NATS

A nervous system for modern distributed systems

Derek Collison

@derekcollison

https://github.com/derekcollison

derek@apcera.com

derek.collison@gmail.com

Why Even Listen to Me?

Derek Collison

Google 6yrs

TIBCO > 10yrs

Architected TIBCO **Rendezvous** and **EMS**

Architected the OpenPaaS **CloudFoundry**

Building **Messaging** Systems

and **Solutions** > 20yrs

Why Messaging?

Background

- MicroServices Architectures
- Event-Driven Architectures
- HTTP as an interface only goes so far
- 1:N / 1:1 of N Patterns
- Cascading Request/Reply
- Subject/Topic based routing

a brief **Network Recap**

Networks

- IP: TCP and UDP
- Streaming vs limited packet size and unreliability
- Effective 1:N -> UDP Broadcast / Multicast
- Late 90s TCP becomes only fast-path option

Networks

- Multicast has too much admin, failed
- Multicast trunked or disallowed
- UDP BC TOR trunked in most Cloud Platforms

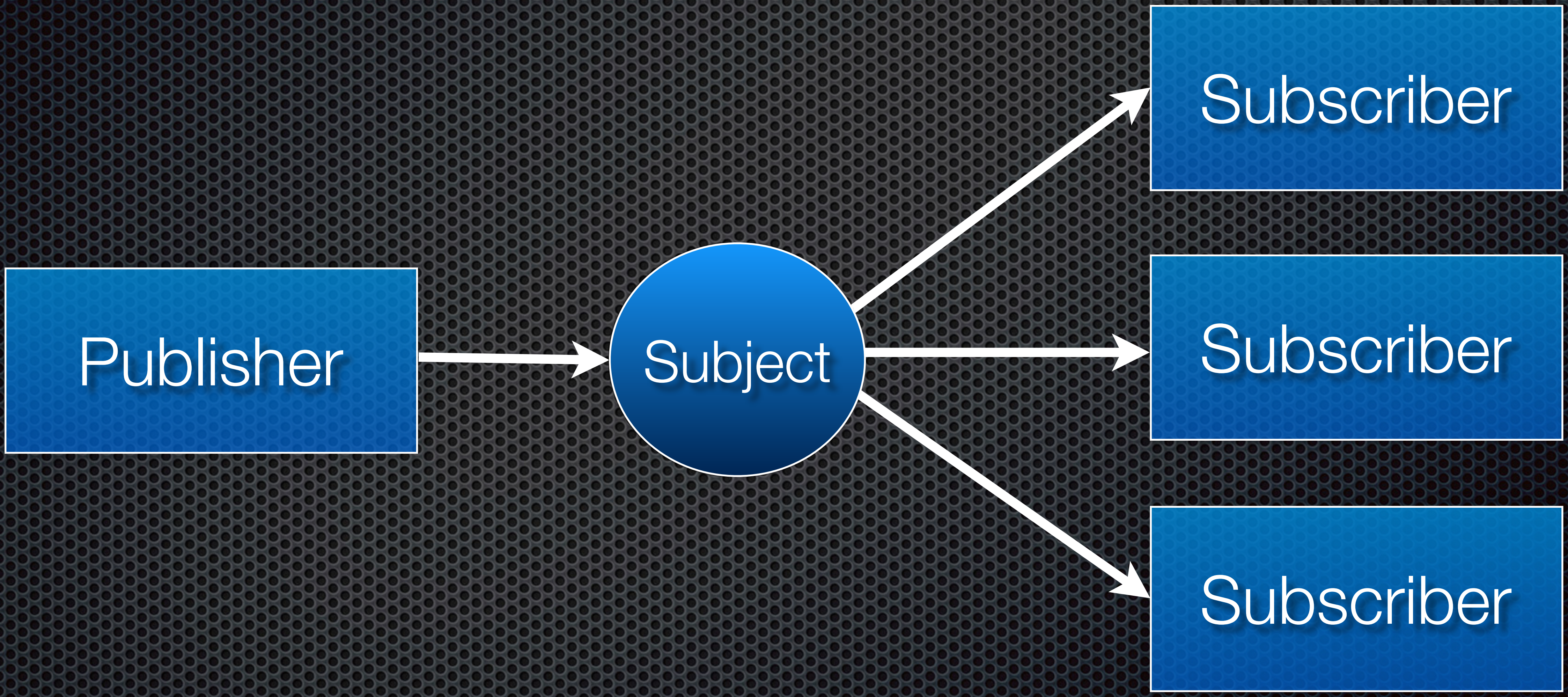
Messaging

Basic Messaging Patterns

- ✓ Publish-Subscribe
- ✓ Queuing
- ✓ Request-Reply

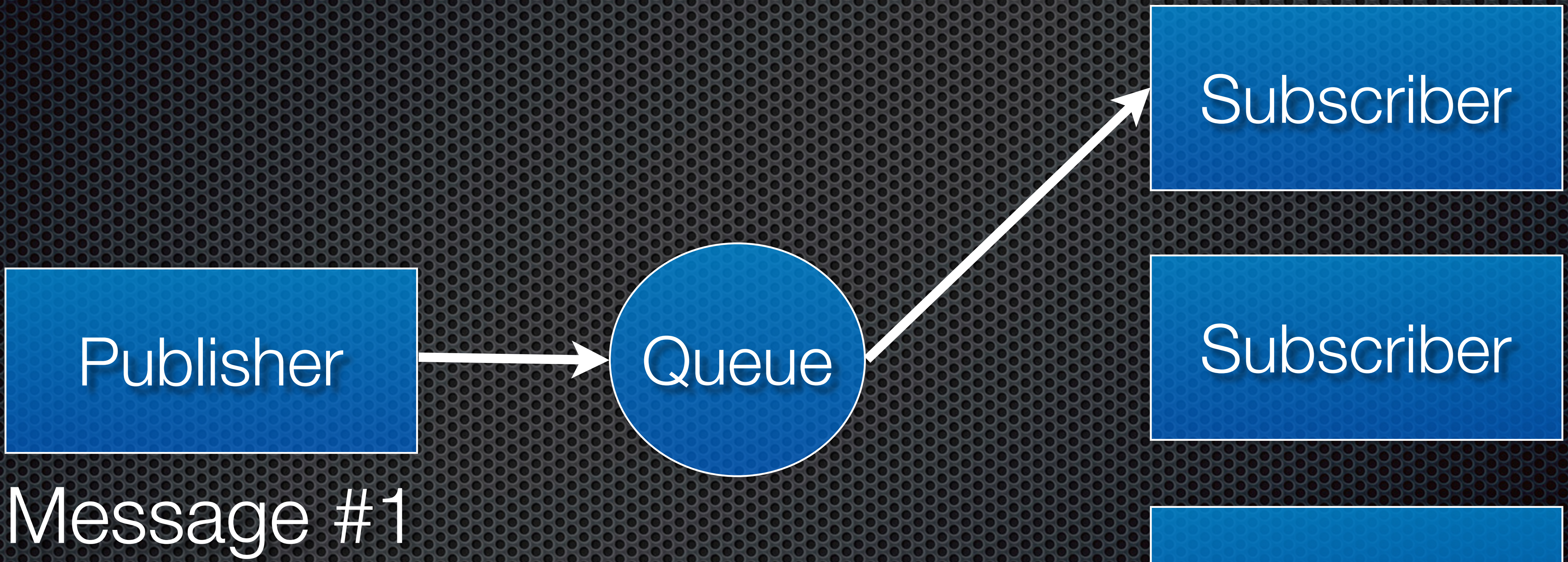
Messaging - Publish Subscribe

1 : N



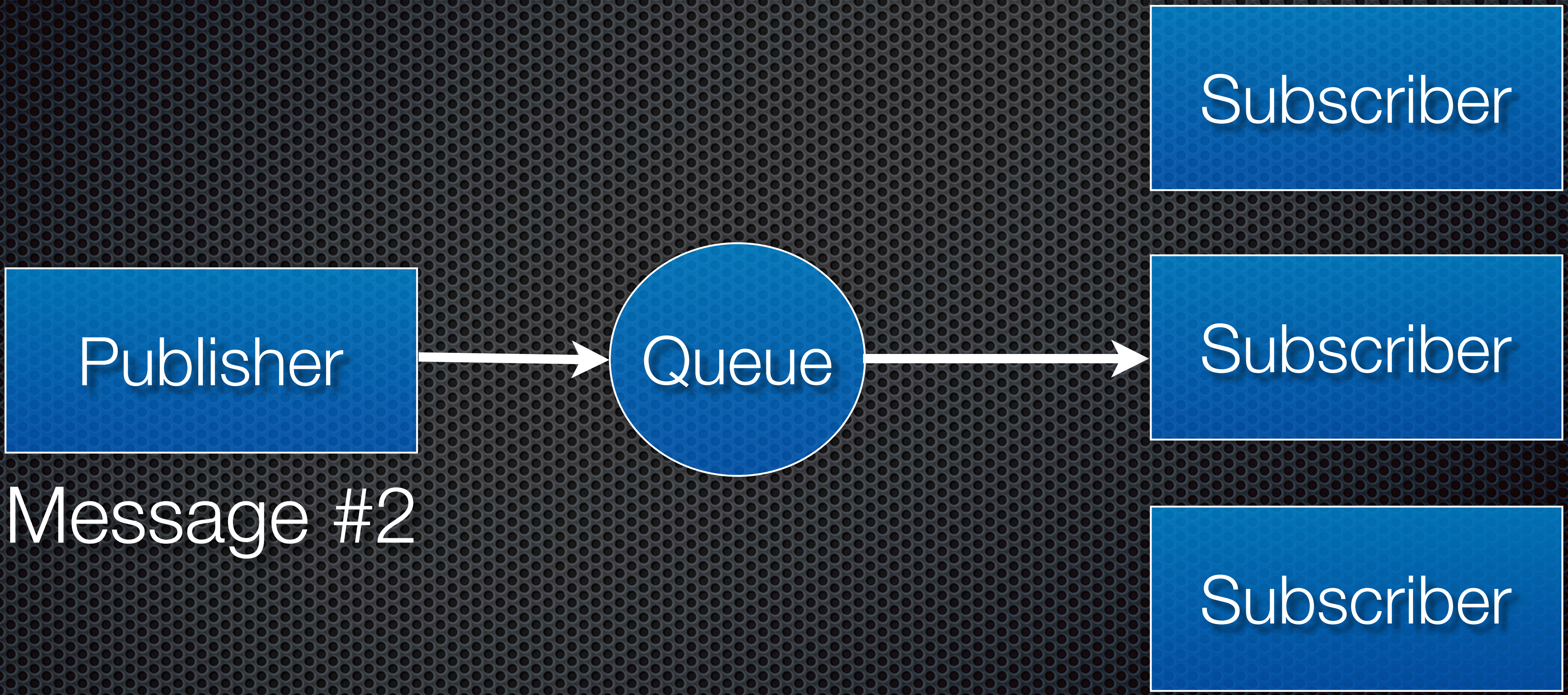
Messaging - Queuing

1 : 1



