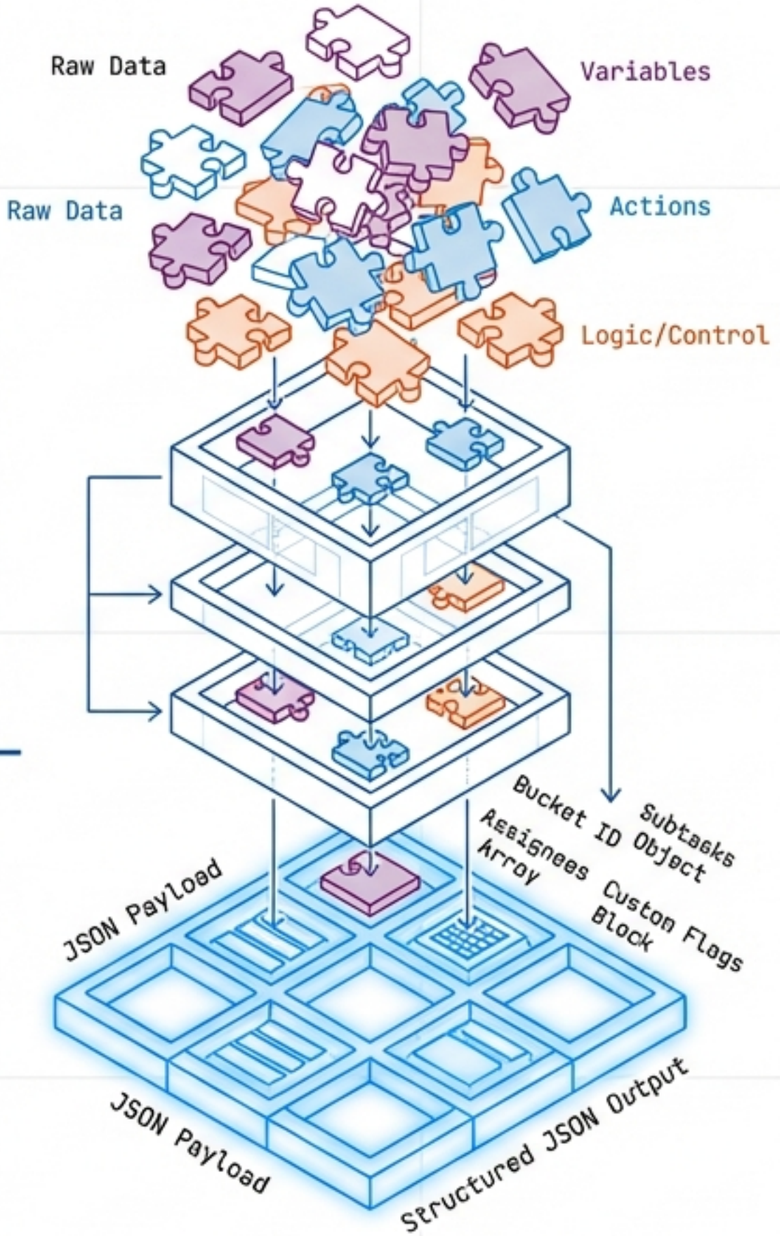
Mastering Planner Data Extraction

A Power Automate Deep Dive

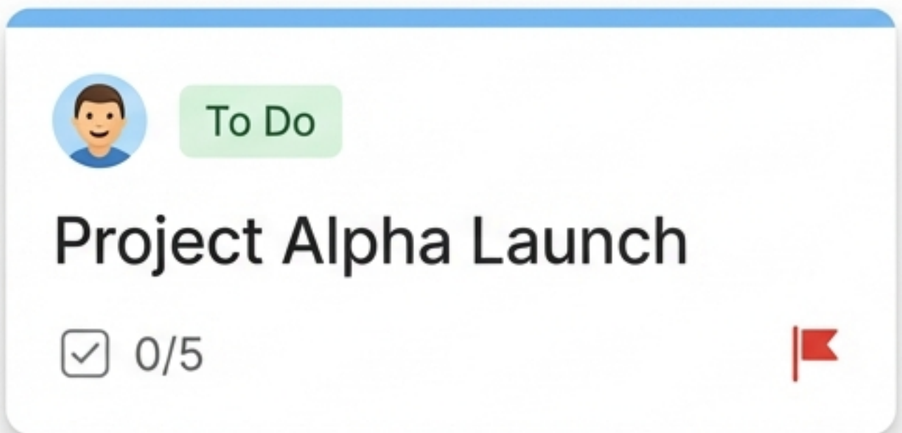
A step-by-step architectural guide to extracting Assignees, Subtasks, Buckets, and Custom Flags into a unified JSON dataset.



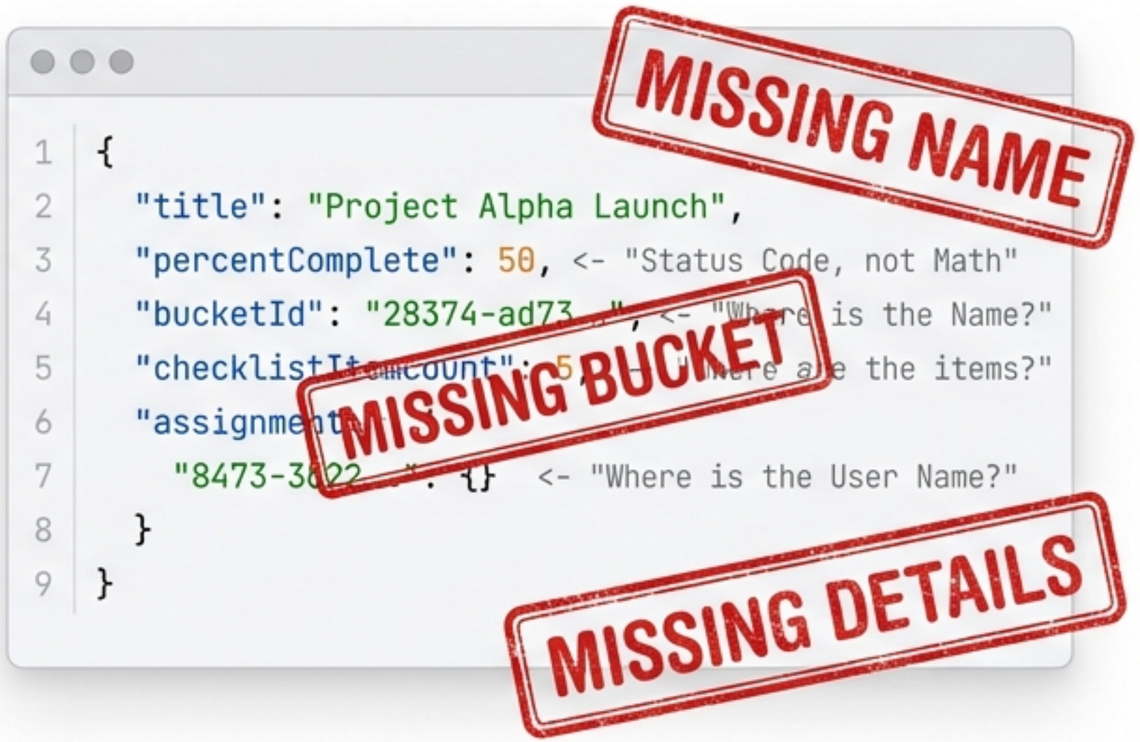
Designed for Power Automate Developers & Data Analysts

