Edge Computing

Lecture 08: Edge & Cloud: Middleware

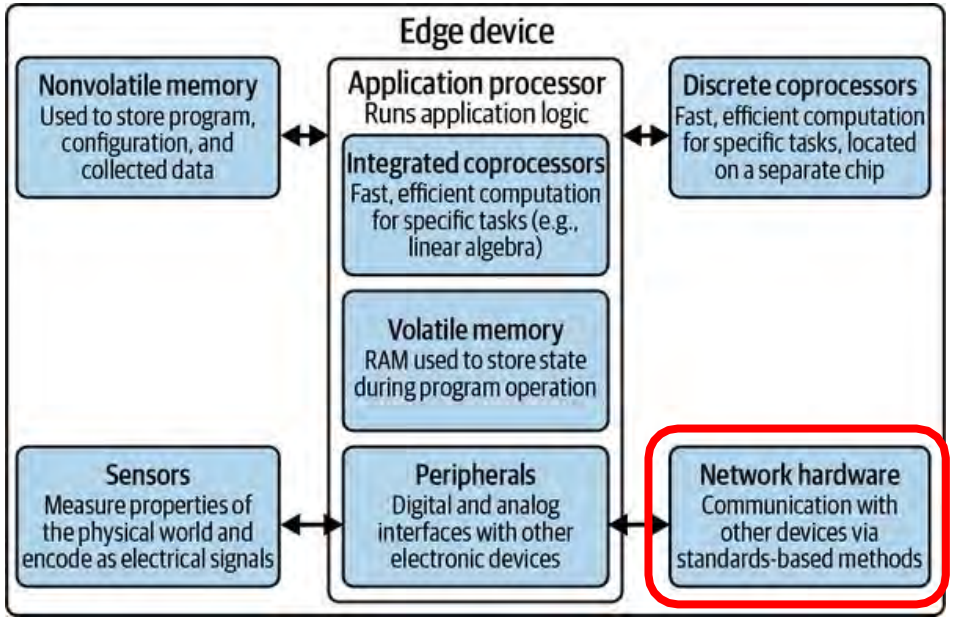
Paper sources

- Conference programs
  - MLSys: Machine Learning Systems, e.g. MLSys'23
  - Mobisys: Mobile Systems and Applications, e.g. Mobisys'23
  - CVPR: Computer Vision and Pattern Recognition, e.g. CVPR'23
  - SEC: Symposium on Edge Computing, e.g. SEC'23
- Google scholar
  - Search by topic, e.g. drone perception, smart agriculture
  - Search by name, e.g. Bill Dally, MLSys board member and steering committee
- Source file: UCR Library
- Is it a good paper?
  - Top tier conference? CSRankings
  - High citation?
  - Github source code with many stars?

Recap

Hardware

- CPU
- Memory
- Cache
- RISC vs CISC
- Special accelerators
- Sensors



Agenda

Networking with others

- Middleware: What and why
- Design goals
- Examples

System Architect

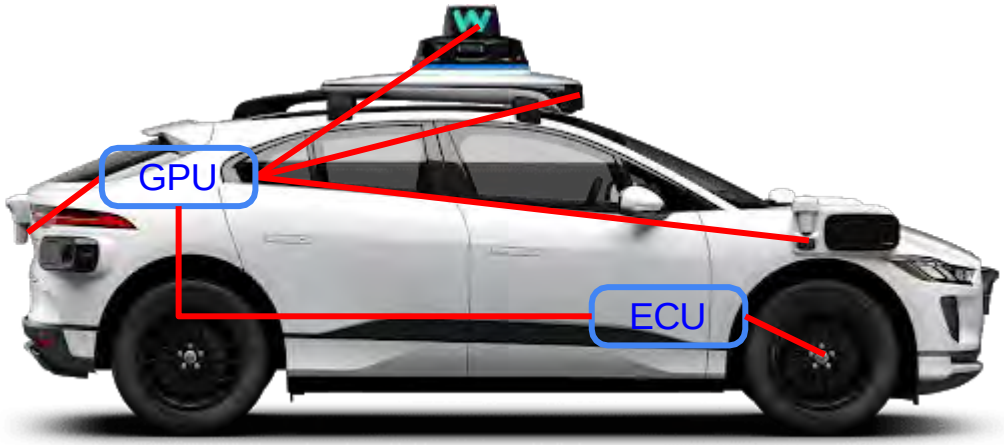
- A person who designs hardware, software, or networking applications and services of a specified type for a business or other organization
- Edge computing: a designer’s perspective

Designing an autonomous ride-sharing service

- What data? How does the data flow?

