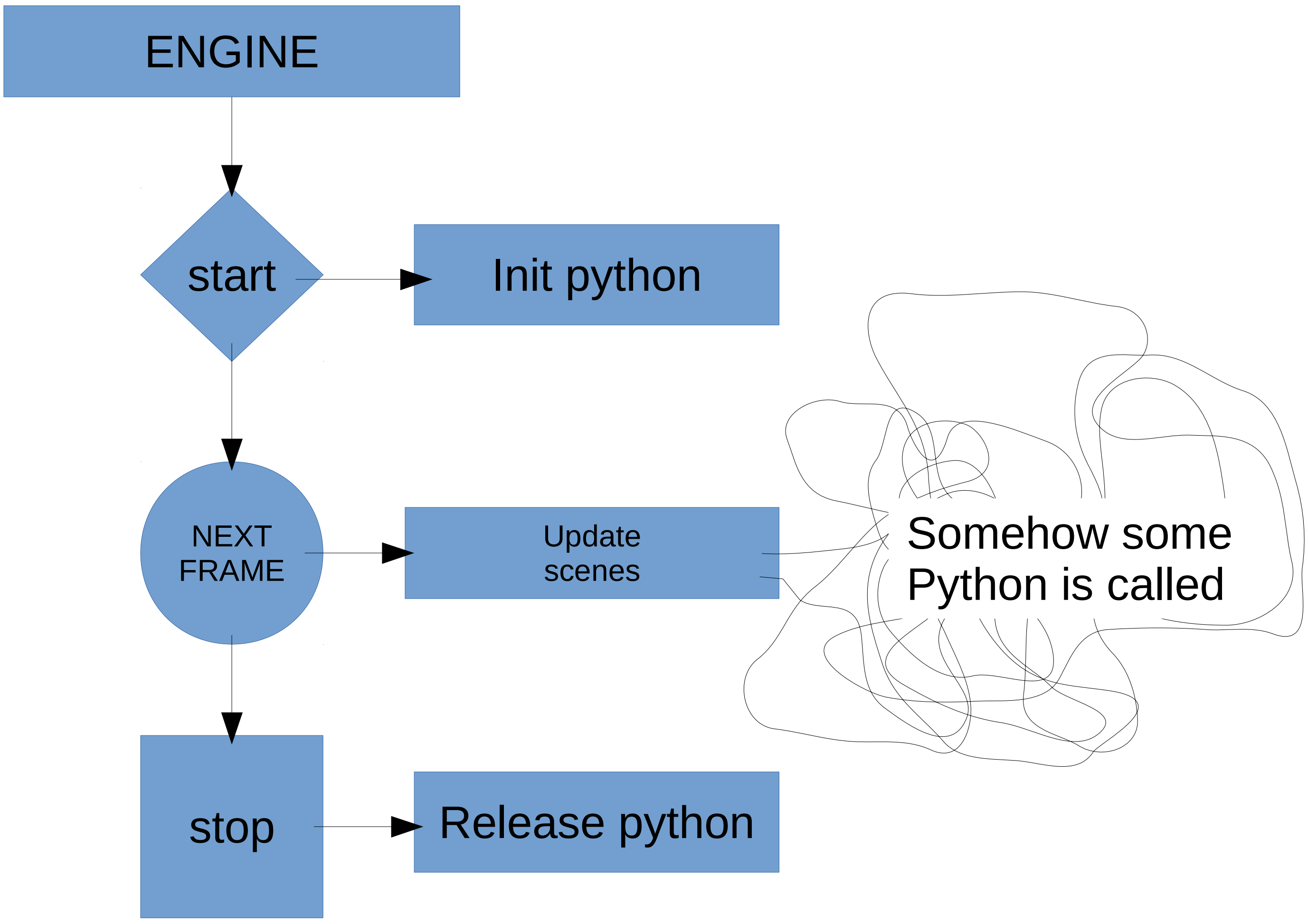
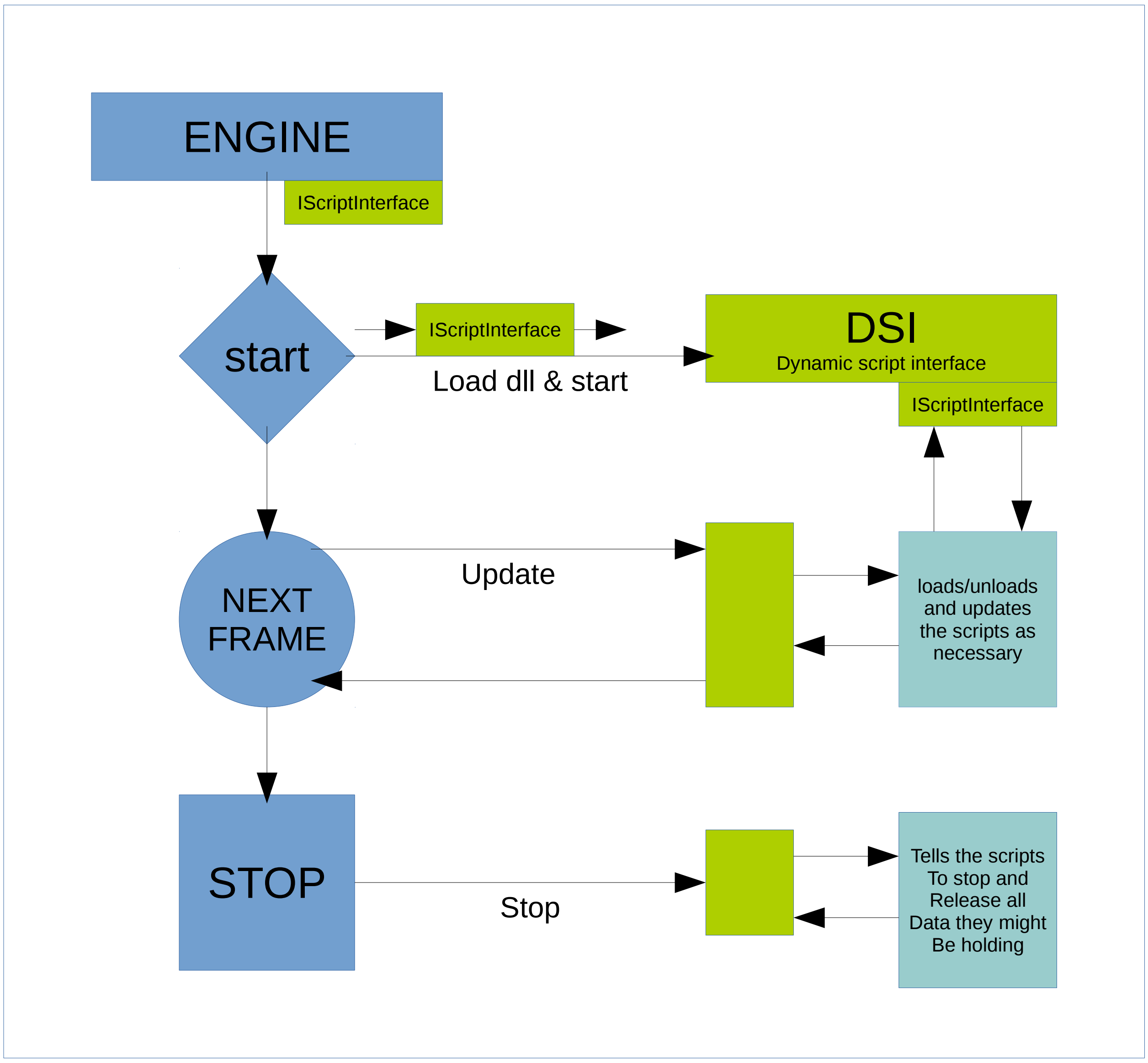
BGE Scripting – How it works now
(a little biased view)



