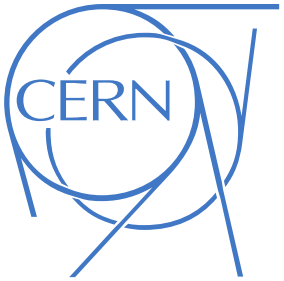


Crash Course on Messaging

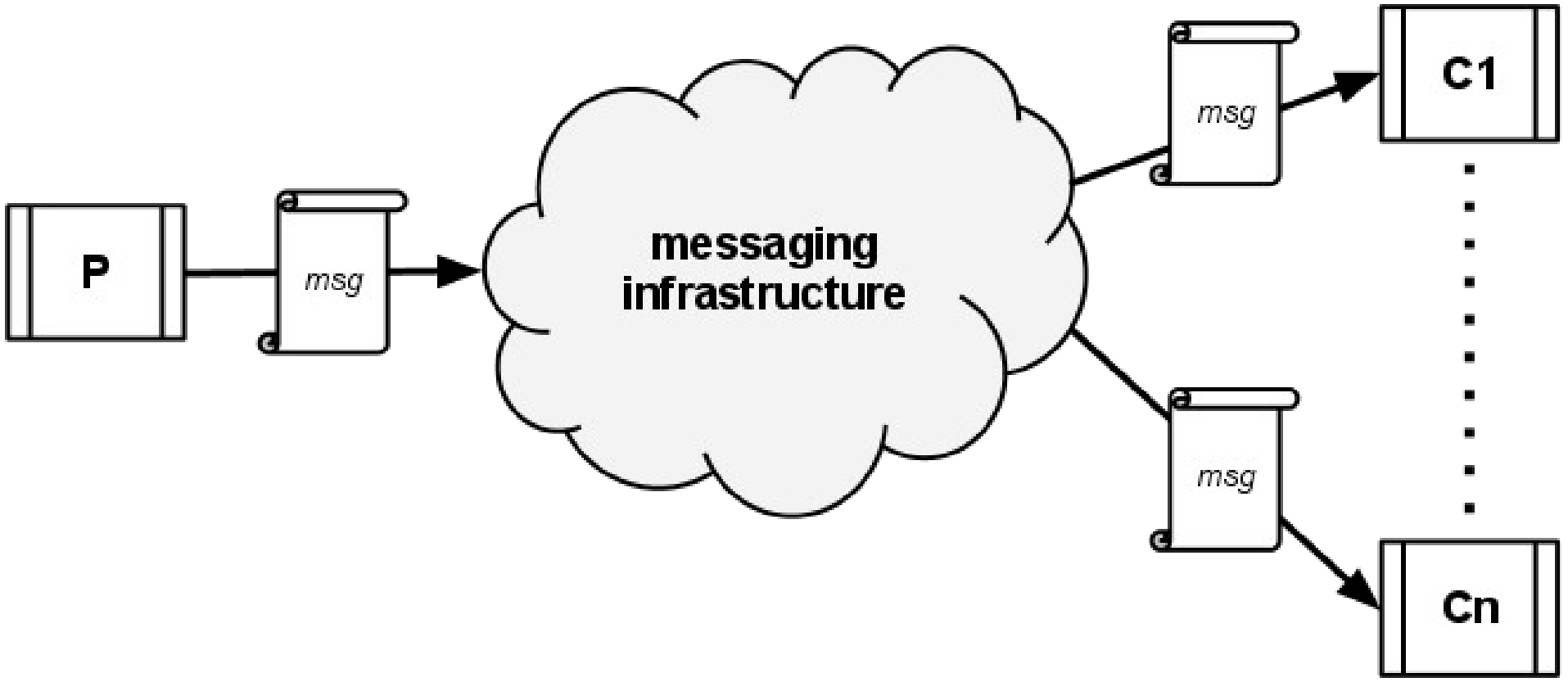
... everything you ever wanted to know about messaging
... and that can fit in 30 minutes

What is messaging?