Designing an autonomous ride-sharing service

- What data? How does the data flow?

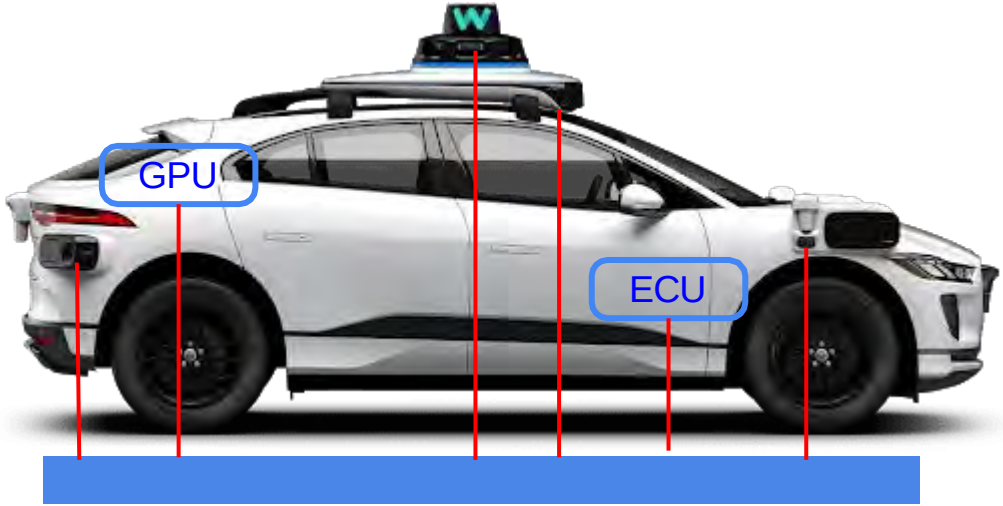


G A S

TCM	Transmission control module
PCM	Powertrain control module
ECM	Engine control module

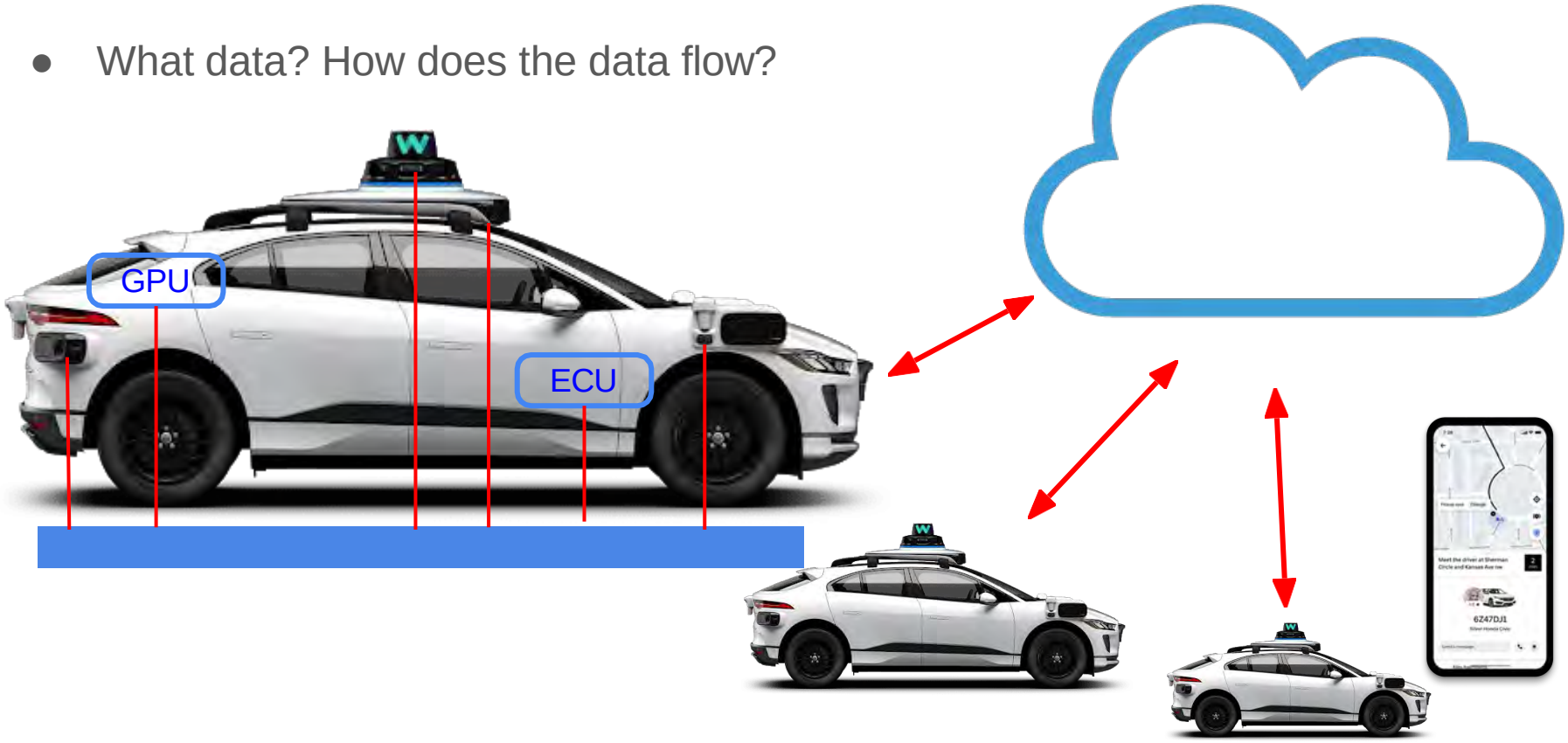
Designing an autonomous ride-sharing service

