Course Name:
IoT & Applications (18EI2T09)

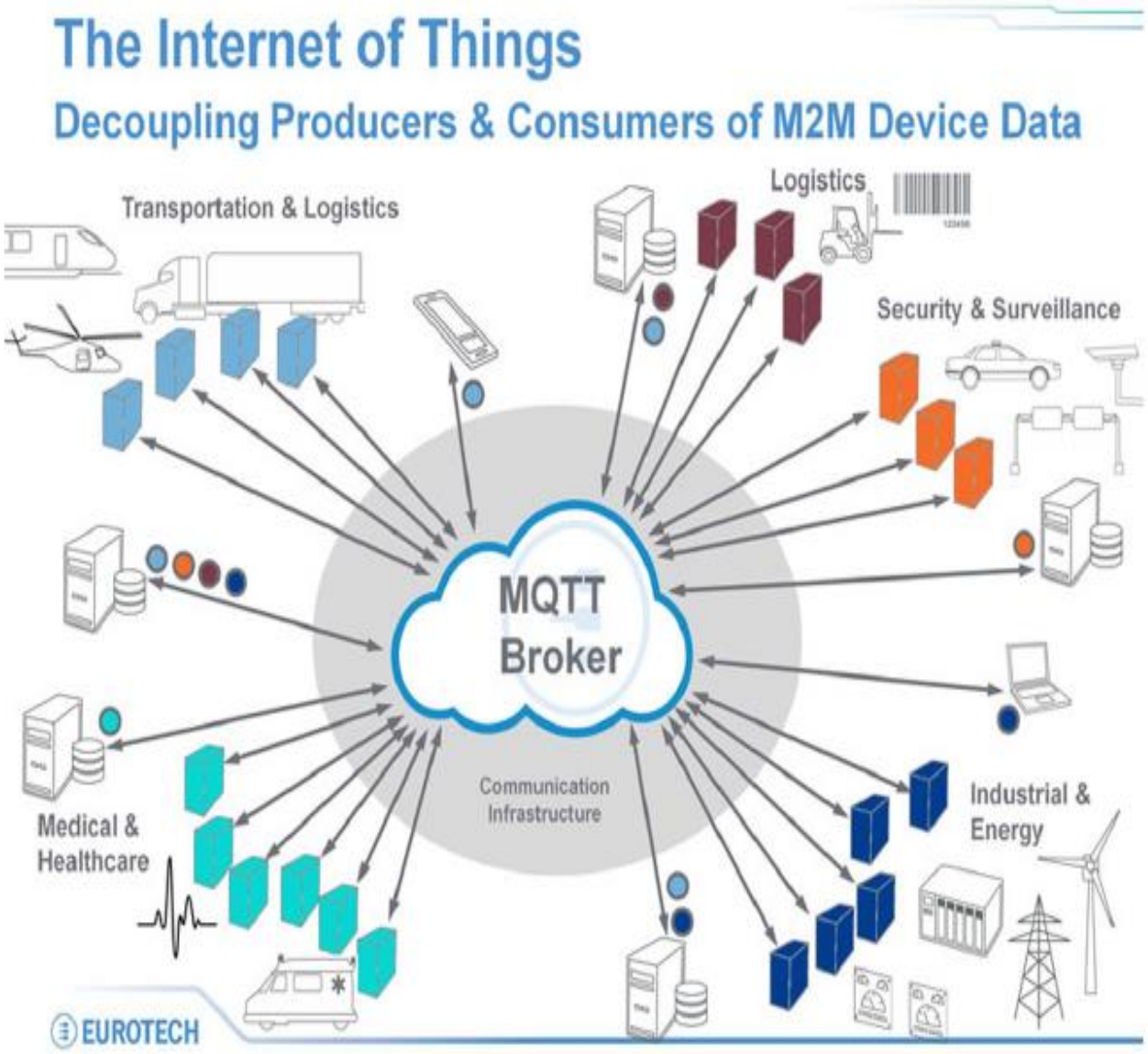
Class-2

Module-2: **IoT Strategies-** Networking, Communication, Adaptive & Event Driven Processes, Virtual Sensors, Security, Privacy & Trust, Low power communication, Energy harvesting, IoT related standardization;

IoT Protocols: MQTT, CoAP, AMQP, JMS, DDS, REST, XMPP.

IoT Protocols and Convergence

- The interconnected devices need to communicate using lightweight protocols that don't require extensive use of CPU resources.
- C, Java/Java Script, PHP, Python and some scripting languages are the preferable choices used by IoT applications.
- The IoT nodes use separate IoT gateways if there is needed protocol conversion/translation, database storage, or decision making in order to supplement the low-intelligence node.



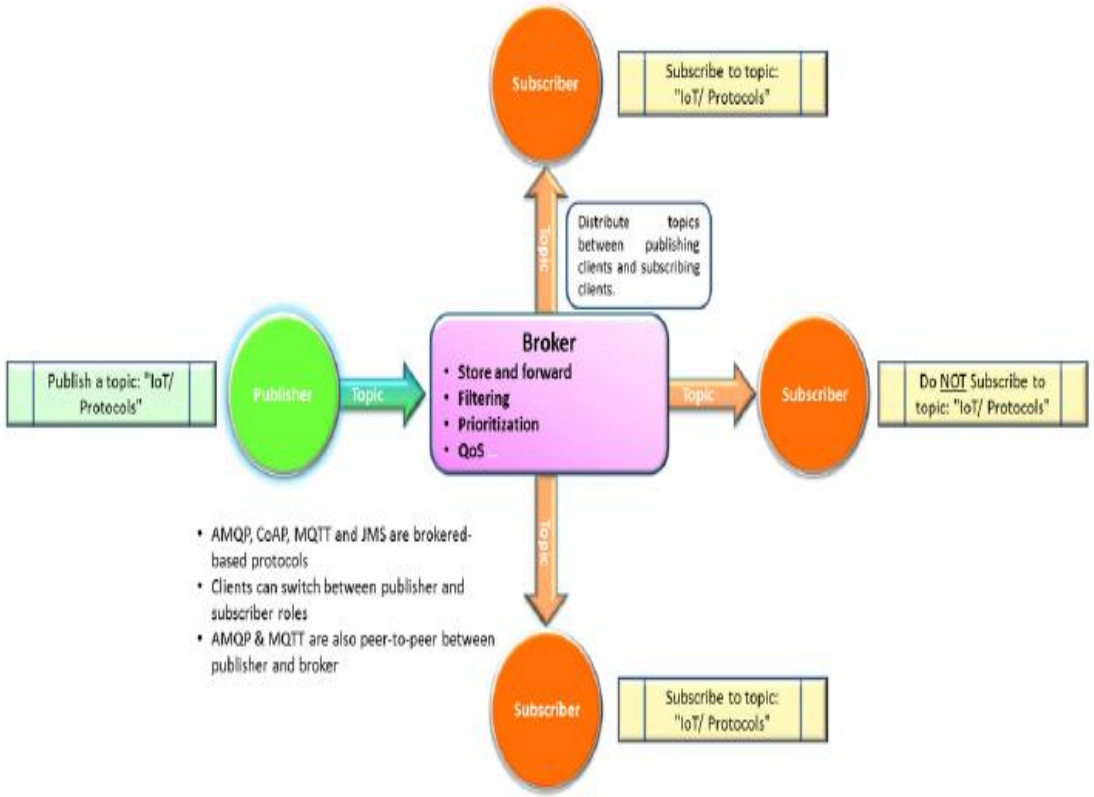
Message Queuing Telemetry Transport (MQTT) publish/subscribe protocol used to implement IoT and M2M applications (Source: Eurotech)

- One of the most important aspects for a convergence protocol that **support information exchange between domains**, is the **ability to convey the information (data)** contained in a particular domain to other domains.

