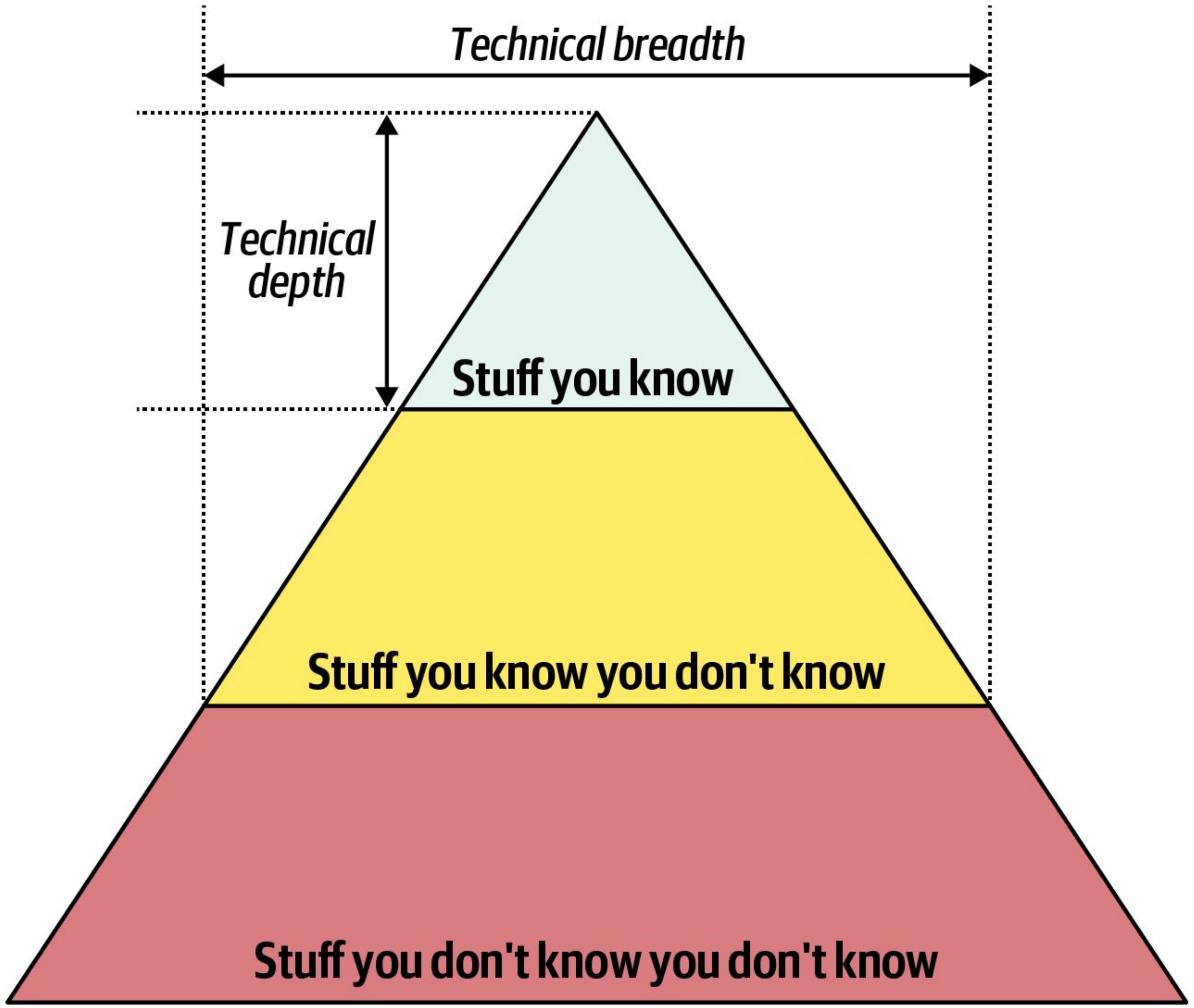
Ca' Foscari University of Venice


Technical breadth
Technical depth
Stuff you know
Stuff you know you don't know
Stuff you don't know you don't know

Deployability	the ease, frequency, and risk of deployment	**Performance**	amount of used resources, including time behavior, resource utilization, and capacity
Elasticity	the ability to handle bursts of users	**Reliability**	degree to which a system functions under specified conditions for a specified period of time
Evolutionary	the ability accept incremental and guided changes	**Scalability**	ability for the system to perform and operate as the number of users or requests increases
Fault tolerance	does the software operate as intended despite hardware or software faults	**Simplicity**	the ease of understanding the overall system
Modularity	degree to which the software is composed of discrete components	**Testability**	the ease of and completeness of testing
Overall cost	“additional” effort required to implement the system		

Ca' Foscari University of Venice

SMART objectives

- When we set up an object we should have that it is:
  - Specific: target a specific area for improvement.
  - Measurable: quantify or at least suggest an indicator of progress.
  - Assignable: specify who will do it.
  - Realistic: state what results can realistically be achieved, given available resources.
  - Time-related: specify when the result(s) can be achieved.
- https://en.wikipedia.org/wiki/SMART_criteria
- We are still missing measurable
  - And we are going to underspecify it, but we must discuss…

Ca' Foscari University of Venice

Task 2

Please submit a pdf file where you define **what architecture characteristics** among the ones we identified are **of interest** to your IT system, **relating** each architecture characteristics to the points of the kata that led you to identify that characteristic. For each architecture characteristic you picked up, please **"quantify"** exactly at what level you need it for your IT system.

PS: the pdf can contain also an **updated version of the kata** provided in the previous tasks (e.g., to number the various requirements and refer to them in the new task)

Ca' Foscari University of Venice

Schedule

- The schedule has been finalized:
  - in person lectures always on Thursday at 2pm in Aula B except
    - Nov. 14th in Lab. 3
    - Tuesday Nov. 19th at 8:45 in Aula C
  - Nov. 28th invited lecture by Unox
    - https://www.unox.com/us_us/
  - Dec. 5th invited lecture by Gianluca Caiazza
    - Zamperla project
    - https://www.zamperla.com/

Ca' Foscari University of Venice

Monolithic architectures [Concepts]

Software architectures
Pietro Ferrara
pietro.ferrara@unive.it

Ca' Foscari University of Venice

Outline

1. Ball of Mud architecture
2. Layered (n-tier) architecture
3. Pipeline architecture
4. Microkernel architecture

Ca' Foscari University of Venice

1st year of (any) Bsc in Computer Science

Write a program that plays checkers. In particular, implement the following methods:

```
Player(const Player&);
piece operator()(int r, int c, int history_offset = 0) const;
void load_board(const string& filename);
void move();
void store_board(const string& filename) const;
```

A board is represented by a text file like this one:

```
o o o X 
 o o   o
o o o   
   o   o
    o x 
 x x   x
x   x x 
 x x x x
```

https://docs.google.com/document/d/1I4HaqplHf-sVeD3xYB-xBbyf3Jt0V4lDNy9BebDI_C4

Ca' Foscari University of Venice

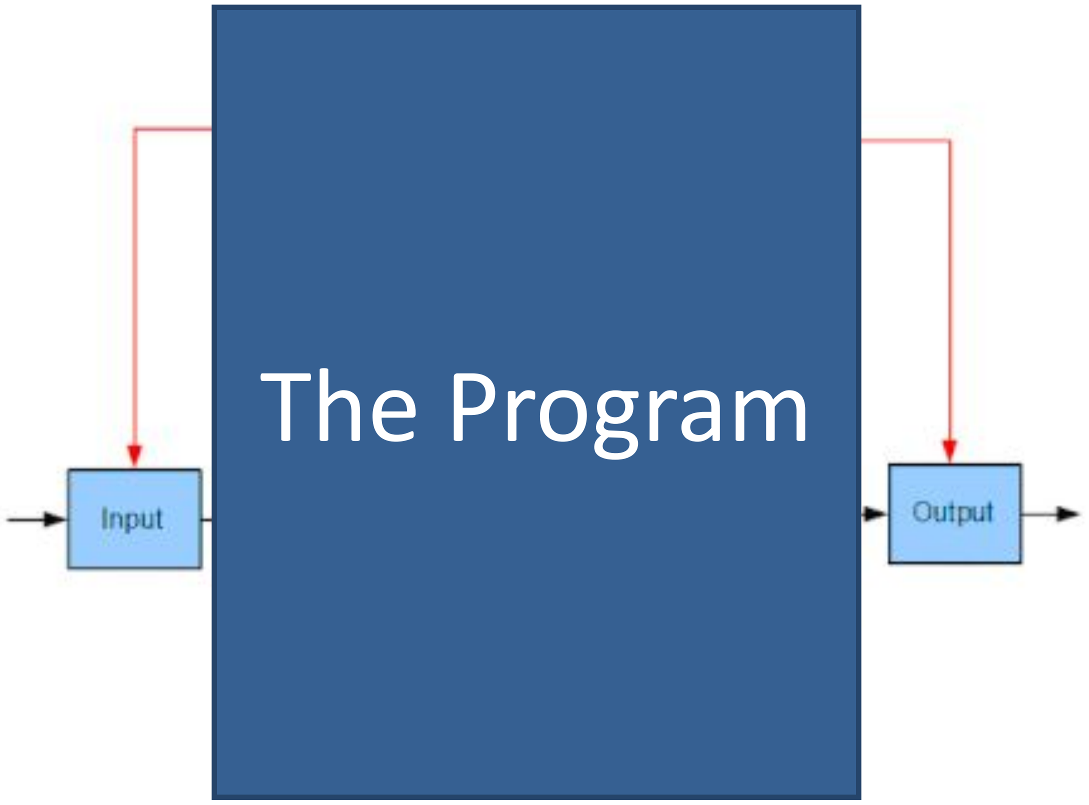
John von Neumann (1903-1954)

- Mathematician, physicist, computer scientist, and polymath
- His work spanned from pure mathematics to the Manhattan project
  - Produced the first nuclear weapons
- He introduced the first formalization of computer architecture
  - The so-called von “Neumann architecture”

