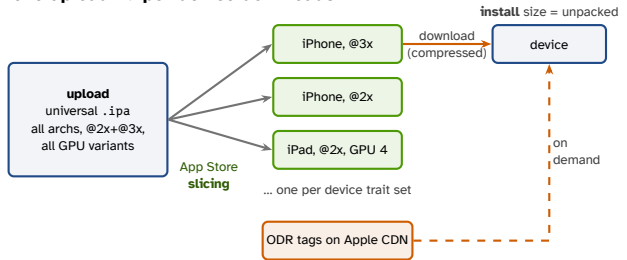


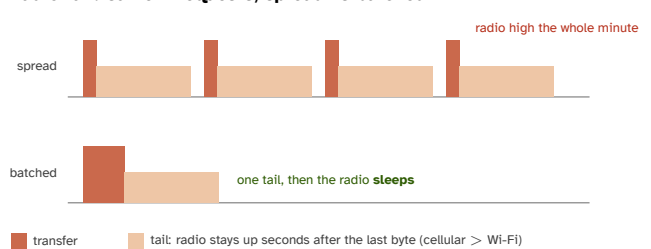
In one line: **Size:** the App Store *slices* one universal upload into per-device variants, **On-Demand Resources** move optional content out of the download, **bitcode is gone**. **Energy:** do less work, less often, in *bigger batches*, woken by events not timers, at the lowest fidelity the user accepts — the radio **tail**, GPS and wakeups cost more than bytes.

one upload → per-device downloads



cellular: **download** size vs the 200 MB prompt · loose files in Copy Bundle Resources ship to every variant

radio tail: same 4 requests, spread vs batched



bytes are cheap; *waking* the radio is not — the tail dominates small requests

App size — how it works

- **Slicing** — a variant per device: arch + asset-catalog traits (scale, idiom, Metal GPU family, memory class). Only **asset catalogs** slice.
- **Bitcode** — deprecated in Xcode 14; the App Store no longer takes it.
- **Three sizes** — *upload* (universal) · *download* (compressed variant; the cellular prompt checks it) · *install* (unpacked). Over cellular iOS asks above **200 MB** (iOS 13+; user-adjustable in Settings).
- **ODR** — tagged resources hosted by the App Store: *initial install*, *prefetched* (after install), *on demand*. `NSBundleResourceRequest(tags:)` + `beginAccessingResources`; the **live request** pins them — after `endAccessingResources`/deinit they are purgeable. Own CDN: **Background Assets** (iOS 16).
- **Measure** — Organizer export, *App Thinning: all variants* → App Thinning Size Report.txt (per variant, compressed + not); App Store Connect file sizes; `assetutil` on `Assets.car`.
- **Shrink** — catalog assets (HEIC, vectors), drop unused images/localizations; `-Osize`, dead-code stripping, symbols into the dSYM, fewer dynamic frameworks.

Energy — what drains

- **Radio tail** (diagram) — every wake of cellular/Wi-Fi pays seconds of high power. Polling every 10 s = the radio never sleeps; push (APNs rides one shared system connection) = the OS pays.
- **CPU wakeups** — timers, polling loops, chatty background work stop deep idle. `timer.tolerance ≥ 10%` lets iOS coalesce. Off-main ≠ cheaper: same joules.
- **GPS** — continuous `kCLLocationAccuracyBest` is among the worst. Least accuracy you need, `distanceFilter`, one-shot `requestLocation()`, significant-change (~ 500 m, ≥ 5 min) or region monitoring, **stop** when the screen closes. “When In Use” can still drain: Idle + accuracy decide, not the permission.
- **Display + GPU** — `isIdleTimerDisabled` left on, needless 120 Hz animation, offscreen passes.

Energy — what to do

- **Batch** — buffer analytics, prefetch in one burst, flush on Wi-Fi/charging. `isDiscretionary = true`: iOS picks the mo-

ment (automatic for transfers started in the background). Respect Low Data Mode (`allowsConstrainedNetworkAccess`).

- **Low Power Mode** — pause prefetch, autoplay, decoration; never break what the user just asked for. Read `ProcessInfo.processInfo.isLowPowerModeEnabled`, observe `NSProcessInfoPowerStateDidChange`.
- **Thermal** — `thermalState.nominal/.fair/.serious/.critical`: at `.serious` cut frame rate, quality, effects.
- **Measure** — Xcode *Energy Impact* gauge; Instruments **Energy Log** (on device); field: **MetricKit** (iOS 13) `MXMetricPayload` `~daily` — `cpuMetrics`, `locationActivityMetrics`, `networkTransferMetrics`, `cellularConditionMetrics`, `MXCPUExceptionDiagnostic`; Organizer → Energy.

Example

```
let req = NSBundleResourceRequest(tags: ["level-3"])
req.beginAccessingResources { error in // keep 'req' alive
    if let error { return showRetry(error) }
    loadLevel3() } // later: req.endAccessingResources()
let cfg = URLSessionConfiguration
    .background(withIdentifier: "sync")
cfg.isDiscretionary = true // iOS picks the moment
cfg.allowsExpensiveNetworkAccess = false // no cellular
let pi = ProcessInfo.processInfo
if pi.isLowPowerModeEnabled
    || [.serious, .critical].contains(pi.thermalState) {
    prefetcher.pause() } // shed optional work
```

Interview traps

- ! “Enable bitcode for thinning” — removed; slicing + ODR remain.
- ! ODR read by bundle path without a live request — maybe never downloaded, or purged. ODR moves *when* bytes arrive: loading UI, retry, offline.
- ! “One small request can’t matter” — tail energy. Batch it.

Remember

Slice, tag, strip. Batch, defer, coarsen, stop.

Likely questions

1. Download vs install size? — compressed variant vs unpacked.
2. Keep ODR from purging? — hold the request object.
3. Real-user energy data? — MetricKit, Organizer.