The Data Gap: What ‘List Tasks’ Actually Gives You

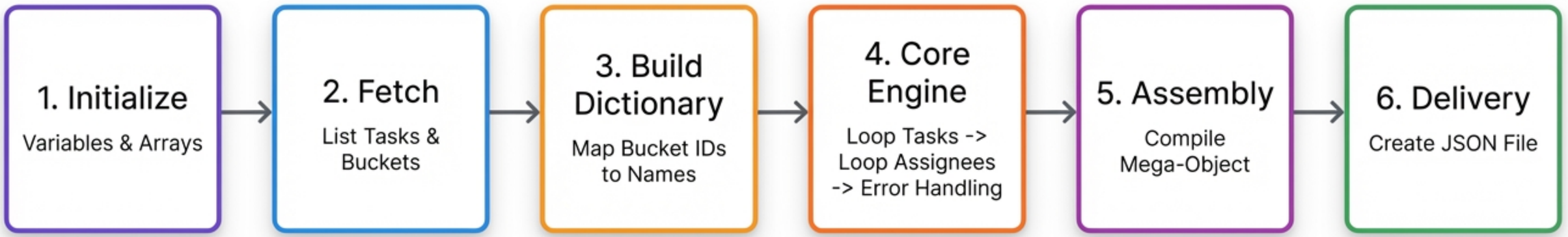
What You See in Planner



What You Get (Raw JSON)



