ChatGPT

Live-State Flicker Trace: `targetNoteIds` & Membership in Snapshot Preview

Snapshot Preview Workflow Overview

When switching workspaces (or loading a snapshot), the `useNoteWorkspaces.previewWorkspaceFromSnapshot` routine applies the snapshot's panel and open-note state to the workspace. The code performs these key steps in order:

1. **Collect Declared Note IDs:** It starts by gathering the set of note IDs that should be open (`declaredNoteIds`). This initially includes all notes listed in the snapshot's `openNotes` array. If the snapshot's openNotes is empty, it falls back to include all note IDs found in the snapshot's panels [1]. It then **adds any notes that were previously open** in this workspace (from the last session or cached state) to ensure none are lost [2]. This yields the full intended membership of open notes for the preview.

2. **Set Workspace Membership:** The hook updates the workspace's membership set to `declaredNoteIds` [3]. At this point, `workspaceNoteMembershipRef` for the workspace contains all notes that *should* be open according to the snapshot (plus any carried over from previous open state).

3. **Filter Panels by Membership:** Using the updated membership, the code filters the snapshot's panel list to include only panels whose `noteId` is in the workspace membership [4]. This `filterPanelsForWorkspace` step prevents applying panel data for notes not considered part of the workspace. If a note was **missing from the snapshot's openNotes**, and it wasn't already in the membership from a previous state, its panel will be filtered **out** here (since the membership set wouldn't contain that note). **This is a critical point:** a note omitted from `snapshot.openNotes` can get dropped before we even apply the panels, confirming **Hypothesis H1** (the note is missing from the preview snapshot data).

4. **Prepare Target Note IDs:** The code constructs `targetIds` as a set of all note IDs that need to be ensured on the canvas. It starts with the `declaredNoteIds` and also adds any note IDs from the filtered panel list [5] [6]. In practice, after the filtering in step 3, `targetIds` will include the same notes as the current membership (plus any extra from panels, if not already included). If a note was dropped in step 3 due to missing membership, it **won't appear in** `targetIds` either (again pointing to H1 if a note is absent here).

5. **Commit Open Notes (Snapshot Open Slots):** Next, `commitWorkspaceOpenNotes` is called with the normalized list of open notes from the snapshot [7]. This saves the snapshot's open-notes into the hook's state and updates the membership and cached snapshot open-notes list. Importantly, by default this call updates the membership to exactly match the given open-notes list [8]. In our

scenario, if the snapshot's `openNotes` was missing a note (e.g. note B), the membership may already have omitted B. The commit will then simply reinforce that state (membership containing only the notes present in `normalizedOpenNotes`). In cases where the snapshot had *no* open notes at all, this commit can even set the membership to an empty set, potentially clearing out a note that was momentarily in `declaredNoteIds` from panel inference. (For example, if `snapshot.openNotes` was empty but a panel existed for note B, we'd add B to membership, then commit an empty open-notes list, which would overwrite membership to empty. B would then be lost from membership at this point.)

6. **Apply Panel Snapshots:** Finally, `applyPanelSnapshots` is invoked with the filtered `panels`, the `targetIds` set, and any component data [9]. Inside this function, the code uses the current membership and `targetIds` to decide how to update the canvas DataStore:

7. It fetches the workspace's membership set again (`getWorkspaceNoteMembership`) [10].
8. If membership is unknown (not set) it would bail out, but in our case membership is known (even if it might be incomplete).
9. It then constructs an **allowed** set of note IDs initialized from the membership, and **explicitly adds all** `targetNoteIds` **to it** [11]. This ensures that notes intended to be targeted will not be pruned even if they weren't originally in the membership.
10. The DataStore's existing panel entries are iterated, and any panel whose note ID is neither in the allowed set nor in the target set is removed (pruned) [12]. Panels for notes not in the workspace (or no longer targeted) get deleted here.
11. Then, for each panel in the incoming snapshot list, if its noteId is in `targetIds`, it gets applied to the DataStore (added/updated) [13].

