Recoverable Response Linearizability

How to formally define recoverable response linearizability?
All current definitions uses history and sequential specification in order to specify the requirements. Given a history H describing a run, any definition specify the way we need to complete pending operations and linearize the resulting history so that we get a legal sequential history. However, the language of histories cares only for the order of operation and the status of the object (to be consistent). We can not use this language in order to describe our requirement for a process to know the return value of the last operation. This implies that if we want to use histories as a definition, we need to specify the return value requirements as part of the object behaviour.

In our model we require the object to have a recover function, such that if a process fails in the middle of an operation, that is, in the history there is an invocation step followed by a crush step, then the recover function (which is executed automatically upon recovery) will add the response step corresponding for the last pending invocation step of the process. This capture the main idea of recovering as handling the last interrupted operation and completing it (unlike other definitions that does not have it), but it says nothing regarding the return value.

Take for example a primitive swap. Then, what does it means to recover an operation as $res \leftarrow swap(R, val)$?! Assuming only R is in the shared memory, as an operation can only access a single location in the shared memory. In the language of histories, once the program counter moves to the next line then the operation is done and there is a response in the history. If the process cruses at this point then upon recovery it knows the operation is complete because of the pc. However, the return value res is lost, although we might need it for the rest of the computation.

On the other hand, if we consider a primitive write, then the fact that program counter moved to the next line is enough for the process to recover, as a write returns only ack. In this case, considering the move of the program counter as the response of the write operation, then being able to re