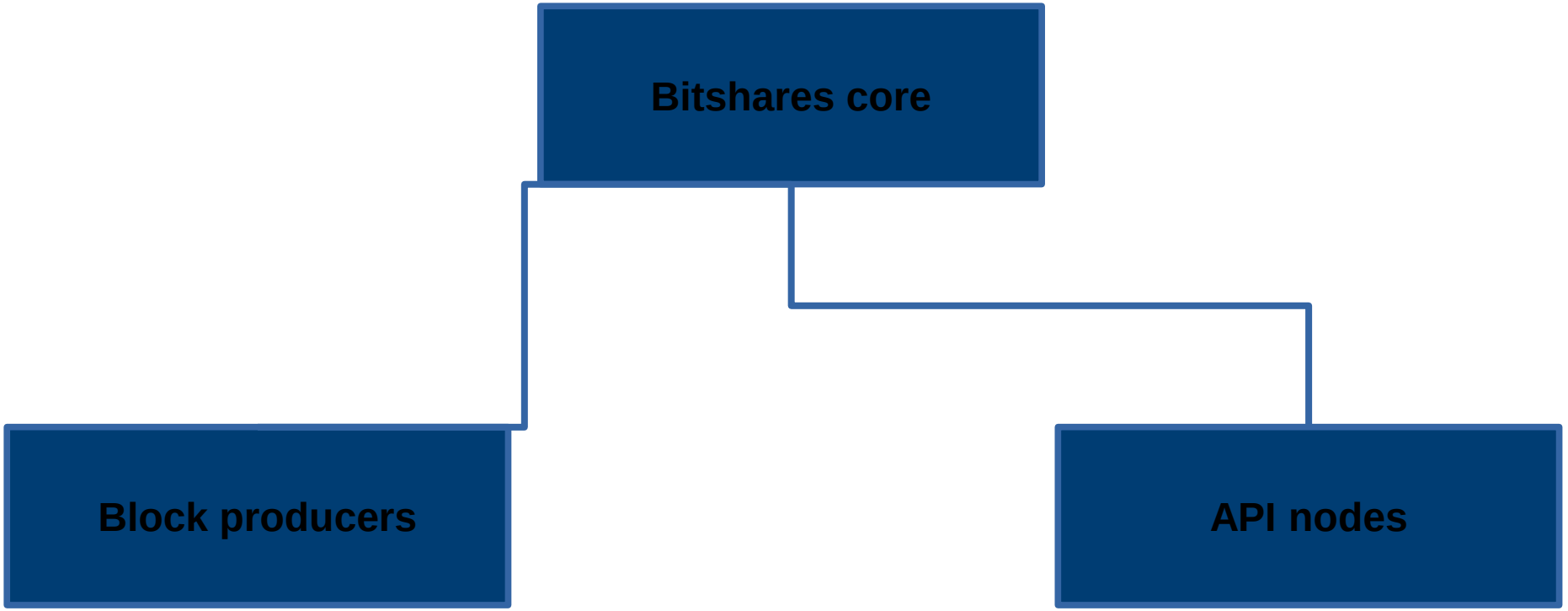
The future of Bitshares plugins

Stability, performance and speed

- Bitshares is stable as it executes only a pre defined set of operations(smart contracts)
- Bitshares is pretty safe from malicious input by the same reason.
- Bitshares is fast as consensus data is stored in RAM.
- Consensus changes are only applied once or twice a year, business and developers need features that can be exposed faster.