Analogy: *messaging is for software components what electronic mail is for people*.

- Messages have a header and a body
  - header is used for routing and can be modified
  - body is opaque and immutable
- Messages go through intermediaries
  - multiple hops, delays, retries, failures...

10,000 feet overview



Debunking Common Myths

Things happen as you expect in *most* cases.

However, even if very unlikely:

- messages can be discarded or lost
- messages can be delivered multiple times
- messages can arrive in a different order
- messages can stay on the broker and not be delivered to you, even if you ask for them

JMS

- Java Message Service

  “is a messaging standard that allows application components to create, send, receive, and read messages”

- is mature (1.1 in 2002) and part of J2EE
- is (mainly) an API
- supports (only) two models:
  - point-to-point model (queues)
  - publish and subscribe model (topics)

STOMP

- Streaming Text Orientated Messaging Protocol

  "provides an interoperable wire format so that any of the available Stomp Clients can communicate with any Stomp Message Broker to provide easy and widespread messaging interop among languages, platforms and brokers"

- is (mainly) a wire protocol
- has a concise specification document
- 1.0 spec widely used but needs improvements

AMQP

- Advanced Message Queuing Protocol

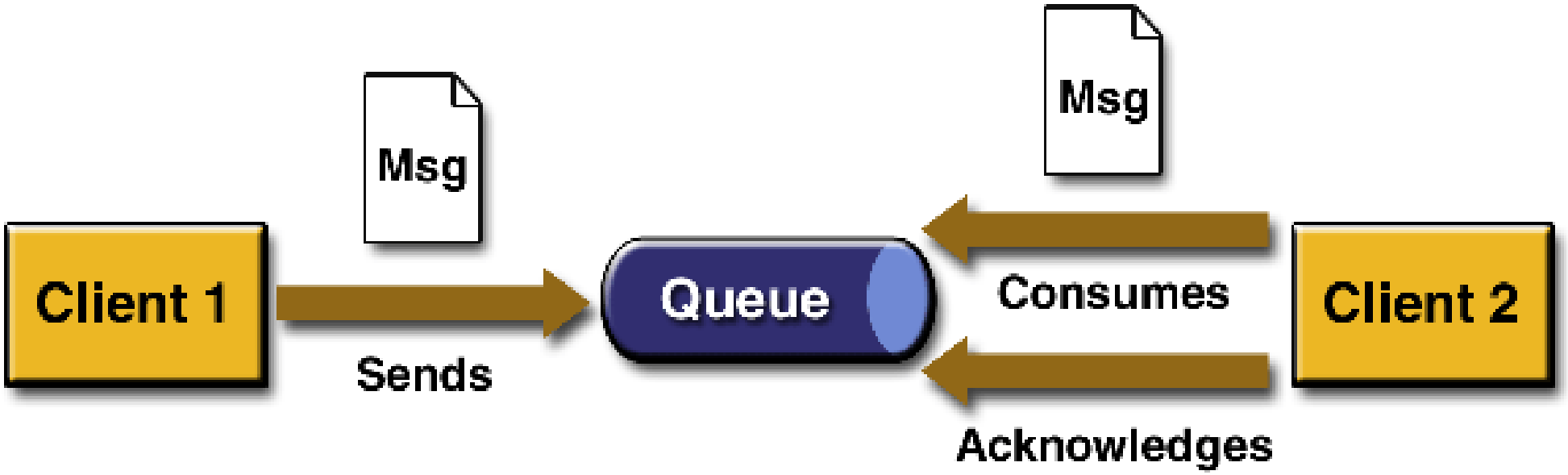
  “is an Open Standard for Messaging Middleware”

- includes:
  - a defined set of messaging capabilities (*AMQP Model*)
  - a network wire-level format
- is being developed by banks but also Cisco, Microsoft, Novell, Red Hat...
- is unfortunately not yet mature