The Logic Architecture

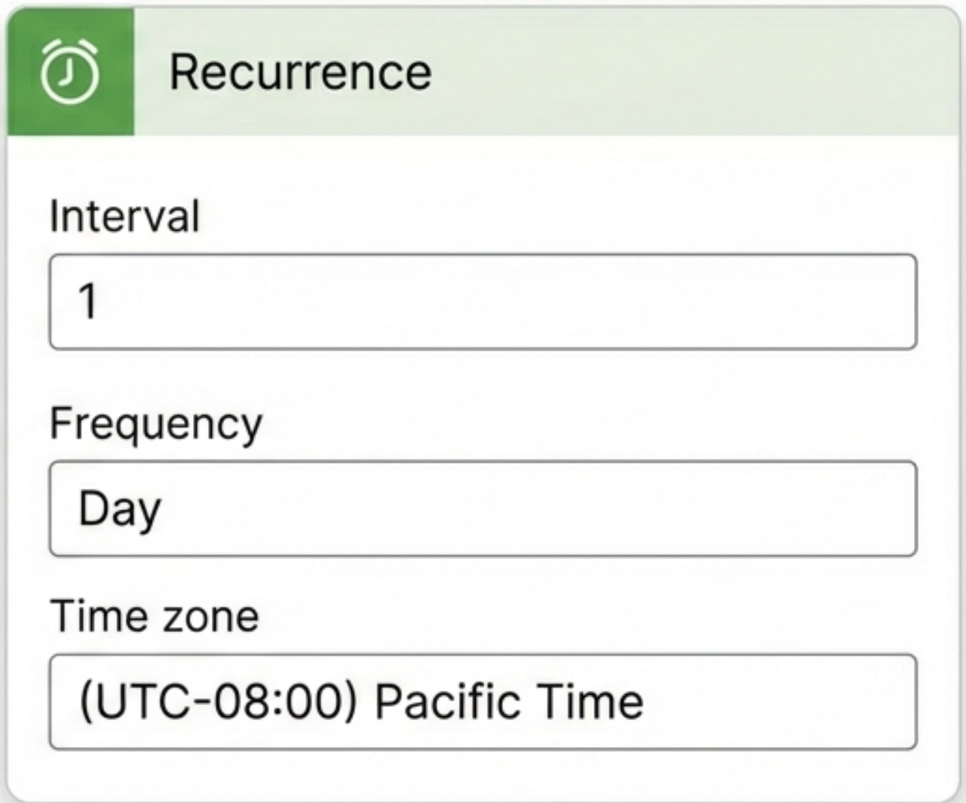


Step 1: The Trigger & Schedule

Action: Scheduled Cloud Flow

Frequency: Daily

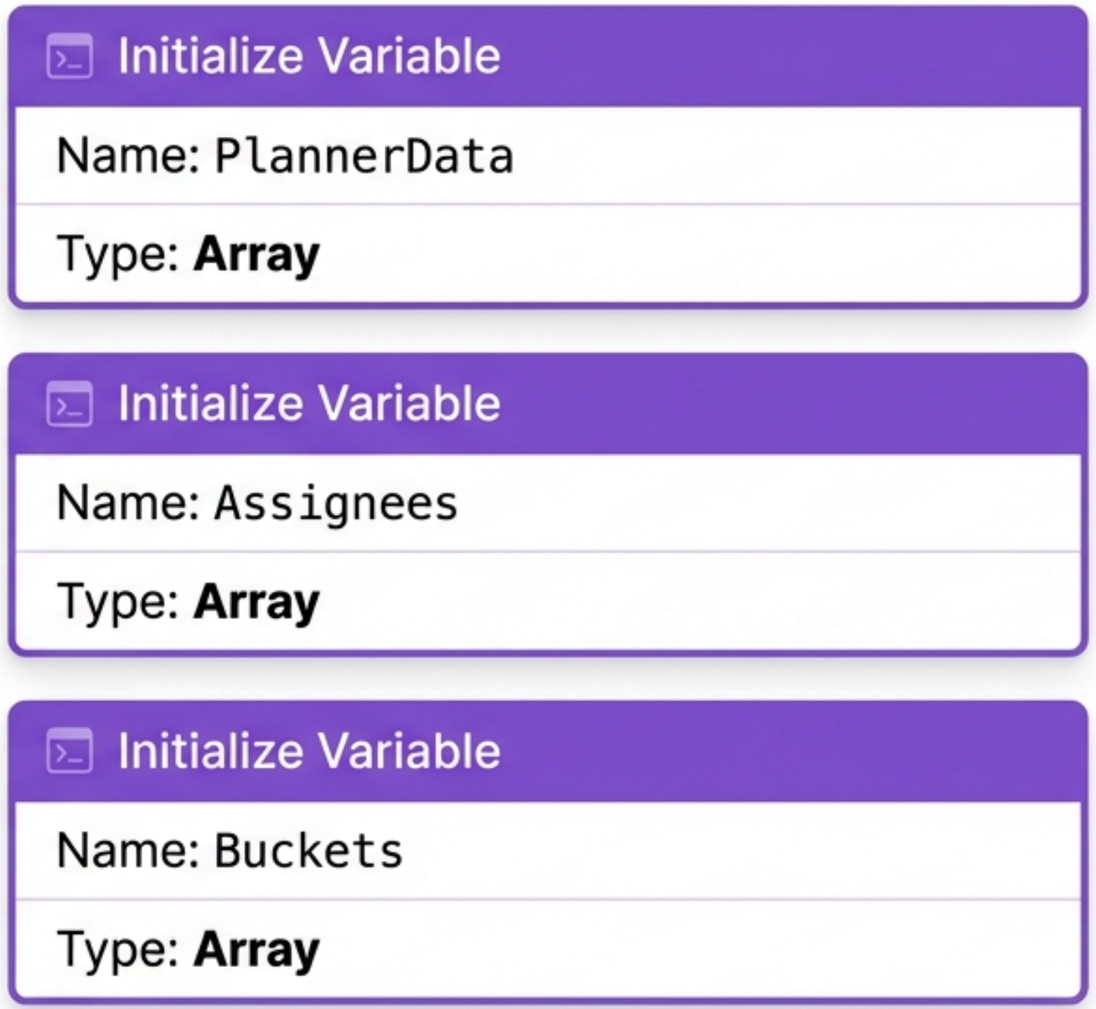
Context: Acts as a live data source
Context: Acts as a live data source
for Power BI refresh cycles.



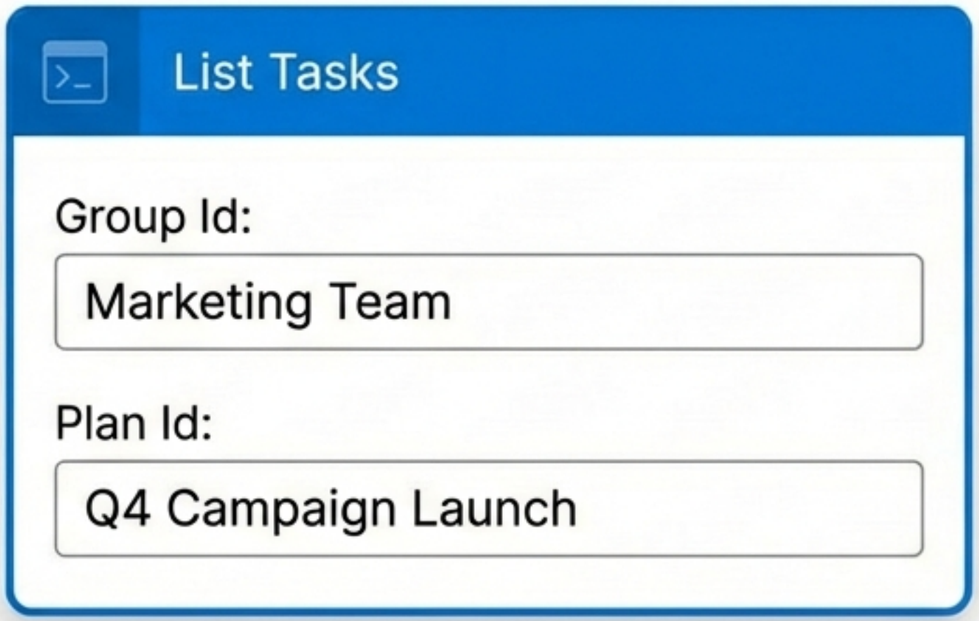
Step 2: Initializing Storage Containers

We must initialize three specific **Array** variables. These containers will hold our fragmented data as we process it.

1. **PlannerData**: The final consolidated output.
2. **Assignees**: Temporary buffer for user profiles.
3. **Buckets**: The ID-to-Name directory.