- Today there are **two dominant architectures for data exchange protocols**;
  - **Bus-based, and**
  - **Broker-based**.
- In the broker-based architecture, the broker controls the distribution of the information.
- For example, it stores, forwards, filters and prioritizes publish requests from the publisher (the source of the information) to the subscriber (the consumer of the information) clients.

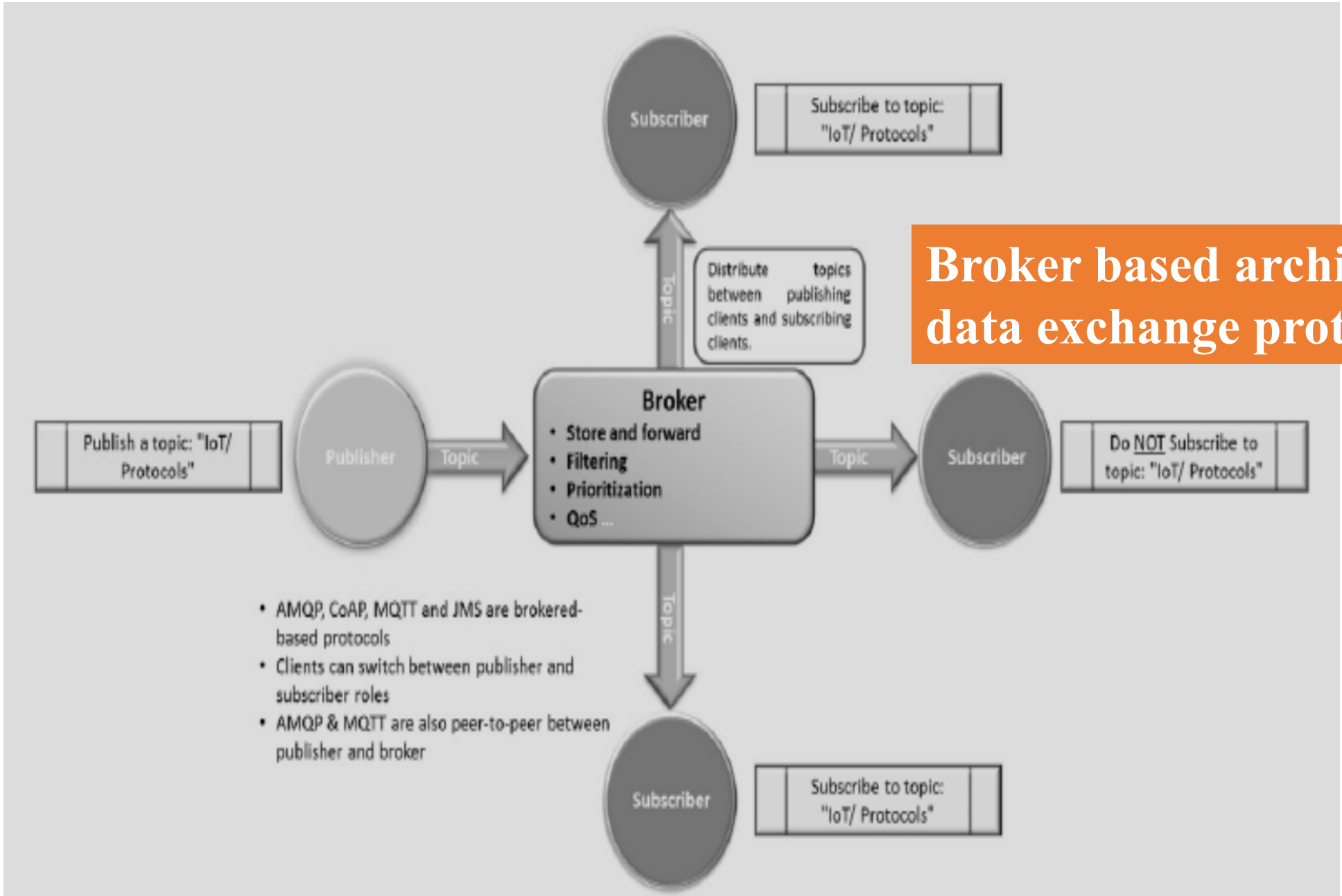
Broker-based protocols

- Clients switch between publisher and subscriber roles depending on their objectives.
- Examples of broker –based protocols include
  - Advanced Message Queuing Protocol (AMPQ),
  - Constrained Applications Protocol (CoAP),
  - Message Queue Telemetry Transport (MQTT) and
  - Java Message Service API (JMS).



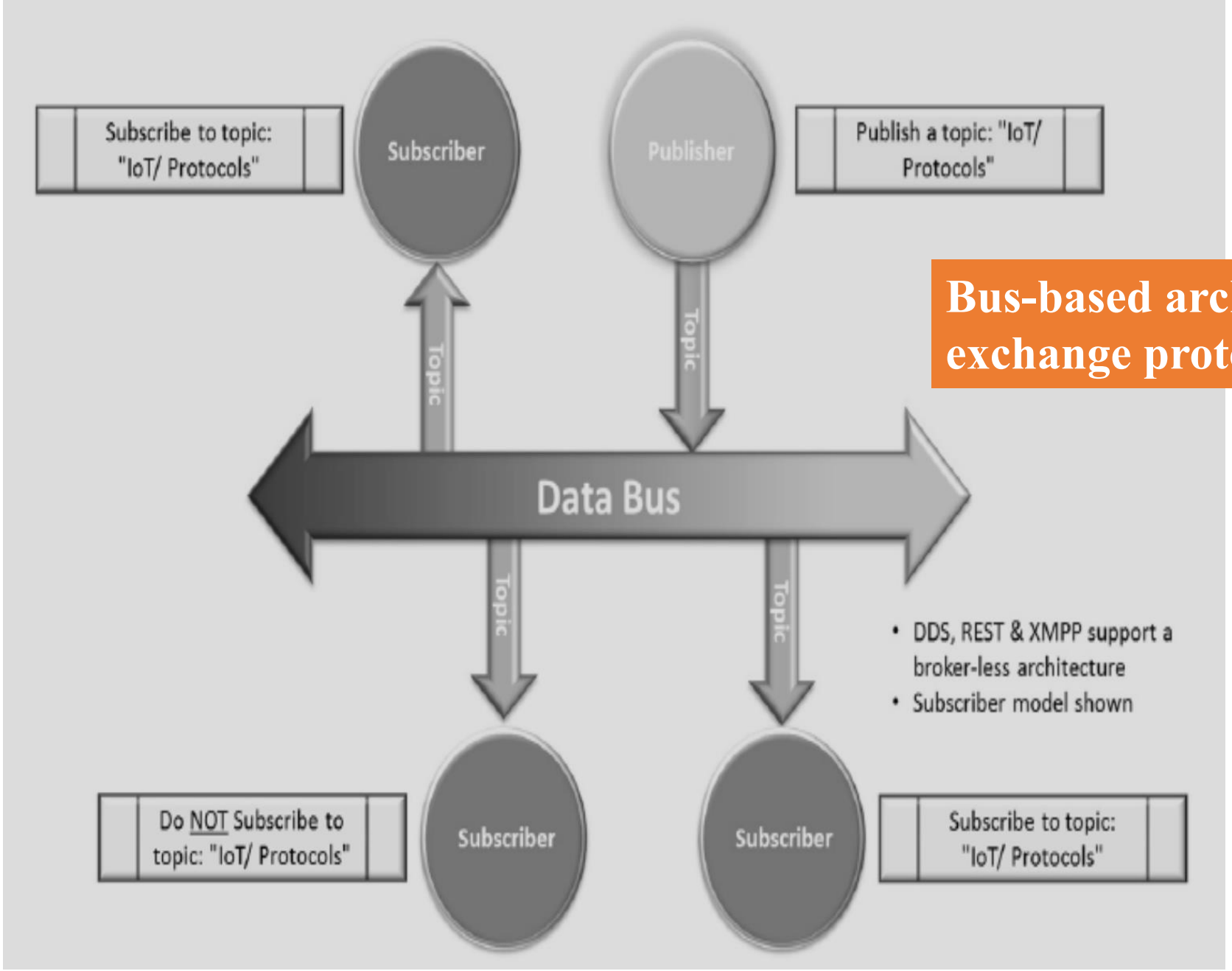
Broker based architecture for data exchange protocols