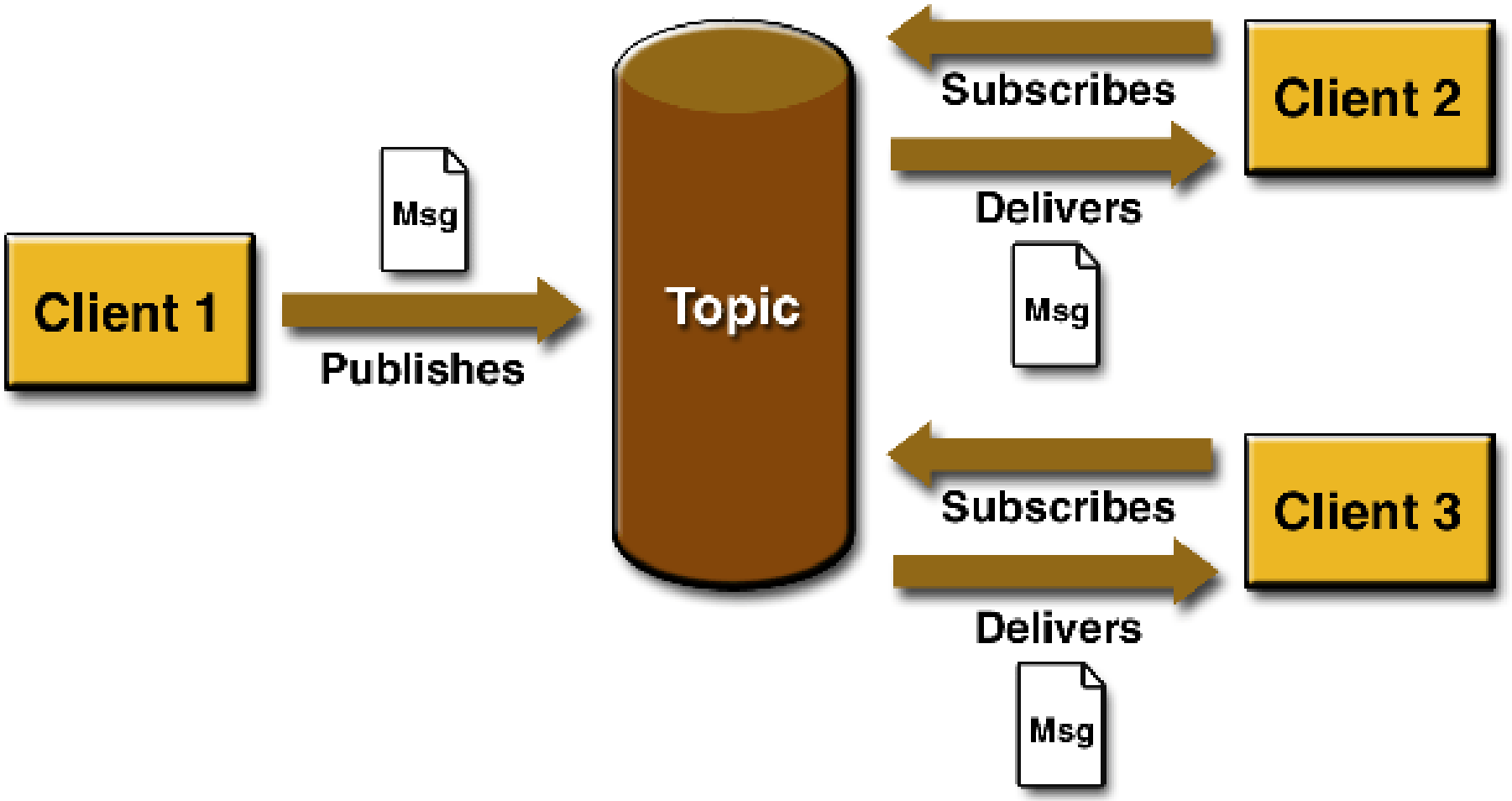
Messaging Models

- point-to-point = queue = load balancer
- publish/subscribe = topic
- AMQP 0-*
- RabbitMQ's AMQP 0-*
- AMQP 1-0