- What data? How does the data flow?



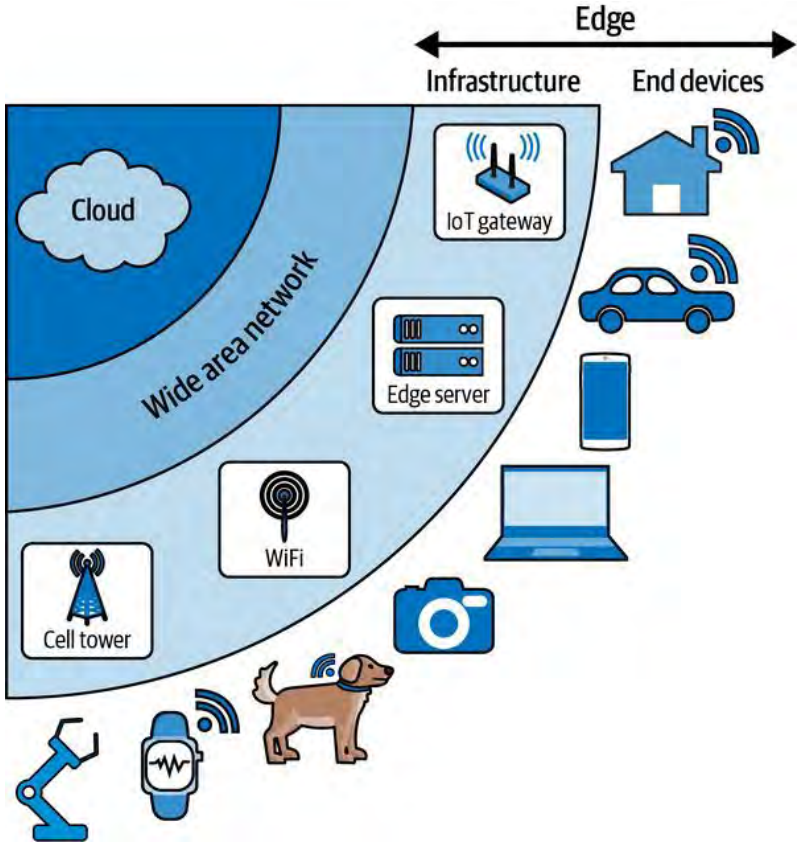
Designing an autonomous ride-sharing service

- What data? How does the data flow?



