

In one line: A list keeps only the **visible** cells alive and **recycles** them — so a dequeued cell is *used*, never blank. A **diffable data source** replaces index math: you describe the *whole desired state* as a **snapshot of unique Hashable IDs**, apply it, and UIKit computes the inserts/deletes/moves.

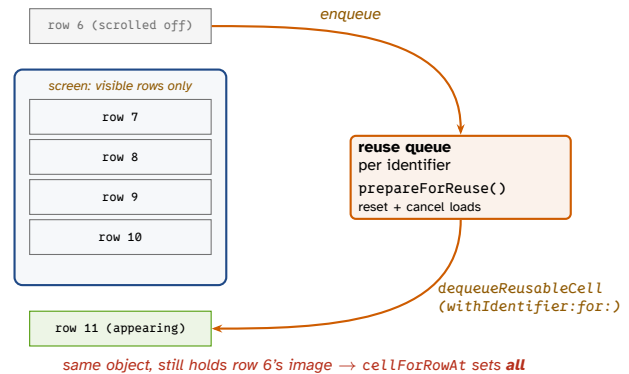
How it works

- **UITableViewDataSource** (*what to show*) — required: `tableView(_:numberOfRowsInSection:)`, `tableView(_:cellForRowAt:)`; optional `numberOfSections(in:)`. **UITableViewDelegate** (*behaviour*) — `didSelectRowAt`, `heightForRowAt`, `willDisplay`. Both held weak.
- **Reuse:** `register(_:forCellReuseIdentifier:)` once, then `dequeueReusableCell(withIdentifier:for:)` — always returns a cell (crashes if unregistered). Old `dequeueReusableCell(withIdentifier:)` may return `nil`. A cell leaving the screen goes to a per-identifier **reuse queue**; `prepareForReuse()` runs before it comes back.
- **Stale content** = a recycled cell still shows the last row's image/check/colour, because `cellForRowAt` set it only on some paths, or an async image load finished *after* reuse. Fix: set **every** property on every path; in `prepareForReuse` reset transient state + **cancel** the load; on completion check the cell still shows the same ID.
- **Manual updates:** change the model *first*, then `performBatchUpdates` with `insertRows/deleteRows`. Counts must add up or `NSInternalInconsistencyException` ("invalid number of rows").
- **UICollectionView** = same data source/delegate/reuse, plus a **layout** object: `UICollectionViewFlowLayout` or `UICollectionViewCompositionalLayout` (item → group → section); lists via `UICollectionViewLayoutListConfiguration` (iOS 14).
- **Diffable** (iOS 13): `UITableViewDiffableDataSource / UICollectionViewDiffableDataSource<SectionID, ItemID>`; both generic params **Hashable**; created with a cell provider closure. `NSDiffableDataSourceSnapshot` is a *struct*: `appendSections`, `appendItems(_:toSection:)`, `deleteItems`, `moveItem(_:beforeItem:)`, then `apply(_:animatingDifferences:)`. Edit the live one via `dataSource.snapshot()`. Map positions with `itemIdentifier(for:)` / `indexPath(for:)`.
- **Same ID, new content:** `reconfigureItems` (iOS 15) re-runs the config on the *existing* cell — cheap, keeps state. `reloadItems` (iOS 13) throws the cell away and dequeues a new one. Prefer `reconfigure` unless the cell *type* changes.
- **CellRegistration** (iOS 14): typed cell + config closure, no string IDs; `dequeueReusableCellUsingIdentifier(using:for:item:)`. Headers need `supplementaryViewProvider` (collection) or a subclass overriding `titleForHeaderInSection` (table).

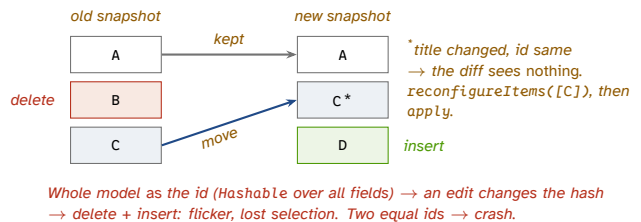
Example

```
enum Section: Hashable { case main }
let reg = UICollectionView.CellRegistration
<UICollectionViewListCell, Todo.ID> { cell, _, id in
    var c = cell.defaultContentConfiguration()
    c.text = store[id]?.title // read LIVE data by id
    cell.contentConfiguration = c // set EVERY property
}
ds = UICollectionViewDiffableDataSource<Section, Todo.ID>(
    collectionView: cv) { cv, ip, id in
    cv.dequeueReusableCell(using: reg, for: ip, item: id)
}
var snap = NSDiffableDataSourceSnapshot<Section, Todo.ID>()
snap.appendSections([.main])
snap.appendItems(todos.map(\.id)) // ids UNIQUE
ds.apply(snap, animatingDifferences: true)
```

Picture 1 — the reuse pool



Picture 2 — snapshot diff: IDs, not content



Interview traps

- ! **"Diffable detects content changes."** No — it diffs **identifiers** only. Same ID + new title = no update until `reconfigureItems/reloadItems`.
- ! **Duplicate IDs** anywhere in the snapshot (across sections too) → runtime crash. Never use a display name as the ID.
- ! Caching an `IndexPath` across an `apply` — positions shift; convert to the ID with `itemIdentifier(for:)` immediately.
- ! Config closure capturing a model copy → `reconfigure` shows old data. Look it up by ID.
- ! Apply from main or always the same background queue — never mix.
- ! `prepareForReuse` is not the place to set content — that is `cellForRowAt`.

Remember

"The pool gives you a *used* cell — set everything, assume nothing." **"Snapshot = IDs; content = your store; reconfigure = repaint."**

Likely questions

1. Why do old images appear? — reuse; reset/cancel in `prepareForReuse`, set all in `cellForRowAt`.
2. What must identifiers be? — **Hashable** and **unique** in the snapshot.
3. Reconfigure vs reload? — same cell re-configured vs new cell dequeued.
4. What did diffable replace? — `performBatchUpdates` index math + its crashes.