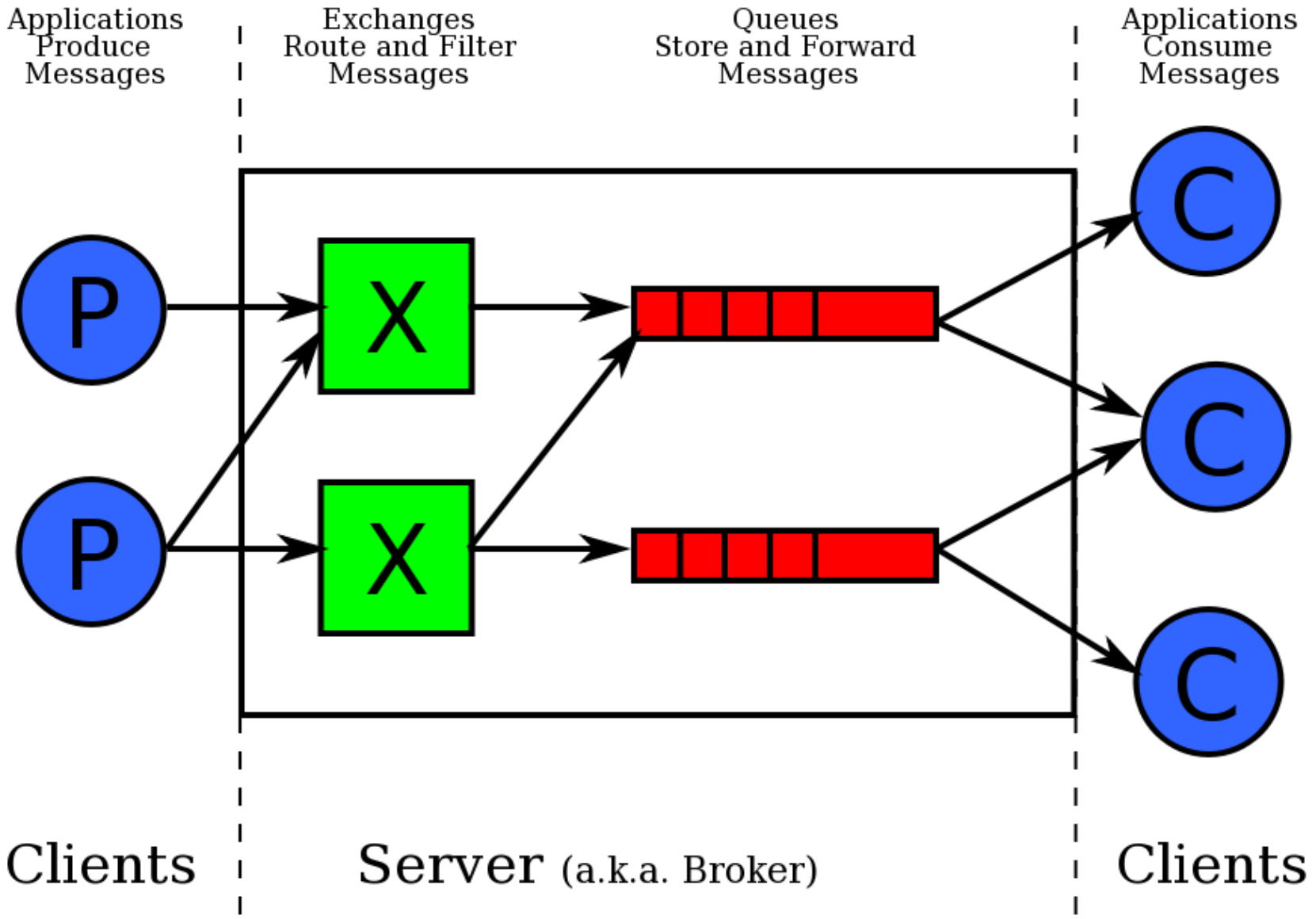
Point-To-Point Model



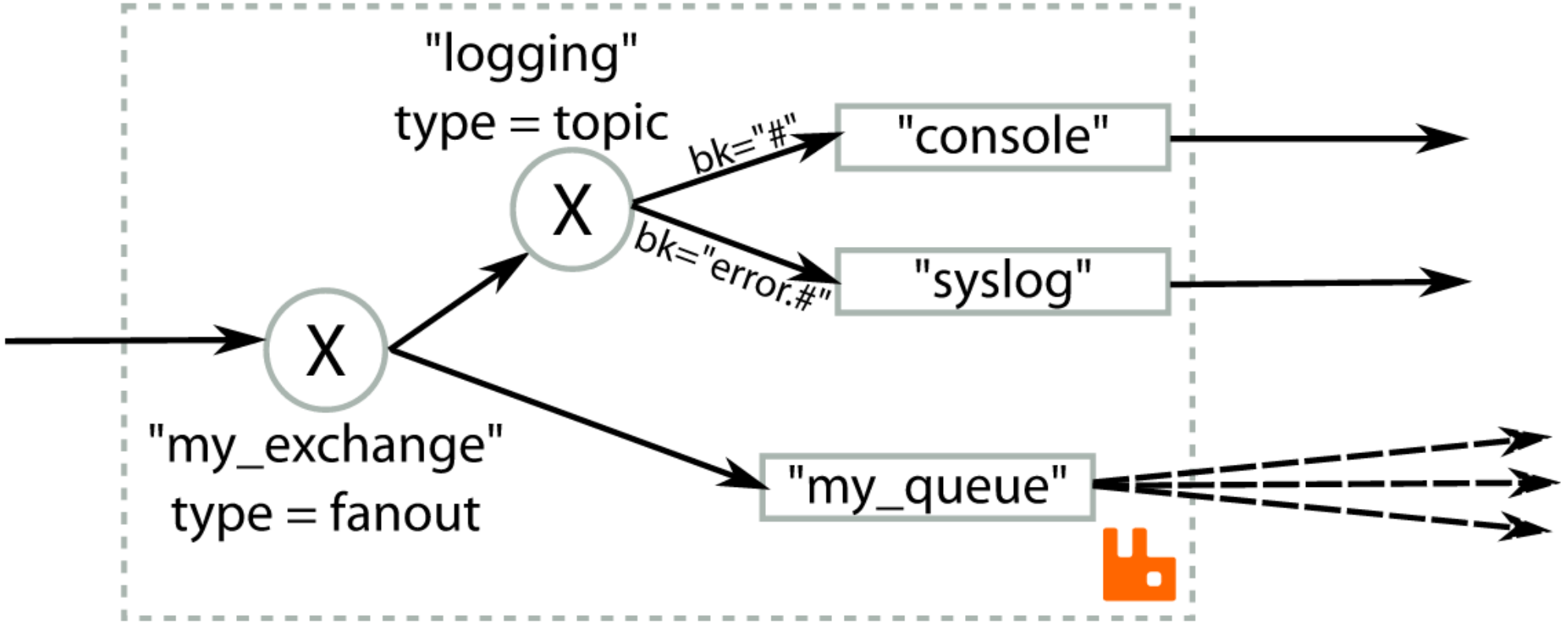
Publish/Subscribe Model