Middleware

- Broadly defined:
  - Software between OS and App.
- ***Hidden translation layer***, enabling ***communication and data management*** for distributed applications.
- In edge computing context, a bridge
  - Among edge devices
  - Between edge and cloud

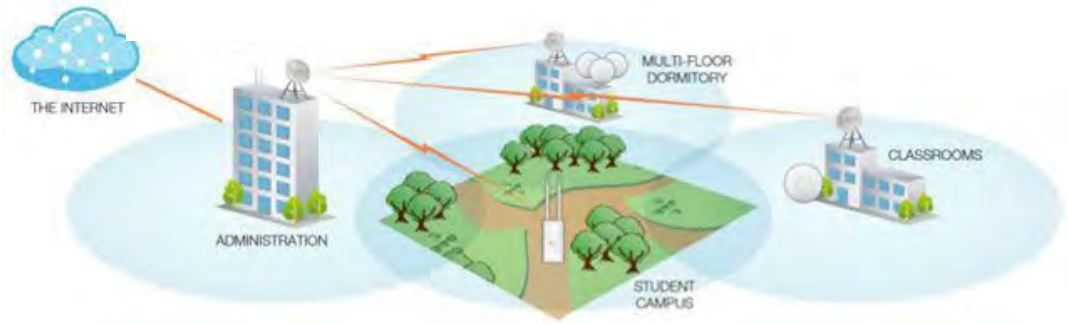


Situnayake, D., & Plunkett, J. (2023). AI at the Edge. O'Reilly

Middleware Design Goals

- Ad-hoc Discovery
  - Automated communication channel setup
  - Dynamically join or leave the channel
  - Data flow integrity and security

UCR-Secure



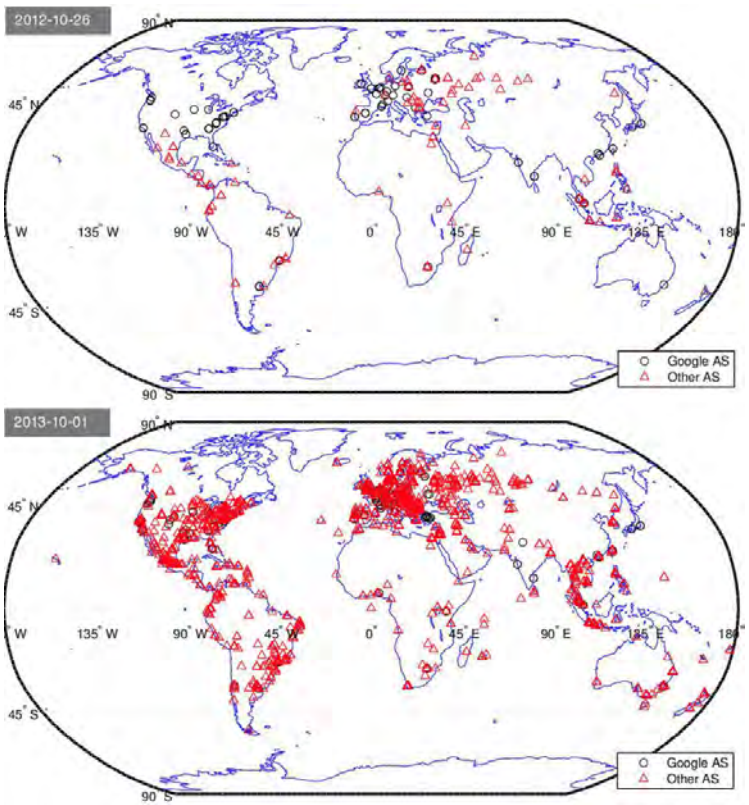
Middleware Design Goals

- Run-time Execution
  - Remotely execute programs (on edge devices or cloud) from a central hub
    - Deploy (code download)
    - Execution, start / stop service
    - Update (code)
    - Delivery of results

Google colab

Middleware Design Goals

- Minimal Task Disruption
  - Anticipate and handle disruptions
  - Reliability of middleware, via replica, backup, etc.
  - Resilience to mobility, network disconnections, memory overflow, etc.



