ECE4574 – Large-Scale SW Development for Engineering Systems
Lecture 8 – Middleware, Brokers and MQTT

Creed Jones, PhD

Course Updates

- Sprint 1 Begins Today!
  - Periodic “stand-up” meetings
  - Working on the sprint backlog items
  - Communicate with each other often!
- Homework 1 is due this Friday, September 22
  - 11:59 PM
- Quiz 3 is THIS Wednesday, September 20
  - covers lectures 6-8

Today’s Objectives

Middleware

- Concept
- Distributed Objects
- Message-oriented Middleware
- Application Servers
- Message Brokers

An example: MQTT

- The Internet of Things
- MQTT – clients and brokers
- Implementations

MIDDLEWARE

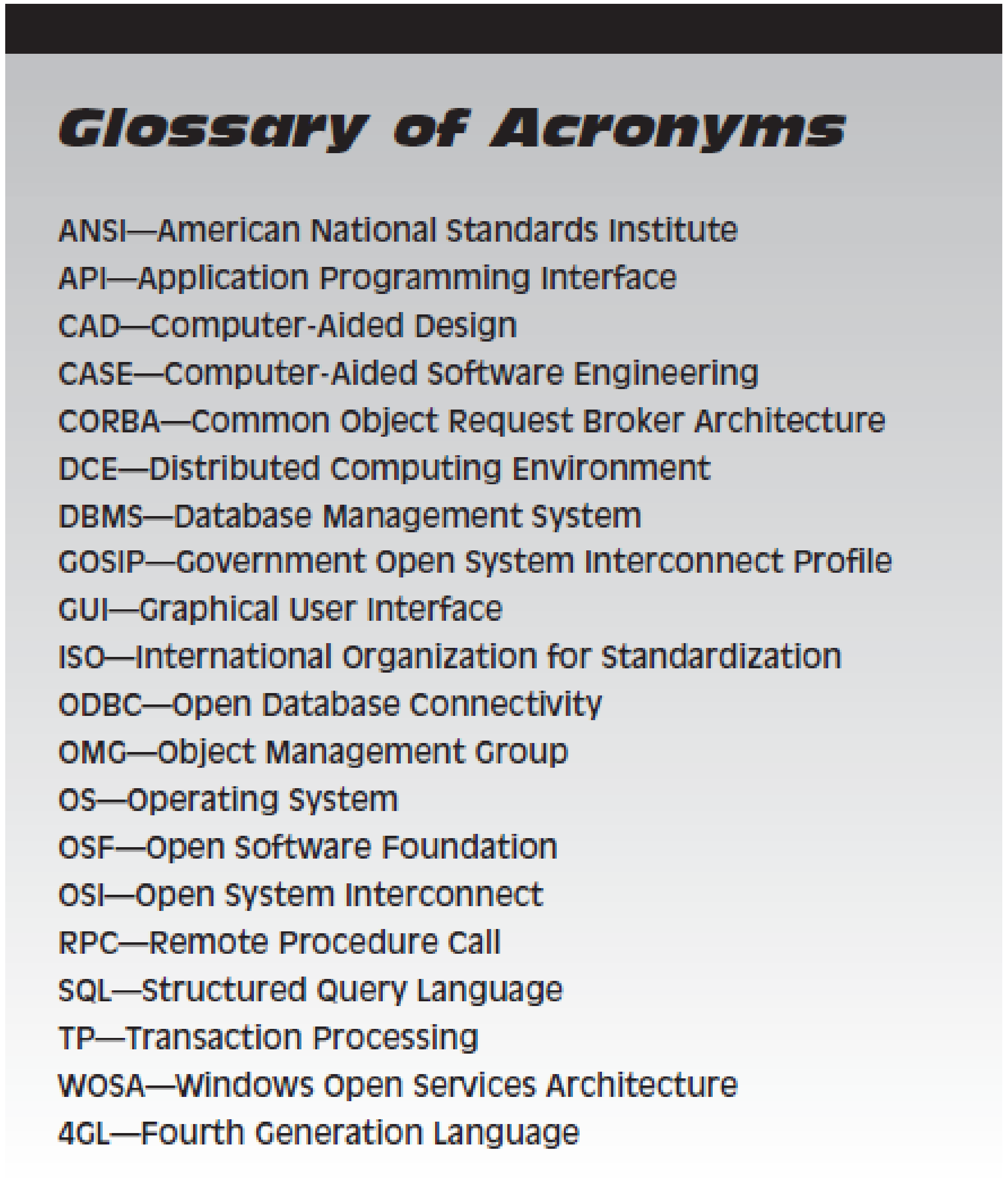
Middleware is software that connects components

- The connections between a database and an application front-end
- Software that connects and coordinates pre-existing subsystems
- Software that functions as a conversion or translation layer
- The (hidden) infrastructure that makes an application work

The concept of middleware is still present in the design and architecture of software systems

It has evolved in several different directions:

- Transaction Processing
- Distributed Services
- Database Services
- Secure Services



Glossary of Acronyms

ANSI	American National Standards Institute
API	Application Programming Interface
CAD	Computer-Aided Design
CASE	Computer-Aided Software Engineering
CORBA	Common Object Request Broker Architecture
DCE	Distributed Computing Environment
DBMS	Database Management System
GOSIP	Government Open System Interconnect Profile
GUI	Graphical User Interface
ISO	International Organization for Standardization
ODBC	Open Database Connectivity
OMG	Object Management Group
OS	Operating System
OSF	Open Software Foundation
OSI	Open System Interconnect
RPC	Remote Procedure Call
SQL	Structured Query Language
TP	Transaction Processing
WOSA	Windows Open Services Architecture
4GL	Fourth Generation Language

Common middleware technologies fall into several levels of the software hierarchy

