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. Most notably async calls that generate callbacks on separate threads

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

Definition

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

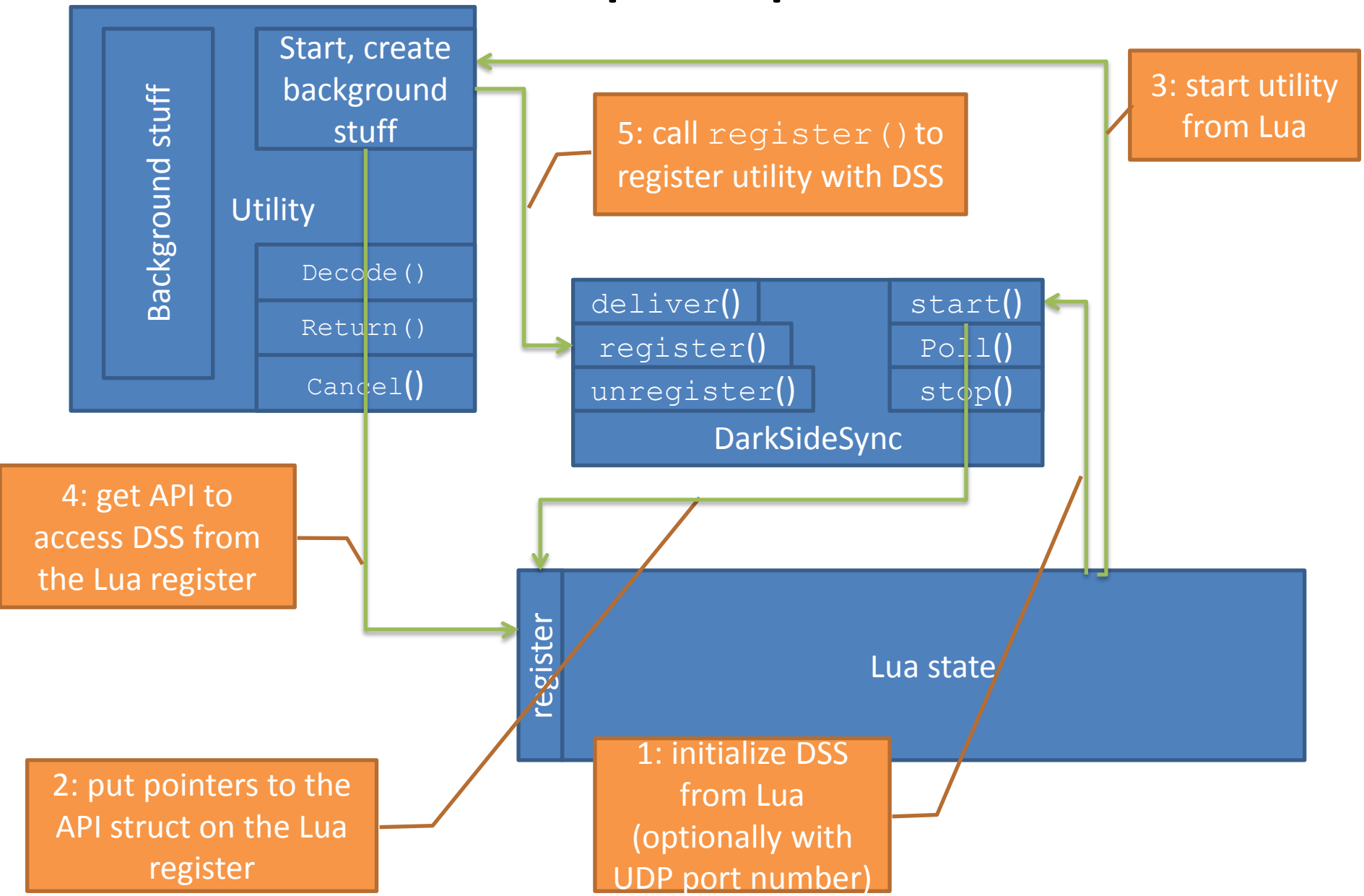
Key elements/design principles

- DSS puts a pointer to a struct containing its API (a.o. `deliver()`, `register()` and `unregister()` methods) on the Lua register, so the utility lib knows what methods to call
- Utility libs must call `register()` before delivering anything and `unregister()` when done.
- DSS lib supports multiple utility libraries
- No locks, queues and synchronization should be required in the utility binding library, all in DSS
- No external threads allowed in Lua state, Lua thread calls out to external libs
- All calls thread-safe so async access is always safe
- Initial data (=pData) will travel through all stages (`deliver()`, `decode()`, `return()`) and after release of the calling thread it will hold the results