Messaging - Queuing

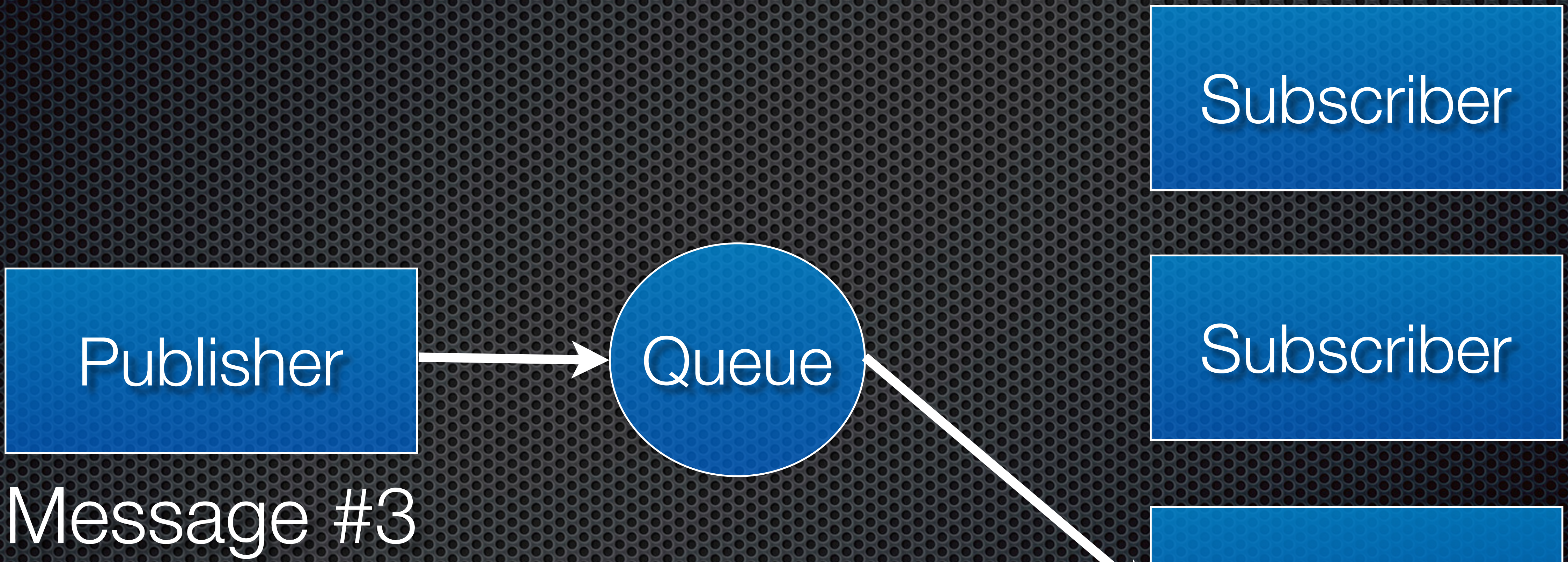
1 : 1



Message #2

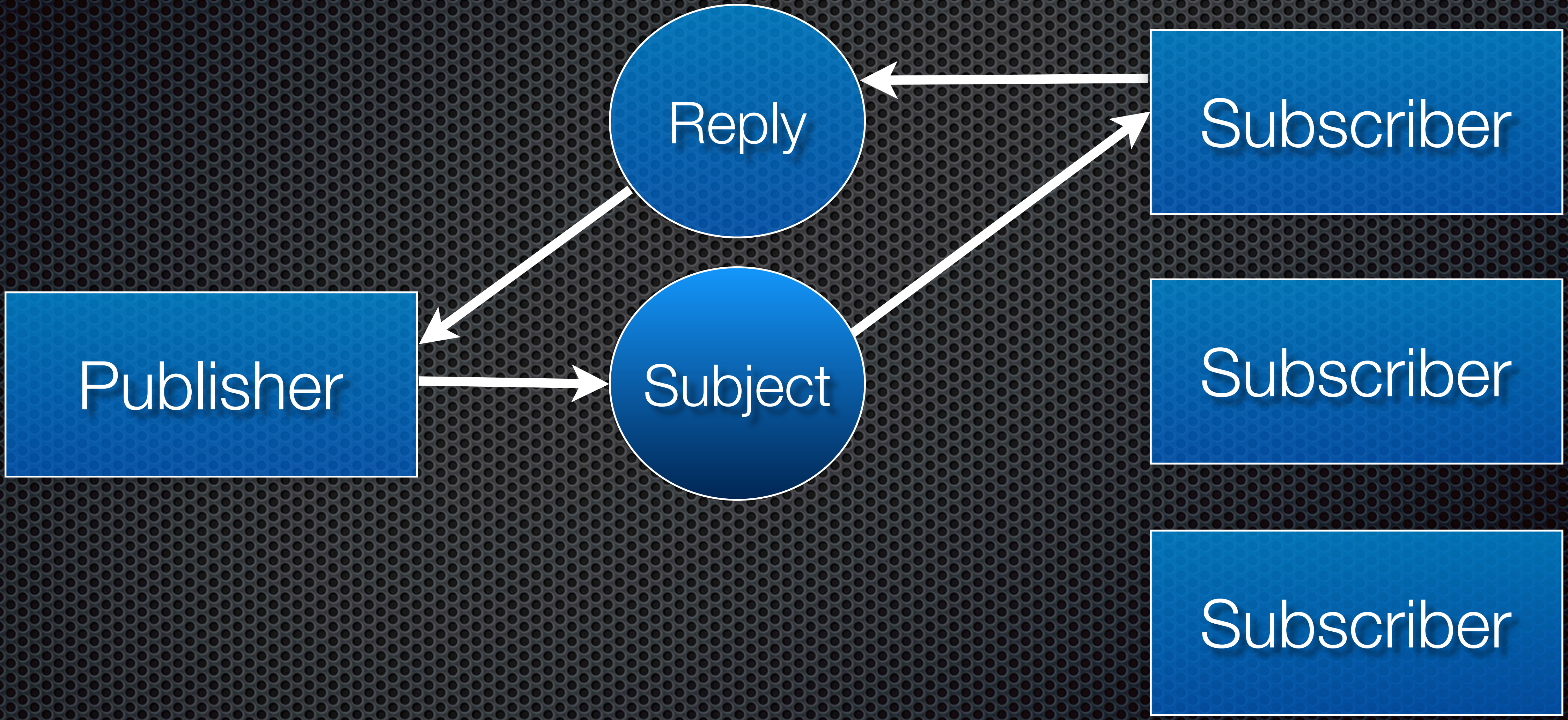
Messaging - Queuing

1 : 1



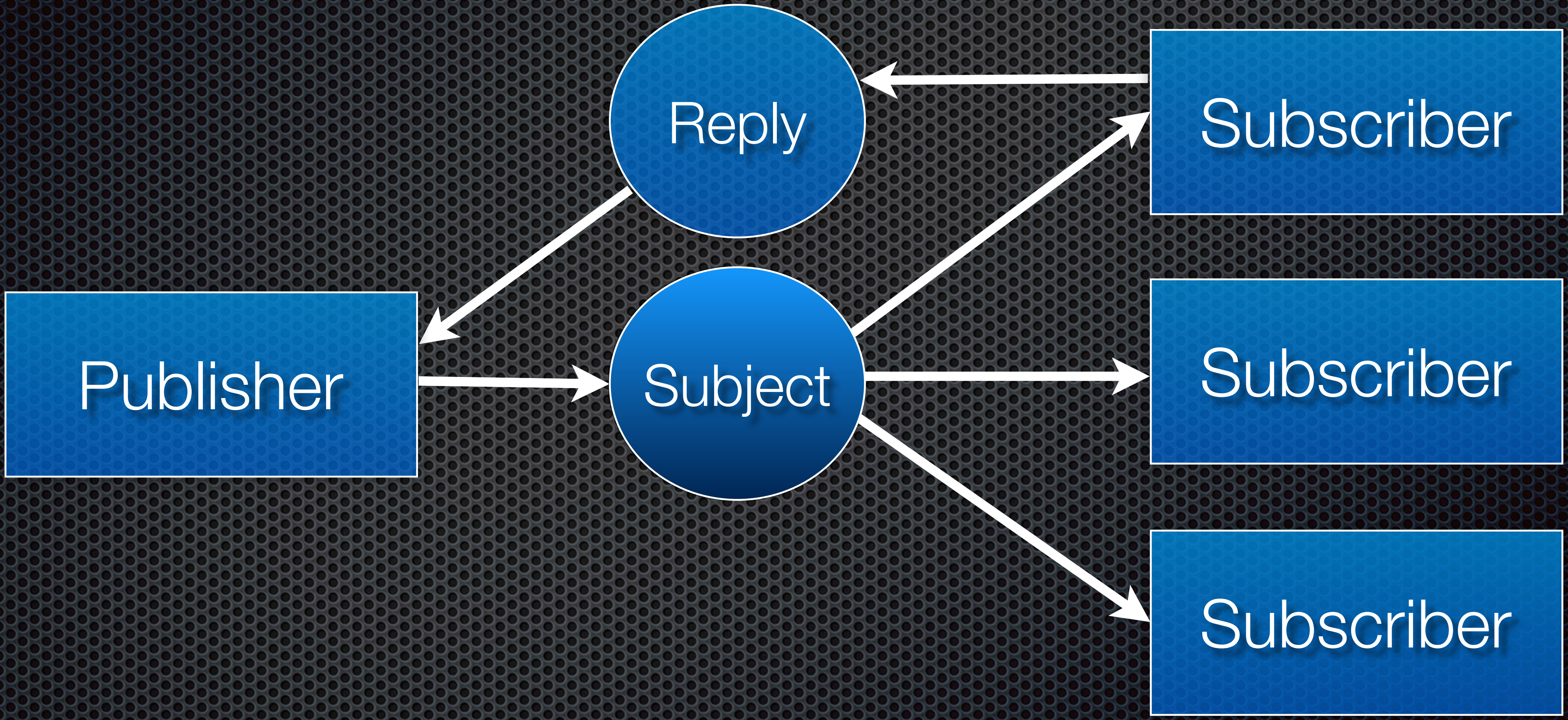
Messaging - Request Reply

1 : 1



Messaging - Request Reply

1 : N



Messaging Use Cases

- ✓ Addressing, discovery
- ✓ Command and control - Control Plane
- ✓ Load-balancing
- ✓ N-way scalability
- ✓ Location Transparency
- ✓ Fault-Tolerance

Why Pub-Sub?

Publish-Subscribe

- ✓A radio vs a phone call
- ✓E.g. Wallstreet quote distribution
  - ✓programatic trading
  - ✓fairness and delivery embargo
- ✓**Don’t assume the Audience!**

Queueing

Queueing

Publish or Subscribe operation?

Queueing

Publish is Store and Forward

Queueing

Subscribe is distributed queueing

Request-Reply

Request-Reply

- ✓ Don’t assume audience!
- ✓ How many responders?
- ✓ Always built on Publish-Subscribe

Enterprise Messaging Patterns

- ✓ Persistence
- ✓ Store & Forward
- ✓ Distributed Transactions
- ✓ Enhanced Delivery Models

Delivery

Delivery Models

- ✓At Most Once
- ✓At Least Once
- ✓Exactly Once

Delivery Models

Exactly Once is very **HARD***!*

If you do it
Correctly

What if we looked at the problem differently?

