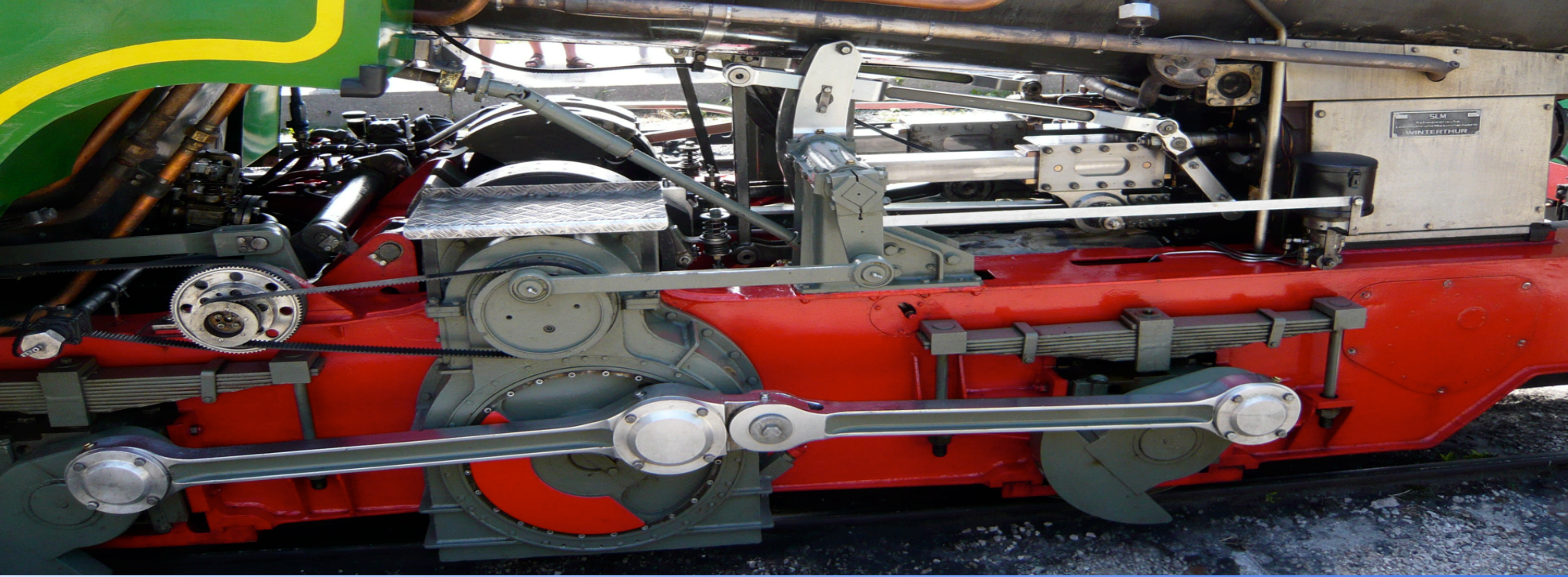


Why do I have to use a Message Queue System ?

Fabrizio Manfred Furuholmen

FrOSCon
Free and Open Source Software Conference

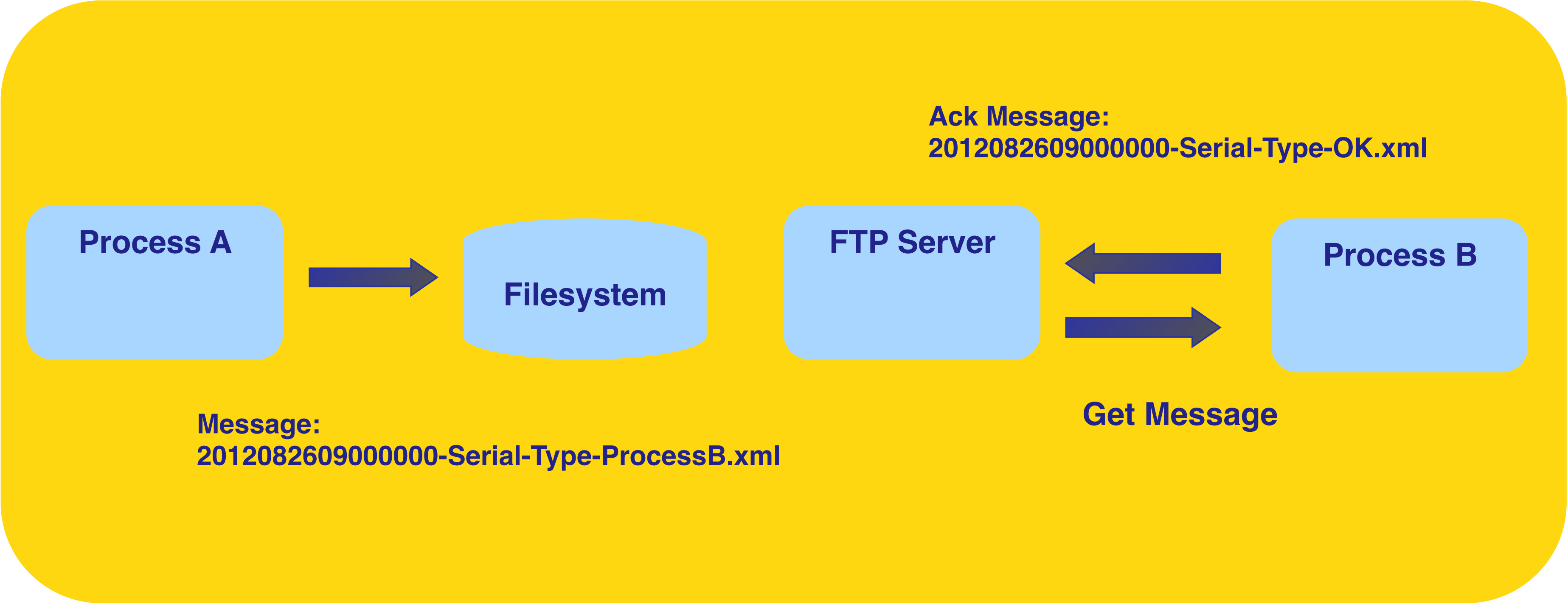
Beolink.org

Agenda

- Introduction
  - History
  - Basic components
- Message Queue
  - Usage type
  - Advantages
- Implementation
  - Solution
  - Performance
  - Scalability/High Avaibility
- Big Data
  - Distributed
  - Cloud Computing

Introduction: Example

Multimedia Format Transcoding

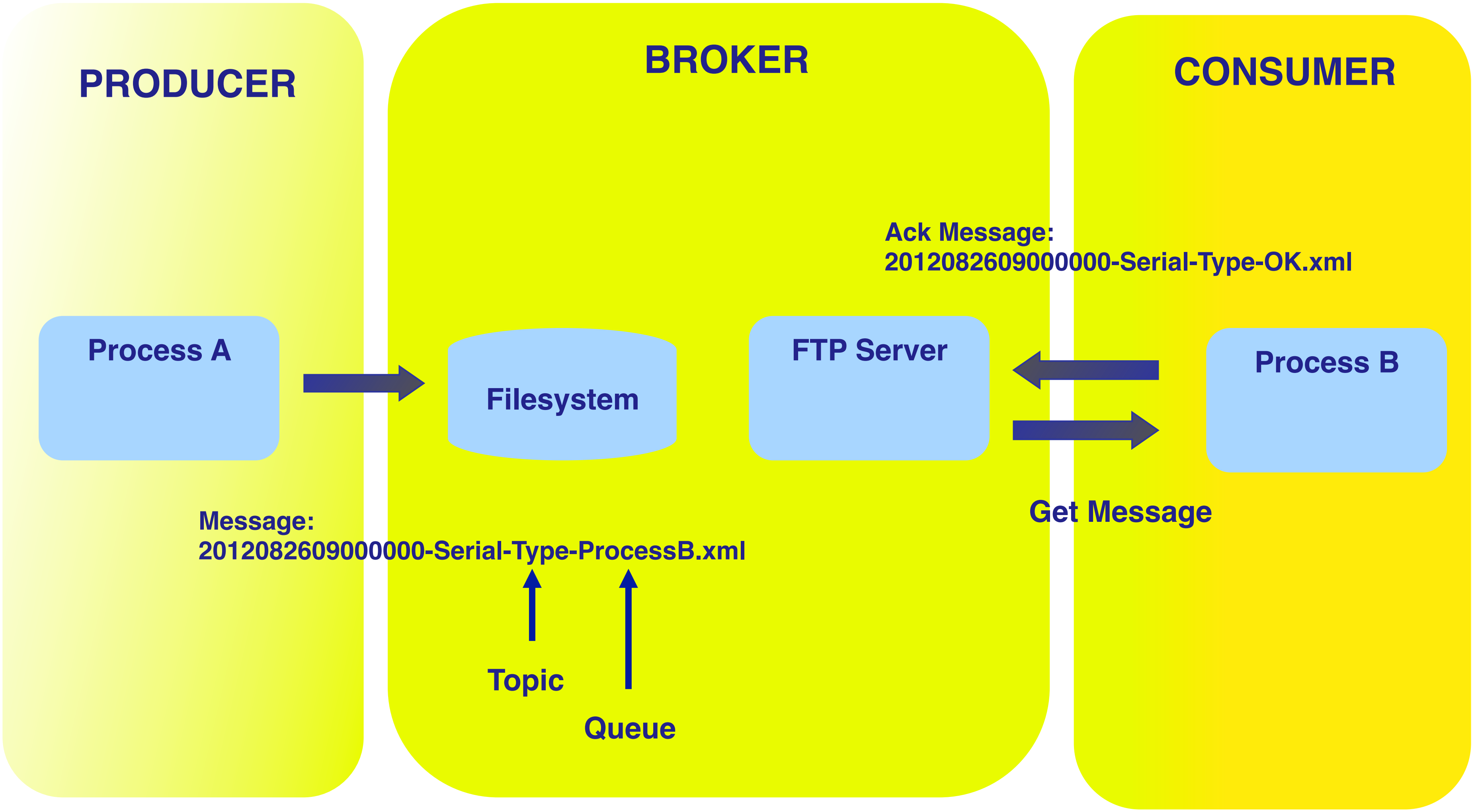

Ack Message:
2012082609000000-Serial-Type-OK.xml
Process A
Filesystem
FTP Server
Process B
Get Message
Message:
2012082609000000-Serial-Type-ProcessB.xml

More than 10 years ago

Introduction: Definition

“…message queueing is a method by which process (or program instance) can exchange or pass data using an interface to a system-managed queue of message...”

Introduction: Components



Introduction: Broker

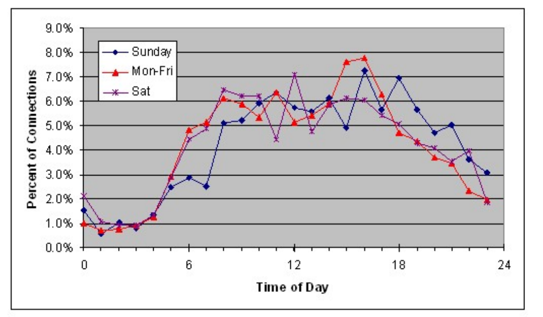
Message-oriented middleware (MOM)

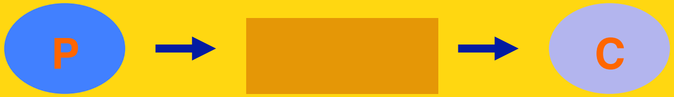
“…message broker is an architectural pattern for message validation, message transformation and message routing. It mediates communication amongst applications, minimizing the mutual awareness that applications should have of each other in order to be able to exchange messages, effectively implementing decoupling…”

Is message queue middleware only a temporary storage ?

