Going brokerless: The transition from Qpid to 0mq

Paul Mathews, Systems Architect
EIG/Bluehost

OpenStack Summit, November 2013

RPC Messaging

- One major limiting factor we encountered when scaling OpenStack
- Many services depend upon a reliable messaging system
  - Compute
  - Neutron
  - Conductor
  - Celiometer
  - VNC
  - Cells

Why Qpid?

- Used by Redhat
- Clustering
  - Offered the “possibility” of horizontal scaling
  - Removed in 0.18

Qpid experience

- Single instance was not reliable
  - Unable to scale
  - Single point of failure
- Compute connections to broker are lost
  - Restart of compute service required
- Problematic due to missed messages

RabbitMQ?

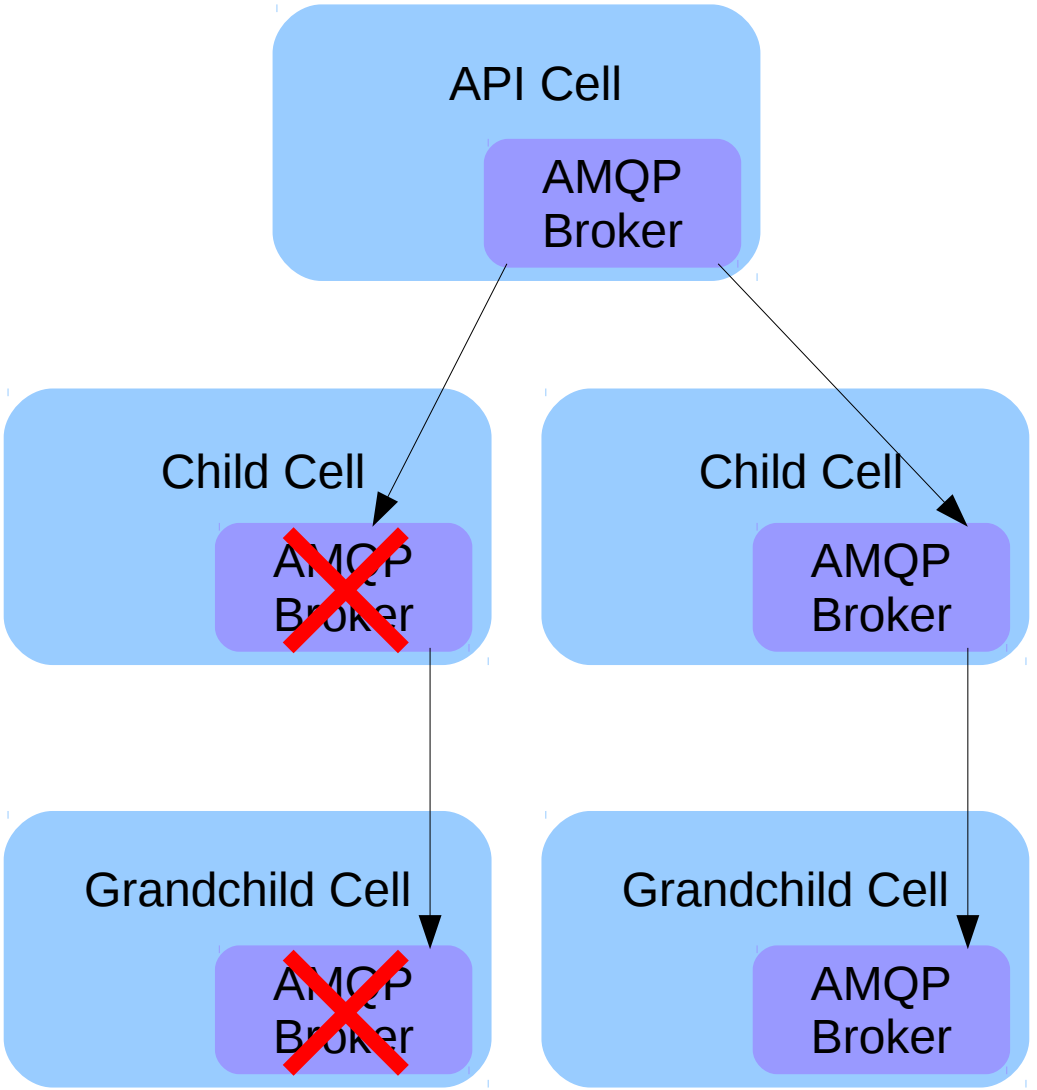
- Similar design as Qpid
- Broker model has the same drawbacks
- Problematic experiences from other users