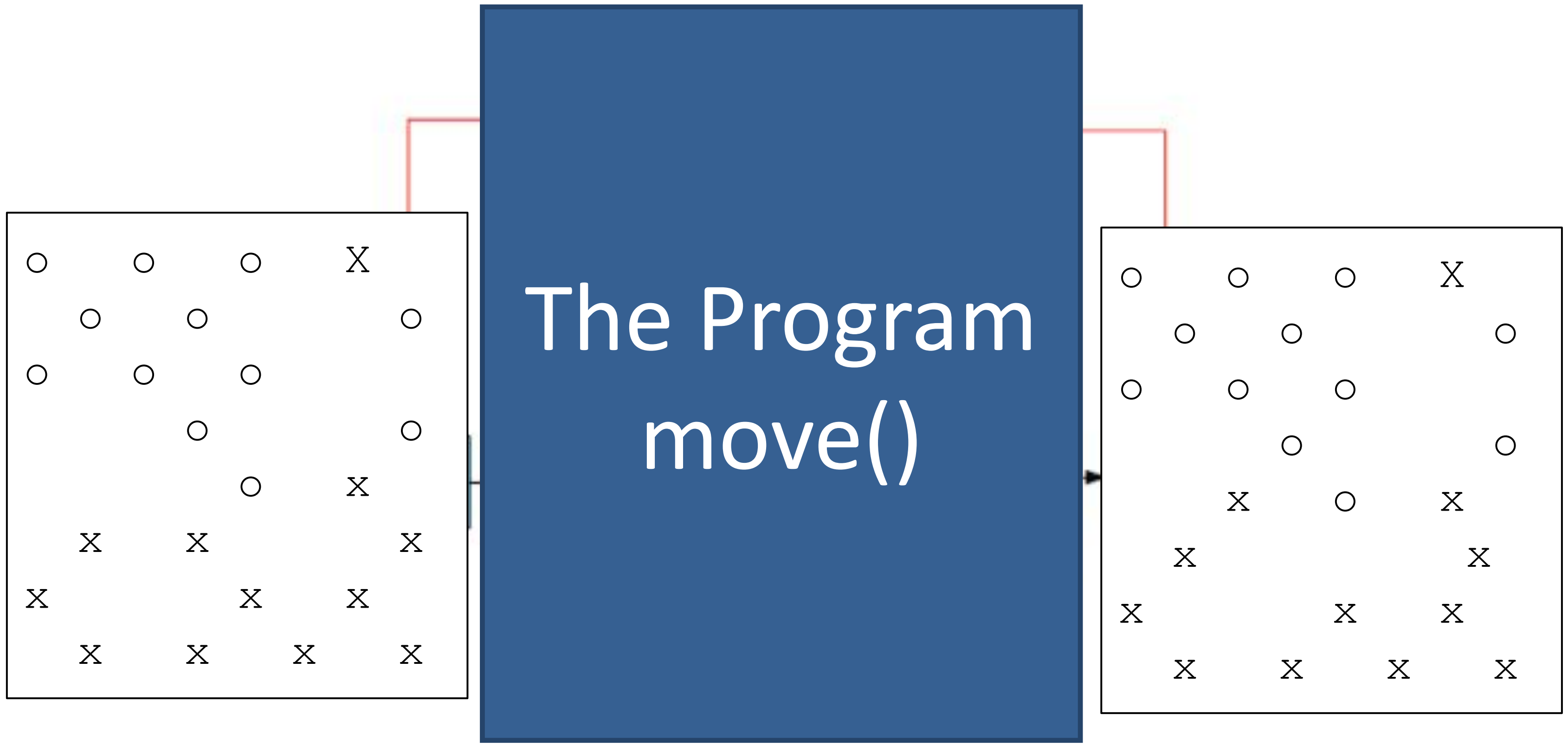
Ca' Foscari University of Venice

von Neumann architecture

- Central Processing Unit (CPU)
- Main components
  - Control Unit
    - Manage the computations of the CPU
  - An arithmetic & logic unit
    - Execute the logic operations
  - Memory
    - Store operations and data under execution
  - Input
  - Output $\longrightarrow$ Peripherals



1st year of (any) Bsc in Computer Science



Ca' Foscari University of Venice

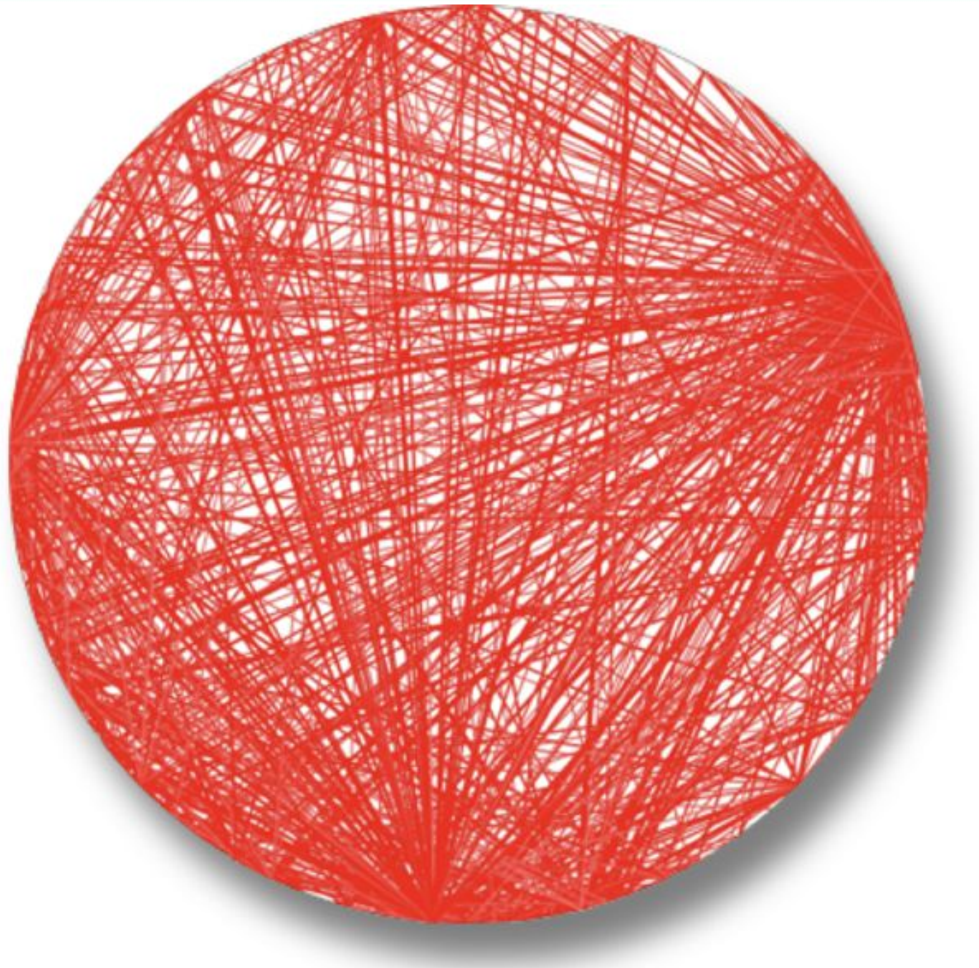
Executing a program

- We need a way to interact with our program
  - Define an input and output format
  - Run the program in some ways
    - Command Line Interface
    - Application Programming Interface
- A computer then automatically executes a program
  - That is, a sequence of instructions
  - Supported by our CPU
    - Arithmetic operations
    - Jumps
- At the very basic, the program is just a black box
  - Monolithic!

Ca' Foscari University of Venice

Well… what's the problem here?

- All program components depend on all the other components
  - In the long term
- Modularity is completely lost
  - Hard to maintain such codebase
  - You cannot locally reason on one component
- Run only on a single machine
  - Neither scalable nor elastic
- Run only end-to-end
  - Hard to deeply test is
- Fast and easy to develop
  - Just implement the logic of the program
- Intuitively, Ball of Mud == no architecture


The Ball of Mud architecture

Ca' Foscari University of Venice

Our architecture characteristics (from lecture 2)

Deployability	the ease, frequency, and risk of deployment	**Performance**	amount of used resources, including time behavior, resource utilization, and capacity
Elasticity	the ability to handle bursts of users	**Reliability**	degree to which a system functions under specified conditions for a specified period of time
Evolutionary	the ability accept incremental and guided changes	**Scalability**	ability for the system to perform and operate as the number of users or requests increases
Fault tolerance	does the software operate as intended despite hardware or software faults	**Simplicity**	the ease of understanding the overall system
Modularity	degree to which the software is composed of discrete components	**Testability**	the ease of and completeness of testing
Overall cost	"additional" effort required to implement the system		

Your wooclap poll will be displayed here

1

Install the **Chrome** or **Firefox extension**

2

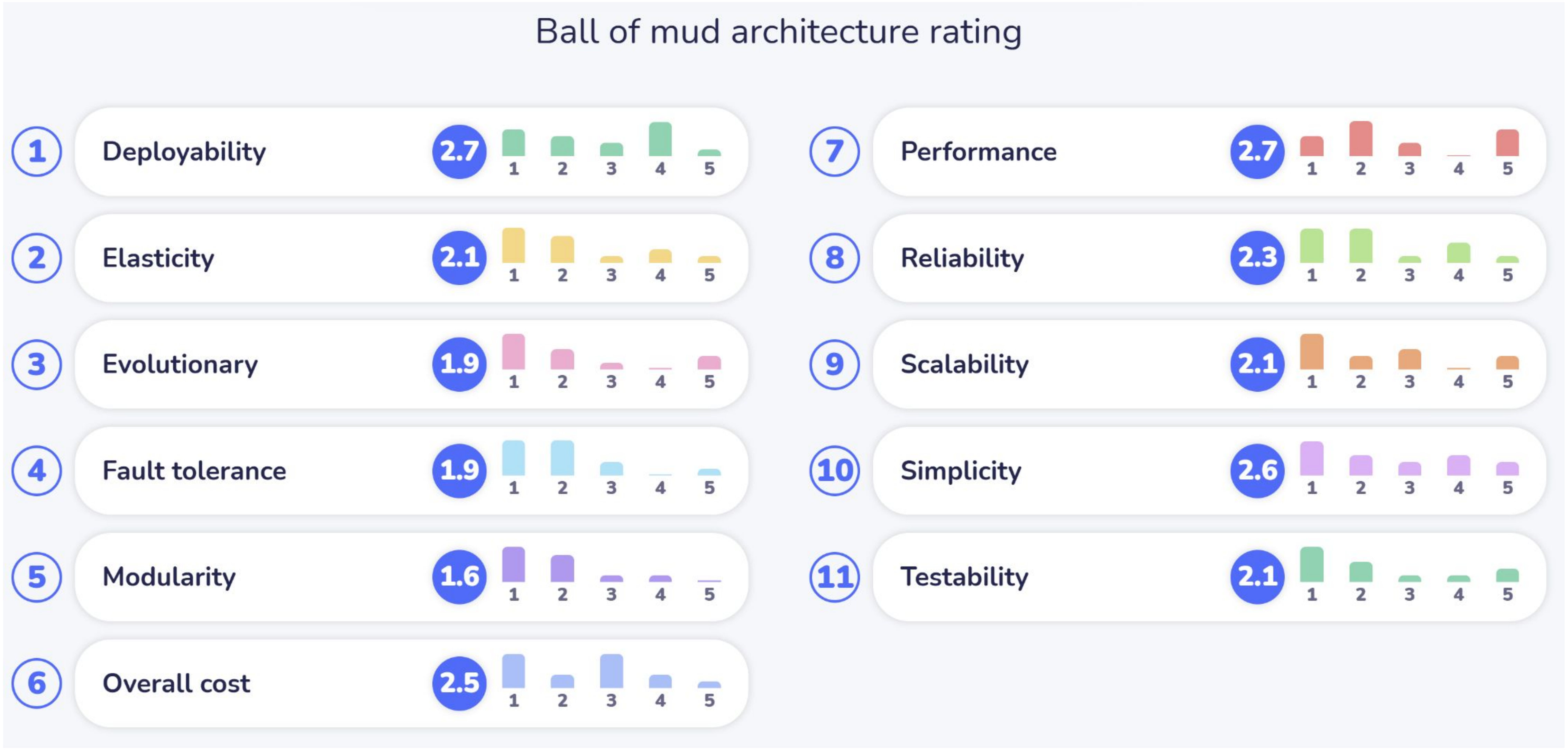
Make sure you are in **presentation mode**

wooclap

https://app.wooclap.com/events/SADM/0 question 12

Ca' Foscari University of Venice

Ball of mud architecture rating



#	Characteristic	Rating
1	Deployability	2.7
2	Elasticity	2.1
3	Evolutionary	1.9
4	Fault tolerance	1.9
5	Modularity	1.6
6	Overall cost	2.5
7	Performance	2.7
8	Reliability	2.3
9	Scalability	2.1
10	Simplicity	2.6
11	Testability	2.1