Should it do everything?

OR..

Should
it do
much less?

NATS

nats.io

the Inspiration

What is NATS?

What NATS is..

- ✓High-Performance
- ✓Always on and available
- ✓Extremely light-weight
- ✓Fire and Forget - At Most Once
- ✓Pub/Sub
- ✓Distributed Queues
- ✓Request/Reply

What is NATS **NOT**?

What NATS is NOT**..**

- ✓Enterprise Messaging System
- ✓Persistence
- ✓Transactions
- ✓Enhanced Delivery Models
- ✓Queueing Product

Disclaimer!

I built NATS for myself!

What’s
Unique?

What is **Unique?**

- ✓Clustered mode server
- ✓Cluster aware clients
  - ✓Go, Node.js, Java, Scala, Python, Ruby
- ✓**Auto-pruning** of interest graph
- ✓Always Pub/Sub, **NO** Assumptions
- ✓Distributed queueing across clusters
- ✓Text-based protocol

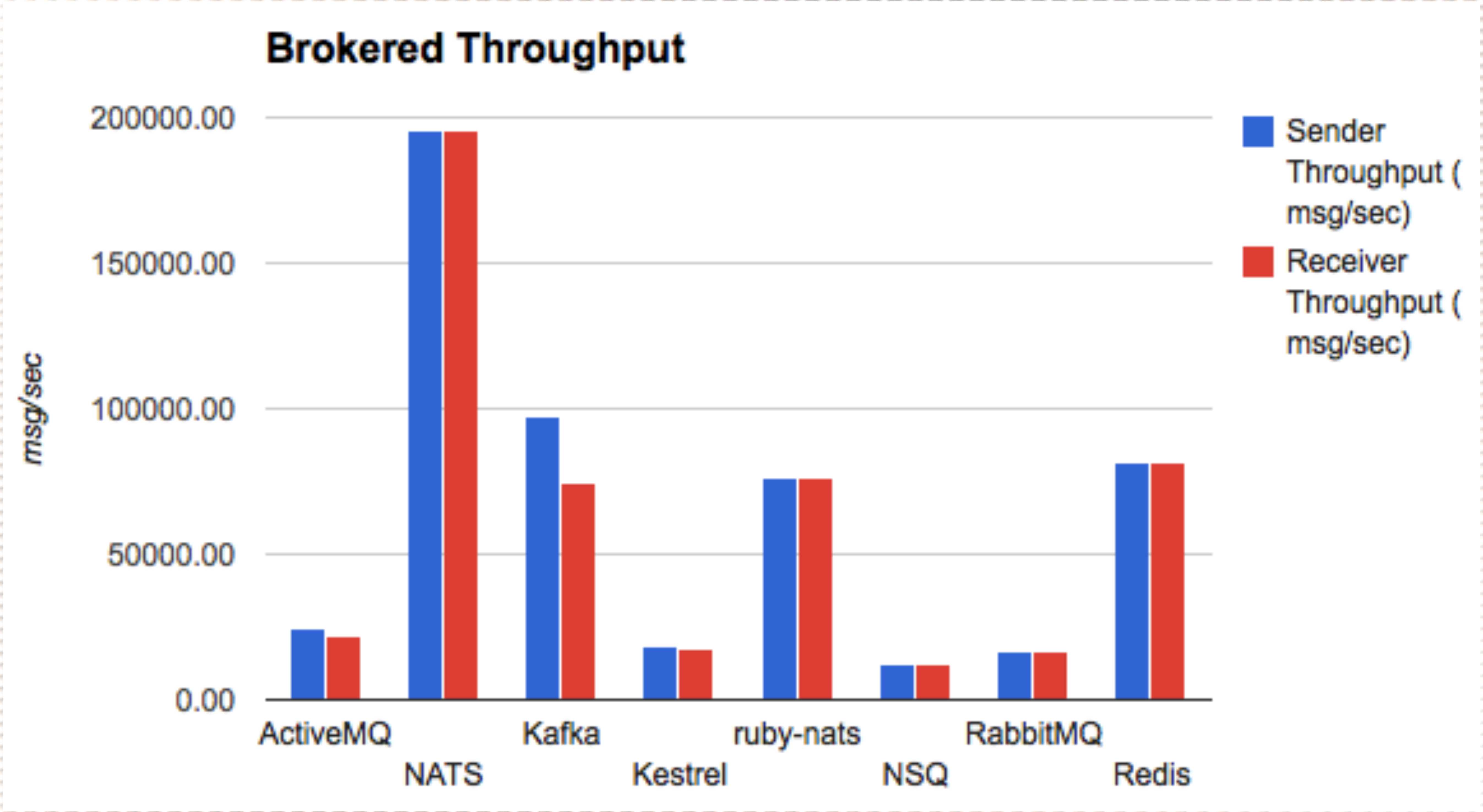
Performance

Performance

- Originally written to support CloudFoundry
- In use by CloudFoundry, HTC, Baidu, Apcera and others
- Written first in Ruby -> 150k msgs/sec
- Rewritten at Apcera in Go (Client and Server)
- First pass -> 500k msgs/sec
- **Current Performance** -> **5-6m msgs/sec**

Performance 4k payloads

Courtesy - http://www.bravenewgeek.com/dissecting-message-queues/



Demo

More Info

slideshare.net/derekcollison/gophercon-2014

Text-Based?