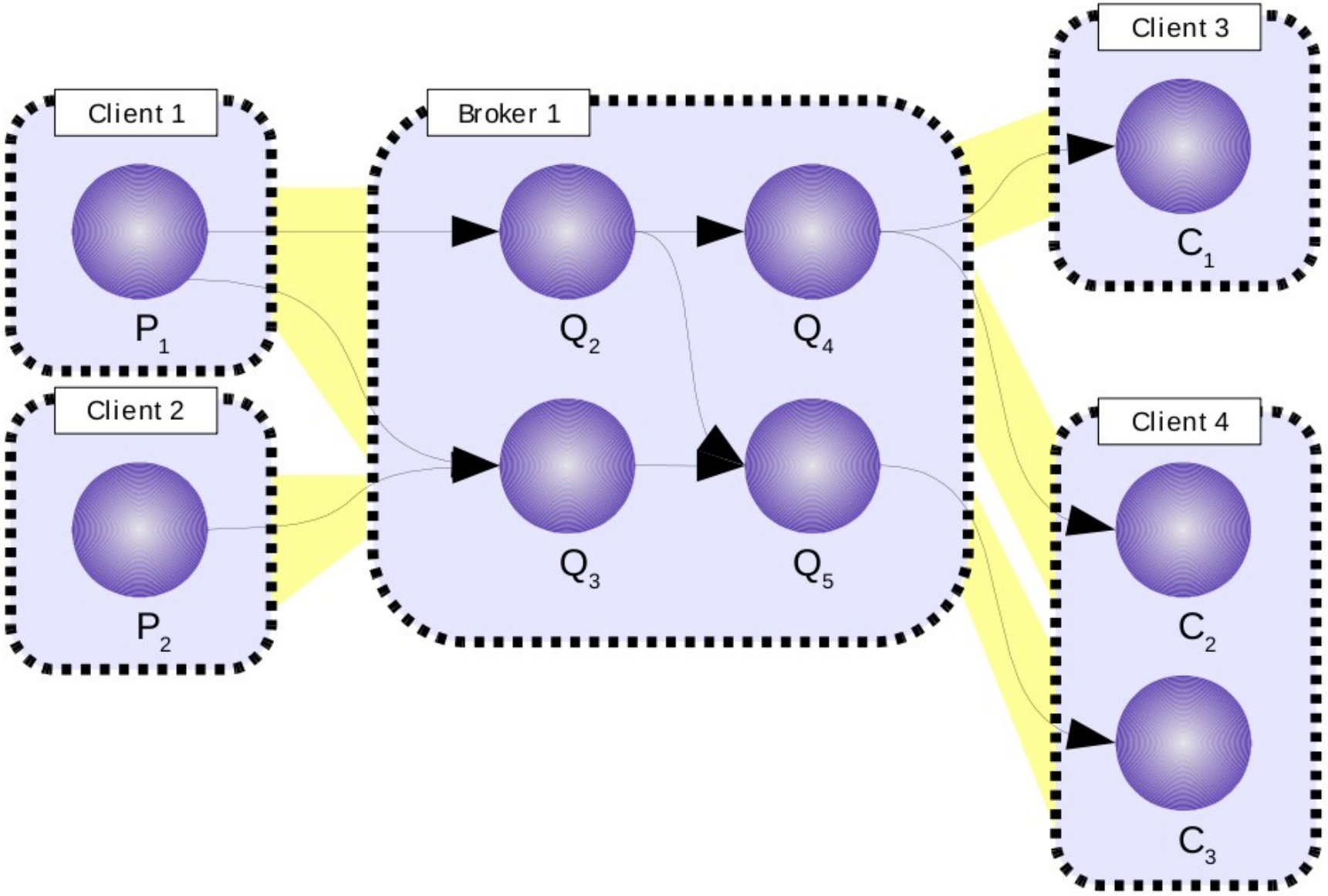
AMQP 0-* Model


Applications
Produce
Messages
Exchanges
Route and Filter
Messages
Queues
Store and Forward
Messages
Applications
Consume
Messages
P
P
X
X
C
C
C
Clients
Server (a.k.a. Broker)
Clients