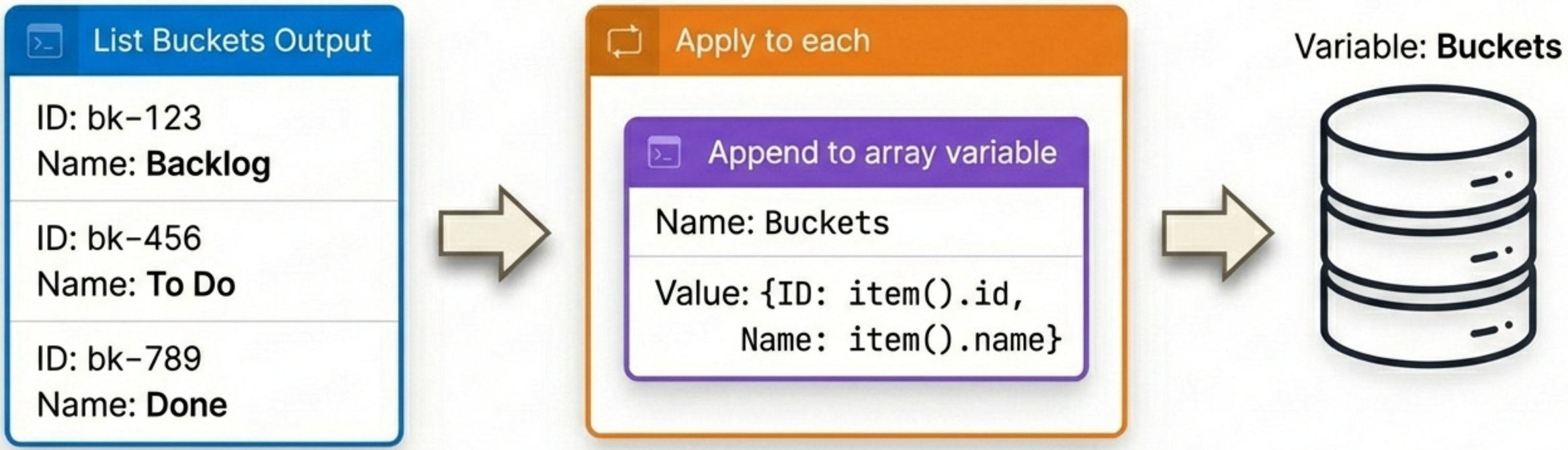
Step 3: Fetching the Raw Lists



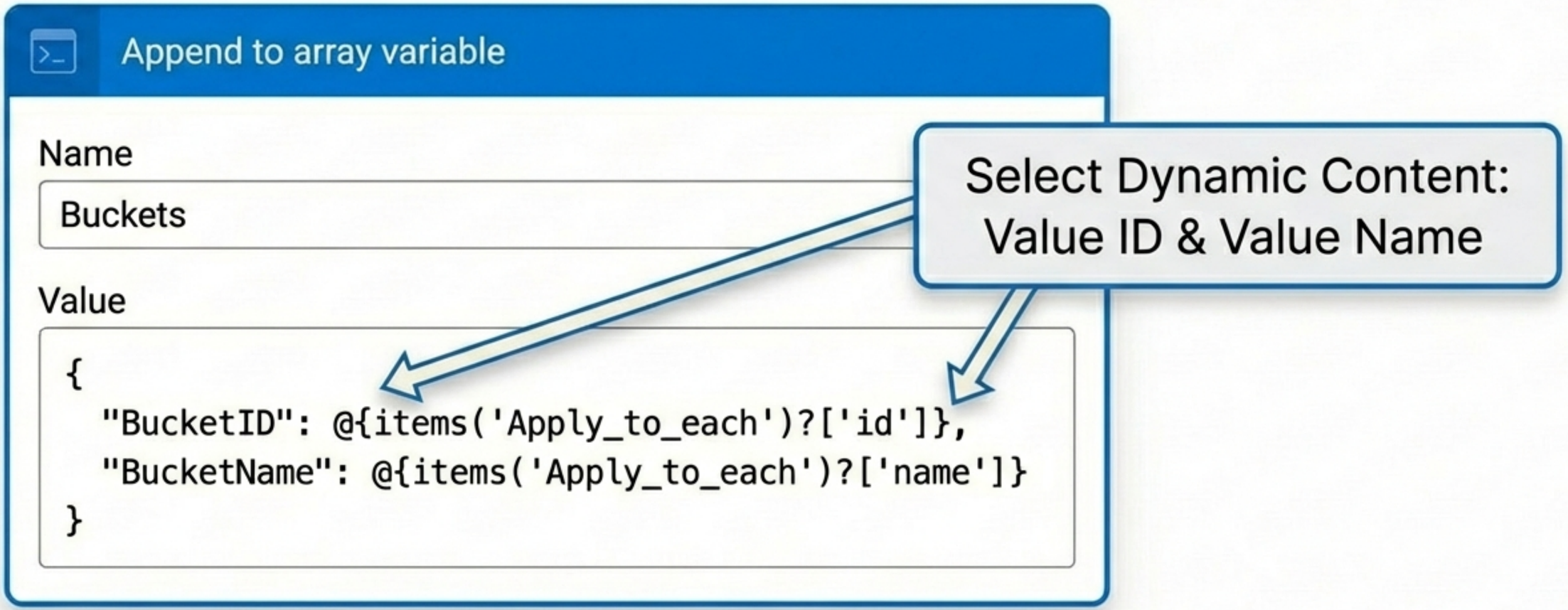
Pro Tip: The 'Plan ID' field accepts Custom Values. Use this to dynamically iterate through multiple plans.

Step 4: The Bucket Strategy

Tasks only contain a 'BucketID'. To report on meaningful names (e.g., '**Backlog**'), we must first build a reference directory.

