LM in Informatica

Distributed Software Systems 2024-25

Paolo Ciancarini

University of Bologna, Italy

Who am I

Professor of Computer Science@UniBo

Researcher on sw technologies for distributed programming and systems

… and who are you?

Prerequisites

- This course presents the fundamental ideas on distributed systems and their principles of construction
- We also present middleware technologies and the related methods for distributed programming

- Prerequisites:
  - Be a graduate student – this is an advanced course
  - Be able to design program (object oriented and patterns)
  - Basic notions of networking and operating systems
  - Basic notions of software engineering and architecture

Effort and scheduling

- Effort : 6 cfu
- Course duration: from sept to december
- Class hours:
  - Tuesday 12-14 room Ercolani 3
  - Thursday 13-15 room Ercolani 1

Syllabus

Introduction to distributed systems

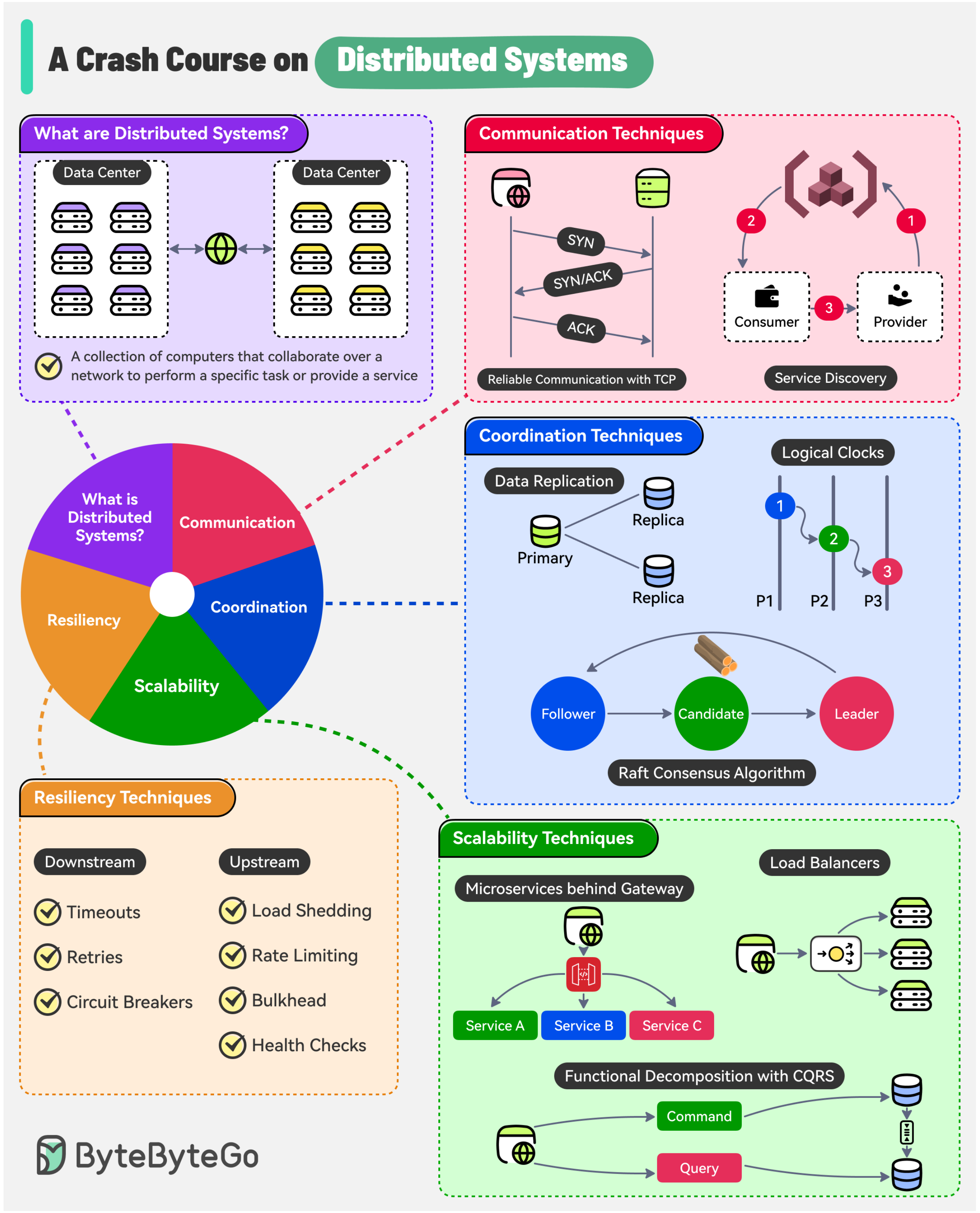
Principles of distributed programming

Middleware technologies

Theory issues

(eg. Distributed time, consensus)

Hands-on work on specific problems
in distributed programming



Goals of this course

- Understand principles and concepts underlying **distributed computing:** communication, concurrency, coordination and related algorithms
- Learn how to **design** simple reliable **distributed applications** using (open source) **middleware**
- Gain **practical** experience on the development of simple **distributed systems** (eg. in **Java** or **Python**)

Exam

- Labs 10% each (please present your result of at least 5 labs) – max 50%
- Midterm report & presentation: 50%
- Attendance: +/- bonus up to 10%
  (negative if you do not attend classes or get too many late deliveries of labs)

