In the following we present a simplified version of the flush protocol proposed by K. Birman et al. in "Lightweight Causal and Atomic Group Multicast".

We assume that the process p most recently installed $view_i$. Notice that a process may be running the flush protocol for $view_{i+k}$ despite not yet having installed $view_{i+1}$. We also assume reliable channels, therefore we omit step 5 from the original algorithm.

1. On receiving $view_{i+k}$, process p increments a local counter variable `inhibit_sends`; while the counter remains greater than 0, new data messages will not be initiated. Process p forwards a copy of any unstable message m that was sent in $view_j$ ($j < i + k$) to all processes in $view_{i+k}$ and then marks m as stable. Lastly, p sends $\boldsymbol{flush_{i+k}}$ to each member of $view_{i+k}$.
2. On receiving a copy of message m, p examines the index of the view in which m was sent. If p most recently installed $view_i$ and m was sent in $view(m) < i$, p ignores m as a duplicate. If m was sent in $view_i$ and is not a duplicate, p delivers m. If m was sent in $view(m) > i$, p saves m until $view(m)$ has been installed.
3. On receiving $\boldsymbol{flush_{i+k}}$ messages from all processes in $view_{i+1} \cap view_{i+k}$ (for any $k \geq 1$), p installs $view_{i+1}$. It decrements the `inhibit_sends` counter and if the counter is now zero, permits new messages to be initiated. Any message m that was sent in $view_i$ and has not yet been delivered may be discarded.
4. A message can be discarded as soon as it has been delivered locally and has become stable. Notice that a message becomes stable after having been forwarded at most once (in step 1).