Broker based architecture for data exchange protocols



Bus-based protocols

- In the bus-based architecture, clients publish messages for a specific topic which are directly delivered to the subscribers of that topic.
- **There is no centralized broker or broker-based services.**
- Examples of bus-based protocols include
  - Data Distribution Service (DDS),
  - Representational State Transfer (REST) and
  - Extensible Messaging and Presence Protocol (XMPP).



Bus-based architecture for data exchange protocols

- **Another important way to classify these protocols** is whether they are message-centric or data-centric.

- **Message centric protocols** such as AMQP, MQTT, JMS and REST focus on the delivery of the message to the intended recipient(s), regardless of the data payload it contains.
- A **data-centric protocol** such as DDS, CoAP and XMPP focus on delivering the data and assumes the data is understood by the receiver. Middleware understands the data and ensures that the subscribers have a synchronized and consistent view of the data.

Question: What is a Middleware?

- Middleware is software that lies between an operating system and the applications running on it. Essentially functioning as **hidden translation layer, middleware enables communication and data management for distributed applications.**

- Yet another fundamental aspect of these protocols is whether it is web based like CoAP or application-based such as with XMPP, and AMQP.
- These aspects have fundamental effect on the environment, performance and tools available for implementers.

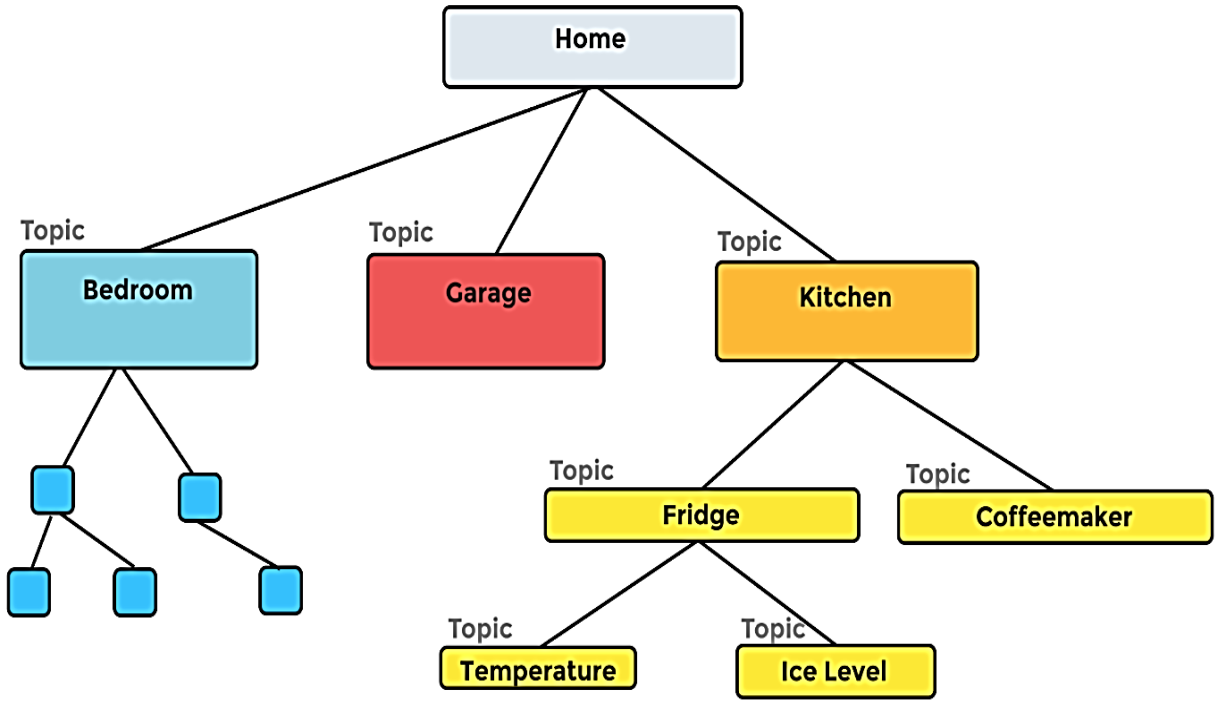
Another Classification: Web-based or Application-based

Web based like CoAP

Application-based such as with XMPP, and AMQP

1. Message Queue Telemetry Transport (MQTT)

- MQTT is an open-sourced protocol for passing messages between multiple clients through a central broker. It was designed to be simple and easy to implement.
- The MQTT architecture is broker-based, and uses long-lived outgoing TCP connection to the broker.
- MQTT also supports hierarchical topics (e.g., "subject/sub-subject/sub-sub-subject") file system structure.



Some of the main features of MQTT:

- MQTT can be used for two-way communications over unreliable networks where cost per transmitted bit is comparatively high. **(Bi-directional communication)**
- It is also compatible with low power consumption devices.
- The protocol is light-weight (simple) and therefore well suited for constrained environments.
- MQTT has a mechanism for asynchronous communication and for communicating disconnect messages when a device has disconnected. The most recent message can also be stored and forwarded.
- Multiple versions of MQTT are available to address specific limitations.

- With MQTT, only **partial interoperability** between publishers and subscribers can be guaranteed because the meaning of data is not negotiated.

- Clients must know message format up-front.
- In addition, **it does not support labeling messages with types or metadata.**

- MQTT may include large topic strings that may not be suitable for small packet size of some transport protocols such as IEEE 802.15.4.

- MQTT may require EXI (Efficient XML Interchange) to compress the message length that could reduce communication efficiency.

- TCP may negatively affect the network efficiency as the number of nodes (connection to the broker) increases.

