SOC

LECTURE NOTES

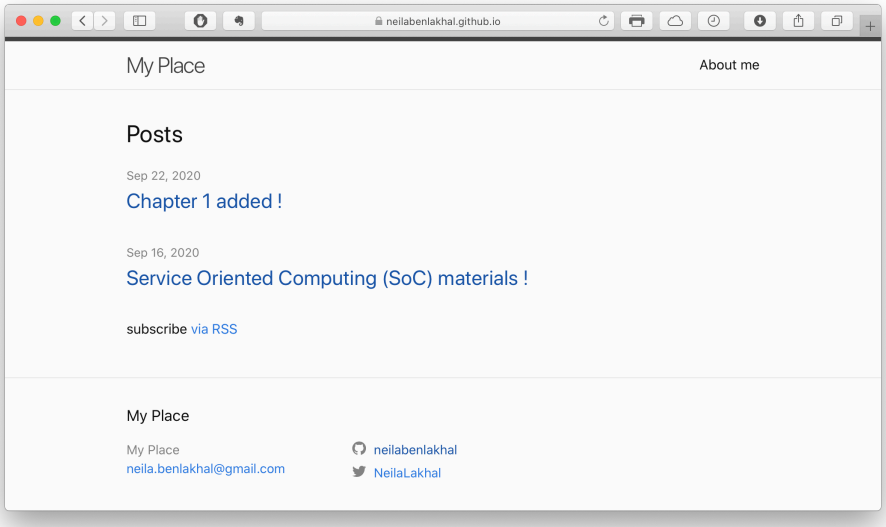
Service Oriented computing
Computing

By, **Dr. Neila Ben Lakhal**
Ph.D from Tokyo Institute of Technology
Assistant Professor @ ENICARTHAGE

Chapter 1. Demystifying Software Architecture

Today's lecture

Chap1



https://neilabenlakhal.github.io/

Motivational question

What is Software architecture ?

ISO/IEC/IEEE 42010 definition

***Software architecture**, which resulted from a joint effort between the ISO and IEEE.*

"Fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution."

Excerpt From: Joseph Ingeno. "Software Architect's Handbook."

Software Architecture: definitions

- “The industry as a whole has struggled to precisely define software architecture…Software architecture consists of the **structure of the system**, combined with **architecture characteristics** (“-ilities”) the system must support, **architecture decisions**, and finally **design principles**.”

  Excerpts From: Mark Richards. “Fundamentals of Software Architecture.”

- “Software architecture is an abstraction of a software system. The structures of a software system consist of its elements. Software architecture concerns itself with defining and detailing the structures, their elements, and the relationships of those elements with each other.”

  Excerpt From: Joseph Ingeno. “Software Architect's Handbook.”

- “People in the software world have long argued about a definition of architecture. For some it's something like the fundamental organization of a system, or the way the **highest level components are wired together**. Software architecture is the high-level structure of a software system.”

