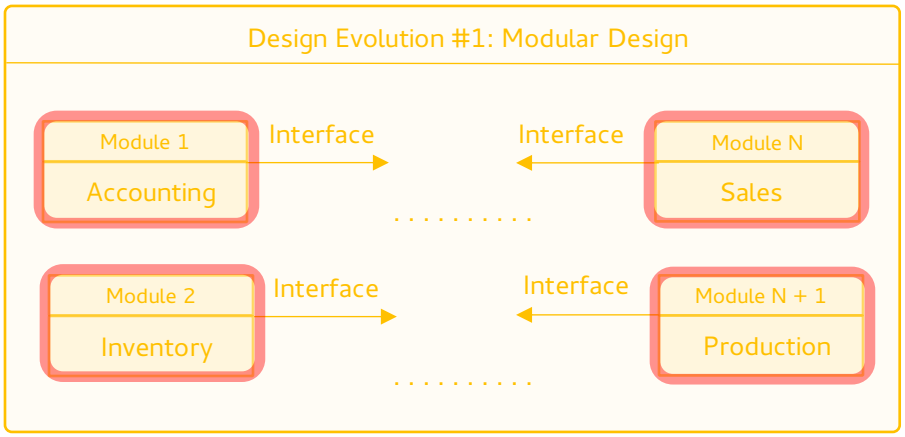
NULS 2.0 Platform

Design reasoning and explanation

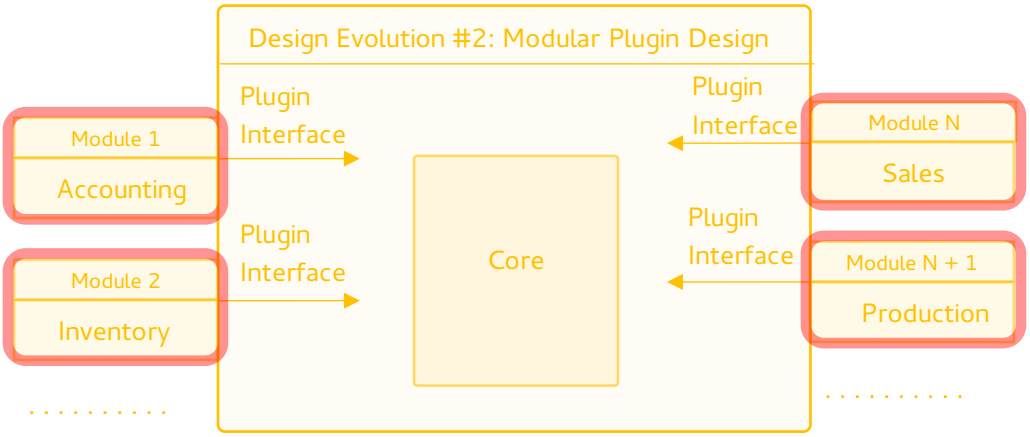
A few years ago I started with the design and development of a relative big project, an ERP (Enterprise Resource Planning) system customized for my country's legislation and it's tax laws, we begun the development and soon we noticed that modules could be developed concurrently if we could define a common way on how this modules could interact with the rest of the system, in other words, a standard interface was defined as shown in the simplified Fig. 1. This is more or less how Nuls 1.x is currently structured which is a lot better than the rest of crypto implementations.



This worked of course but as all modules were part of the same code structure we still needed to process and compile all modules in one go so a lot of coordination effort was required assuring all individual components were in good state to perform preprocessing checks and compilation, which was not something trivial to do because 25 modules plus 300+ components conformed the whole system.

The other problem was that even that some customers didn't need a module it was loaded nevertheless and consumed resources needlessly.

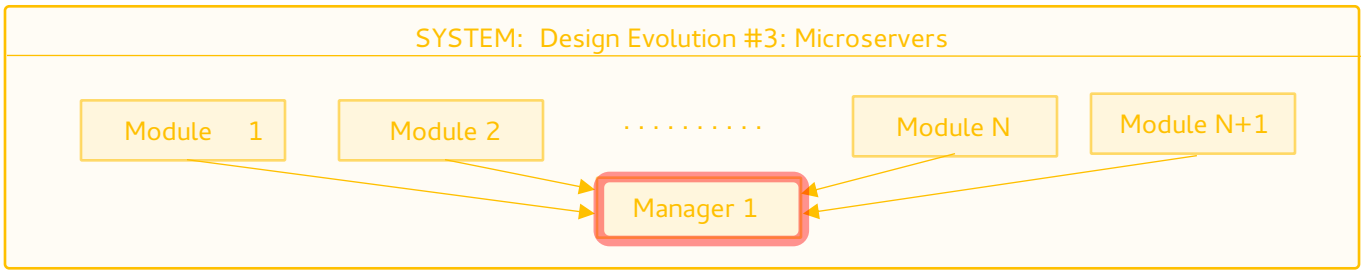
The solution I proposed was to refactor the whole code base in a way that modules could be decoupled from the monolithic source code allowing them to be compiled and deployed independently and also giving the possibility that the application could load them runtime only if needed, this is called a ***plugin.***

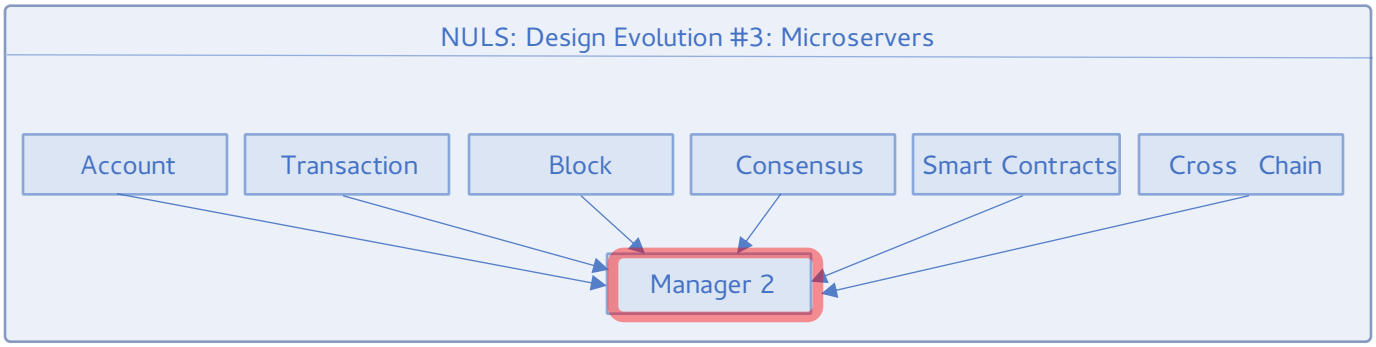


This architecture worked well a few years but it is now showing its age, the main problems are that testing and deployment still needs to be applied to the whole project, the user interface is still integral part of the code base, also just one development language needs to be used to enhance the system and make modifications which is less than ideal; the same problems apply to Nuls 1.x.