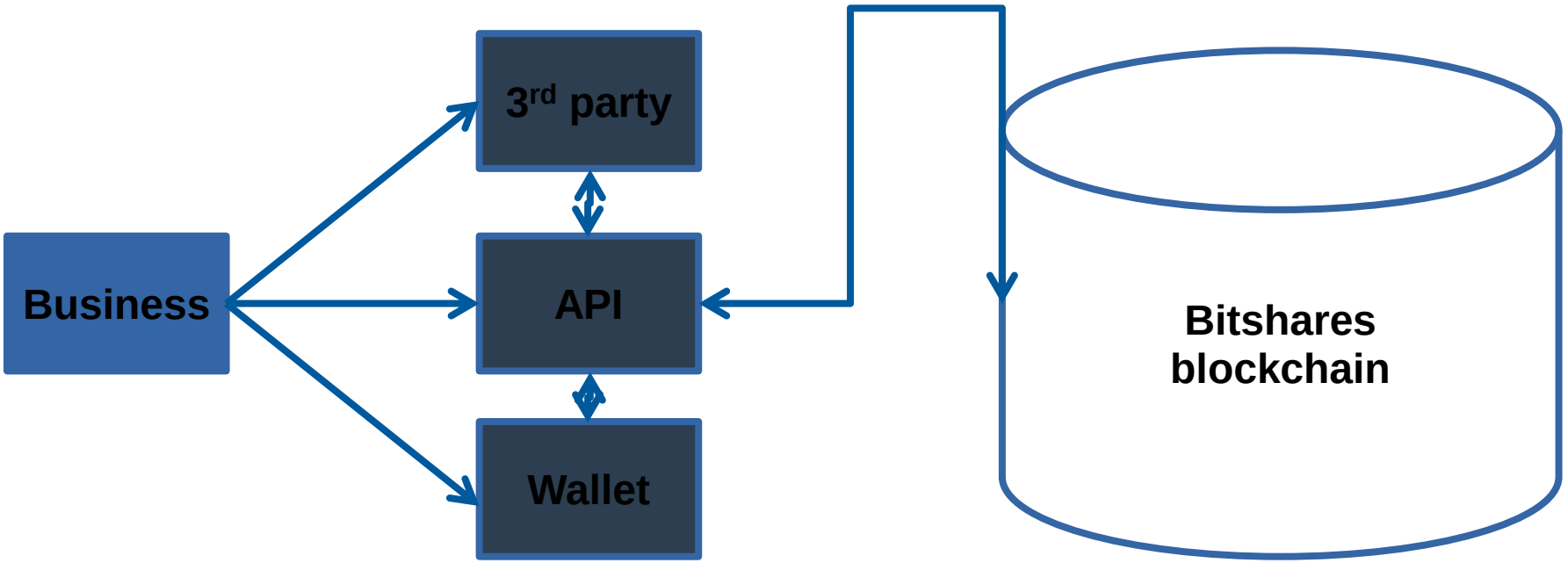
Bitshares core Software



BP sign blocks by plugin

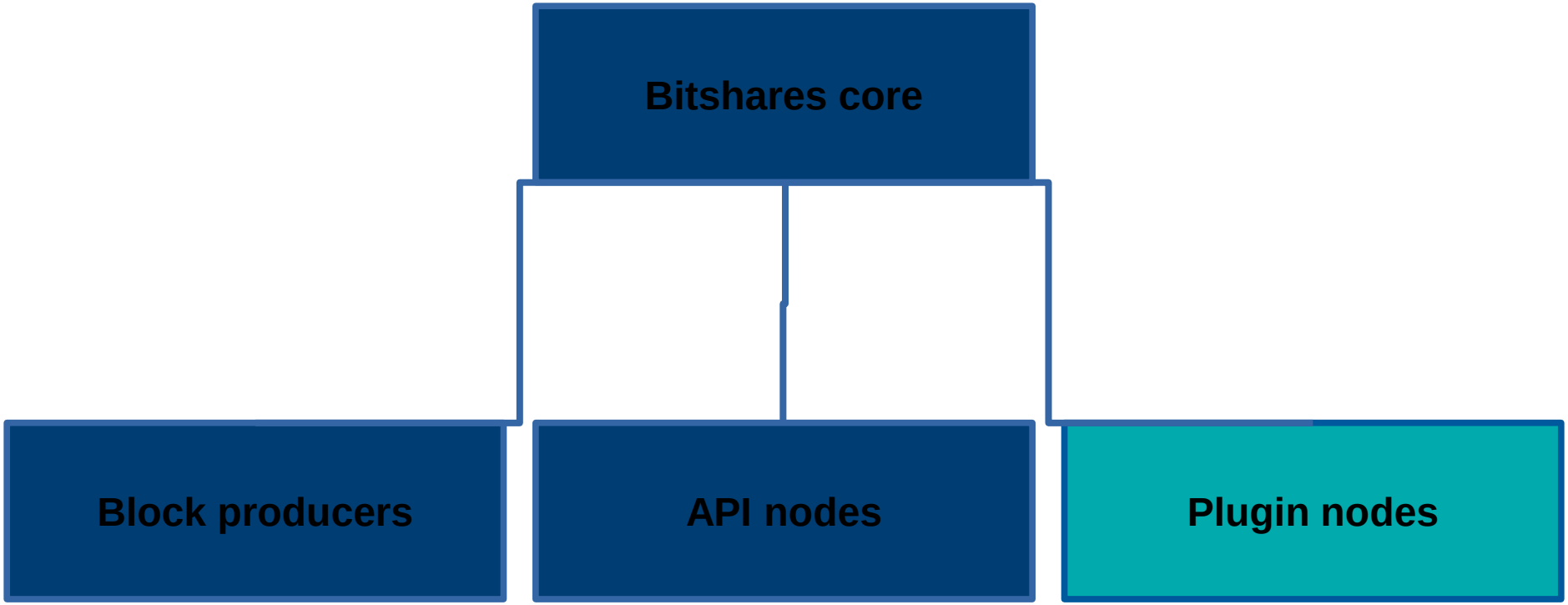
- Bitshares UI
- Command Line Wallet
- Python(python-bitshares, upticks, exp
- bitsharesjs