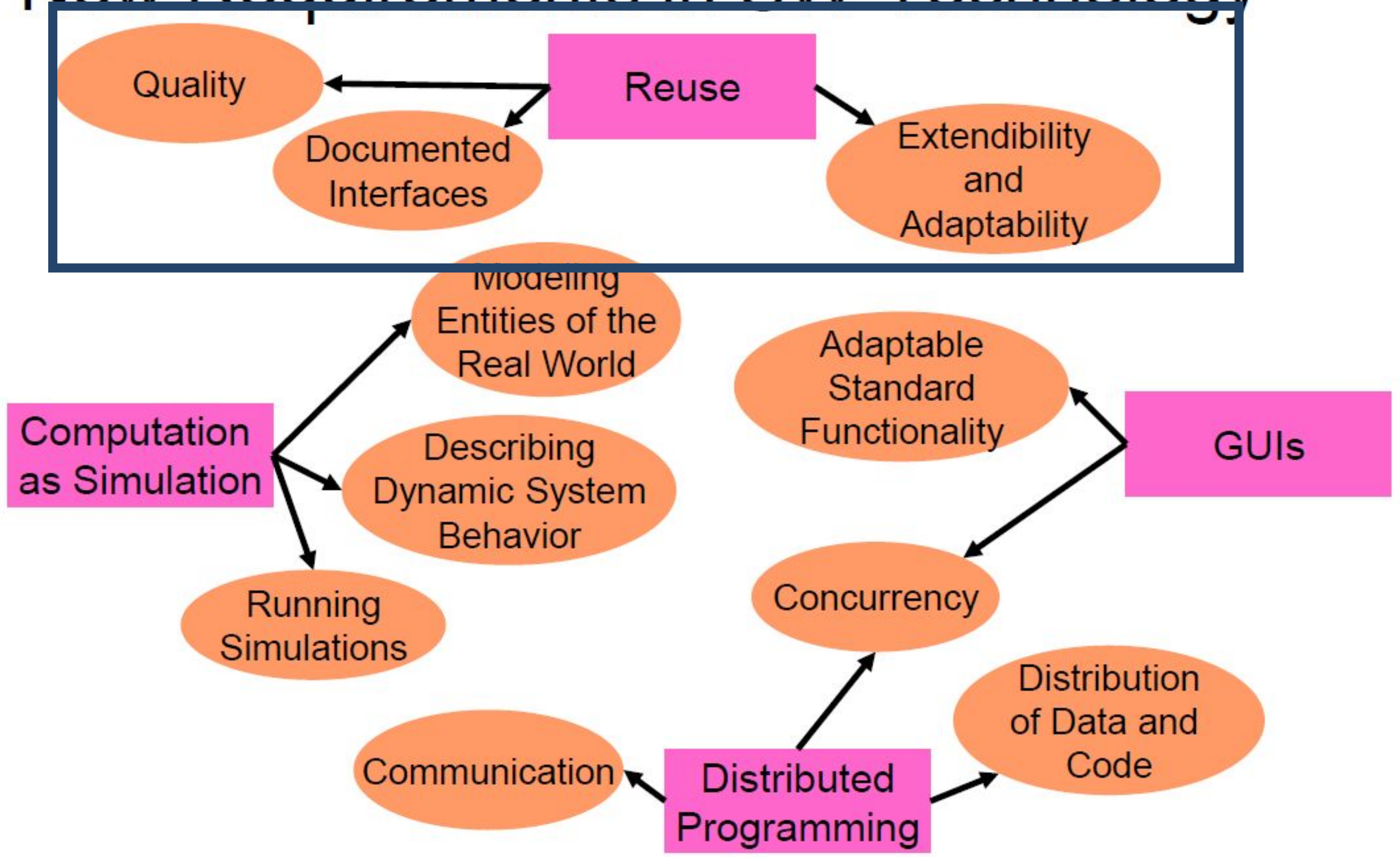
Ball of Mud architecture rating "My solution"

Deployability	★★	**Performance**	★★★★★
Elasticity	★	**Reliability**	★★
Evolutionary	★	**Scalability**	★
Fault tolerance	★	**Simplicity**	★
Modularity	★	**Testability**	★★
Overall cost	★★★★★		

From Ball of Mud to modules

- Programming languages evolved to enforce modularity
- Allow developers to hide implementation details
  - Inside the monolith!
- Object oriented programming appeared for this reason
- Encapsulation: data bundled with the methods operating on it
- Information hiding: limit the access to object states
- A client must have
  - Access all information to use the module
  - No access to any other data of the object
- Minimize the ball of mud

New Requirements in SW-Technology


Quality
Reuse
Documented Interfaces
Extendibility and Adaptability
Modeling Entities of the Real World
Adaptable Standard Functionality
GUIs
Computation as Simulation
Describing Dynamic System Behavior
Running Simulations
Concurrency
Distribution of Data and Code
Communication
Distributed Programming

https://www.pm.inf.ethz.ch/education/courses/COOP.html

Goal of OO programming languages (from PO1, 2nd year)

- Improve code reuse
  - Allow a clean code structure through encapsulation
    - Hide information of software units that should not be visible from outside
  - Allow to extend and specialize existing code through inheritance
  - Allow to develop reusable algorithms through classification, polymorphism and dynamic method binding
- Main outcome: a programming language that allows to modularly reason on software capsules
  - Advantages: a lot of well documented and easy to use libraries
  - Weaknesses: efficiency and conciseness

From encapsulation to modules, artifacts, libraries, etc… (2nd year)

- Nowadays, monoliths have anyway some internal structure
- Programming languages evolved to support complex scenarios
- Let’s consider for instance Java
  - Information hiding since the very first version (1995)
  - Packages aggregating several classes together as well (1995)
  - Jar files (aka, artifacts) since Java 1.2 (1997)
  - Repositories of jar files (e.g., Maven) (~2005)
  - Modules since Java 9 (2017)
- All this just on a single monolithic system
  - Nothing about distributed systems so far (and still for a bit)

Ca' Foscari University of Venice

Monolithic vs distributed architectures

- Monolithic architecture: single deployment unit of all code
  - Note that it can be split into several modules/artifacts/…
  - But all together this should be a unique unit of code
  - It runs on a single machine
- Distributed architecture: multiple deployment units connected through remote access protocols
  - Remote implies that the units run on (logically) distinct machines
  - Different units need to communicate with each other in some ways
- Distributed is strictly more complex than monolithic
  - But there are other advantages…
- For now we focus only on monolithic architectures

Ca' Foscari University of Venice

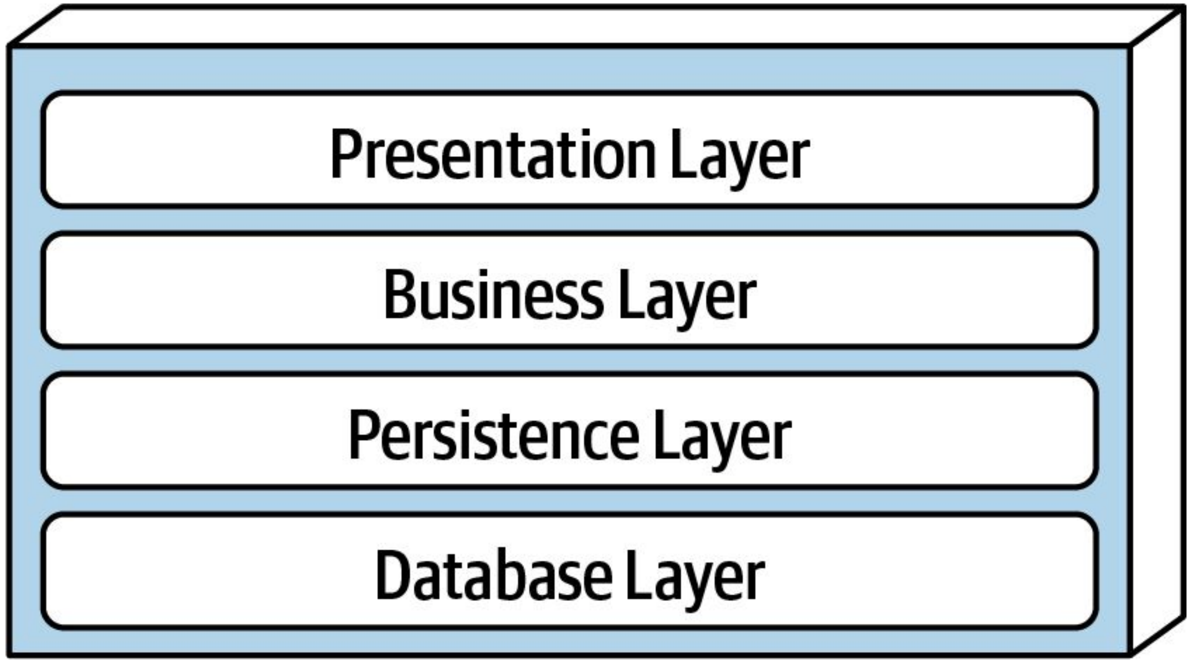
Do ball of muds still exist?

