Picking a Message Queue.

Vladislav Kirshtein

Software engineer/consultant

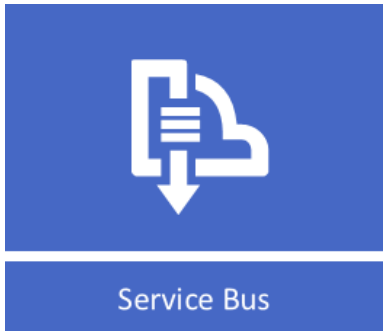
vladk@codevalue.net

OzCode

Your Road to Magical Debugging

CodeValue

Inspiring Code, Creating Value.


Service Bus


IronMQ


stormmq


ActiveMQ


kafka


Apollo


Qpid


RabbitMQ


beanstalkd


ZAQAR
an OpenStack Community Project


WebSphere MQ


NATS

Amazon SQS


HornetQ


ØMQ


ISO 9001
SGS

About me

- Senior consultant at CodeValue since 2015
- Experienced in large scale software development
- 10 years of software engineering experience
- Particularly interested in highload projects, distributed systems, and cloud computing

Agenda

- Message oriented protocols
- AMQP - RabbitMQ
- ZeroMQ
- Cloud
- NATS

Protocols

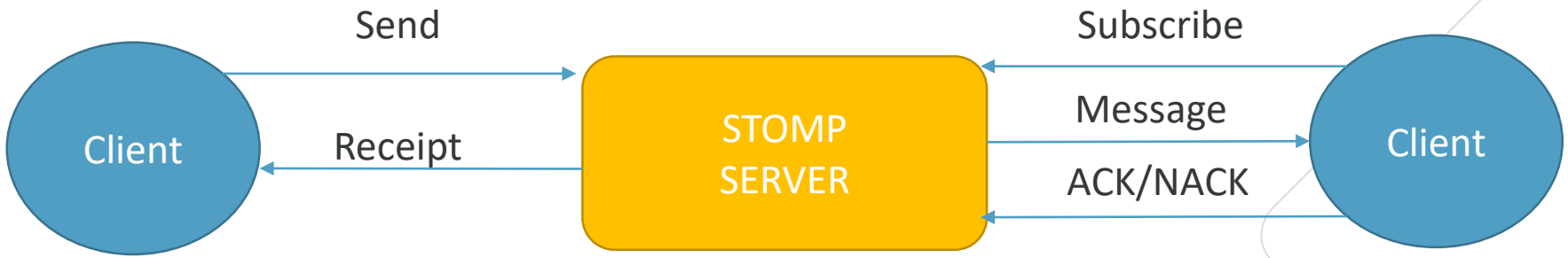
- Stomp – Simple (or Streaming) Text Oriented Message Protocol
- MQTT – Message Queue Telemetry Transport
- AMQT – Advanced Message Queuing Protocol
- Vendor specific (Kafka,NATS,ZeroMQ etc)

STOMP – Simple Text Oriented Message Protocol

- Simple interoperable protocol for asynchronous messaging
  - coming from HTTP school of design
- Text Based
- Communication between client and server is through a 'frame'
- Client frames : SEND,SUBSCRIBE/ UNSUBSCRIBE, ACK/NACK…
- Server frames: MESSAGE,RECEIPT,ERROR

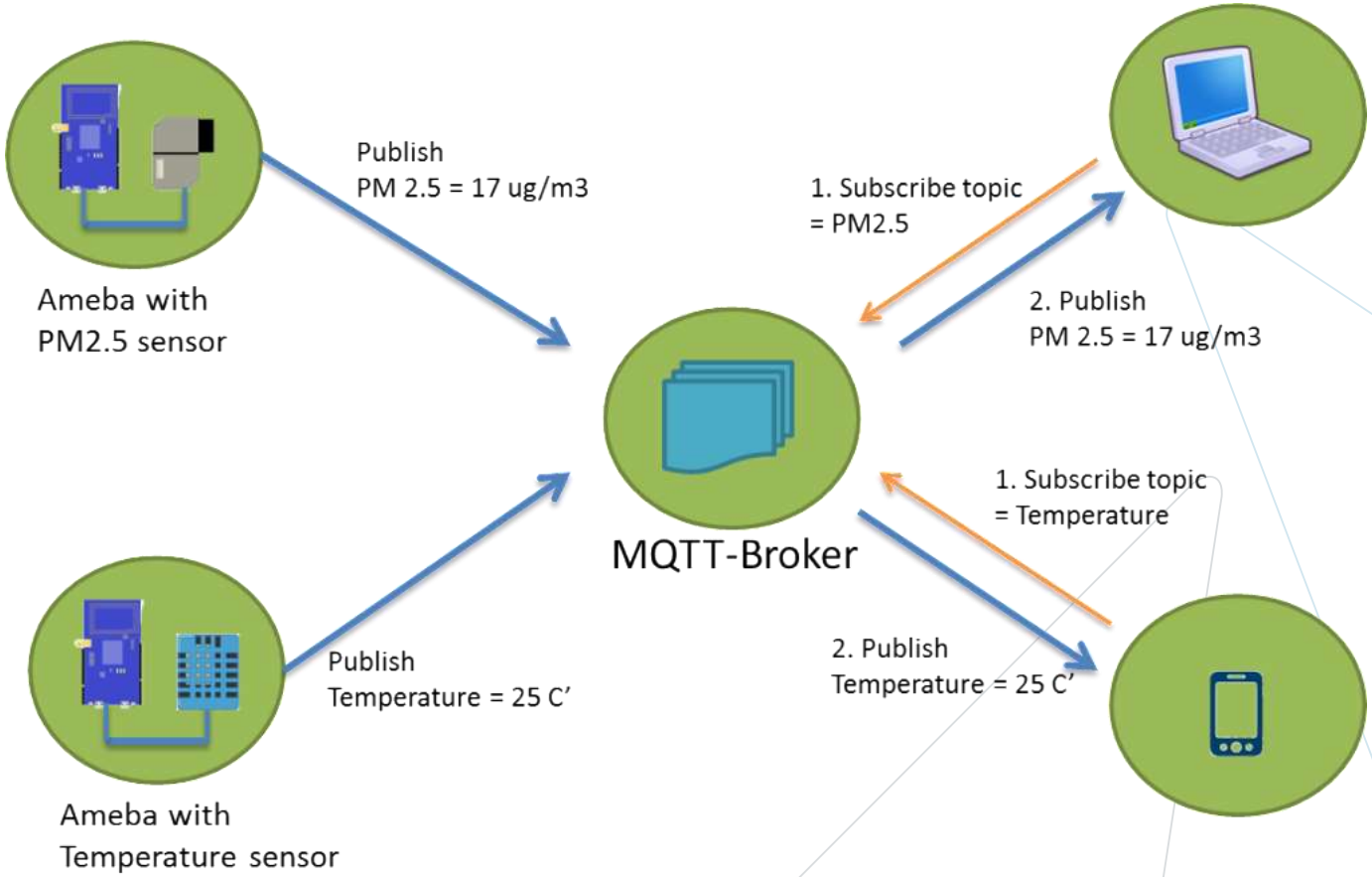
```
SEND
destination:/queue/a
receipt:message-12345

hello queue a^@
```



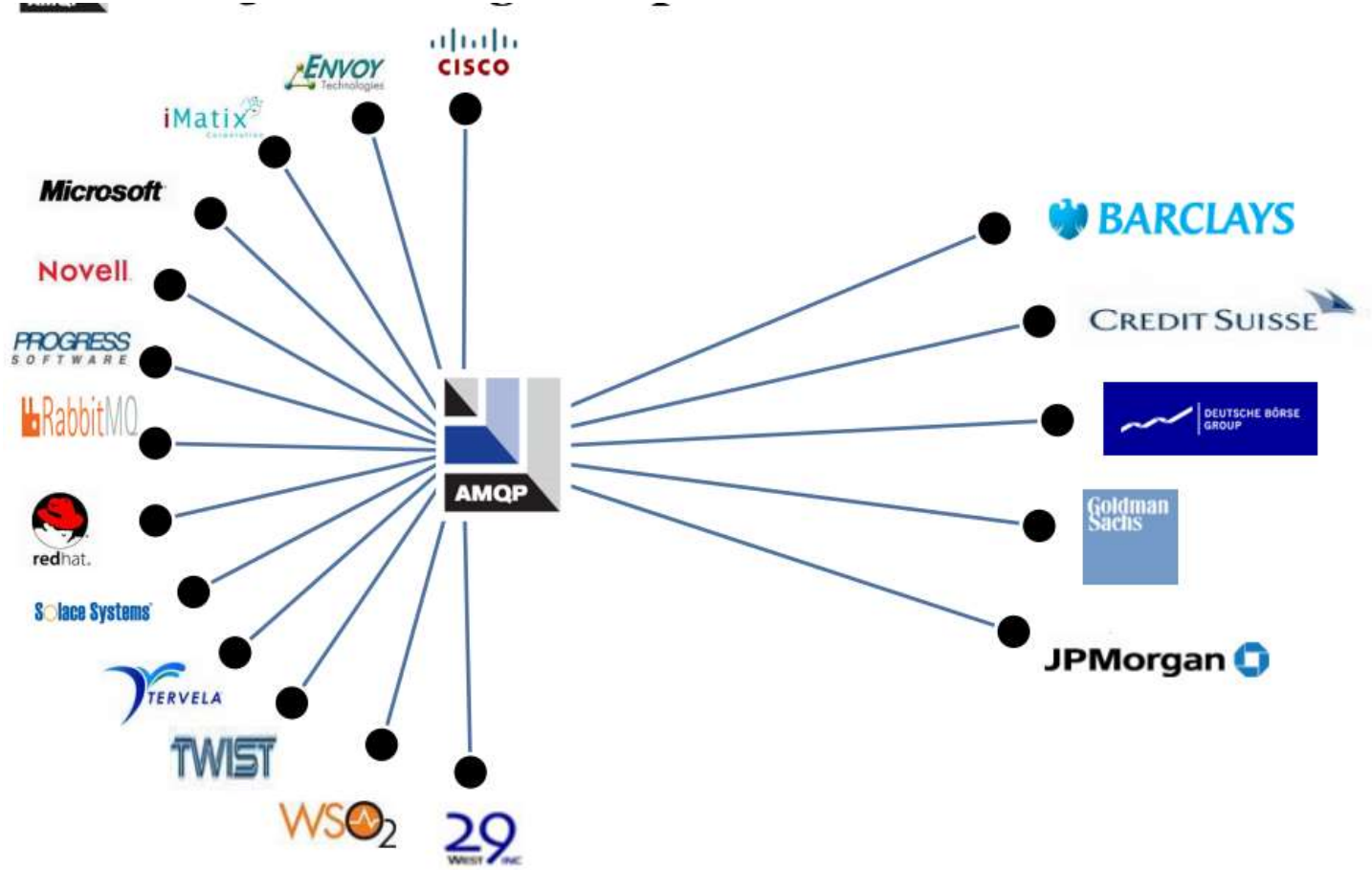
MQTT – Message Queue Telemetry Transport