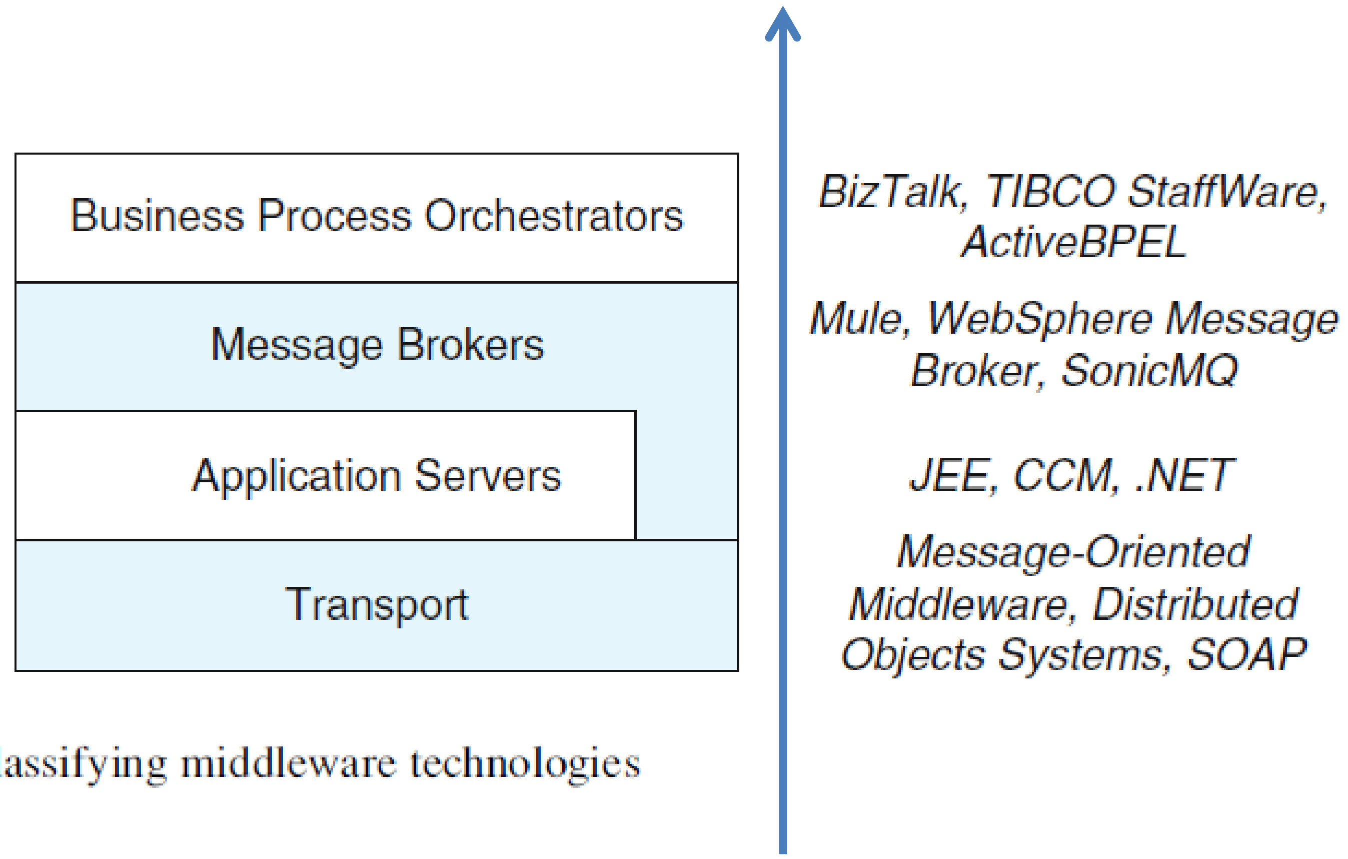

Business Process Orchestrators
Message Brokers
Application Servers
Transport
BizTalk, TIBCO StaffWare, ActiveBPEL
Mule, WebSphere Message Broker, SonicMQ
JEE, CCM, .NET
Message-Oriented Middleware, Distributed Objects Systems, SOAP

Fig. 4.1 Classifying middleware technologies

- BPOs allow long and complex workflows to be implemented
- Message brokers process messages and support multithreading
- Application servers also handle transactions and security
- Transport layer sends requests and moves data

Let's look at three common commercial middleware components in use today (not the only three)

1. Red Hat JBoss Enterprise Application Platform (EAP)

2. IBM WebSphere

3. Oracle WebLogic

JBoss is open-source middleware based on Java Enterprise Edition

1. Red Hat JBoss Enterprise Application Platform (EAP)

Red Hat JBoss Enterprise Application Platform (EAP) is a mouthful of a name, and it packs a powerful punch as well. Version 7 of the software was released last year, providing a cloud-compatible, flexible solution for enterprises looking to increase their agility.

<u>Benefits of JBoss EAP</u>

Solid architectural foundation, with quick start-up times and low memory usage

Out-of-the-box integration with DevOps tools such as Maven, Jenkins and Arquillian

“JBoss Migration Center” at JBoss.org makes it simpler to move your existing applications

High-quality support, consistently outperforming the other two solutions in customer surveys

<u>JBoss EAP Pricing and Support</u>

A standard one-year subscription to JBoss EAP running on 16 CPU cores will cost your organization $8,000, which includes web and phone support during standard business hours. If you need premium support, which includes 24/7 assistance for problems of urgent and high-level importance, you’ll be paying $12,000 annually.

https://shadow-soft.com/middleware-software-tools/

IBM WebSphere is a popular middleware component that can be used in fairly large systems

2. IBM WebSphere
IBM promotes its solution WebSphere Application Server as a high-performance, feature-rich option – the Ferrari to Red Hat's Ford Focus, if you will. But are the advantages worth the higher price tag?