Mid-term (individual) presentation

Your task:

- you have to deliver a small paper (five pages) and a presentation (at least 15 slides, talk 20')

Possible topics (examples):

- Middleware for distributed applications
- Important examples of distributed systems (architecture, behavior)

Where to look for papers:

- IEEE Transactions on Parallel & Distributed Systems
- Journal of Parallel and Distributed Computing (Elsevier)
- Distributed computing (Springer)
- Journal of Internet services and applications (Springer)
- ACM Symposium on Principles of Distributed Computing

Suggested topics for midterm

A report on any distributed system with an interesting distributed sw architecture

Middleware technologies preferred

When you choose (write your choice on the googlesheet) I will send you some references (eg. Books)

Apache middleware	Other middleware
Apache Cassandra	Akka
Apache Flink	Celery
Apache Hadoop	Dapr
Apache Kafka	gRPC
Apache Karaf	Hyperledger Fabric
Apache Mesos	ISIS toolkit
Apache NiFi	ISTIO
Apache Openwhisk	Jini
Apache Pulsar	Kubernetes
Apache River	Omnitude
Apache RocketMQ	RabbitMQ
Apache Spark	Settlemint
Apache Storm	Keycloack
Apache Thrift	Swagger/OpenAPI
Apache Wildfly	ZeroMQ

Labs

Exercises

I will assign weekly during the course

Solutions (by individuals or pairs)

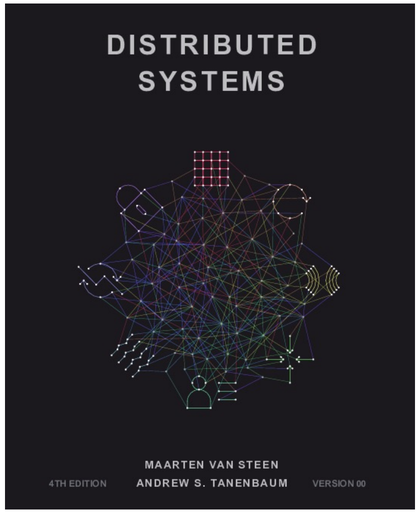
are expected after one week max, presented and discussed in class

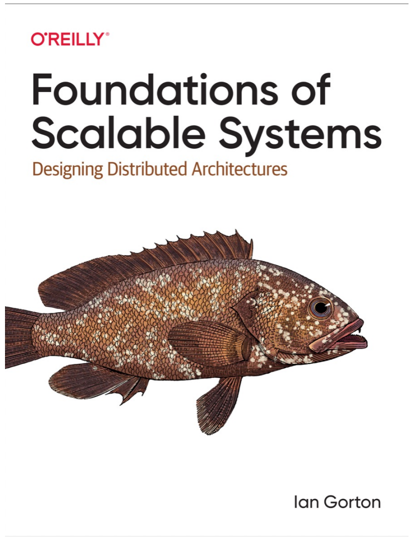
I will give at least 6 labs assignments

Channels

- **Telegram** (course): UniBo Distributed Sw Systems
- **Telegram** `@PaLoCaPa (personal)`
- **Email** `snmp://paolo.ciancarini@unibo.it/`
- **Linkedin:** search me and ask for contact!