- Limitation:

If the number of nodes is greater than a thousand, poor performance and complexity may also result because automatic/dynamic discovery is not supported in MQTT.

Note:

Because the protocol was designed to be simple, users must decide whether **it is too simple and susceptible to potential hacking.**

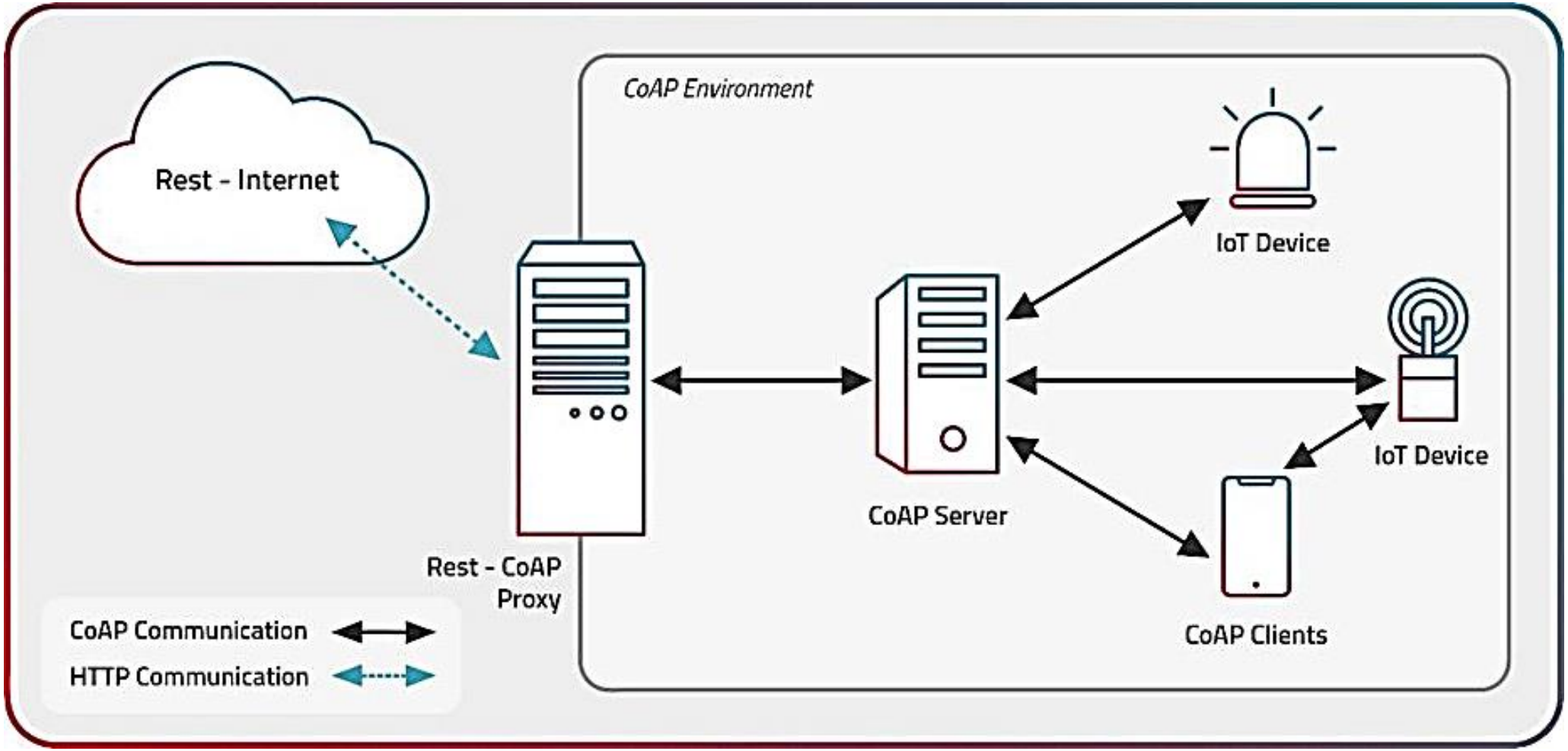
2. Constrained Applications Protocol (CoAP)

- CoAP is an internet-based client/server model document transfer protocol similar to HTTP but designed for constrained devices.
- A sensor is typically a “server” of information and the “client” the consumer who can also change the states.
- It supports a one-to-one protocol for transferring state information between client and server.
- CoAP utilizes User Datagram Protocol (UDP), and supports broadcast and multicast addressing. It does not support TCP.

2. Constrained Applications Protocol (CoAP)

- Constrained Application Protocol (CoAP) is a specialized **web transfer protocol** for use with constrained nodes and constrained networks in the Internet of Things.
- CoAP is designed to enable simple, constrained devices to join the IoT even through constrained networks **with low bandwidth and low availability**.
- It is **generally used for machine-to-machine (M2M) applications** such as smart energy and building automation.
- The protocol was designed by the Internet Engineering Task Force (IETF), CoAP is specified in IETF RFC 7252.

2. Constrained Applications Protocol (CoAP)



2. Constrained Applications Protocol (CoAP)

CoAP Features

Web Protocol Used in M2M With Constrained Requirements

Asynchronous Message Exchange

Low Overhead

Very Simple To Perform Syntactic Analysis

(URI) Uniform Resource Identifier

Proxy and Caching Capabilities

- CoAP communication is through connectionless datagrams, and can be used on top of SMS and other packet-based communications protocols.

- CoAP supports content negotiation and discovery, allowing devices to probe each other to find ways to exchange data.

- CoAP was designed for interoperability with the web (including HTTP and RESTful protocols), and supports asynchronous communications.