Possible solutions for scaling broker

- Cells
- Clustering/HA

Cells

- Cells do not address performance issues
- Cells lower the load on individual brokers
- Magnify problems when they occur by chaining services



Clustering/HA

- Qpid
  - Clustering is slow, and unreliable
  - Node sync causes problems
  - New HA module is active/passive
- RabbitMQ
  - Does have an active/active (HA) mode
  - Complicated setup, many moving pieces
- Scaling a broker is not practical
  - At best, minimal gains with addition of nodes
  - Loss of nodes causes further issues

No more brokers!!!

- Brokers are a single point of failure
- Not horizontally scalable
- Reduced reliability with addition of nodes
- HA provides minimal benefit, and adds complexity

Requirements for messaging

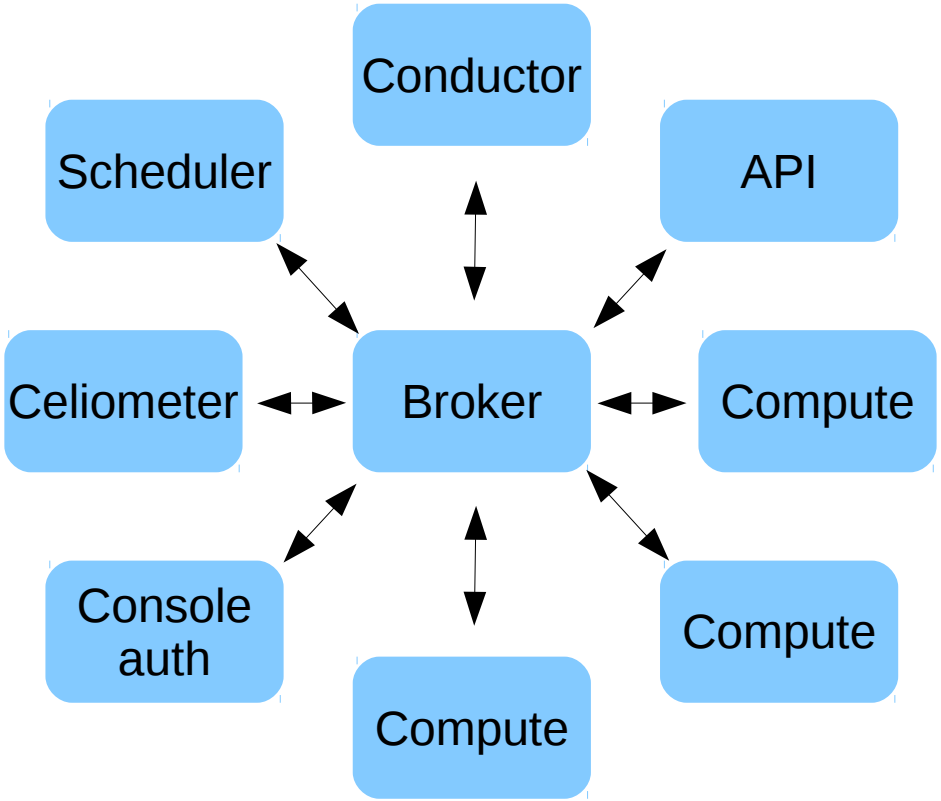
- No single point of failure
- Horizontally scalable
- Reliable at scale

- No more centralized broker
  - Receiver on each node
  - Routing handled by matchmaker