Business software model

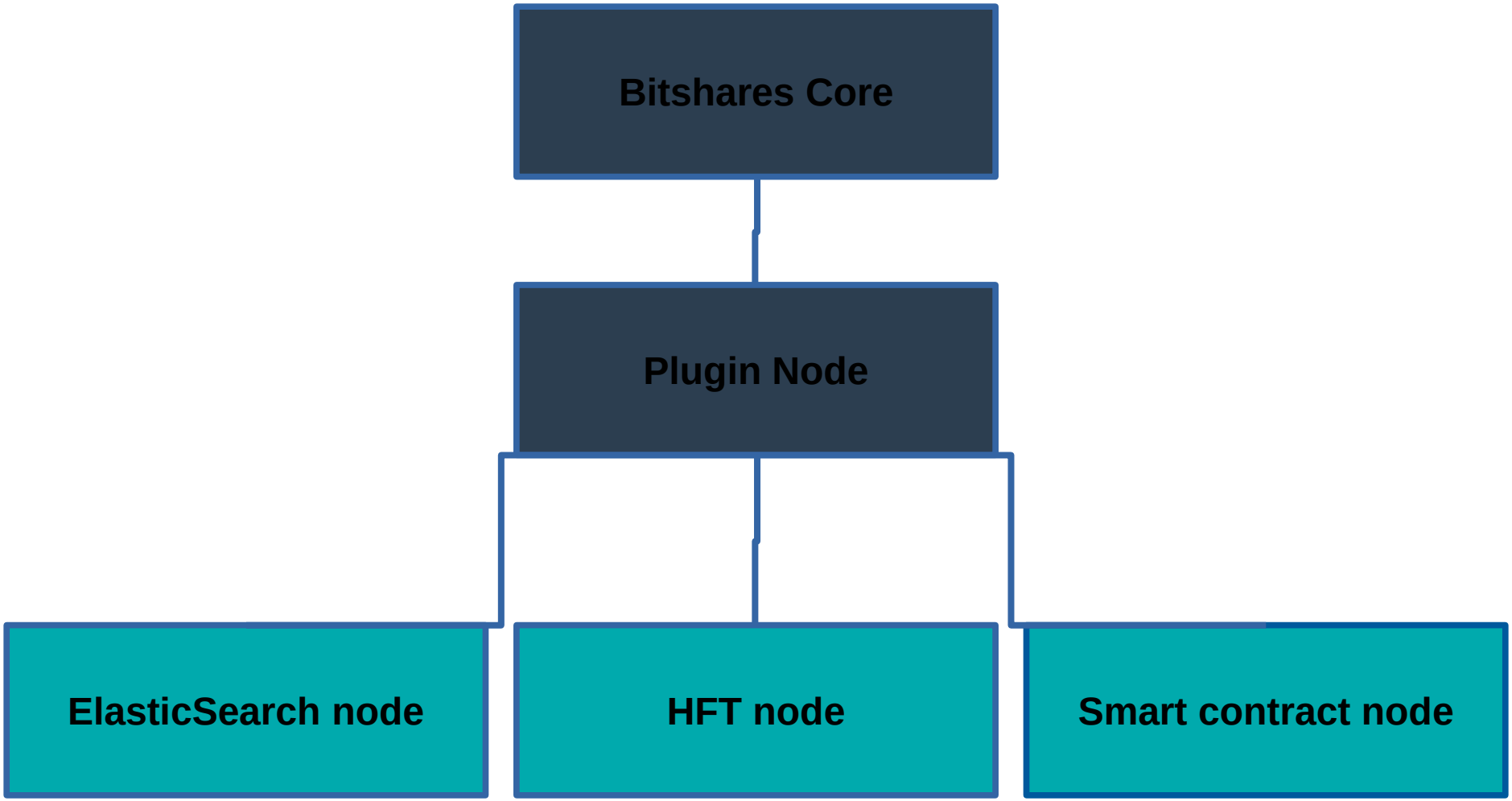


- Integrated API or consensus rules will never have all calls all business will require(perform