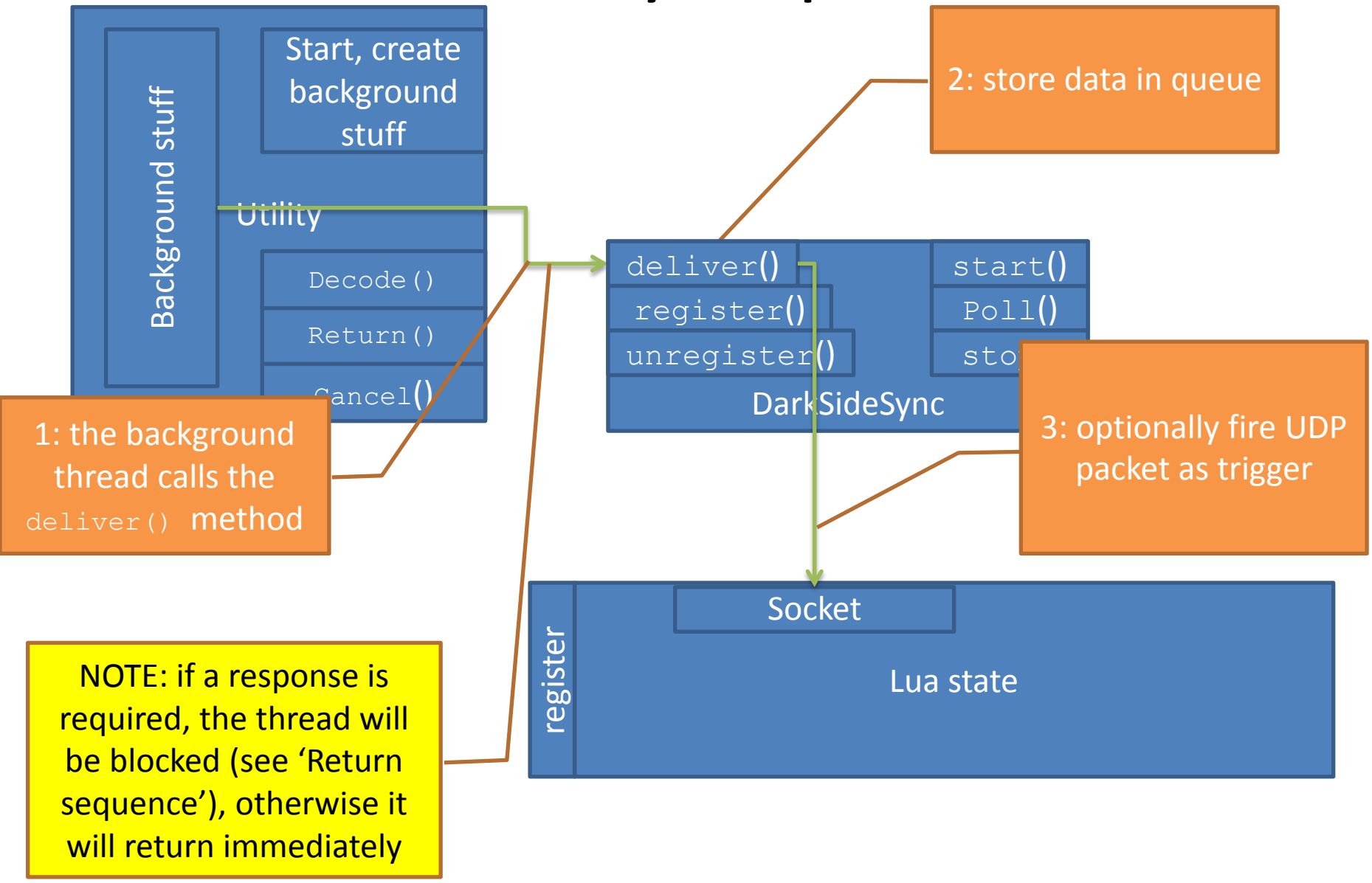
DSS library

1. Background threads deliver results (= pData) to DSS
2. DSS stores results in a queue
3. DSS (optionally) sends a UDP packet to the designated port as a trigger
4. The Lua side of DSS collects the information from the queue by calling the `poll()` method (directly loop-driven or when its coroutine returns from the `select()` statement)
5. The `poll()` method of DSS calls the originating library to `decode()` the content (= pData) and deliver it on the Lua stack
6. The `decode()` method will return a Lua callback and the parameters for calling it (if no response is required, then it is done now)
7. if a response is required the thread remains blocked until Lua calls the `setresult()` method
8. The arguments supplied to `waitingthread_callback()` will be forwarded in a call to `return()` on the utility library
9. The `return()` method will take the arguments on the Lua stack and rework them for the utility (= pData). When `return()` returns, the blocked thread will be released and can access the results (= pData)