Message Queue

- ❑ Asynchronous communication
  - ❑ Lock
  - ❑ Concurrent Read/Write
- ❑ Burst Message
- ❑ Decoupling
  - ❑ Reliability
- ❑ Multi platform





Es.

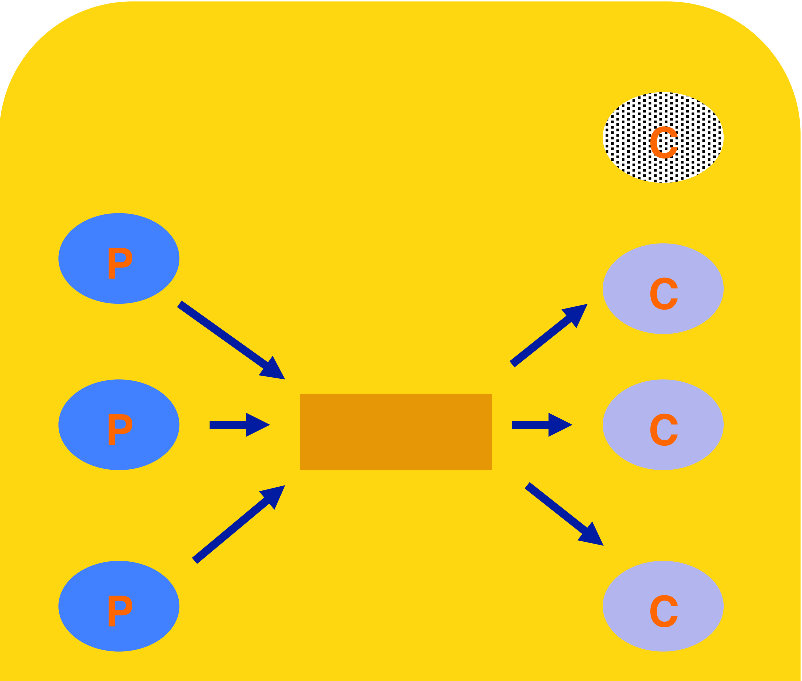
Multimedia Converter

SMS gateway

Message Queue: Multi Processing

Beolink.org

- ❑ Parallel processing
- ❑ Load Balancing
- ❑ High Availability
- ❑ Elastic
- ❑ Maintenance operation



Es.

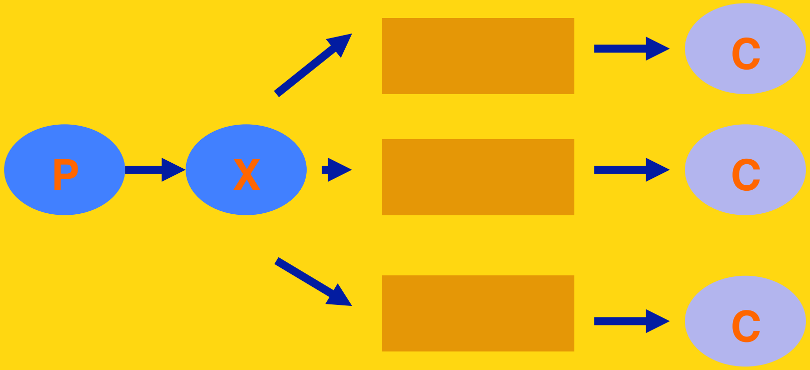
Image converter

Billing Event

User Provisioning

Message Queue: Pub/Sub

- ❑ Sending messages to many consumers at once
- ❑ Event Driven



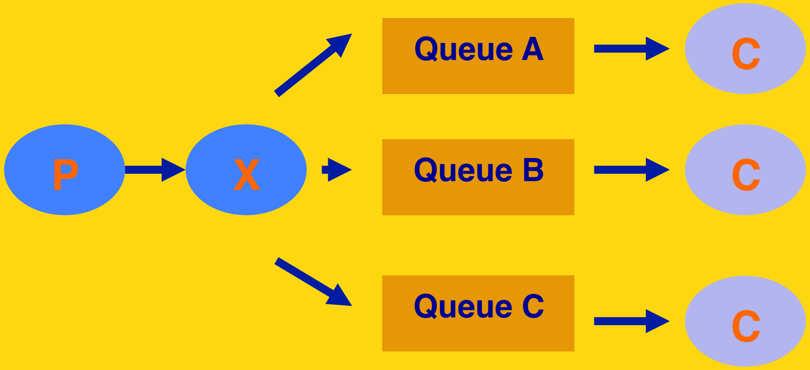
Es.

Push notification

Chat Room

Message Queue: Routing

- ❑ Static with routing key
- ❑ Pattern base
  - ❑ Pattern topic
  - ❑ Dynamic with header evaluation



Es.

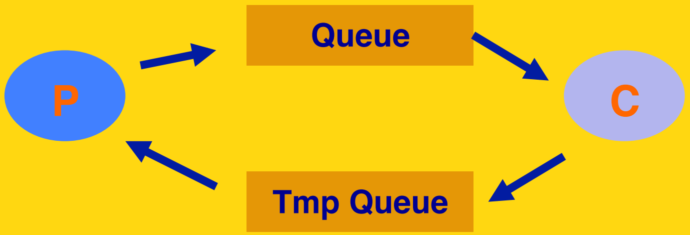
Logging collector

User Provisioning on
target System

Info Sync

Message Queue: RPC

- ❑ Remote Procedure Call
  - ❑ Single queue for Consumer
  - ❑ One queue for each Producer
  - ❑ Reply to options

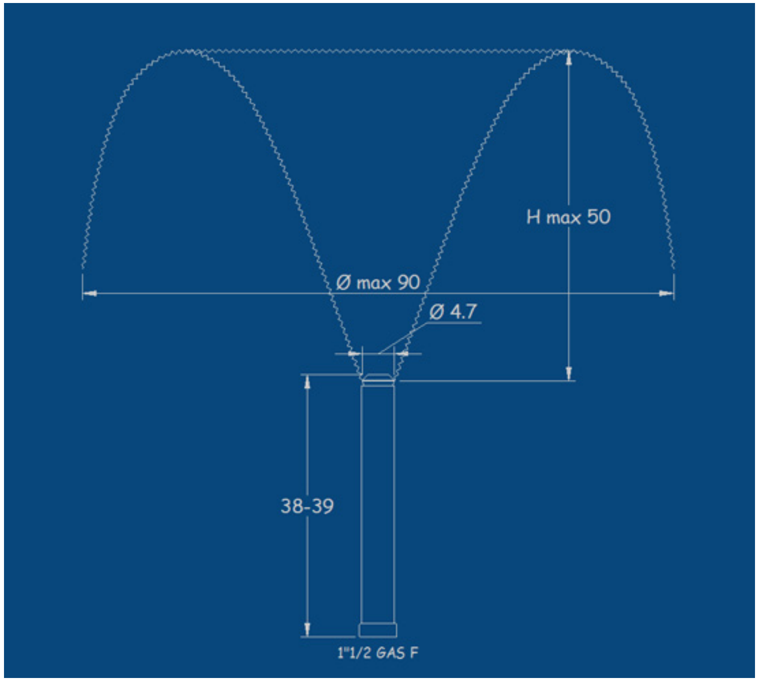


Es.

Distributed Scheduler
CallBack
WAN comunication

Message : More …

- ❑ Persistent Message
- ❑ Queue
  - ❑ Priority / Re ordering
  - ❑ Message Group
  - ❑ QOS / rating
- ❑ Deduplication
- ❑ Broker Network
  - ❑ Cluster
  - ❑ Load distribution over WAN
  - ❑ Message routing



Simple solution to a complicated problem!

Implementation

- ❑ **Internal implementation**
  - ❑ Python (Queue), Perl (Thread::Queue) ...
- ❑ **Nosql Based**
  - ❑ Redis, MongoDB, Memcache …
- ❑ **Framework**
  - ❑ Generic application framework: Gearman
  - ❑ Stomp Based: ActiveMQ, Apollo…
  - ❑ AMQP Based:RabbitMQ, Qpid…
  - ❑ Other : kafka…
- ❑ **Alternative solutions**
  - ❑ Broker less (0MQ, Crossroads I/O)
- ❑ **Services**

Implementation:Message Format

- ❑ **Internal / Object**

- ❑ **STOMP**

  Simple (or Streaming) Text Oriented Message Protocol (STOMP) is a simple text-based protocol, designed for working with Message Oriented Middleware

- ❑ **AMQP**

  Advanced Message Queuing Protocol is an application layer protocol, designed to efficiently support a wide variety of messaging applications and communication patterns.

- ❑ **XMPP**

  Extensible Messaging and Presence Protocol

- ❑ **JSON**

  JavaScript Object Notation, is a text-based

Implementation: NoSQL

Redis Internal Function

```
self.redis = redis.StrictRedison (…)

def send(self,queue,message):
 self.redis.rpush(queue,message)

def recv(self,queue)
 return self.redis.blpop(queue)
```

Queue Name	= KEY
Message	= Value
Queue	= List
Notify	= Event

RestMQ

The HTTP operation on url:
/queue/<queuename>

Post Message

```
{
  "cmd": "add",
  "queue": "genesis",
  "value": "abacab"
}
```

Get Message

```
{
  "cmd": "take",
  "queue": "genesis"
}
```

The message can be formatted as a json object

Amazon SQS

Demo sub/pub :https://gist.github.com/348262

Implementation: AMQP

RabbitMQ

Producer

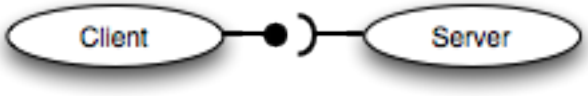
```
#!/usr/bin/env python
import pika

connection =
pika.BlockingConnection(pika.ConnectionParameters(
        host='localhost'))
channel = connection.channel()

channel.queue_declare(queue=’myqueue')

channel.basic_publish(exchange='',
                      routing_key=’myqueue',
                      body=’message 1 ')
print " [x] Sent ’Message 1"
connection.close()
```

Consumer

```
#!/usr/bin/env python
import pika

connection =
pika.BlockingConnection(pika.ConnectionParameters(
        host='localhost'))
channel = connection.channel()

channel.queue_declare(queue=’myqueue')

print ' [*] Waiting for messages. To exit press CTRL+C'

def callback(ch, method, properties, body):
    print " [x] Received %r" % (body,)

channel.basic_consume(callback,
                      queue=’myqueue',
                      no_ack=True)

channel.start_consuming()
```

Implementation: Alternatives

ZeroMQ



Producer

```
#!/usr/bin/env python

import zmq
context = zmq.Context()
socket = context.socket(zmq.REQ)
socket.bind("tcp://127.0.0.1:5000")

while True:
   msg =”my msg”
   socket.send(msg)
   print ”Send", msg
   msg = socket.recv()
```