Textbooks




O'REILLY
Foundations of Scalable Systems
Designing Distributed Architectures
Ian Gorton

vanSteen and Tanenbaum, *Distributed Systems*, 4ed, 2023

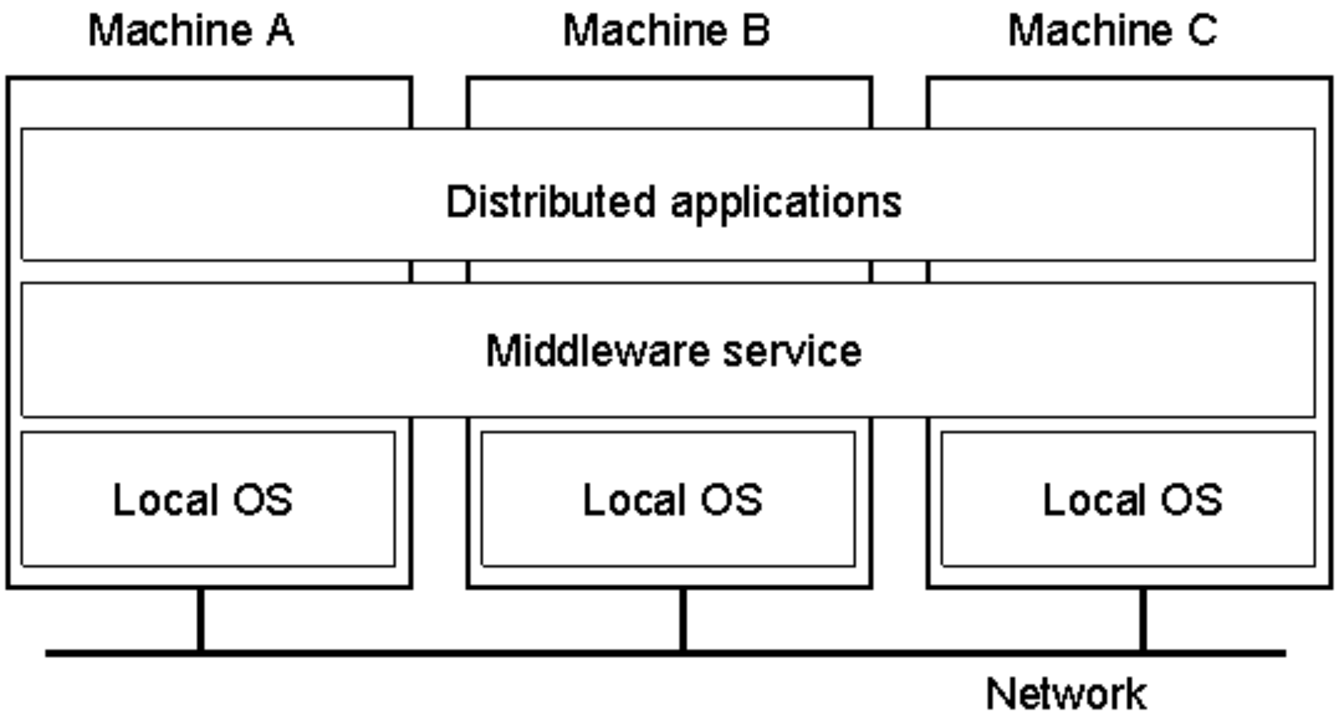
Gorton, *Foundations of scalable systems*, O'Reilly, 2022

Additional material (slides, papers, books) distributed during the course

Questions?