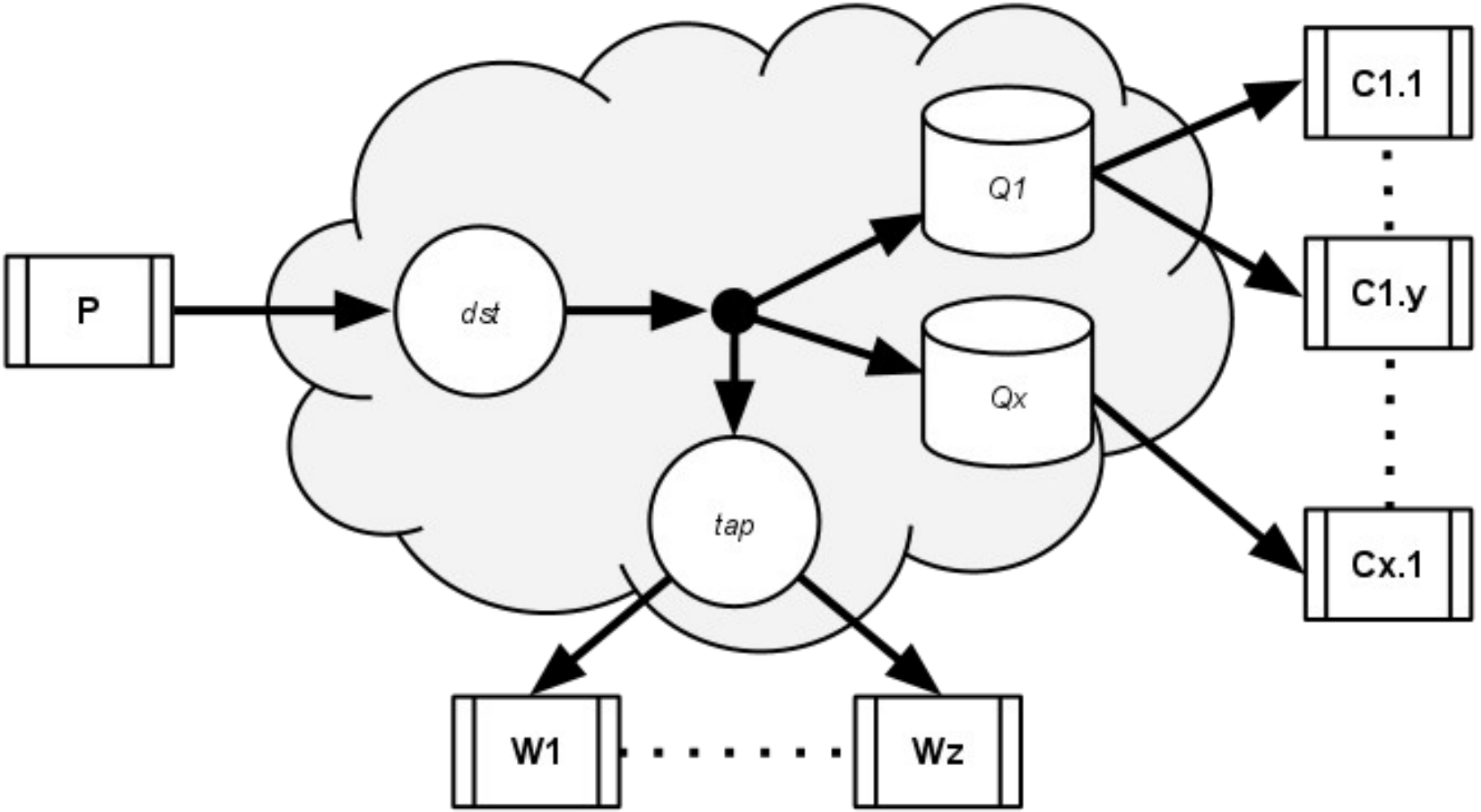
RabbitMQ's AMQP 0-*



AMQP 1-0 Model



Basic Block



Messaging Solutions

- ActiveMQ / Fuse Message Broker
- Qpid / Red Hat Enterprise MRG
- RabbitMQ
- HornetQ
- ZeroMQ

ActiveMQ

- Open source, ASF project
- Written in Java (220k SLOC)
- Supports: JMS, OpenWire, STOMP
  - future: AMQP 1-0 and others
- Currently used by EGEE/EGI
- Commercially supported by FuseSource (a Progress Software company)

Qpid

- Open source, ASF project
- Written in Java (200k SLOC) and C++ (100k)
- Supports: JMS, AMQP 0-10
  - future: AMQP 1-0
- Enhanced and supported by Red Hat as Red Hat Enterprise MRG