- Created by IBM for embedded devices telemetry
- Ideal for constrained networks (low bandwidth, high latency, data limits, and fragile connections)
- Binary format (2-byte header)
- Pub/Sub
- Reliable – QoS for reliability on unreliable networks
- Security
  - Authentication
  - TLS/SSL
- Simple – 43 page spec.
- Connect/Disconnect + Publish + Subscribe/Unsubscribe

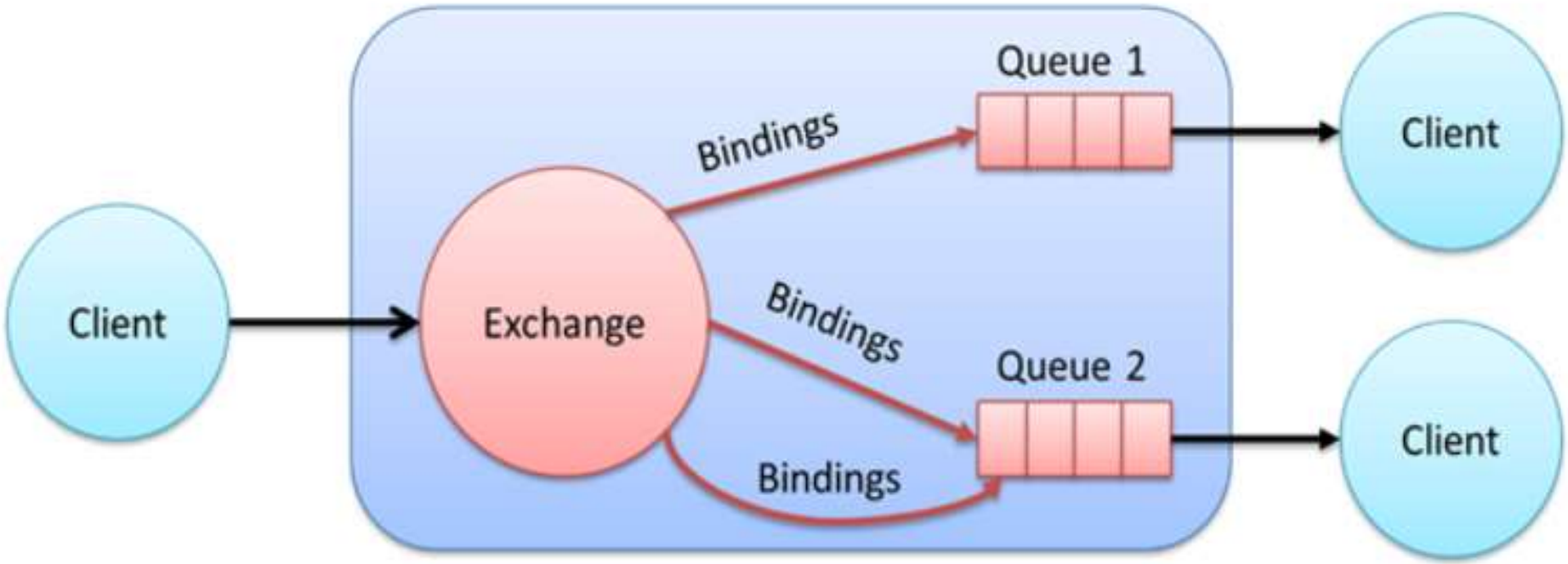


AMQP – Advanced Message Queue Protocol

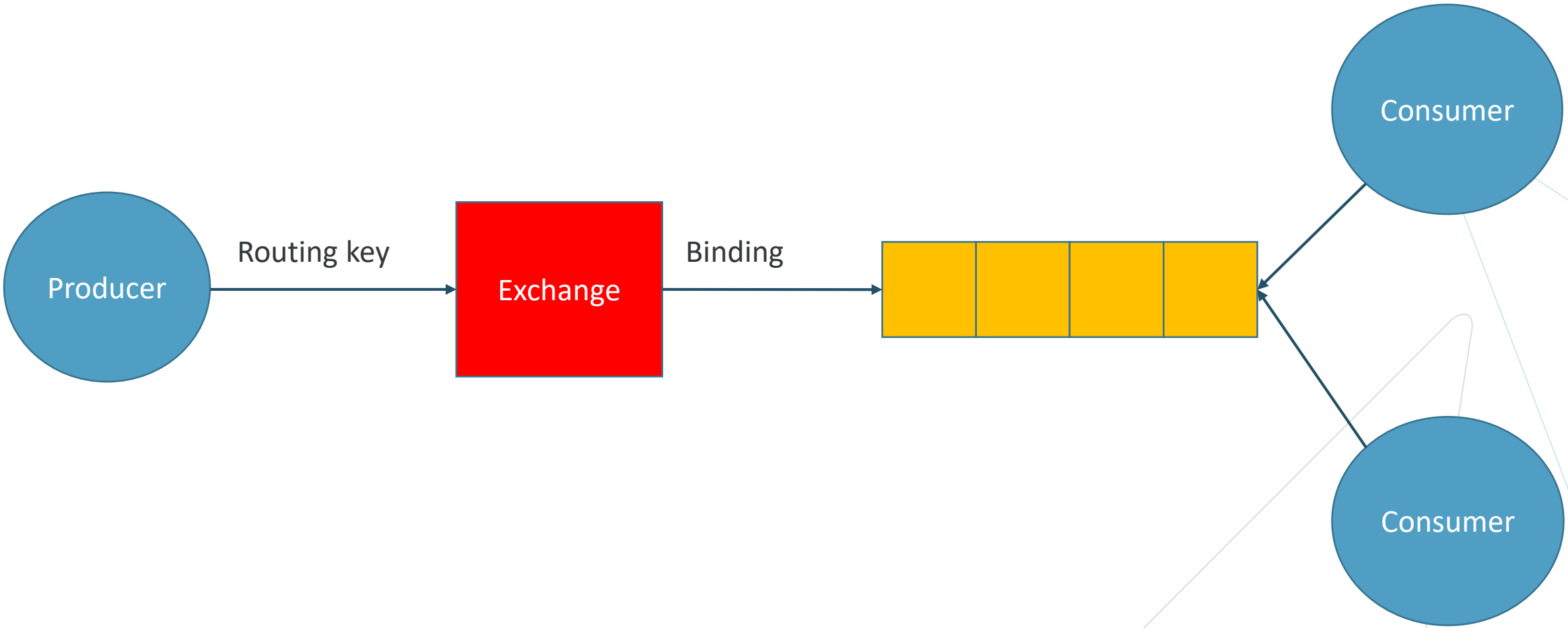
- Conceived by JP Morgan in 2006
- Answer to current Middleware Hell
  - 100's of applications
  - 10,000's of links
  - every connection different
  - massive waste of effort
- Goal to make interoperable Message Oriented Middleware