- So far we described a naive "ball of mud" architecture
  - Good to describe projects of "introduction to coding" courses
    - Written by one (or few) student, executed few times and then abandoned
  - Maybe realistic at the very beginning of computer science (50s/60s?)
- However, it is still worth to discuss them!
  - Definitely the simplest and cheapest architecture
- And yep… it is still in use!
  - Ever heard of things like data science scripting in Python/R?
- What would you do if you have few minutes to implement a very simple task to solve quickly and dirty a problem you have?
- Intuitively, no architecture at all => no architectural overhead

Layered (n-tier) architecture

- De facto standard
- Application split into several horizontal layers
  - Each one technically different
  - Front end, back end, DB, …
- Reflects the organization structure
  - UI developers
  - Backend developers
  - Rules developers
  - Database experts
- The number of layers can vary
  - Sometimes 3, sometimes 5

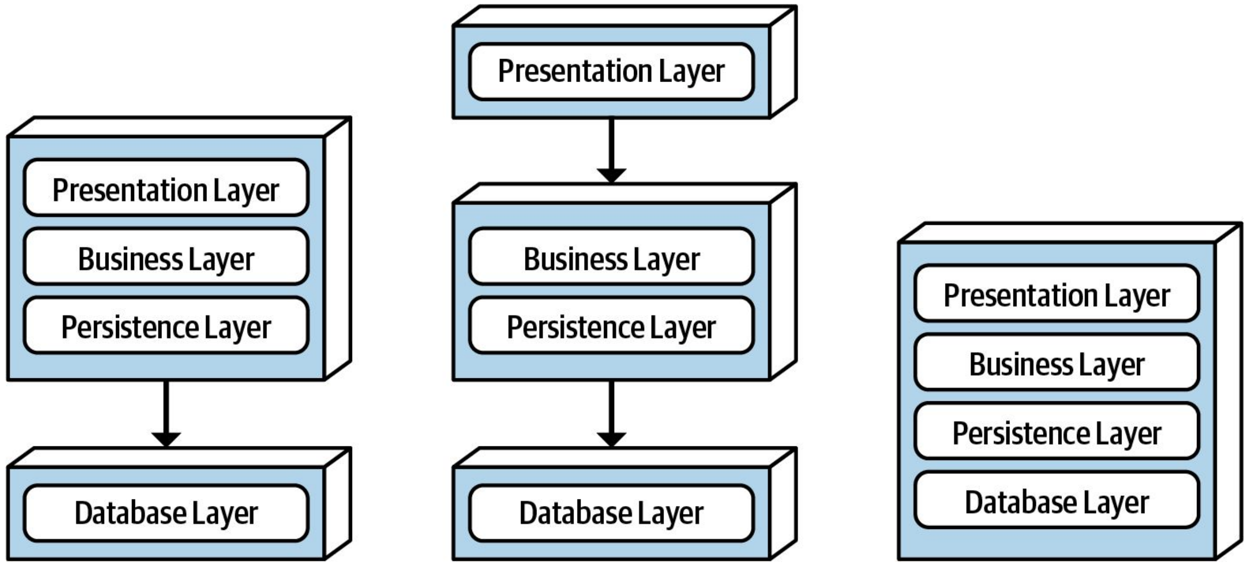
Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure. (Melvin E. Conway)



Ca' Foscari University of Venice

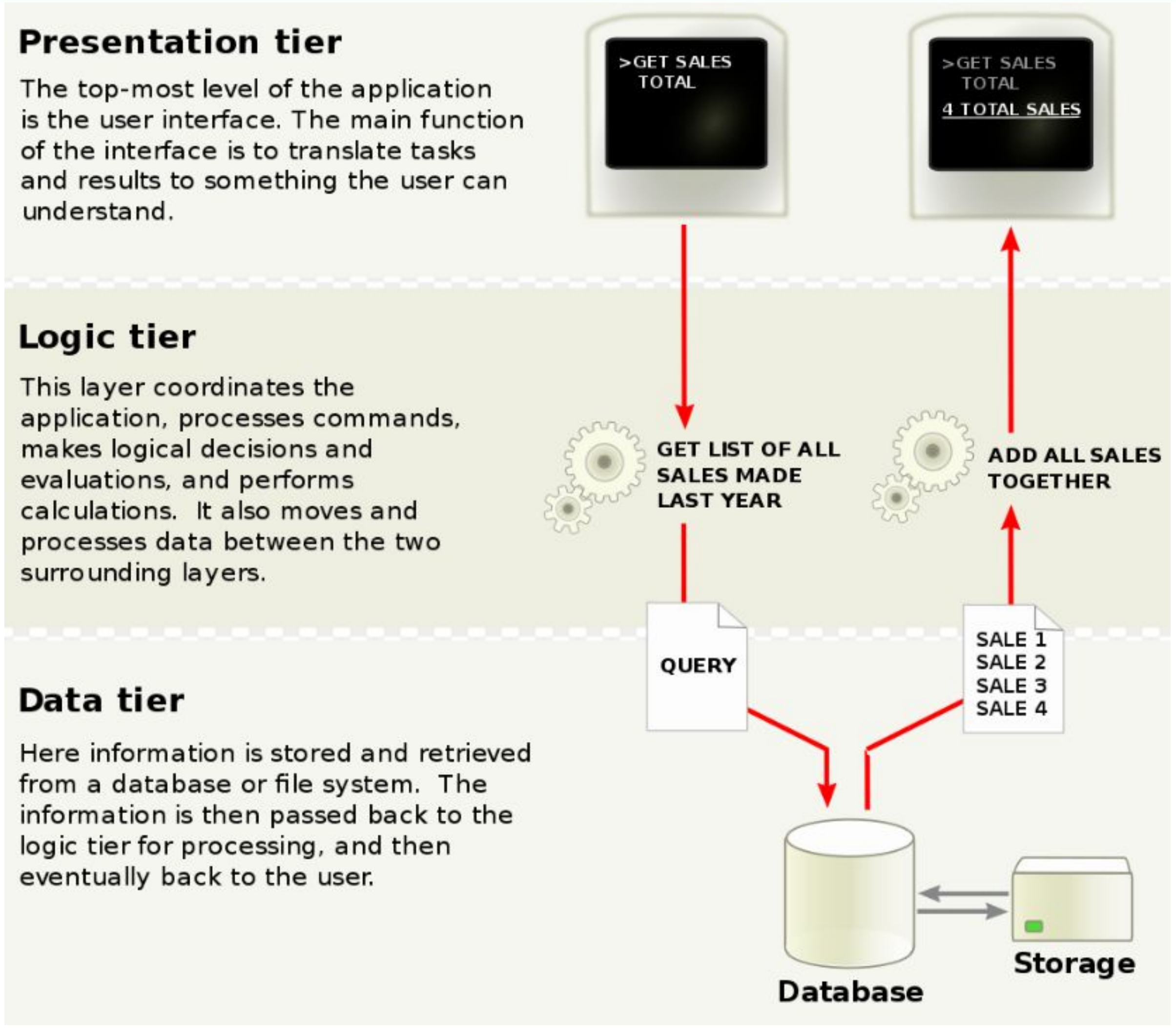
Deployment variants

- We can deploy in different physical units different layers
  - In the same machine (monolith)
- Usually the database is distinct
- Sometimes also the UI
- All together still interesting
  - Deliver a unique artifact
    - On-premises
  - Small applications with embedded database



Multitier, 3-tier, … architectures

- n-tier: n not defined!
- Several variations around
- Standard for Web applications: 3
  - Front-end (presentation)
  - Back-end (business+persistence)
  - Database
  - (ignore this is distributed for now!)
- There is no magic number
- As always, it depends on practice
- Do you know OSI vs TCP/IP models?



Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

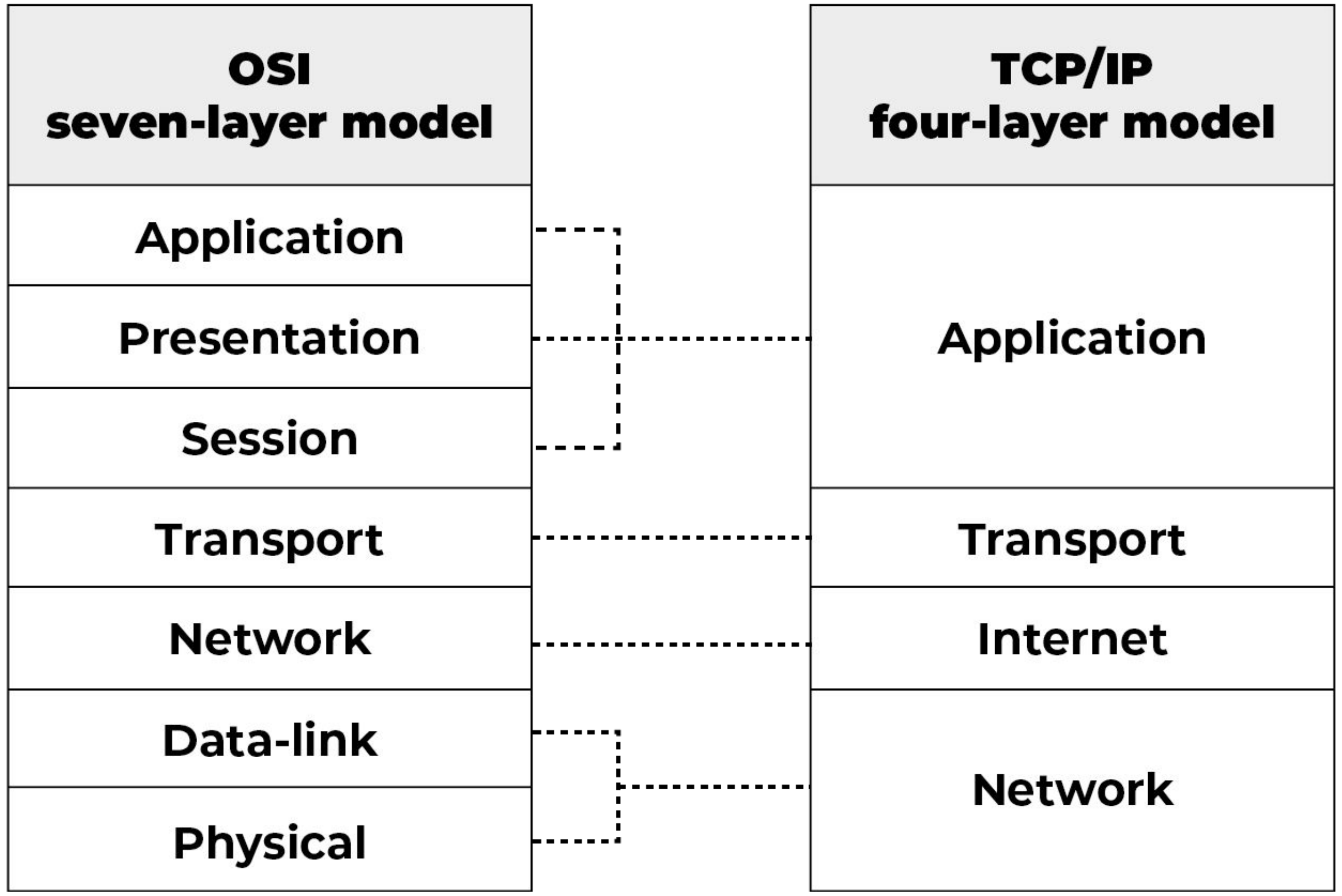
Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

https://en.wikipedia.org/wiki/Multitier_architecture

Ca' Foscari University of Venice

OSI vs TCP/IP models (from here)