<u>Benefits of IBM WebSphere</u>
Rapid installation, deployment and development
Support for on-premises, cloud-based and hybrid solutions
Integration with other IBM products in the cloud, such as the Watson artificial intelligence and the dashDB SQL database service
Flexible, robust architecture that can scale to match demand

<u>IBM WebSphere Pricing and Support</u>
WebSphere Application Server is free of charge without support, and with a maximum heap size of 2 gigabytes for the Java Virtual Machine. After that, the cost to run IBM WebSphere is based on PVUs (Processor Value Units), which vary based on the type of processor you're using. As of writing, the annual cost of a subscription and support plan was $55.25 per PVU, or $14,100 per limited use socket.

https://shadow-soft.com/middleware-software-tools/

Oracle WebLogic is a popular choice for middleware

3. Oracle WebLogic

Oracle WebLogic is the third contender when it comes to enterprise options for middleware – with much to recommend it.

Benefits of Oracle WebLogic

Oracle's Enterprise Grid Messaging is a high-performance architecture for communication between applications

Hassle-free application deployment, resulting in lower operational costs

Oracle JRockit, at the hood of WebLogic, claims to be the "fastest JVM in the industry"

Easy integration with other Oracle applications and databases

Oracle WebLogic Pricing and Support

Oracle allows enterprise users to license WebLogic via either the number of users or the number of processors. An annual subscription will cost $5,000 per processor or $100 per user, as well as a license and support fee equal to $5,500 or $110 respectively. Multi-year and perpetual licenses are also available for purchase.

https://shadow-soft.com/middleware-software-tools/

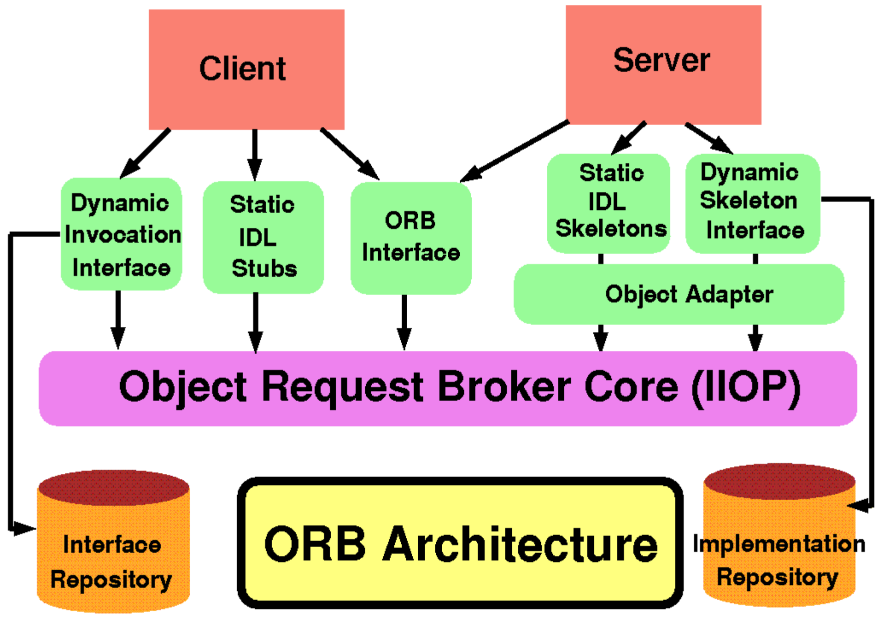
MIDDLEWARE AND MESSAGE BROKERS

Common Object Request Broker Architecture (CORBA) is a standard for communication between software objects through a common middleware component

- Distributed Objects allows the objects to be resident on different machines and to communicate through an Object Request Broker (ORB)

- Objects in CORBA obey all of the pleasant characteristics that OOP brings to us
  - Separation of interface from implementation
  - cohesion and encapsulation

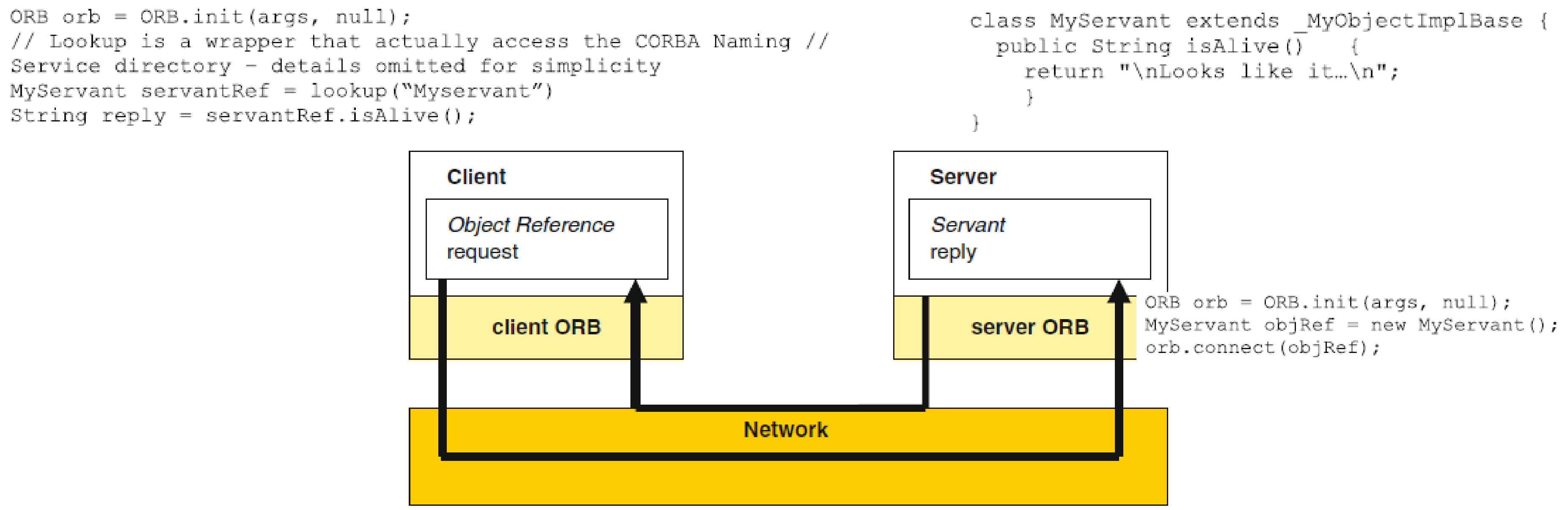
- In CORBA, objects communicate by sending requests to, and responding to requests from, the ORB