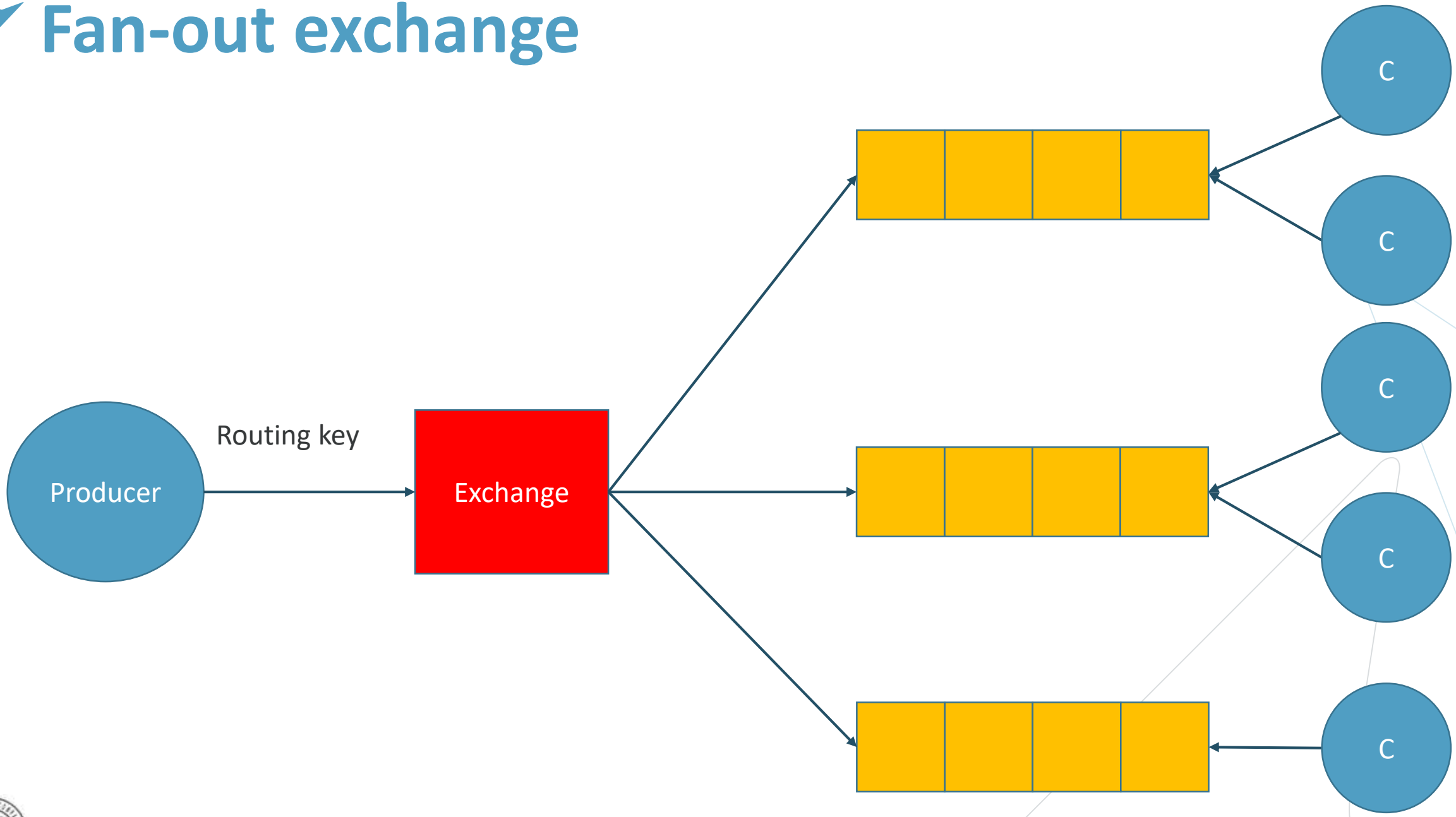
AMQP 0-10 vs 1.0



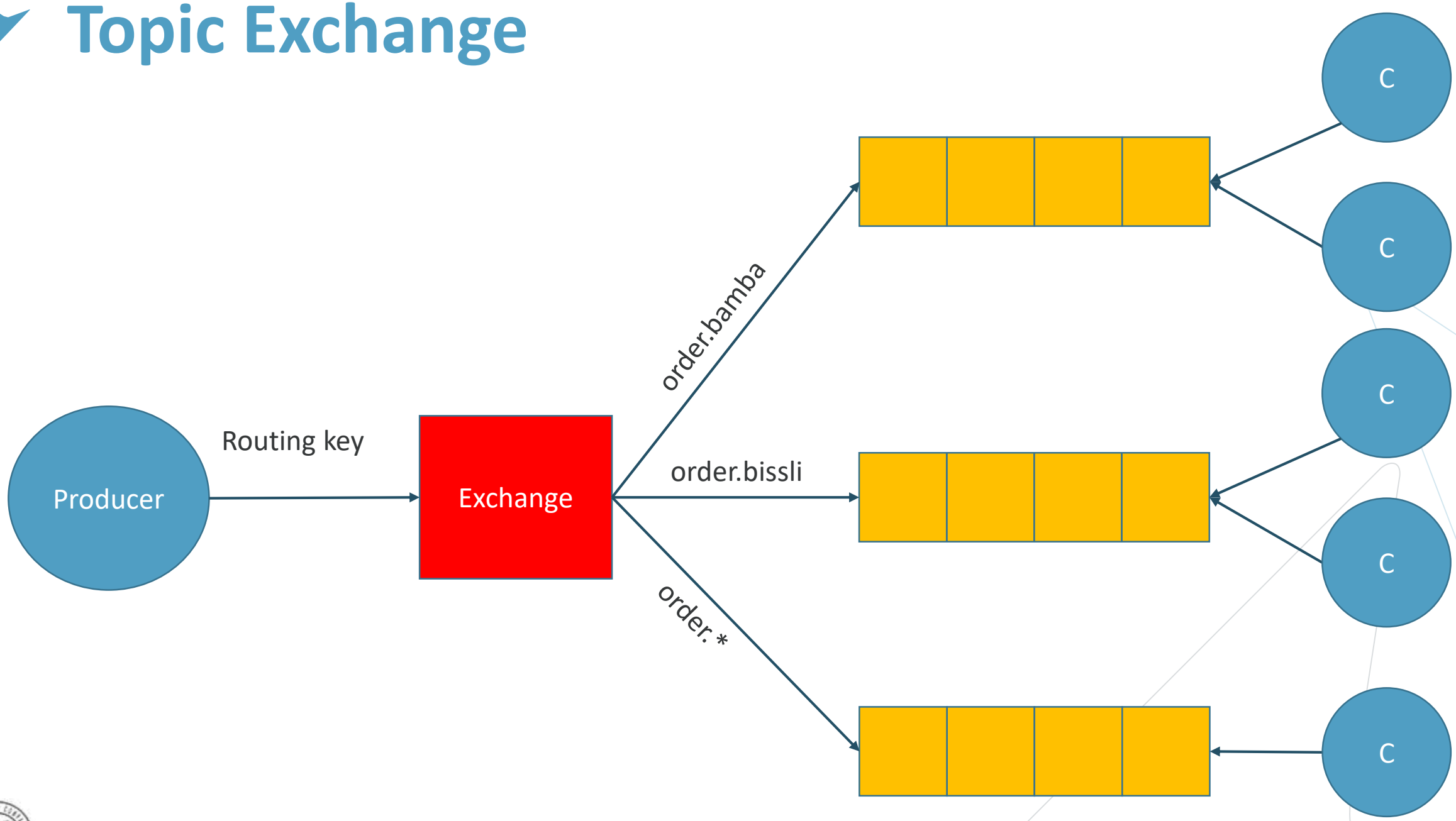
Direct Exchange



Fan-out exchange



Topic Exchange

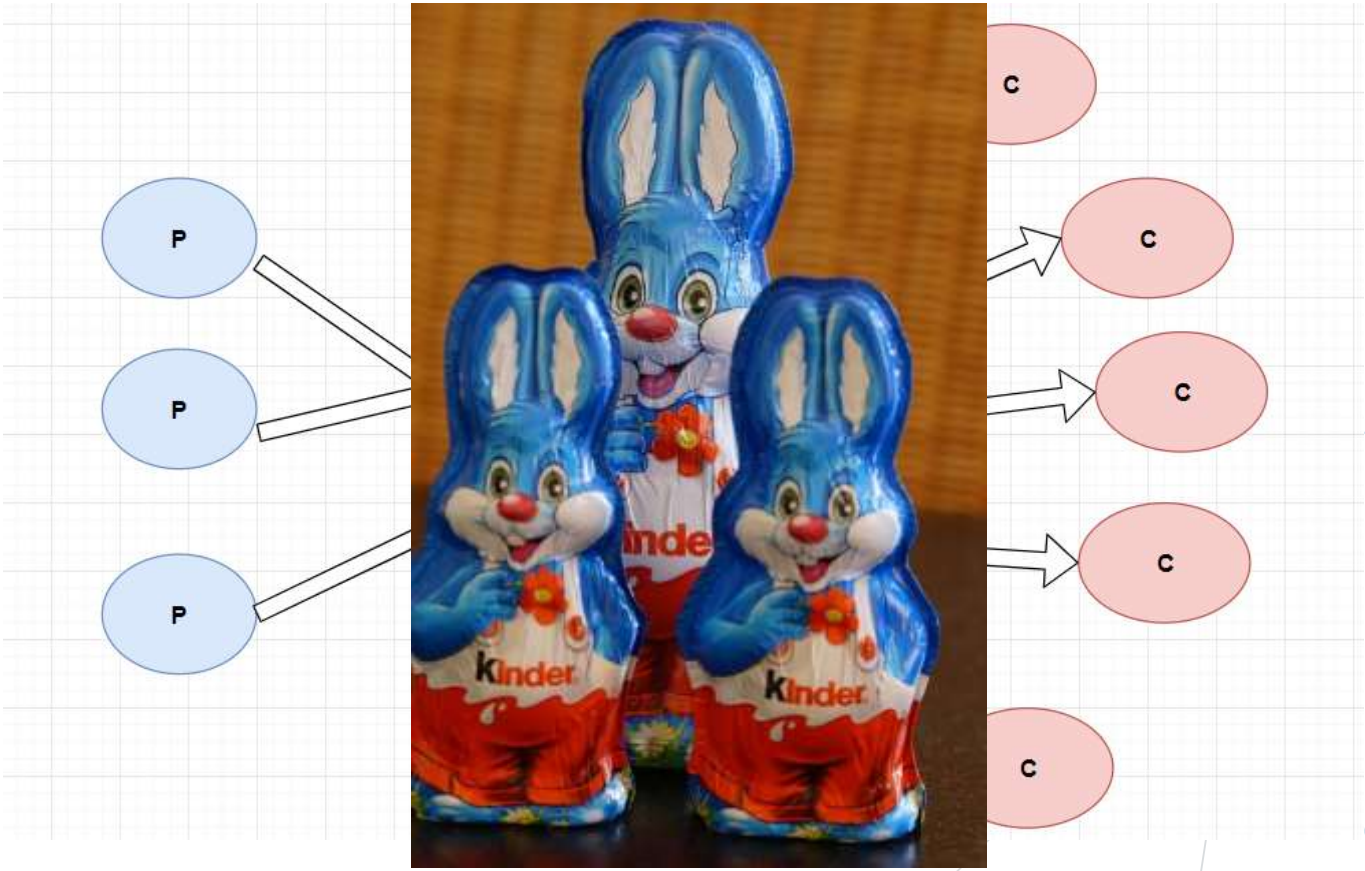


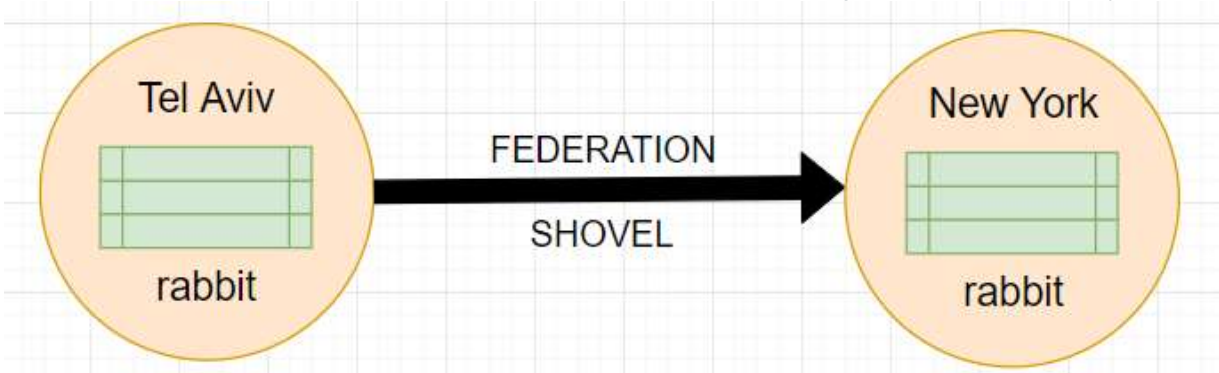
RabbitMQ

- Open source
- Support multiple protocols AMQP,MQTT,STOMP,HTTP
- Routing capabilities
- Reliability
- Scalability through clustering
- High availability
- Management and monitoring
- Security
- Traceability and debugging
- Pluggable architecture

RabbitMQ

- Clustering
  - Act as single logical unit
  - Consumer seens cluster as a single node
  - Automatic metadata replication
- Mirrored queues
- Federation/Shovel
  - Consumes and republishers
  - Exchange to exchange
  - Works well accros WAN
  - Two main difference
    - Shovel is low level
    - Shovel is more flexible





