

In one line: Every constraint is one **linear equation** $\text{item1.attr} \{=, \leq, \geq\} \text{multiplier} \times \text{item2.attr} + \text{constant}$ — the engine solves them all so each view gets exactly **x, y, width, height**. Too few = *ambiguous*; contradictory *required* ones = *unsatisfiable*. **Priorities** decide who yields: **1000 / 750 / 250**.

How it works

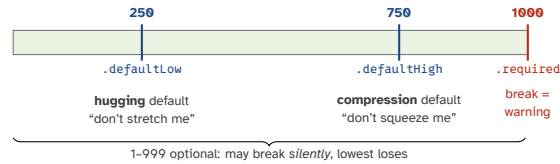
- `translatesAutoresizingMaskIntoConstraintsIntoConstraints = false` on every **code-created** view — else UIKit turns the autore-sizing mask into constraints that fight yours (wrong place / zero size / conflict log). IB sets it for you. Then `NSLayoutConstraint.activate([...])`.
- **anchors:** leading/trailing (flip for RTL — prefer them over left/right), top/bottom, centerX/Y, width/height, firstBaseline. Trailing/bottom constants are **negative** (-16).
- **Intrinsic content size:** UILabel, UIButton, UIImageView, UISwitch know their natural size, so position constraints alone suffice. Plain UIView has none. Custom: override `intrinsicContentSize`, call `invalidateIntrinsicContentSize()` when it changes.
- **Hugging** (default **250**) = resist growing *beyond* intrinsic size. **Compression resistance** (default **750**) = resist shrinking *below* it. Per axis. Lower value **loses**: stretches first / truncates first.
- **UILayoutPriority:** `.required = 1000`, `.defaultHigh = 750`, `.defaultLow = 250`, `.fittingSizeLevel = 50`. <1000 = optional: broken *silently*, lowest first. Tie at 750 vs 750 → arbitrary; nudge one to 749 / 251 to make it deterministic.
- **Unsatisfiable** (“Unable to simultaneously satisfy constraints”) = ≥ 2 **required** constraints contradict (leading 0 + width 300 + trailing 0 in 320 pt). The engine breaks one — “Will attempt to recover by breaking...” — which one is arbitrary. Fix: drop one below 1000. Debug: symbolic breakpoint `UIViewAlertForUnsatisfiableConstraints`, the View Debugger, `hasAmbiguousLayout`.
- **Safe area:** pin content to `view.safeAreaLayoutGuide`. Superview edges put text under the notch / Dynamic Island and buttons under the home indicator (the system gesture eats the tap). Full-bleed *backgrounds* may go to the real edges.
- **Size classes:** two axes (horizontal, vertical) × two values (`.compact`, `.regular`). iPhone portrait = `wChR`; iPhone landscape = `wChC` (Max/Plus: `wRhC`); iPad full screen = `wRhR`; iPad narrow split = `wC`. Read `traitCollection.horizontalSizeClass`; react on trait change (swap constraints / stack axis).
- **Animate** a constraint: change `.constant`, then `UIView.animate { superview.layoutIfNeeded() }`.

Example — a title + price row

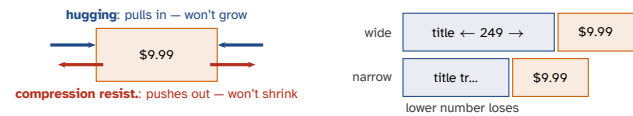
```
let row = UIStackView(arrangedSubviews: [title, price])
row.axis = .horizontal; row.spacing = 8 // direction, gap
row.distribution = .fill // sizing ALONG the axis
row.alignment = .firstBaseline // position ACROSS the axis
row.translatesAutoresizingMaskIntoConstraints = false // #1 bug
view.addSubview(row)
let m = view.layoutMarginsGuide // inset, inside the safe area
NSLayoutConstraint.activate([ // height: intrinsic size
    row.topAnchor.constraint(equalTo: m.topAnchor),
    row.leadingAnchor.constraint(equalTo: m.leadingAnchor),
    row.trailingAnchor.constraint(equalTo: m.trailingAnchor)])
title.setContentHuggingPriority(.defaultLow - 1, for: .horizontal)
title.setContentCompressionResistancePriority(.defaultHigh - 1, for: .horizontal)
```

Result: title (249 / 749) stretches to fill *and* truncates first; price (250 / 750) stays compact and readable. `.fill` picks the stretcher by lowest hugging.

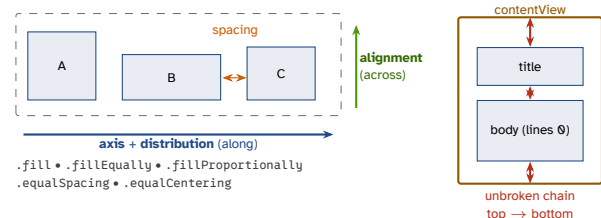
Picture — the priority scale



Picture — hugging vs compression



Picture — stack view • self-sizing cell



Cell: `tableView.rowHeight = UITableView.automaticDimension + tableView.estimatedRowHeight = 80 (non-zero) + the chain`; no `heightForRowAt`. Stack: add with `addArrangedSubview` (not `addSubview`); `isHidden` collapses a child.

Interview traps

- ! **Priority numbers** (Q25, blank): **1000 / 750 / 250**, and why **749**: a strict loser beats a random tie.
- ! **UIStackView** (Q19, blank): **axis, distribution, alignment, spacing**. Distribution = *along*, alignment = *across*.
- ! **Self-sizing cells** (Q23): *not* the delegate — `automaticDimension + estimatedRowHeight + chain`.
- ! Hugging is about growing past *content*, not about “edges” (Q8).
- ! Wrong place / zero size in code → first suspect translates... (Q7).
- ! Size classes = compact/regular per axis, not device sizes (Q24).

Remember

“A thousand is law, 750 guards the content, 250 lets it grow.”
Stack: **A-D-A-S** — **A**xis, **D**istribution, **A**lignment, **S**pacing. Cell: *auto + estimate + chain*.

Likely questions

1. Four facts per view? — x, y, width, height (constraints or intrinsic size).
2. Why translates...= false? — autore-sizing constraints clash with yours.
3. Label truncates instead of its neighbour? — lower its compression resistance.
4. Required conflict vs optional? — warning + arbitrary break vs silent, lowest loses.