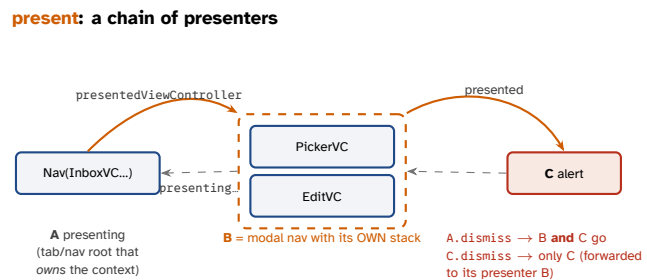
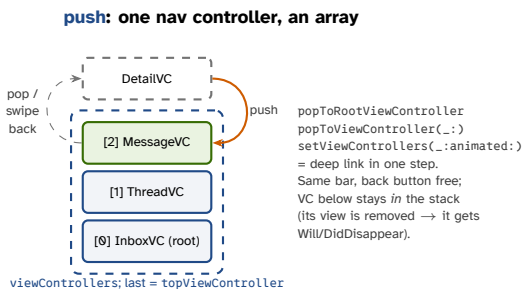


In one line: **Push** adds a VC to a UINavigationController's viewControllers array (drill-down, back button, pop). **Present** starts a separate **presentation chain** linked by presentedViewController / presentingViewController (a modal task you finish with dismiss). The modalPresentationStyle decides whether the presenter's view leaves the screen — and so whether it gets viewWillDisappear/viewWillAppear.



How it works

- Push** needs a container: navigationController is nil in a bare modal — present a UINavigationController wrapping it.
- Who presents?** UIKit walks up to the context-defining VC: from a child, the modal's presentingViewController is the nav/tab controller, not the child.
- Dismiss rule** (Apple): the *presenting* VC is responsible. Calling dismiss on the presented VC forwards to its presenter; calling it lower in the chain dismisses its presented child *and everything above*.
- One presented VC per presenter, presenter in the window — else a warning and **nothing** shows. visibleViewController includes a modal.
- Sheets** (iOS 15+): sheetPresentationController .detents = [.medium(), .large()] (.custom iOS 16) .prefersGrabberVisible; largestUndimmedDetentIdentifier keeps the presenter usable.
- Swipe-down:** isModalInPresentation = true blocks it → presentationControllerDidAttemptToDismiss. presentationControllerDidDismiss fires only for user (swipe) dismissal, never for your dismiss().

modalPresentationStyle

Style	Looks like	pres. dis-appears	swipe close
.automatic	iOS 13+ default; → .pageSheet	no	yes
.pageSheet	card; presenter shrinks behind	no	yes
.formSheet	centred box (iPad)	no	yes
.fullScreen	presenter view <i>removed</i>	yes	no
.overFullScreen	covers all, presenter stays	no	no
.overCurrentScreen	over a context-defining VC	no	no
.popover	iPad bubble; sheet on iPhone	no	tap out

Example — push, present, delegate back

```
protocol EditDelegate: AnyObject { func didSave(_ i: Item) }
class EditVC: UIViewController { weak var delegate: EditDelegate? }
class ListVC: UIViewController, EditDelegate {
    func open(_ i: Item) { // push: same stack
        navigationController?
        .pushViewController(DetailVC(i), animated: true) }
    func edit(_ i: Item) { // present: new chain
        let e = EditVC(i); e.delegate = self
        let nav = UINavigationController(rootViewController: e)
        nav.sheetPresentationController?.detents = [.medium()]
        nav.isModalInPresentation = true // no swipe-close
        present(nav, animated: true) }
    func didSave(_ i: Item) { // presenter closes it
        dismiss(animated: true); reload() } } // no viewWillAppear!
```

Related: app & VC lifecycle · Coordinator · retain cycles & weak · delegation vs closures · SwiftUI state · state restoration

SwiftUI

- NavigationStack(path: \$path) (iOS 16, replaces NavigationView): NavigationLink(value:) pushes a *value*; .navigationDestination(for: T.self) maps it — **once per type**, on a persistent ancestor (not inside a lazy List row).
- NavigationPath = type-erased Hashable list (mixed types); [Route] if one type. Push append, pop-to-root removeLast(path.count), deep link = assign the whole path. Codable routes → path.codable → @SceneStorage.
- Modals: .sheet(item:) (prefer over isPresented + separate state — stale item race), .fullScreenCover, .presentationDetents. @Environment(\.dismiss) closes the *nearest* context: inside a pushed view it **pops**, it does not close the sheet. One NavigationStack per tab; never nest stacks.

Deep links

URL scheme → scene(_:openURLContexts:); universal link → scene(_:continue:); on a **cold launch** both arrive in willConnectTo's connectionOptions instead. Parse to a route enum, then setViewControllers / assign path (SwiftUI: .onOpenURL).

Interview traps

- ! iOS 13 .pageSheet default: the list under a sheet gets **no** viewWillAppear on dismiss — refresh via delegate/closure.
- ! push from a VC presented without a nav → nil optional chain, silent no-op.
- ! Delegate: weak var + protocol: AnyObject (weak needs a class type, and is Optional because it auto-nils). Closures: [weak self].
- ! Coordinator must drop its child coordinator on the *back-swipe* pop too — detect it in UINavigationControllerDelegate.didShow.

Likely questions

- dismiss inside the modal? — forwarded to its presenter.
- A→B→C, A.dismiss? — B and C go; only the top animates.
- Block swipe-close? — isModalInPresentation = true.
- Deep link 3 levels? — setViewControllers / set path.