The solution for this is decoupling completely the modules from the core project making them applications themselves. This way each module could be developed on any language provided they follow some conventions and protocols. Each one of these applications is called a Microserver; from now on Microserver and module will be interchangeable terms.

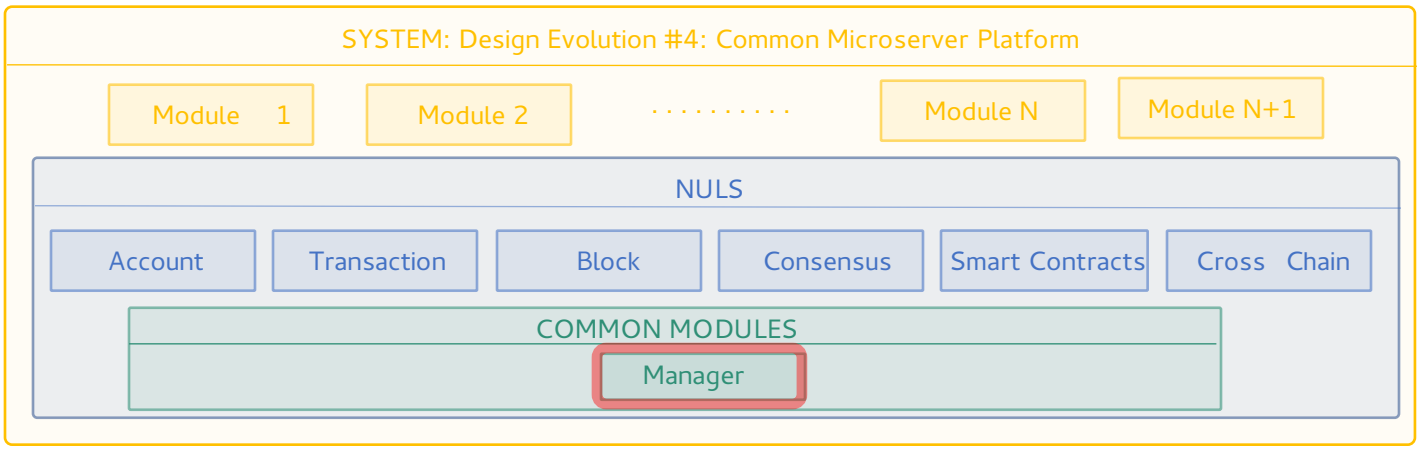
For this to work at least one Microserver should be responsible of coordinate how other Microservers should behave and exchange information, this module is called Manager.





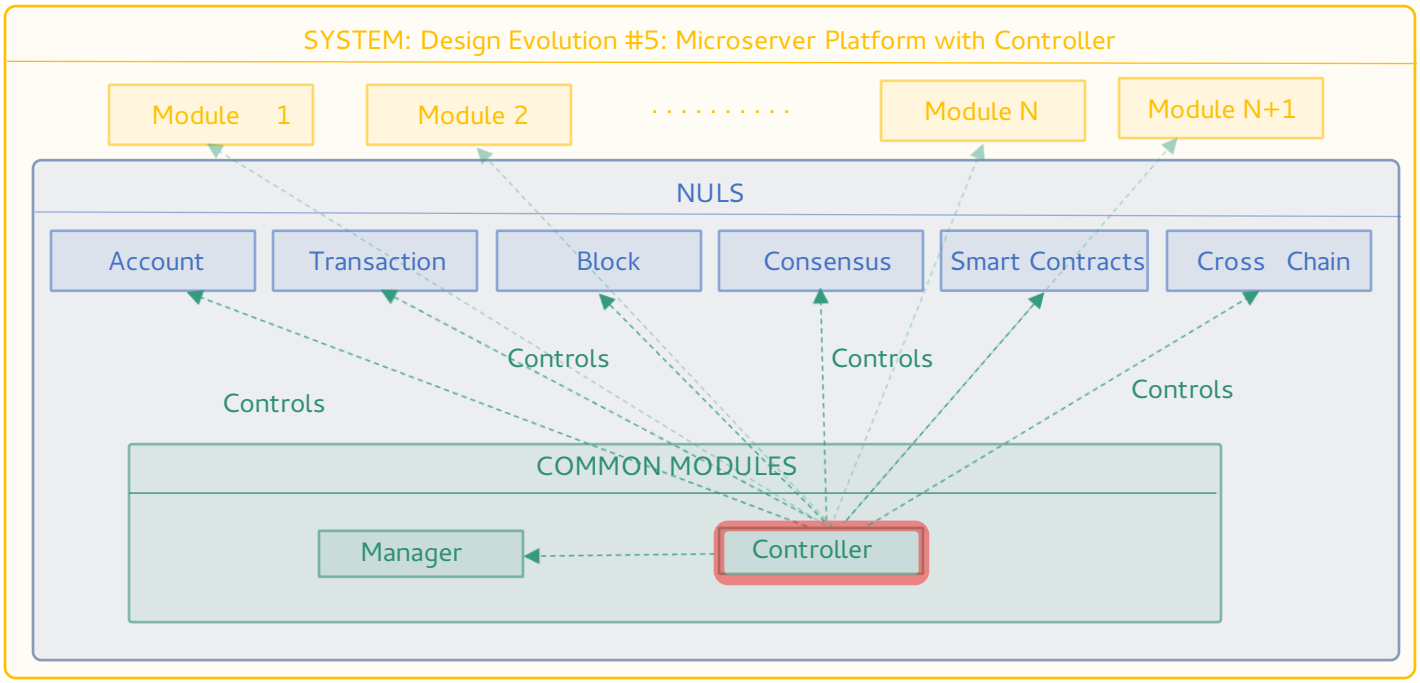
The idea can work for most system types so why not abstracting the Manager module and develop it so it can work for every use case?

The result is that the Manager module can be reused to code other systems without modification.