Consumer

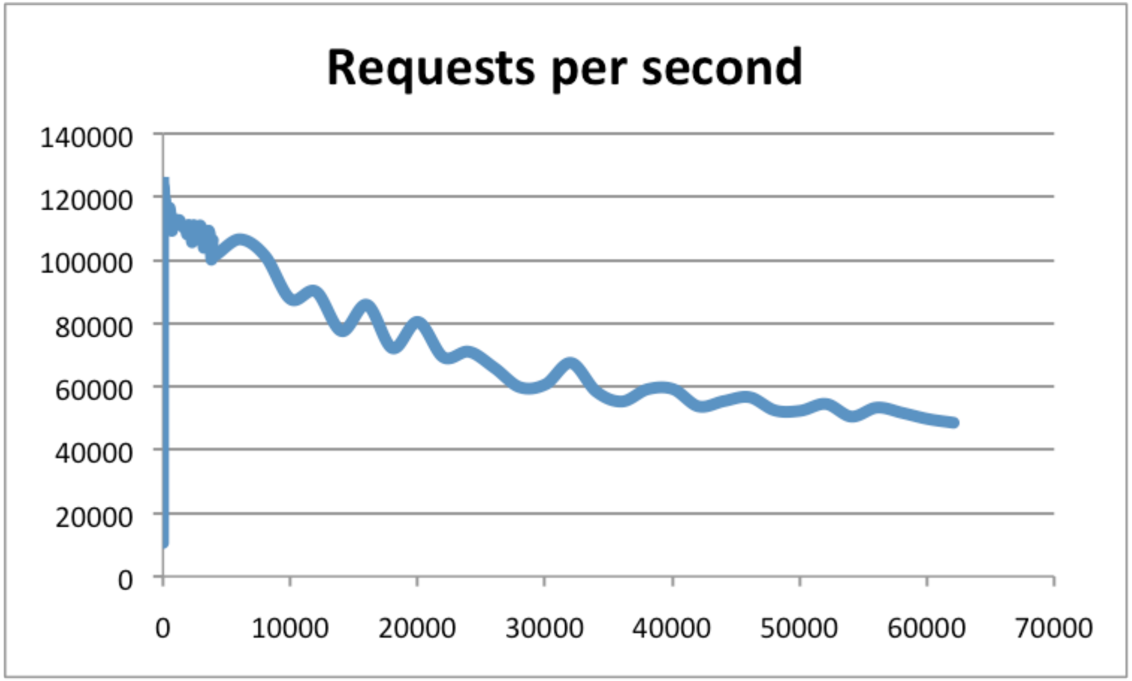
```
#!/usr/bin/env python

import zmq
context = zmq.Context()
socket = context.socket(zmq.REP)
socket.bind("tcp://127.0.0.1:5000")

while True:
   msg = socket.recv()
   print "Got", msg
   socket.send(msg)
```

…, but it is not fast enough …

Performance: Redis

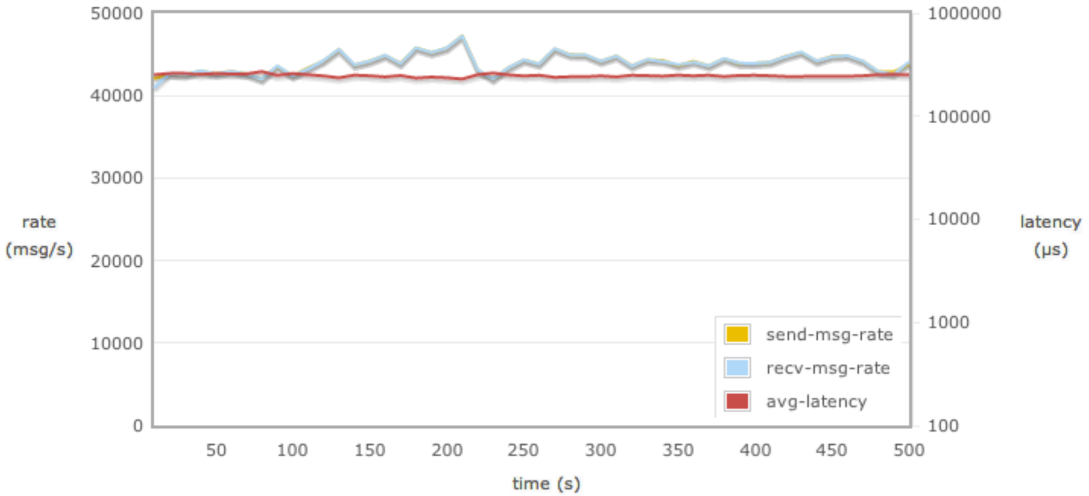
The Linux box is running Linux 2.6, it's Xeon X3320 2.5 GHz.

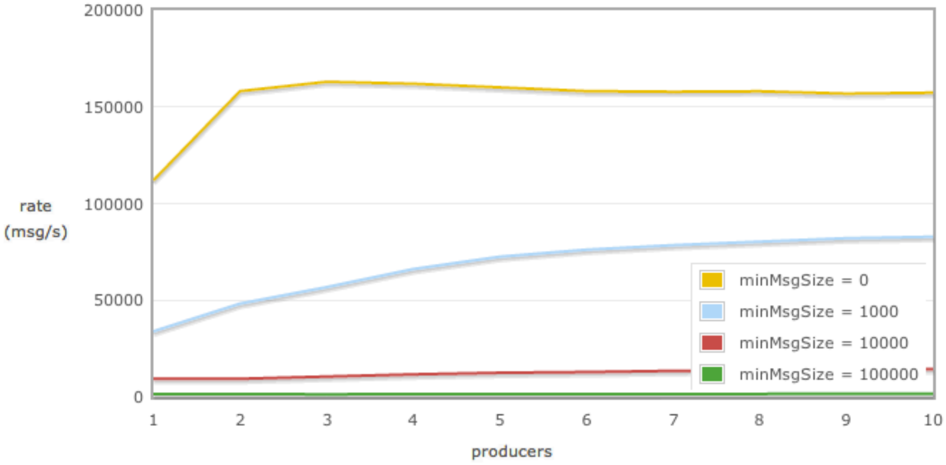
Text executed using the loopback interface (127.0.0.1).



Performance: RabbitMQ

PowerEdge R610 with dual Xeon E5530s and 40GB RAM





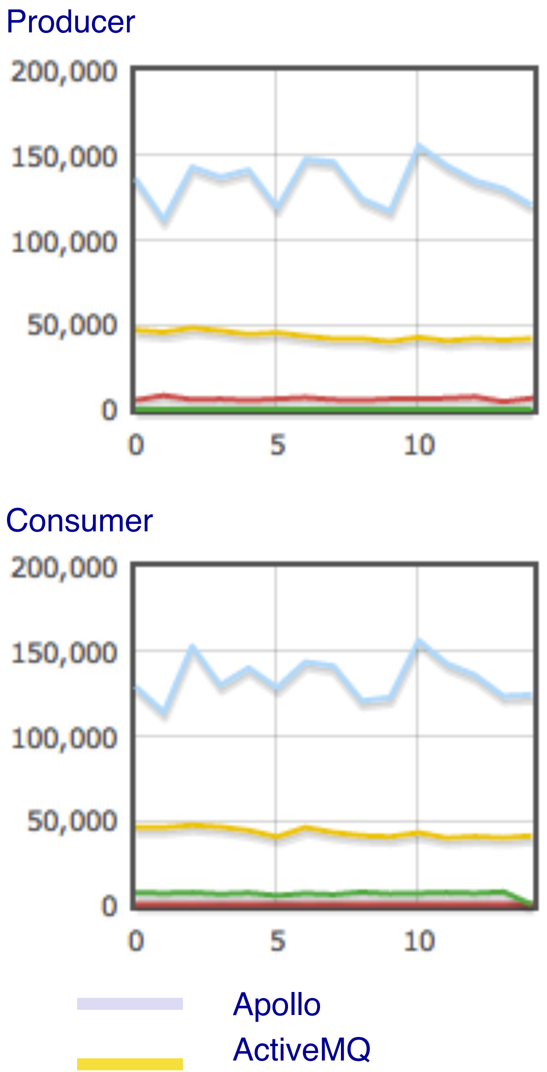
Performance: ActiveMQ/Apollo

EC2 High-CPU Extra Large Instance EC2 xlarge

7 GB of memory
20 EC2 Compute Units (8 virtual cores with 2.5 EC2 Compute Units each)
model name : Intel(R) Xeon(R) CPU E5506 @ 2.13GHz

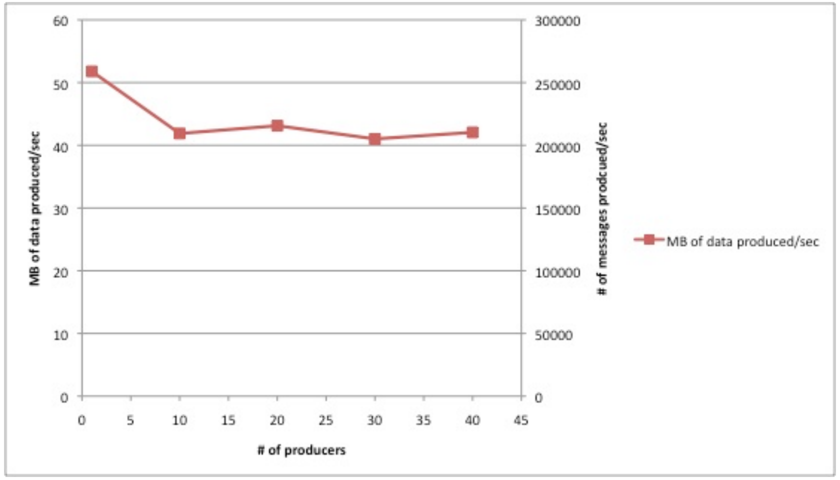
OS: Amazon Linux 64bitLinux ip-10-70-206-42 2.6.35.14-97.44.amzn1.