Definition: distributed system

A distributed system is a collection of independent computers that appears to its users as a single coherent system thanks to some **middleware** which makes transparent the underlying infrastructure



Lamport’s definition of a Distributed System

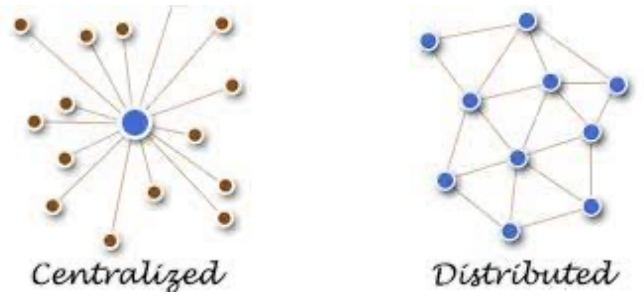
A distributed system is one in which
the failure of a computer
you didn’t even know existed
can render your own computer unusable

(Leslie Lamport)

Distributed vs centralized

Centralized system: state stored on a single computer

- simpler to design
- easier to understand
- global clock



Distributed system: state divided over multiple computers

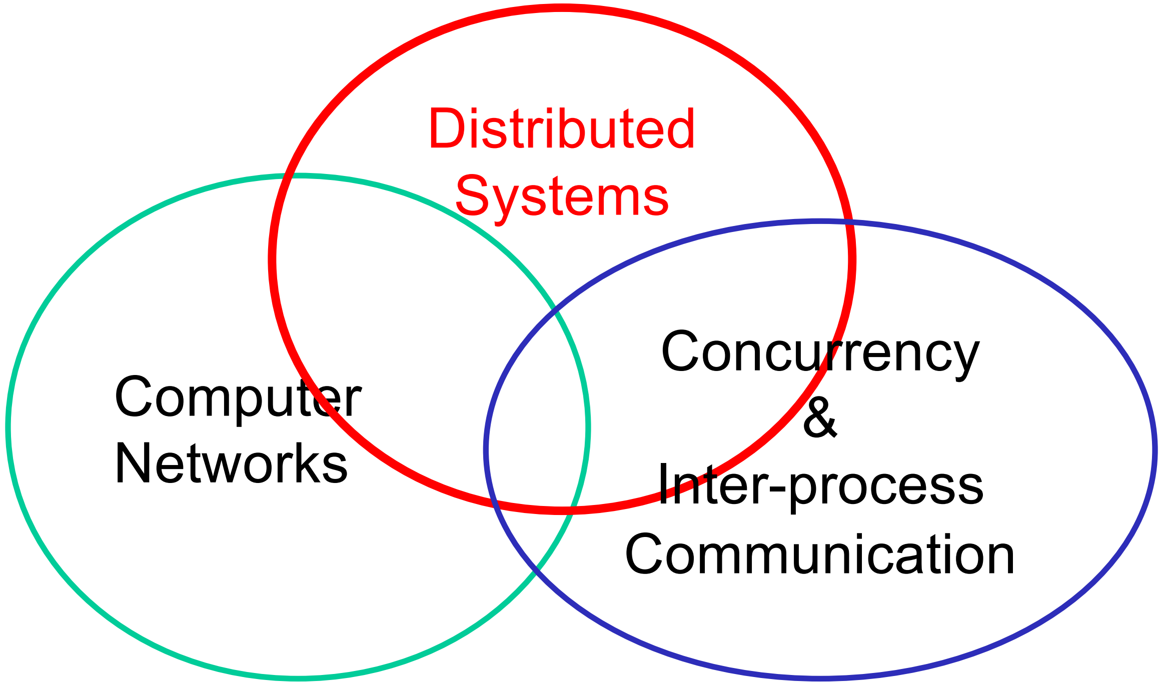
- Scalable
- Fault tolerant (under specific assumptions)
- Complex, No global time

Areas

Distributed systems hide network problems …

… but in order to understand and develop them, we need competencies in

- Computer networks and Internet
- Concurrency & Inter-process communication



Two consequences of distribution

- Data travel at (max) the speed of light, but usually the speed is quite slower
- Independent things fail independently

