

# AI and distributed systems

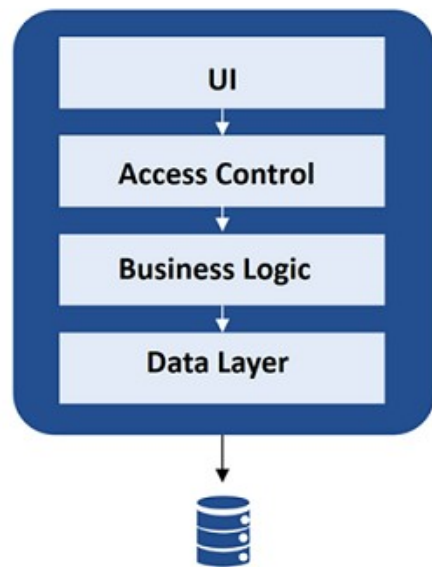
---

- Originally, AI was used to build standalone AI applications (e.g., chess players)
- AI functionalities are more and more embedded into software applications and systems (smartphones, office applications, robots, ...)
- Such systems are frequently distributed and heterogeneous
- Claim: there is a non trivial interplay between the AI techniques and the architecture of the distributed systems where they are used

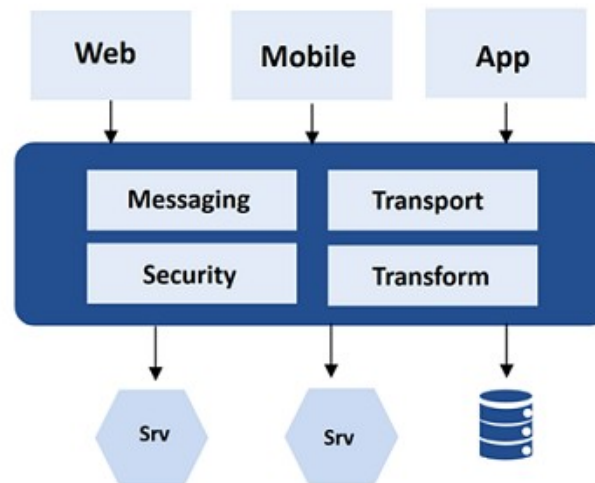
# Distributed systems architecture

- Distributed system architecture evolved over the years from simple client-server to more complex patterns

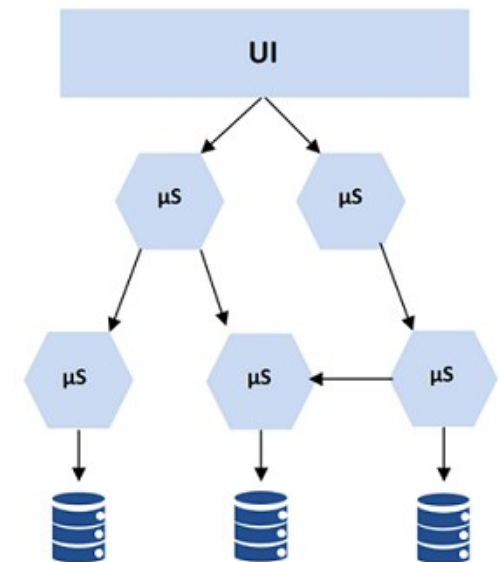
## Evolution of Software Architectures



Monolithic



Service-oriented



Microservices

# Microservices

---

- A software architectural style advocating the structuring of systems as the composition of small, loosely-coupled microservices
- Each microservice provides a restricted and coherent set of capabilities
- Microservices are deployed independently, frequently into containers (e.g., Docker) on the Cloud or on the edge
- Microservices can be independently scaled and updated
- Aim: maximizing flexibility and scalability

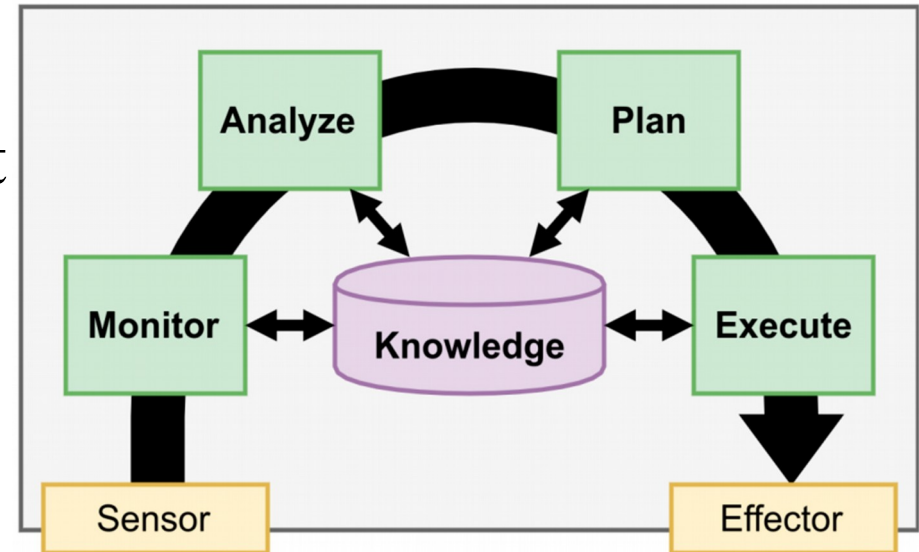
# How to add AI capabilities to microservices?

---

- Microservices should adapt to changing environment and user requirements
- AI capabilities can help in this direction, minimizing human intervention
- We consider the autonomic computing (self-\*) approach
  - Suitable to monitor changes in the environment and adapt accordingly

# The **MAPE-K** feedback control loop

- **M**onitor: acquires data from the system and its environment
- **A**nalyze: refine and extract information from data
- **P**lan: decide which actions need to be taken to reach system goals
- **E**xecute: takes the planned actions
- **K**nowledge: keeps track of the known information



# Design decision: who is in charge?

---

- A main design decision for autonomic microservices: who is in charge of the MAPE phases and of K?
- Different possibilities:
  - The microservices
    - Each microservice or dedicated microservices
    - MAPE-K as a service?
  - The infrastructure
    - Containers, container managers (e.g., Kubernetes), the Cloud infrastructure
  - The IT personnel
  - A combination of the above

# Some general tradeoffs

---

- If IT personnel is in charge: the system is not autonomic
- If the infrastructure is in charge: autonomic infrastructure managing dumb microservices
  - Vendor lock-in: moving the system to a different infrastructure causes loss of autonomic capabilities
- Microservices are in charge: not always easy
  - May not have access to all the information
  - Need for coordination
- Both infrastructure and microservices: need for an interface
  - If not standard can cause again vendor lock-in



# Sample instance: monitoring

---

- Who is in the best position to get the data?
- The infrastructure for environmental data
  - E.g., allocated and used resources
- Microservices for internal data
  - E.g., which functionalities are more heavily used
- IT personnel has understanding of (changing) requirements
  - E.g., which functionalities and non-functional properties are more relevant at a given moment
- Having all the actors interact for monitoring may require complex coordination and interfaces

# Future directions and challenges

---



- A paper on this topic is currently submitted to IEEE Software
- How the tradeoffs change in different application areas or in other architectural styles (e.g., serverless)?
- Which are suitable interface to share responsibility of phases among different actors?
- How to combine distribution and flexibility with timely and precise adaptations?
- How to provide autonomic capabilities in multicloud scenarios?

Finally

---

**Thanks!**

**Questions?**