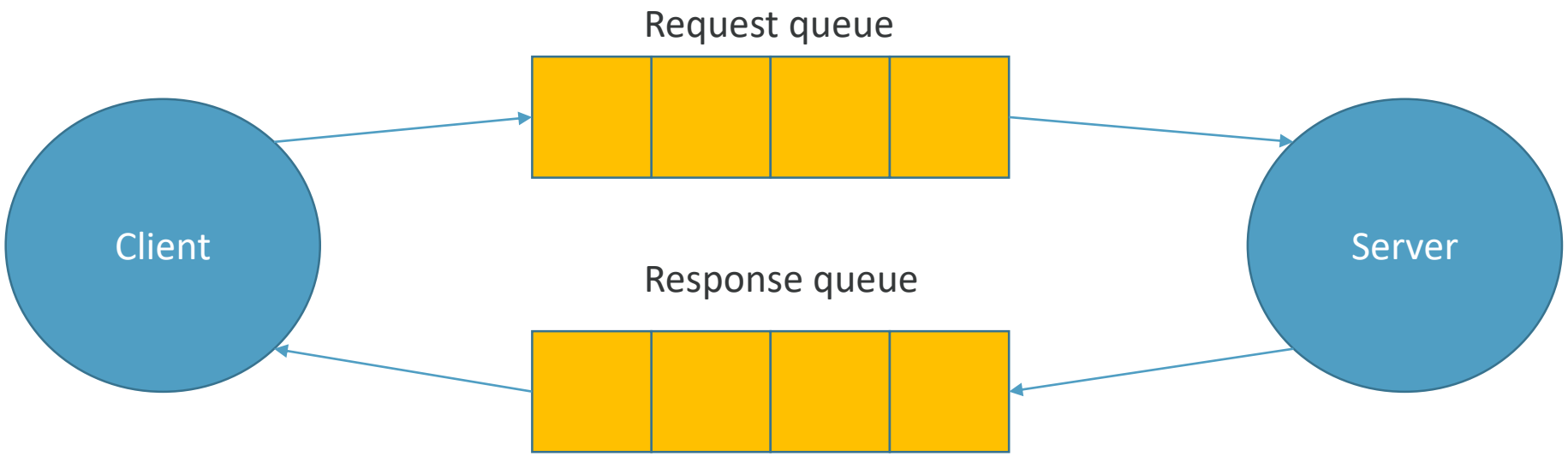
RabbitMQ

- Know your rabbit (DevOps!)
- Erlang
  - Configuration are Erlang tuples
  - You will need to deal with Erlang's message passing framework
  - Reading source and extend the system will be difficult
- Optimized for discrete message handling
- Content filtering
- It's not too fast
  - Optimized for discrete message handling
  - Fan-out ~35000 mps
  - Direct ~35000 mps
  - Topic ~10000 mps
  - Consistent-Hash ~40000 mps

RabbitMQ

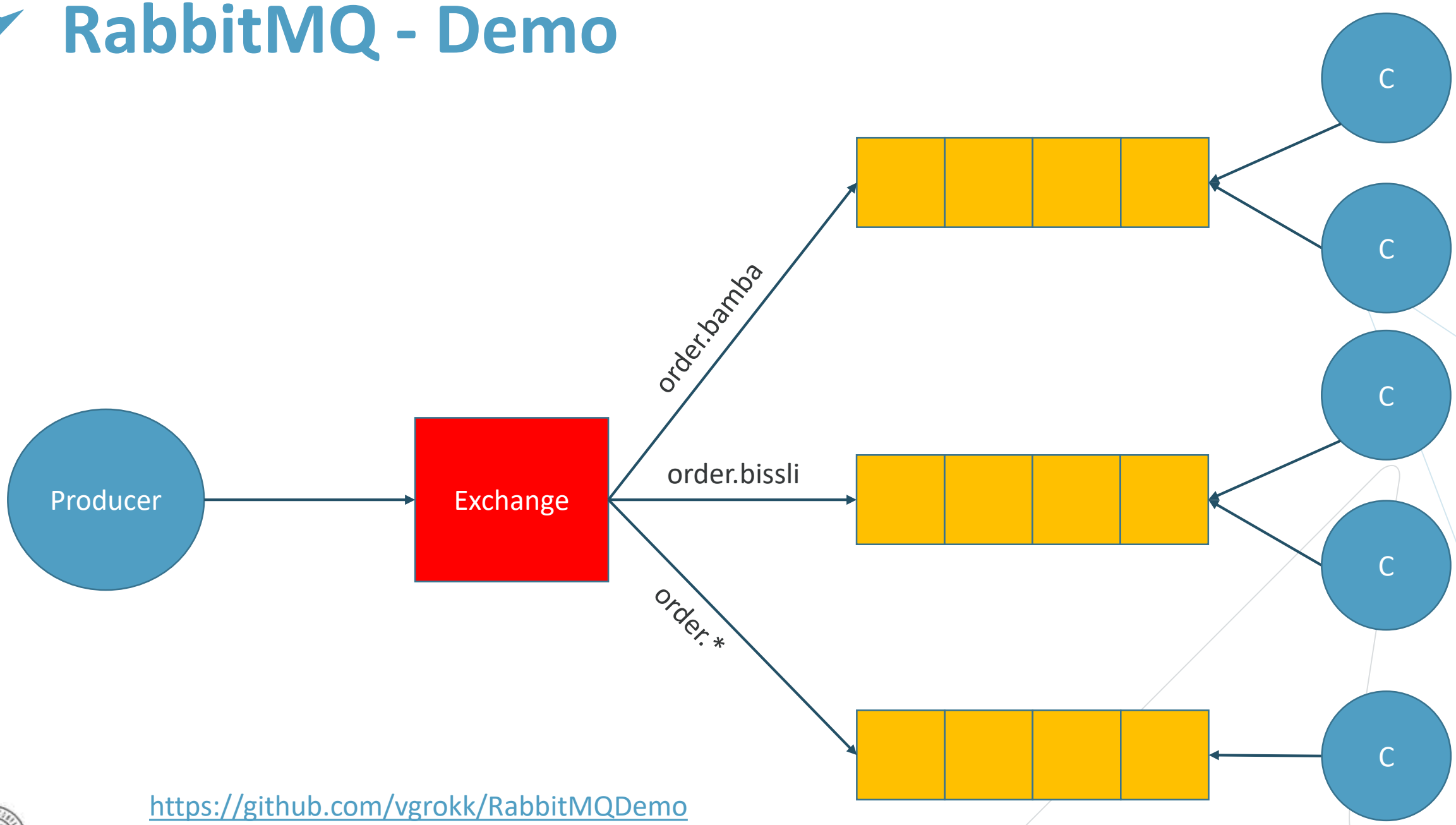
- Community support
  - https://groups.google.com/forum/#!forum/rabbitmq-users
  - Slack - https://rabbitmq-slack.herokuapp.com/
- Official site - http://www.rabbitmq.com/
- Books
  - RabbitMQ in Action
  - RabbitMQ in Depth
  - RabbitMQ Essentials
- RabbitMQ best practices – CloudAMQP
  - https://www.cloudamqp.com/blog/2017-12-29-part1-rabbitmq-best-practice.html
  - Video

RabbitMQ - Demo



https://github.com/vgrokk/RabbitMQDemo

RabbitMQ - Demo



https://github.com/vgrokk/RabbitMQDemo

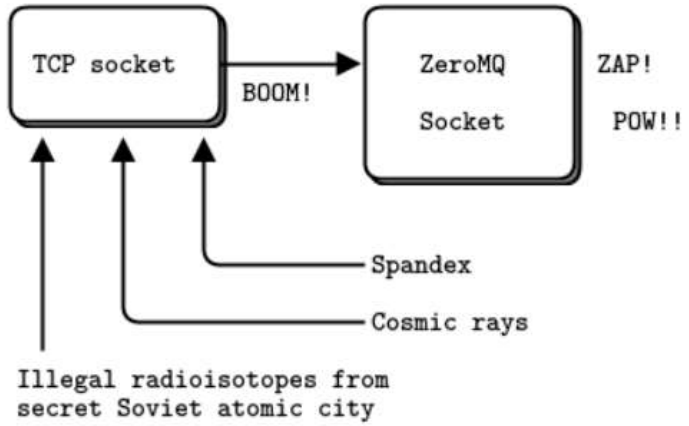
No more brokers

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

How It Began

We took a normal TCP socket, injected it with a mix of radioactive isotopes stolen from a secret Soviet atomic research project, bombarded it with 1950-era cosmic rays, and put it into the hands of a drug-addled comic book author with a badly-disguised fetish for bulging muscles clad in spandex. Yes, ZeroMQ sockets are the world-saving superheroes of the networking world.

Figure 1 - A terrible accident...



ZeroMQ

What Ø come for?

The original Ø meant:

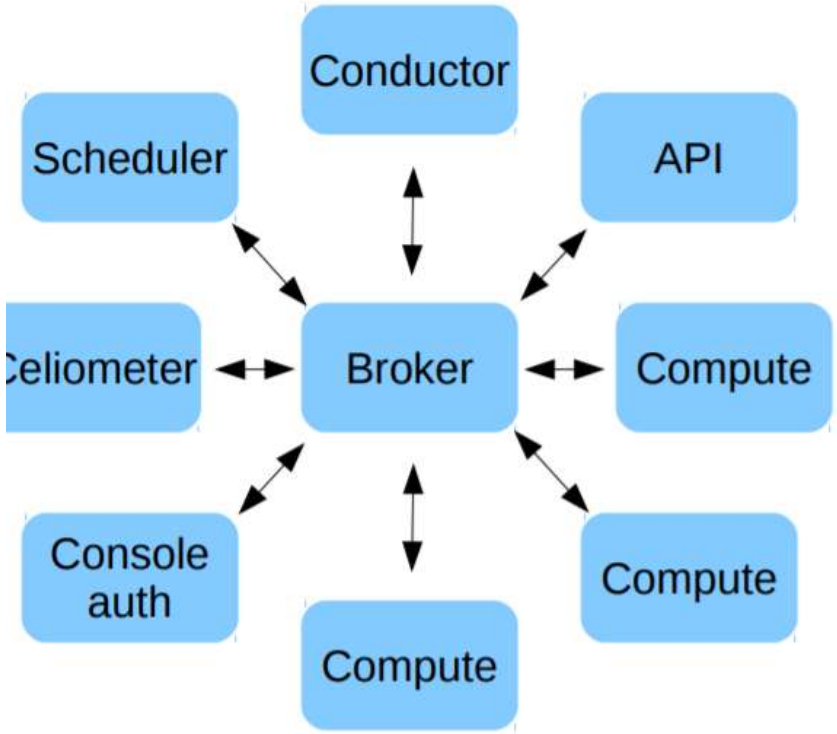
- Zero broker
- Zero latency (as close as possible)