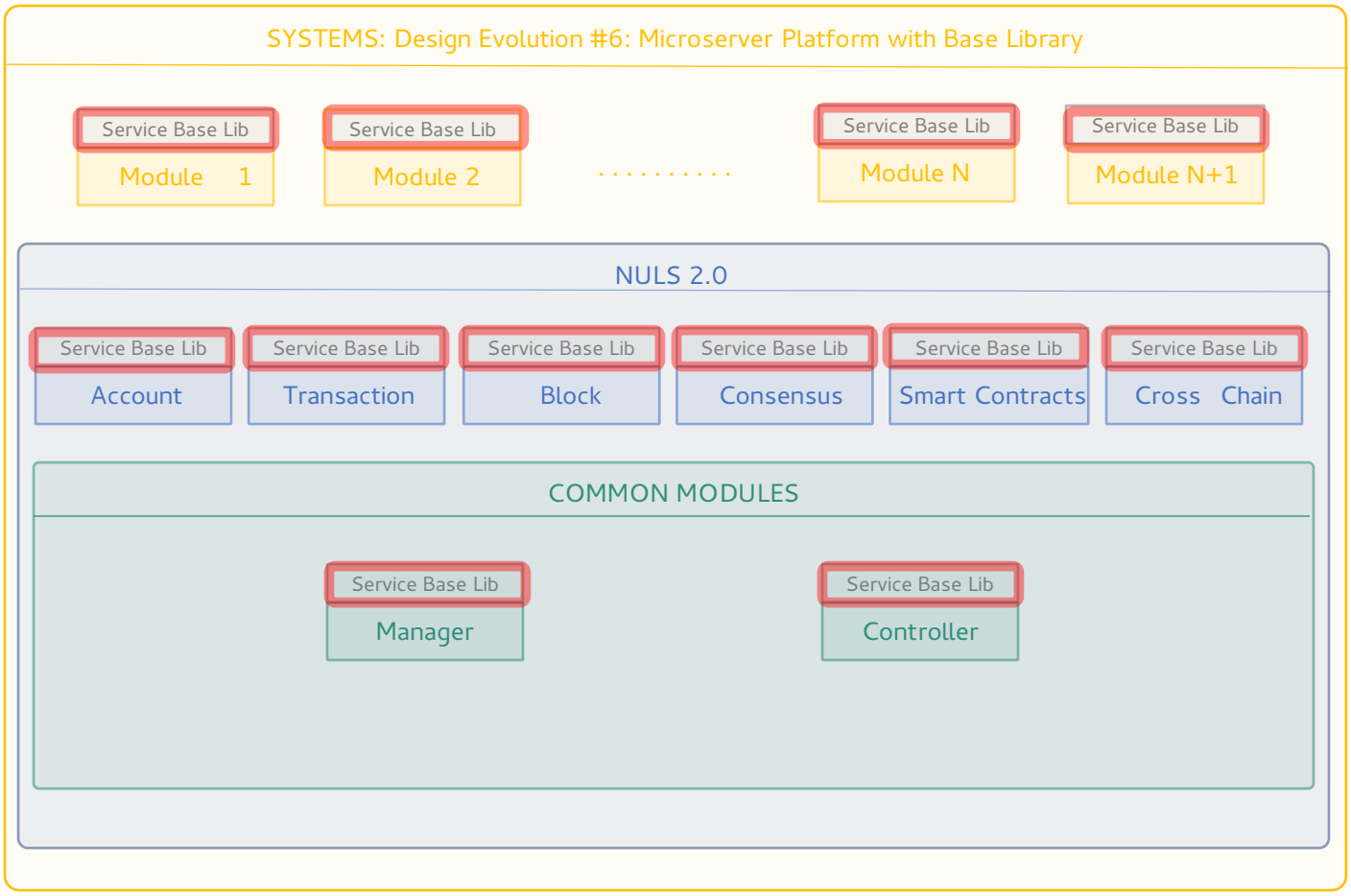
As every module is an application itself could it be possible to run them in different servers? This may be useful if some module requires more computing power than others of maybe it is a module that requires special security features. How can we make this work?

The answer is adding another common Microserver to the system: The Controller, which has the only functionality to start and stopping Microservers according to certain predefined rules, if some Microservers run in other servers then an instance of the Controller should be deployed on each server and configured to report information to the main server.



Looking good! But the number of modules is growing fast, and each one should know how to behave itself inside the system, they must understand how messages are sent and received and also establish a common format, they must know how to connect to other modules and when; all this work needs to be duplicated inside each module .... or not?

The best way to attack this problem is to develop a piece of software that all modules need to incorporate to their code in order to inherit all the common functions described so these rules don't need to be written again. This piece of software is called the Service Base Library.