The current interface

The current interface to scripting is distributed over different objects. Each object defines its own set of python bridges to native functions. The overall scripting api is the sum of all these little bits.

BGE Dynamic Script Interface



The script interface is defined in a dynamic library.
The engine loads this library when it starts.
The library exports three functions:

start(IScriptInterface*)
update()
stop()

The IScriptInterface is an abstract C++ class.
That class declares the functions that the engine provides to the scripting world.
The class contains the entire interface available to the scripting interface.
The DSI acts as a bridge between those functions and the scripting runtime.

BGE Dynamic Script Interface – A little detail

The IScriptInterface is a opengl style collection of functions that operates on identifiers.

```
Class IScriptInterface
{
public:
        long findObjectByName(std::string& name);
        void setObjectLocalLinearVelocity(long objectid, float x, float y, float z);
        void getObjectLocalLinearVelocity(long objectid, float* xyzBuffer);
        long getParent(long childId);
        ...a lot of functions...
}
```

The engine will provide an implementation of this class using the available data structures to find objects and the existing set of functions to access the properties of each element.

The dynamic library receives an instance of this IscriptInterface when it is started – by the engine:

```
//bgescripting.so
IScriptInterface* si;
void start(IScriptInterface* scriptInterfaceInstance)
{
        si = scriptInterfaceInstance;
}
```

The update and the stop methods depends on the vm of choice (subliminal...jvm), as well as the actual scripts loading and running. The implementation will expose the functions provided by the IScriptInterface instance, coming from the engine, to the vm languages, according to the specifications of the vm itself.
For the jvm, it means having a set of JNI functions that maps incoming bytecode calls of native methods into the corresponding IScriptInterface functions, translating data formats as needed.

The good

The engine looses its ties to the scripting environment.
The scripting layer can be updated or changed separatedly.
The scripting layer can use a different vm.
Using a proper vm (jvm, mono, **jvm**) allows the definition of
one interface for multiple languages. Did I say jvm? Java, Python,
JavaScript, LUA, Scala and so on. Jvmjvmjvm. It's gpl too.
The new layer can be more consistently defined, documented and
Updated.
The new layer can be added along the old one. The current api
will continue to work as it does now, there is no interference.

The bad

Everyone has a plan 'till they get punched in the mouth.

M.Tyson.