Text-Based Protocol

- ✓ Easy to get started with new clients
- ✓ Does not affect performance
- ✓ Can telnet directly to server

Demo

```
telnet demo.nats.io 4222
```

Monitoring

Monitoring

- ✓HTTP based monitoring
- ✓Modeled off of /varz in Google
- ✓Simple JSON payloads

Demo

```
curl demo.nats.io:8222/varz
curl demo.nats.io:8222/connz
```

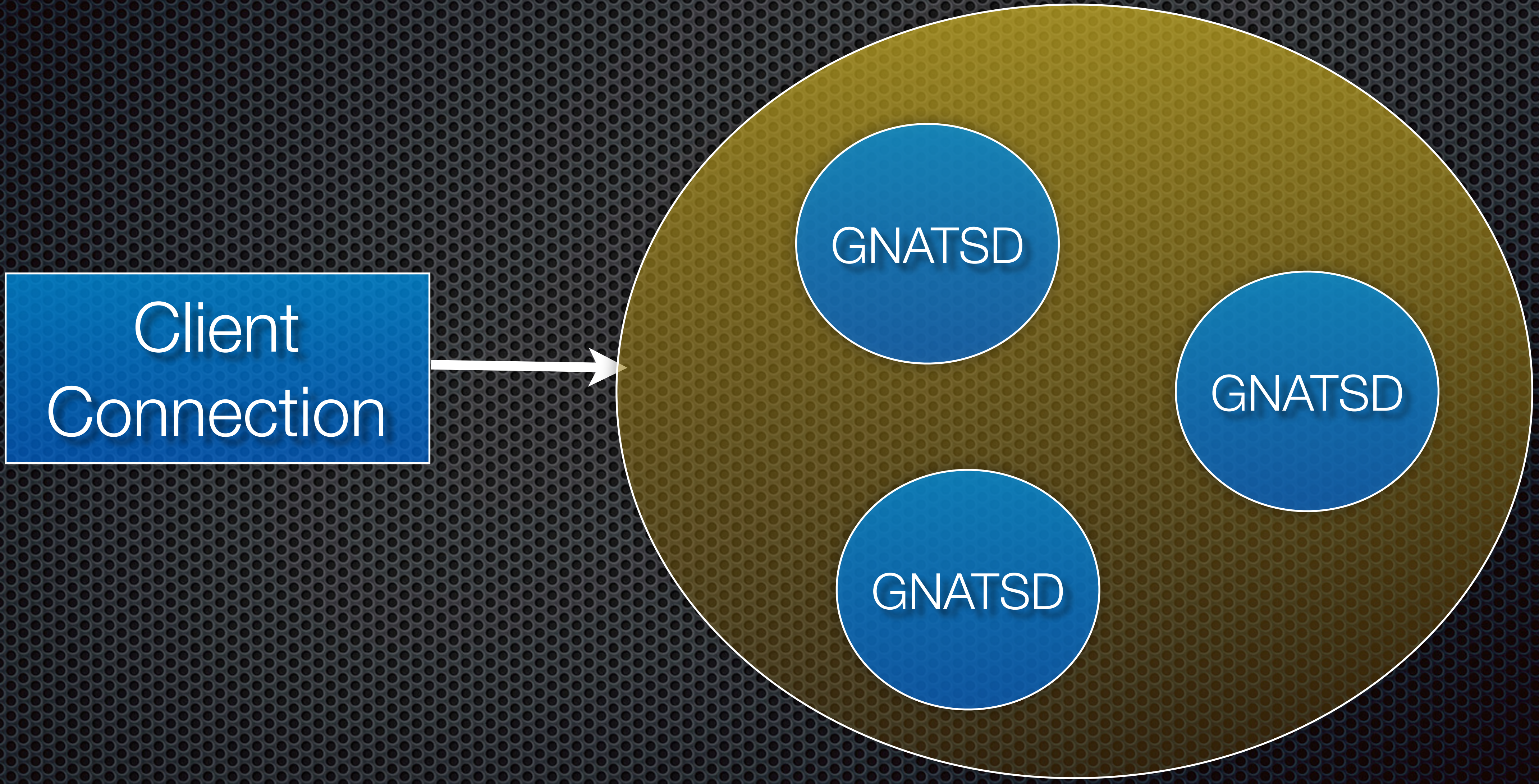
Clients

Clients

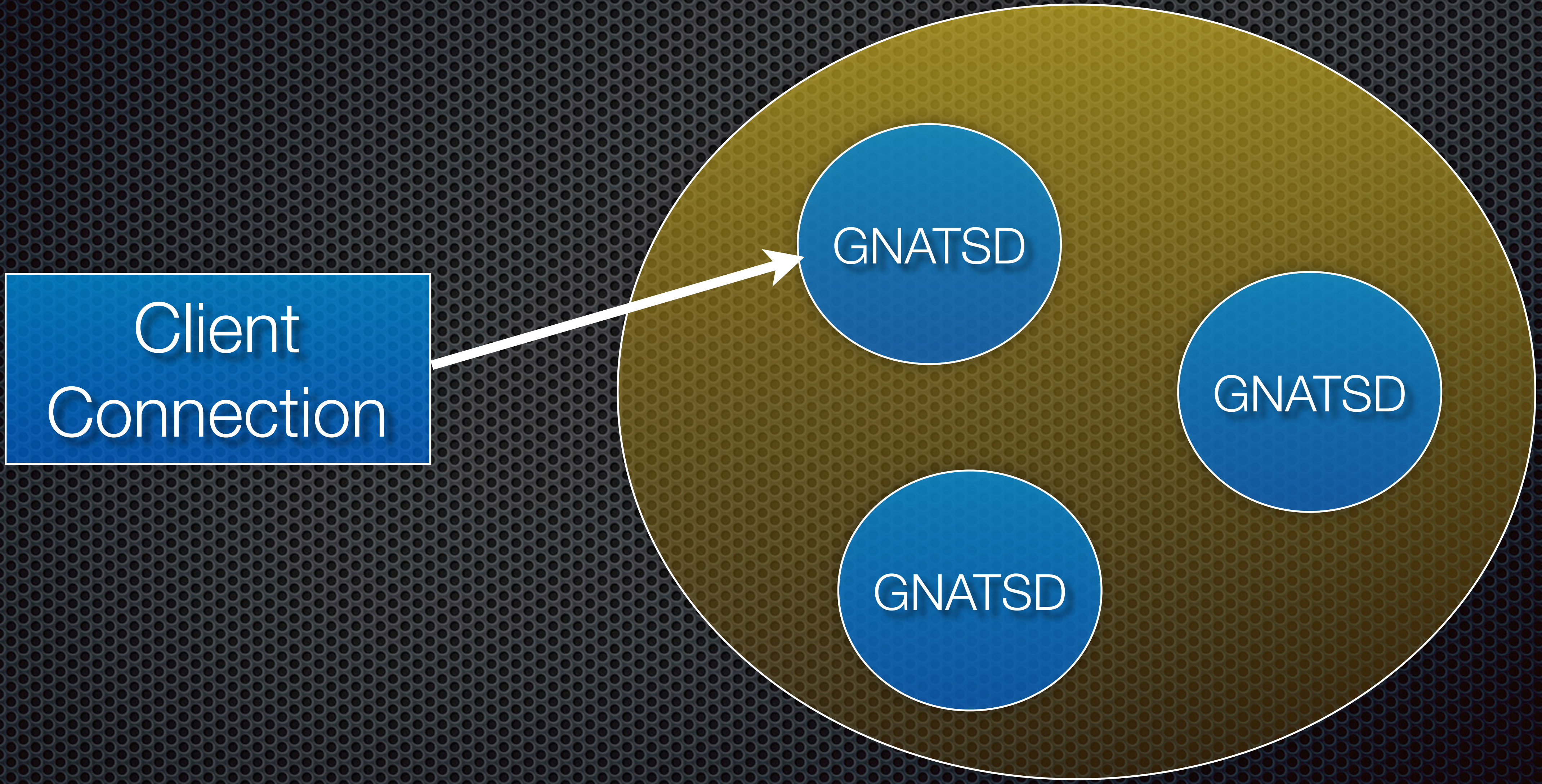
- ✓ Go
- ✓ Node.js
- ✓ Java/Scala
- ✓ Ruby
- ✓ Python