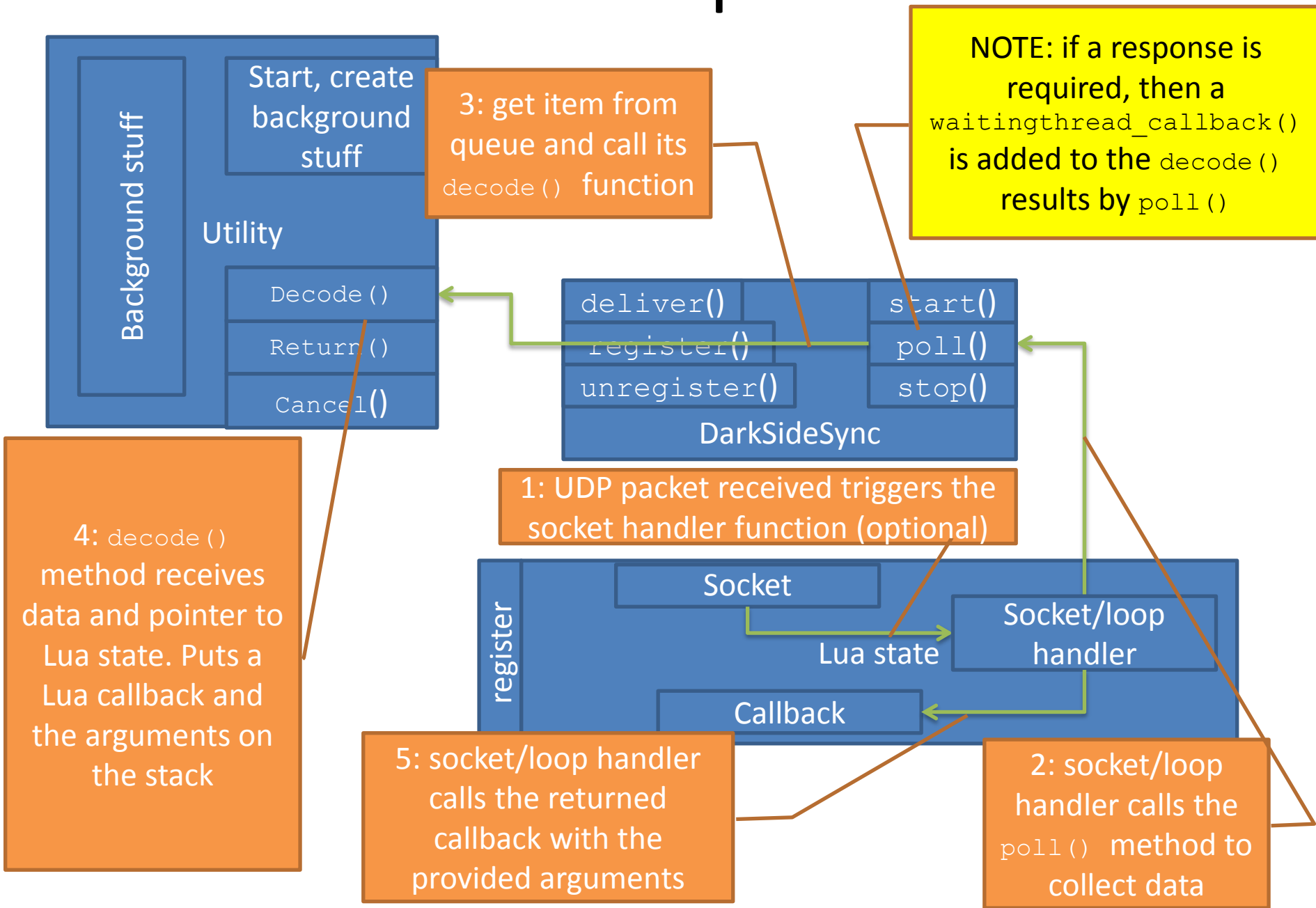
Startup sequence