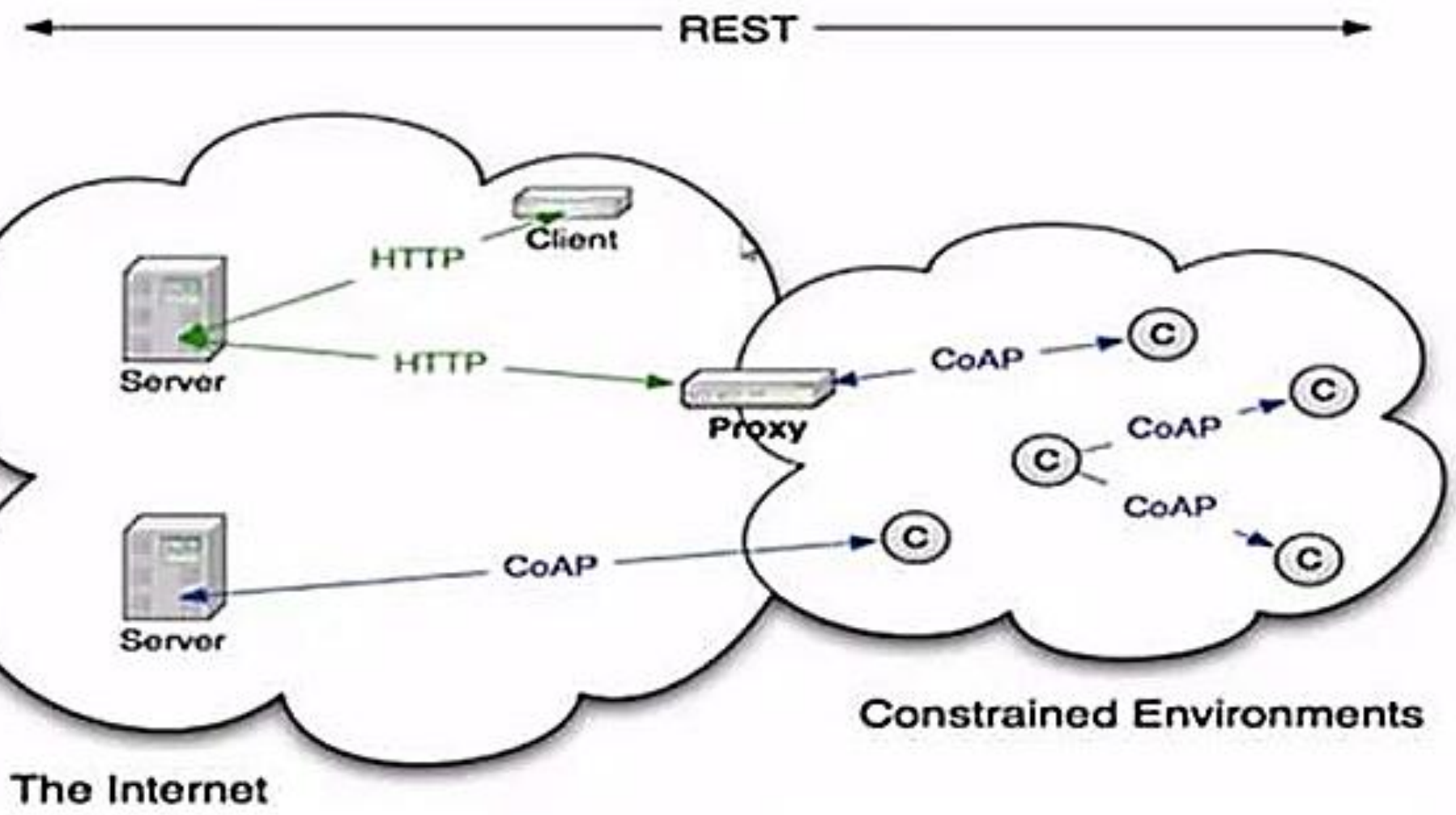
- CoAP supports "observing" resource state changes as they occur so it is best suited to a state-transfer model, not purely an event-based model.

- CoAP supports resource discovery.

- UDP may be easier to implement in microcontrollers than TCP, but the security tools used for TCP (SSL/TLS) are not available in UDP. Datagram Transport Layer Security (DTLS) can be used instead.

CoAP Security:

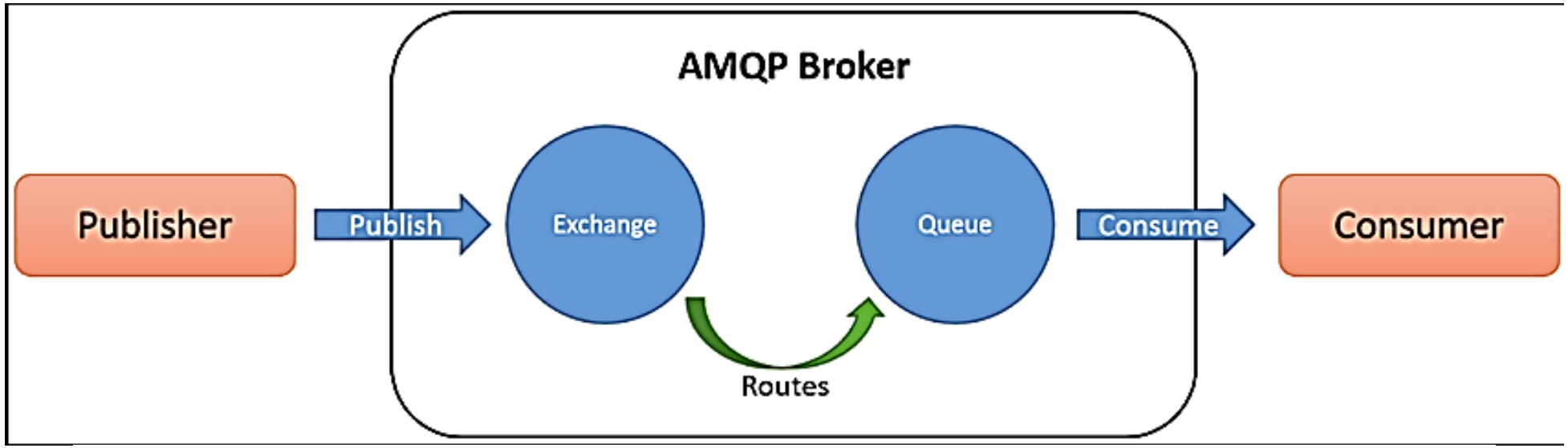
Datagram Transport Layer Security (DTLS) is a communications protocol providing security to datagram-based applications by allowing them to communicate in a way designed to prevent eavesdropping, tampering, or message forgery.