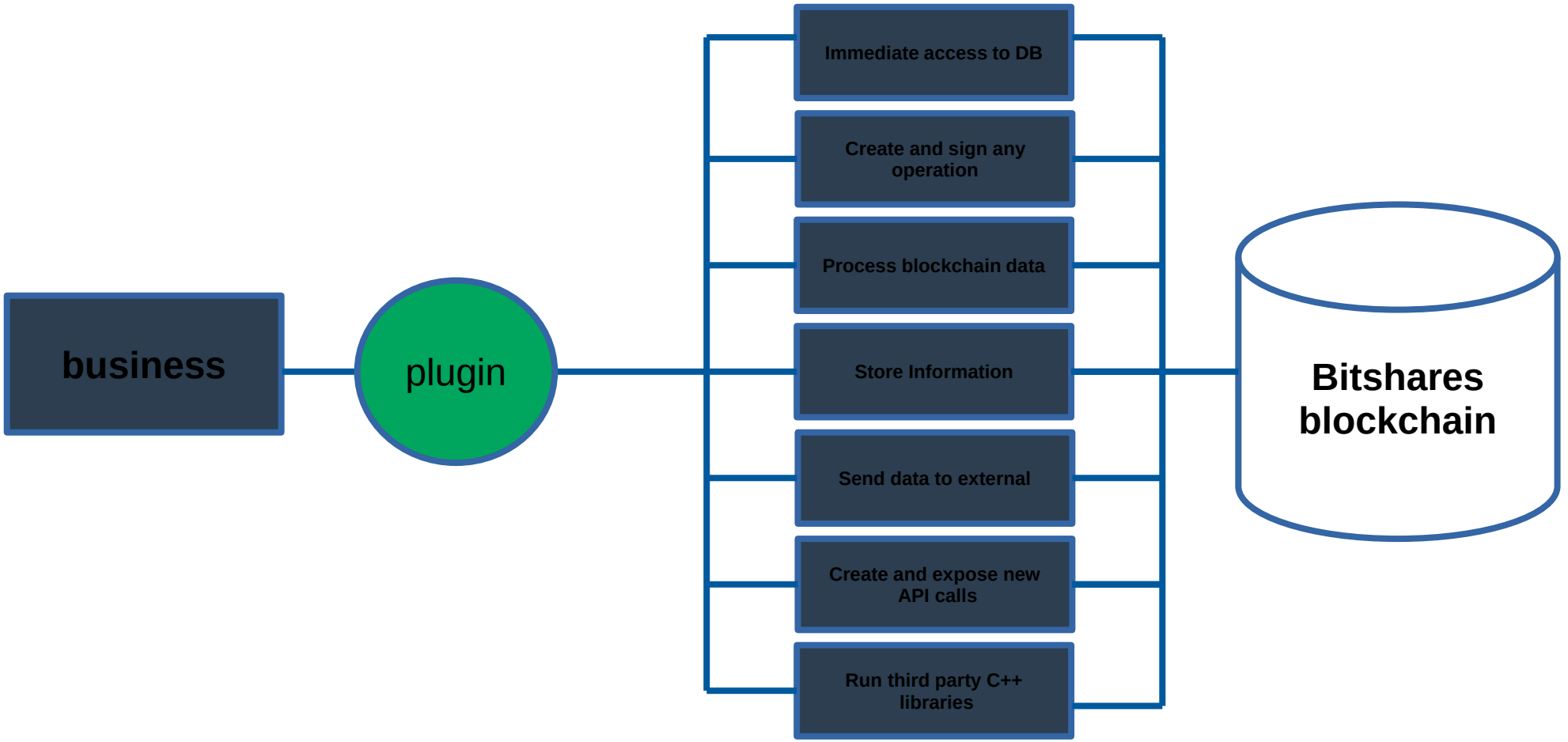
Plugin nodes



Plugin nodes

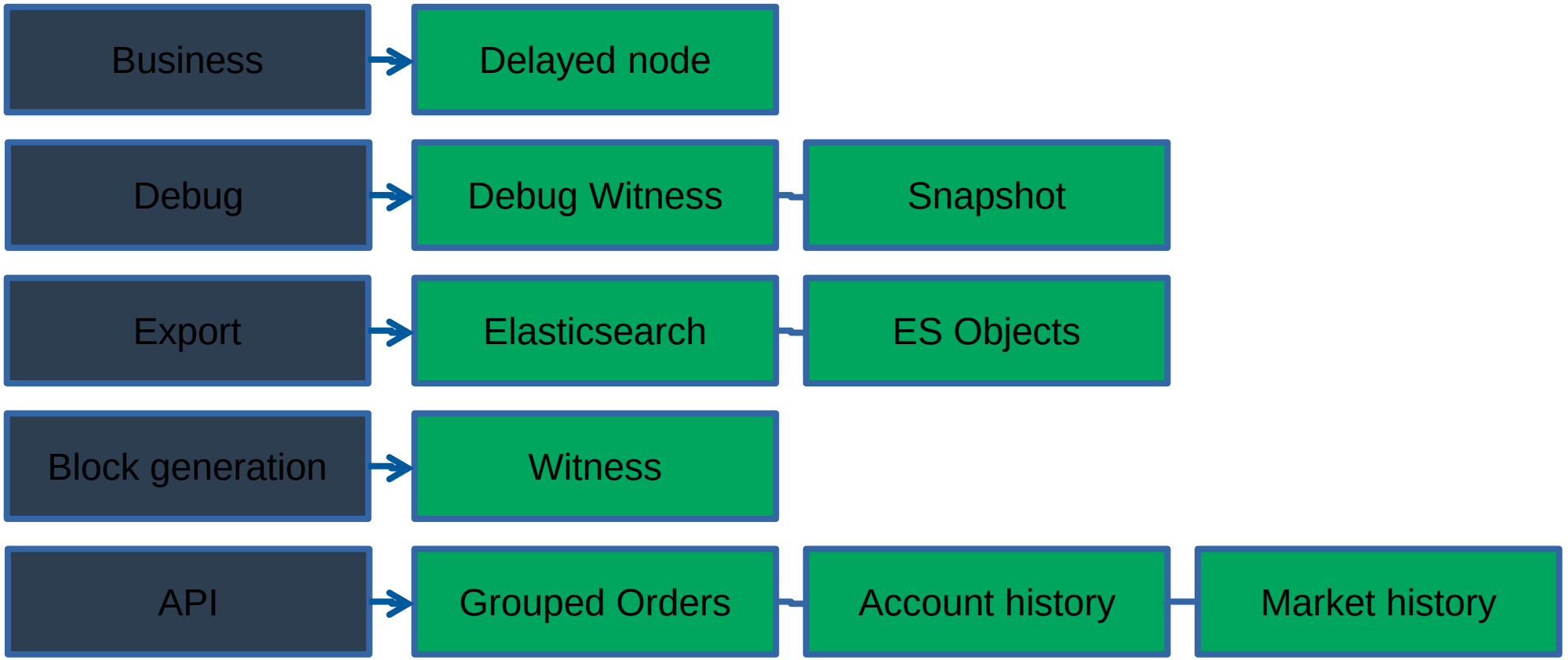


Plugin model



C++ Skills are needed to build plugins.

Current state of plugins

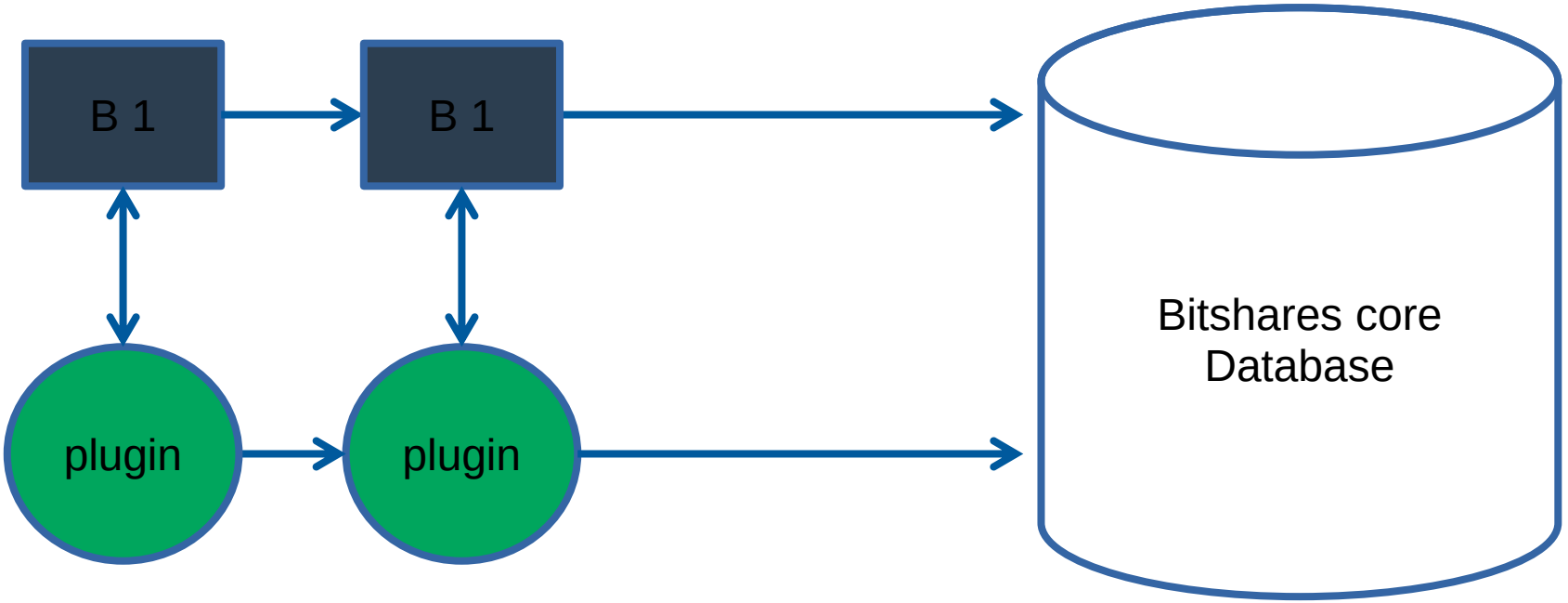


9 plugins total.

Can we do more?

Plugin hooks

Sidechain: on each applied block do something



Have all blockchain data available on each signal event.

Common plugin signals

- Connect to each applied block.
- Connect to each new created object.
- Connect to each modified object.
- Connect to any deleted object.
- Combinations.
- Create new signals.

Connect to each block

.Hello world of plugins.