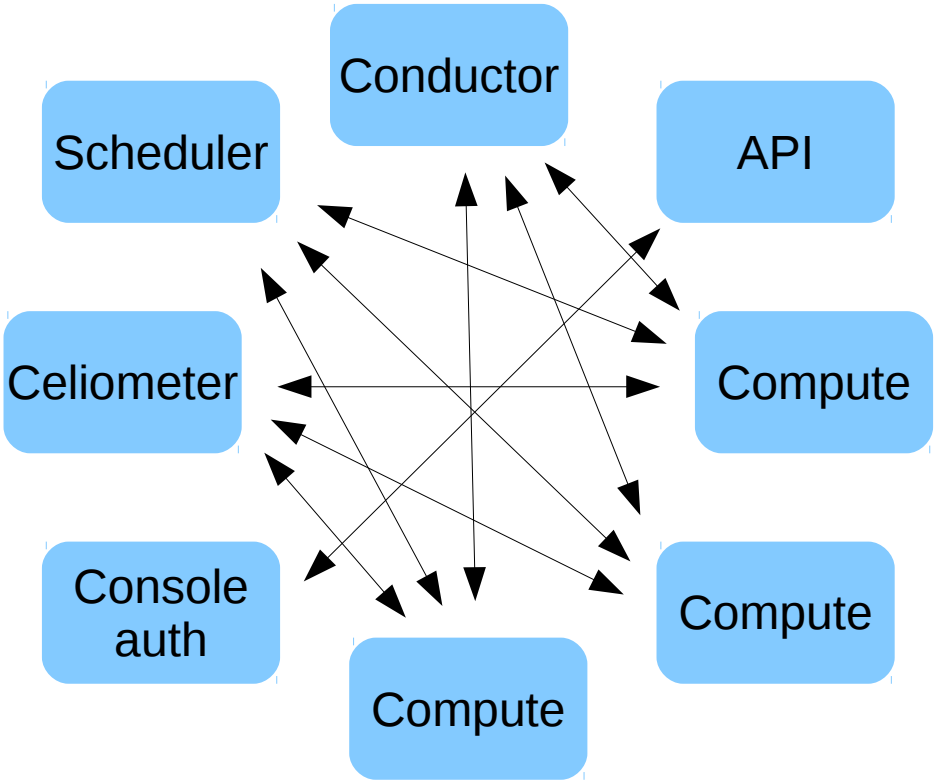
```
{
    "scheduler": [ "sched1", "sched2" ],
    "consoleauth": [ "cauth1", "cauth2" ]
}
```