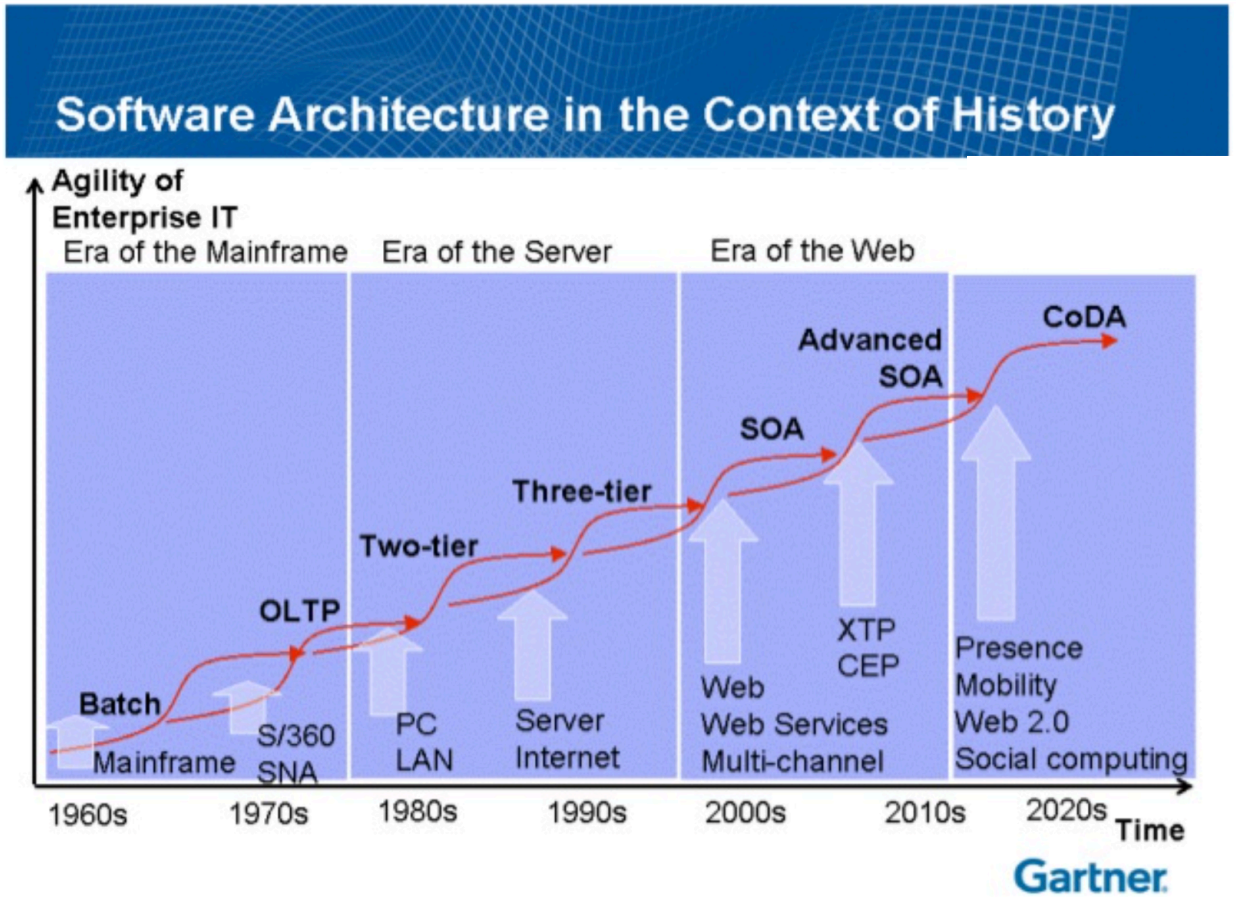
  Excerpt From: Martin fowler HP https://martinfowler.com/architecture/