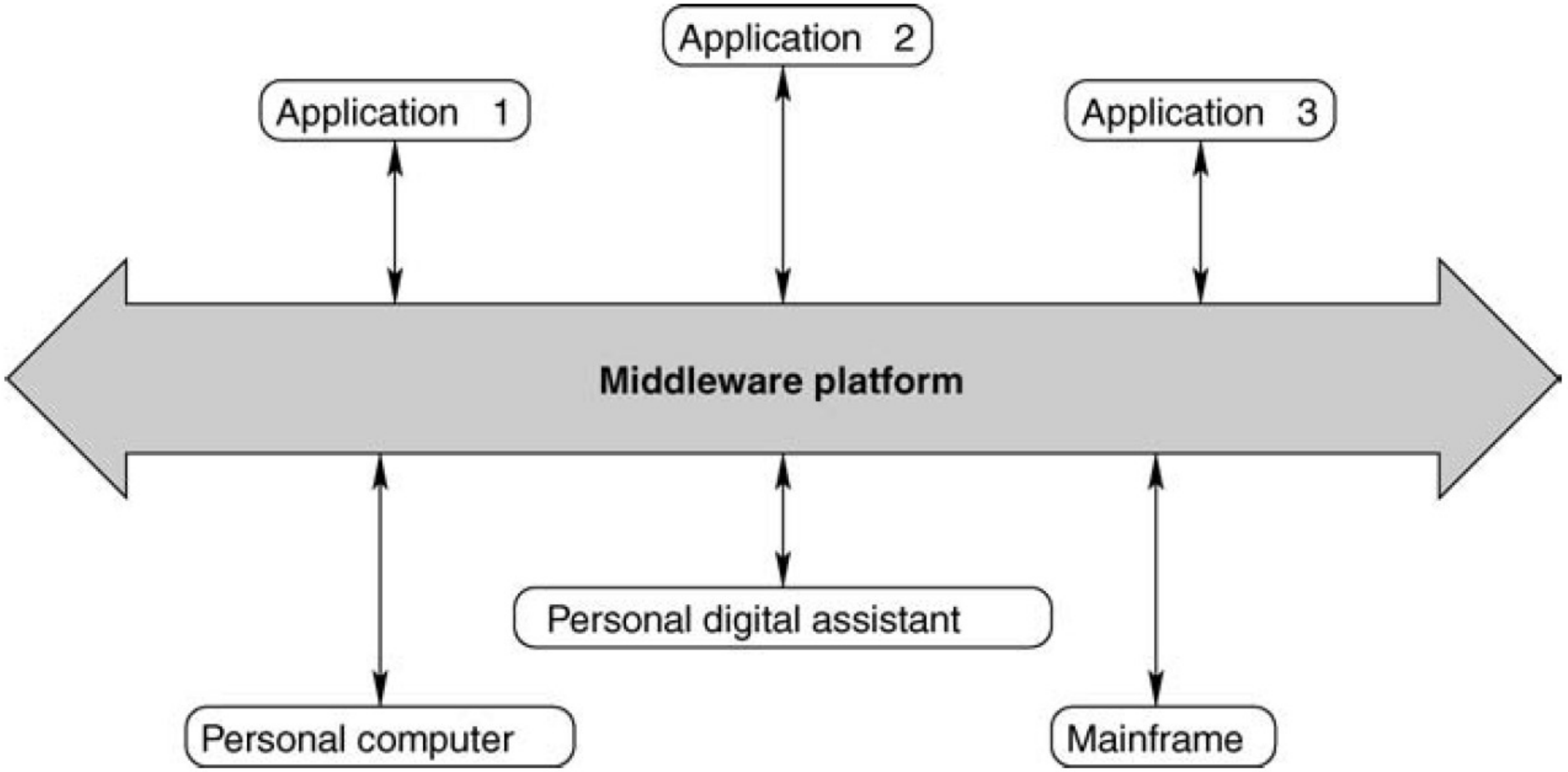
Building distributed systems means dealing with space, time, and having more than one active computing element.

These constraints allow several possible system designs, and after this course you'll have a better idea of how distance, time, and consistency models interact.