Clustered

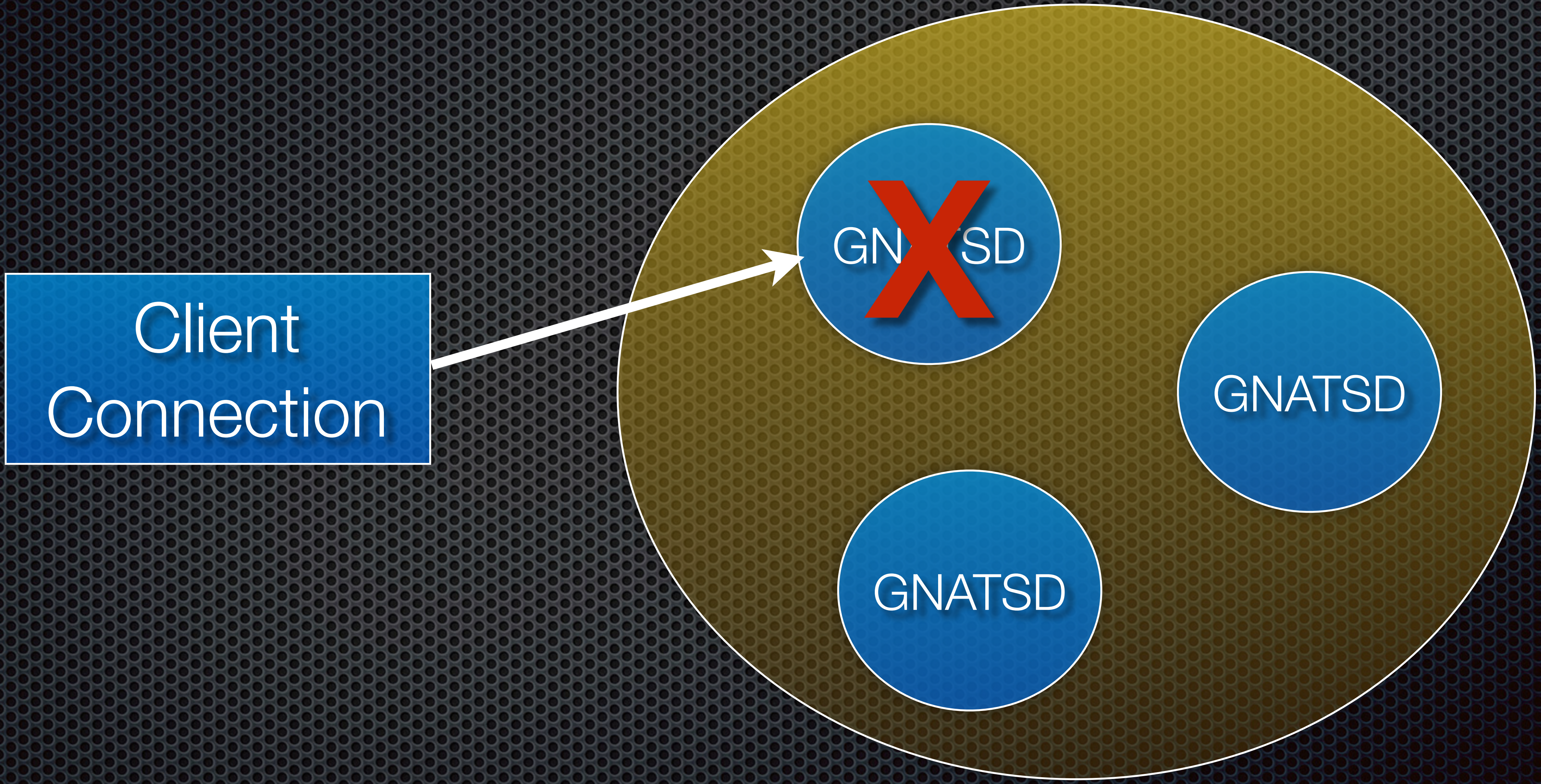
Clustering



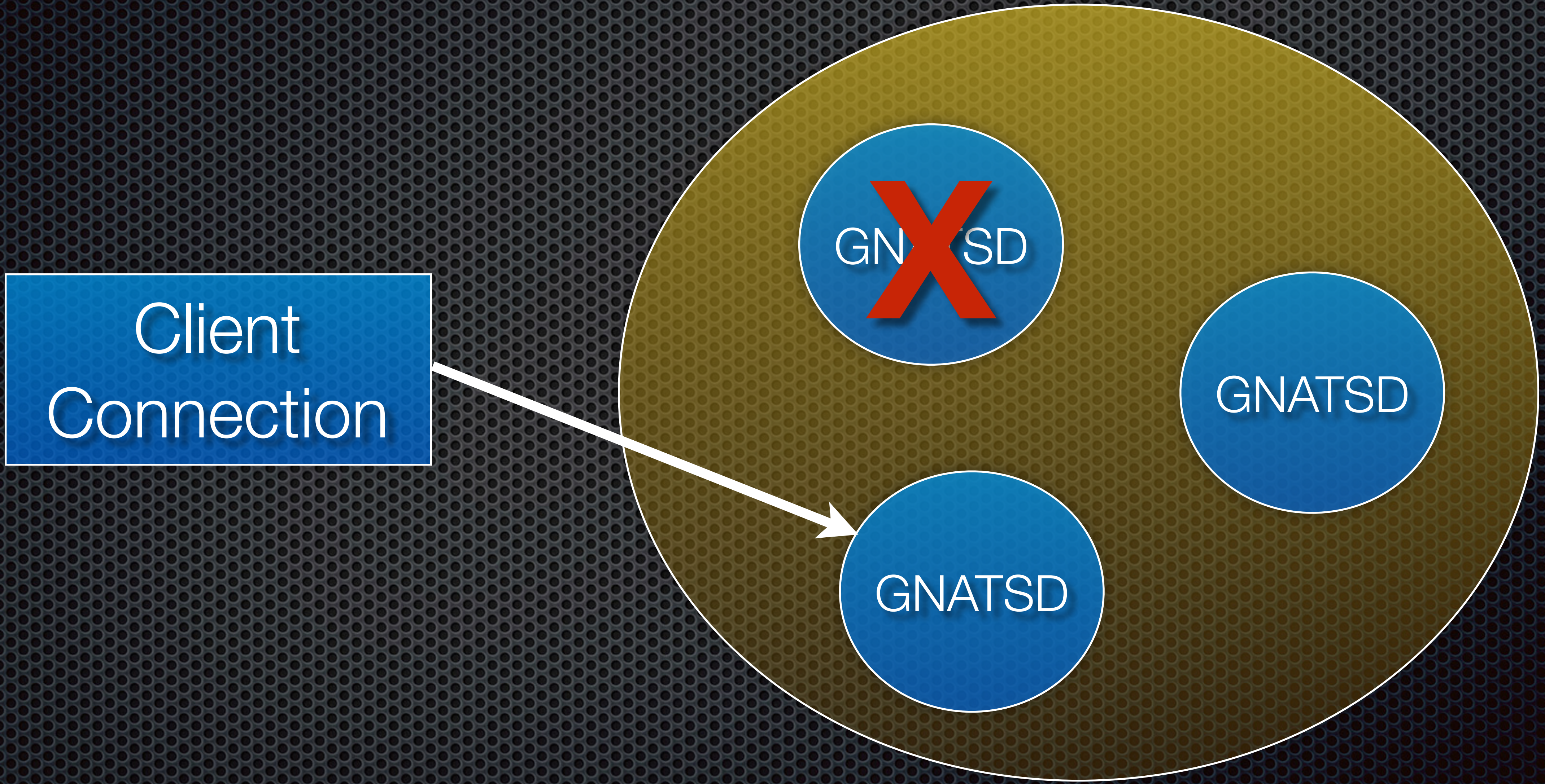
Clustering



Clustering



Clustering



Auto-Pruning

Big **DEAL**!

(to me)

Why?

1:1 of
large **N**
(think Google)

Auto-Pruning

- ✓Able to express limited interest a priori
- ✓Systems uses circuit breakers
- ✓1:1 Requests to large N is very efficient!
- ✓Easily accessible in protocols
- ✓All clients support in Request/Reply

Summary

Summary

- ✓ Modeled to be always-on dial-tone
- ✓ Always available - NATS protects itself
- ✓ High-Performance server
- ✓ Clustered Servers / Cluster aware Clients
- ✓ Clients in many languages, contribute!

Futures

Futures

- ✓NGINX C++ client to OSS
- ✓Performance gains in server and clients
- ✓C/C++, LUA clients
- ✓Monitoring dashboards
- ✓Auto-configuration service

Thanks!

Resources

https://nats.io

https://registry.hub.docker.com/u/apcera/gnatsd/

https://github.com/apcera/gnatsd

http://www.slideshare.net/derekcollison/gophercon-2014

Questions?