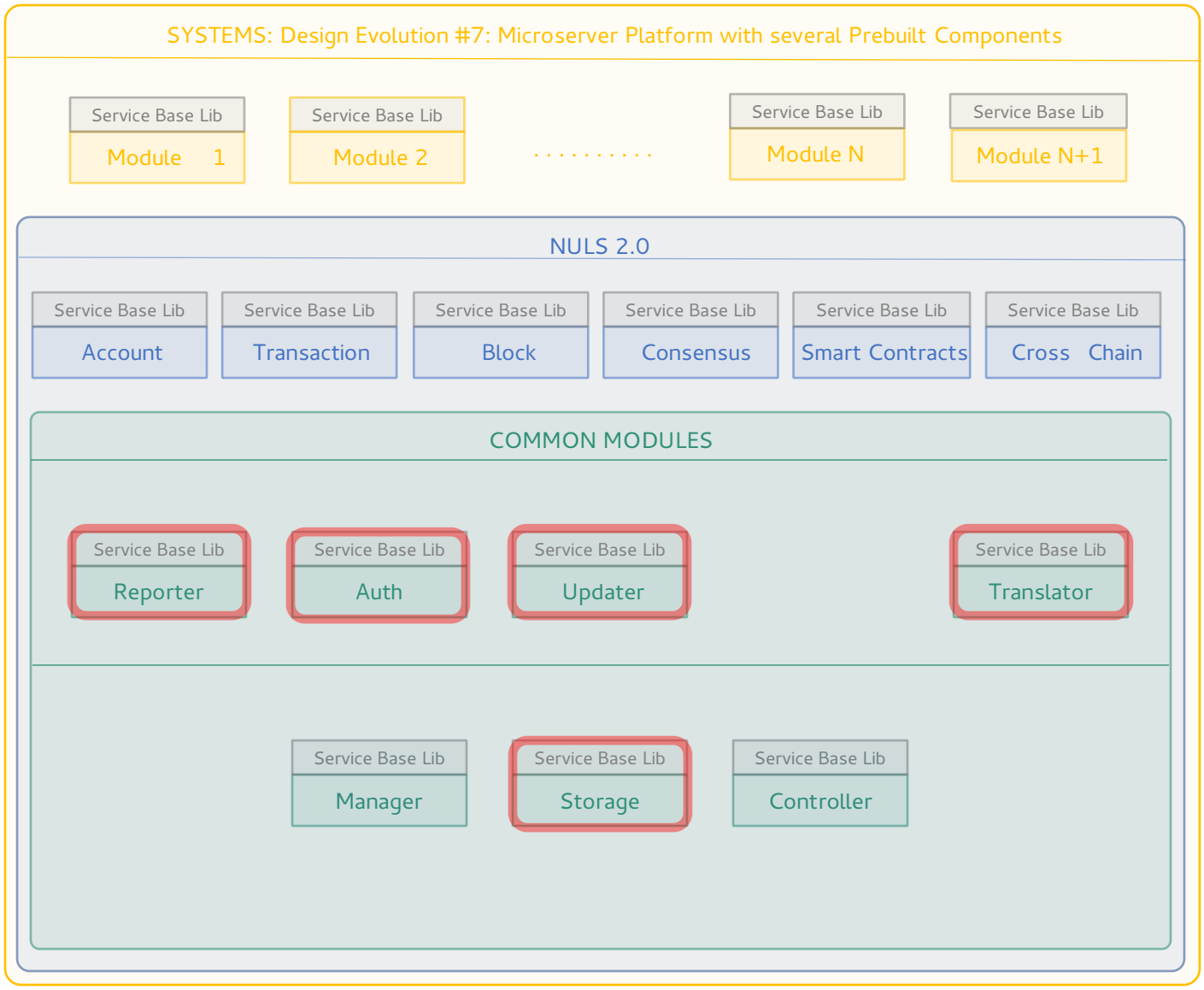


Now all modules can behave themselves without duplicating any work!

Avoiding code duplication for developers is always a good idea so why do not stretch the idea a bit more? Many systems share a lot of common functions:

- Storing data, whether in a file or in a database.
- Some type of authentication.
- Generating reports.
- Offer translations to different idioms.
- Update the software.

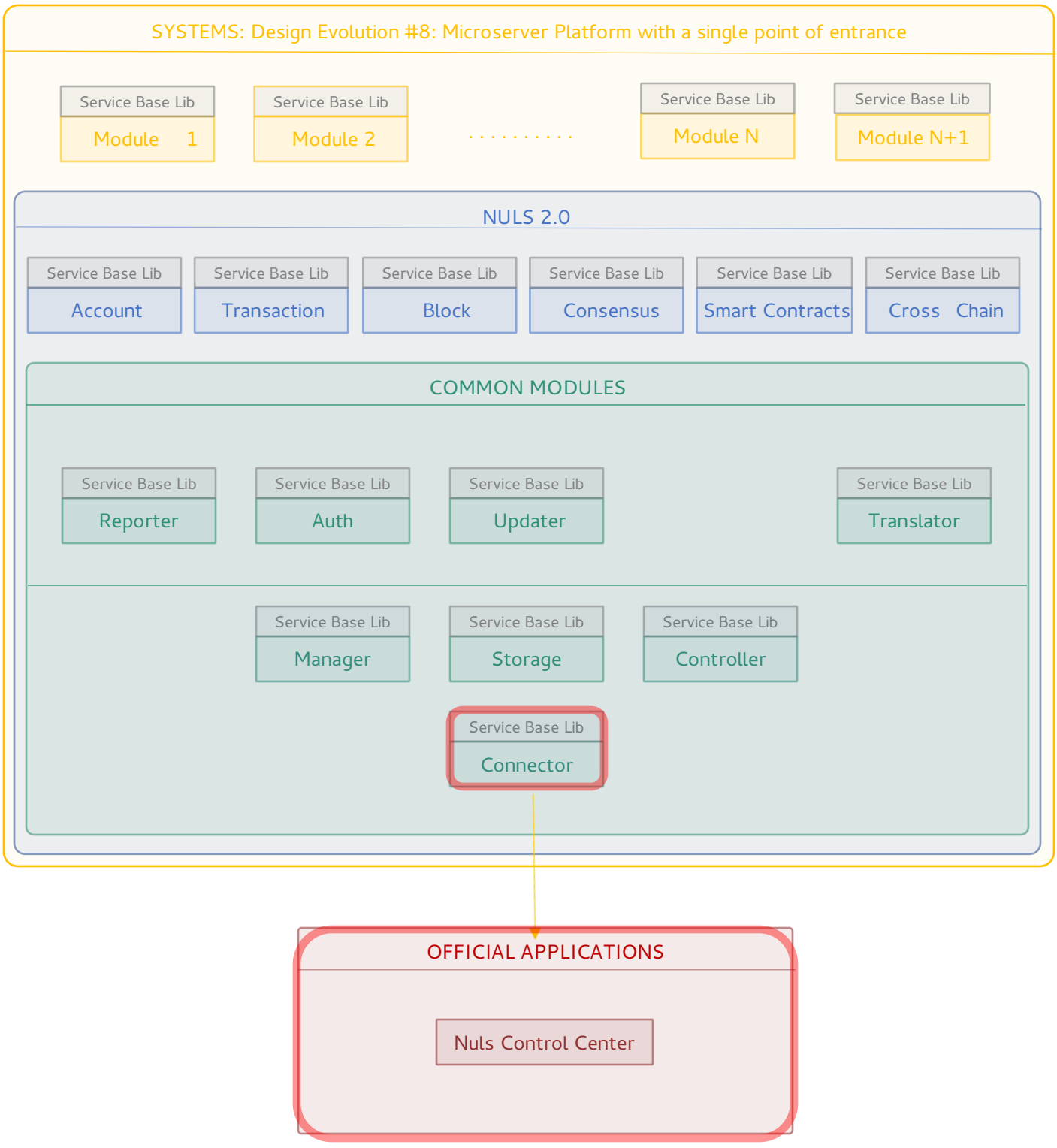
It seems logic that in order to avoid duplication of work for developers we just need to develop a standard set of modules to offer these common functionalities for different types of systems.



Wait! What about the user interface? Or speaking more broadly what about external applications that require to connect to the system? Do they need to connect to ALL Microservers at the same time?

For this we need to incorporate another module: The Connector, this module will collect the list of services and functions that the other modules provide and will expose them as it were just one module to the outside world, this is called the API (Application Programming Interface).

The Conenctor Microserver is the only point of entrance and therefor the only module the user interface application (GUI) and other programs will need to interact to take advantage of the whole system. The GUI will serve not only as a wallet but will have many more functions thereby a name change is in order: Nuls Command Center