With the time

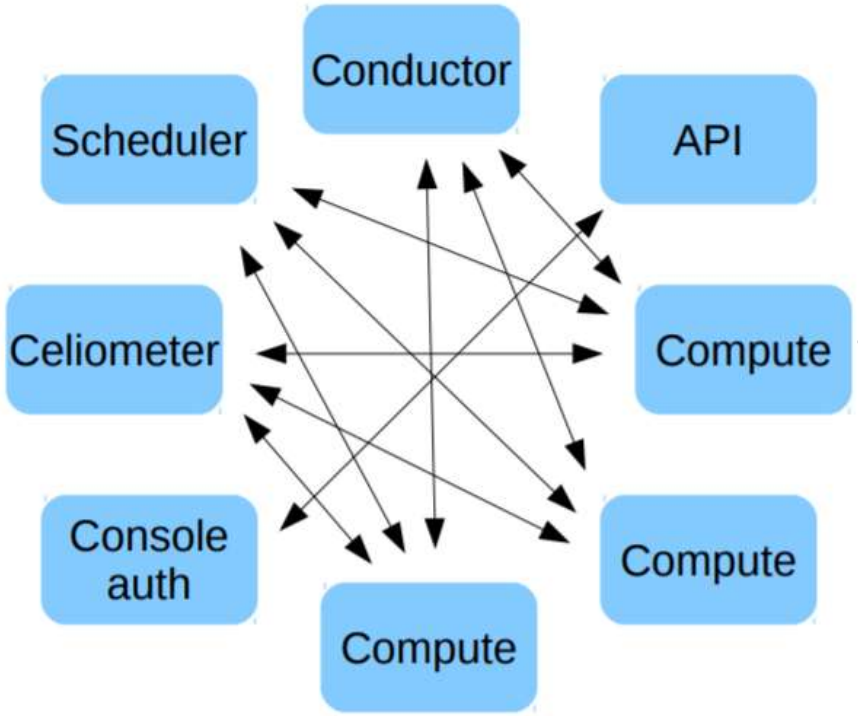
- Zero administration
- Zero cost
- Zero waste

ZeroMQ

Brokers are limited to a star topology



ZeroMQ is a partially-connected mesh



ZeroMQ

- Creators - iMatix CEO Pieter Hintjens and Martin Sustrik
- Super socket library
  - Language agnostic (40+ languages)
  - Multiple network protocols
- Multiple connection patterns
- High throughput/low latency (~5m mps, 30usec latency)
- Messages
  - Batching
  - Atomic
- Small (~25k lines of code)
- Open source. Large supported community. Multiple forks (nanomessage)

ZeroMQ - Cons

- Durability
- Static routing
- No out of the box solution
  - Service discovery
  - Management
  - Delivery deadlines
  - QoS
- Low level
- Internal queue management

ZeroMQ - Sockets

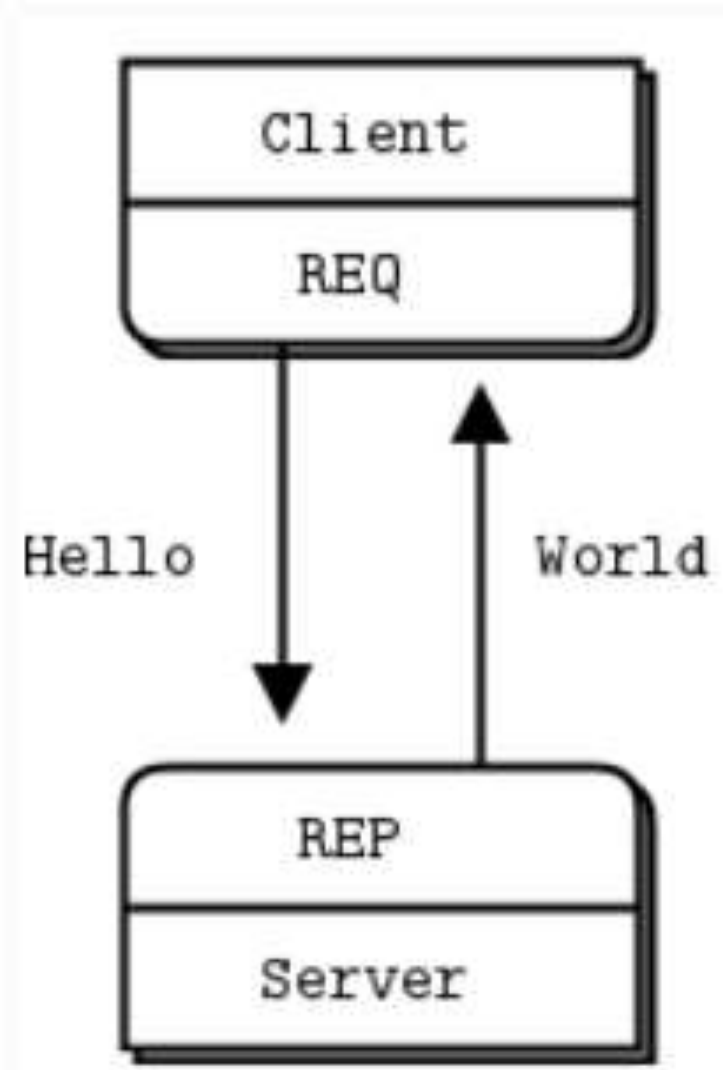
- Unicast – inproc, ipc (except on Windows), tcp
- Multicast – pgm, epgm
- Asynchronous
- Designed for particular messaging patterns
- Connect to multiple endpoints
- Automatically reconnect
- Queue messages when under heavy load

ZeroMQ - Sockets

- PUB - Outgoing broadcast
- SUB - Broadcast listener
- REQ - Synchronous (1 msg at a time) request
- REP - Synchronous reply
- DEALER - Async request
- ROUTER - Async reply
- PUSH - One way outgoing to PULL(s)
- PULL - One way incoming from PUSH(es)
- PAIR - One-to-one 2-way with another PAIR

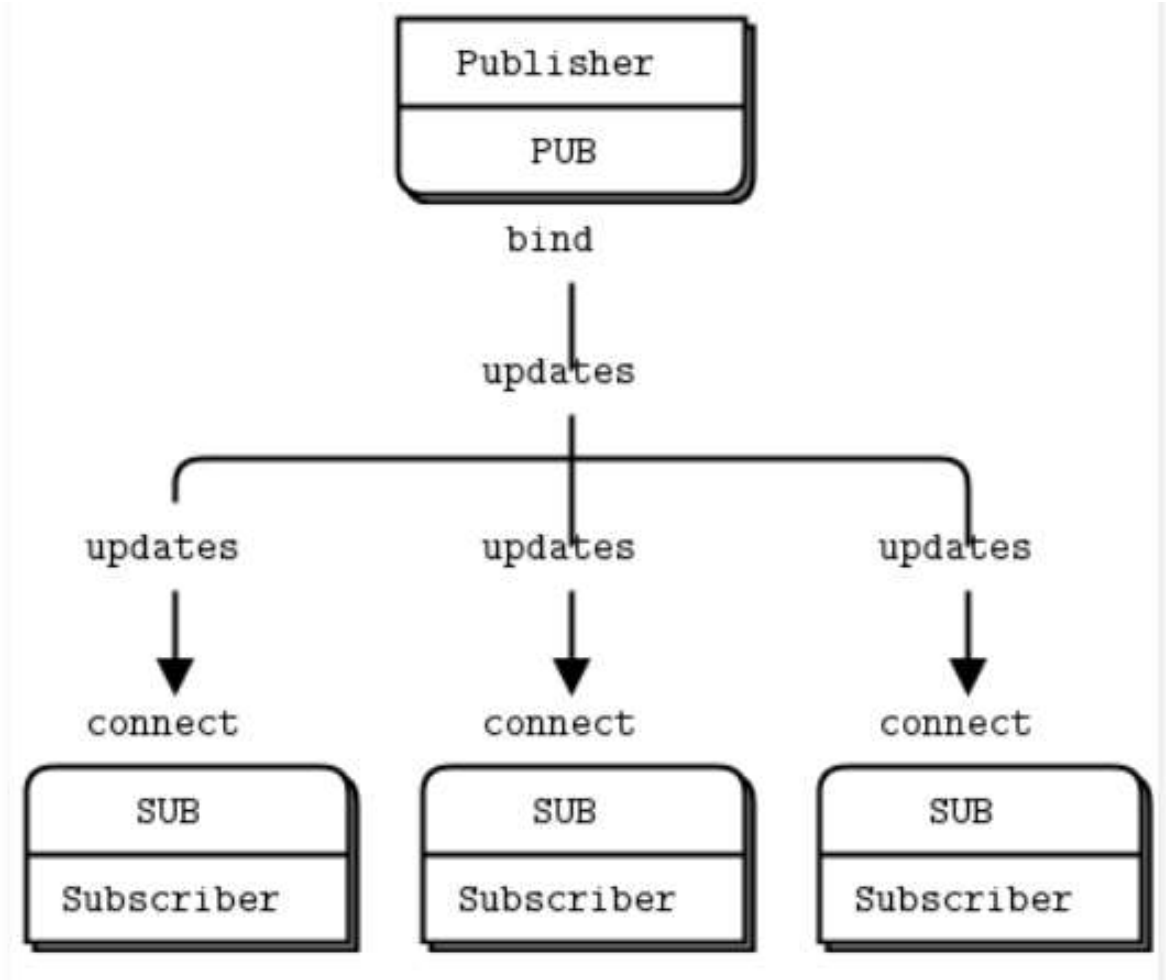
- PUB and SUB
- REQ and REP
- REQ and ROUTER
- DEALER and REP
- DEALER and ROUTER
- DEALER and DEALER
- ROUTER and ROUTER
- PUSH and PULL
- PAIR and PAIR