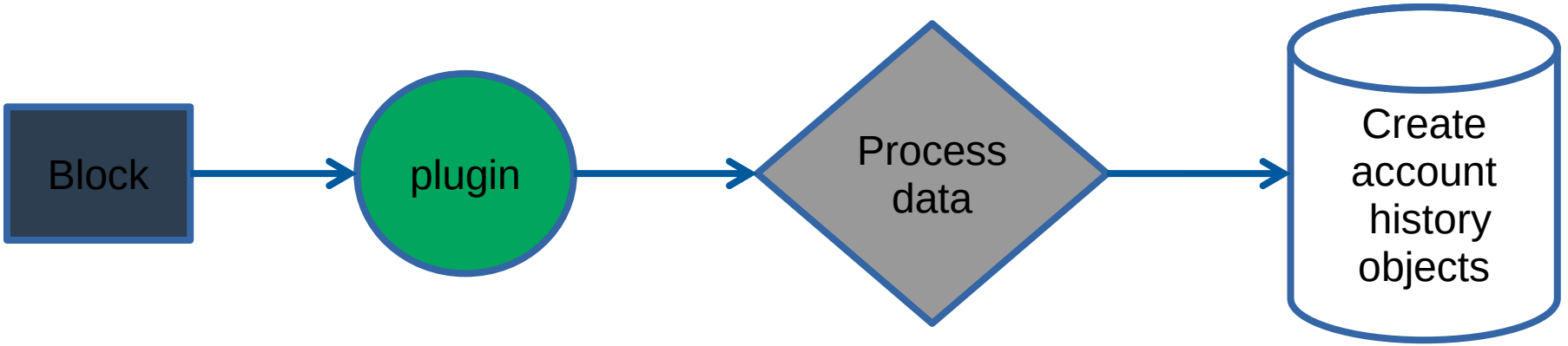
```
void my_plugin::plugin_initialize(const boost::program_options::variables_map& options)

   database().applied_block.connect( [&]( const signed_block& b) {
      my->onBlock(b);
   } );
}



void my_plugin_impl::onBlock( const signed_block& b )
{
   graphene::chain::database& db = database(); // call the database
   auto block_num = b.block_num(); // get current block number
   ilog("Block number: ${b}", ("b", block_num)); // print block number
}
```

Block number: 1
Block number: 2
Block number: 3
Block number: 4
...

Lets see some work done.

Account History plugin

Simplified account history plugin functionality



On each applied block

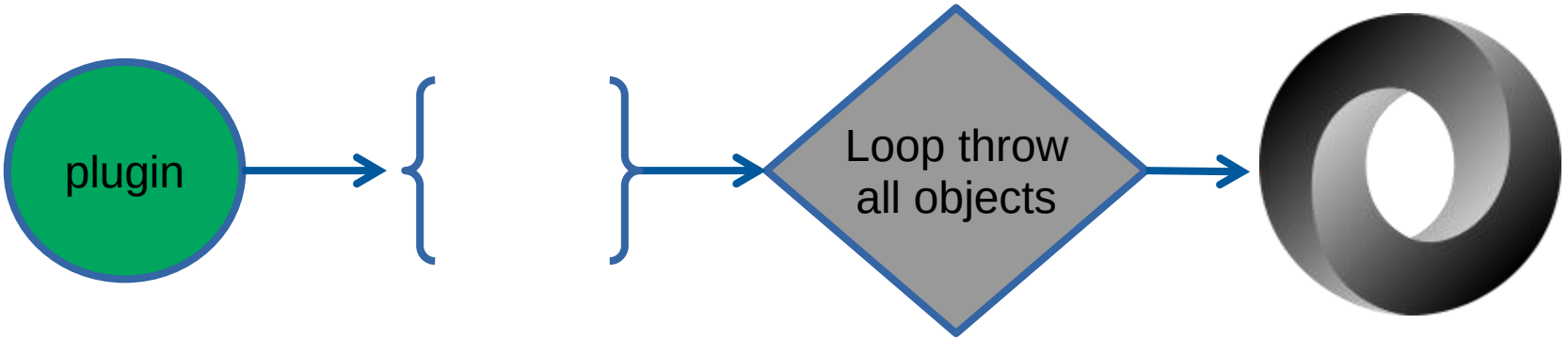
Call plugin update_account_histories

Process operation inside applied block

Data is now available throw
- get_account_history
- get_relative_account_history
- get_account_history_operations
- etc