Client
Server
Dynamic Invocation Interface
Static IDL Stubs
ORB Interface
Static IDL Skeletons
Dynamic Skeleton Interface
Object Adapter
Object Request Broker Core (IIOP)
Interface Repository
ORB Architecture
Implementation Repository

http://www.omg.org/gettingstarted/corbafaq.htm

CORBA supports remote invocations – where the requesting and serving objects are on different machines



```
ORB orb = ORB.init(args, null);
// Lookup is a wrapper that actually access the CORBA Naming //
Service directory - details omitted for simplicity
MyServant servantRef = lookup("Myservant")
String reply = servantRef.isAlive();
```

```
class MyServant extends _MyObjectImplBase {
  public String isAlive()   {
    return "\nLooks like it…\n";
    }
}
```

```
ORB orb = ORB.init(args, null);
MyServant objRef = new MyServant();
orb.connect(objRef);
```

Fig. 4.2 Distributed objects using CORBA

- Internet InterORB Protocol (IIOP) maps CORBA messages through the Internet protocol

How do we use CORBA?

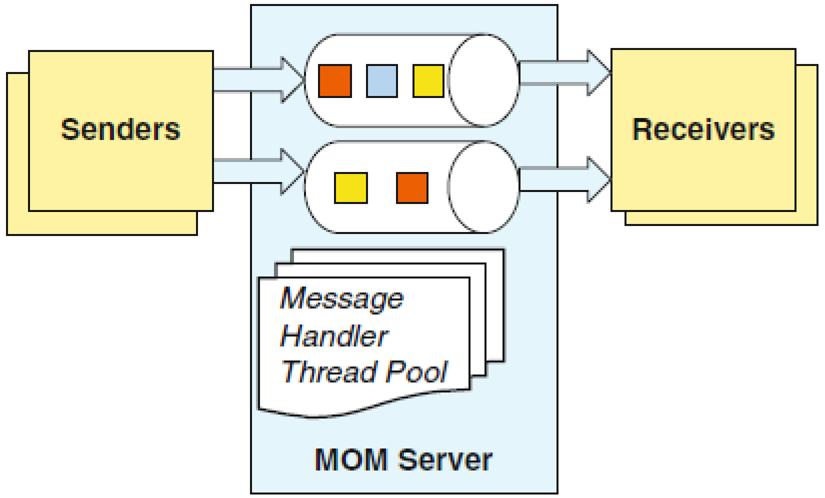
- Learn by example
- Much of the application software has to do with creating and registering components with the ORB

- In early days there were problems with incompatibility; these have been resolved
- CORBA is a standard – to use it, we must obtain an implementation; several open-source versions are available

Message-Oriented Middleware provides a standard queue between message creators and receivers (clients and servers)

- CORBA is an example of a synchronous architecture
  - Servers respond to client requests "promptly"

- MOM supports multiple queues
  - One or more processes can send messages to a given queue
  - Any process can receive from one or more queues
  - FIFO message receipt
  - Queues are referred to by name
  - Sending is a non-blocking operation (in general)
  - TCP/IP sockets are commonly used



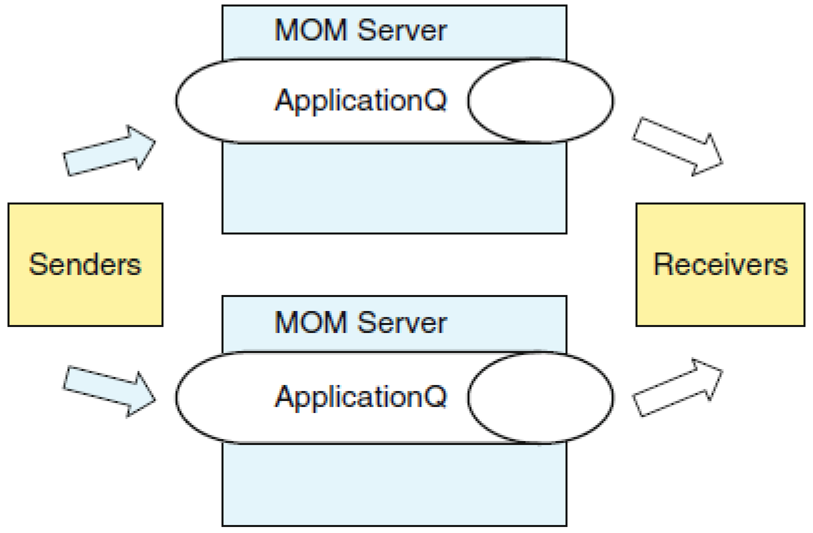
Message-Oriented Middleware supports Reliable Message Delivery

Three levels of Quality of Service (QoS) can be chosen

1. Best effort: "We will try to deliver the message. If we don't deliver it and there is a failure, it's gone"
2. Persistent: "If there is a failure, we have a copy of the message and will reload it and deliver when possible"
3. Transactional: "You can group messages into 'all or nothing' bundles"