ZeroMQ - Demo



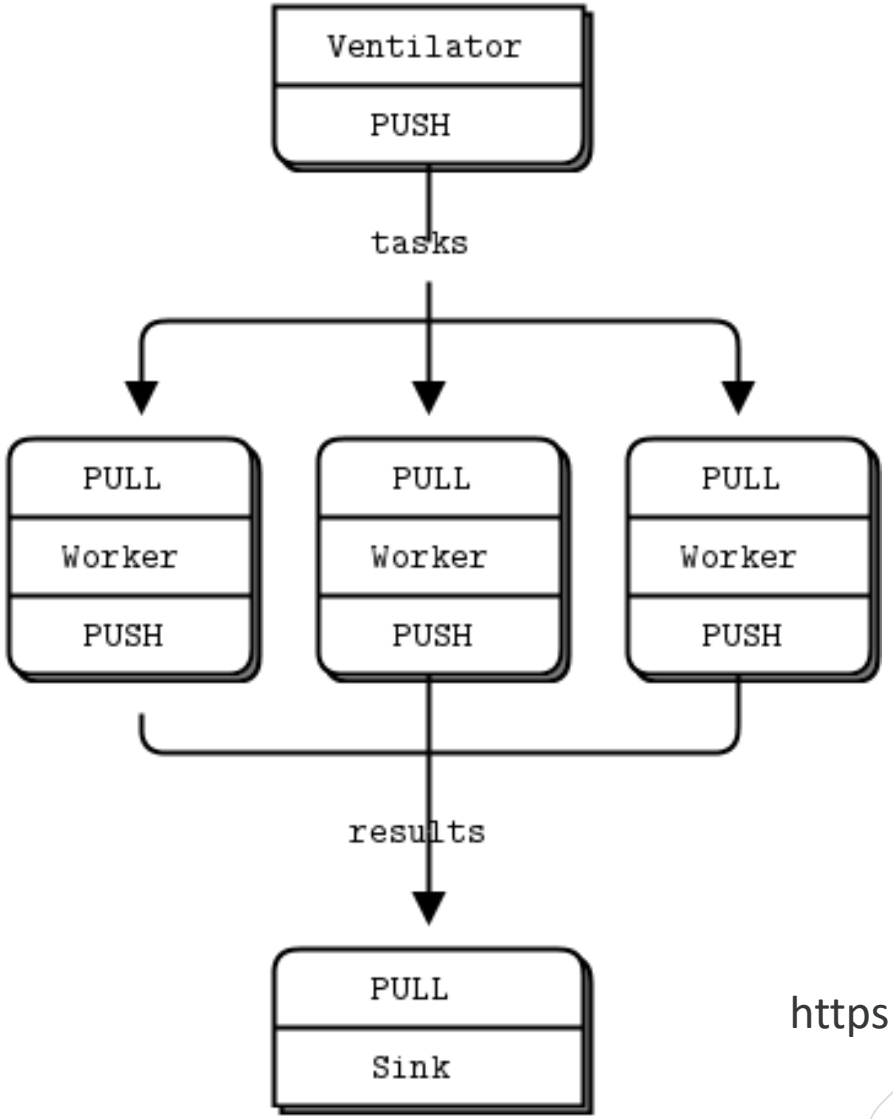
https://github.com/vgrokk/ZeroMQDemo

ZeroMQ - Demo



https://github.com/vgrokk/ZeroMQDemo

ZeroMQ - Demo



https://github.com/vgrokk/ZeroMQDemo


amazon
web services
Google
Cloud Platform
Microsoft Azure
We cover your *aaS

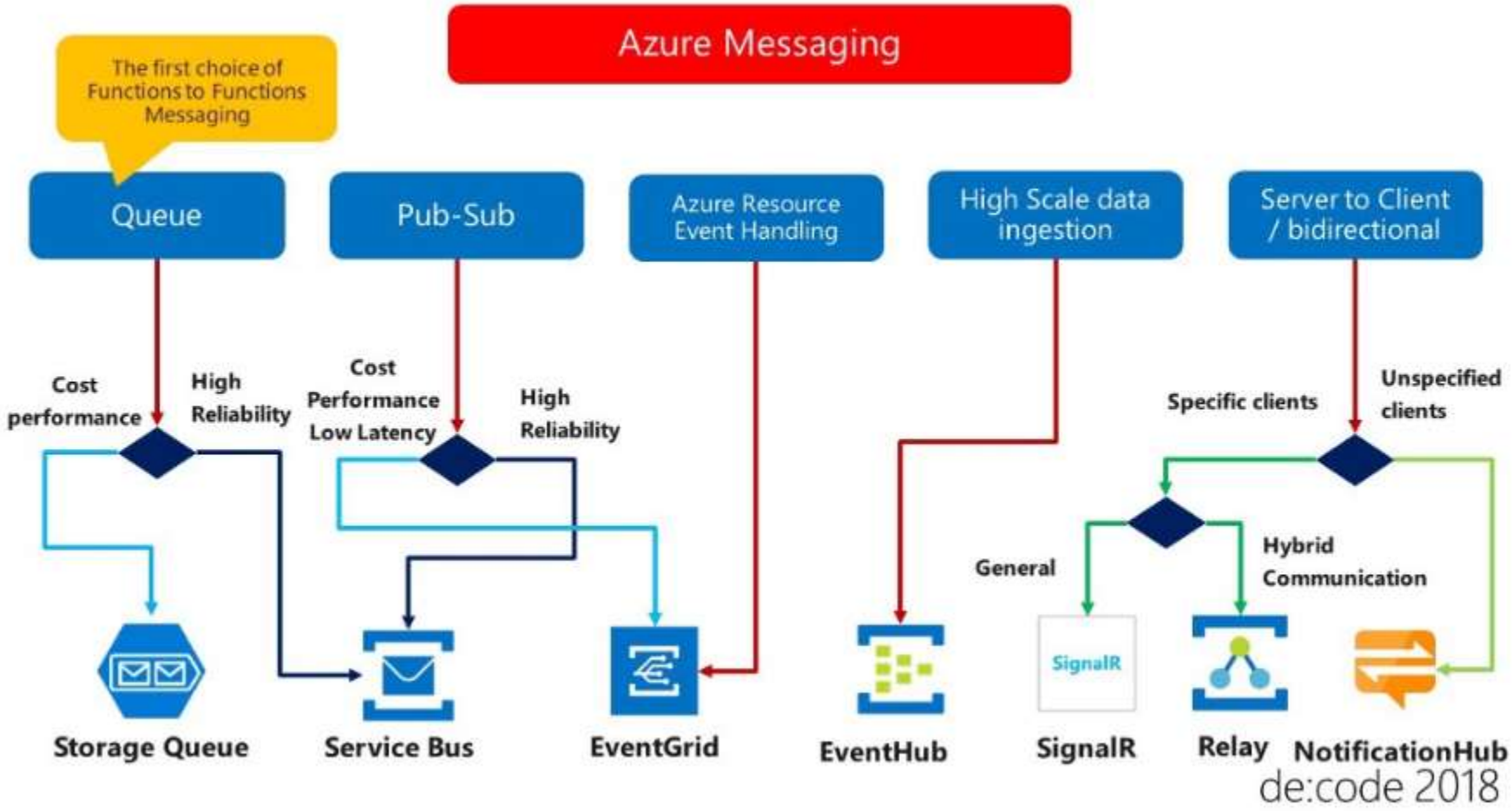
Cloud

- You host your services on cloud
- You don’t have a dedicated budget or skills to maintain your own queueing system
- You aren’t necessarily interested in lowest latency
- You prefer easy solutions over powerful solutions when queuing is considered
- You don’t plan on processing large amounts of messages

Self - hosted

- Your system is distributed or hosted outside of public cloud
- You want an open solution that you can reuse
- You're worried about latency
- You want to configure exchanges or use some other non-trivial methods of message distribution
- You want your queuing system to be extendable and configurable, and
- You want maximum performance and have dedicated resources to invest in your messaging solution.

Azure


Azure Messaging
The first choice of Functions to Functions Messaging
Queue
Pub-Sub
Azure Resource Event Handling
High Scale data ingestion
Server to Client / bidirectional
Cost performance
High Reliability
Cost Performance Low Latency
High Reliability
Specific clients
Unspecified clients
General
Hybrid Communication
Storage Queue
Service Bus
EventGrid
EventHub
SignalR
Relay
NotificationHub
de:code 2018

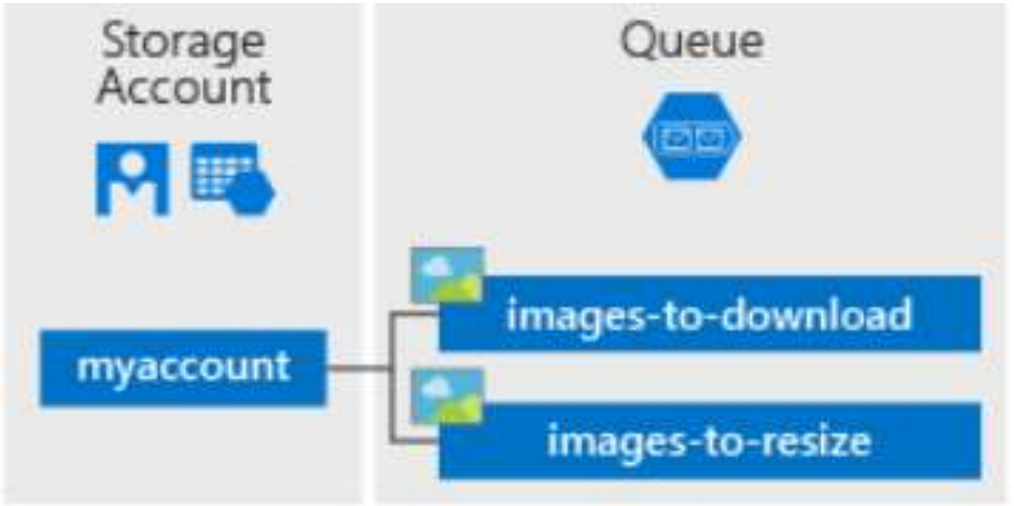
Storage queue

PROS

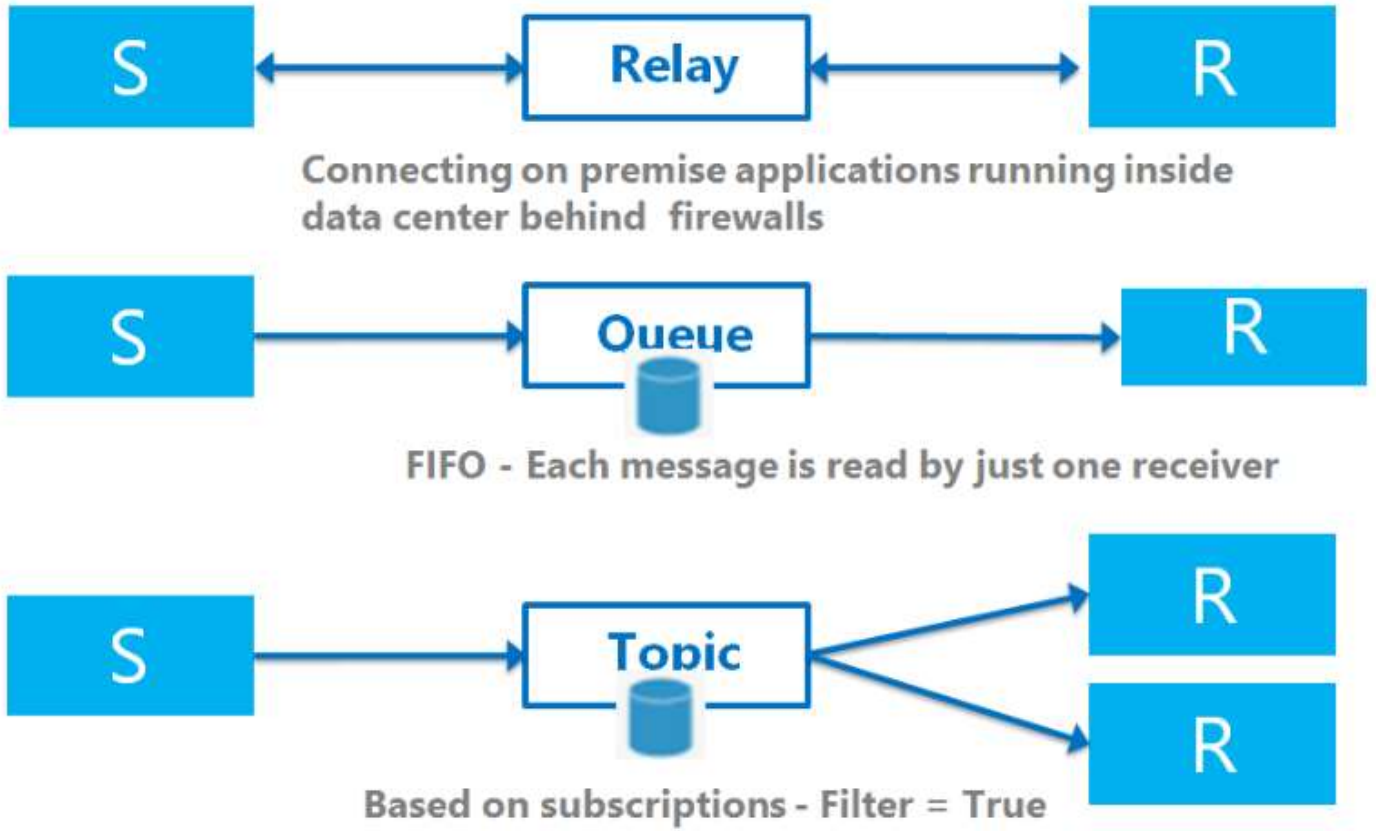
- Can store over 80Gb in queue
- Low cost (100x cheaper than Azure Service Bus)
- Batch receive (up to 32 messages)
- Only pay for storage and operations
- Redundancy(LRS, ZRS, GRS, RA-GRS)
- Rest api