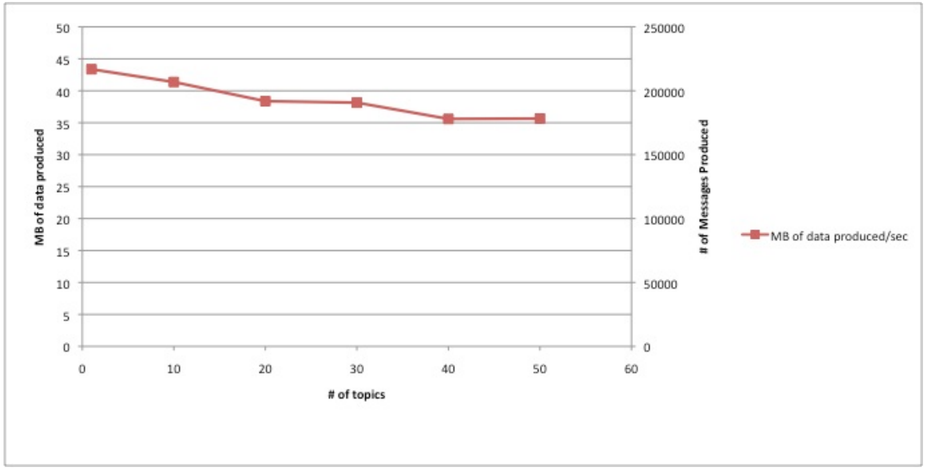
5 Consumer
5 Producer



Performance: kafka

message size = 200 bytes
batch size = 200 messages
fetch size = 1MB
flush interval = 600 messages

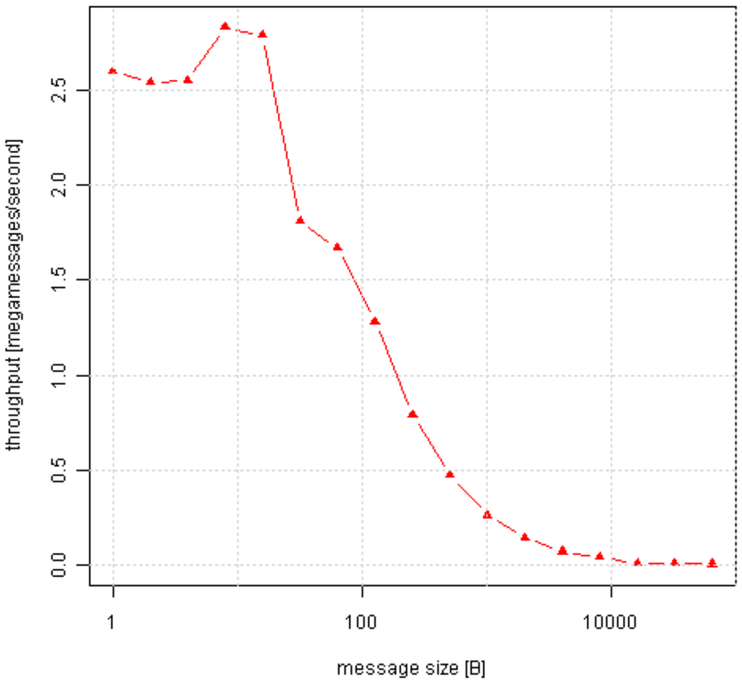




Performance: zeroMQ

Box 1:
8-core AMD Opteron 8356, 2.3GHz
Mellanox ConnectX MT25408 in 10GbE mode
Linux/Debian 4.0 (kernel version 2.6.24.7)
ØMQ version 0.3.1

Box 2:
8-core Intel Xeon E5440, 2.83GHz
Mellanox ConnectX MT25408 in 10GbE mode
Linux/Debian 4.0 (kernel version 2.6.24.7)
ØMQ version 0.3.1



Throughput gets to the maximum of 2.8 million messages per second for messages 8 bytes long

Performance

Performance

- ❑ **Persistence**

  message can fault down to hundreds of message per Second

- ❑ **Bandwidth**

  Message size and Acknowledge increase the usage of bandwidth

- ❑ **Topics**

  The routing based on the value of header, increase the delay

- ❑ **Queue**

  Number of queue increase the delay

- ❑ **Cluster**

  Replication message increase the time for the acknowledgement

Big Data

Big data spans three dimensions

- ❑ **Volume**

  Enterprises are awash with ever-growing data of all types, easily amassing terabytes—even petabytes—of information.

- ❑ **Velocity**

  Sometimes 2 minutes is too late. For time-sensitive processes such as catching fraud, big data must be used as it streams into your enterprise in order to maximize its value.

- ❑ **Variety**

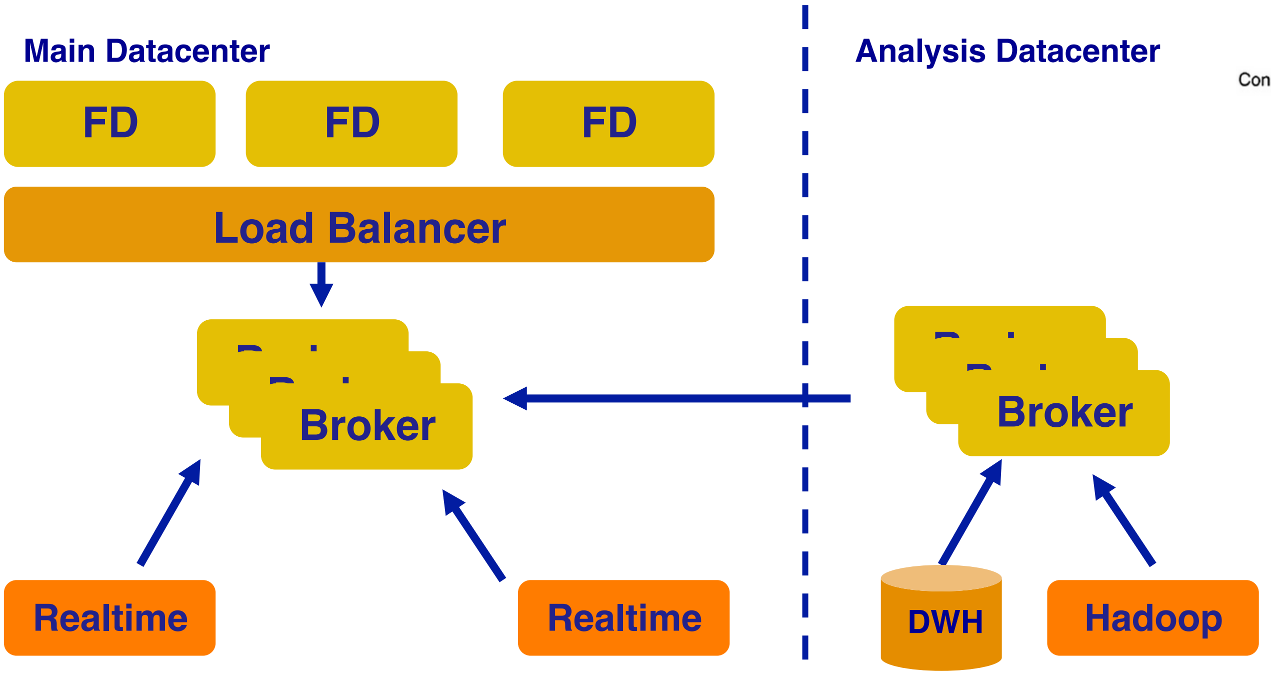
  Big data is any type of data - structured and unstructured data such as text, sensor data, audio, video, click streams, log files and more.

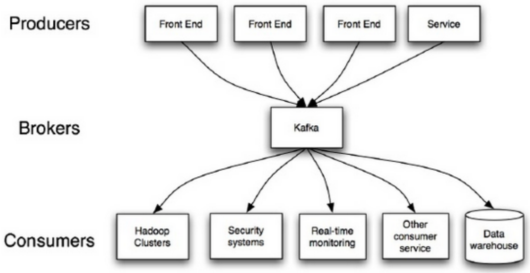
Big Data

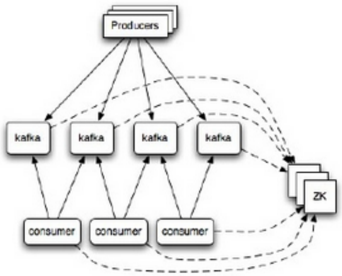
Big Data	Message Queue
Volume	Load Balancing: - with Multi Brokers Conf - with Multi queues Conf
Velocity	Parallel Processing - Balance base on time spent - Increase capacity on demand High Availability
Variety	Routing Key Path - Header analysis - Topic

Big Data: Linkedin

End User action Tracking
Operational Metrics
Forntend (Producer)= 100Mb/s
Bakend (Consumer) = 200Mb/s

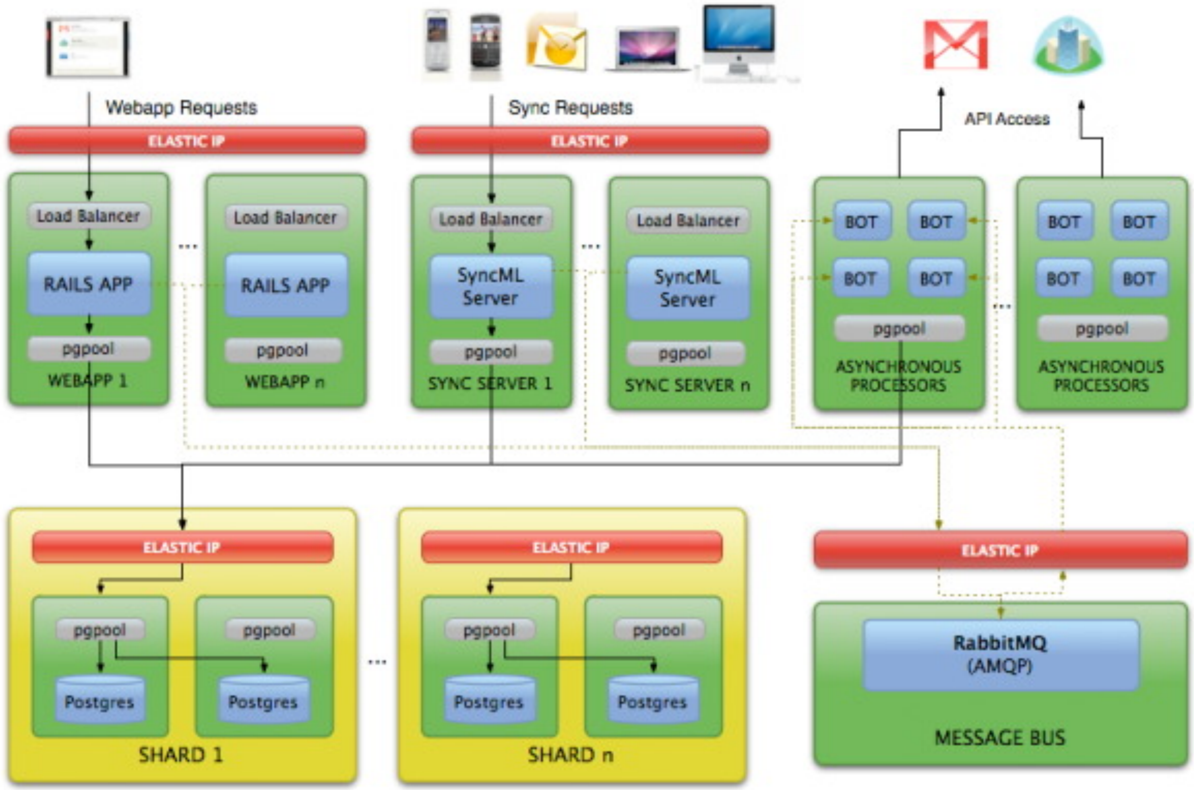






http://incubator.apache.org/kafka/design.html
http://research.microsoft.com/en-us/um/people/srikanth/netdb11/netdb11papers/netdb11-final12.pdf

