Filters and Pipelines in Erlang

Erlang processes can be filters that receive input and pass it on as possibly modified or combined output only if the input satisfies certain properties.
Here is an `Echo` (also on Moodle) process.

```
-export([echo/0])

echo() ->
   receive
           stop -> ok;
           Msg -> io:format("Echo:␣~p\n", [Msg]), echo()
   end.

...

Echo = spawn(?MODULE, echo, [])
```

The echo process however prints its input on the terminal. We now want to create echo like filter processes that send messages to other processes.
We want to define processes P_i ($i = \{1, 2, \ldots\}$) that consume some amount of messages and forward only every i^{th} message unaltered to some other process. For example the process P_5 shall only forward every 5^{th} message it receives. The filter processes P_i shall (at least) act on the following message types:

{**`set_sender, Pid`**} configure the filter process so that it forwards messages to the process with the process id `Pid`.

{**`filter, Msg`**} forwards the message {`filter, Msg`} to the configured process but only every i^{th} message.

1. **Every Other Message**
   Please program and create the filter process P_2 that only forwards every other message. Test your filter by configuring it to forward its messages to the above echo process (the one that prints the messages it receives to the terminal), then send your filter process the messages {`filter, 1`}, {`filter, 2`}, {`filter, 3`}, {`filter, 4`}, {`filter, 5`}. The echo process is supposed to print {`filter, 2`}, {`filter, 4`}.

2. **Collector**
   Please program and create a filter process C, that assembles items and forwards lists of assembled items. Process C shall react on the following message types:

   {**`set_sender, Pid`**} (as before) configure the collector process so that it forwards messages to the process with the process id `Pid`.

   `reset` start a new collection. Reset the internal list to the empty list `[]`.

   {**`filter, Msg`**} collect the value `Msg`, append it to the internal list and forward the internal list as {`filter, List`} to the configured process, where `List` is the extended list.

Configure your process C to forward messages to the Echo process. Test your process with the messages `reset`, {`filter, 1`}, {`filter, b`}, {`filter, 3`}. The Echo process shall output {`filter, [1]`}, {`filter, [1,b]`}, {`filter, [1,b,3]`}.

3. **Filter Pipelines**
   Filters can be combined to pipelines by connecting them, i.e. by configuring processes to forward messages to other filter processes.

   Please build a filter pipeline: $P_2 \rightarrow C \rightarrow$ Echo

   Send the following sequence of messages to the P_2. What is the output of Echo?

```
P2!{filter,120},
P2!{filter,109},
P2!{filter,150},
P2!{filter,101},
P2!{filter,155},
P2!{filter,114},
P2!{filter,189},
P2!{filter,114},
P2!{filter,27},
P2!{filter,121},
P2!{filter,68},
P2!{filter,32},
P2!{filter,198},
P2!{filter,99},
P2!{filter,33},
P2!{filter,104},
P2!{filter,164},
P2!{filter,114},
P2!{filter,212},
P2!{filter,105},
P2!{filter,194},
P2!{filter,115},
P2!{filter,24},
P2!{filter,116},
P2!{filter,148},
P2!{filter,109},
P2!{filter,173},
P2!{filter,97},
P2!{filter,8},
P2!{filter,115},
P2!{filter,191},
P2!{filter,33}
```

There is no need to be prepared for face-to-face technical discussions this week. It is sufficient to repeat the output of the Echo process verbally to me to fulfil the exercise.