Given the logic above, whether the second note "drops out" **before or during** `applyPanelSnapshots` depends on if it made it into the `targetIds` and membership to begin with:

- **If the preview snapshot omitted the second note in its openNotes:** The code will not list it in `normalizedOpenNotes`. Unless that note was already open in the previous state (thus added to `declaredNoteIds` via previousOpenSlots), it will *not* be in the membership when filtering panels. Any panel snapshot for that note would be filtered out [4], so it never reaches `applyPanelSnapshots`. In effect, the note is already gone *before* the apply step – it's neither in `targetIds` nor in the membership by the time we call `applyPanelSnapshots` [9]. This matches **H1**, indicating the note was missing from the preview snapshot data. The second note would not be opened in the preview at all.

- **If the snapshot did include both notes (or a previous state added it to membership):** In this case, `declaredNoteIds` and `targetIds` would contain the second note. The membership would also include it, so the panel data would **not** be filtered out. The second note's panel would be applied to the DataStore in `applyPanelSnapshots`. There is no code inside `applyPanelSnapshots` that would singly "drop" a targeted note's panel. In fact, the allowed-set logic explicitly *preserves* any note in `targetIds` [11]. Thus, if the second note made it this far, `applyPanelSnapshots` will keep it on the canvas rather than remove it. This suggests **H2 (a stale membership overwrite during apply)** is *not* occurring – the apply step isn't spontaneously dropping a note that was included.

- **Could `commitWorkspaceOpenNotes` overwrite membership with stale data?** The only time membership gets "overwritten" is when syncing with an out-of-date open-notes list. In the preview flow, we should consider the `useEffect` that syncs the context's openNotes into the workspace state [14]. This effect runs when the global `openNotes` (from the CanvasWorkspace context) change. In theory, if this ran at an inopportune time with a stale list, it might remove a note from membership. For example, if the UI's `openNotes` didn't yet include the second note while the workspace's membership did, calling `commitWorkspaceOpenNotes` with that UI list would drop the note from membership. However, in practice the code checks that the context's `openNotesWorkspaceId` matches the current workspace before syncing [14], and it updates after the preview is applied. In our trace, once the preview opens the notes (calling `openWorkspaceNote` for each missing entry), the context's `openNotes` list is up-to-date. The subsequent commit sync just mirrors the correct open-notes set (and in many cases it's a no-op since nothing changed) rather than introducing stale data. We found no evidence that this effect was overwriting membership with older data; it fires to reflect *actual* open notes, not an outdated set.

Conclusion: H1 vs H2

From this code-level tracing, it appears the **second note is dropped before `applyPanelSnapshots` is called**, i.e. during the preparation of the snapshot preview, rather than pruned during the apply step. The culprit is an **incomplete snapshot** (missing the note in `openNotes`) combined with the membership filtering logic:

- If the snapshot's `openNotes` did not include the second note (and there was no lingering membership from a previous session to catch it), that note never makes it into `targetNoteIds` or the open-notes list for the preview. The panel data for it is filtered out due to the membership mismatch [4]. In short, the note is absent from the preview snapshot state itself, so it disappears **before** any panel application occurs (supporting **H1**).

- We did not find a scenario where a note that *was* included in the preview snapshot was later removed by a faulty membership overwrite inside `applyPanelSnapshots`. The apply logic is designed to preserve targeted notes, and any membership syncing (`commitWorkspaceOpenNotes`) occurs either before or after the apply step using intended data. Thus, **H2 (stale membership overwrite)** is not borne out by the code path – membership updates only ever reflect the snapshot's intended open notes (or the UI's actual open notes), and do not independently drop a note that was properly included.

Bottom line: The "flicker" (the second note vanishing) is caused by the note never being fully included in the preview's open-note set to begin with (H1), rather than it being included then erroneously removed during the application (no evidence of an H2-style removal). The code path confirms that if a note is missing from `snapshot.openNotes` and not accounted for elsewhere, it will be omitted in the preview – pointing to the snapshot generation/content as the root cause of the flicker. [1] [4]

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] use-note-workspaces.ts
file://file_00000000ef34720c8dc2ff80f62b1e19