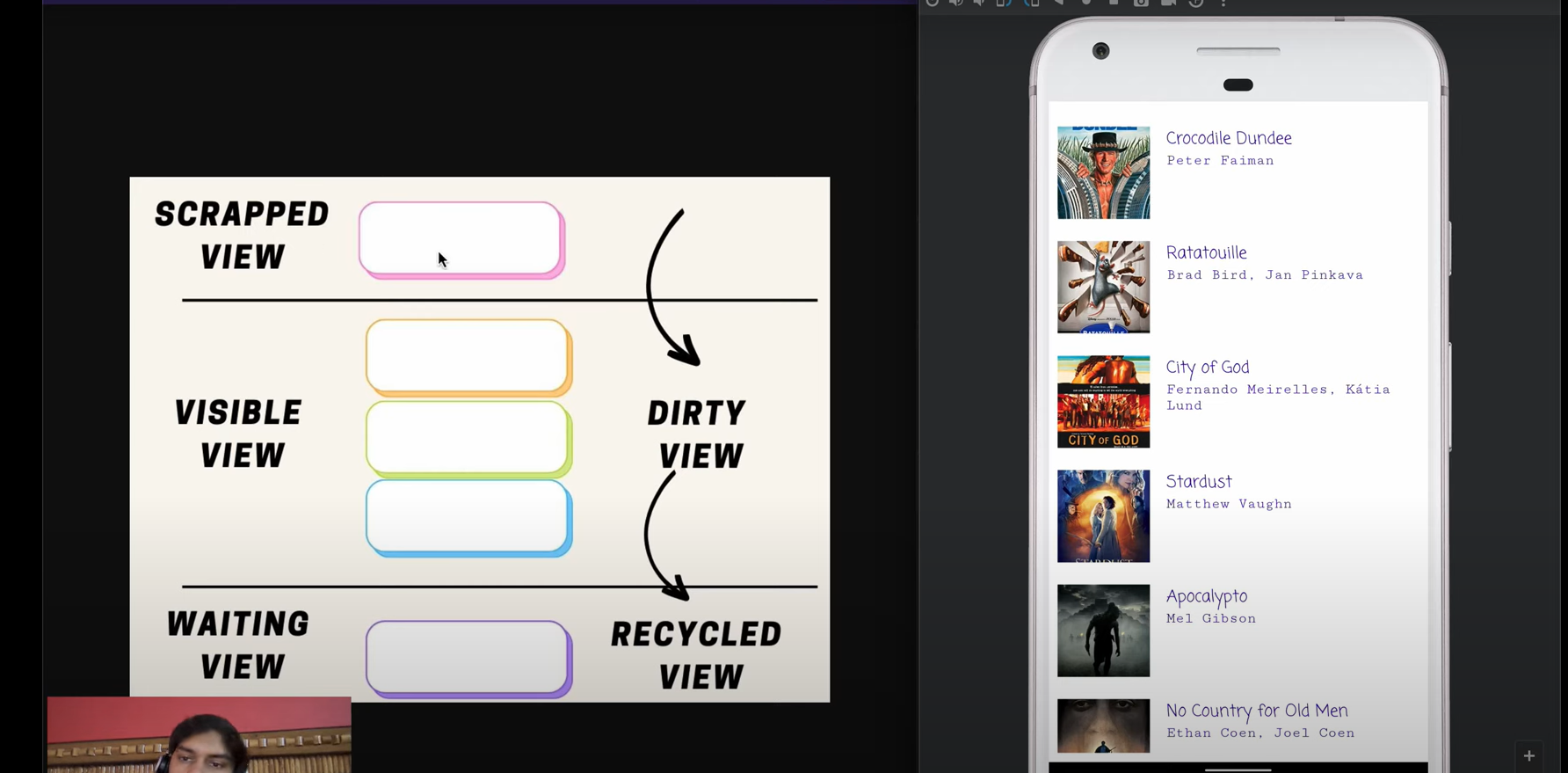
# Recycler View Internal Working:

### How RecyclerView Cache Works

RecyclerView is designed to handle large data sets efficiently by caching views and reusing them when possible. This process is key to its performance. To understand how the cache works and how many views will be created, let’s break down the caching mechanism and the creation of views based on a specific scenario with 20 items.



The more you scroll, more views from upper are called as scrapped views. It will take dirty view from scraped view and use it for waiting views, hence recycling it.

RecyclerView has three main caches:

1. \*\*View Cache\*\*: Stores detached views that can be reused without rebinding them.

2. \*\*Scrap Heap\*\*: Temporarily holds views that are being removed or are off-screen but still attached to the window.

3. \*\*RecycledViewPool\*\*: A pool of detached, recycled views for reuse, shared across multiple RecyclerViews.

### RecyclerView Internal Caching Mechanisms

1. \*\*ViewHolder Pattern\*\*: Each visible item in the list is represented by a `ViewHolder` object, which holds references to the individual views that make up the item (like `TextView`, `ImageView`, etc.). Instead of creating new views for each item, RecyclerView reuses the `ViewHolder` objects and updates the contents to display new data when needed.