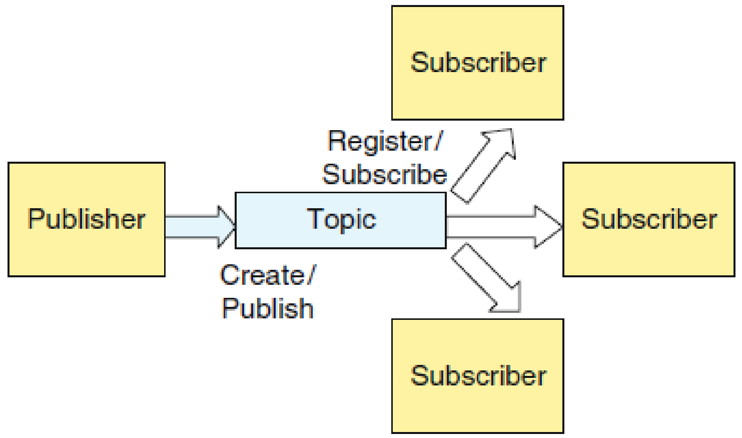
MOM servers can be clustered, to provide greater reliability

- This is ideal for applications with strict uptime requirements
- Also allows for spreading the workload across machines
- There must be a cluster controller, which can be a single point of failure



Message-Oriented Middleware is based on a 1-1 messaging style, but we can also build "publish-subscribe" systems

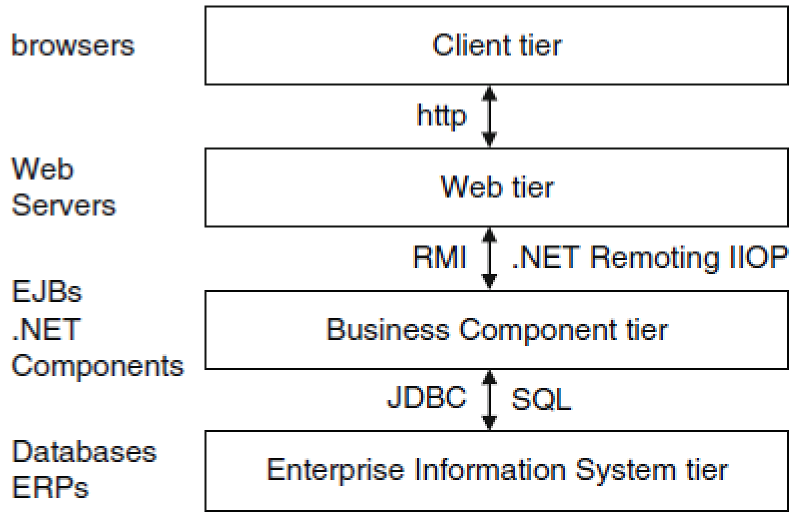
- Many applications require 1-many, many-1 or many-many communication between components
- Sources of messages are called *Publishers*
- Publishers send a message to a named *topic* or subject
- Subscribers register to *listen* to a particular topic
- All messages on that topic are sent to its subscribers



Application Servers are middleware that provide: distributed communications, security, transactions and persistence

- A familiar example is JEE, Java Enterprise Edition (still often called J2EE)

- An application server will implement the two middle tiers of the N-tier model shown here



Message Brokers are central software elements that receive, validate, transform and forward messages between clients

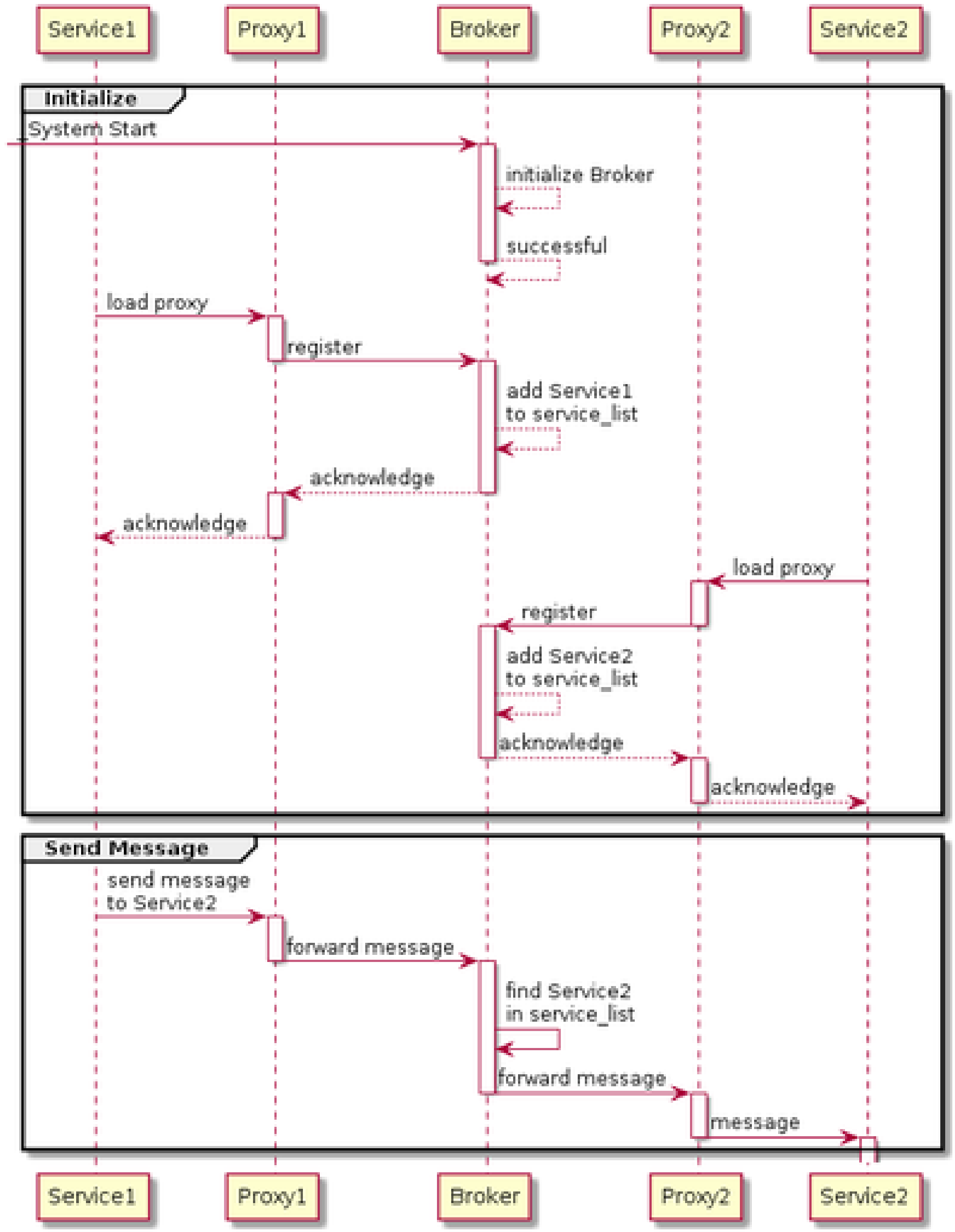
- Checking, translation and forwarding
- Generally centered on one or several message queues
- Provide asynchronous capability
  - Clients don't have to wait for brokers and vice versa