Snapshot Plugin

Send all objects to JSON at selected block



Plugin is loaded

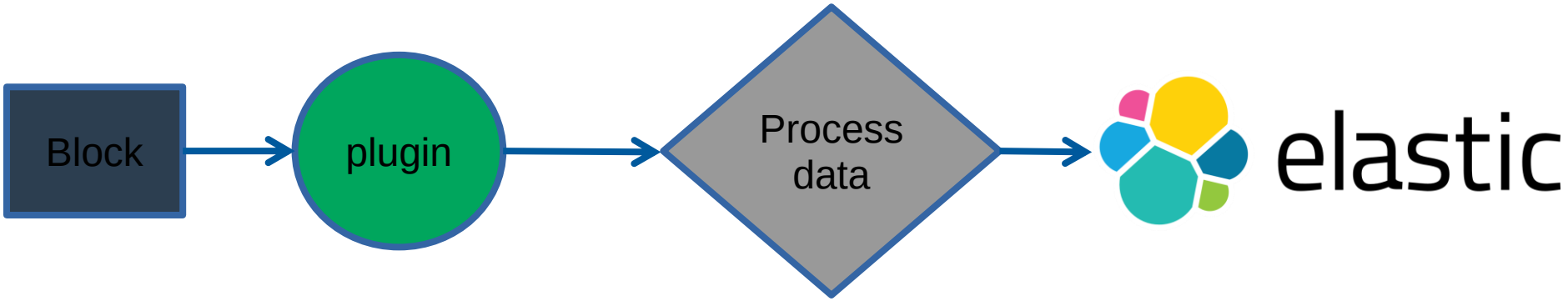
On user selected block or time

Process objects

Export current blockchain to JS

Elasticsearch Plugin

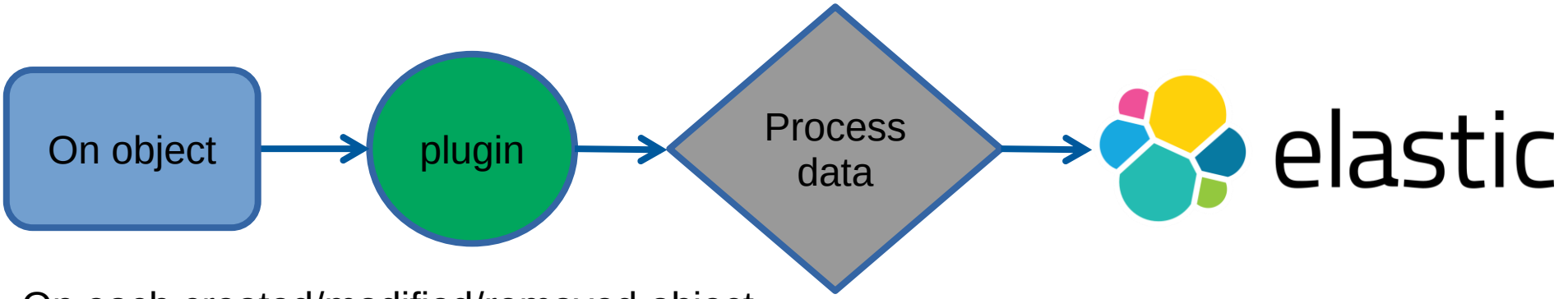
Starting to integrate 3rd party technology



Elasticsearch plugin allows to fast search operation history and decrease hardware requirements to run a full full node.

Elasticsearch Objects

Persistence and easy query

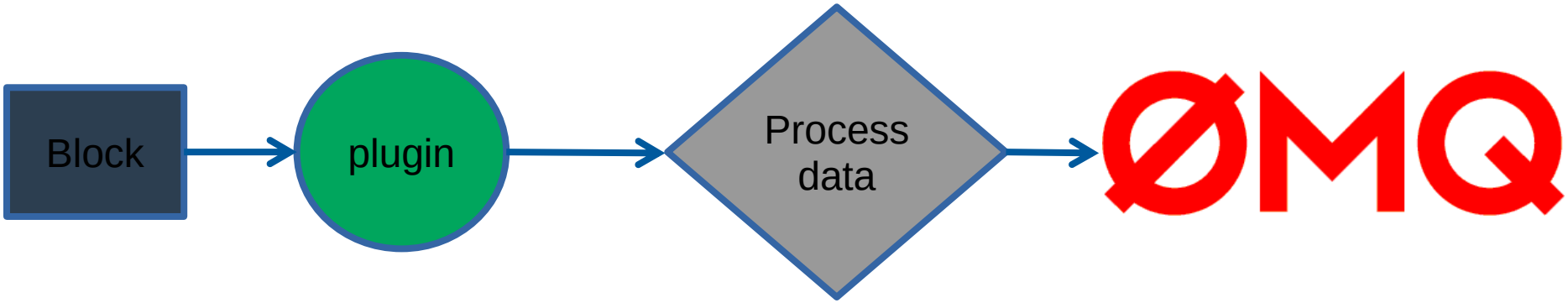


The ES Objects plugin can capture changes in objects that otherwise are lost, for example a

