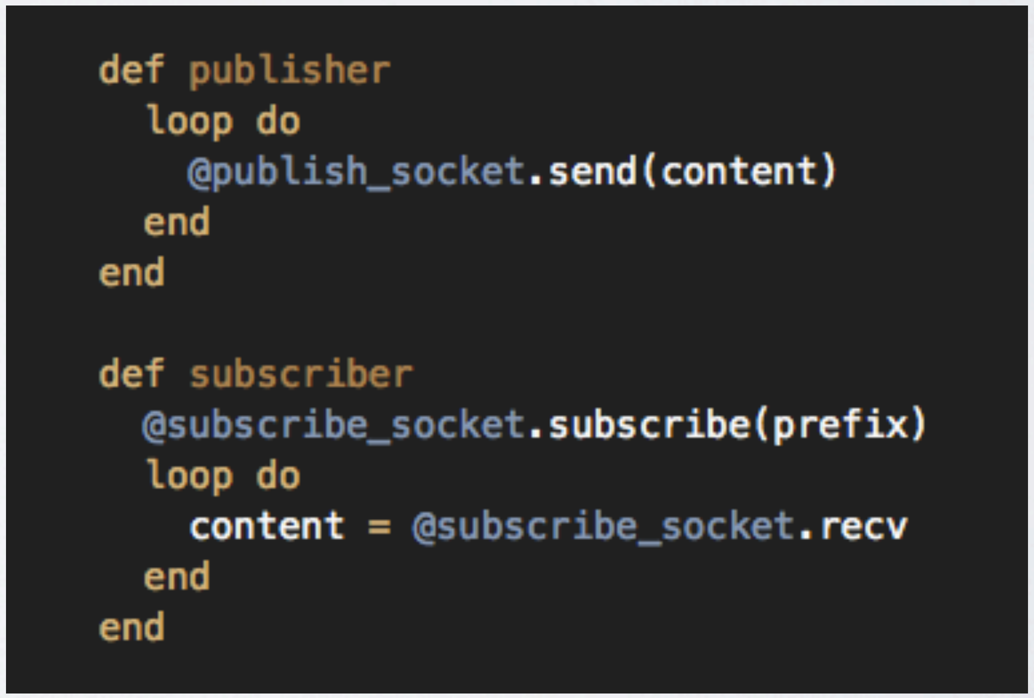


Messaging Patterns:

Publish <=> Subscribe

Asynchronous, one way messaging (pub/sub).
Messages are sent to all subscribers with matching filter.





```
def publisher
  loop do
    @publish_socket.send(content)
  end
end

def subscriber
  @subscribe_socket.subscribe(prefix)
  loop do
    content = @subscribe_socket.recv
  end
end
```

Examples

Run through some of the ZeroMQ Guide examples...

req/rep:
hwserver.rb / hwclient.rb / [mtserver.rb]
rrclient.rb / rrworker.rb / rrbroker.rb

push/pull:
taskvent.rb / taskwork.rb / tasksink.rb

pub/sub:
wuserver.rb / wuclient.rb
psenvpub.rb / psenvsub.rb

More Examples!

Distributed method dispatch...

remote_proxy.rb

Resources

ZeroMQ:
http://www.zeromq.org

"Code Connected Vol 1" (The guide):
http://hintjens.com/blog:30

Ruby Bindings:
https://github.com/methodmissing/rbczmq
http://www.zeromq.org/bindings:ruby-ffi

Examples:
https://github.com/mattconnolly/zguide-rbczmq