Defining software architecture

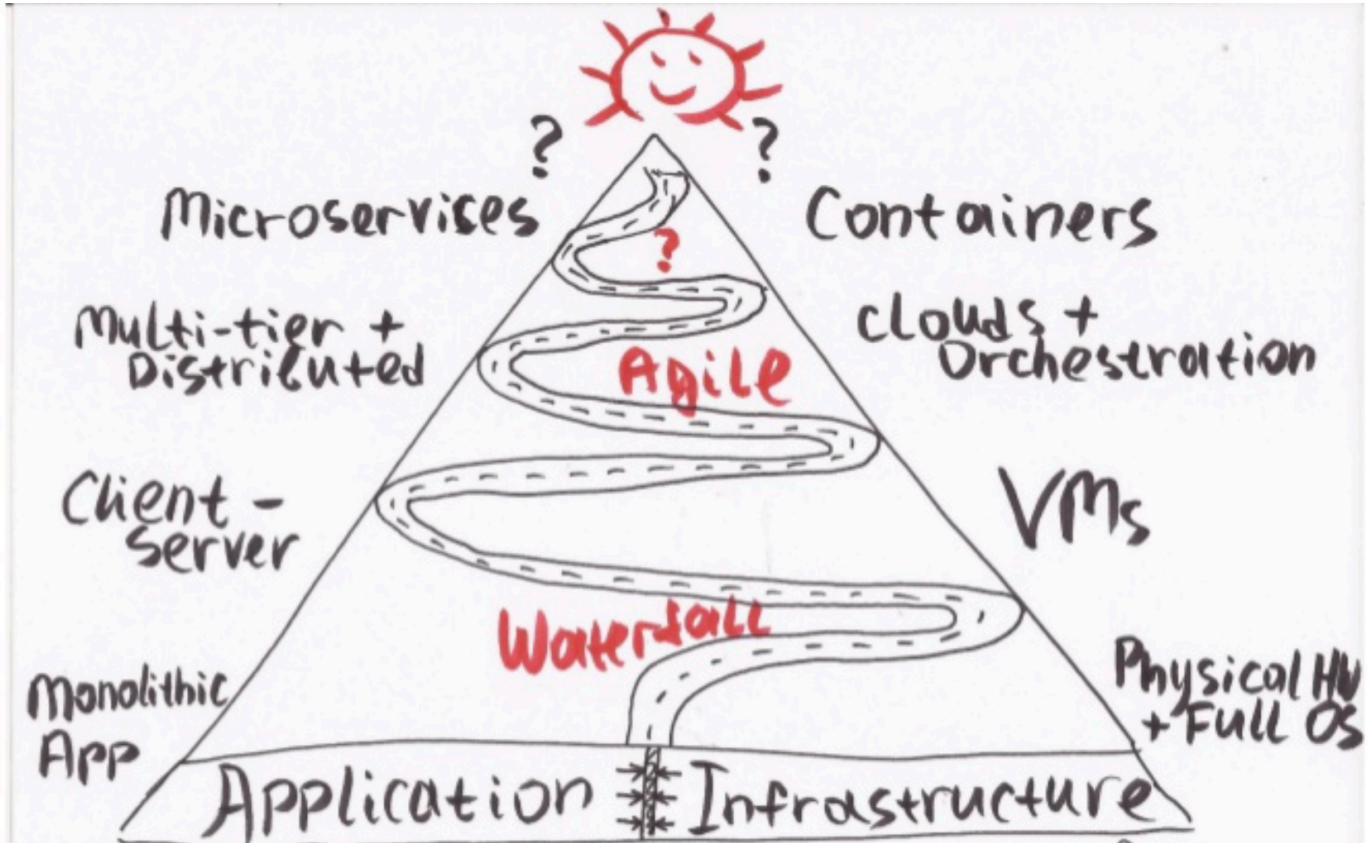
- To define a software architecture, 4 dimensions are to be considered:
  1. **Architectural styles** : refer to the structure used to implement a system.
  2. **Architecture characteristics** : define the success criteria (e.g., scalability, fault-tolerance..) of a system, generally orthogonal to the functionality of the system.
  3. **Architecture decisions** : form the constraints of the system and direct the development teams on what is and what isn't allowed (e.g., only the business and services layers within a layered architecture can access the DB, restricting the presentation layer from making direct DB calls).
  4. **Design principles:** are guidelines rather than a hard-and-fast rule (e.g., development teams should leverage asynchronous messaging between services within a microservices architecture to increase performance).

