Lua background worker lib

Multithreaded background comm lib
for single threaded Lua state

Issue

- A Lua state is a single threaded block of C code
- When using standard libraries which use multiple threads there are thread synchronization issues

- Try to create a single reusable solution for the synchronization problem

Helper lib

- Background threads deliver results to the helper
- Helper stores results in a queue
- Helper sends a UDP packet to the designated port to wake-up Lua from a `select()` statement (filehandles may be used if supported?)
- The Lua side of the helper collects the information from the queue by calling the `poll()` method when its coroutine returns from the `select()` statement
- The `poll()` method of the helper calls the originating library to decode the content and deliver it to the Lua state
- When the `poll()` method returns, the Lua side of the helper calls the callback method with the additional returned parameters

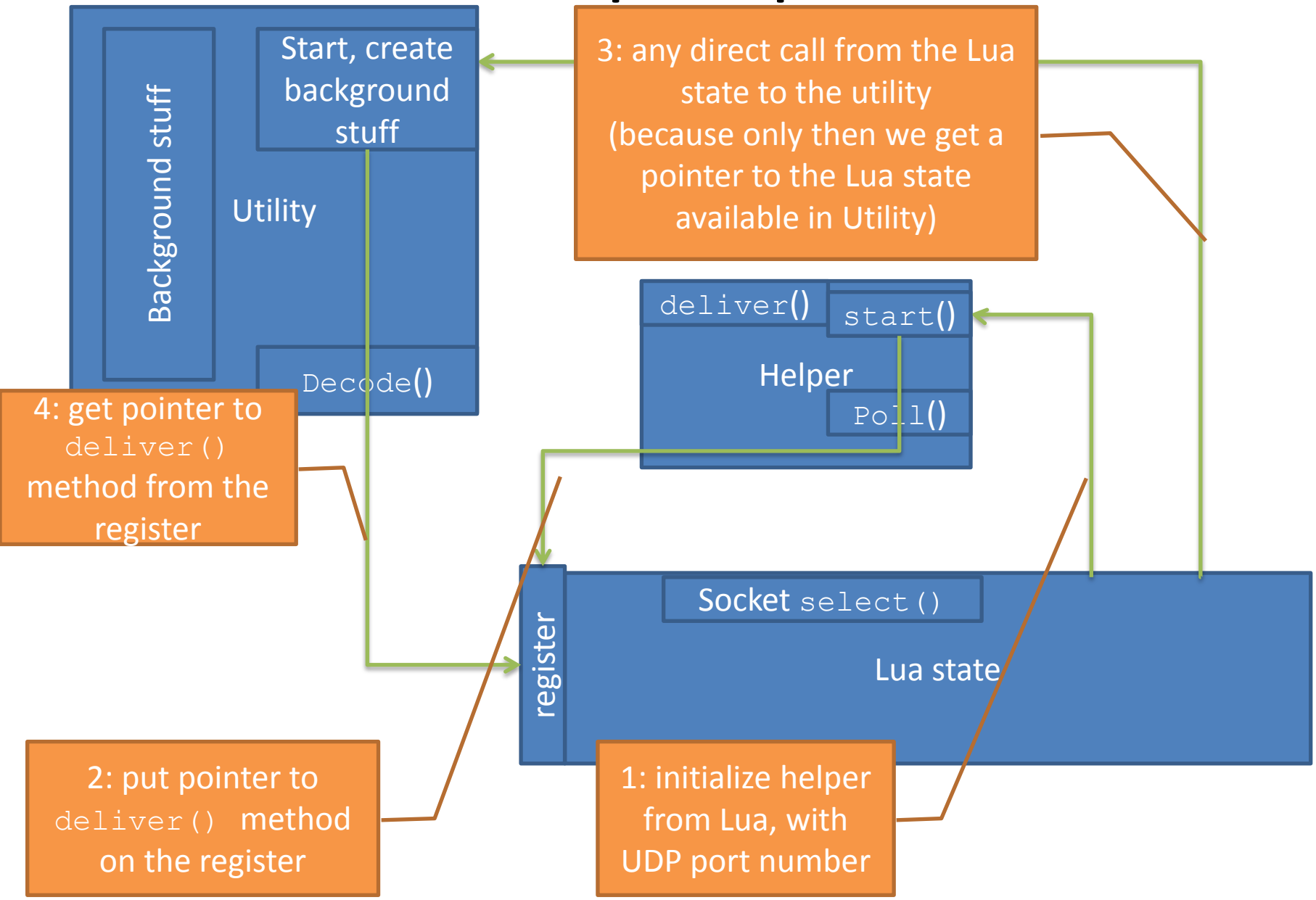
Definition

- Helper lib: the synchronization helper library
- Utility lib: any external library that uses the Helper lib to get its results delivered to the Lua state.