Big Data: Soocial.com

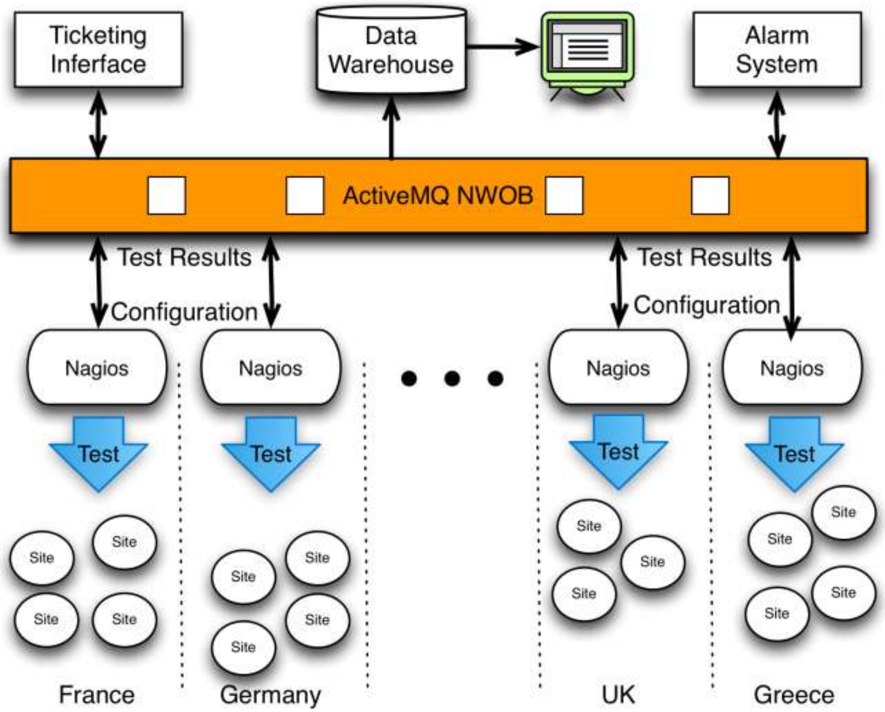


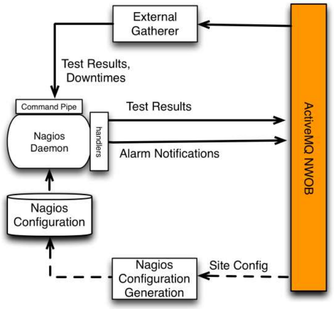
http://aws.typepad.com/aws/2008/12/

Synchronization btw different applications

Collect tracking data

Big Data: CERN



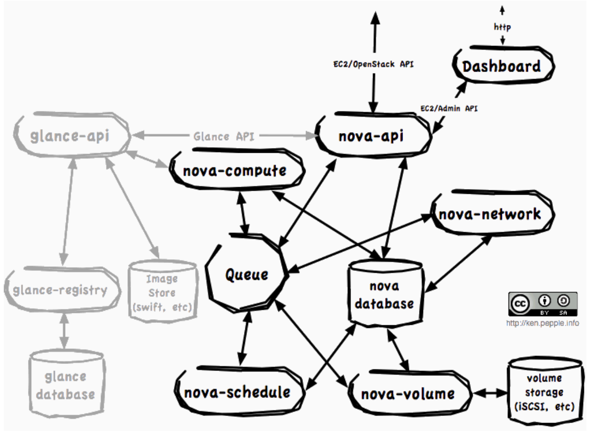


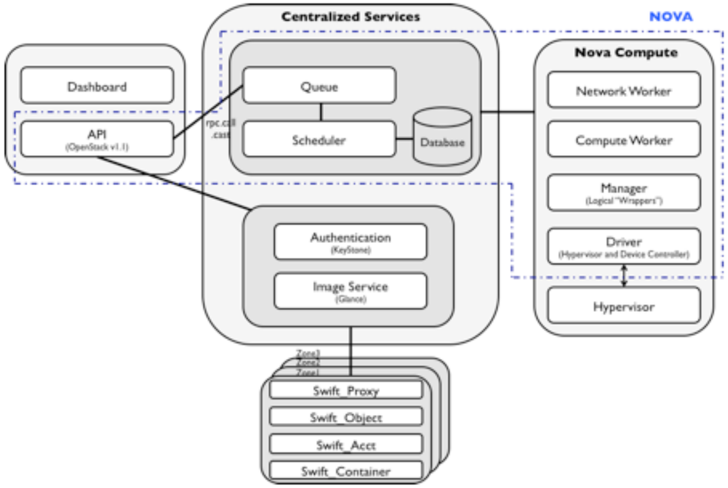
fusesource.com/collateral/download/82/

Monitor reliability and availability of European distributed computing Infrastructure (GRID)

100K system Monitored

Big Data: OpenStack

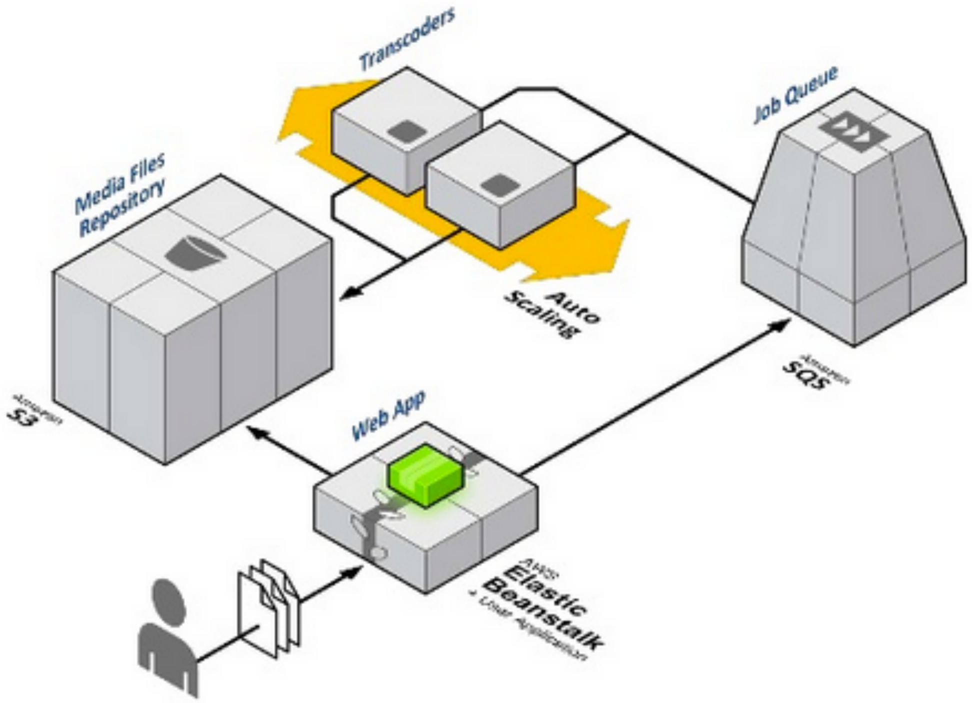




OpenStack Coordination and provisioning

RackSpace Cloud Management Tasks

Cloud Computing: Netflix



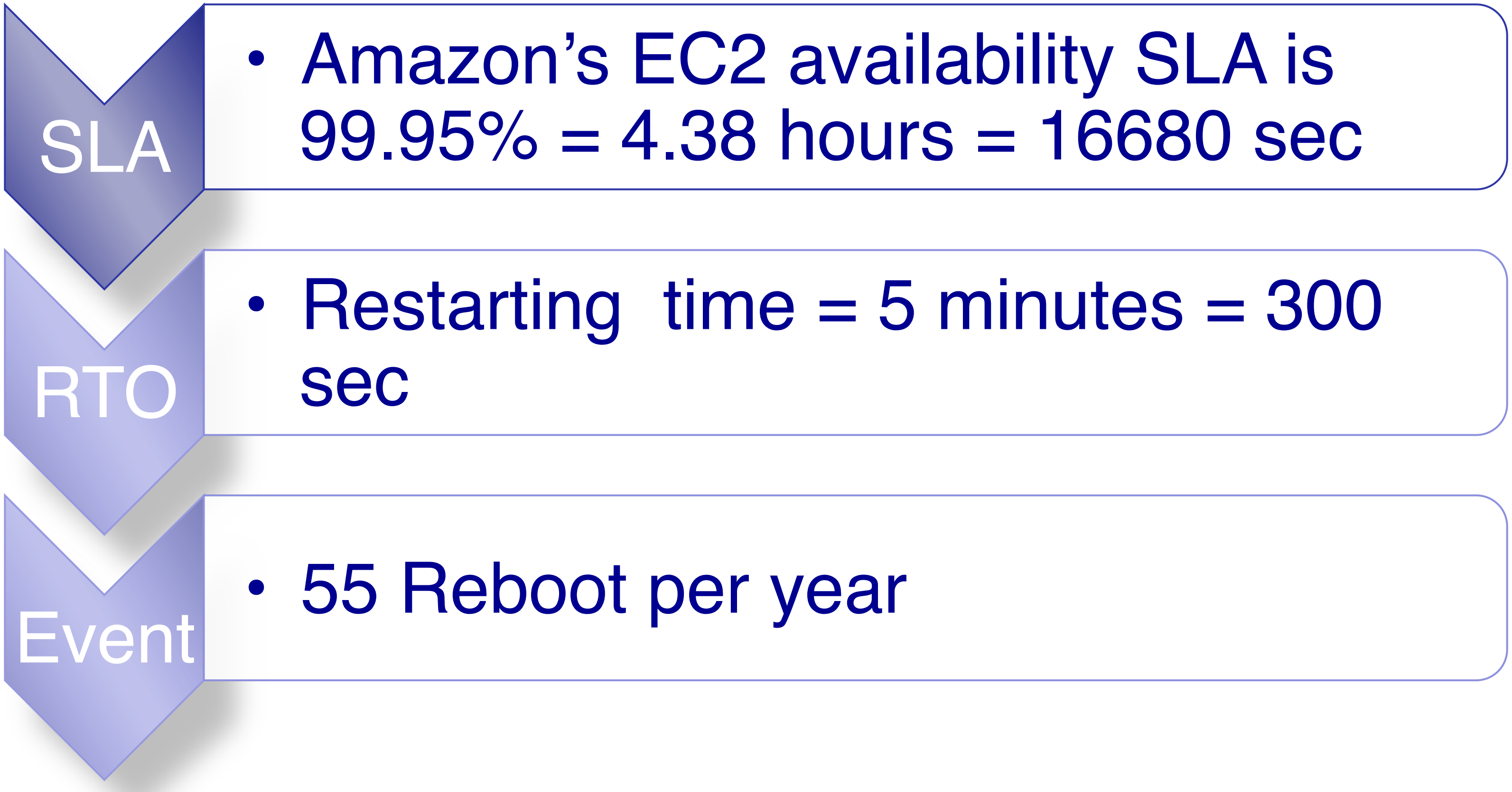
Transcoding media conversion for different device and channels

“Everything fails all the time”