- In This course, we are only dealing with **architectural styles.**

Software architecture history



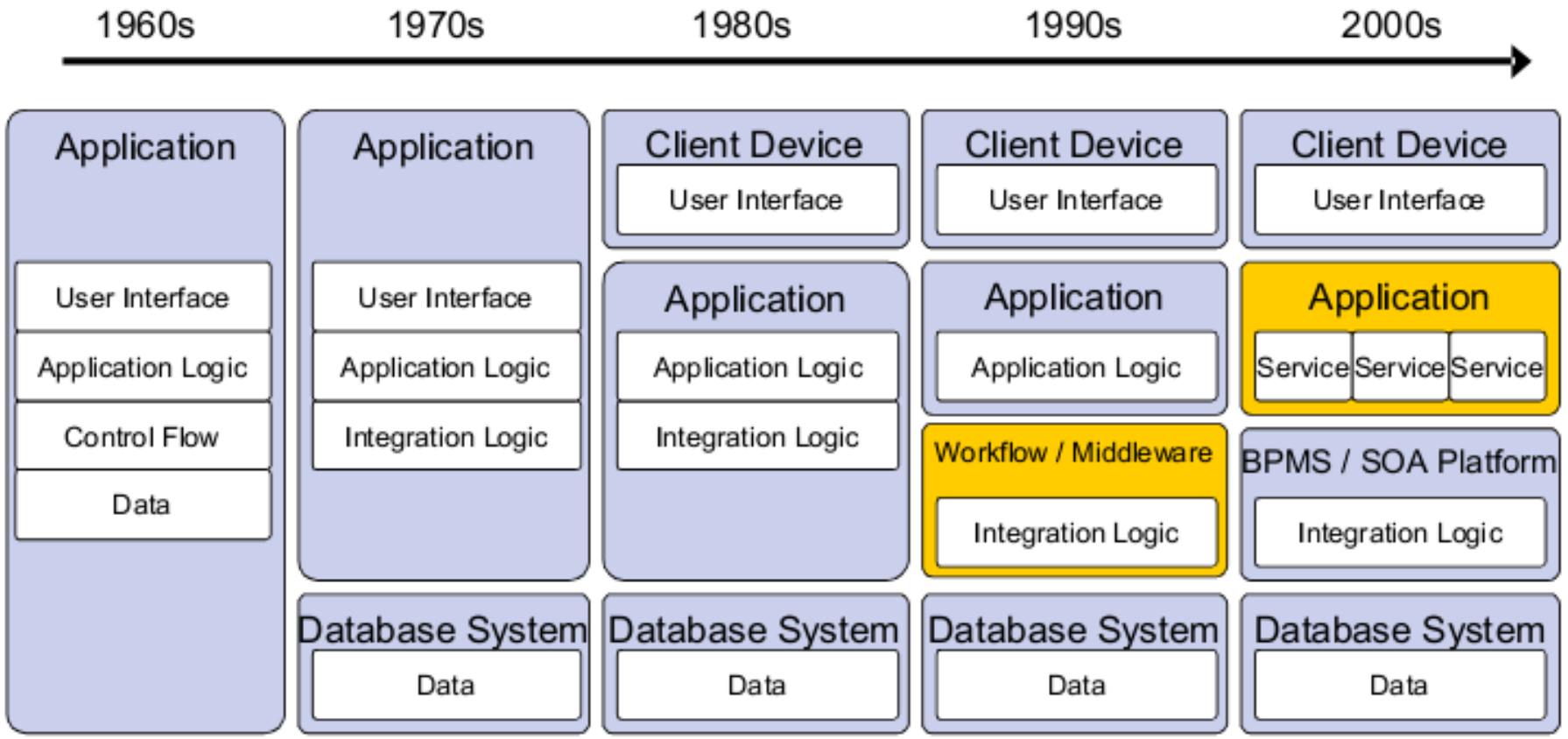
Software architecture evolution



Architectural styles

- **Architecture style** describes the relationship of components or building blocks of computerized IS.
- It provides a useful perspective on organizing code, deployments etc.
- Several architecture styles appeared throughout the history of software architecture.
- 2 classifications
  - **Fundamental styles ("old-fashioned")**
    1. **Unitary** architecture
    2. **client/server** architecture
  - **Modern styles**
    1. **Monolithic** architecture (***single*** deployment unit of all code)
    2. **Distributed** architecture (***multiple*** deployment units connected through remote access protocols)

Fundamental “Old-fashioned” Styles


1960s
1970s
1980s
1990s
2000s
Application
User Interface
Application Logic
Control Flow
Data
Application
User Interface
Application Logic
Integration Logic
Database System
Data
Client Device
User Interface
Application
Application Logic
Integration Logic
Database System
Data
Client Device
User Interface
Application
Application Logic
Workflow / Middleware
Integration Logic
Database System
Data
Client Device
User Interface
Application
Service
Service
Service
BPMS / SOA Platform
Integration Logic
Database System
Data

Unitary Architecture

- Software ran on **Mainframe** computers:
  - “dumb terminals” attached to the system. All centralized in the mainframe.
- Software ran on **PCs**:
  - When PC appeared, software development focused on single machines.

- **Is this architecture still used?**
  - Yes, among others, it is the defacto choice for **embedded systems**, **highly constrained environments**.

- Limitations
  - Software systems tend to grow in functionality over time, requiring separation of concerns to maintain operational architecture characteristics, such as performance and scale.

- **Client/server architecture** emerged, with networked PCs.

Client/server Architecture

- This architecture portioned the system into parts.

- Many flavours of this architecture exist, depending on the era and computing capabilities:

  - 2-tiers architecture
  - 3-tiers architecture
  - N-tiers architecture

2-tiers architecture

- 2-tiers architecture (Desktop + DB Server)
  - Rich desktop applications in user interfaces like Windows,
  - Data in a standalone DB server that could connect via standard network protocols.
  - Presentation layer resides on the desktop, while the more computationally intense action occurred on more robust database servers (stored procedure e.g., PL/SQL).
  - LAN-based, limited user accessed.
  - Very limited scalability.
  - Up to 10 users can access simultaneously.

3-tiers | N-tiers architecture

- 3-tiers architecture (Browser+Web server+ DB Server)
  - Emergence of application servers e.g., Java EE and .NET.
  - Presentation Layer+ Application Layer+ Data Layer (RDBMS).
- N-tiers architecture (Browser+Web server+ DB server)
  - Data separated into 2 layers:
    - Data persistence Layer+ Data Layer
      - **Persistence layer** : components for data access (e.g., ORM tool)
      - **Database layer** : the actual data store (e.g., RDBMS.)
  - Why ? To offer ability to switch out your data access or database technology for a different one.