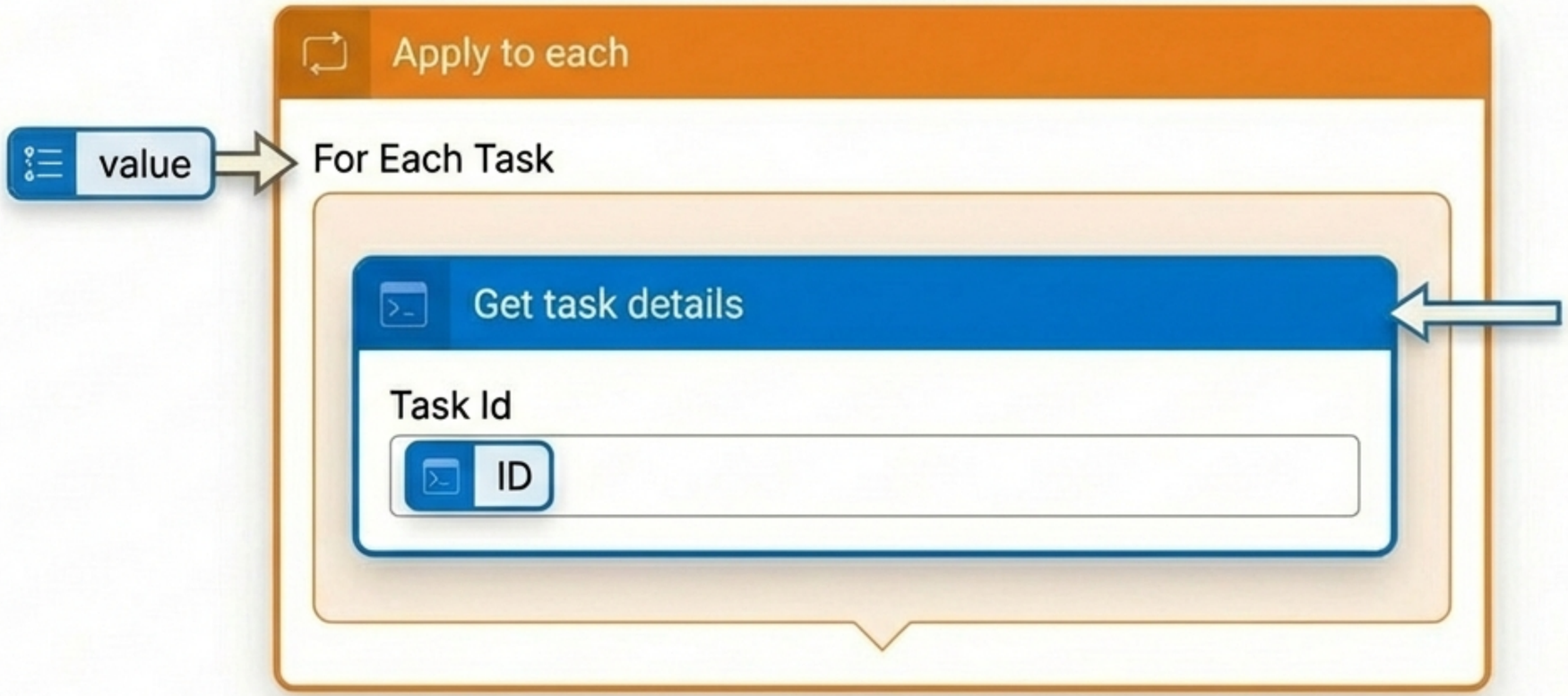


Step 5: Constructing the Bucket Directory



Step 6: The Main Loop & Detail Retrieval

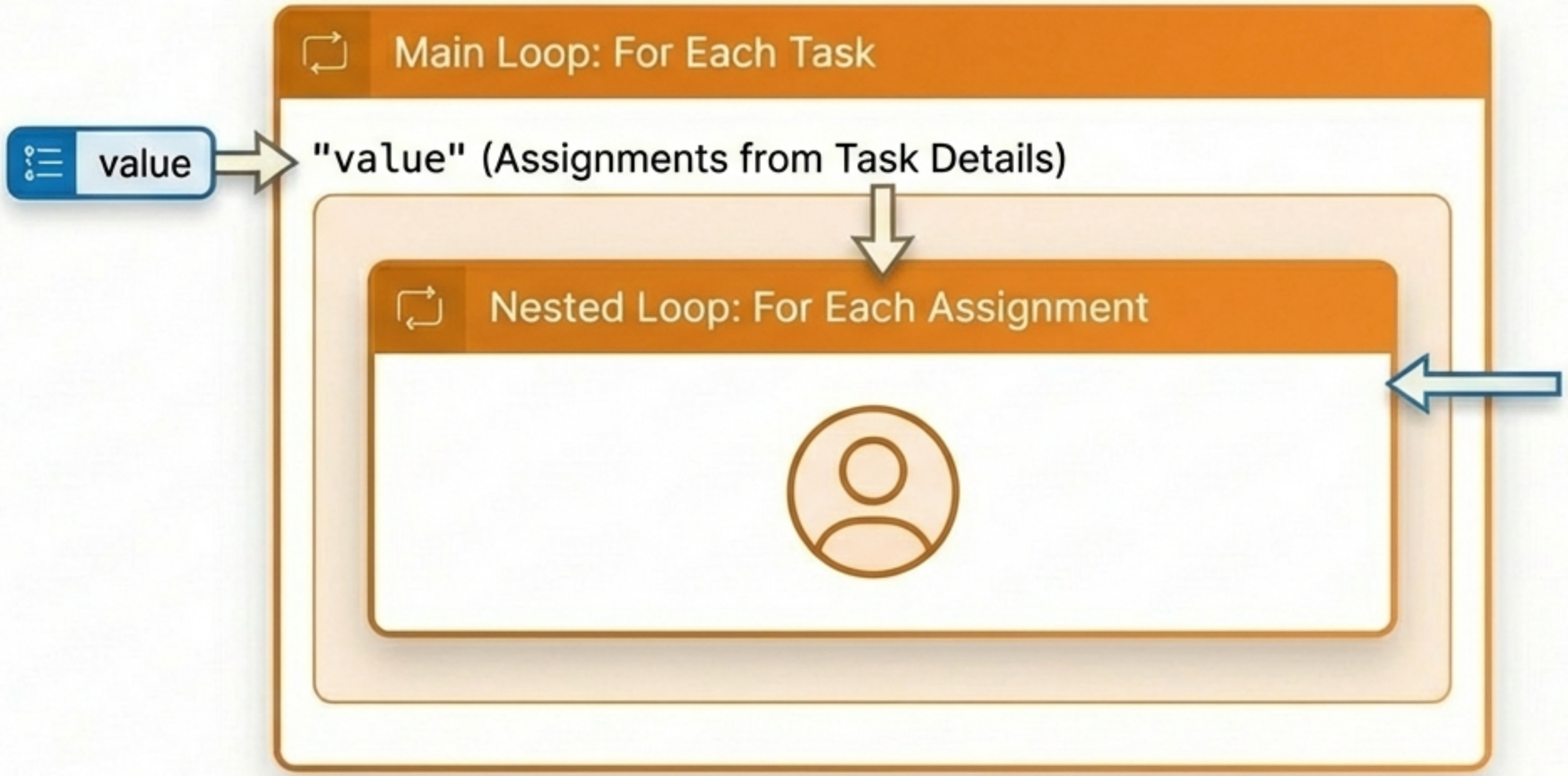
We now begin the core processing engine. We iterate through every task and immediately fetch its full details.



Crucial: “List Tasks” gives the skeleton. “Get Task Details” gives the meat (Checklists, Descriptions).