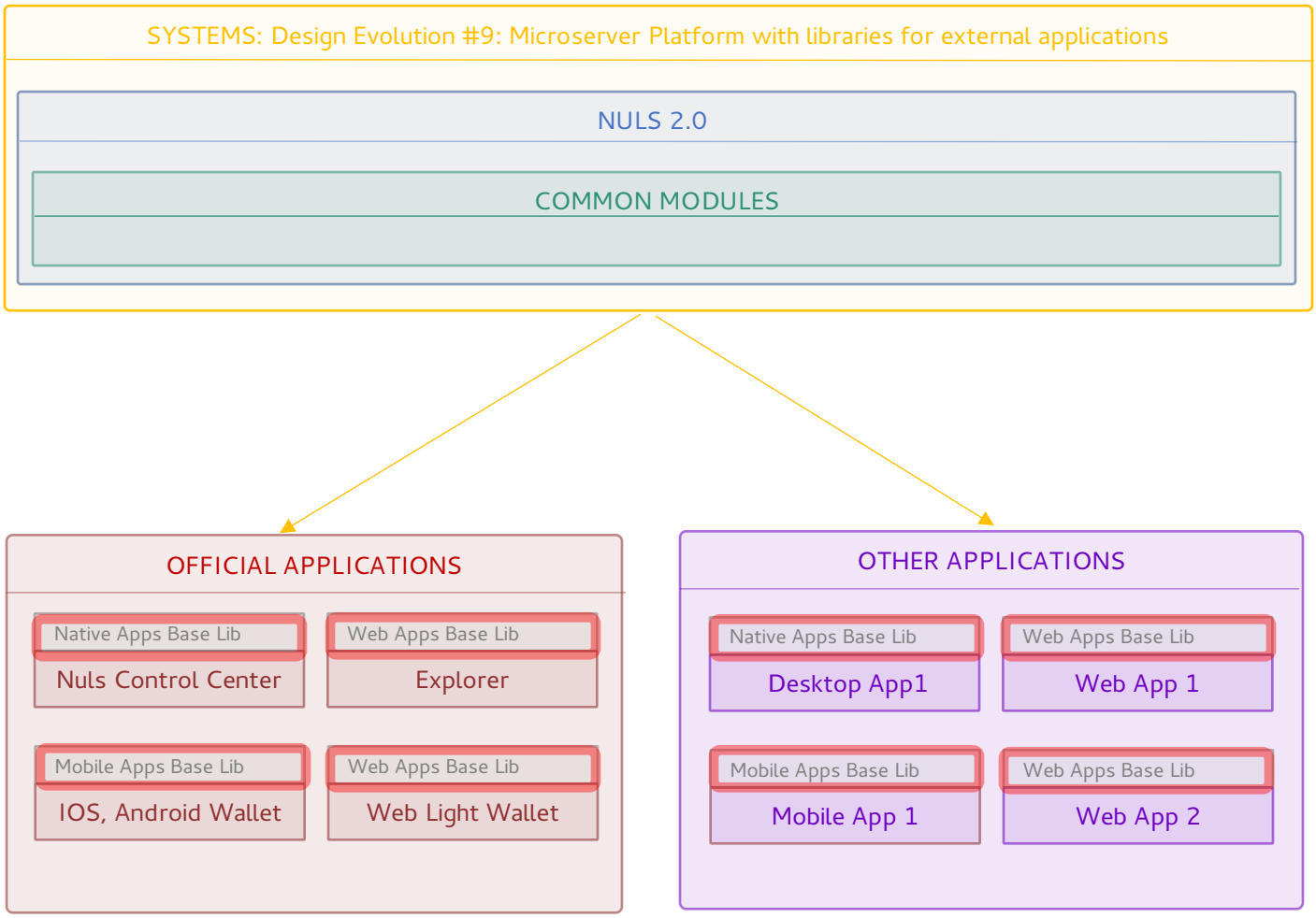

SYSTEMS: Design Evolution #8: Microserver Platform with a single point of entrance
Service Base Lib
Module 1
Service Base Lib
Module 2
Service Base Lib
Module N
Service Base Lib
Module N+1
NULS 2.0
Service Base Lib
Account
Service Base Lib
Transaction
Service Base Lib
Block
Service Base Lib
Consensus
Service Base Lib
Smart Contracts
Service Base Lib
Cross Chain
COMMON MODULES
Service Base Lib
Reporter
Service Base Lib
Auth
Service Base Lib
Updater
Service Base Lib
Translator
Service Base Lib
Manager
Service Base Lib
Storage
Service Base Lib
Controller
Service Base Lib
Connector
OFFICIAL APPLICATIONS
Nuls Control Center

OK we have a GUI! But surely this isn't enough to satisfy blockchain technology hungry supporters these days, so other official applications must be crafted, probably Nuls 1.x applications like the explorer, light wallet and others will have to be tweaked a little to work with new platform, but in order to reduce work for application developers (which is common theme in this article) it would be reasonable to build libraries that allow external applications to connect seamlessly to the platform.

These Libraries are pieces of code that *know* the gory details on how to manage network resources, crafting messages, protocol handling, etc. so developers will only need to focus on the problem that their application want to solve and NOT on how to integrate them to our platform. In time these libraries should be ported to multiple languages to attract more external developers to the project.

These libraries are:

- Native Base Application Library.
- Mobile Base Application Library.
- Web Base Application Library

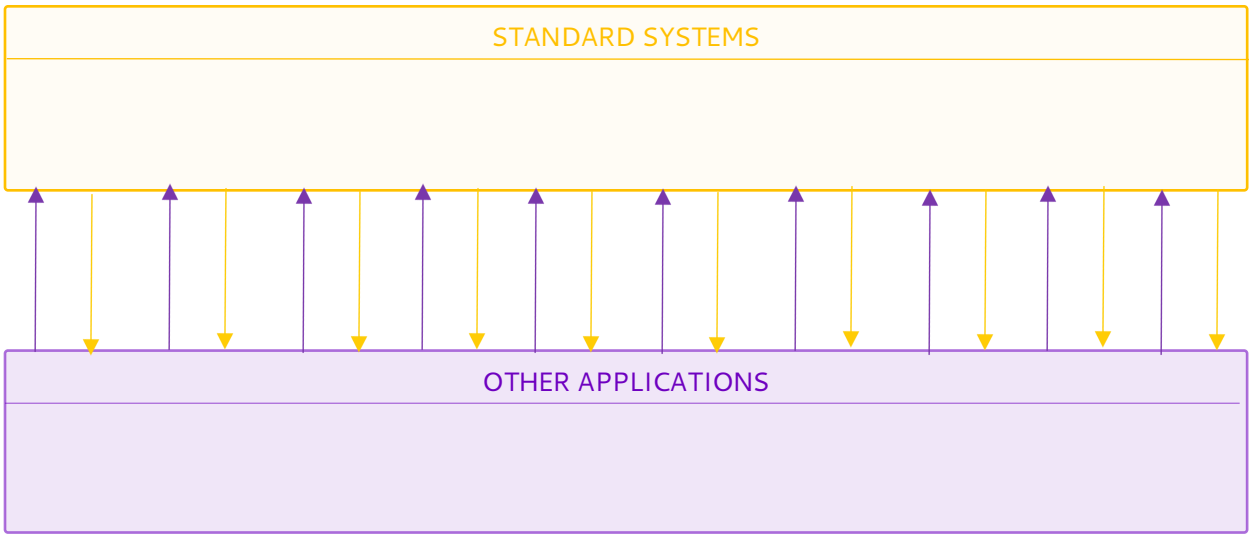


External applications need to interchange message with NULS 2.0 system, how will this interchange work?