Two main styles of message broker

- Point-to-point
  - one message will go to one receiver
- Publish-subscribe
  - concept of broadcast

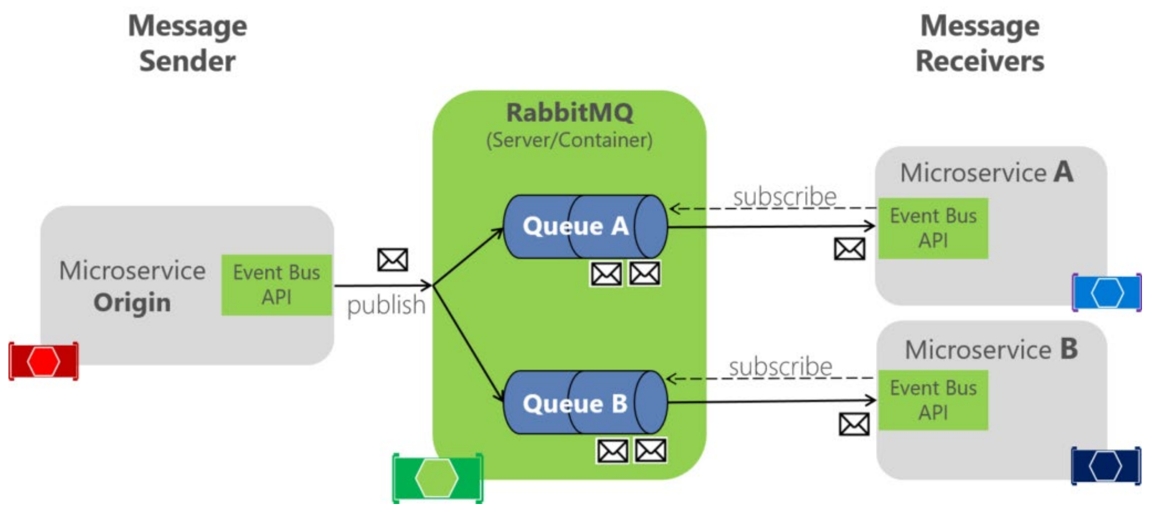
Most Message Brokers implement the *publish-subscribe* design pattern

note this example uses proxies for the services

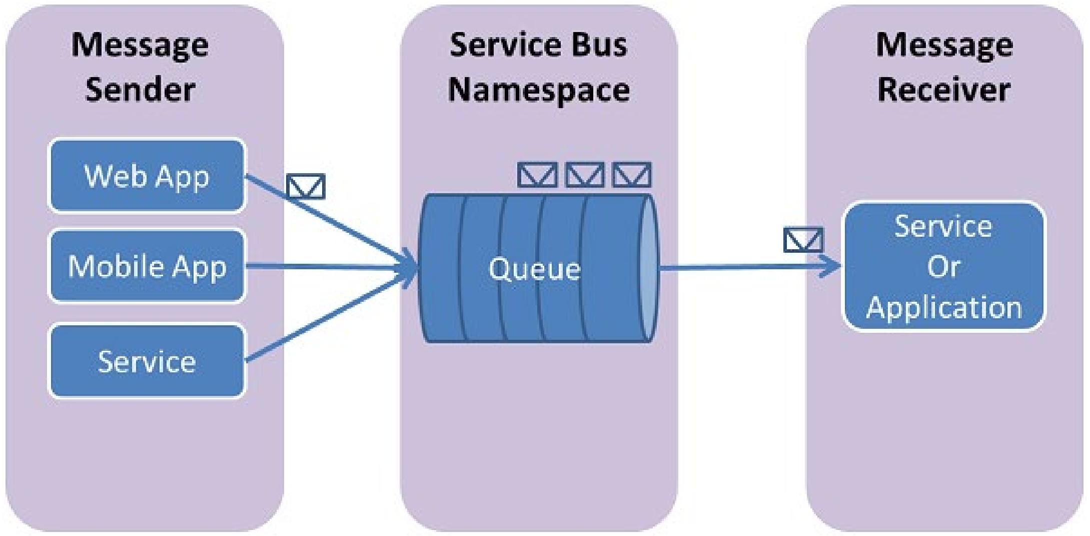


Some message brokers implement a *service bus* scheme – can be simpler and offer better transaction *rollback* but may be harder to scale

RabbitMQ message Broker



Azure service bus



Some common Message Brokers

- Amazon Web Services
- Apache ActiveMQ
- MQTT-compatible brokers
- Google Cloud Pub/Sub
- IBM MQ Series
- JBoss Messaging
- Microsoft Azure Service Bus
- Microsoft Biz Talk Server
- Open Message Queue
- Rabbit MQ
- SAP PI

MQTT

MQTT is an industry standard for messages that support IoT – the Internet of Things

- MQTT defines message formats and protocol for exchanging messages with “things”

- Many clients and message brokers have been implemented using the MQTT specification
- Generally, interoperability works (up to the limits of support for the spec)

- MQTT 5 is the current version

- See mqtt.org for more information

Recall the IoT concept – the "Internet of Things" – connection of all sorts of devices over IP networks