Step 7: The Assignee Challenge

Tasks can have multiple assignees. This requires a nested loop structure.



We must iterate “Assignments”, not “Assignees”.

Step 8: Resolving User Profiles

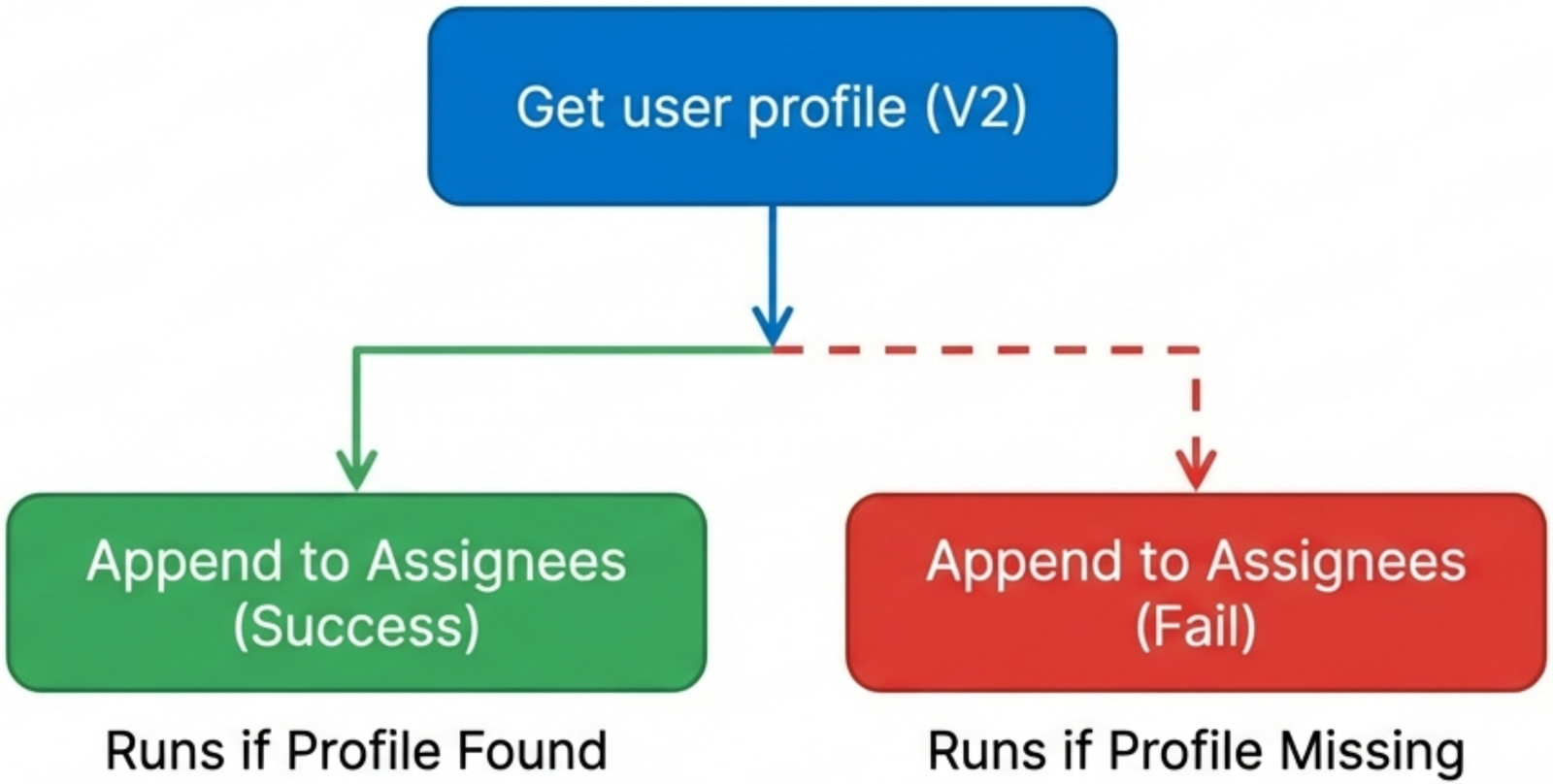
Get user profile (V2)
(Office 365 Users)

User (UPN): Assignee User ID

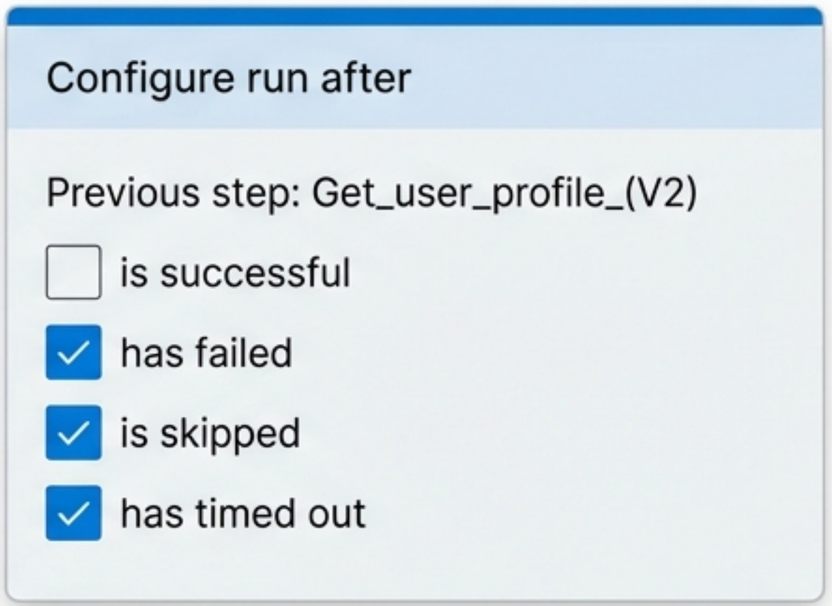
FAILURE POINT: If a user has left the organization, this action will fail. Without error handling, the entire flow crashes here.

Step 9: Advanced Error Handling (Parallel Branches)

We split the timeline to handle success and failure gracefully.