Werner Vogels

CTO of Amazon

Cloud Computing: SLA



SLA

- Amazon’s EC2 availability SLA is 99.95% = 4.38 hours = 16680 sec

RTO

- Restarting time = 5 minutes = 300 sec

Event

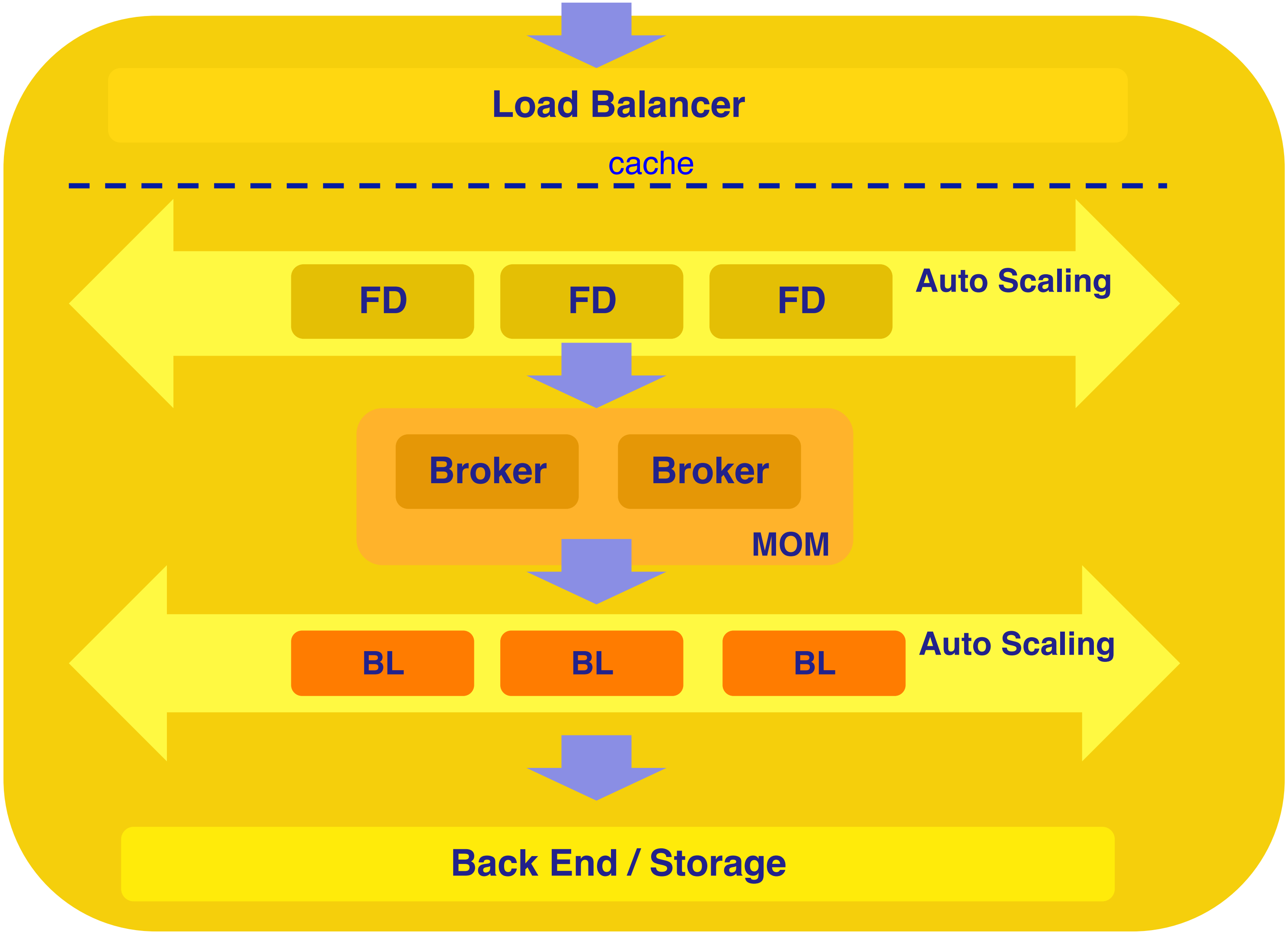
- 55 Reboot per year

No one declare the MTBF !!!

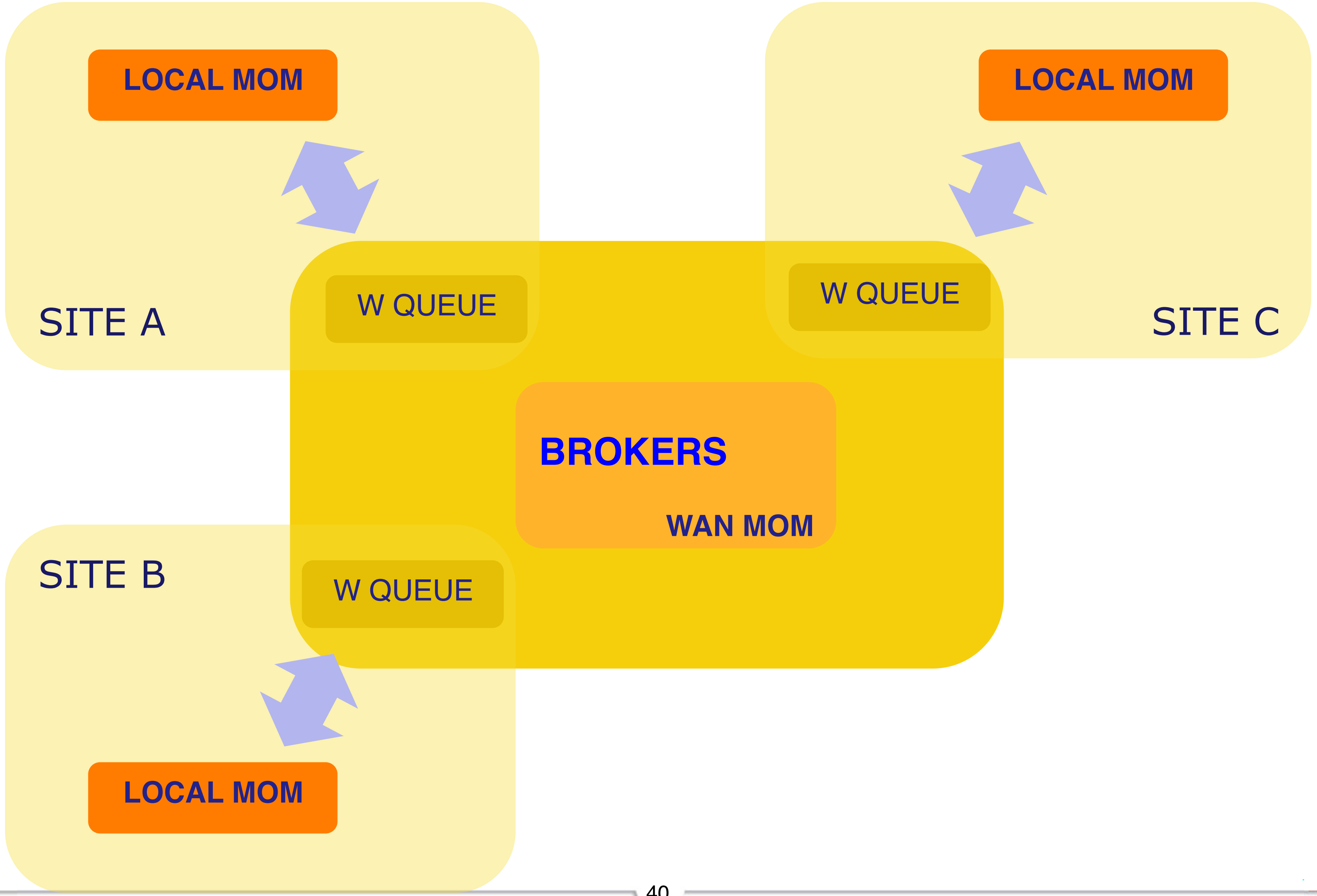
Design your application architecture for failure. Don’t look for alternatives

…split your applications into different components and, make sure every component of your application has redundancy with no common points of failure…

Cloud Computing: Architecture



Cloud Computing: Multi site



High Availability

Cloud Computing: Scalability / HA

❑ Master-Salve topology

queue is assigned to a master node, and all changes to the queue are also replicated to a salve node. If the master has failed, the slave can take over. (e.g. Qpid and ActiveMQ, RabbitMQ).

❑ Queue Distribution

queues are created and live in a single node, and all nodes know about all the queues in the system. When a node receives a request to a queue that is not available in the current node, it routes the request to the node that has the queue. (e.g. RabbitMQ)

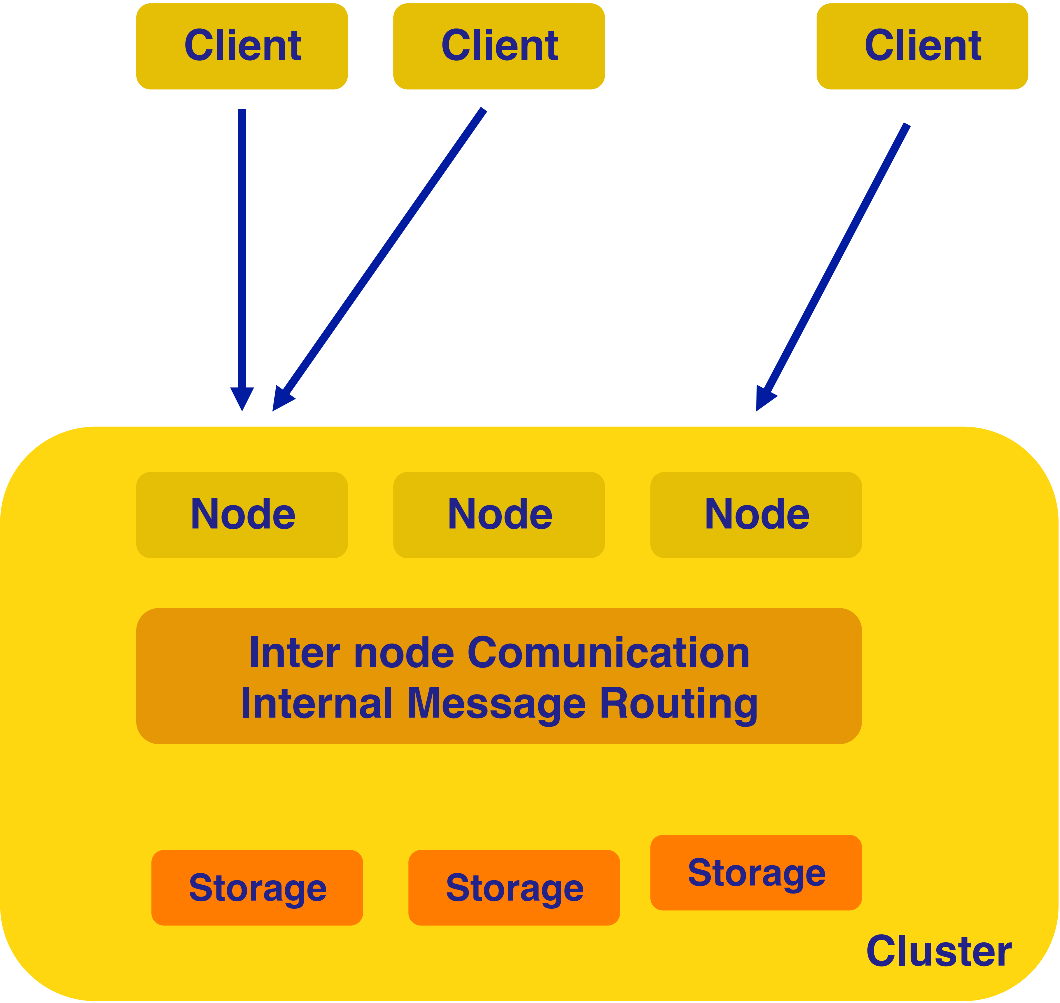
❑ Cluster Connections

Clients may define cluster connections giving a list of broker nodes, and messages are distributed across those nodes based on a defined policy (e.g. Fault Tolerance Policy, Load Balancing Policy). It also supports message redistribution, and it plays a minor role in this setup.