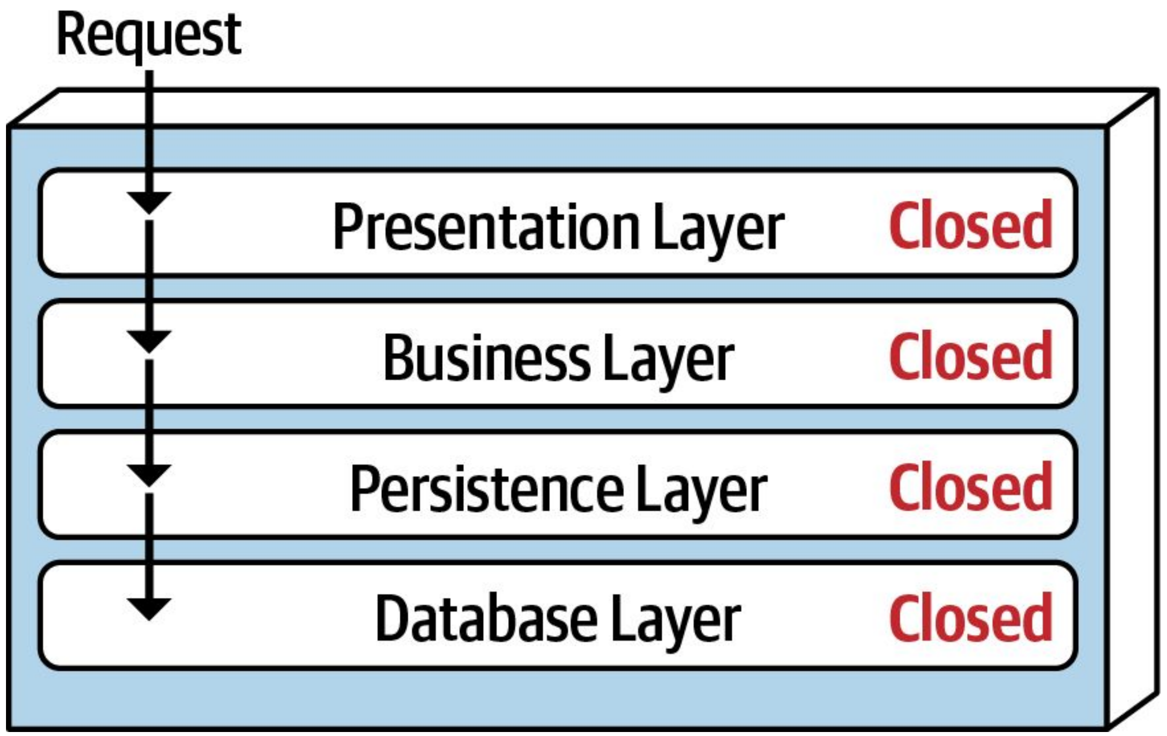


https://en.wikipedia.org/wiki/OSI_model https://en.wikipedia.org/wiki/Internet_protocol_suite

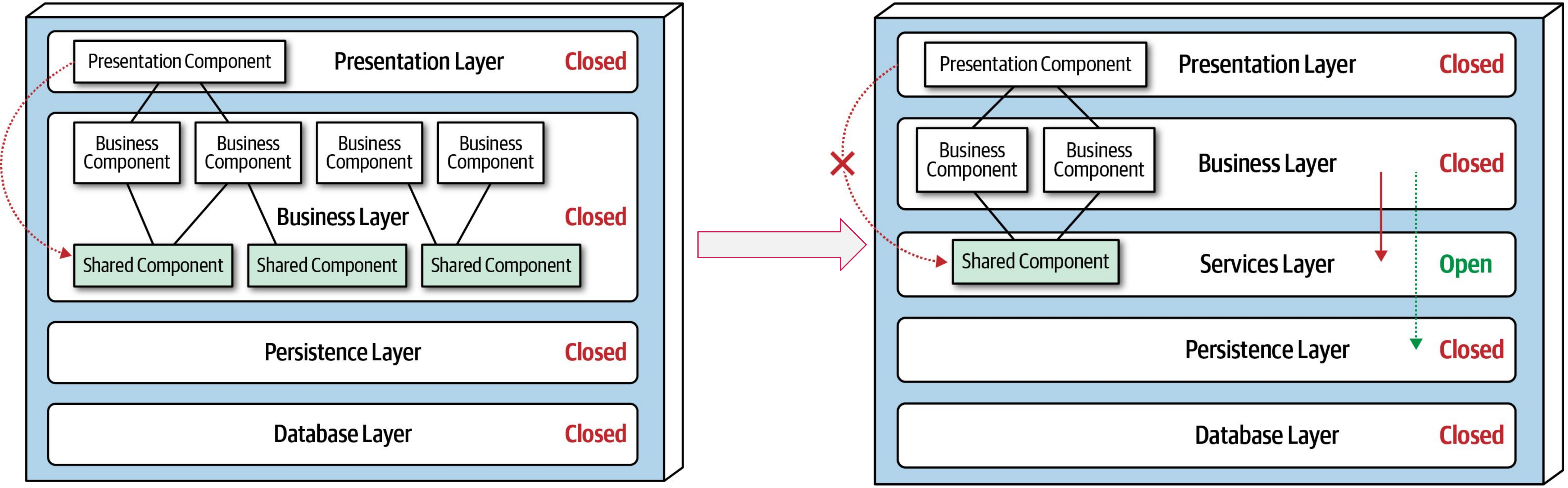
- Only adjacent layers can communicate directly
- Therefore, Each layer
  - Makes the overall system slower
  - (Should) simplify logical reasoning
- There is no best solution
  - Practice decides
- Exactly the same happens with n-tier architectures

Isolating layers

- Layers are either open or closed
  - Closed: the upper layer can communicate only with the lower one
  - Open: the layer can be "skipped"
- Layers of isolation
  - Changes in one layer do not affect other layers if contracts do not change
  - Otherwise, affect only upper layer
    - If this is closed!
- Performances vs coupling
  - Each closed layer adds more steps
  - Each open layer allow layers to depend on more than one



Adding layers



- Adding layers usually require they are open
  - Otherwise they should either expose all the interfaces of the lower layer, or the upper layer should be mostly rewritten

Summary of the layered architecture

- Good starting point to develop the system
- Reasonable and very natural choice
  - Easy to assign who should what inside the organization
- Still widely used (maybe for specific distributed subcomponents)
- Often architecture sinkhole antipattern appears
  - Layers are just forwarding calls to the lower layers
    - Slowing down the execution without improving decoupling
- In general, good choice for small (Web) applications

Your wooclap poll will be displayed here

1

Install the **Chrome** or **Firefox extension**

2

Make sure you are in **presentation mode**

wooclap

https://app.wooclap.com/events/SADM/0 question 13

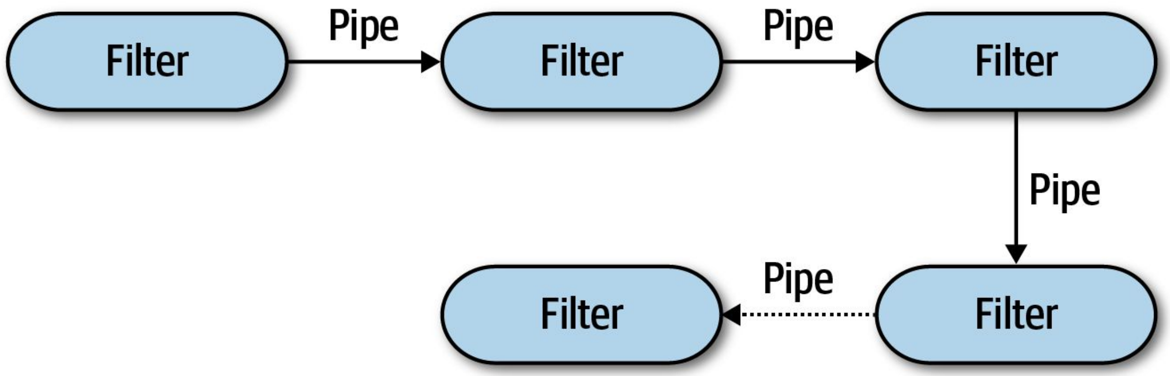
Layered architecture rating
Textbook solution

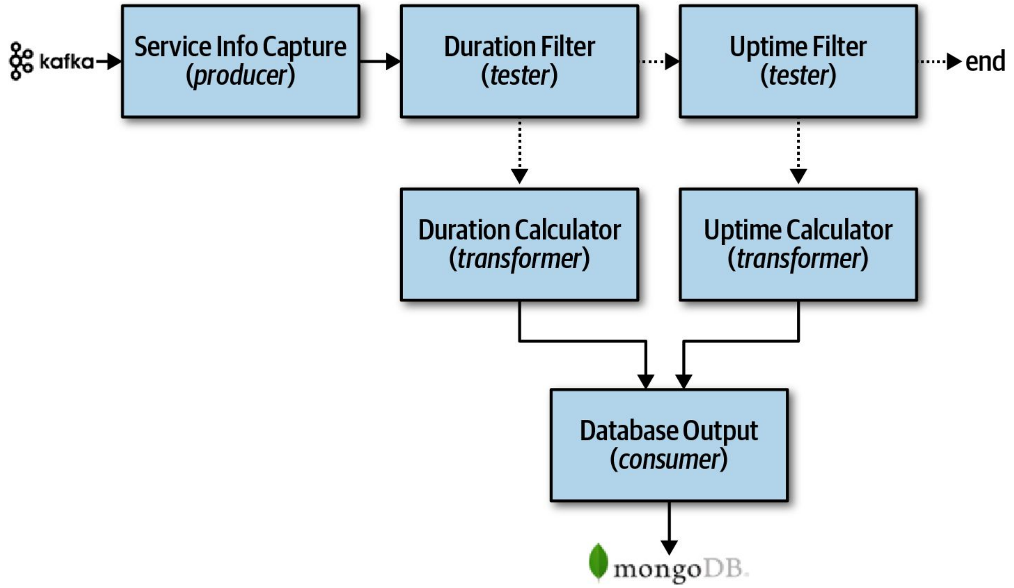
Deployability	★	**Performance**	★★
Elasticity	★	**Reliability**	★★★
Evolutionary	★	**Scalability**	★
Fault tolerance	★	**Simplicity**	★★★★★
Modularity	★	**Testability**	★★
Overall cost	★★★★★		

Ca' Foscari University of Venice

Pipeline architecture

- Are you familiar with bash scripts?
  - I hope so :)
  - Well, that's pipeline architecture!
- Pipes let data flow
  - From one filter to another one
- Filters perform some kind of processing




kafka
Service Info Capture (producer)
Duration Filter (tester)
Uptime Filter (tester)
end
Duration Calculator (transformer)
Uptime Calculator (transformer)
Database Output (consumer)
mongoDB.

Read a file of text, determine the n most frequently used words, and print out a sorted list of those words along with their frequencies.