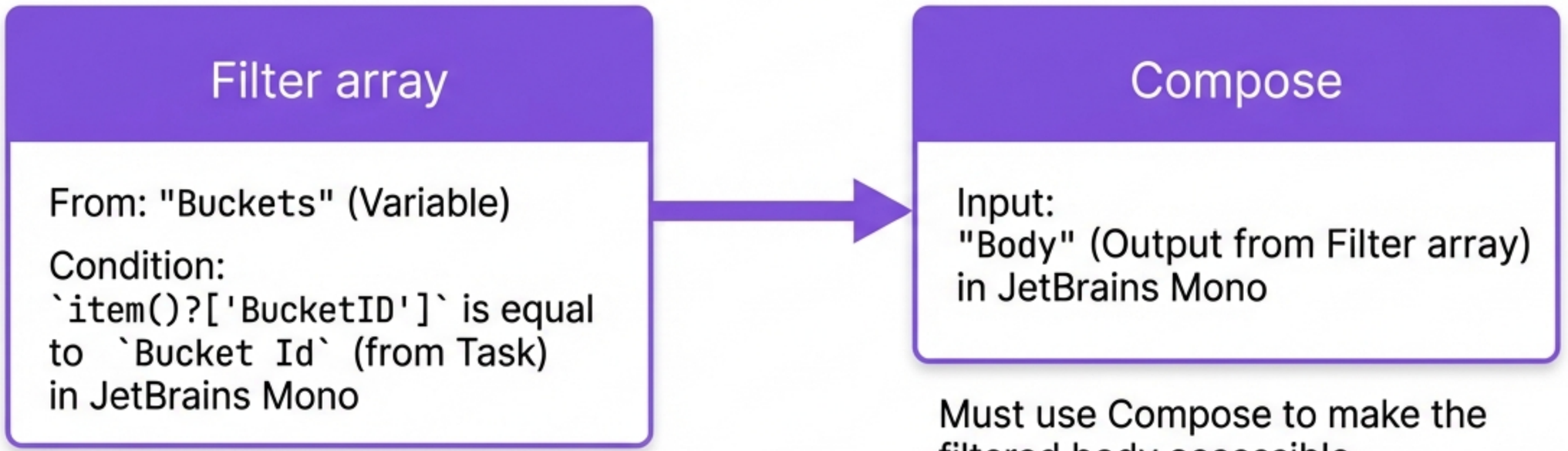
Step 10: Handling the “Ghost User”




```
{
  "AssigneeName": "Unavailable",
  "AssigneeID": @{items('Apply_to_each_assignment')?['userId']}
}
```

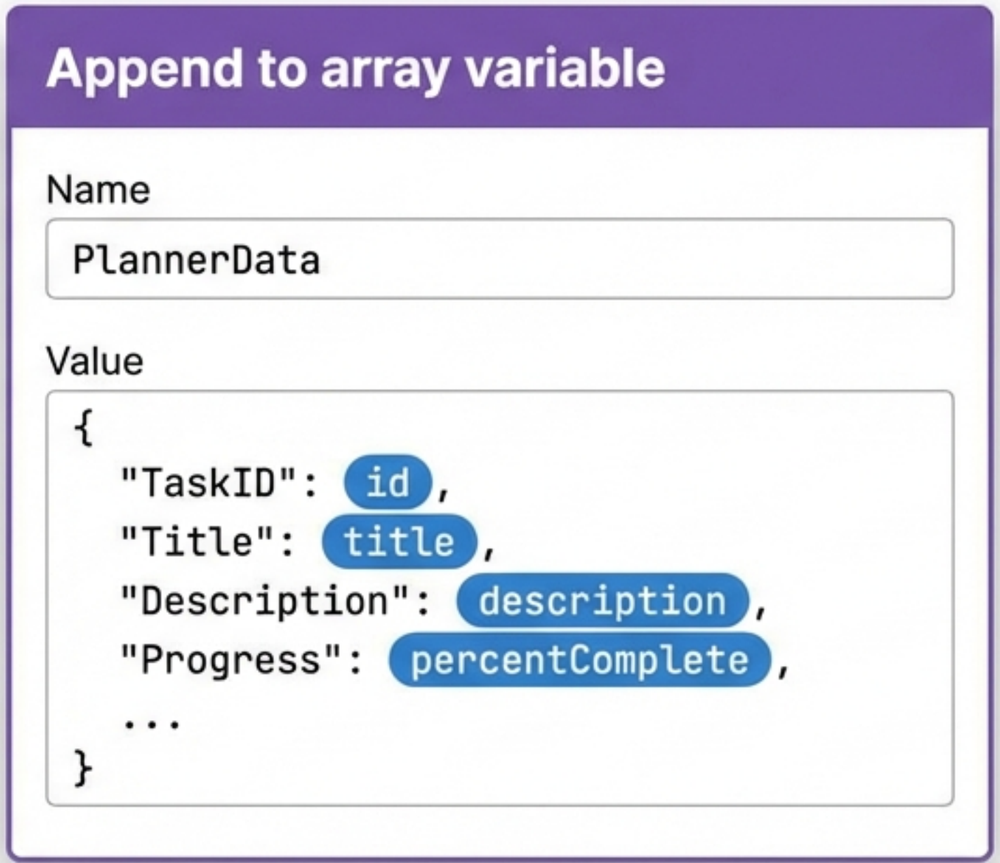
Step 11: Filtering the Buckets

Back in the Main Loop, we match the current task's Bucket ID against our Directory variable.



Step 12: Mapping the Mega-Object (Part 1)

We begin constructing the single JSON object that holds all task data.



Technical Architectural Clean

Step 13: Mapping the Mega-Object (Part 2)

Injecting the complex arrays and dates.

Append to array variable

Value

```
  ...
  "StartDate": "startDateTime",
  "DueDate": "dueDateTime",
  "Assignees": @{variables('Assignees')},
  "Subtasks": @{outputs('Get_task_details')?['checklist']},
  "Bucket": @{outputs('Compose_filtered_buckets')}
}
```

Step 14: Decoding Colored Flags

Power Automate sees colors. You see meanings. You must manually map these keys.