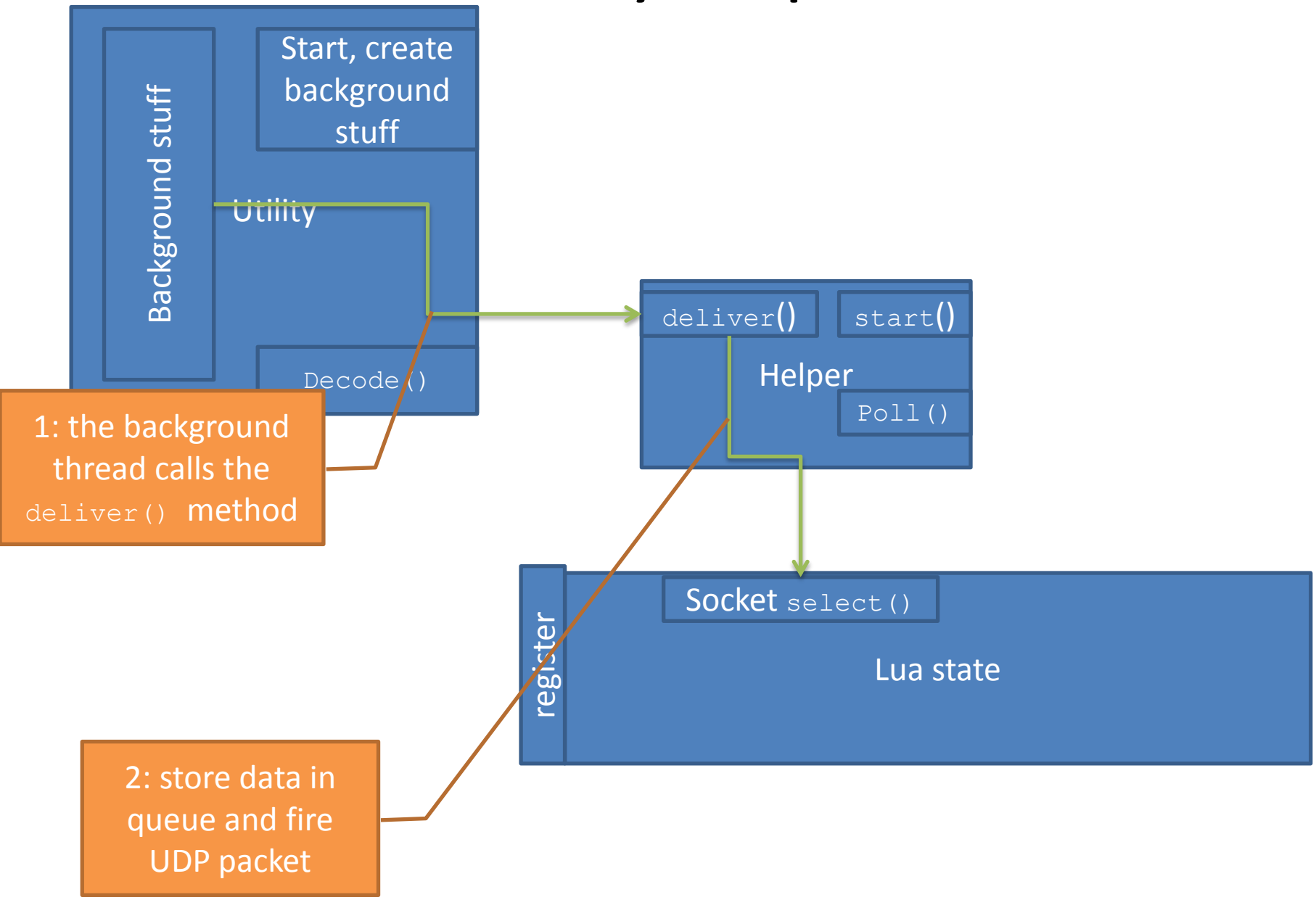
Key elements

- The helper lib puts a pointer to its deliver() method on the Lua register, so the utility lib knows what function to call
- The deliver() method is thread safe and accepts 3 arguments;
  - A pointer to the actual data
  - A pointer to the decode() function to decode the data
- The utility lib must be provided with a Lua function to use as 1st result by the decode() method