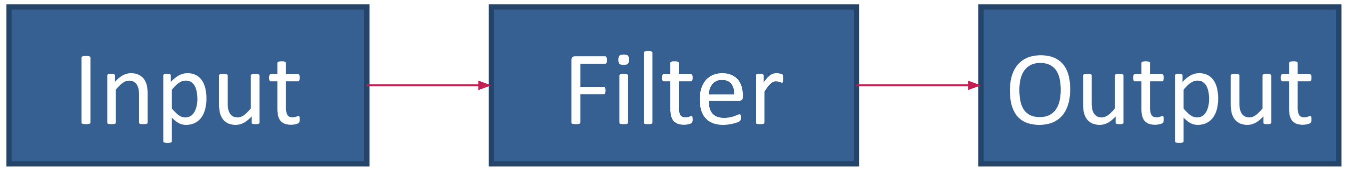
```
tr -cs A-Za-z '\n' |
tr A-Z a-z |
sort |
uniq -c |
sort -rn |
sed ${1}q
```

http://www.leancrew.com/all-this/2011/12/more-shell-less-egg/

Ca' Foscari University of Venice

Types of filters

- Filters must be
  - Self-contained
  - Independent
    - Work whatever is before or after
  - Stateless
    - Rely only on input
    - No interaction with persistent data
    - Given an input, always same output
  - Performing one task
    - Split complex tasks into many distinct filters



- Producer
  - The entry point: no input
- Transformer
  - Transform the input in some ways and and forwards it to the output
- Tester
  - Check some condition on the input
  - If it passes the test, forwards to the output
- Consumer
  - The exit point: no output
  - Show the result, store it, …

Your wooclap poll will be displayed here

1

Install the **Chrome** or **Firefox extension**

2

Make sure you are in **presentation mode**

wooclap

https://app.wooclap.com/events/SADM/0 question 14

Ca' Foscari University of Venice

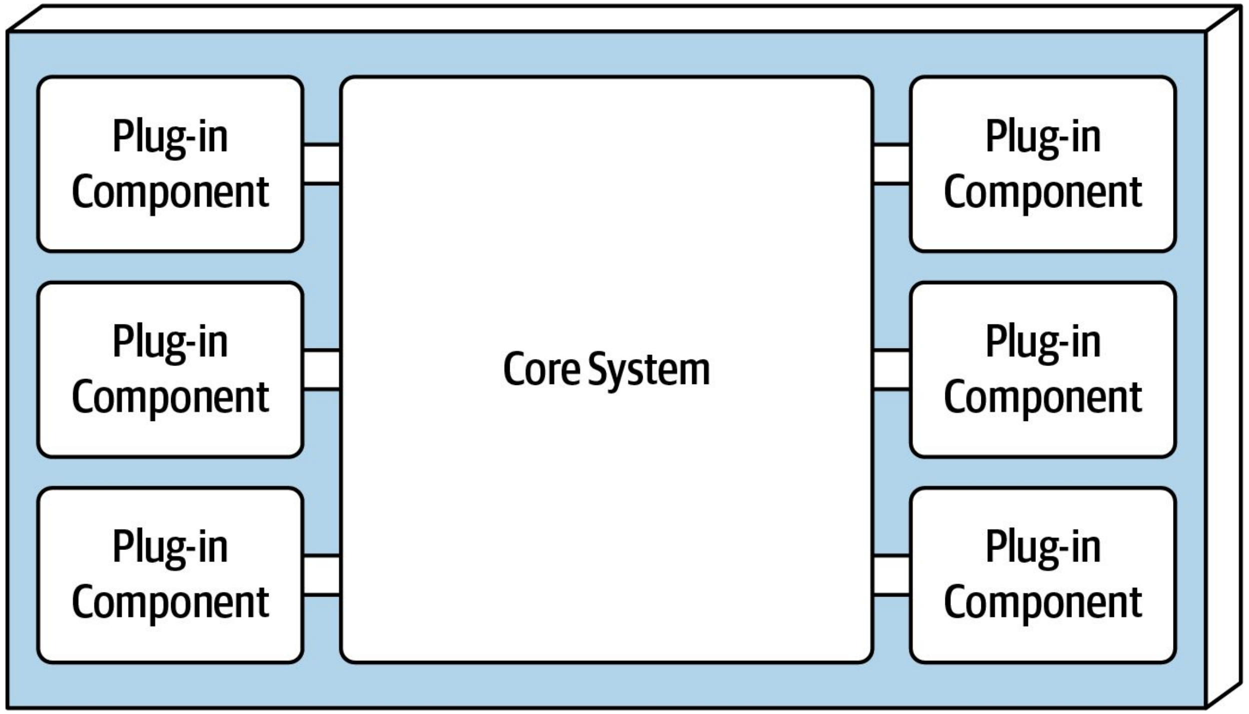
Pipeline architecture rating Textbook solution

Deployability	★★	Performance	★★
Elasticity	★	Reliability	★★★
Evolutionary	★★★	Scalability	★
Fault tolerance	★	Simplicity	★★★★★
Modularity	★★★	Testability	★★★
Overall cost	★★★★★		

Ca' Foscari University of Venice

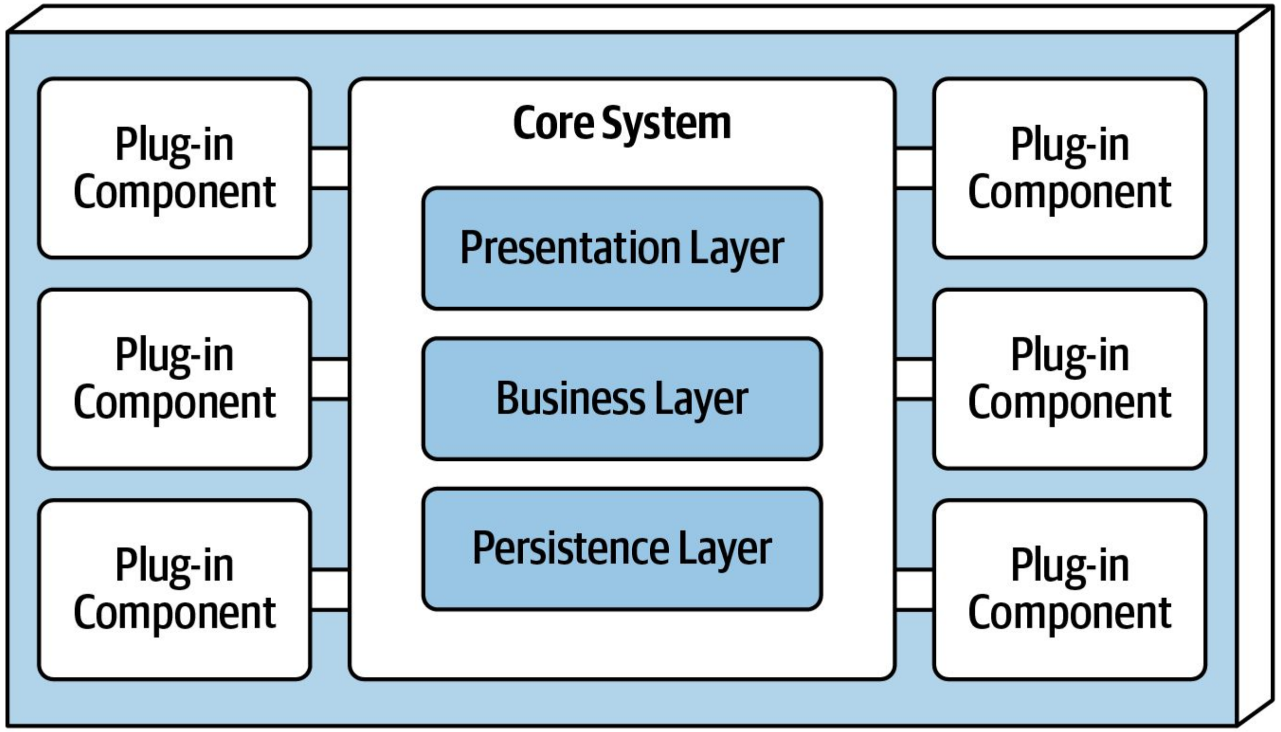
Microkernel (plug-in) architecture

- Two main components:
  - A core system
  - Plugins extending the core system
    - They add more and more features
- Widely used for several decades
  - E.g., Eclipse adopts this model
- Easy to extend
  - Just add more and more plugins
- It can be adapted
  - Different plugins in different contexts
  - Natural fit for product-based apps
- Isolate different features



Core system

- Minimal functionality required to run the system
- Eclipse: essentially notepad[++]
  - Core system not usable/interesting
- The core system might be complex
  - Structured in several layers
- Remember: architectures are not mutually exclusive!!!
  - Indeed, often combined together
  - Sometimes slight differences
    - Not clear the boundaries of one approach w.r.t. a similar one

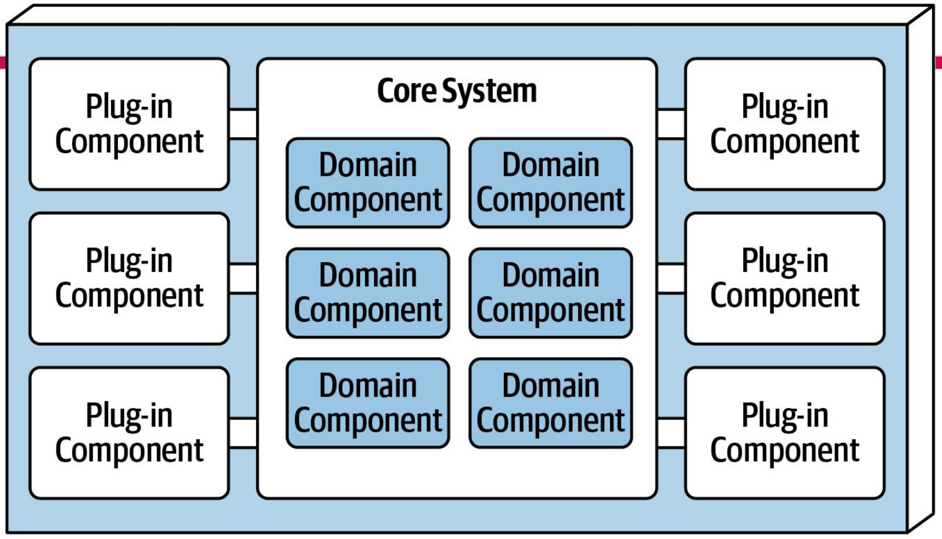


Layered Core System (Technically Partioned)

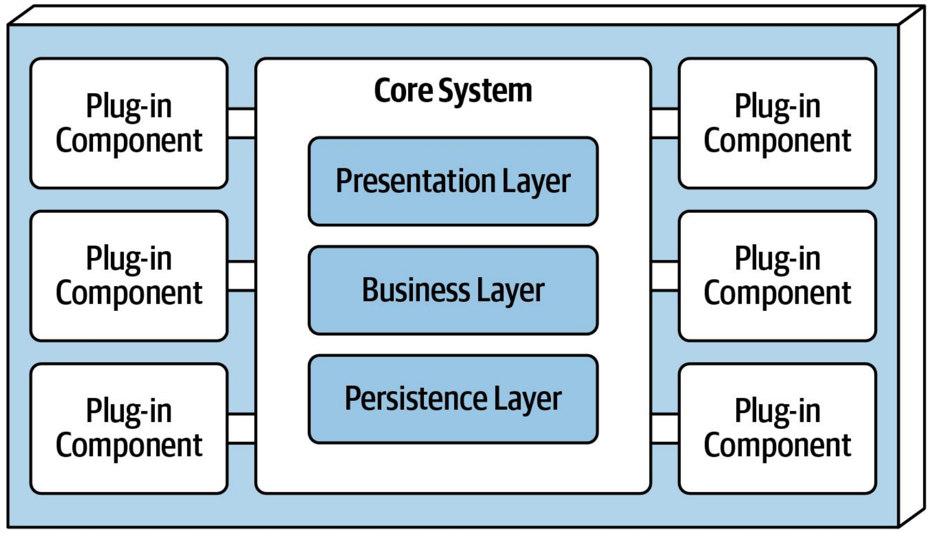
Ca' Foscari University of Venice

Technical vs Domain partitioning

- Layered architecture split technically
  - Natural choice in an enterprise
  - People with the same competences work on the same layer
- But also domain knowledge is important
  - Payment vs delivery vs inventory vs …
- The core system might be split into
  - Technical layers (presentation, business, DB …)
  - Domain components
- Plugins interface only with a domain
- Note that each domain component might be structured into several layers