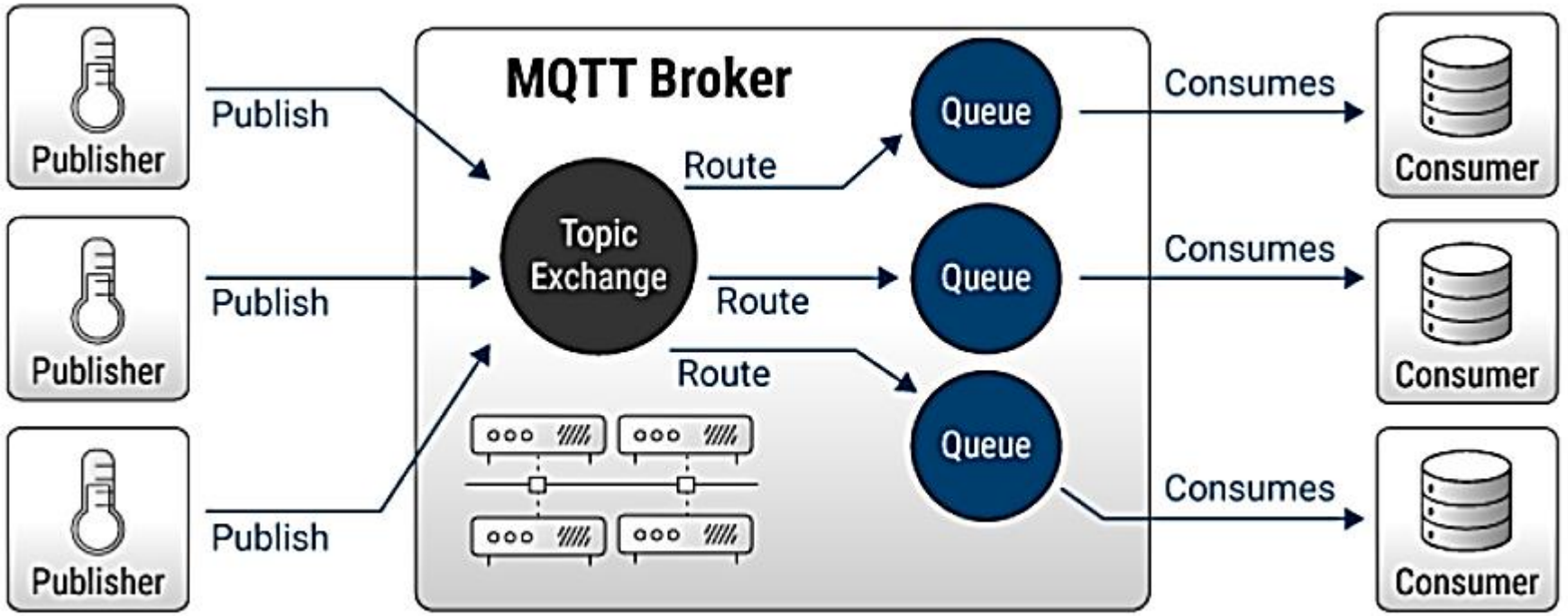


CoAP Architecture

3. Advanced Message Queuing Protocol (AMQP)

- AMQP is an application layer message-centric brokered protocol that emerged from the financial sector with the objective of replacing proprietary and non-interoperable messaging systems.
- The key features of AMQP are message orientation, queuing, routing (including point-to-point and publish-and subscribe), reliability and security.
- Discovery is done via the broker.


AMQP Broker
Publisher
Publish
Exchange
Queue
Consume
Consumer
Routes


MQTT Broker
Publisher
Publisher
Publisher
Publish
Publish
Publish
Topic Exchange
Route
Route
Route
Queue
Queue
Queue
Consumes
Consumes
Consumes
Consumer
Consumer
Consumer

3. Advanced Message Queuing Protocol (AMQP)

Key Features

AMQP was designed with the following main characteristics as goals:

- Security
- Reliability
- Interoperability
- Standardized
- Open Source

Key Capabilities

AMQP connects across:

•Organizations – applications in different organizations

•Technologies – applications on different platforms

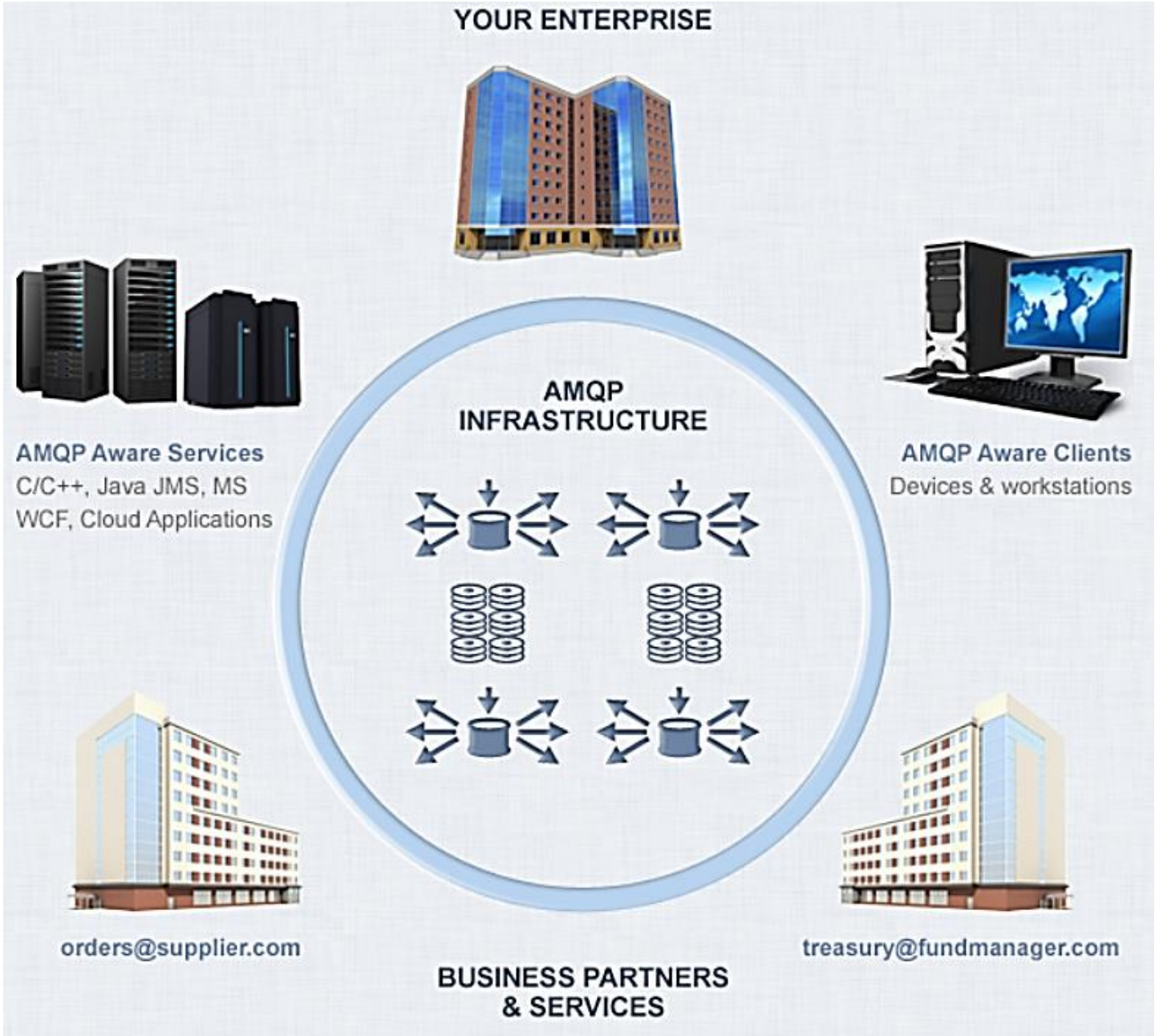
•Time – systems don't need to be available simultaneously