❑ Broker networks

The brokers are arranged in a topology, and subscriptions are propagated through the topology until messages reach a subscriber. Usually, this uses Consumer priority mode where brokers that are close to the point of origin are more likely to receive the messages. The challenge is how to load balance those messages. (e.g. ActiveMQ)

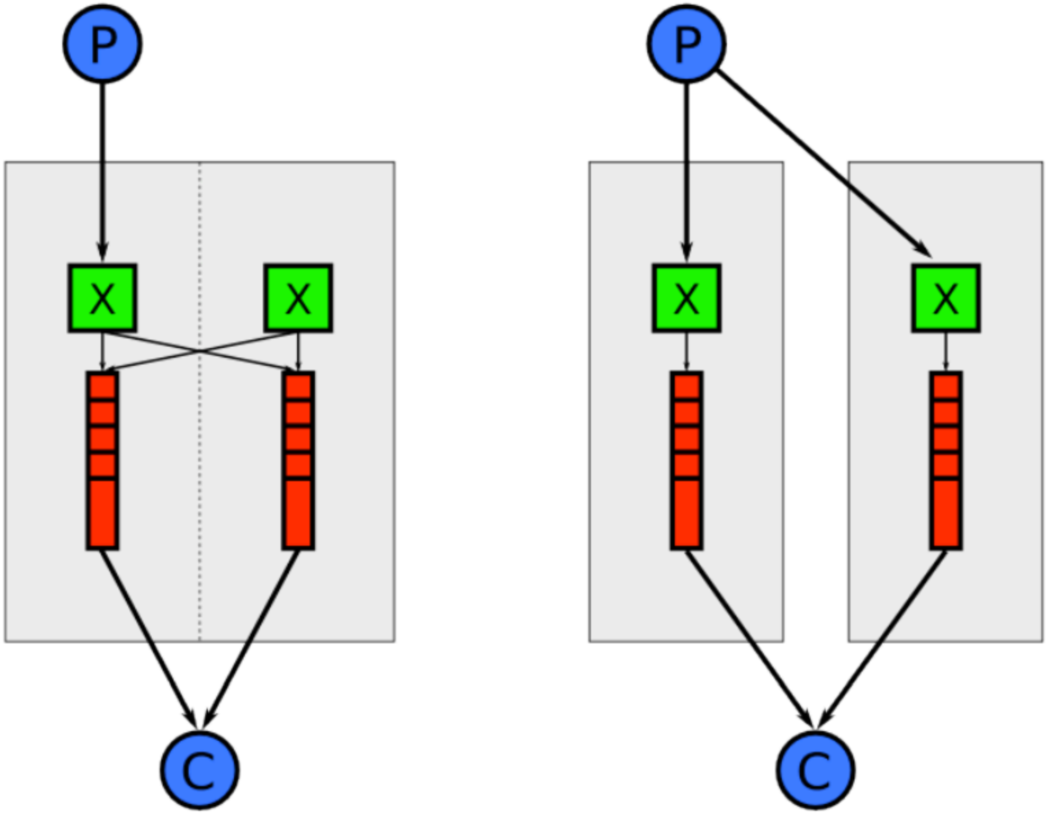
Cloud Computing: Cluster



Lookup

- Defined IP
- Multicast
- BootStrap
- Agent

Cloud Computing: Cluster



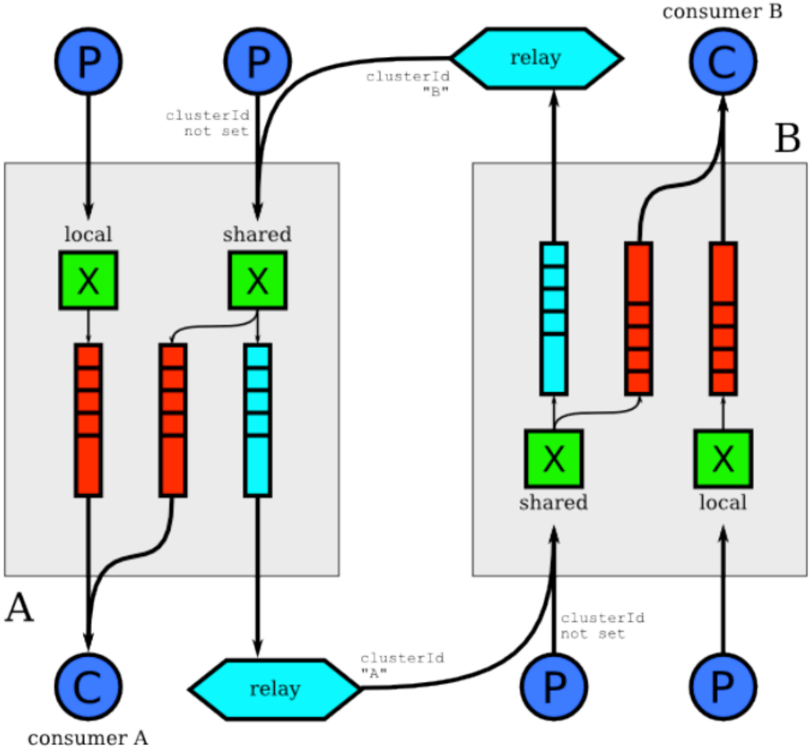
Configuration

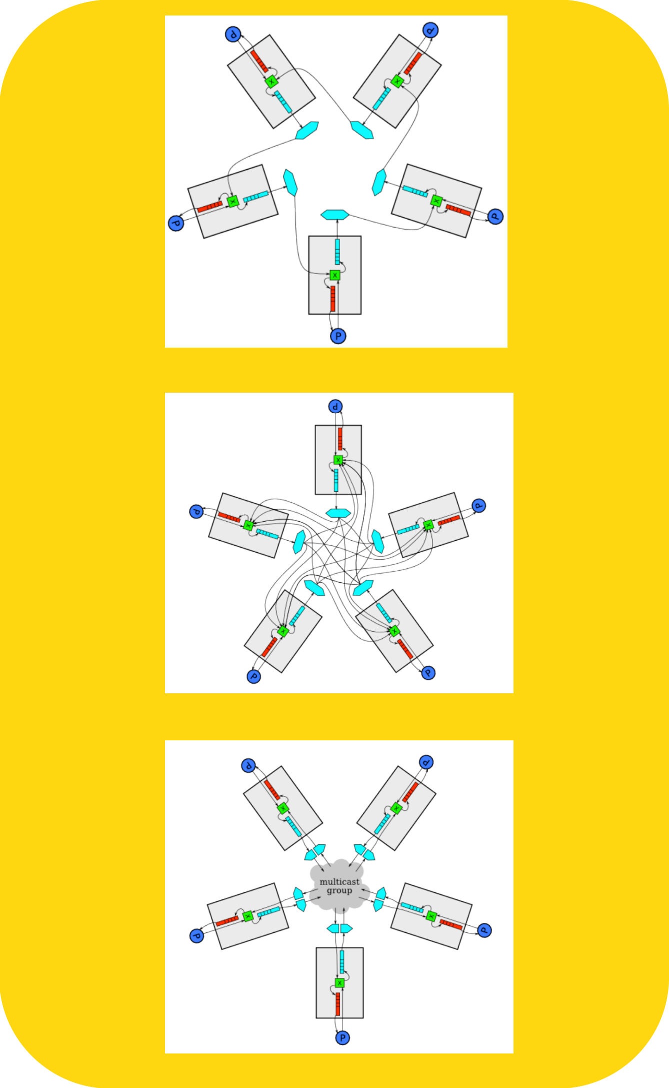
Two Cluster
with one node

Single Cluster
with two nodes

RabbitMQ:
http://skillsmatter.com/custom/presentations/talk4.rabbitmq_internals.pdf

Cloud Computing: Federation/ Shovel





RabbitMQ:
http://skillsmatter.com/custom/presentations/talk4.rabbitmq_internals.pdf

Are you happy?

Critical Points

- **Dimension**
  - Message Size
  - Number of Queue
  - Persistence
  - Delay of the queue
- **Persistence only when you need**
- **Cluster on client side or via boostrap**
- **Acknowledge when you need**
- **Topic vs Queue**
- **Queue Length**
- **Performance Test**

$E = (P/C-1)*T$

$L = (P-C)*T$

More complex

$$\ln(P(\mathbf{h}|\boldsymbol{\theta}, H_1)) = \ln\left[\left(\prod_{i=1}^{\tau-1}\frac{e^{-\lambda_{11}}\lambda_{11}^{h_i}}{h_i!}\right)\left(\prod_{i=\tau}^{n}\frac{e^{-\lambda_{12}}\lambda_{12}^{h_i}}{h_i!}\right)\right]$$

$\rho = \lambda / \mu$

Exponential probability density

$$T = \frac{1}{\mu - \lambda}$$

$$N = \frac{\rho}{1-\rho}$$

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$$

All customers have the same value

Any arbitrary probability distribution

$$P_0(t) = e^{-\lambda t}$$

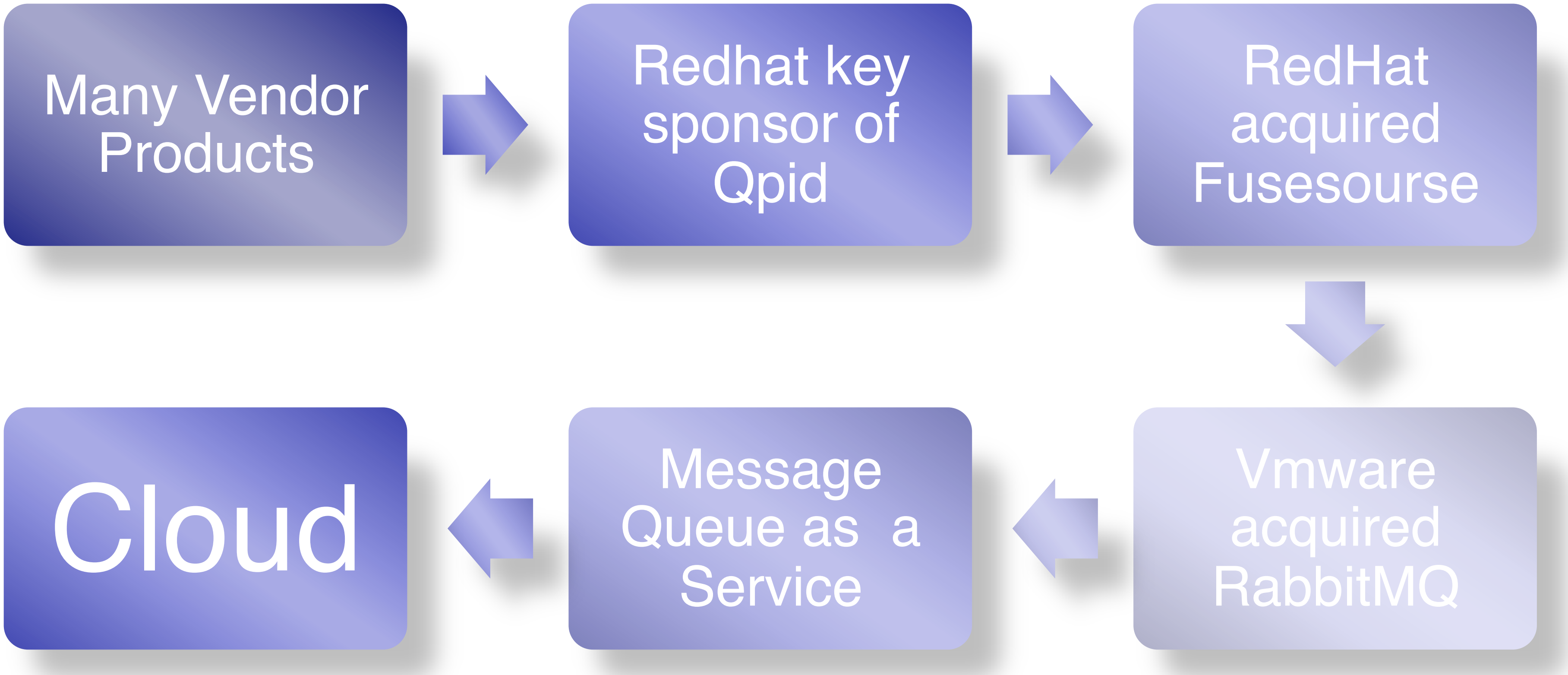
Processing Delay

Transmission Delay

Propagation Delay

$$\ln(P(\mathbf{h}|\boldsymbol{\theta},H_0)) = \ln\left(\prod_{i=1}^{n}\frac{e^{-\lambda_0}\lambda_0^{h_i}}{h_i!}\right) = -n\lambda_0 + \ln(\lambda_0)^* \sum_{i=1}^{n} h_i - \sum_{i=1}^{n} \ln(h_i!)$$

Last but not ..