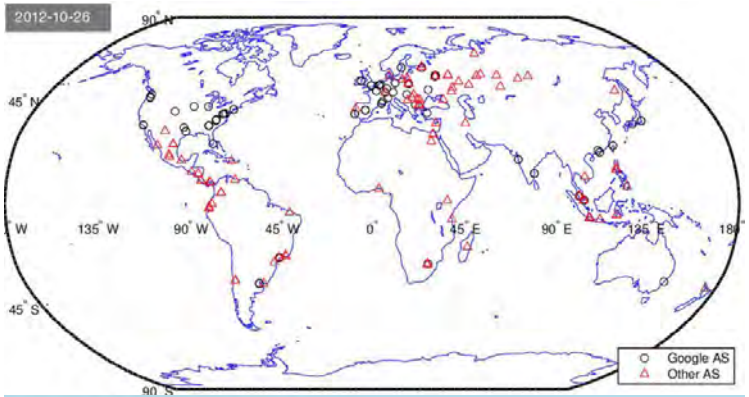
Google’s CDN expansion

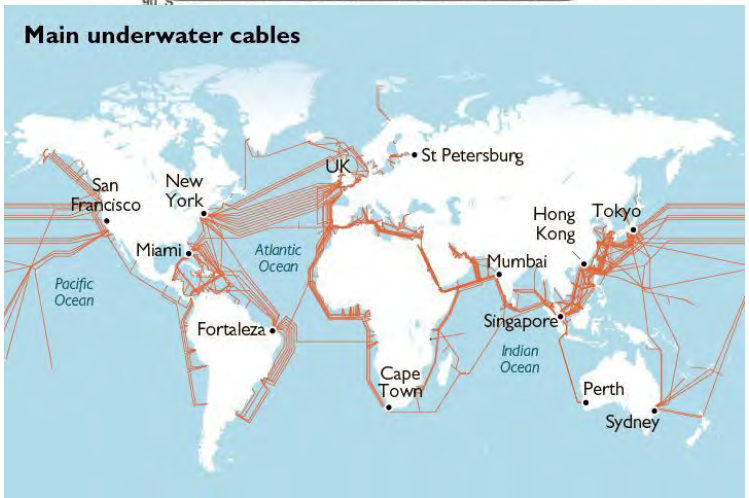
Mapping the Expansion of Google’s Serving Infrastructure, Calder et al, Sigcomm 2013

Middleware Design Goals

- Operation Overhead
  - Bandwidth
  - Latency
  - Compute
  - Energy

Google's CDN expansion





Mapping the Expansion of Google's Serving Infrastructure, Calder et al, Sigcomm 2013
https://law-in-action.com/tag/trans-atlantic-fiber-cable/

Middleware Design Goals

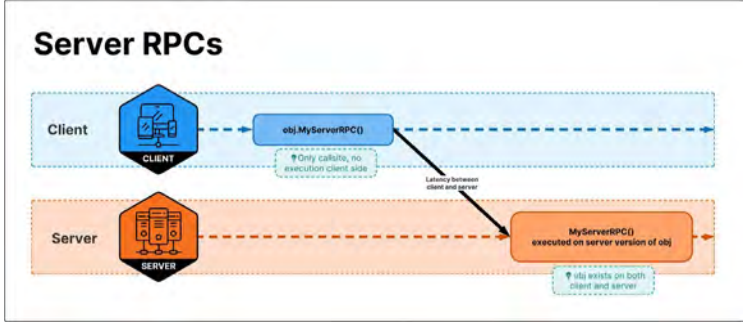
- Context-aware adaptive design
  - Hardware
  - Power budget
  - Network condition
  - User activity
  - Dynamics

Design Goal & Solutions

- Ad-hoc Discovery
- Run-time Execution
- Minimal Task Disruption
- Operation Overhead
- Context-aware adaptive design

- Probing & selection of participants
- RPCs, Virtualization
- Replica, distributed system
- Resource management, optimization
- Context monitoring and prediction

Remote Procedure Calls (RPC)