2. \*\*Cache System\*\*:

- \*\*Scrap View Cache (Attached Views)\*\*: This cache holds views that are currently on-screen but need to be re-bound as new data is scrolled into view. Views in this cache are still attached to the window but have become invisible due to scrolling.

- \*\*RecycledViewPool (Detached Views)\*\*: This pool holds fully detached views that are no longer visible and have been removed from the screen. When a new item needs a `ViewHolder`, RecyclerView tries to grab one from this pool rather than creating a new one.

3. \*\*Item Recycling\*\*: When scrolling, instead of creating new `ViewHolder` objects for each new item, RecyclerView recycles off-screen views and re-binds them with new data, minimizing the cost of inflating views and creating UI components.

### Scenario: 20 Items in a RecyclerView

Let’s assume you have a \*\*LinearLayoutManager\*\* with 20 items. The screen can display 5 items at once.

1. \*\*Initial View Creation\*\*:

- Initially, 5 `ViewHolder` instances will be created to display the first 5 items (items 0–4).

- Each of these views will go through the normal `onCreateViewHolder()` and `onBindViewHolder()` process to be created and bound with data.

2. \*\*Scrolling Down\*\*:

- When you scroll down, items 0 and 1 will move off the screen, while items 5 and 6 will become visible.

- Instead of creating new `ViewHolder` objects for items 5 and 6, RecyclerView will \*\*recycle\*\* the `ViewHolder` objects that were used for items 0 and 1, and rebind them with the data for items 5 and 6.

3. \*\*View Recycling Example\*\*:

- Imagine scrolling from item 0–19. The process would look like this:

- Initially: Views for items 0–4 are created.

- Scroll down: Items 0–1 go off-screen, and views are reused for items 5–6.

- Scroll further: Items 2–3 go off-screen, and their views are recycled for items 7–8, and so on.

4. \*\*Cache Usage\*\*:

- RecyclerView keeps a few off-screen views in memory for faster access. For example, if only 5 items are visible, it may keep a couple of additional views cached (let's say 2 views above and 2 views below). So, 7 views might be created even though only 5 are visible at any time.

- If you quickly scroll back up, the views for items 0–1 might still be in the \*\*Scrap Heap\*\* or the \*\*View Cache\*\* and can be reused without going through the full re-creation process.

5. \*\*When the View is Recycled\*\*:

- Views are recycled once they move far enough off-screen and RecyclerView no longer considers them for display.

- Detached views go into the `RecycledViewPool`, where they can be reused by binding new data to them. This prevents the system from creating a new view unnecessarily.

### How Many Views Will Be Created?

- Let’s say you have 5 visible items at a time, and RecyclerView caches 2 additional views above and below.

- Total number of created views in the first scroll: `5 visible views + 2 extra views above + 2 extra views below = 9 views created at once`.

- As you scroll further down, \*\*new views will not be created\*\* beyond the cached views unless necessary. The RecyclerView will recycle the older, off-screen views.

### Data Structure Used in RecyclerView Backend

RecyclerView uses \*\*efficient arrays and pools\*\* to store the information:

- \*\*SparseArray\*\*: Used internally for holding the references to the `ViewHolder` objects in the \*\*RecycledViewPool\*\*. SparseArray is memory-efficient for large lists.

- \*\*LinkedLists\*\* or \*\*Stacks\*\* may be used for managing `ViewHolder` scrap items, ensuring quick access to recycled views.

RecyclerView’s state and scrolling logic are also supported by \*\*LayoutManager\*\*, which handles the arrangement of views on the screen and manages the recycling process in a way that minimizes overhead and memory usage.

### How RecyclerView Manages Fast Scrolling

- \*\*Pre-fetching\*\*: RecyclerView can pre-fetch views that will soon come on-screen, especially useful for fast scrolling. It predicts which views will be needed and tries to load them into the cache in advance.

- \*\*Batch Layout Operations\*\*: Instead of doing layout changes one by one, RecyclerView batches the layout operations and applies them all at once, reducing the processing overhead.

### Summary for Interview

If asked about the internal working of RecyclerView:

1. \*\*Efficient View Recycling\*\*: It uses the `ViewHolder` pattern to recycle views that move off-screen, minimizing memory use and avoiding repeated inflation of the same view structure.

2. \*\*Three Caches\*\*: RecyclerView manages views through a \*\*Scrap Heap\*\* (temporary detached views), \*\*RecycledViewPool\*\* (fully detached views), and a \*\*View Cache\*\* (holding invisible views that are attached).

3. \*\*Optimal View Creation\*\*: When displaying a list of 20 items with 5 visible at a time, RecyclerView will initially create only 7–9 views (including a couple of pre-cached views) and reuse them as you scroll down.

4. \*\*LayoutManager\*\*: Controls the placement and arrangement of items (Linear, Grid, Staggered), improving the speed of layout operations.

5. \*\*Data Structures\*\*: Efficient use of \*\*SparseArrays\*\*, \*\*LinkedLists\*\*, and internal caching ensures smooth scrolling and quick view recycling.

This detailed understanding will help you explain RecyclerView's inner workings and its memory efficiency during an interview.