CONS

- Maximum size of a message is 64KB
- Maximum mps is 2000
- Maximum TTL for each message is 7 days
- No order guaranty
- Only polling (for receiver)
- No batch support for sender



Service bus


S
Relay
R
Connecting on premise applications running inside data center behind firewalls
S
Queue
R
FIFO - Each message is read by just one receiver
S
Topic
R
R
Based on subscriptions - Filter = True

Service Bus

- Scheduled delivery
- QoS
- Poison message handling
- Defer
- Auto Forward
- Sessions
- Batching
- Ordering
- Auto-delete on idle
- OnMessage (.NET only)
- Duplicate detection
- Content filters
- Transactions
- Dead lettering

Amazon

Amazon SQS

Managed queue

Amazon SNS

Managed pub/sub

Amazon MQ

Managed Apache ActiveMQ

Amazon Kinesis

Managed event streaming

AWS IOT

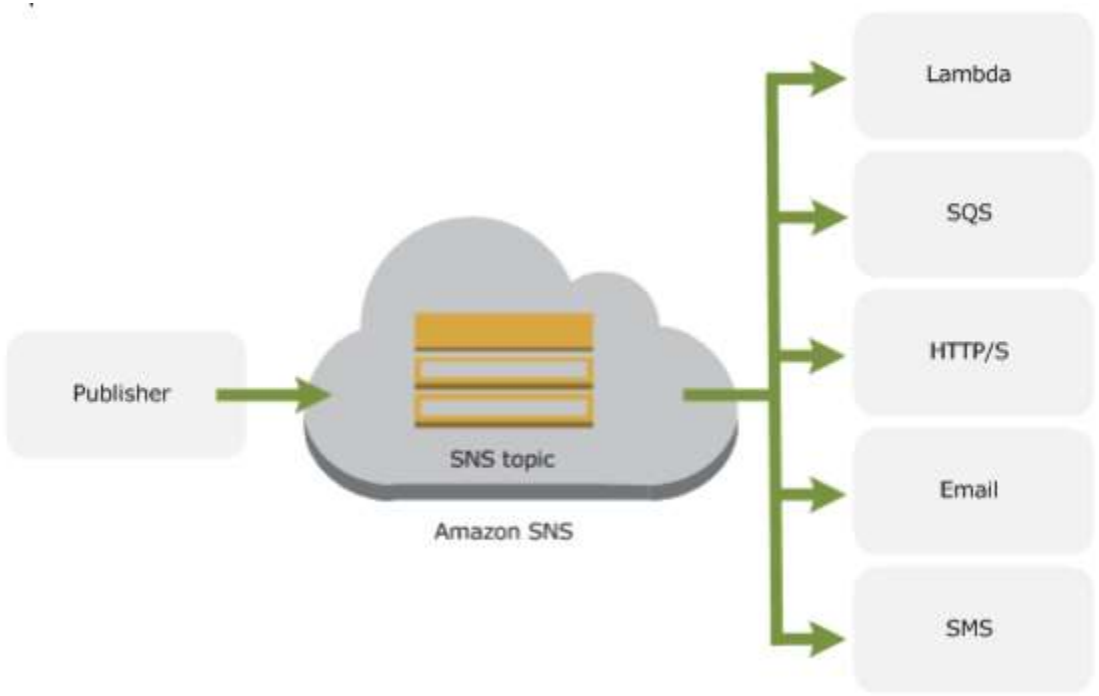
Amazon SQS

- Unlimited queues and messages
- Payload Size: up to 256KB
- Batches
- Long polling
- Retain messages in queues for up to 14 days

- Send and read messages simultaneously
- Message locking
- Queue sharing
- Server-side encryption (SSE)
- Dead Letter Queues (DLQ)
- Amazon CloudWatch metrics + alerts

Amazon SNS

- Fan-out pattern
- Reliability
- Flexible
- Nearly unlimited throughput
- Secure
- Payloads up to 256 KB
- Amazon CloudWatch



Amazon MQ

Amazon MQ is a managed
message broker service
for Apache ActiveMQ that makes
it easy to set up and operate
message brokers in the cloud.

Amazon MQ

ActiveMQ Features

- Transient & Persistent
- Industry standard APIs and protocols
- Local & Distributed transactions
- Queues & topics (FIFO)
- Message filtering
- Request/reply
- Scheduled messages
- Unlimited message size
- Unlimited retention

AmazonMQ Features

- Provisioning
- Updates
- Security
- Maintenance
- Monitoring
- Troubleshooting
- High availability

Demo time

- Was created by Derek Colission in 2010
- Open source, lightweight, high-performance cloud native messaging system
- Capable of sending 11-12 million messages per second
- Written in Go
- Available in two interoperable modules
  - NATS Server
  - NATS Streaming Server

eWeek: "NATS Messaging Project Joins Cloud Native Computing Foundation"

By Kristen Evans | March 16, 2018 | News

NATS

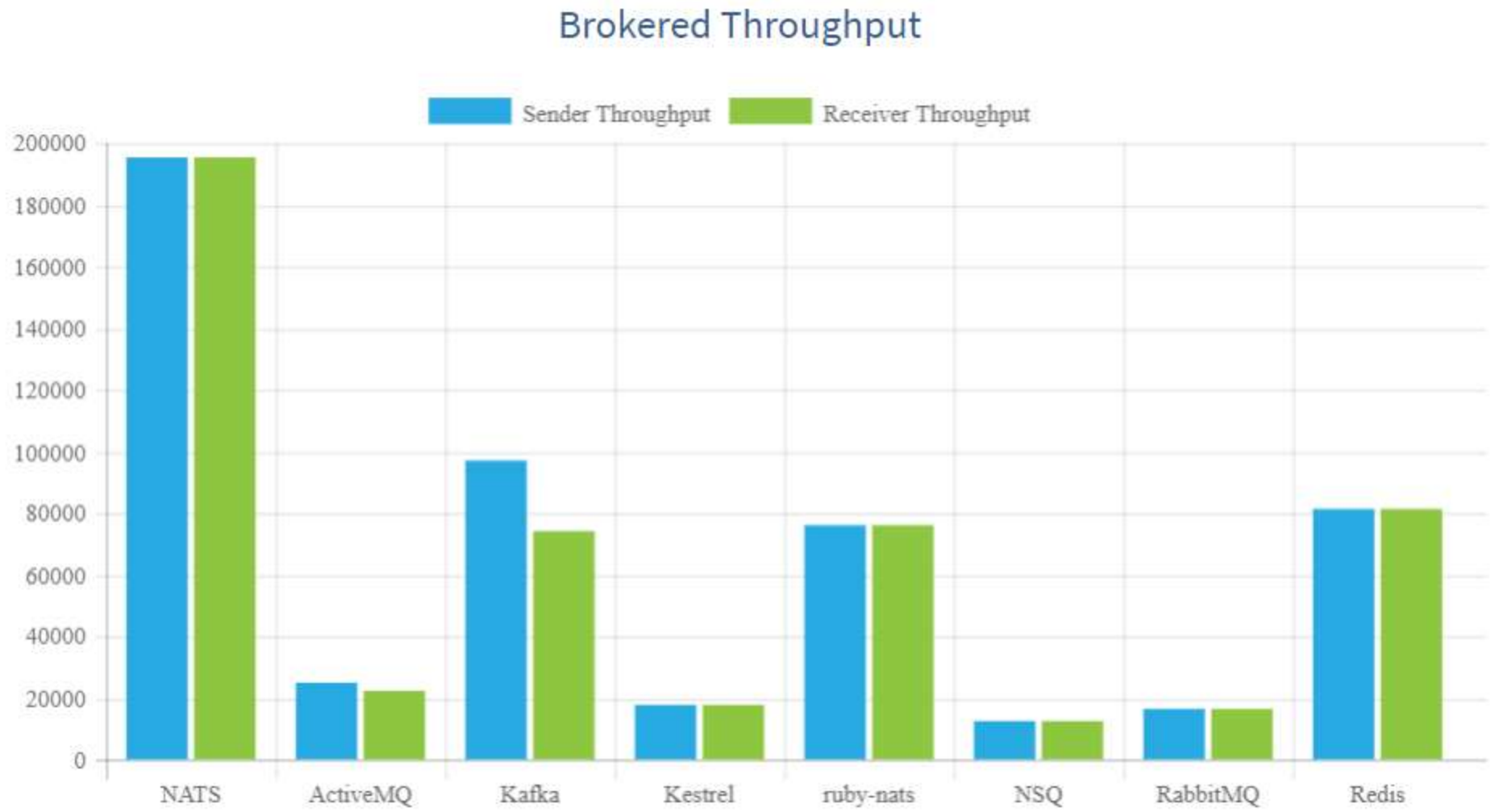


Chart source: bravenewgeek.com/dissecting-message-queues

NATS Server

- Highly performant, extremely lightweight
- Clustered servers/clustered aware clients
  - Built in resilience and high availability
  - Auto discovery for topology
- Text based protocol (payload as array of bytes)
- Monitorable on a dedicated port
- TLS support
- Auto Pruning of Clients
- Pure Pub/Sub
- Request/Reply ,Queueing pattern
- Smart routing with wildcards

- Durability
- Transactions
- Enhanced Delivery Models
- Flow control (rate matching/limitng)