Messaging topologies

Brokers are limited to a star topology



ZeroMQ is a partially-connected mesh



Flexibility

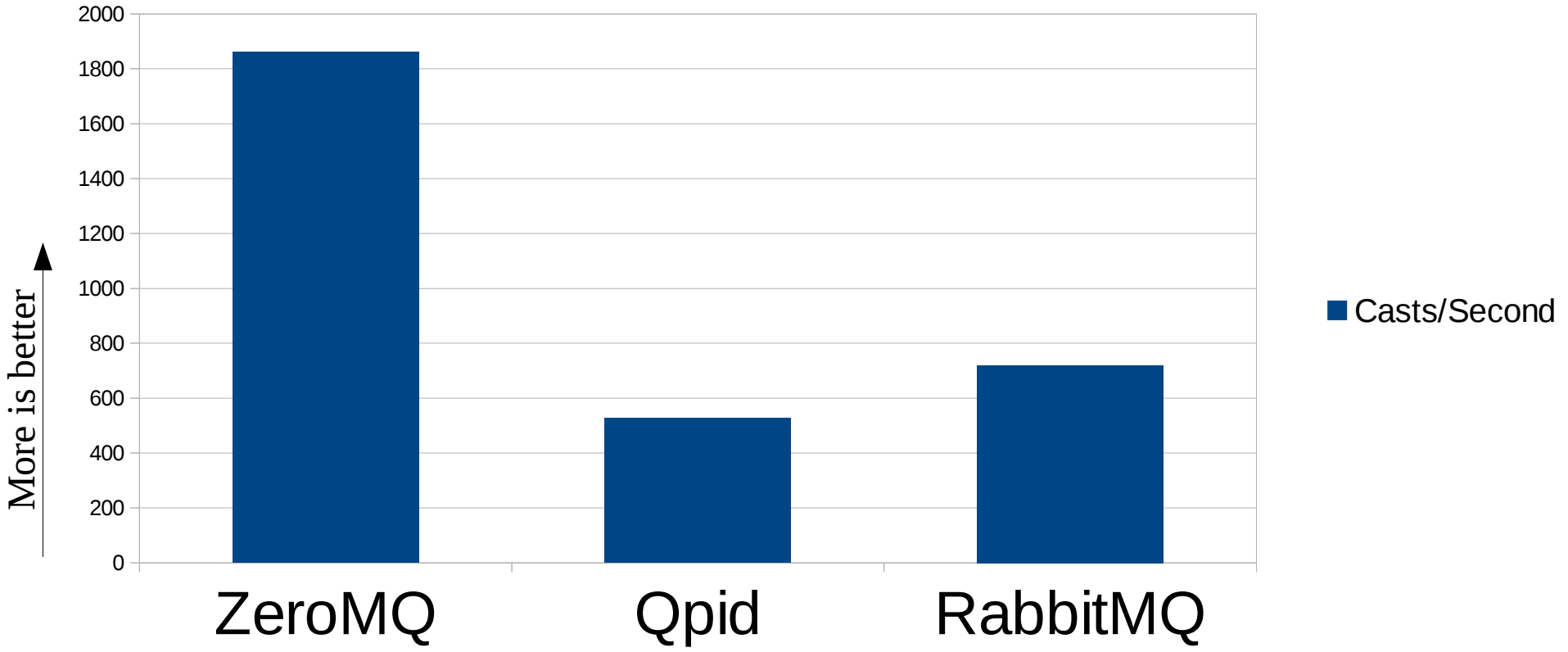
- Brokers have a rigidly defined structure
  - Queues, exchanges, subscriptions, fanouts
- ZeroMQ has four simple methods
  - Connect, bind, send, recv
- ZeroMQ lets us define our own messaging models

Lightweight messaging

- ZeroMQ uses simple socket connections
  - Low resource utilization
- FAST

RPC Cast Performance

(on a single-core VM)



ZeroMQ Configuration

- Edit nova.conf to use zmq

```
rpc_backend = nova.openstack.common.rpc.impl_zmq
rpc_zmq_matchmaker = nova.openstack.common.rpc.matchmaker.MatchMakerRing
matchmaker_ringfile = /etc/nova/matchmaker.json
rpc_zmq_ipc_dir = /var/run/zeromq
```

- Configure matchmaker file

```
{
    "scheduler": [ "sched1", "sched2" ],
    "consoleauth": [ "cauth1", "cauth2" ]
}
```

- Start zmq-receiver

RPC Migration

- You can't get there from here!
  - No easy way to move between messaging systems
  - No logical divisions
- Only one backend allowed
  - All or nothing switch