Lets see some work in progress

ZeroMQ Plugin

Send data to socket



On each applied block

Call plugin

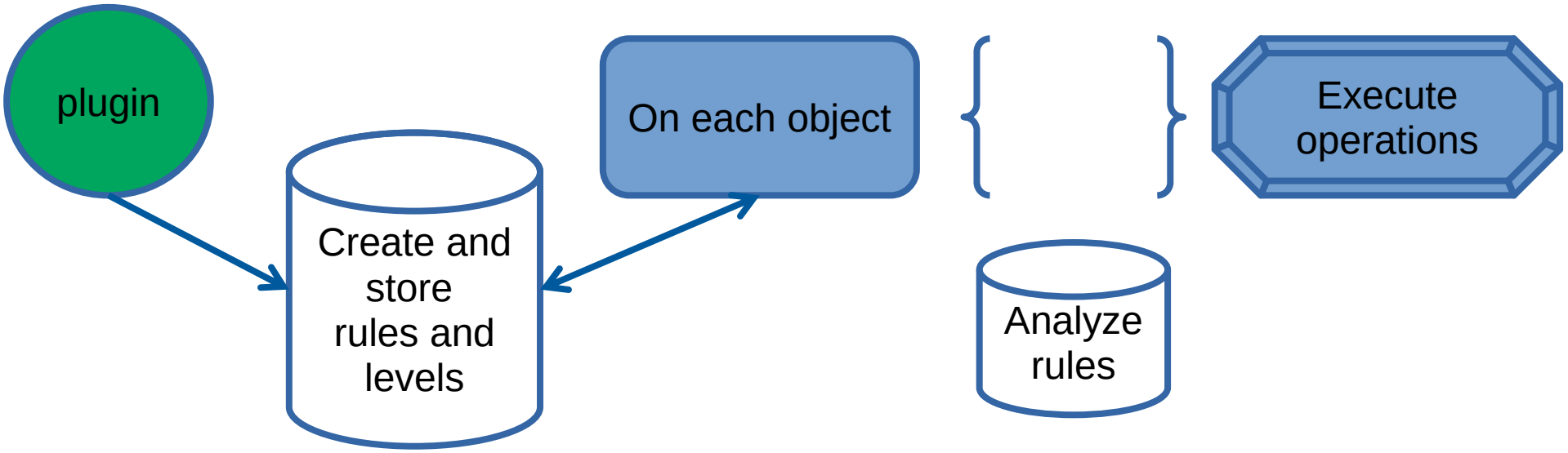
Process operation coming from blockaid

Data is now available in tcp s

Plugin acts a server, client will be listening and receiving operations from the plugin.

High Frequency trader

Creating and signing operations from plugin



- Plugin must have private key of the trader.
- When operations are executed from inside plugins the normal fees are applied to accounts ir

Stoploss Plugin

- **A possible set of rules will be:**

  If price of BTS vs CNY drops below my predefined level:

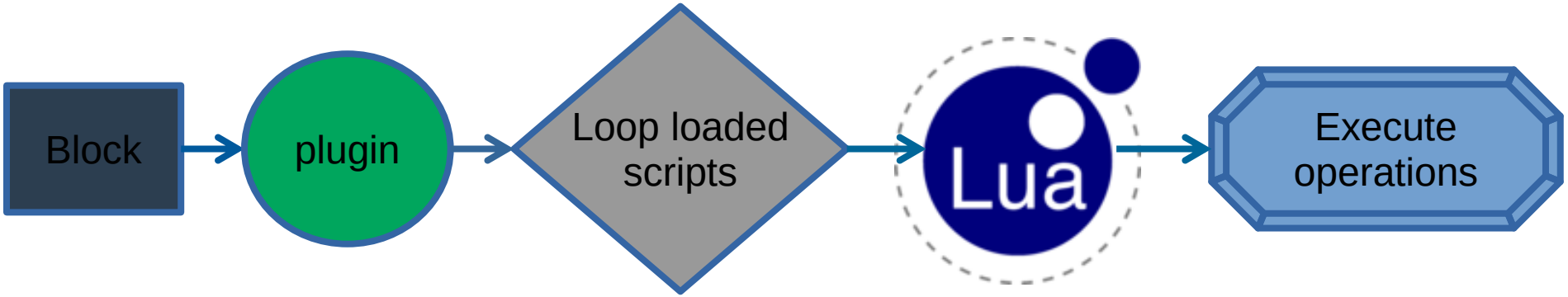
  Buy CNY and stop loss.

  If price of BTS vs CNY is above my predefined level:

  Buy CNY and take profits.

Lua Scripting and Virtual Machine

Lua: execute user loaded simple scripts



Scripts can be loaded throw cli_wallet or UI.

A Lua VM is created with each script on each applied block.
Operations are executed as they are found inside script.

C++ skills are NOT needed to build Lua scripts.

```
transferDate = "2019-02-01"
user_account = "bob"
block_time = Bitshares:getCurrentBlockTime()
if block_time > transferDate then

    Bitshares:transfer("my-account", user_account, "100", "BTS")

    Bitshares:quit()
end
```

Plugins as a service

I want to offer my clients the possibility to run Lua scripts

Node with Lua plugin

- Client create lua script and upload to Lua plugin node by cli_wallet(or UI).
- Script will be executed every block until quit() is found on script or if script expires.
- Client need to send private key to Lua plugin node at loading script.

The private key issue

- BSIP 40: Custom Authorities will reduce the impact of the private key stored in plugin node.
- Some specific use cases can be done by executing proposals behind the scene. This removes the need to send any key to plugin server.
- A HF with 1 or more new operations can be added to consensus.

Other issues

- Centralization: If plugin node gets down for any reason operations will not be executed. Possible solution: Distribute plugin nodes.
- Resources: Plugin without making any operation can consume too much computation power in the node. Possible solution: Get the cycles each script consumes on every run and charge for running in a GAS style.