Typically, IoT devices are thought of as:

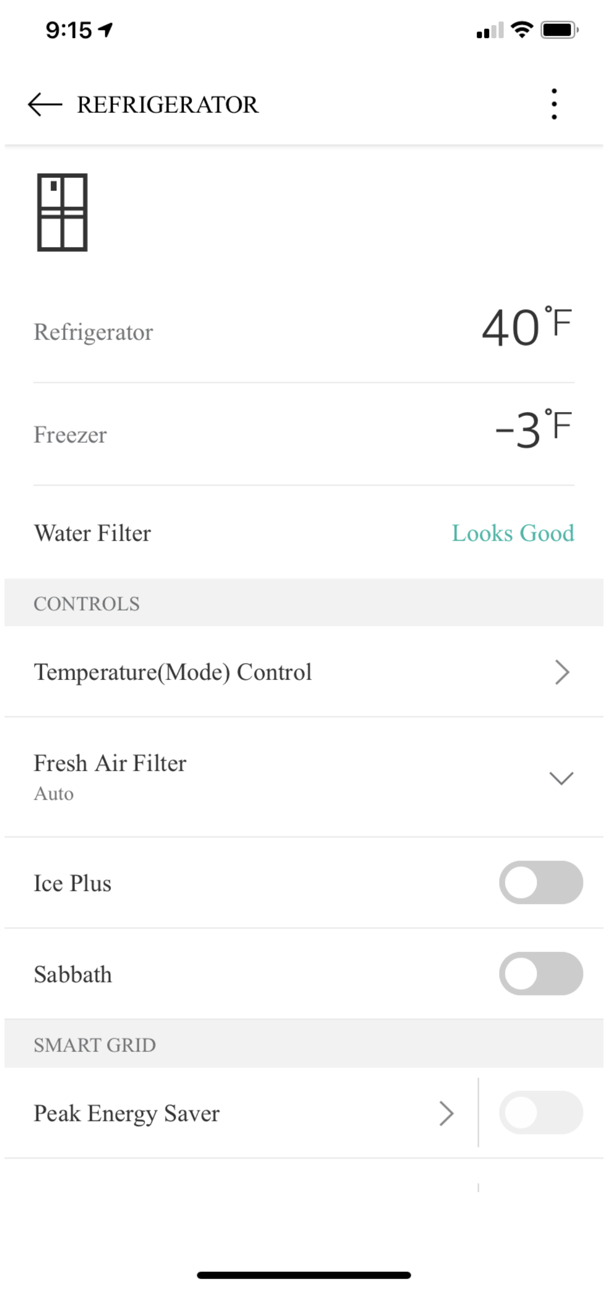
- Lightweight computing devices
- Unmanned
- Data sources or consumers
- Distributed geographically
- Low cost

While the usual web protocols work fine for this case, we need to think about the system architecture needs