Tiers versus layers

- When partitioning application logic, **layers** are a way to organize functionality and components.
- Layers are **logical** separations of a software application.
- Tiers are **physical** location of the functionality and components.
- Different **layers** do not necessarily have to be located on different physical machines.
- It is possible to have multiple **layers** on the same machine.
- For example :
  - A **3-layered architecture**, the logic may be separated into presentation, business, and data layers. All layers can be located on the same machine (WAMP)
  - A **3-tiered architecture** with presentation, business, and data tiers implies that those 3 tiers have been physically deployed to 3 separate machines and are each running on those separate machines.

Excerpt From: Joseph Ingeno. “Software Architect's Handbook.”

3-tiers | N-tiers architecture

◉The components of the system might be physically distributed to different locations in the globe.

◉Enterprise branches are connected in a TCP/IP protocol-based network : over long-range private networks (e.g., a private WAN or Virtual Private Network (VPN) tunnel over the internet)

◉How to communicate ?

- To enable communication between (heterogenous) components, **Middlewares** are used.
- What is a **Middleware** ?

Middleware ?

- "Middleware is the software 'glue' that helps programs and databases (which may be on different computers) work together. Its most basic function is to enable communication between different pieces of software." [Gartner Group]
- Middleware facilitates and manages the interaction between components across heterogeneous computing platforms.
- Different types of Middlewares:
  - Object-Oriented Middleware : e.g., CORBA
  - Message-Oriented Middleware : e.g., Java Message Service
  - Event-Based Middleware : e.g., Cambridge Event Architecture
- These Middlewares build on RPC spec. They focus on how to distribute objects, invoke their methods from a remote machine. They allow :
  - Cross-platform and Cross-language support for clients and server.

For more details see :

- Book "Essentials of Microservices Architecture Paradigms, Applications, and Techniques", Chap 1
- http://www.cl.cam.ac.uk/teaching/1011/CDSysII/12-middleware.pdf

Limitations of Middleware

- Non-standard interfaces. Traditional middleware systems and tools suffer from lack of standardization: they are not compatible. Thus, it is very expensive to build integrated distributed systems across different middleware platforms.

- Lack of trust. How to trust the clients? Building integrated systems spawning across different trust domains can be difficult.

- Middleware systems are (logically) centralized. Thus, there is no place for them in B2B Integration scenarios as they should be distributed across all partners. Point to Point integration does not scale.

- Slow interactions across organizational boundaries and should be handled asynchronously.

Modern classification

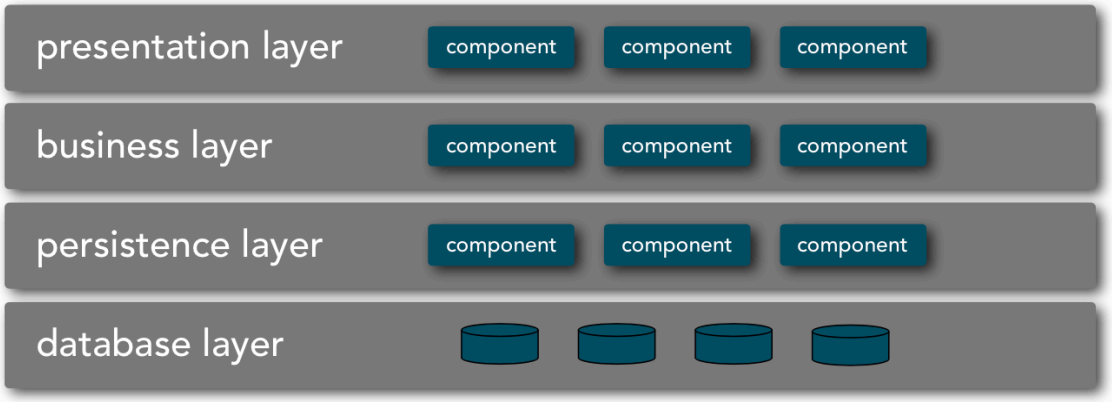
Modern styles

- While no classification scheme is perfect, software architecture have been using this classification scheme of the various architecture styles:
- **Monolithic**
  1. **Layered** architecture
  2. **Pipeline** architecture
  3. **Microkernel** architecture
- **Distributed**
  1. **Event-driven** architecture
  2. **Service-based** architecture
     - Service-oriented architecture (SOA)
     - Microservices architecture (MSA)
  3. ...