Languages for distributed computing

- There are problems in distributed system environments that do not exist in single-machine environments.
- Partial failure, concurrency, and latency are three problems that make distributed computing fundamentally different from local computing
- So languages for distributed programming are often normal languages (eg C++, Java, Python) extended with some middleware

Middleware



Middleware as an infrastructure for distributed systems

Variants of distributed systems

Internet of Things

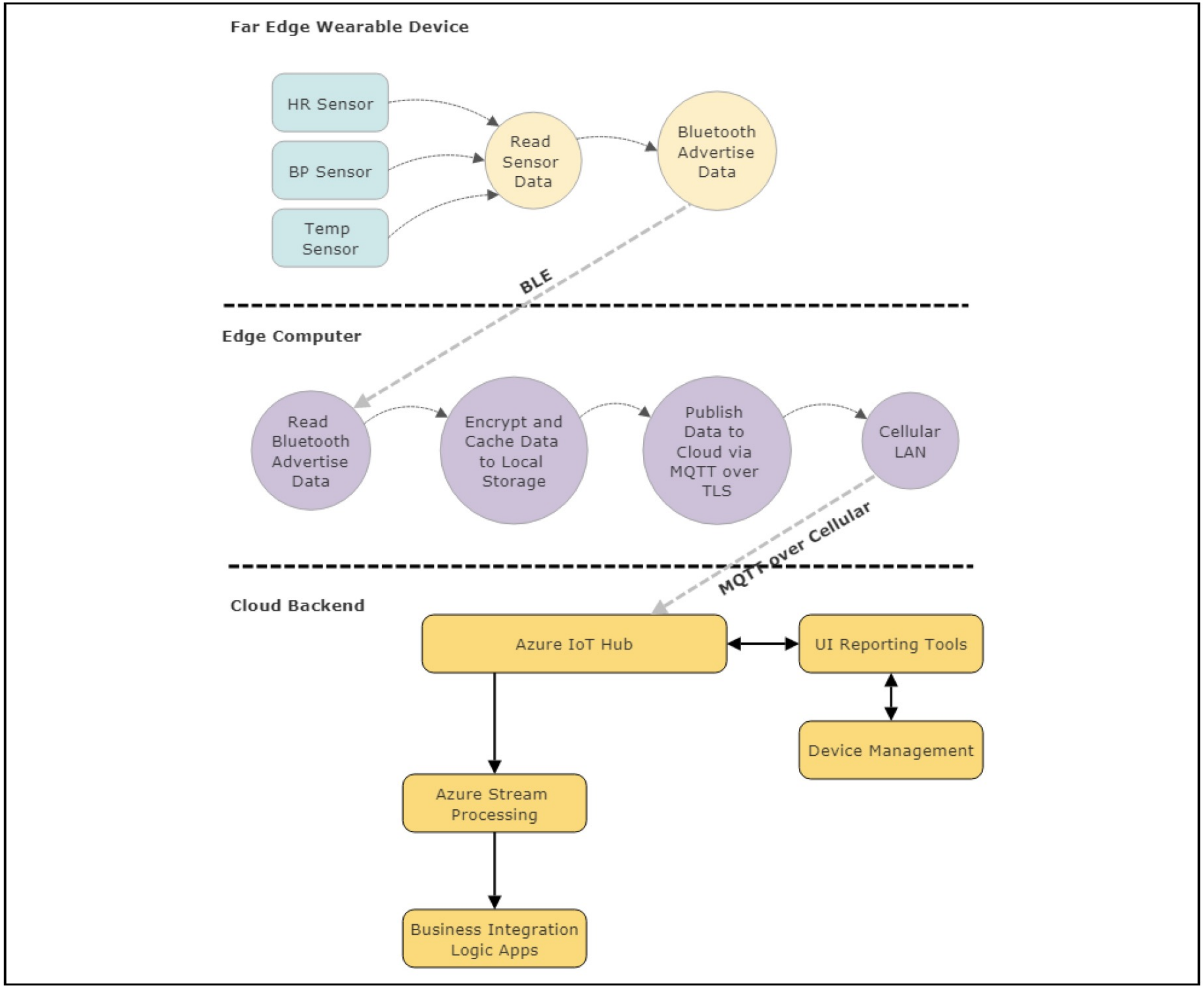
Cloud computing

Fog computing

Edge computing

Service oriented computing

Multi-agent systems



From Lea, *IoT and Edge computing for architects*, Packt 2020

Comparing centralized with distributed systems

Criteria	Centralized	Distributed
Economics	low	high
Availability	low	high
Complexity	low	high
Consistency	simple	difficult
Scalability	poor	good
Technology	homogeneous	heterogeneous
Security	high	low

Which problems should we solve when programming a distributed system?

