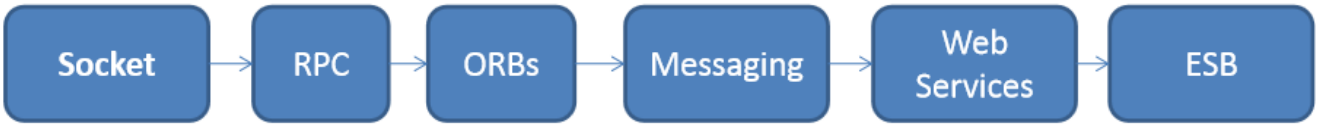
Following figure shows the six classical Software Integration methods.



Development methods of Information System Components

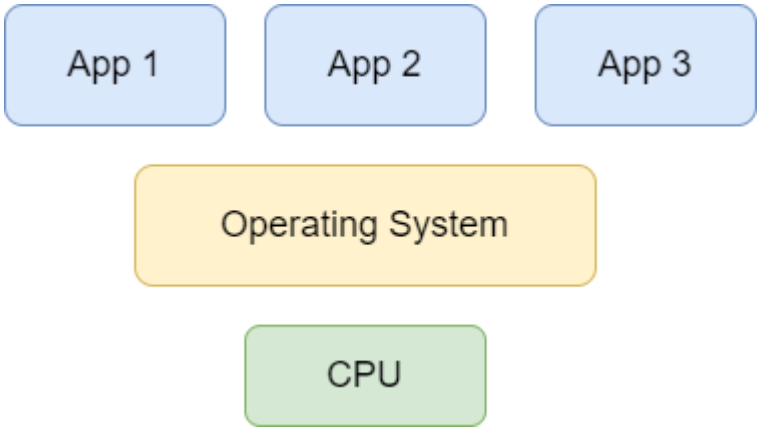
Below we present the basic methods by which IT system components can be developed. IT systems and some of their components differ from traditional stand-alone applications in that, **we expect them to function as a service with almost constant availability**.

However, in order for an application/software to function as a service, it immediately raises the following questions:

- How to control the life cycle of the component?
- How can we manage the resources of the component?
- Where/how can we get the configuration information needed to run?
- How can we communicate with the environment, or the other sw components?

Native development method

Although this is the oldest method, but it is still used in many ways today. For example, in embedded systems, or in containers.



We compile the source code for a specific CPU and Operating System combination: e.g. (x64/Ubuntu). It is possible to fine-tune the code to be run, to optimize its speed or size. The requirement to run the components continuously as a service, requires advanced software developer knowledge.

- c/c++/d compilers: msvc, gcc, clang, dlang
- strong knowledge of: pointers, references handling, heap/stack memory management, multi-threaded resource management
- it is the responsibility of the developer to release the allocated memory can be a big challenge: API calls must be known at the operating system level
- the integration of the various components is difficult: custom serialization methods must be implemented

There is no built-in resource management

- resource management is the responsibility of the developer

There is no widely used dependency management

- the standard handling of used dependencies (components developed by others) is not uniform, but after 2020, we can use https://conan.io/ or https://vcpkg.io/en/ or the modern https://cmake.org/cmake/help/latest/guide/using-dependencies/index.html.
- RELEASE/DEBUG mode compilation method slows down development
- it is possible that the operation of the program is different (erroneous, slower) with another compiler

the life cycle of the application (starting, stopping, monitoring) is managed by the operating system

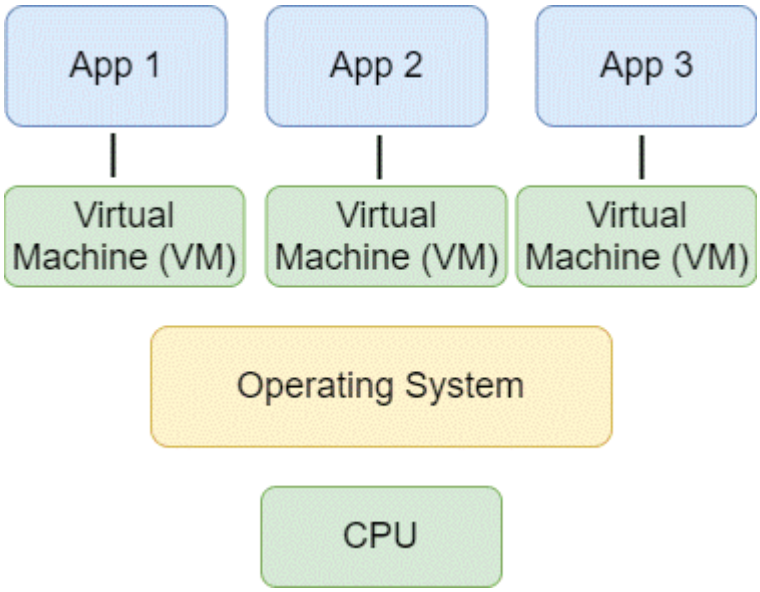
Special areas of application

- where maximum transaction speed is required
- IOT devices: insufficient memory is available to run or FPGA solutions are required

Challenges

- There is no standard exception handling
- The developer has to handle the bugs, unhandled problems lead to system crashes
- It is very difficult to find errors (memory dump, special logs)
- It's very easy to make a mistake - uninitialized data structures - (sanitizers)

Development on a software "Virtual Machine"



It has been widespread since the introduction of the Java VM (1997-). It defines a virtual processor and its associated so-called Byte Code, a set of instructions with its own machine code. It does not translate the source code directly to the CPU, but to the virtual machine's own byte code. Then VM converts the byte code to the machine code of the host system, when running. A virtual machine is usually a native application written in c/c++, which usually works on several platforms.

Most famous Virtual Machine implementations

- Java Virtual Machine (JVM)
- Nodejs, chromium engine
- Common Language Runtime (CLR): **.NET** system
- Zend Engine: **php**
- Adobe Flash Player: runs swf-s on the web (depricated)
- HHVM: php-based VM in facebook development
- ABAP: runs on the SAP Virtual Machine
- LLVM: this is not the classic VM, but it compiles the source into an llvm byte code, which then turns into native code. “LLVM is designed around a language-independent intermediate representation that serves as a portable, high-level assembly language that can be optimized with a variety of transformations over multiple passes.

Just in Time (JIT) translation

- The virtual machine can continuously optimize the application code, the byte code conversion is dynamic.

Memory management

- use of pointers is prohibited (generally)
- a special 'Garbage collection' algorithm is responsible for freeing unused memory

Resource management

- resource management is the responsibility of the developer
- but the VM has basic resource management capabilities
- Built-in, widely used dependency management is available (maven, pip, npm)

RELEASE/DEBUG mode compilation method is not distinguished

- we are running the code in “DEBUG mode”, the optimization remains hidden in the VM.
- the speed of the modern VM's are enough for general tasks.

The life cycle of the application (starting, stopping, monitoring) is managed by the virtual machine

Development of components it is ideal for developing collaborative components, as applications running on the VM can easily communicate with each other using TCP/IP network objects can be easily used and modified with the help of self-analysis. (Java reflection)