There are several ways to do this, by far the most common pattern used (implemented in all other blockchain projects) is one called Request/Reply, the external application sends a request to the system, and the system sends a reply, sounds logical right?

Let's dig a little deeper, assume that our application needs to show the balance of our address, so we would need to send a 'getbalance' Request and wait for a Reply and then display the number returned, but we surely want this to be updated constantly, for this we would need to send the 'getbalance' Request every 5 seconds for example to keep the balance to be displayed correctly.

In reality the operating system needs to add a lot of information to the 'getbalance' Request so the message could travel the internet, hopping from router to router a process which appends even more information to the message so it can finally reach its destination without getting lost in the way, and we need to repeat this process every 5 seconds just to keep the balance updated in our application.

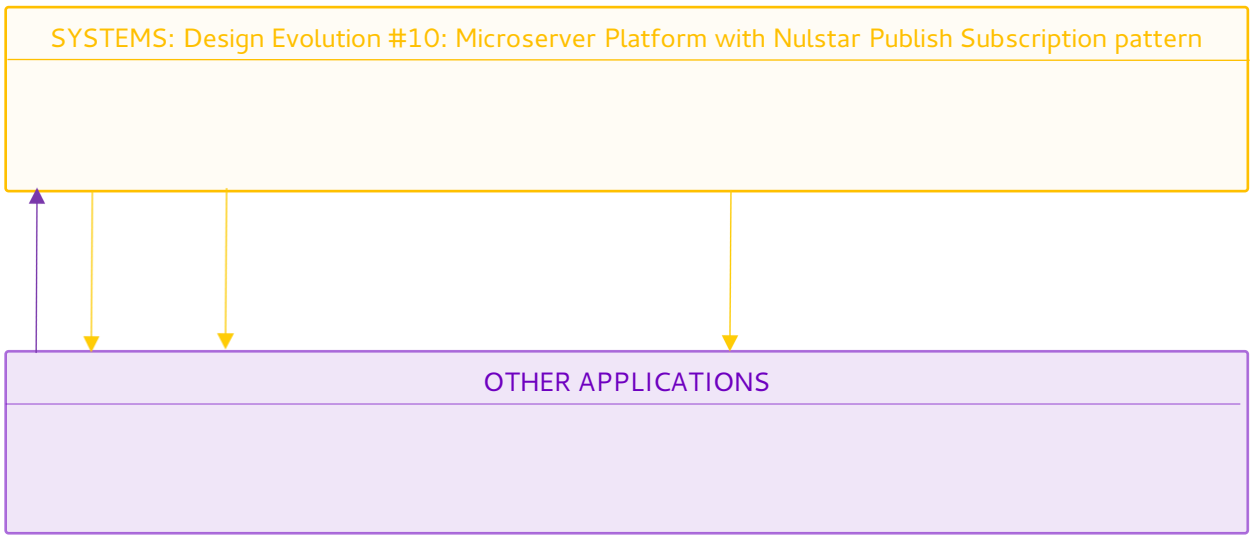


Is there a way we can reduce the amount of traffic generated? Gladly the answer is YES!

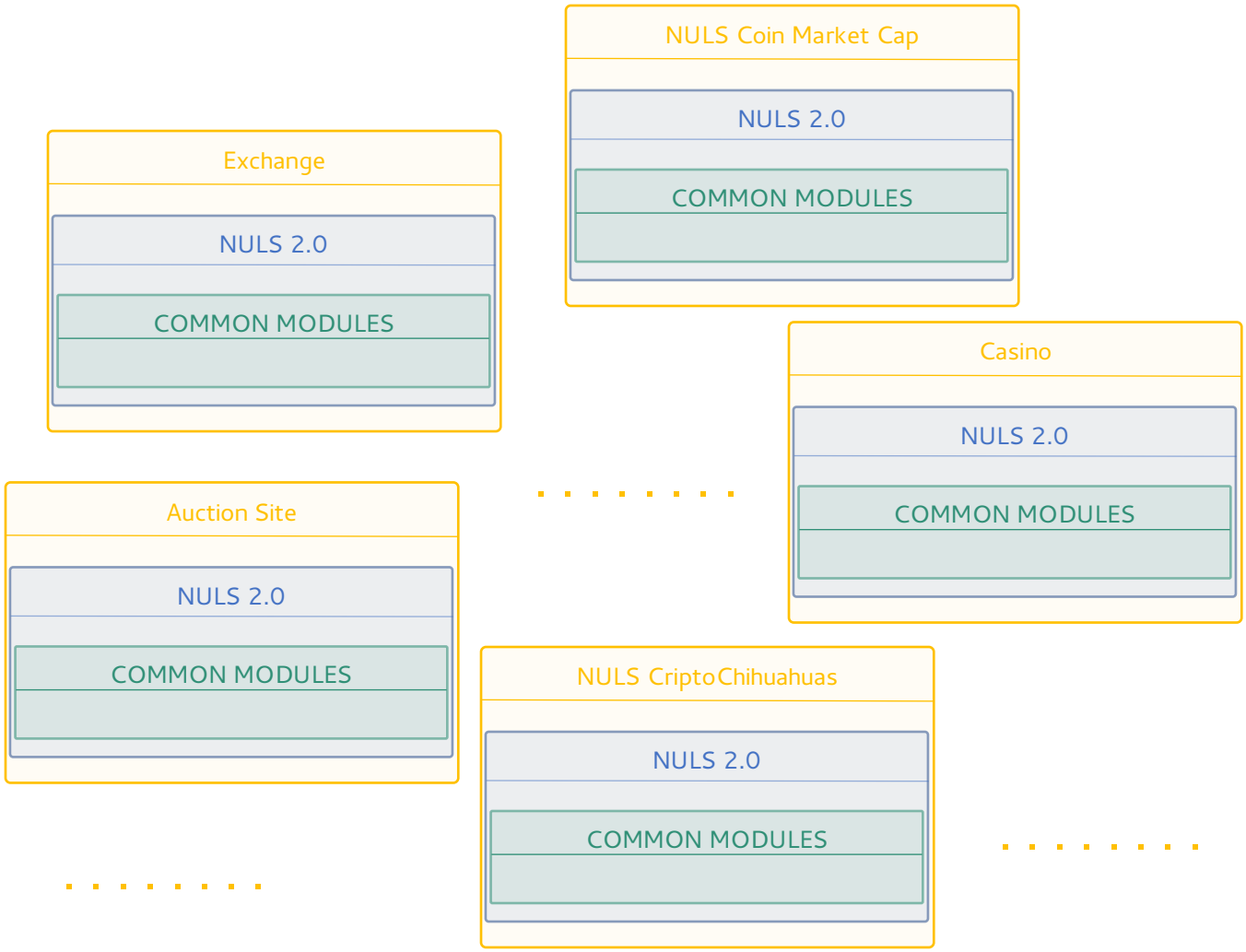
How about instead of just calling a function hundreds of times we just ***subscribe*** to it? so we can tell the system that we want to receive the balance just when it changes.