Access (eg. How do we name resources?)

Transparency (we would like to ignore distribution issues)

Fault tolerance (we have multiple sources to get a service)

Latency (data speed is not infinite!)

… programming without shared memory and no absolute (unique) time is DIFFICULT

Transparency

- A transparency is some aspect of a distributed system that is hidden – abstracted - from the user (programmer, system developer, user or application program), so it can be ignored
- A transparency is implemented by some set of mechanisms in the distributed system at a layer below the interface where the transparency is required
- A number of basic transparencies have been defined; not all of these are appropriate for every distributed system, or are available at the same level of interface.
- In fact, all transparencies have an associated cost, and it is extremely important for implementor to be aware of this cost

There are several transparencies

Transparency: access

Access transparency There should be no difference between local and remote access methods. In other words, the cost of explicit communication may be hidden.

For instance, from a user's point of view, access to a remote service such as a printer should be identical with access to a local printer.
From a programmers point of view, the access method to a remote object may be identical to access a local object of the same class.

Moreover, since the semantics of remote access are more complex, particularly in case of failure, this means that a local access should be a special case of remote access.

Remote access will not always look like local access in that certain facilities may not be reasonable to support (for example, global exhaustive searching of a distributed system for a single object may be unreasonable in terms of network traffic).

Transparency: concurrency

Concurrency transparency Users and applications should be able to access shared data or objects without interference between each other.

This requires very complex mechanisms in a distributed system, since there exists true concurrency rather than the simulated concurrency typical of a centralized system.

For example, a distributed printing service must provide the same atomic access per file as a central system so that printout is not randomly interleaved.

Transparency: replication and fault tolerance

Replication Transparency If the system provides replication for availability or performance reasons, it should not concern the user (as for all transparencies, we include the applications programmer as a user)

Fault Transparency If software or hardware failures occur, these should be hidden from the user. For instance, in a networked system, it is often hard to tell the difference between a failed and a slow running process or processor. Fault transparency can be difficult since partial failure of the communication subsystem is possible, and this may not be reported.
As far as possible, fault transparency will be provided by mechanisms that relate to access transparency. However, when the faults are inherent in the distributed nature of the system, then access transparency may not be maintained. The mechanisms that allow a system to hide faults may result in changes to access mechanisms (e.g. access to reliable objects may be different from access to simple objects).

Transparencies: location and migration

Location transparency The details of the topology of the system should be of no concern to the user. The location of an object in the system may not be visible to the user or programmer. (This differs from access transparency in that both the naming and access methods may be the same). Names may give no hint as to location.

Migration transparency If objects (processes or data) migrate (to provide better performance, or reliability, or to hide differences between hosts), this change of location should be hidden from the user.

Transparency: performance, scaling

Performance Transparency The configuration of the system should not be apparent to the user in terms of performance. This may require complex resource management mechanisms. It may not be possible at all in cases where resources are only accessible via low performance networks.

Scaling Transparency A system should be able to grow without affecting application algorithms. Graceful growth and evolution is an important requirement for most enterprises. A system should also be capable of scaling down to small environements where required, and be space and/or time efficient as required.

Transparencies in a Distributed System

Transparency	**Description**
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be shared by several competitive users
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Hands on Lab 1

Understanding transparency in distributed computing

1. **Goal** The objective of this exercise is to grasp the fundamental aspects of transparency in a distributed system.
2. **Scenario**: Imagine you are part of a team developing a distributed file storage system. Users should be able to access files stored on this system as easily as if they were stored on a local disk. Your task is to make the system transparent in terms of location transparency.
3. **Hints** for solution:
    1. Discuss with your team what location transparency means in the context of a distributed file storage system.
    2. Think about how to abstract the physical location of files so that users don't need to be aware of where the files are stored.
    3. Consider techniques like distributed file naming, mapping logical file names to physical locations, and ensuring seamless access regardless of server locations
4. **Deliverable**: prepare a short presentation outlining your approach to achieving location transparency in the distributed file storage system. Describe the proposed techniques and how they address potential issues.

Questions?