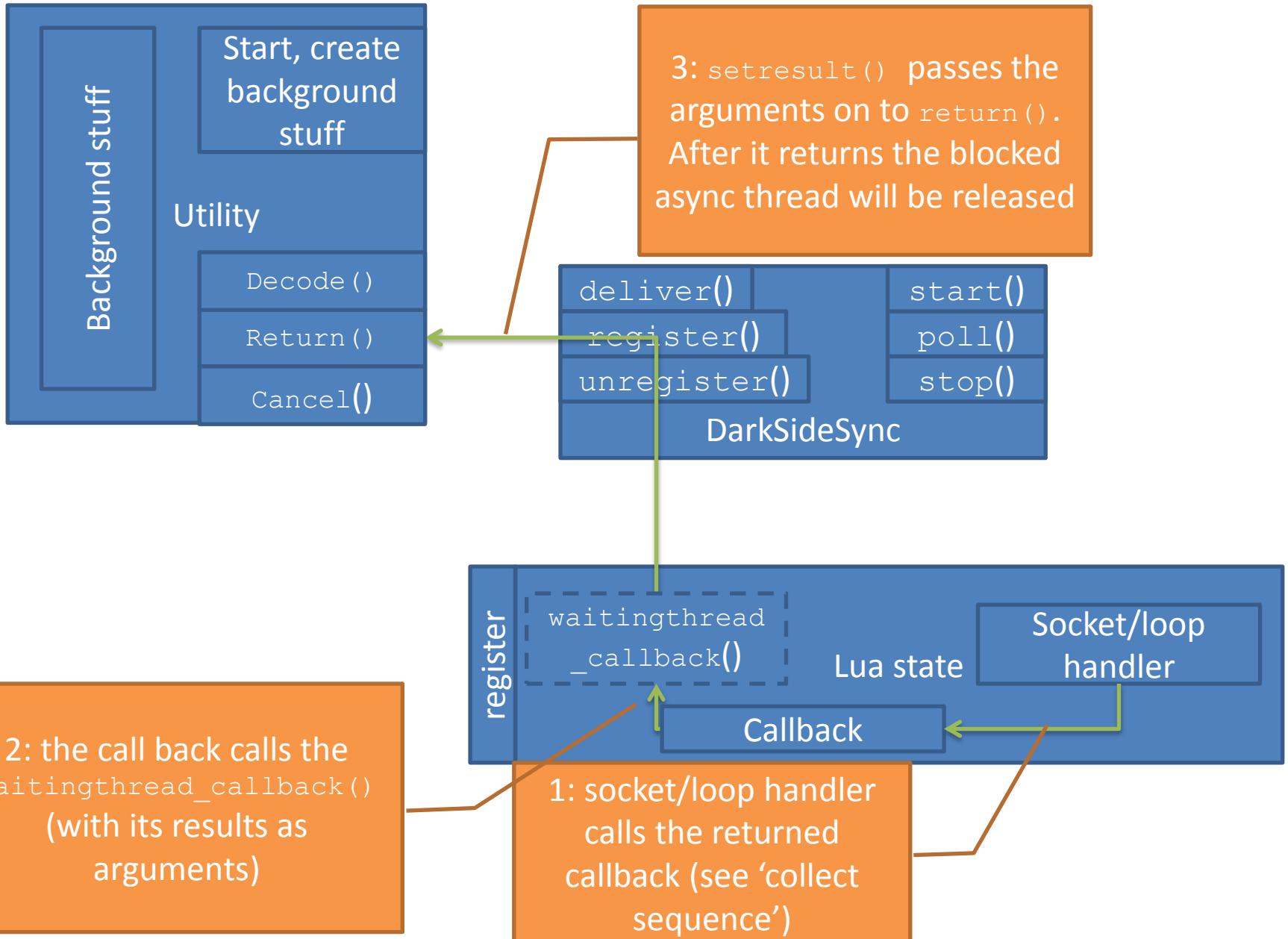
Delivery sequence