This type of message exchanging is a heavily optimized variation of a pattern called Publish/Subscription and its better suited to handle message interchange in modern systems and applications.

I call this pattern: "Nulstar Publish Subscription (NPS)" pattern.

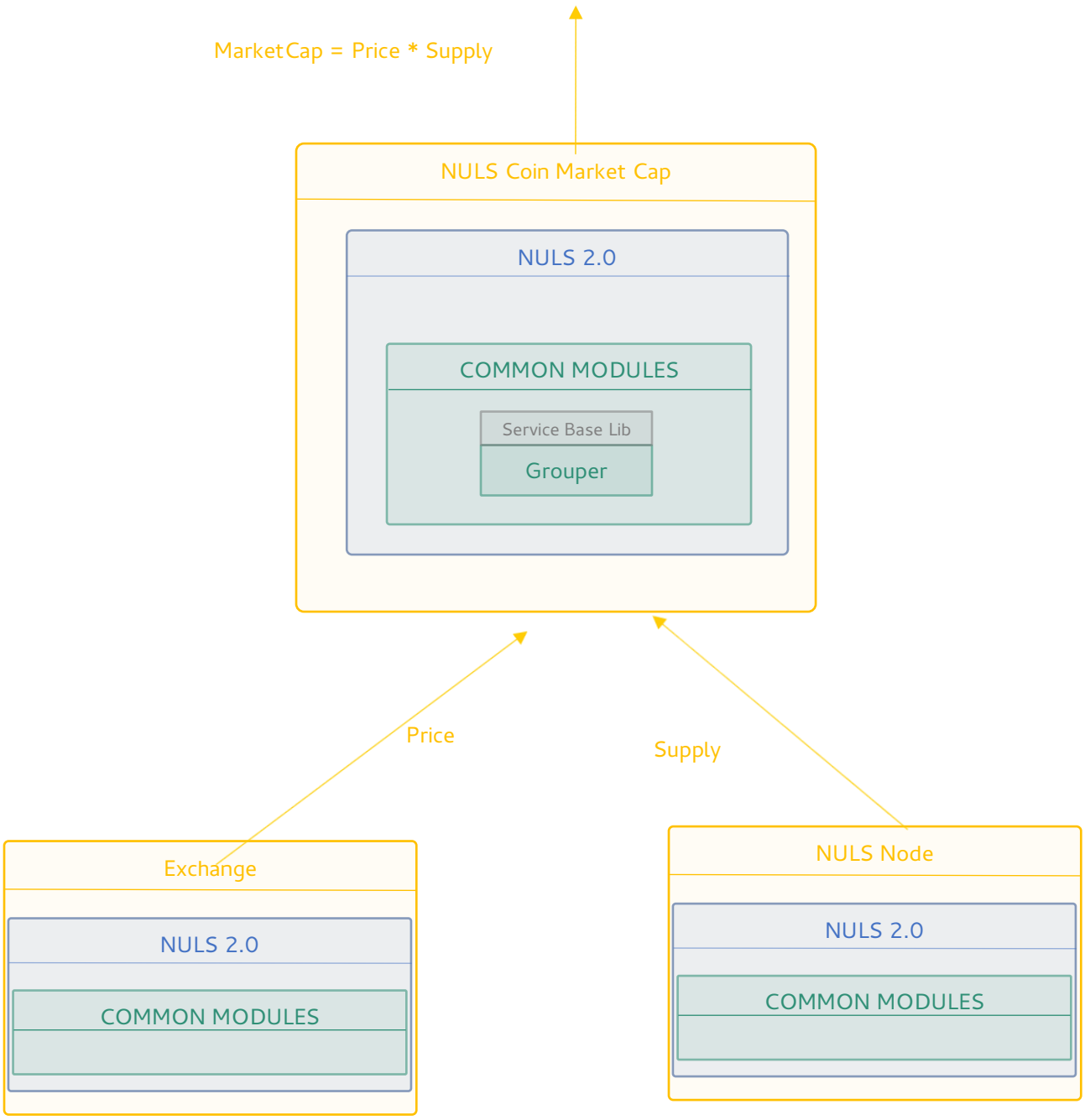


At this instance we are able to build several type of systems reusing the basic components presented and adding only specific modules, shortening the time and therefore the cost for developers and entrepreneurs to create businesses around NULS.