•Space – reliably operate at a distance, or over poor networks

3. Advanced Message Queuing Protocol (AMQP)

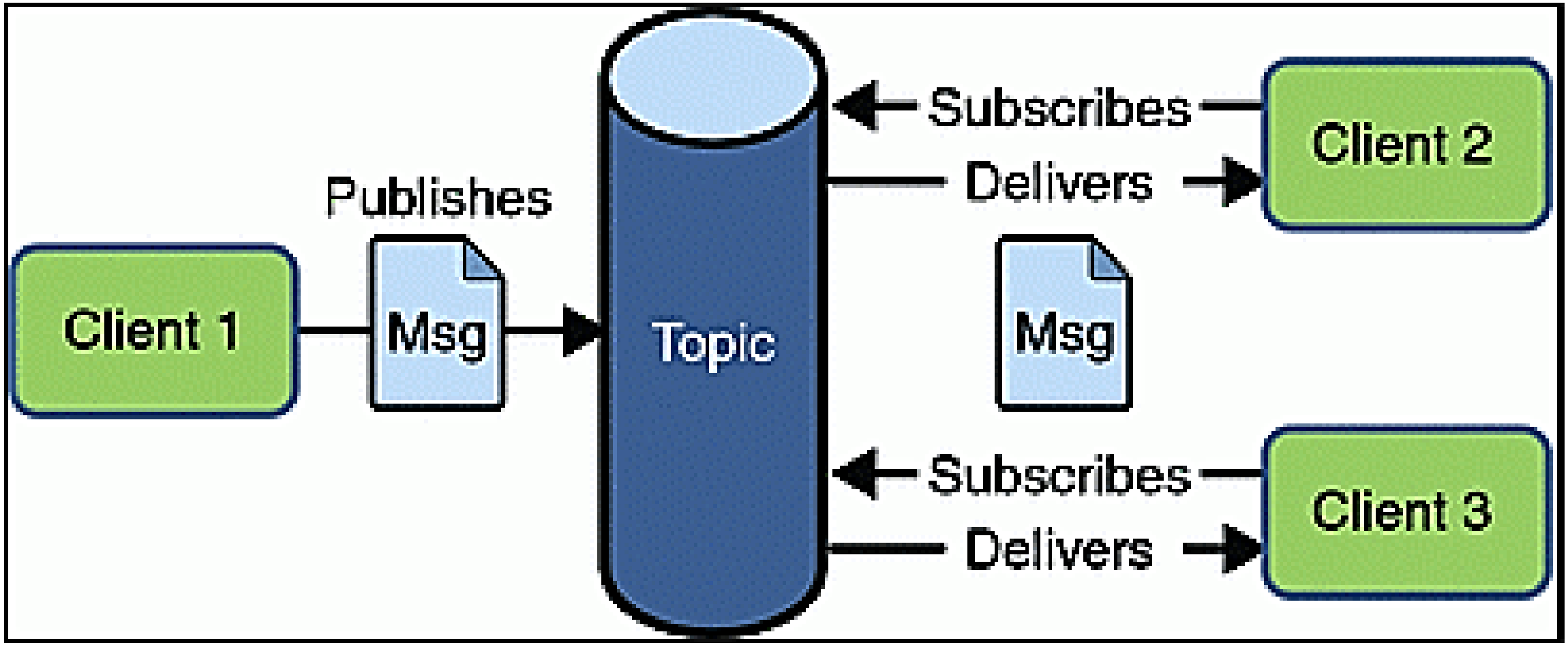
Main Features of AMQP:

- It provides **flow controlled**, **message-oriented communication** with message-delivery guarantees such as at-most-once (where each message is delivered once or never), at-least-once (where each message is certain to be delivered, but may do so multiple times) and exactly-once (where the message will always certainly arrive and do so only once), and **authentication and/or encryption**.
- It assumes an underlying reliable transport layer protocol such as **Transmission Control Protocol (TCP) using SSL/TLS**.


YOUR ENTERPRISE
AMQP INFRASTRUCTURE
AMQP Aware Services
C/C++, Java JMS, MS
WCF, Cloud Applications
AMQP Aware Clients
Devices & workstations
orders@supplier.com
treasury@fundmanager.com
BUSINESS PARTNERS & SERVICES

4. Java Message Service API (JMS)

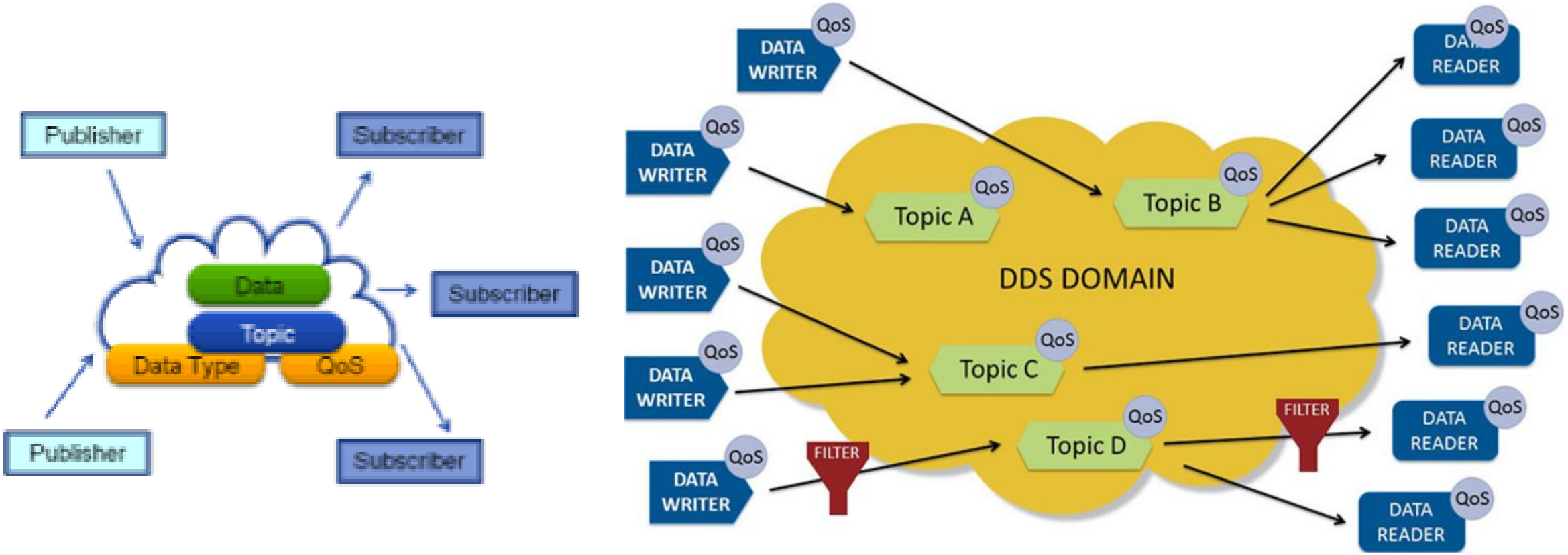
- JMS is a message oriented middleware API for creating, reading, sending, receiving messages between two or more clients, based on the Java Enterprise Edition.
- It was meant to separate application and transport layer functions and allows the communications between different components of a distributed application to be loosely coupled, reliable and asynchronous over TCP/IP.
- JMS supports both the point to point and publish/subscribe models **using message queuing**, and durable subscriptions (i.e., store and forward topics to subscribers when they "**log in**").
- Subscription control is through topics and queues with message filtering.
- Discovery is via the broker (server).



Java Message Service API (JMS)

5. Data Distribution Service (DDS)

- DDS is a data-centric protocol used to enable scalable, real-time, dependable high performance and interoperable data exchanges.
- The original target applications were financial trading, air traffic control, smart grid management and other big data, mission critical applications.
- It is a decentralized broker-less protocol with direct peer-to-peer communications between publishers and subscribers and was designed to be language and operating system independent.
- DDS sends and receives data, events, and command information on top of UDP but can also run over other transports such as IP Multicast, TCP/IP, shared memory etc.
- DDS supports real-time many-to-many managed connectivity and also supports automatic discovery.

