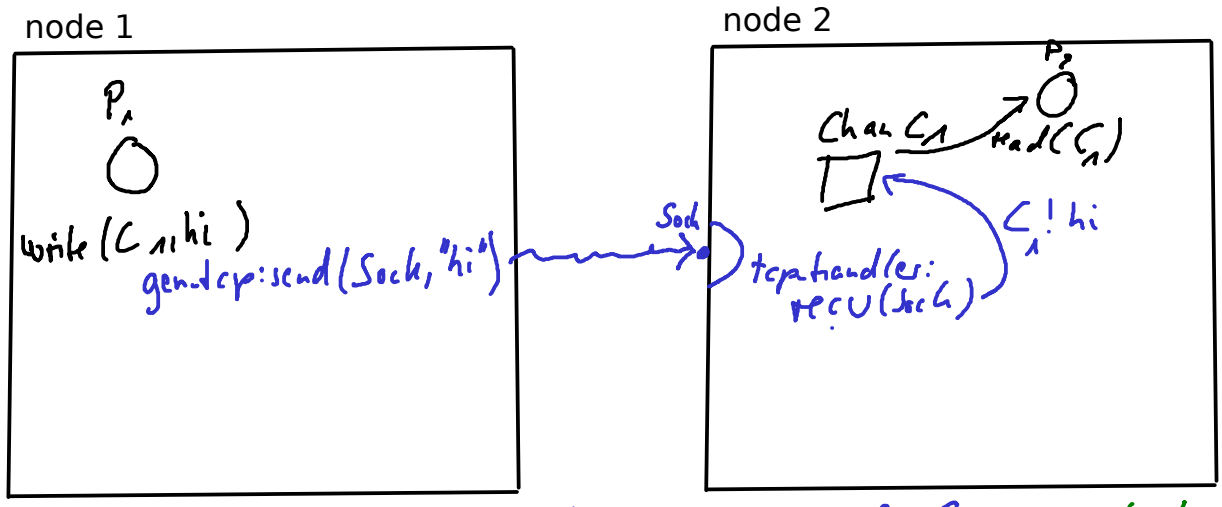
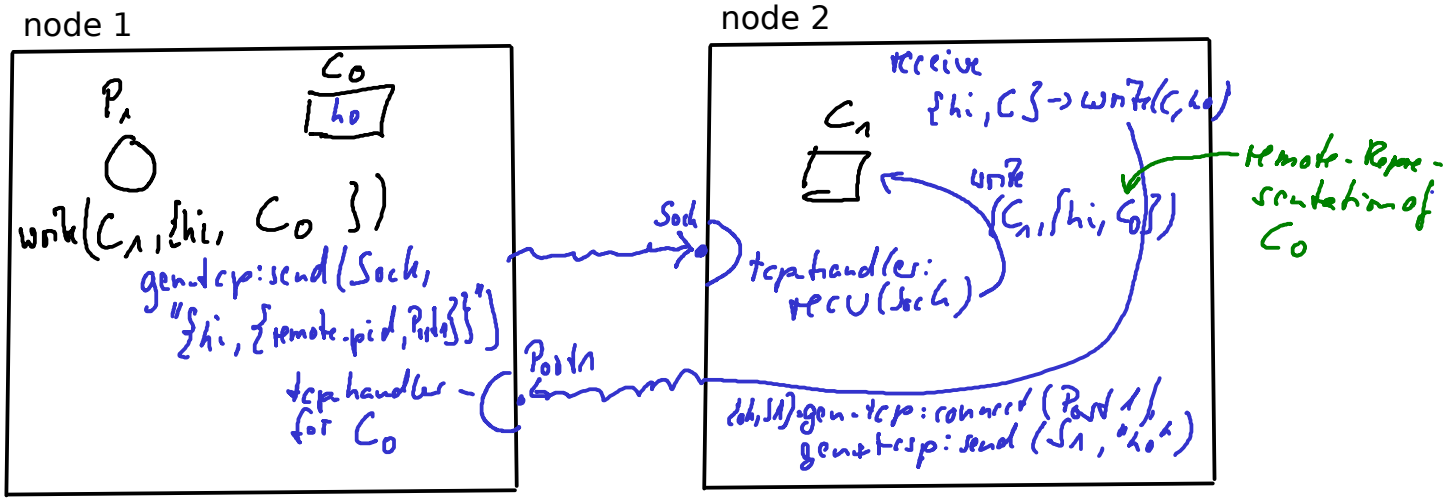


two possibilities: 1. Sock only receives msgs for C_1 — in lecture
2. Sock receives msgs for all processes, then the representation of C_1 on node 1 has to contain identifier for C_1 on node 2 and tcp_handler needs a mapping from ids to channels. — in exercises