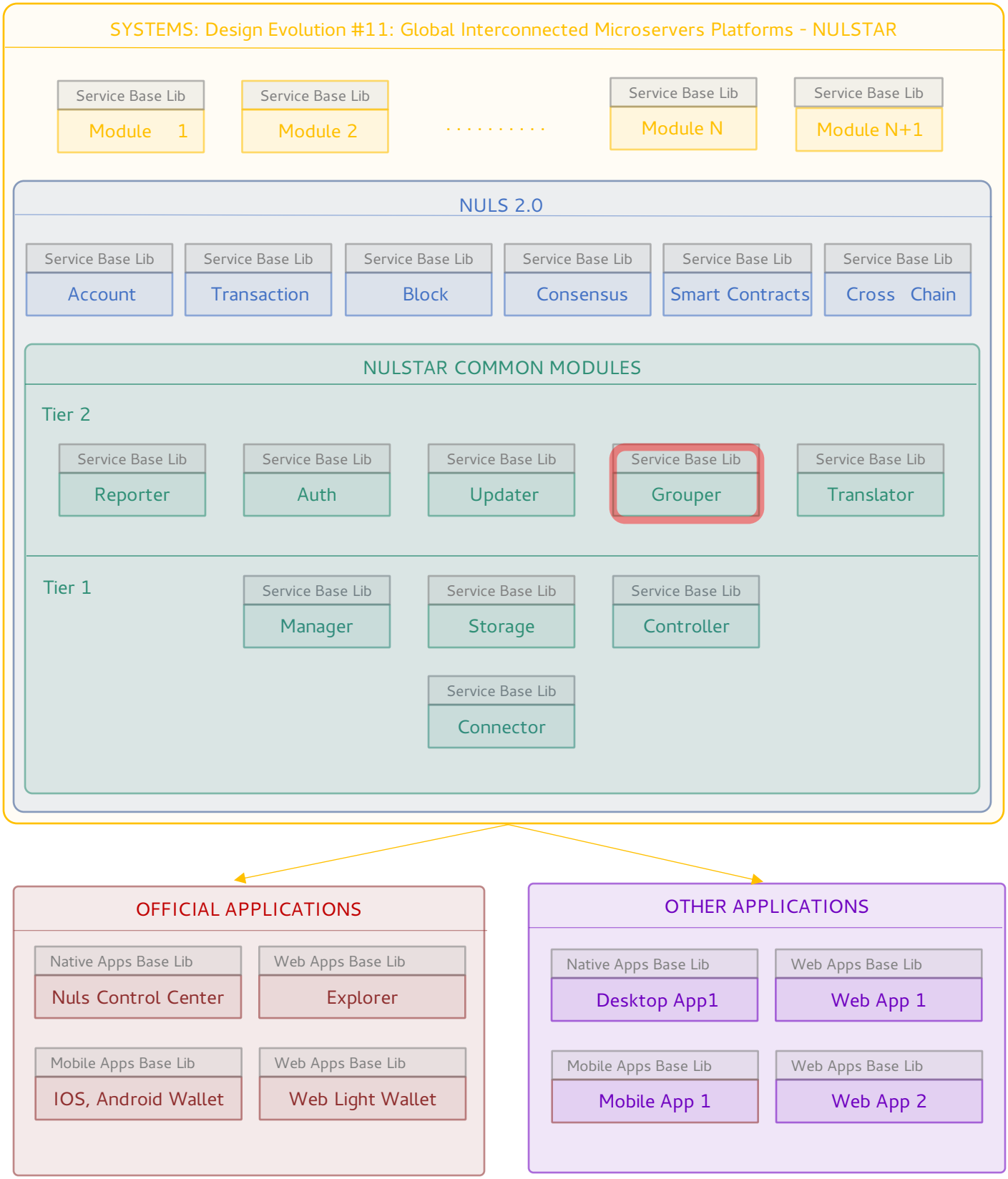


Wouldn't be nice if we could just reuse the information provided from these systems to build new ones? For example, assume we want to create a system like Coin Market Cap where it provides information about all tokens related to NULS, for a basic version we would need the price and the circulating supply, the price could be supplied by an Exchange and the circulating supply from NULS 2.0 interface.

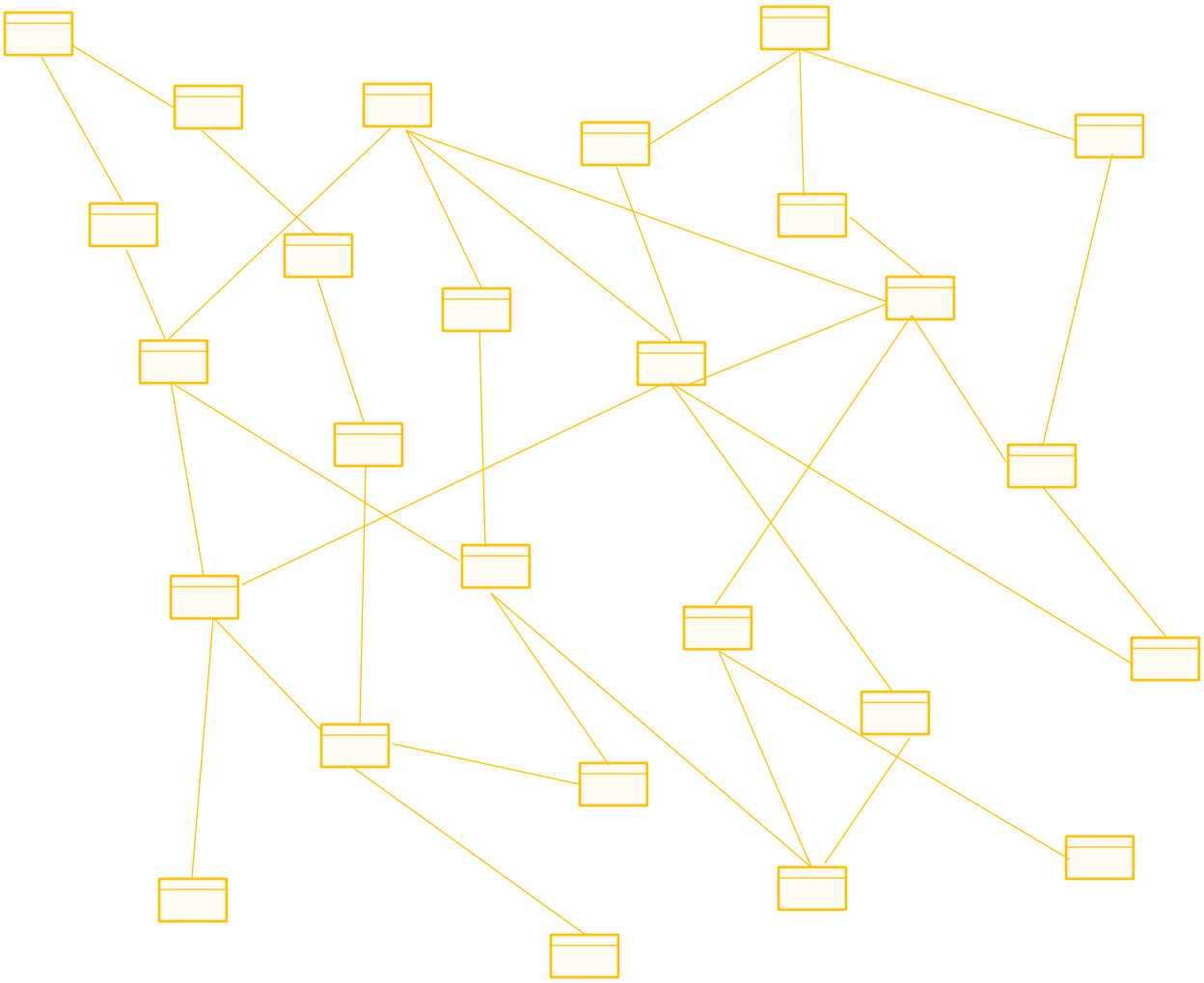
Instead of writing specific code joining these two pieces of information we could just give developers one more gift: The Grouper, this module is in charge to gather information from different NULS 2.0 and present it as if it was generated from our own system, also our system could offer a function as part of its API, let's call it MarketCap that in turn other systems can use it.



The complete platform becomes:



And NULS' ecosystem zoomed out would look like something like this:



A beautiful mesh of fully interconnected systems.

It should be noted that the model presented here is the final vision and not all modules are necessary for the first production version so many will be added in time.

Final remarks

NULS will no longer be just a blockchain node software but a complete platform to build all kind of systems around it (NULS is everything!) that will gladly communicate to each other seamlessly, this will allow business around NULS to flourish a lot faster and at the same time reduce its development costs significantly, representing a very big competitive advantage among our peers.