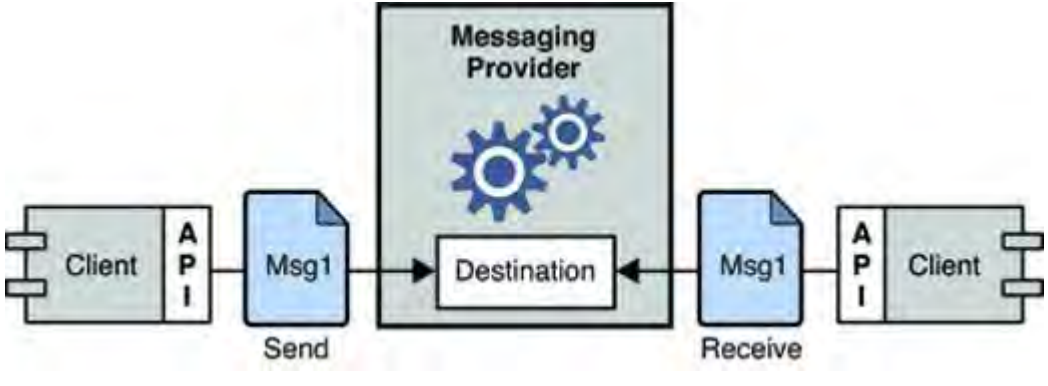


- An object and its methods (or a procedure and its parameters) are “remoted” such that an invocation of a procedure or method can happen across a network separation.
- A local proxy (“stub”) mimic the interface of the remote object and its methods
- An application (client) make a call for remote execution (server) via the stub as if the call was executed locally.
- Tightly-coupled interfaces:
  - each application know the details of how other applications wants to communicate
- Synchronous vs Asynchronous?

https://learning.oreilly.com/library/view/enterprise-service-bus/0596006756/ch05.html#esb-CHP-5-SECT-1.1

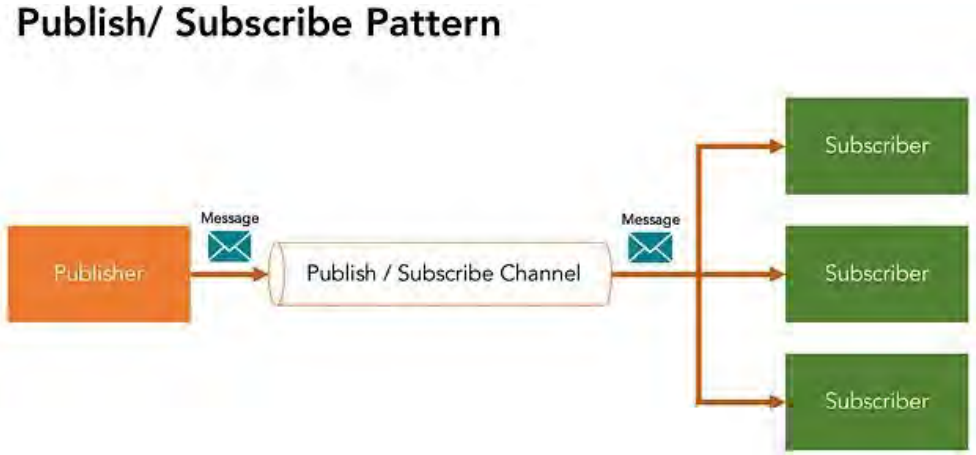
Message-Oriented Middleware (MOM)

- Passing of data between applications using a communication channel that carries self-contained units of information (messages).
- Messages are usually sent and received asynchronously.
- Loosely coupled:
  - MOM API can finish send and receive, clients can be heterogeneous (python vs c++).
  - Clients don’t need to know about each other’s protocol/existence
- Asynchronous design (mostly)



Pub / Sub

- Publish / Subscribe
- Messaging pattern where publishers use message distribution nodes (brokers) where subscribers can retrieve relevant message.