Monolithic architectures

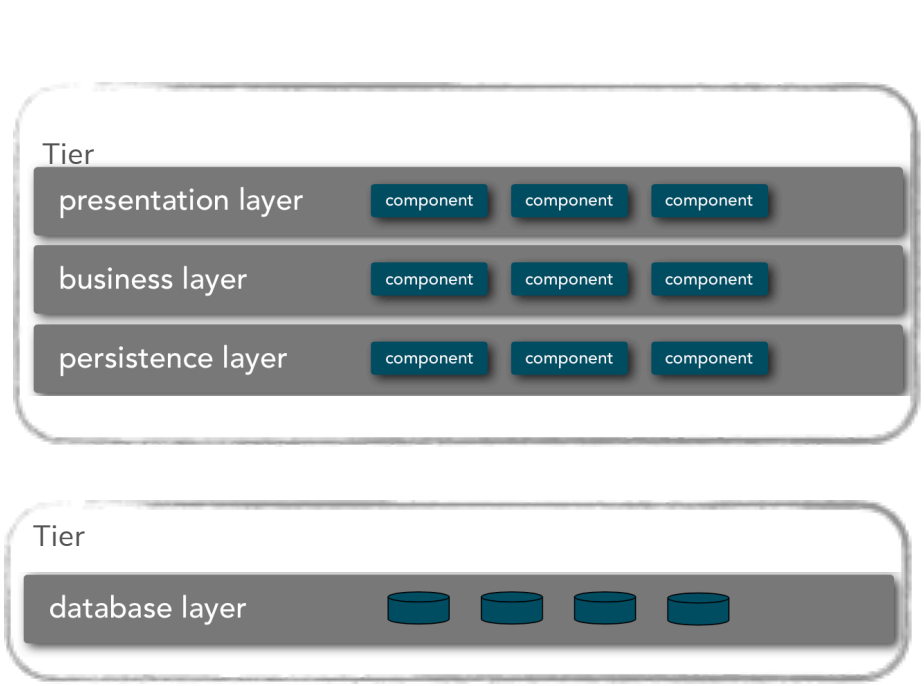
Layered architecture 1/2

- The software application is divided into logical **horizontal layers**, with each **layer** located on top of a lower **layer.**
- Most layered architectures consist of 4 standard **layers**:
  - presentation+ business+persistence + database
- Where is it used ?
  - A good choice for small, simple applications or websites.



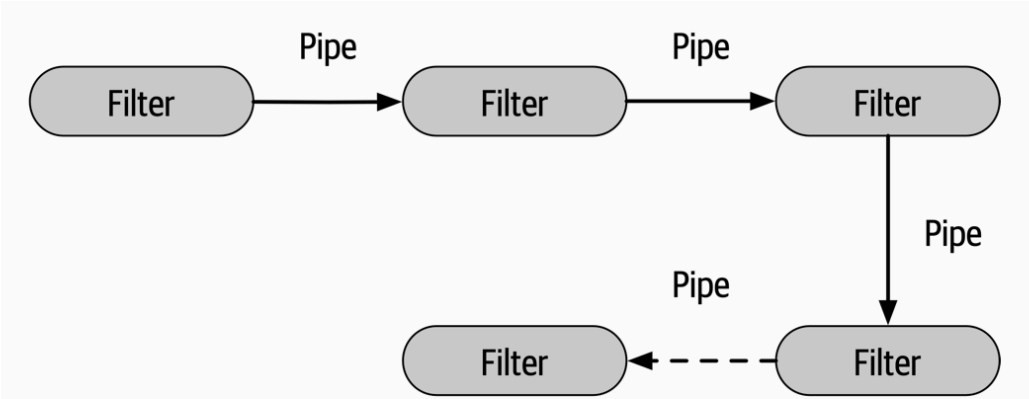
Layered Architecture 2/2

- Various topology variants from a physical layering (deployment) perspective are possible :