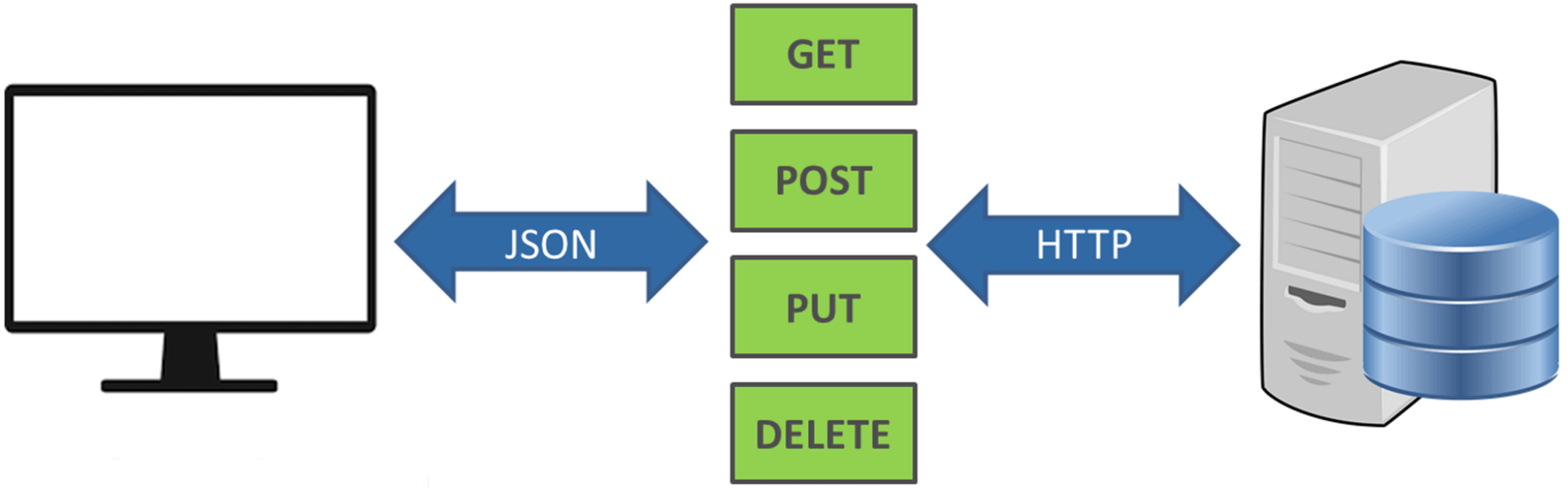


Data Distribution Service (DDS)

6. Representational State Transfer (REST)

- REST is a language and operating system independent architecture for designing network applications using simple HTTP to connect between machines.
- It was designed as a lightweight point-to-point, stateless client/server, cacheable protocol for simple client/server (request/reply) communications from devices to the cloud over TCP/IP.
- Use of stateless model supported by HTTP and can simplify server design and can easily be used in the presence of firewalls, but may result in the need for additional information exchange. It does not support Cookies or asynchronous, loosely coupled publish-and-subscribe message exchanges.
- Support for systems with more than a thousand nodes may result in poor performance and complexity.

6. Representational State Transfer (REST)



Client sends a **request**

HTTP methods

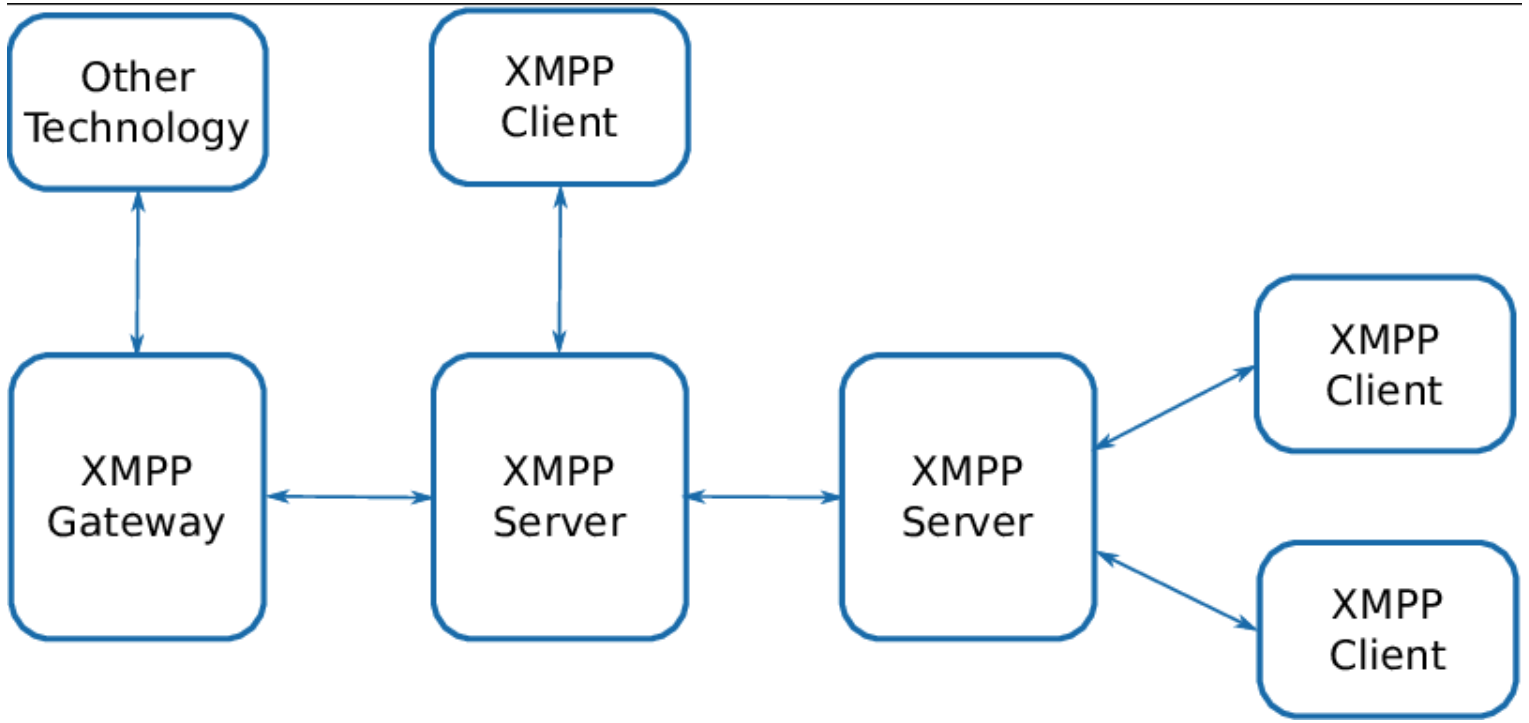
Server sends a **response**

7. Extensible Messaging and Presence Protocol (XMPP)

- XMPP is a communications protocol for message oriented middleware based on XML (formally "Jabber").
- It is a brokerless decentralized client-server model and is used by text messaging applications.
- It is near real-time and **massively scalable to hundreds of thousands of nodes**.
- Binary data must be base64 encoded before it can be transmitted in-band.
- It is useful for devices with large and potentially complicated traffic, and where extra security is required.
- For example, it can be used to isolate security to between applications rather than to rely on TCP or the web.
- The users or devices (servers) can keep control through preference settings.
- New extensions being added to enhance its application to the IoT, including Service Discovery, Concentrators for connecting legacy sensors and devices, Sensor Data, and Control.

```
<?xml version="1.0"?>
<quiz>
 <qanda seq="1">
  <question>
   Who was the forty-second
   president of the U.S.A.?
  </question>
  <answer>
   William Jefferson Clinton
  </answer>
 </qanda>
 <!-- Note: We need to add
  more questions later.-->
</quiz>
```

XML Format



Extensible Messaging and Presence Protocol (XMPP)