We need a new solution

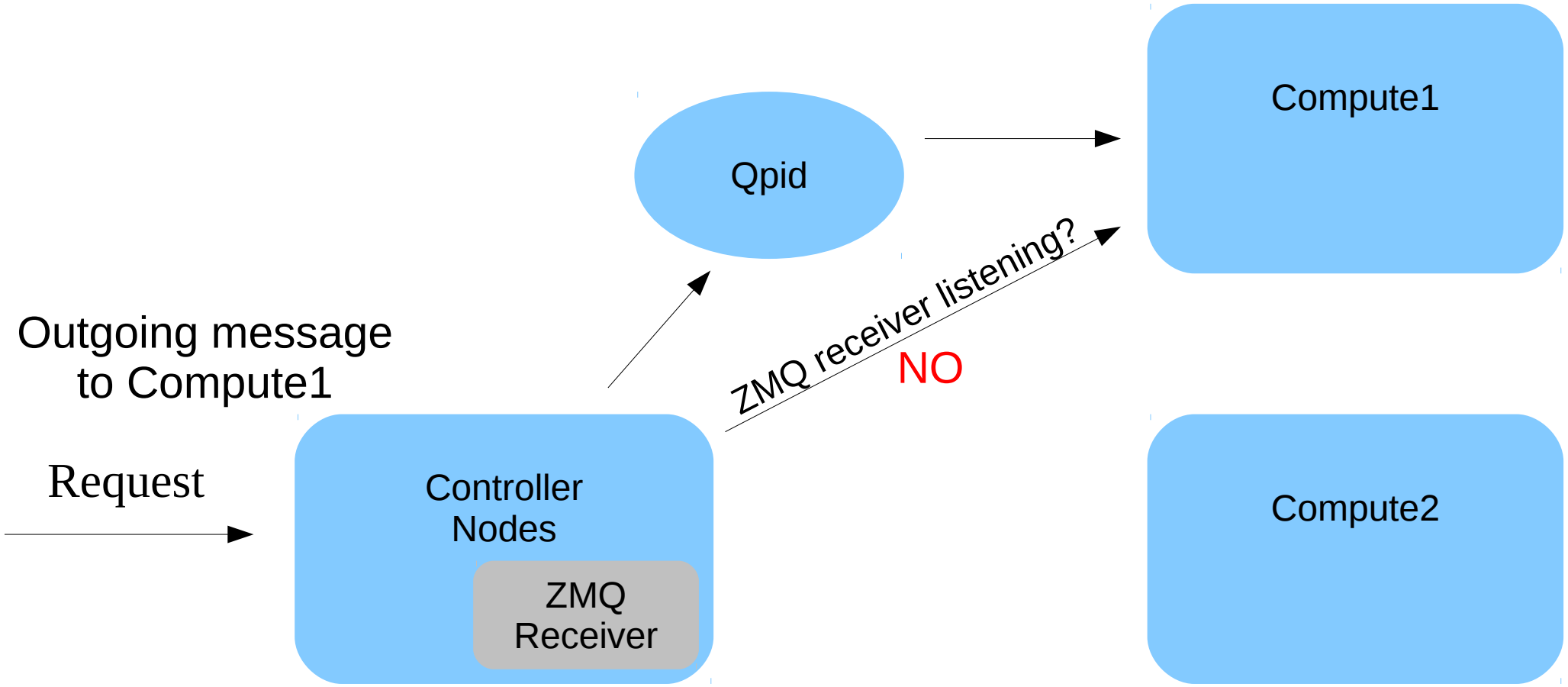
- Moving between messaging systems is painful
  - Prior strategy will not work
- Tens of thousands of nodes to migrate
- Need to migrate with little or no downtime
- Rollout must allow deployment to individual servers

Dual messaging backends

- Nodes use both Qpid and ZeroMQ messaging backends concurrently
- Code can be rolled out without affecting behavior, and enabled later
  - Change config, and start the ZeroMQ receiver
- Once dual backends are enabled, ZeroMQ is attempted first, then fails over to Qpid.