Pipeline architecture

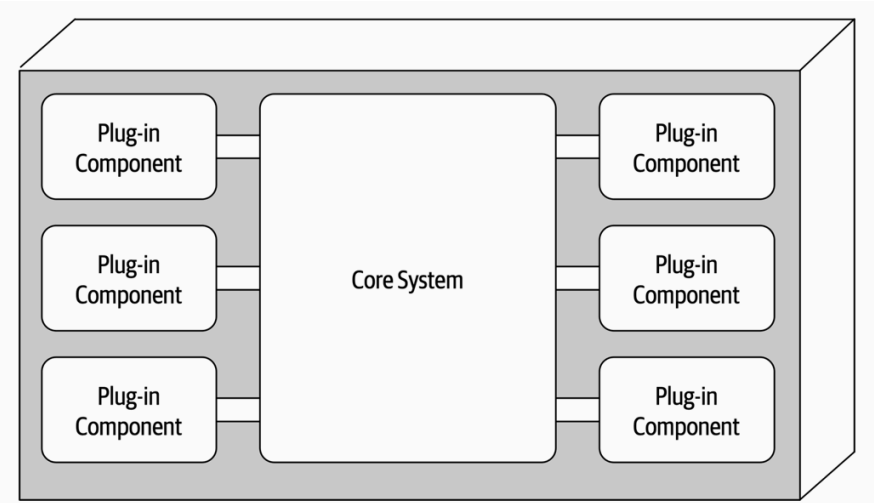
- The topology of the pipeline architecture consists of **pipes** and **filters**: pipes form the communication channel between filters.



- Where is it used ?
- The pipeline architecture pattern appears in a variety of applications, for example :
  - Electronic Data Interchange (EDI) tools : building transformations from one document type to another.
  - ETL tools (extract, transform, and load)
  - The flow and modification of data from one DB or data source to another.
  - Orchestrators and mediators such as Apache Camel utilize the pipeline architecture to pass information from one step in a business process to another.

Microkernel architecture

◉Also referred to as the plug-in architecture consisting of 2 architecture components: a **core system** and **plug-in components**. Application logic is divided between independent plug-in components. The core system the minimal functionality required to run the system.



◉Where is it used ?

- Most of the tools used for developing and releasing software e.g.,: Eclipse IDE, Jenkins...
- Internet web browsers : viewers and other plug-ins add additional capabilities that are not otherwise found in the basic browser representing the core system.
- Complex applications that leverage complex business rules such as insurance system, taxation system,...

Distributed Architectures

Event-Driven Architecture Style 1/2

- Event-Driven Architecture (EDA) is an **asynchronous** software architecture style that integrates applications and components through the production and handling of **events**. By tracking events, we don't miss anything of significance related to the business domain.

- Event ?
  - The occurrence of something deemed significant in a software application, such as a state change.

- Example of an event :
  - The placement of a purchase order.

Event-Driven Architecture Style 2/2

- It is made up of decoupled **event processing components** that asynchronously receive and process **events**.
- Event processing components are :
  1. **Event producers** : they create event messages.
  2. **Event messages** : contain data about an event.
  3. **Event consumers** : they process event messages.
  4. **Event Channels** : are message queues used by event messages to reach an event processor.
- There are 2 topologies within EDA architecture:
  - **Mediator topology** : used when control over the workflow of an event process is required.
    - **Examples of mediator : Apache ODE and the Oracle BPEL Process Manager.**
  - **Broker topology** : used when even processing flow is rather simple and no need for central coordinator.
    - **Examples of Broker : RabbitMQ**

Service-based architecture

- There are 2 service-based architectures, namely, **SOA (service-oriented architecture)** and **MSA (microservice architecture) :**
- Both break an application into small, functionally independent software modules with well-defined interface.
- **SOA** :
  - Building blocs in SOA are called **services**.
  - SOA enables interaction among heterogeneous applications and brings interoperability.
- **MSA**
  - Building blocs in MSA are called **microservices**.
  - MSA is for an individual application that facilitates frequent deployment and continuous delivery.

SOA in more details

- SOA is an architectural pattern for developing software systems.
- A key aspect of SOA is that it decomposes application logic into smaller units called **services** that can be reused and distributed.
- Each service encapsulates a certain piece of logic. This logic may be responsible for a very specific task, a business process, or a subprocess.
- Services can vary in size and one service can be composed of multiple other services to accomplish its task.
- A service is a part of a software application that performs a specific task, providing functionality to other parts of the same software application or to other software applications.
- SOA emphasizes that applications should be developed in such a way so that they can interact with one another **using standard communication protocols**.