Modular Core System (Domain Partioned)

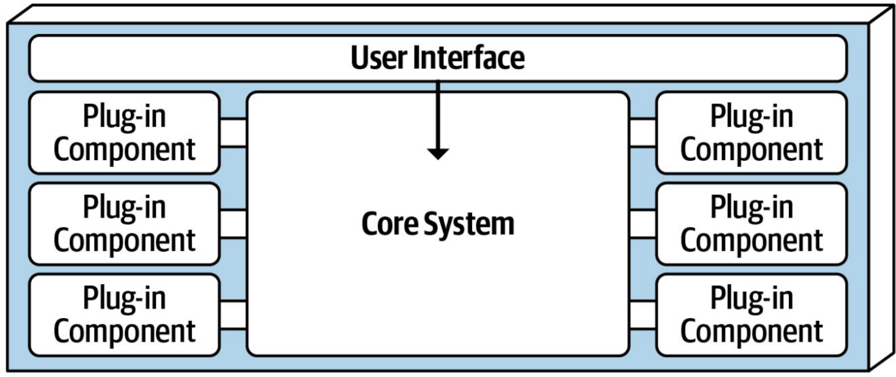


Layered Core System (Technically Partioned)

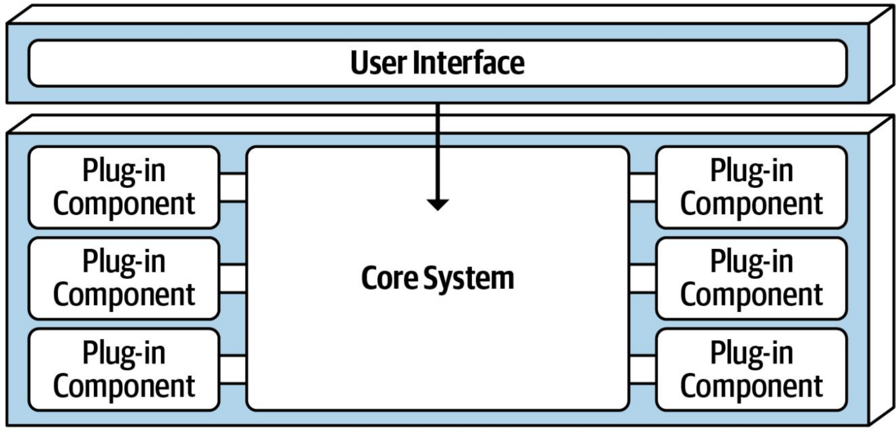
Ca' Foscari University of Venice

Presentation layer

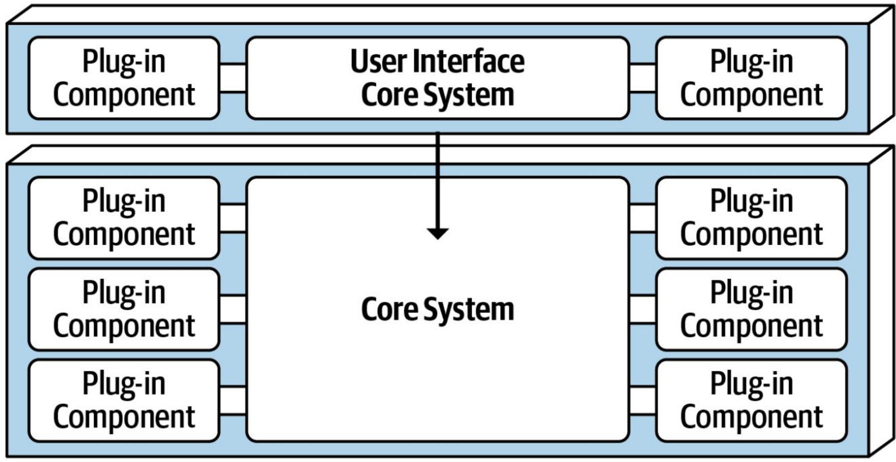
- The presentation layer (aka user interface) can be
  - a unique component with the rest of the system
  - a separated component with no logic
  - a separated component with some logic embedded
    - aka, a part of the core system
  - a pluggable separated components
- If separated, multiple deployments
  - Isolated, testable alone, etc etc
  - Need to be compatible with the rest
- Increased complexity



Embedded User Interface (Single Deployment)



Separate User Interface (Multiple Deployment Units)

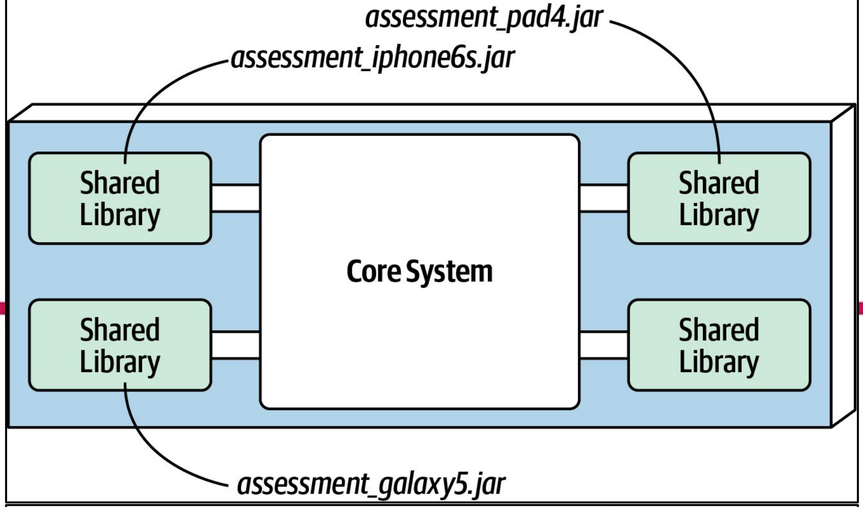


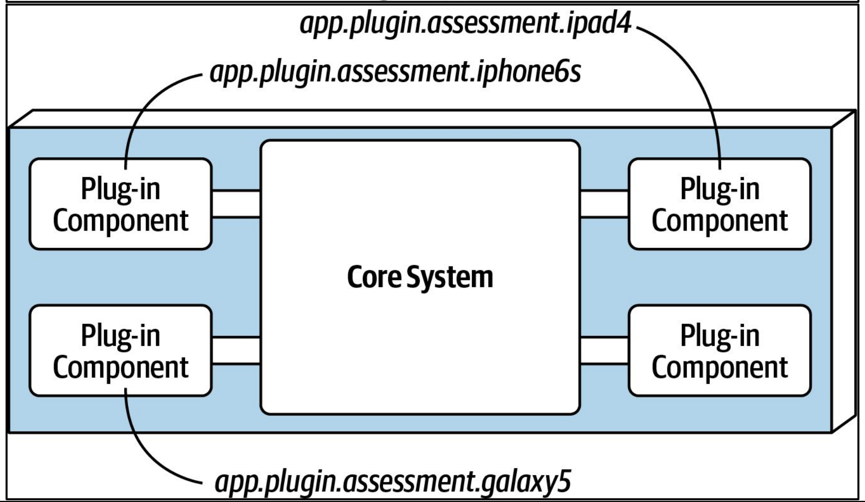
Separate User Interface (Multiple Deployment Units, Both Microkernel)

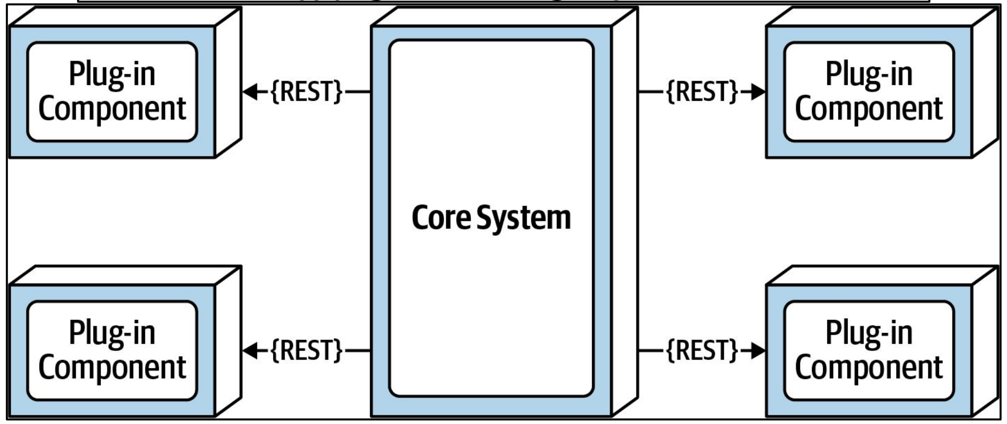
Ca' Foscari University of Venice

Plugins

- Independent, standalone, no dependencies, volatile
  - Tightly coupled with the core
  - Communicate through e.g. method call
- What is a plugin?
  - An artifact (jar file)
  - A namespace/package
  - A remote REST service
    - … but this is not monolithic!
    - Last architecture before distributed



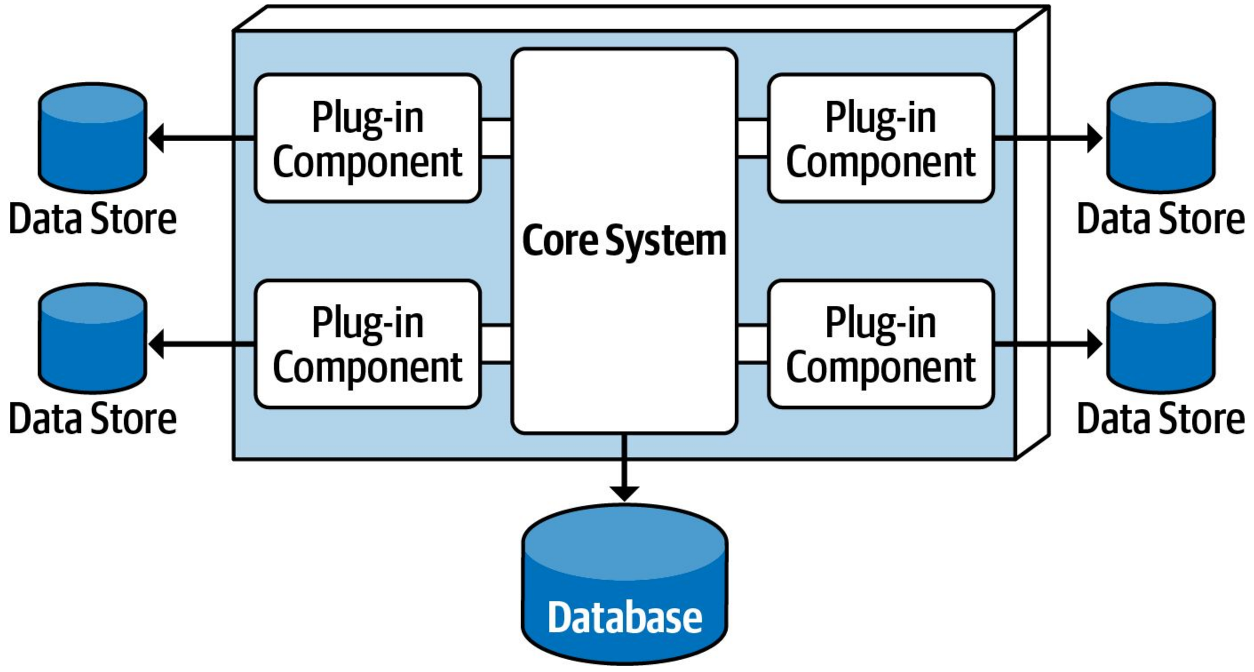

app.plugin.assessment.ipad4
app.plugin.assessment.iphone6s
Plug-in Component
Plug-in Component
Core System
Plug-in Component
Plug-in Component
app.plugin.assessment.galaxy5