Hence, MQTT and related architectures

My refrigerator is an IoT device

I have no idea why this is useful


9:15

REFRIGERATOR

Refrigerator 40˚F

Freezer -3˚F

Water Filter Looks Good

CONTROLS

Temperature(Mode) Control

Fresh Air Filter
Auto

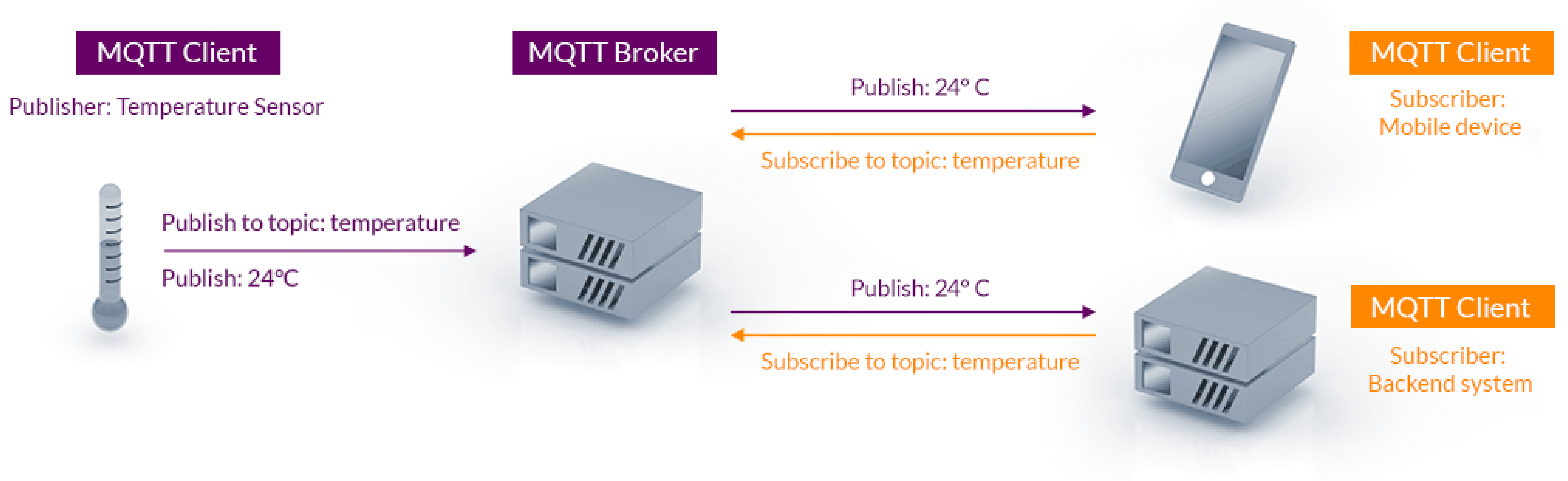
Ice Plus

Sabbath

SMART GRID

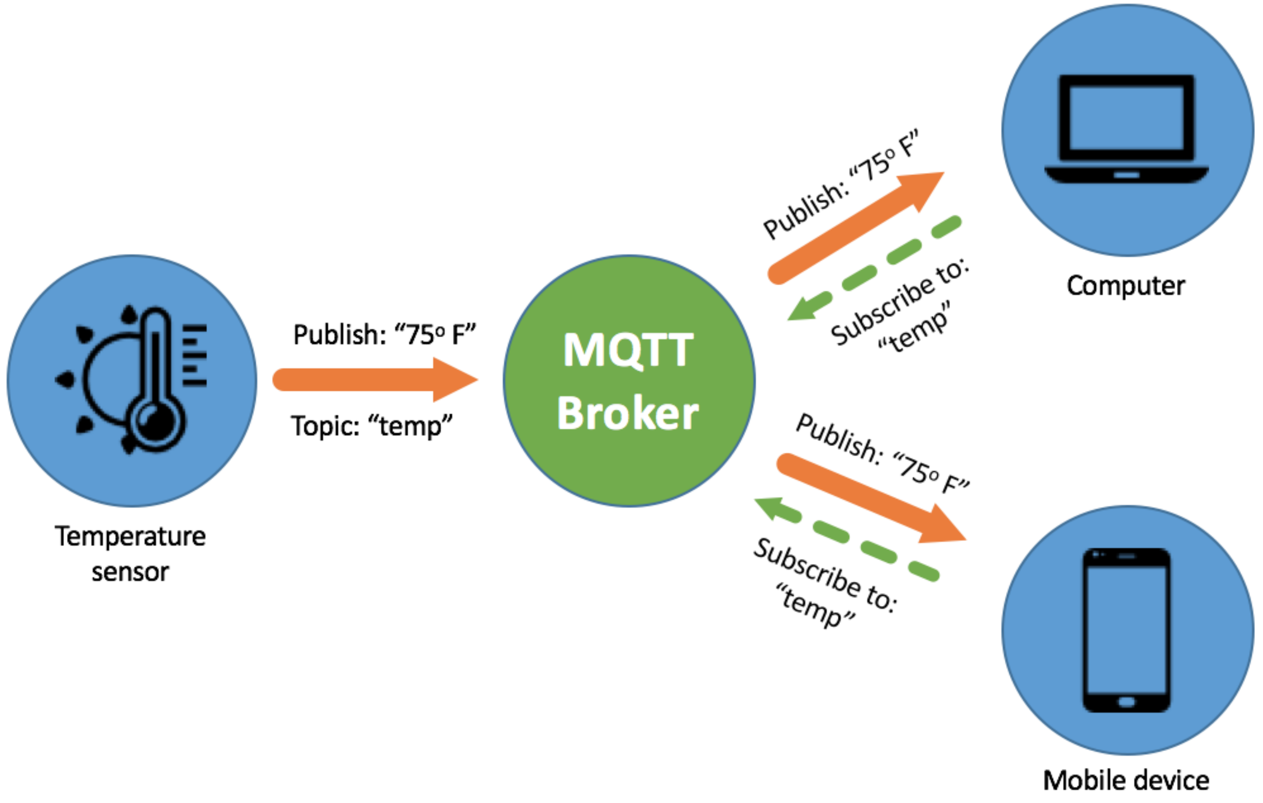
Peak Energy Saver

In MQTT, both publishers and subscribers are clients – lightweight client implementations are possible


MQTT Client
Publisher: Temperature Sensor
Publish to topic: temperature
Publish: 24°C
MQTT Broker
Publish: 24° C
Subscribe to topic: temperature
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Mobile device
MQTT Client
Subscriber: Backend system

MQTT supports broadcasting through publish/subscribe

This is ideal for sensor and other IoT applications



MQTT Quality of Service (QoS) defines three levels of required *integrity* of a particular message

- QoS 0: a message will be delivered at most once
  - but maybe not at all
- QoS 1: a message will be delivered at least once
  - but maybe more than once
- QoS 2: a message will be delivered exactly once
  - nice but requires more overhead

Many MQTT-compatible commercial and open-source implementations exist

- See https://en.wikipedia.org/wiki/Comparison_of_MQTT_implementations for a current list
- Some common ones include:

Implementation	Developed by	Open source	Written in	Type
Adafruit IO	Adafruit	Yes	Ruby on Rails, Node.js,[2]Python[3]	Client
HiveMQ MQTT Client	HiveMQ	Yes	Java	Client
HiveMQ Community Edition	HiveMQ	Yes	Java	Broker
HiveMQ	HiveMQ	No	Java	Broker
M2Mqtt	Eclipse	Yes	C#	Client
Paho MQTT	Eclipse	Yes	C, C++, Java, JavaScript, Python, Go	Client
VerneMQ	VerneMQ/Erlio	Yes	Erlang/OTP	Broker
eMQTT5	Cyril Russo	Yes	C++	Client

Here is a simple example client using HiveMQ community edition in Java; it communicates to a test MQ broker at broker.hivemq.com

```java
public class AsyncDemo {

    public static void main(final String[] args) throws InterruptedException {

        final Mqtt5AsyncClient client = Mqtt5Client.builder().serverHost("broker.hivemq.com").buildAsync();

        client.connect()
            .thenAccept(connAck -> System.out.println("connected " + connAck))
            .thenCompose(v -> client.publishWith().topic("demo/topic/b").qos(MqttQos.EXACTLY_ONCE).send())
            .thenAccept(publishResult -> System.out.println("published " + publishResult))
            .thenCompose(v -> client.disconnect())
            .thenAccept(v -> System.out.println("disconnected"));

        System.out.println("see that everything above is async");
        for (int i = 0; i < 5; i++) {
            TimeUnit.MILLISECONDS.sleep(50);
            System.out.println("...");
        }
    }
}
```

Today’s Objectives

Middleware
- Concept
- Distributed Objects
- Message-oriented Middleware
- Application Servers
- Message Brokers

An example: MQTT
- The Internet of Things
- MQTT – clients and brokers
- Implementations