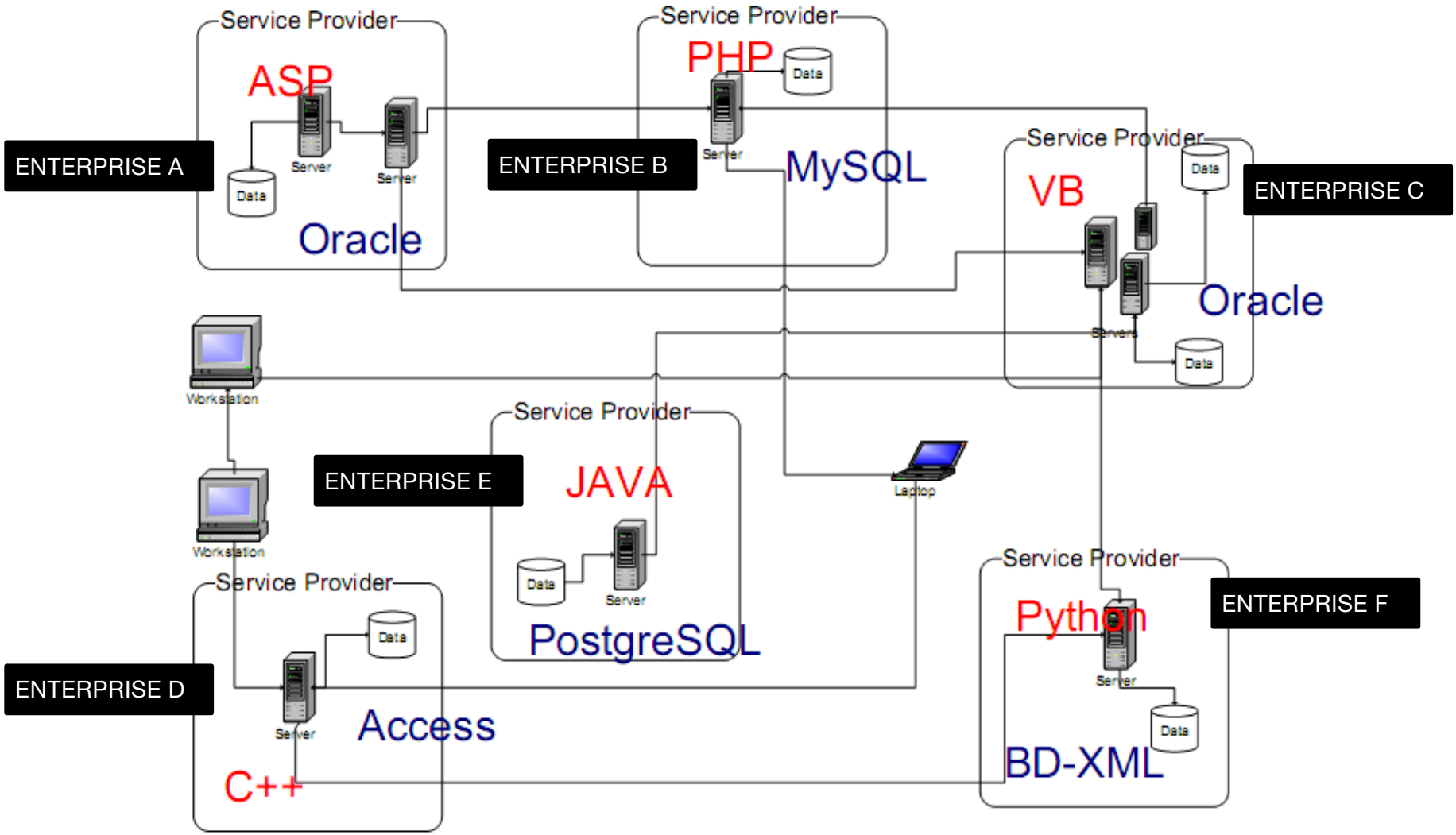
MSA in more details

- MSA is a variation of SOA, Some people even refer to the MSA as SOA done right.

- MSA pattern builds software applications using small, autonomous, independently versioned, self-contained microservices. These services use well-defined interfaces and communicate with each other over standard, lightweight protocols.

- Each microservice focuses on doing one thing well and they can work together with other microservices in order to accomplish tasks that are more complex.

- MSA and SOA architectures emerged, as the result of **the shortcomings and drawbacks of the monolithic architecture**.


Service Provider
ASP
Server
Server
Data
Oracle
ENTERPRISE A
Service Provider
PHP
Data
Server
MySQL
ENTERPRISE B
Service Provider
VB
Data
Servers
Oracle
Data
ENTERPRISE C
Workstation
Workstation
Service Provider
JAVA
Data
Server
PostgreSQL
ENTERPRISE E
Laptop
Service Provider
Data
Server
Access
C++
ENTERPRISE D
Service Provider
Python
Server
Data
BD-XML
ENTERPRISE F