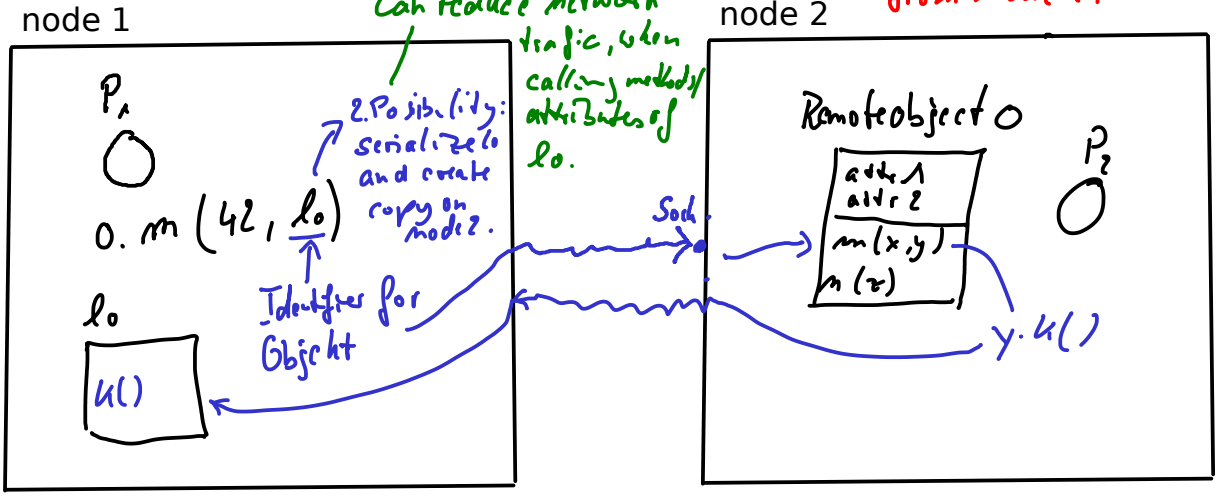


If a local channel leaves the node, it has to be converted into an external representation and we have to guarantee, that a tcp_handler is running.

Comparison with RMI/RPC:

But we produce a copy, hence state of the original lo cannot be changed from node 2.

Can reduce network traffic, when calling methods/attributes of lo.



Registration of special objects necessary for first contact.

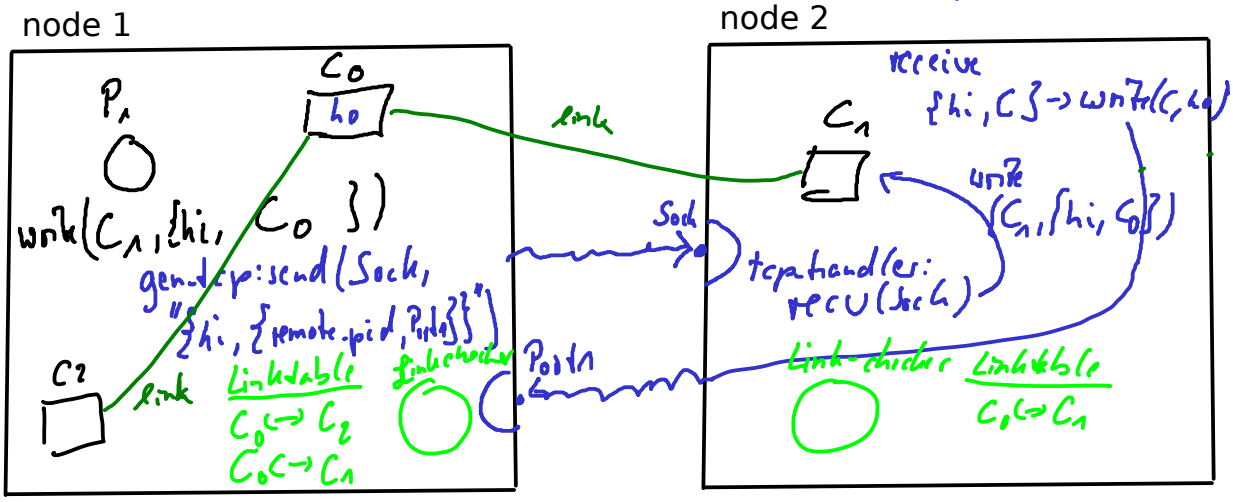
RMI/RPC extends sequential programming to a distributed setting.
However, RMI objects (e.g. servers) can be accessed concurrently by multiple parallel clients.
As a consequence, you have to consider all problems of concurrent programming as well, like critical sections, mutual exclusion, deadlocks ...

Robustness (linking): 2 Possibilities: link a process to a chan or link two chans to each other

idea: send crash message to the linked chan.

Erlang: link processes

Symmetric and therefore the better choice.



A crashed node will not inform linked channels (send a message). Therefore, every node has to send alive messages to linked nodes (channels), to check qhether they still running. This should be done with a given delay in the background (every 2 sec).