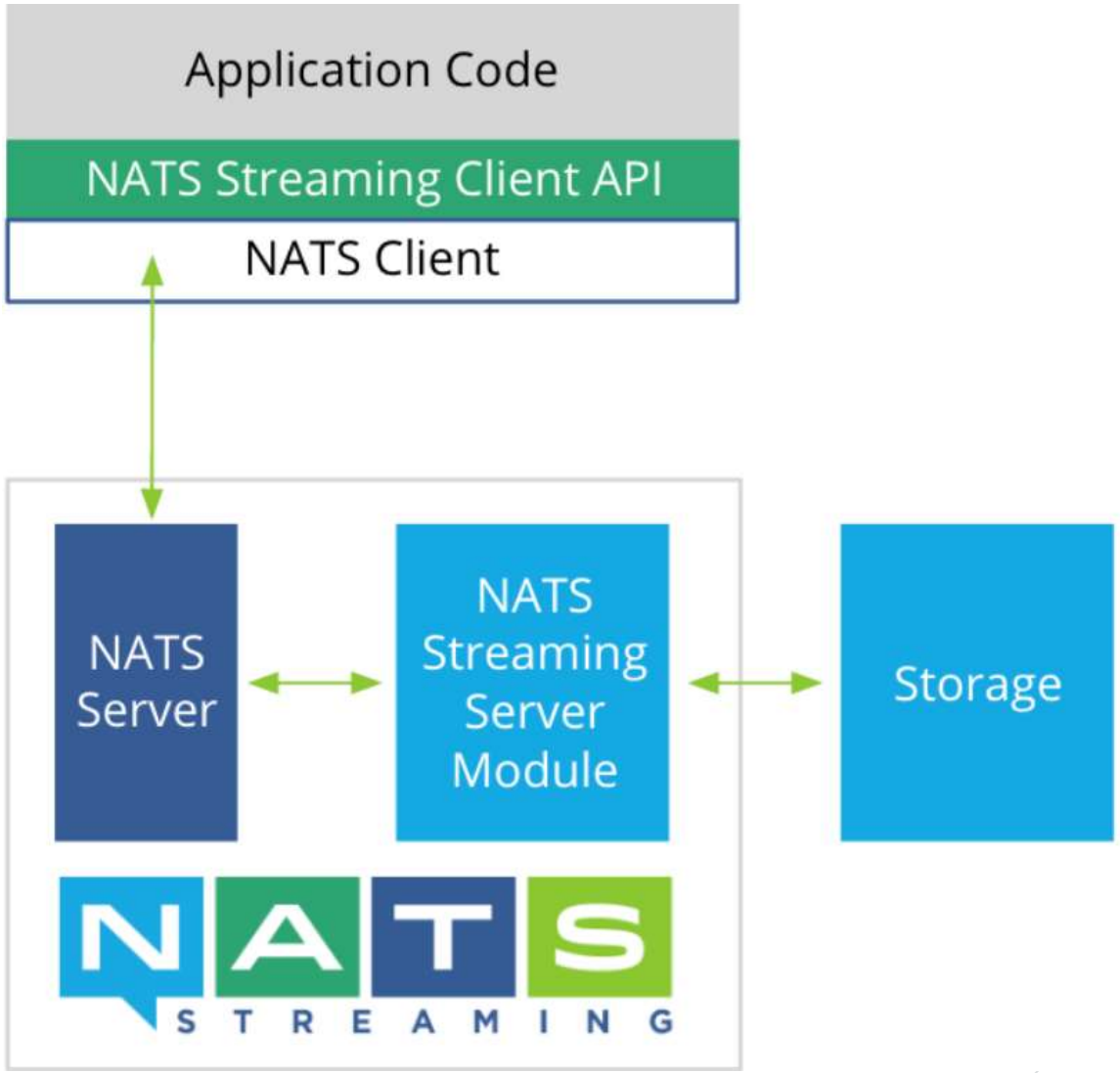
NATS use cases

- High throughput
- Addressing, discovery
- Command and control (control plane)
- Load balancing
- N-way scalability .
- Location transparency
- Fault tolerance

NATS Streaming

- Layer on top of NATS
- Enhanced message protocol - NATS Streaming implements its own enhanced message format using Google Protocol Buffers.
- Message/event persistence
- QoS
- Flow control (rate matching/limiting)
- Message replay by subject
  - All available messages
  - Last published value
  - All messages since specific sequence number
  - All messages since a specific date/time
- Durable subscribers

NATS Streaming



NATS

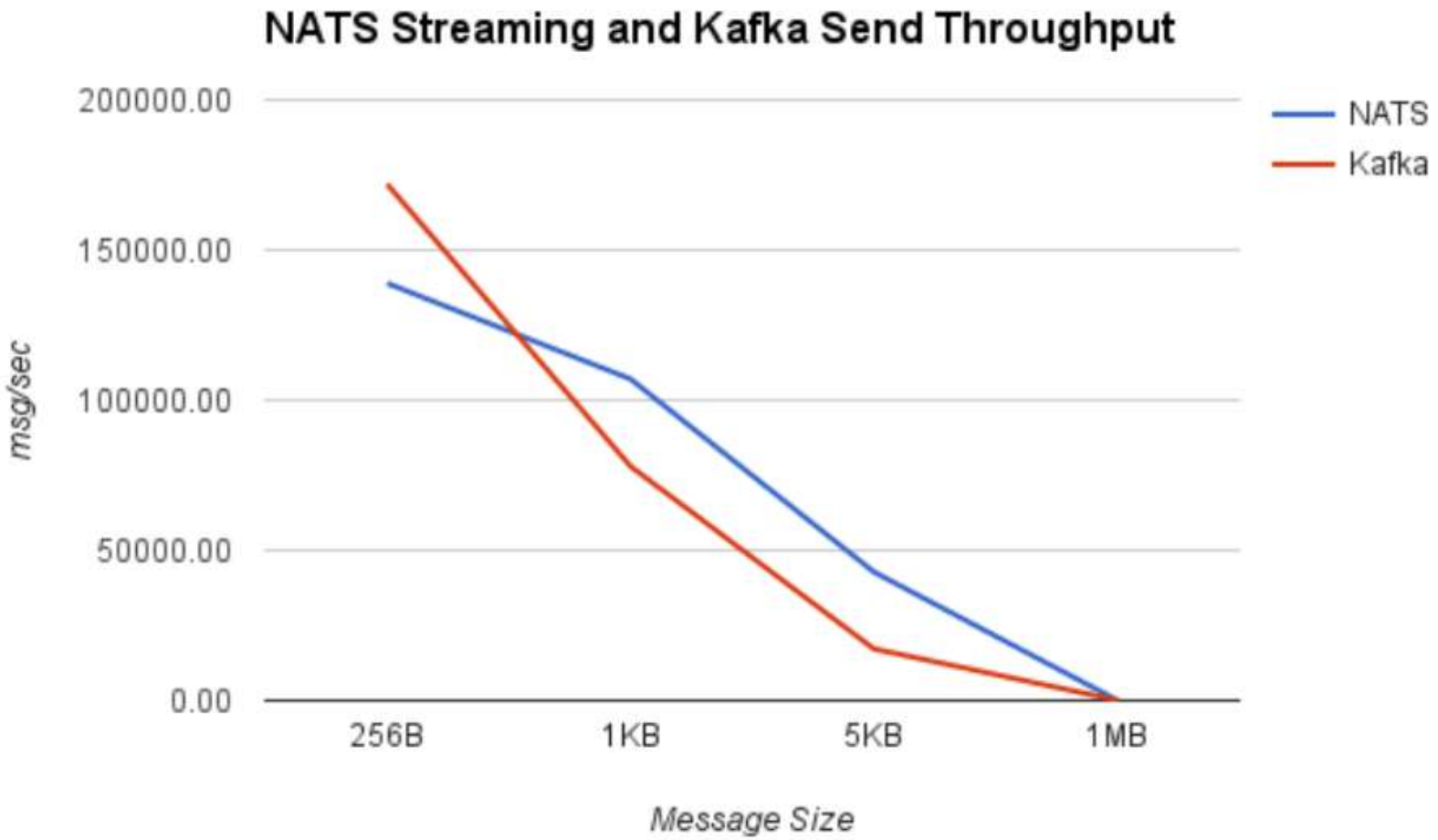
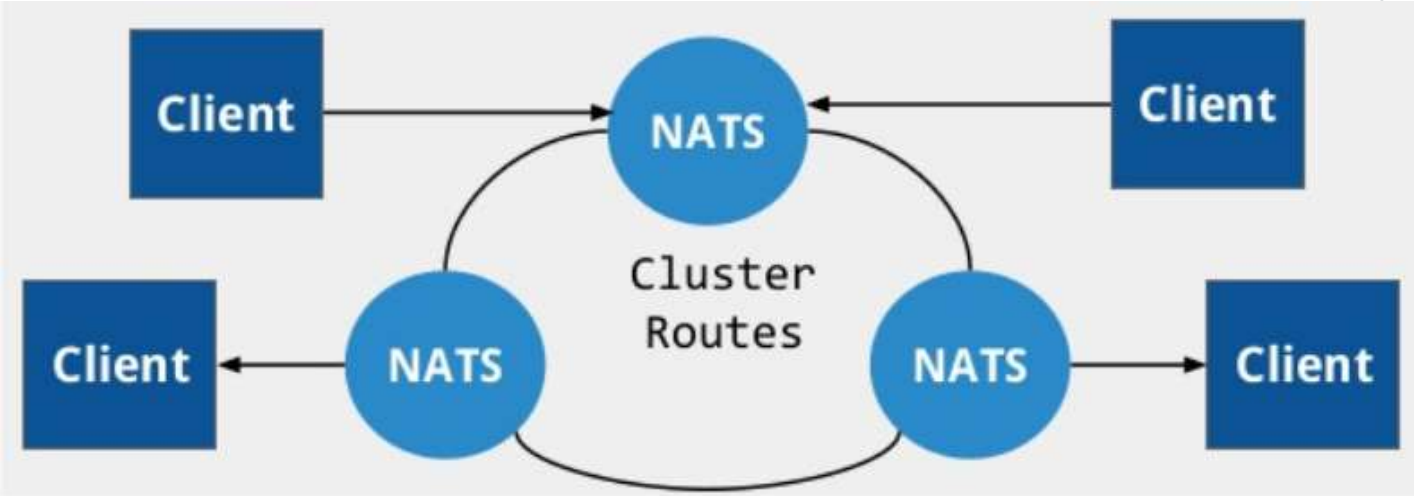


Chart source: https://dzone.com/articles/benchmarking-nats-streaming-and-apache-kafka

NATS cluster

- For high availability, a full mash of NATS servers can be setup
- Clients can connect to any of nodes to communicate with other clients, the NATS cluster would then route the messages
- Routing will be done if clients showed interests in subject
- Whenever a new NATS servers joins then members already in cluster connected as well to form the full mash



NATS In Production

Summary

- Define priorities
- Ask questions
- Dive deep
- Call to consultants

Thank you!