Power Automate Value	Your Report Key
	`pending`
	`urgent`
	`approved`

```
"Urgent": @{items('Apply_to_each')?['appliedCategories']?['category3']}
```

Technical Architectural Clean

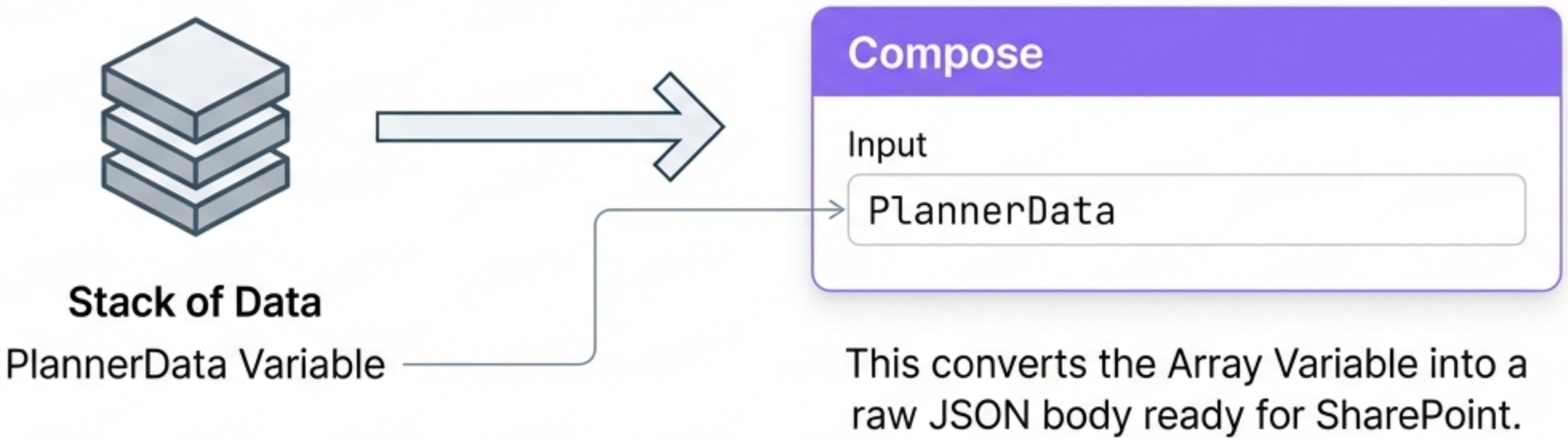
WARNING: The Critical Reset

High-contrast alert is anspacily to achieve the elətricolation providers.

You must **reset the Assignees variable** to **NULL** at the end of **the loop**. If you skip this, Task 2 will inherit Task 1's **assignees**, creating a snowball effect of incorrect data.

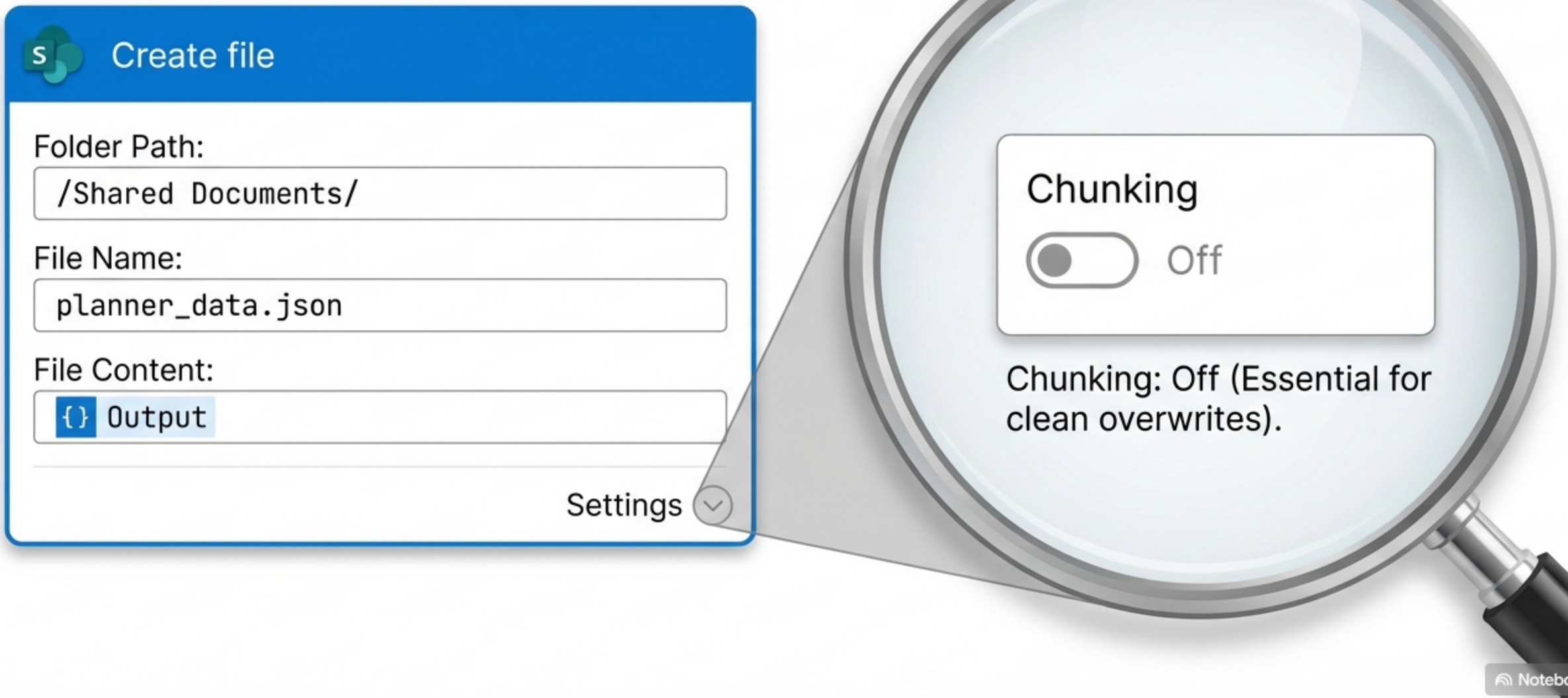
Step 15: Composing the Final Payload

Outside the loops, we prepare the data for file creation.



Step 16: Creating the JSON File

Final step to generate the file on SharePoint.



The Result: Reporting Ready

- Resolved Assignee Names
- Proper Bucket Names
- Subtask Details
- Custom Flag Statuses