RabbitMQ

- Open source
- Written in Erlang (15k SLOC for the broker!)
- Supports: JMS, STOMP, AMQP 0-9 and others
  - future: AMQP 1-0 and more others
- Development effort comes from SpringSource (a division of VMware)

HornetQ

- Open source
- Written in Java (230k SLOC)
  - more than 50% of it for tests...
- Supports: JMS, STOMP
  - future: AMQP
- Will be integrated into JBoss and Red Hat products

ZeroMQ

- Open source
- Written in C++ (14k SLOC)
- Very different model: broker-less messaging
  - not suitable for all use cases
  - can be bridged with RabbitMQ

Features

- Many *unique* features exist:
  - ActiveMQ: Exclusive Consumer, Pending Message Limit Strategy, Virtual Destinations...
  - Qpid: Last Value Queue, Queue Sizing Constraints, Asynchronous Replication of Queue State...
  - RabbitMQ: Alternate Exchanges, Memory-based flow control, Exchange to Exchange Bindings...
- You should stick to your requirements and see which features are *really* needed or not

Summary So Far

- Things are more complex than they seem
- There is no best-of-breed messaging product yet
- Beware of marketing claims such as *“product ABC is fully XYZ compliant”*
- AMQP 1-0 is appealing but not yet mature
  - we are very far from being able to buy a standard message broker from Cisco

➔ Safe bet: common features, basic block, STOMP

Other Considerations

- Message formats
- Scalability
- Reliability
- Security
- Coding

Message Formats

- Header
  - can contain *some* information
  - if possible: *key*: C identifier, *value*: printable ASCII
- Body
  - can contain anything of reasonable size (< 1MB)
    - compression can be useful in some cases
  - frequently used formats: JSON, YAML, XML
- You can always group or split content

Scalability

Main parameters are:

- message size (distribution)
- average and peak message rates
- message replication/amplification (e.g. topic with many subscribers)
- number of messages per session
- protocol used, authentication, encryption
- network throughput and latency

Rules of Thumb

- For messages > 1kB, careful with throughput!
- WAN & STOMP: stay below 1k msg/s
- LAN & binary protocol: stay below 10k msg/s
- Persistent messages are at least x10 slower
- Stay well below new 100 sessions per second (especially with x.509 authentication)

➔ A message is cheap, a session is expensive
➔ Don't trust any claim, measure in realistic setup

Using More Brokers

- Fully connected brokers do not help much
- Auto discovery of producers and consumers only works in some (simple) cases
- It is good to use dedicated brokers
- It is easy to use identical independent brokers:
  - producers connect to any broker (round robin)
  - consumers connect to all brokers

➔ We will likely use several dedicated brokers

Reliability

- The messaging infrastructure
  - can be distributed and redundant
  - cannot be 100% available and reliable
- The application logic has a role to play
- Debunking another myth: persistent messages on queues are not necessarily better than non-persistent messages on topics

Broker Level Security

Authentication:

- How can a broker authenticate a client?
  - ➔ usually: name+password or X.509 certificates

Authorization:

- How can a broker authorize an action?
  - ➔ usually: static configuration file and per-destination granularity

Application Level Security

- Message encryption
  - can be used to prevent snooping
  - asymmetric encryption is easier to manage
- Message signing
  - can be used to identify the sender
  - can be used to guarantee message integrity
  - managing lists of allowed senders is non-trivial
    - but only you can know what you want to allow or not

Security Summary

- By default, do not trust the data you receive
- Security requirements are application specific
- Brokers can provide *some* security
- Extra security must be implemented at application level

Coding

- Factorize common code
- If possible, re-use existing code
- Isolate technology dependent code from the rest
- Abstracting messaging is good
- Messaging technology (like any computing technology) *will* change

Summary

- Messaging can bring many benefits
- It however cannot solve all the problems
- Security is important
- The messaging technology (as a whole) is not yet mature and is moving fast

Change is inevitable but we can make it manageable.