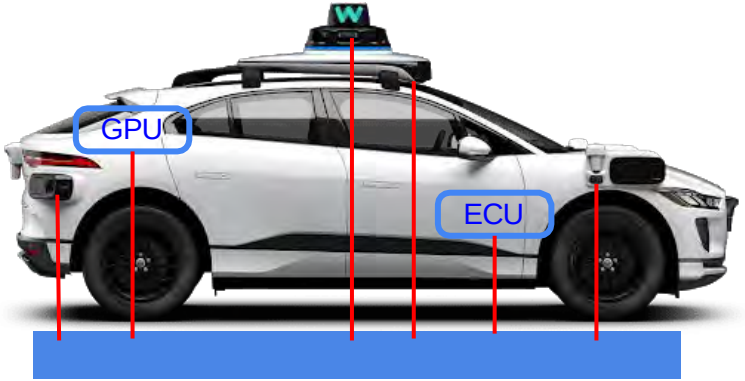


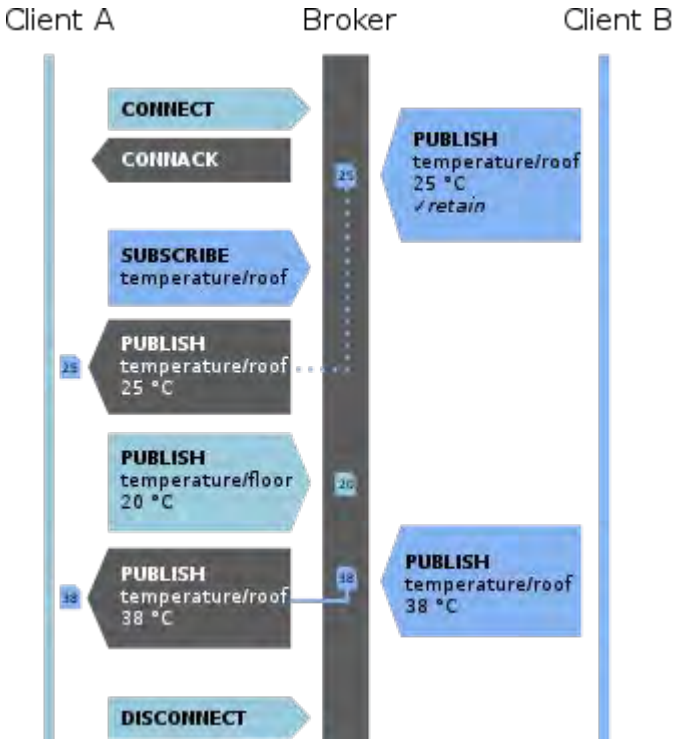
Pub / Sub

- Pros
  - Flexible, dynamic join (subscribe) and remove (unsubscribe)
  - Loosely coupled, publishers and subscribers don’t need to know about each other
  - Scalable design
- Cons
  - Reliability: single node of failure (broker/brokers)
  - Network saturation
  - Latency

MQTT

- Message Queue Telemetry Transport
  - A pub / sub network protocol
  - Four types of messages
    - Connect
    - Disconnect
    - Publish
    - Subscribe



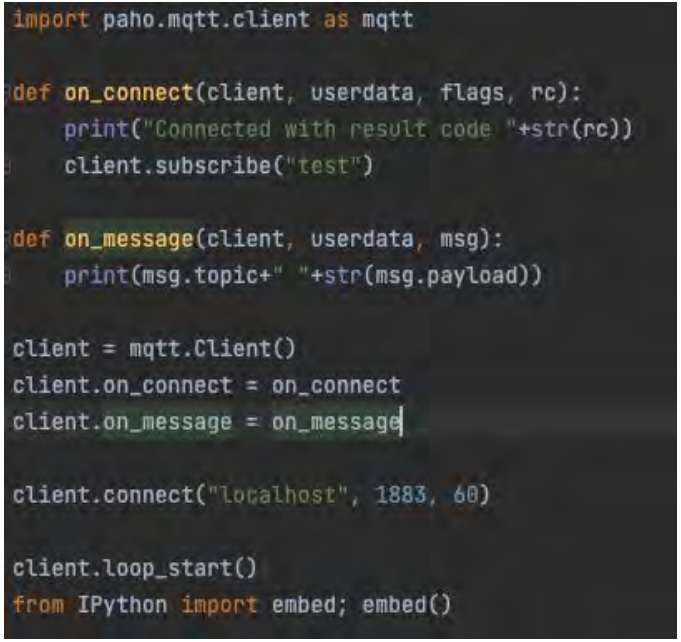


MQTT Quality of Service

- At most once - (fire and forget)
  - the message is sent only once and the client and broker take no additional steps to acknowledge delivery.
- At least once - (acknowledged delivery)
  - the message is re-tried by the sender multiple times until acknowledgement is received.
- Exactly once - (assured delivery)
  - the sender and receiver engage in a two-level handshake to ensure only one copy of the message is received.