Many Vendor Products

Redhat key sponsor of Qpid

RedHat acquired Fusesourse

Vmware acquired RabbitMQ

Message Queue as a Service

Cloud

Conclusion

The science of programming:

"…make building blocks that people can understand and use *easily*, and people will work together to solve the very largest problems."

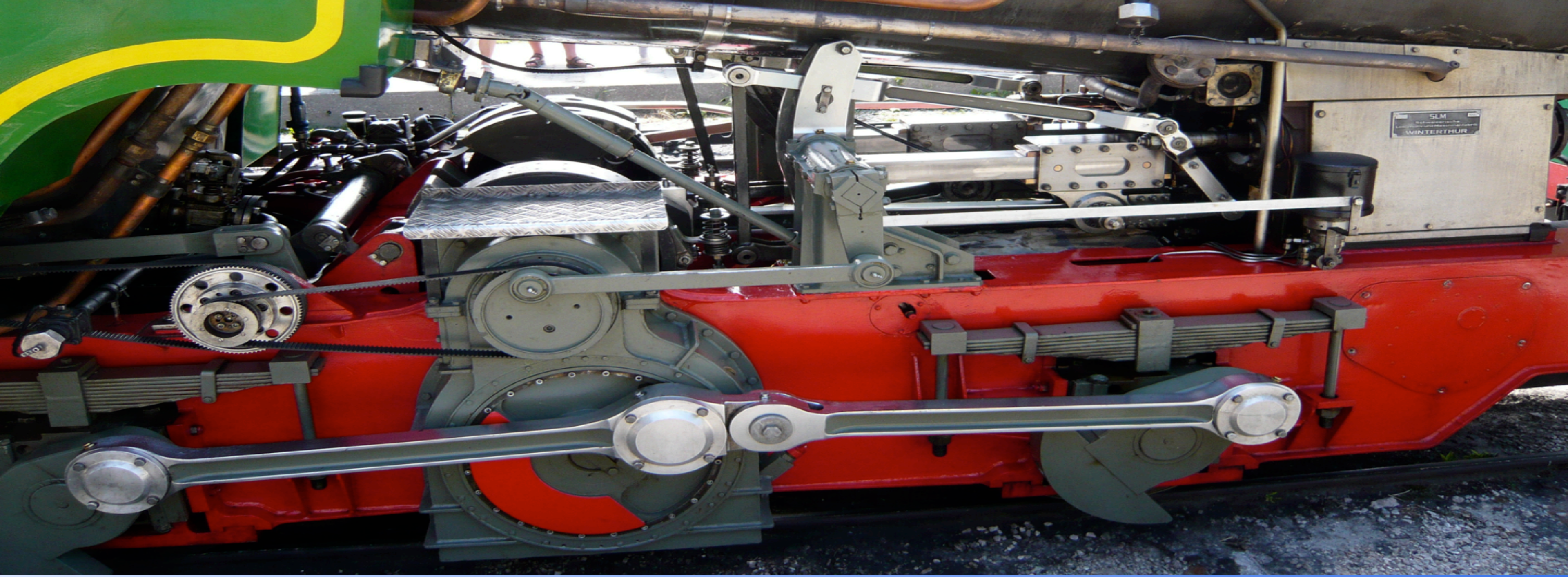
I look forward to meeting you…

XVII European AFS meeting 2012
University of Edinburgh
October 16th to 18th

Who should attend:

- Everyone interested in deploying a globally accessible file system
- Everyone interested in learning more about real world usage of Kerberos authentication in single realm and federated single sign-on environments
- Everyone who wants to share their knowledge and experience with other members of the AFS and Kerberos communities
- Everyone who wants to find out the latest developments affecting AFS and Kerberos

More Info: http://openafs2012.inf.ed.ac.uk/



Thank you

manfred@freemails.ch
http://www.beolink.org

Beolink.org

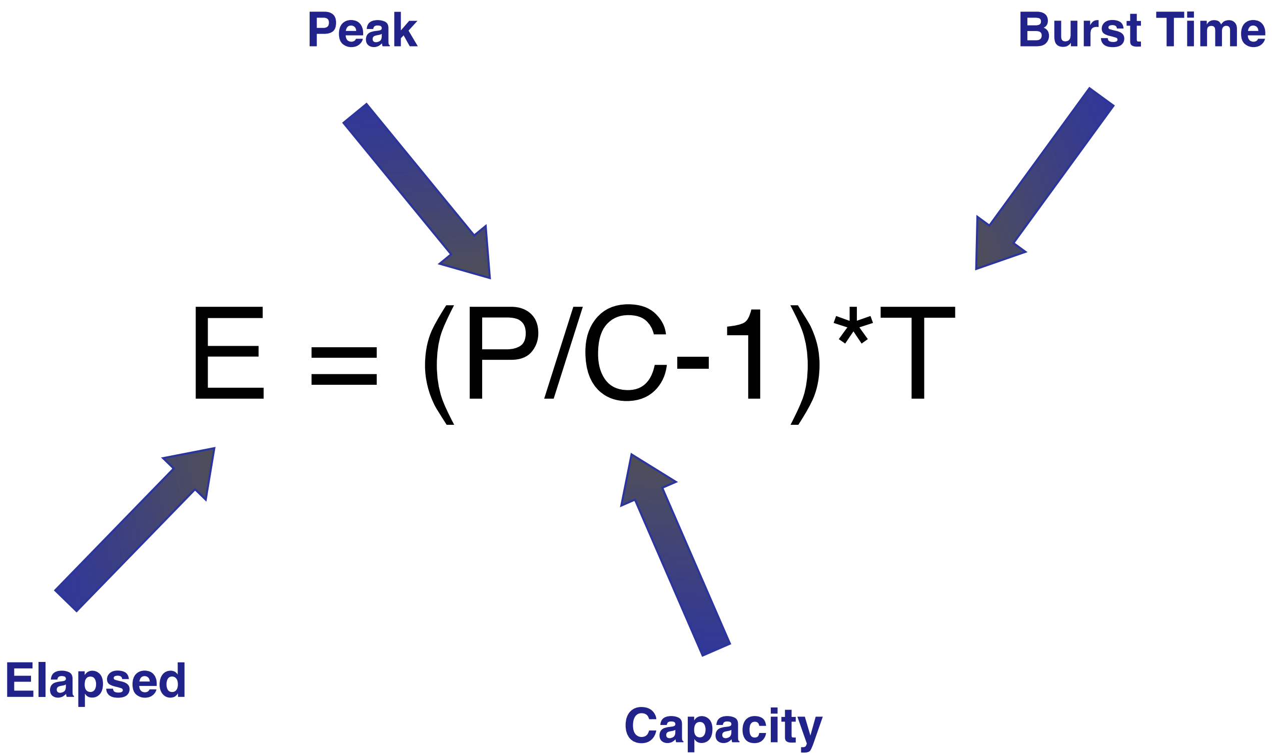
Message Queue Length

Which is the right size ?

…30% extra capacity

OFFER EXTENDED
There's still time to save!
30% OFF
your Gap, Old Navy or Banana Republic purchase

Basic Formula



Peak

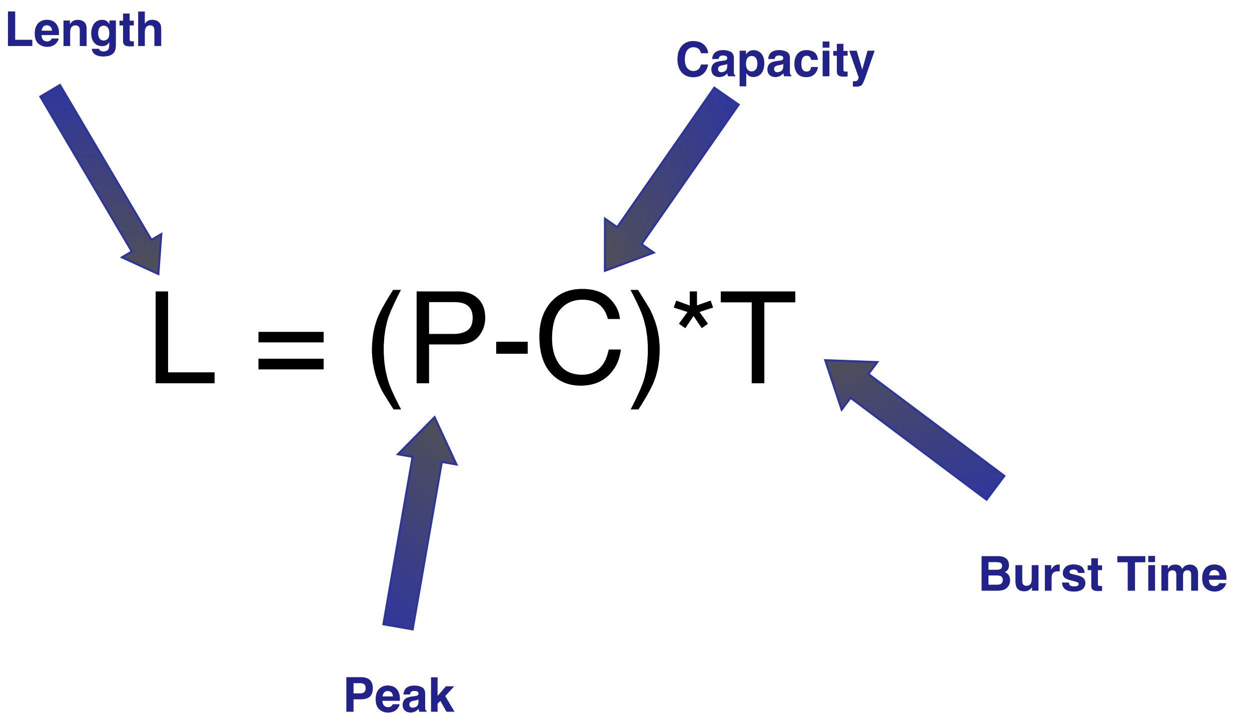
Burst Time

$$E = (P/C-1)*T$$

Elapsed

Capacity

Basic Formula



Length

Capacity

$$L = (P-C)*T$$

Burst Time

Peak