Ca' Foscari University of Venice

Database

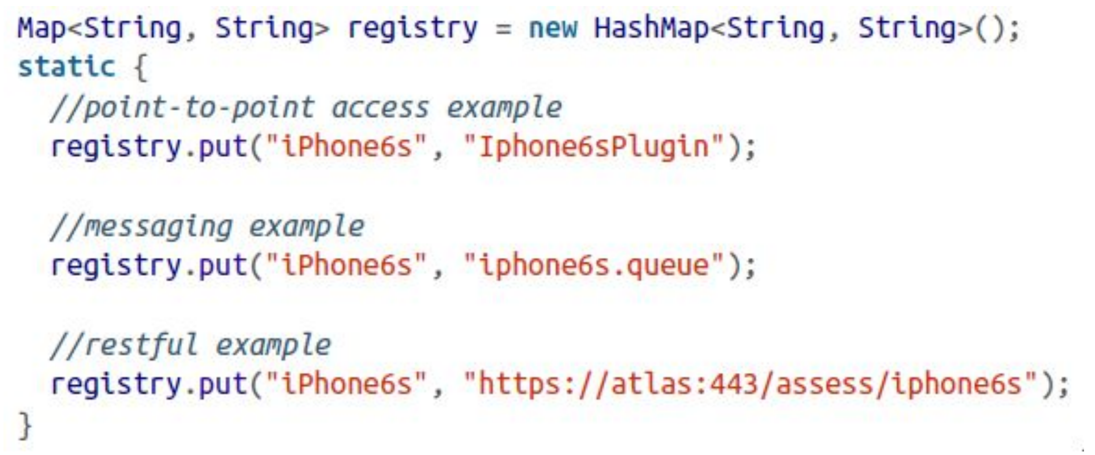
- Each plugins is isolated
  - It cannot rely on the presence of other plugins
- It can rely only on the model of
  - The core system
  - Its internal representation
- Usually, several distinct databases (or storages)
  - Each plugin has its own
  - A central database for the core system



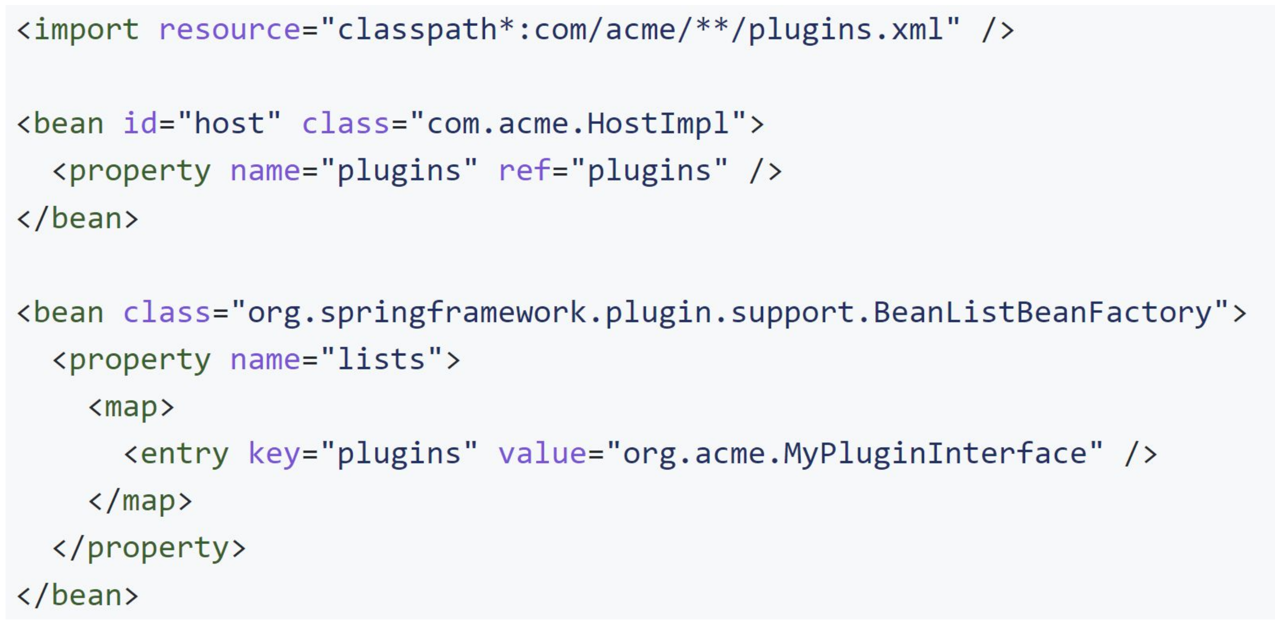
Ca' Foscari University of Venice

Registry

- How can the core system know what plugins are installed?
  - Through a plug-in registry!
  - Kind of a function that for each key returns info about the plugin
- Better if decoupled from code
  - XML, JSON, properties, … files
  - No need to recompile the code
- Then load plugins dinamically
  - Iterating over all defined plugins
  - E.g., through reflections



```java
Map<String, String> registry = new HashMap<String, String>();
static {
  //point-to-point access example
  registry.put("iPhone6s", "Iphone6sPlugin");

  //messaging example
  registry.put("iPhone6s", "iphone6s.queue");

  //restful example
  registry.put("iPhone6s", "https://atlas:443/assess/iphone6s");
}
```



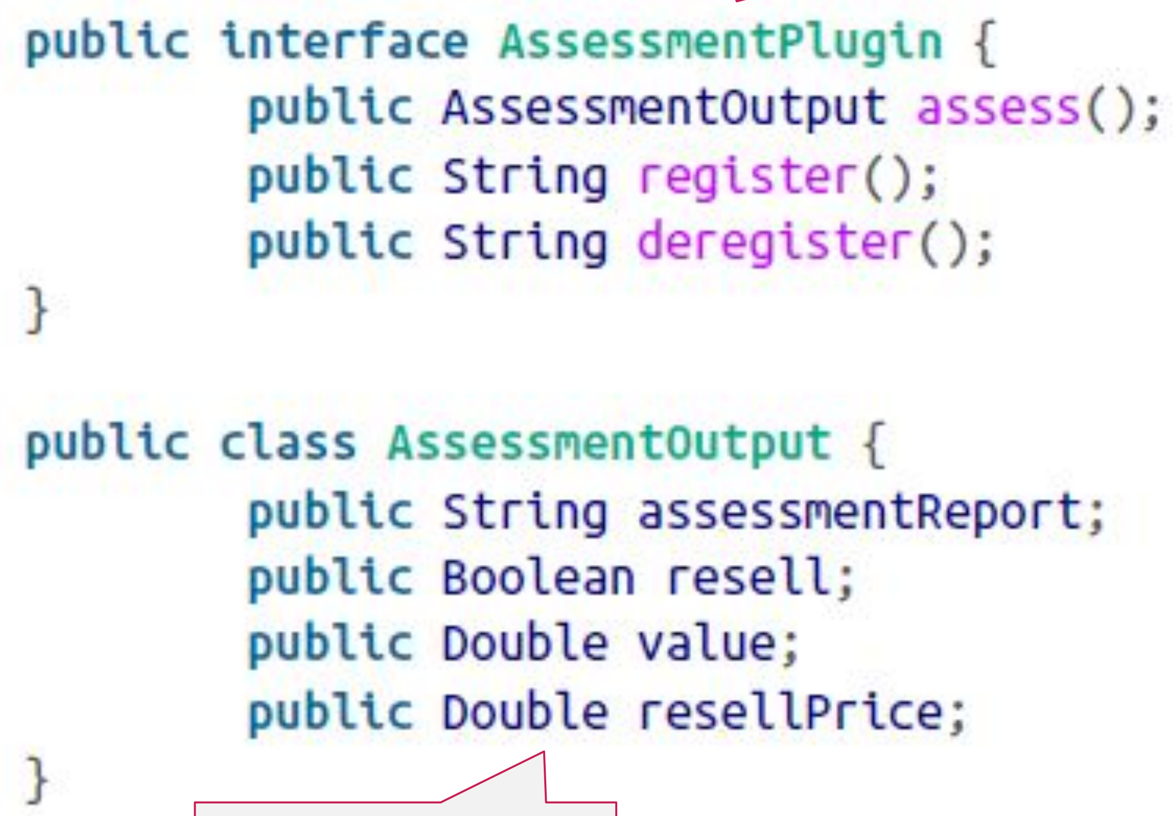
```xml
<import resource="classpath*:com/acme/**/plugins.xml" />

<bean id="host" class="com.acme.HostImpl">
  <property name="plugins" ref="plugins" />
</bean>

<bean class="org.springframework.plugin.support.BeanListBeanFactory">
  <property name="lists">
    <map>
      <entry key="plugins" value="org.acme.MyPluginInterface" />
    </map>
  </property>
</bean>
```

Ca' Foscari University of Venice

Contracts (aka API)

- Plugins and core system communicate
- Need to define a standardized way
  - The core system MUST NOT be specialized for single plugins
  - Add plugins without modifying the core
- Like registry, various ways
  - Standard classes/object, XML, JSON, …
- Various assumptions about how
  - Information is exchanged
  - The plugin can be "activated"
- Different versions of the core system might make different assumptions

How to use the plugin



```
public interface AssessmentPlugin {
        public AssessmentOutput assess();
        public String register();
        public String deregister();
}

public class AssessmentOutput {
        public String assessmentReport;
        public Boolean resell;
        public Double value;
        public Double resellPrice;
}
```

Data format

Your wooclap poll will be displayed here

1

Install the **Chrome** or **Firefox extension**

2

Make sure you are in **presentation mode**

wooclap

https://app.wooclap.com/events/SADM/0 question 15

Ca' Foscari University of Venice

Microkernel architecture rating Textbook solution

Deployability	★★★	**Performance**	★★★
Elasticity	★	**Reliability**	★★★
Evolutionary	★★★	**Scalability**	★
Fault tolerance	★	**Simplicity**	★★★★
Modularity	★★★	**Testability**	★★★
Overall cost	★★★★★		

References

- Textbook, Part II
  - Ball of mud: chapter 9
  - N-tier architecture: chapter 10
  - Pipeline architecture: chapter 11
  - Plugin architecture: chapter 12