1. The first is **Componentization via Service**.  
   What does it mean?  
   Let’s start from the basics.
2. d
   1. **Modular design is always a good idea**. Everyone knows that we always want to make our system modular so that updating it requires a small code change in a well-defined part instead of touching a monstrous piece of code.
   2. **When** talking about modular design, we also talk about components. **components** are the names we give to the parts that together compose the software.  
      In other words, when talking about modular software, we talk about software that has more than one component and each component is responsible for a specific aspect of the software.

Well. This is modularity. But how to implement?

* 1. Modularity can be achieved:
     1. Libraries:
        1. These are external code files being used in your code usually after declaring them using the keyword **import** in java.
        2. We use the libraries by directly calling the code in our code.
        3. The libraries’ code is executed within our system’s process so share the same compute resources and don’t need any kind of mediation (ESB in SOA) such as serializer or network in order to be used.
        4. The advantage is great performance as there is no mediator b/w our component and library and code is directly executed in the same process where caller resides.
     2. Services:
        1. The other type of modularity is using services.
        2. Services are out of process components and are called using out of process mechanisms such as Web API, RPC.
        3. Modern systems use mainly modern Web API such as REST.

1. So, these are two kinds of modularity.   
   Now, what they have to do with Microservices?
2. It has a lot to do with Microservices.
3. In Microservice, we prefer to implement modular design using services not libraries.   
   The componentization is the process of separating the software in separate components thus making it modular, is preferably done using services and not libraries.  
   That is why this attribute is named **Componentization by Services.**Now libraries can and should be used with these Microservices but in this case they are part of the service.   
   They don’t represent the whole componentization of the software, just the service itself.
4. Let’s look at an example:  
   Diagram, schematic

   Description automatically generated  
   This is monolith and modular design consisting of five components.  
   As we already said, in monolith the componentization is done via libraries. In fact, there is no other choice.  
   Monolith by definition runs in a single process. So, our componentization methodology must run in a single process and only libraries meet this requirement.

But in Microservice, componentization is done via services rather than libraries.   
Graphical user interface, application

Description automatically generated  
Every component is a service.   
It runs in its own process and uses the out of process mechanism such as Web API or RPC.  
  
A picture containing graphical user interface

Description automatically generatedif we focus on a specific service, it has libraries.   
As we said earlier, using library is best practice and highly recommended.  
The only thing to remember regarding microservice is that libraries are not used to make the whole system modular but they help in the inner working of service itself.  
In Microservice architecture, the modularity of the system is expressed by the services and this is the componentization by services.

1. Now what is the motivation for componentization by services?  
   Why would we want to use services for the modularity of our system instead of the simple faster libraries?  
   Graphical user interface, text, application

   Description automatically generatedThere are two reasons.
   1. Using services makes our component independently deployable and that too much easier.  
      That means we want to modify a single component, we deploy this single component.  
      If it is library, we need to deploy the whole app as the whole app including libraries run as a single process.
   2. Another reason is that using services as a component forces us, the architects, to define our component interface very well.  
      If we’re going to expose our components to the outer world as a Web API, then we have no choice other than design a well-defined Web API and psychologically when designing something that the whole world can use, we put a lot of thought into it, much more than in library.