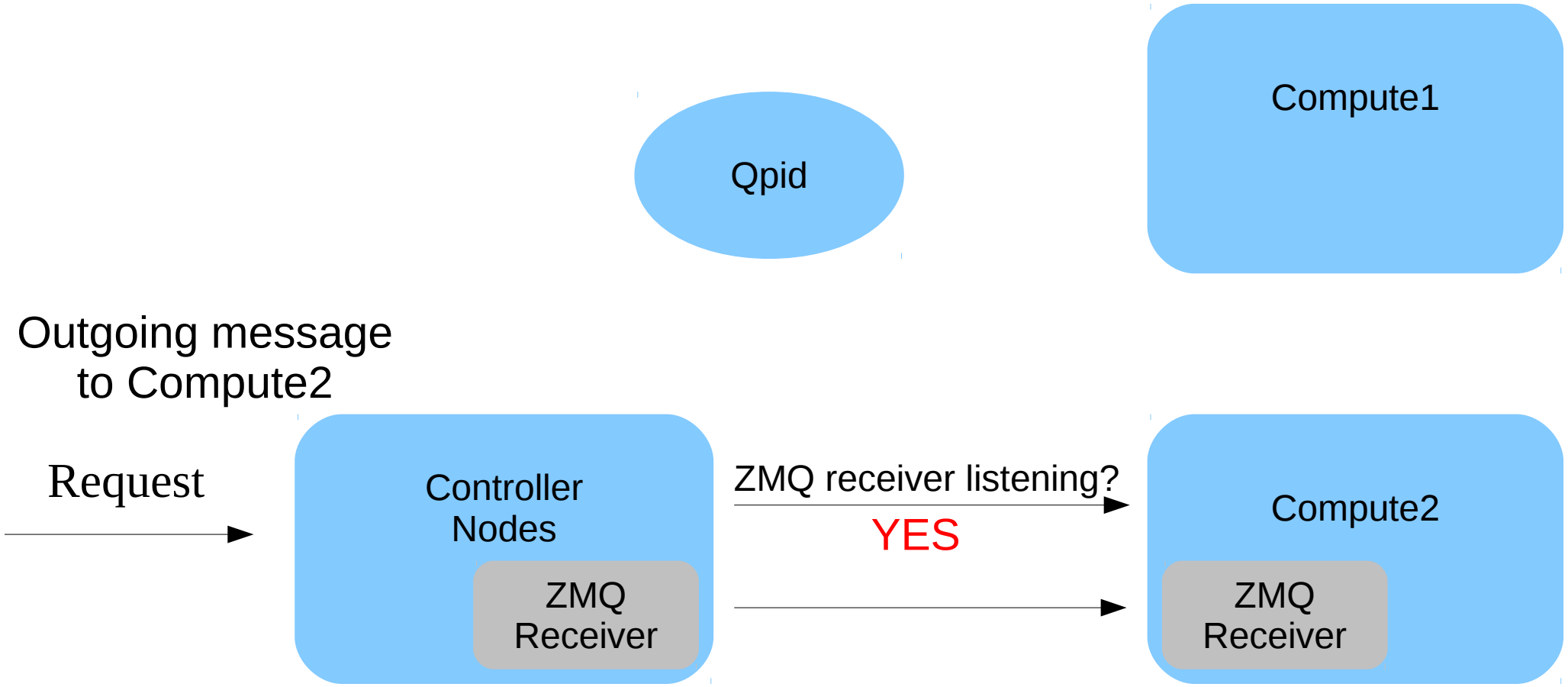
Dual message backends

1. Deploy config to controller nodes



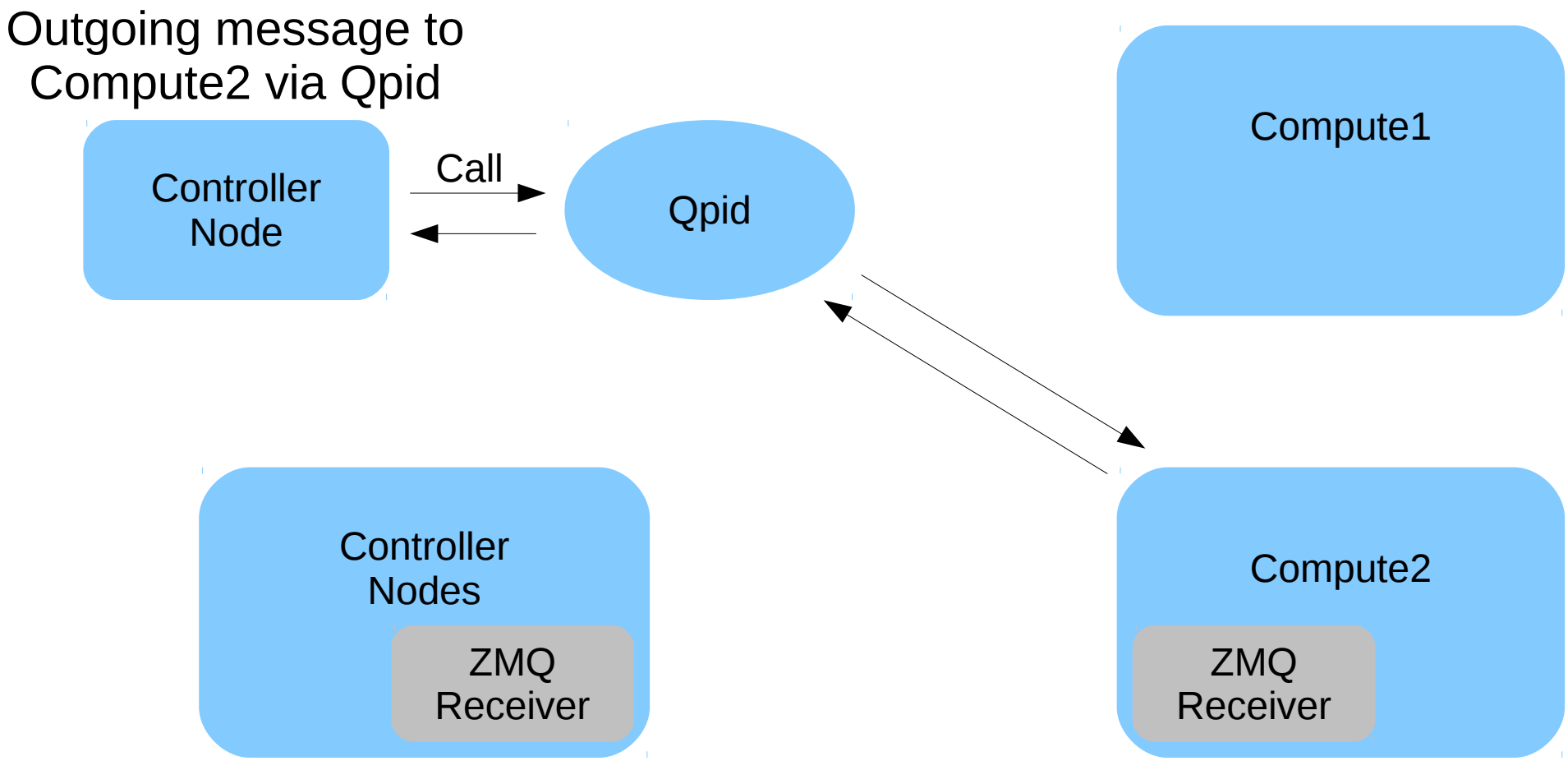
Dual message backends

1. Deploy config to controller nodes
2. Deploy config to compute nodes



Dual message backends

1. Deploy config to controller nodes
2. Deploy config to compute nodes



Configuring dual backends

- Change rpc_backend to impl_zmq, but retain qpid_hostname setting

```
qpid_hostname = qpid1
rpc_backend = nova.openstack.common.rpc.impl_zmq
rpc_zmq_matchmaker = nova.openstack.common.rpc.matchmaker.MatchMakerRing
matchmaker_ringfile = /etc/nova/matchmaker.json
rpc_zmq_ipc_dir = /var/run/zeromq
```

- Nodes will leverage the qpid_hostname value and connect to Qpid, but will attempt delivery via ZeroMQ first
- Once switched, nodes will accept incoming messages from either Qpid or ZeroMQ

Migrating to ZeroMQ

- Dual backend code meant minimal downtime
- Migration was smooth, without unexpected losses in messaging
- Connection checks to the ZeroMQ receiver do not seem to cause undue stress to nodes

ZeroMQ in production

- More reliable than a broker
- Faster than a broker
- Solves scalability issues

Lingering issues

- Occasionally nova-compute stops processing queued messages

Dual backend code

- https://github.com/paulmathews/nova

Questions?