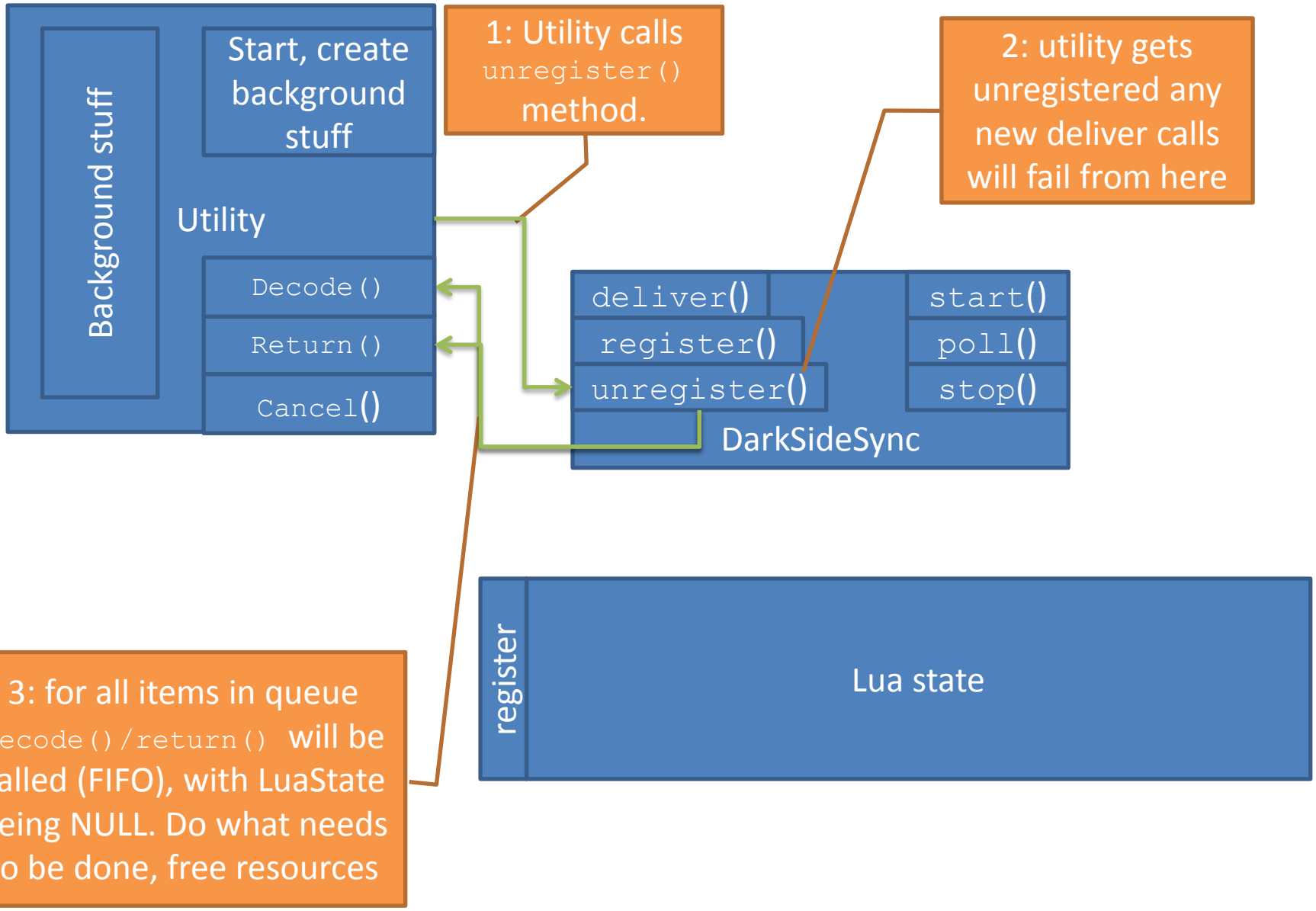
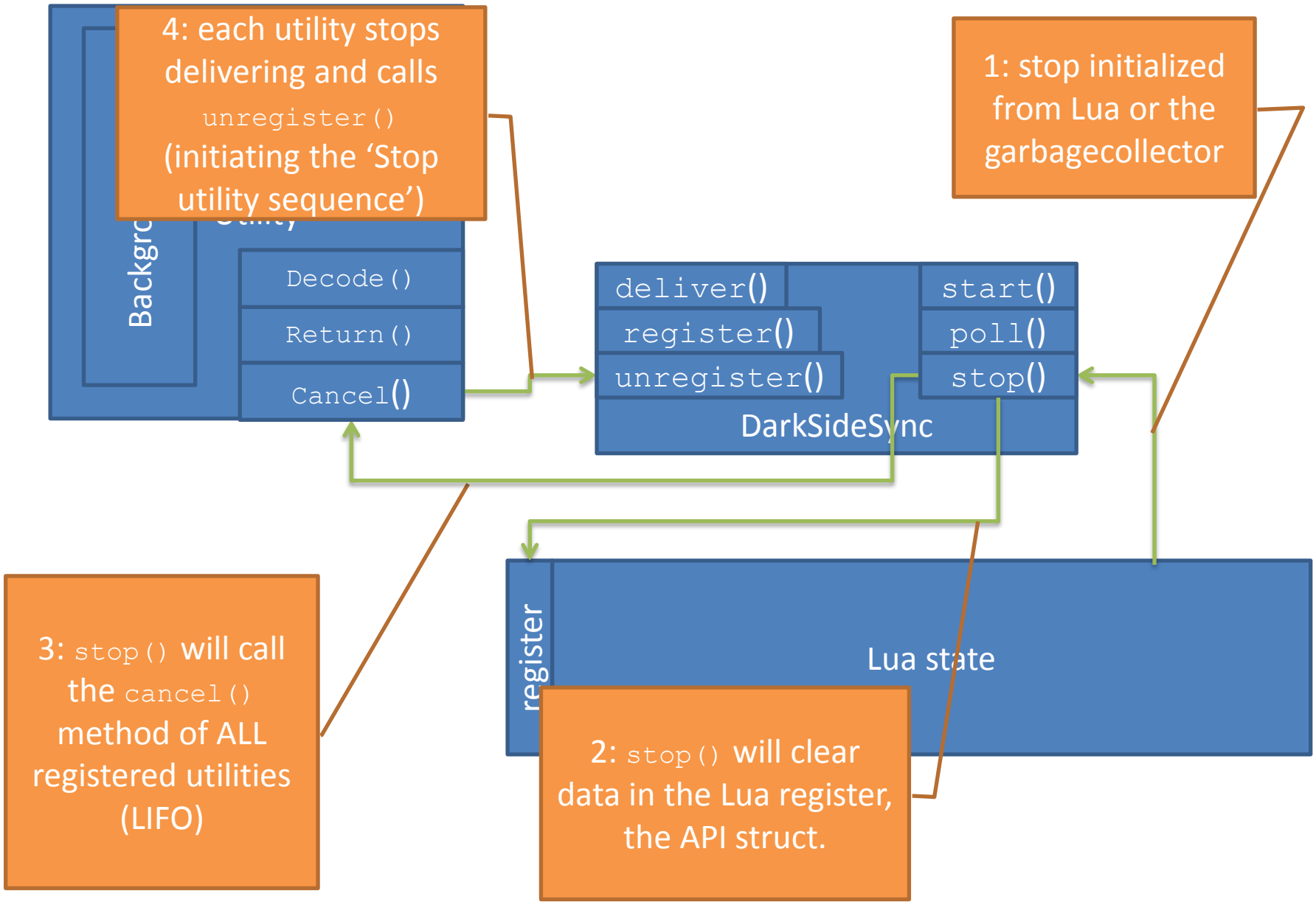
Collect sequence



Return sequence



Stop utility sequence



Stop DSS sequence