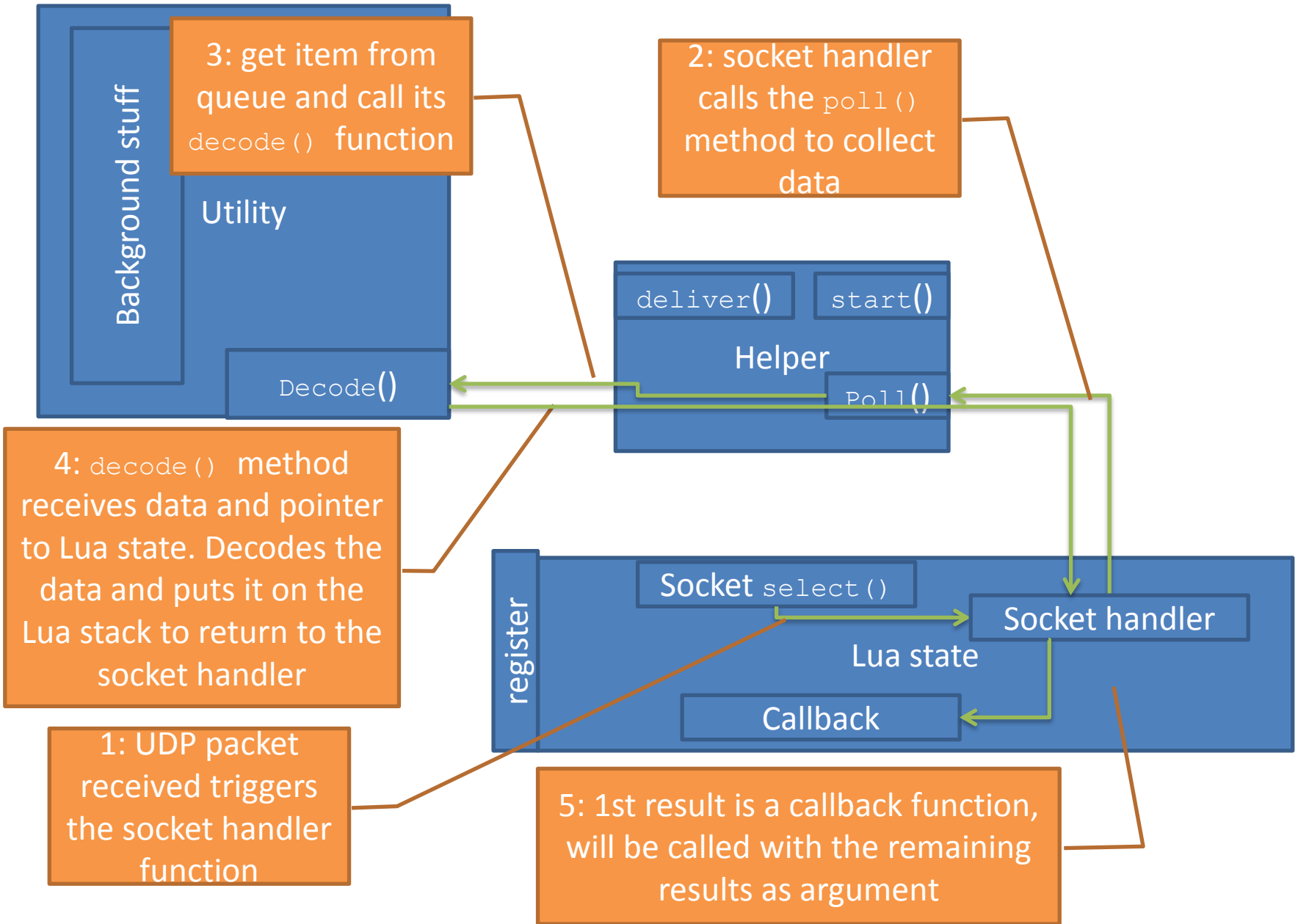
Startup sequence



Delivery sequence



Collect sequence



Functions

- Helper
  - Queue: the helper should maintain a queue with data delivered by different utility libraries background threads. This queue should have a lock to be threadsafe.
  - deliver(*decode(), *data)
    - Use: accepts data from the utility background thread and stores it until the Lua state calls poll() to collect it. Upon receiving data, the helper will fire a UDP packet on the designated port of the 'localhost' adapter.
      This function should probably have a lock to be threadsafe.
    - *decode(): a pointer to the decode() function that is capable of decodeing this block of data. A utility library may have multiple decode() functions for different types of data. The requirement is that de pointer to the decode function must match the data, it must be able to decode it.
    - *data: a pointer to a block of undefined memory containing the data the utility lib needs to deliver to the Lua state. The utility should allocate the memory, but control is handed over to the helper. Once the call to deliver() returns the memory should no longer be considered safe to access. The helper will free the memory after delivery.
  - start(*lua_State, int udpPort)
    - Initializes the helper library. It must place a pointer to its deliver() function into the Lua register, so the utility function have an entry point where to deliver their data.
    - *lua_State: pointer to the Lua state.
    - udpPort: the port to use to signal data has arrived. Whenever data is received (see deliver() function of helper) a UDP packet should be fired on this UDP port to signal the Lua state to go collect the data by calling the poll() function.
  - poll(*lua_State)
    - Use: used to collect data from the queue of the helper. The function should get the next elelment from the queue. This element contains a pointer to the decode() function and a pointer to the accompanying data. Next it should call the decode() function, with the data to decode it. Then it should free the memory of the data block (once decoded) and return control to the Lua state, the decode() function will have decoded the data and put all the return values on the stack, which will be the return values delivered to Lua.
- Utility
  - decode(*lua_State, *data)
    - Use: decodes the data received into Lua variable values and puts them on the stack, so they can be returned to the Lua state as function results (remember: lua functions can return more than 1 result value!). The values returned are defined by the utility library, except for the 1st return value, which must always be a Lua function to be called with the other return values as arguments
    - *lua_State: when poll() is called from Lua it receives a reference to the Lua state, it will pass this on when calling decode()
    - *data: an pointer to an undefined block of memory, earlier delivered by utility to the deliver() method. Helper is owner of the memory and will free it, and once decode() returns the memory should be considered freed.