Example: Paho-MQTT + Mosquitto (broker)

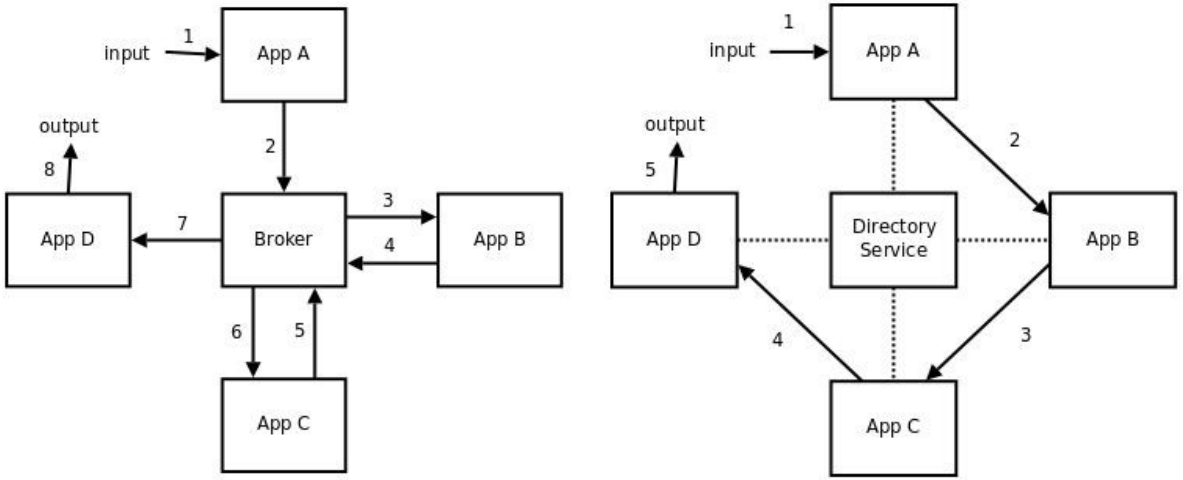
- Install mosquitto broker: Download | Eclipse Mosquitto
- Install paho-mqtt: paho-mqtt · PyPI
- Start mosquitto
- Run the test script on the right
- Publish a message under topic “test”
  - client.publish(“test”, “Hello world!”)



```python
import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe("test")

def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("localhost", 1883, 60)

client.loop_start()
from IPython import embed; embed()
```

Brokerless MOM: Example: ZeroMQ or ZMQ

- Lightweight
  - Decouple broker’s two function: pub / sub register, data transfer
  - Use directory service for pub / sub registration
  - Let clients handle data transfer
- N-to-N patterns
  - Fan-out
  - Pub-sub
  - Task-distribution
  - Request-reply



Other Examples:

1. AWS IoT for the Edge
2. Azure Stack Edge
3. Kaa IoT Platform
4. Apache Kafka
5. ThingsBoard
6. openHAB
7. Eclipse Kura
8. FOGLAMP
9. EdgeX Foundry
10. Apache Edgent

Other Examples:

1. AWS IoT for the Edge
2. Azure Stack Edge
3. Kaa IoT Platform
4. Apache Kafka
5. ThingsBoard
6. openHAB
7. Eclipse Kura
8. FOGLAMP
9. EdgeX Foundry
10. Apache Edgent

Quiz 000: Middleware Examples

- What does it include? (programming API, network modeling, device management, all the main features)
- License type (open source, pay as you go, contract based, etc.)
- At what layer is it deployed (cloud, Edge Network, Fog, Holistic: all of them)
- Is it general purpose or for specific cases?
- Compatibility: Programming language, OS, device, cloud service, etc.

Summary

- Middleware
- Design goals
- Examples
  - MOMs, Pub/Sub, MQTT

Next Lectures

- Lab 3: client-server communication
- Lab 4: connecting to the cloud
  - Google cloud platform (GCP)
  - Amazon Web Service (AWS)
- Paper presentation next week!
- Ethnics & Security
- Edge Computing Research
  - Guest Lecture from CISL lab