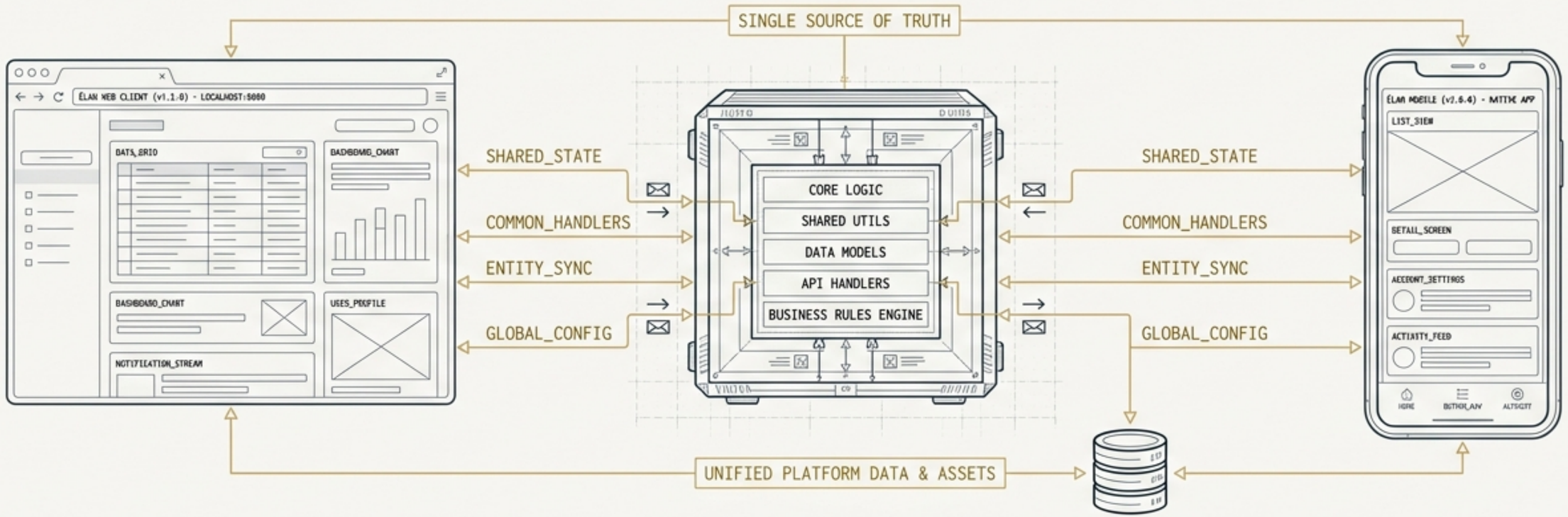
THE REALITY OF CODE REUSE

Architectural lessons from migrating ÉLAN from Web to Mobile



SHARED CODE (%)

CORE LOGIC: 92%

UTILS: 85%

MODELS: 98%

UI COMPONENTS: 40% ⚠ UI ADAPTATION REQUIRED

ARCHITECTURAL COMPONENTS

CROSS-PLATFORM BRIDGE
STATE MANAGEMENT
NETWORK LAYER
LOCAL STORAGE
DIAGNOSTICS

✓ SUCCESSFUL INTEGRATION

⚠ PLATFORM-SPECIFIC ADAPTATION NEEDED

MIGRATION METRICS

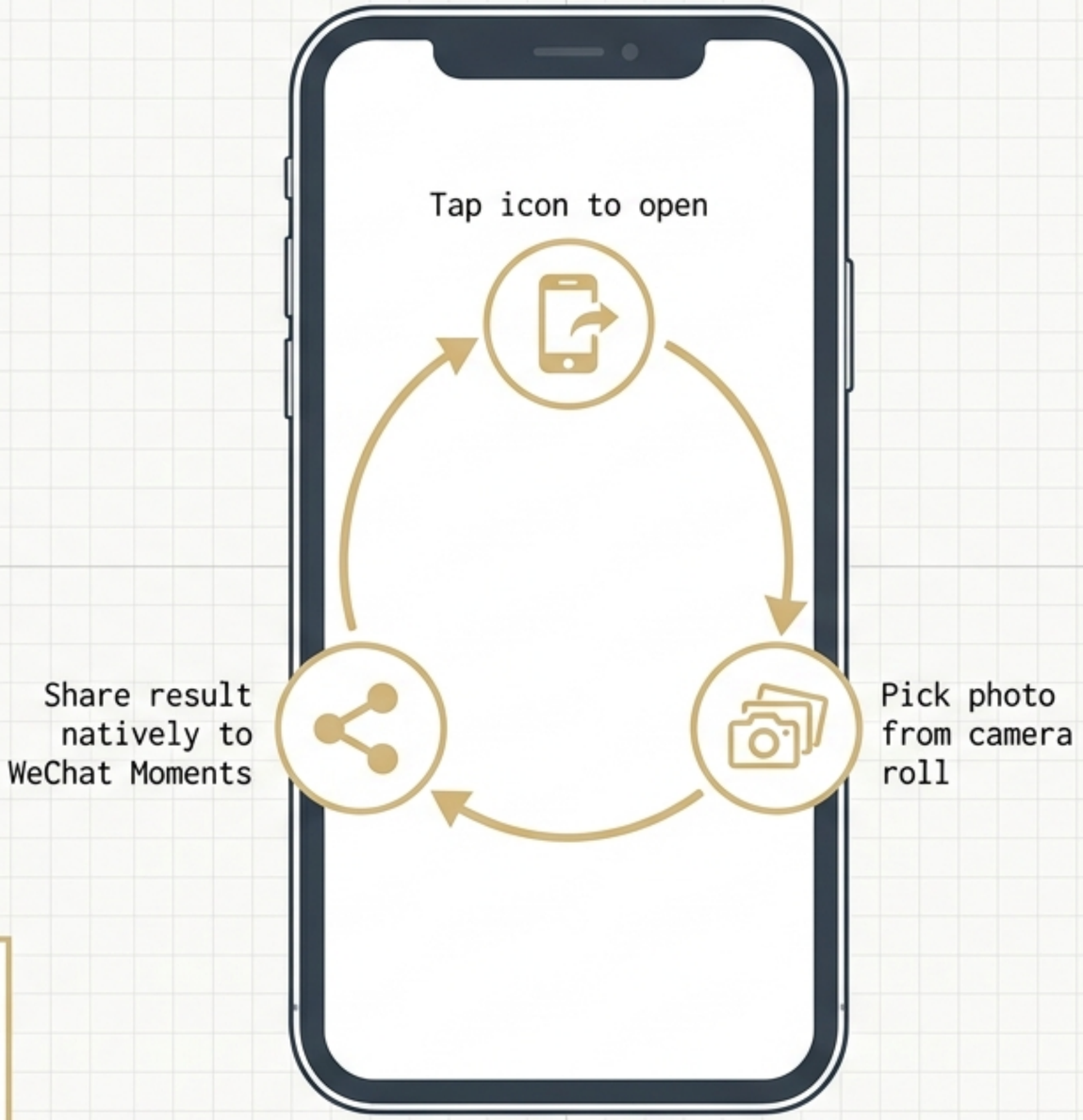
DEVELOPMENT VELOCITY:	+35% ↗
BUG REDUCTION:	-40% ↘
MAINTENANCE EFFORT:	-25% ↘
TIME-TO-MARKET:	-30% ↘

THE CORE USE CASE IS STRICTLY MOBILE-NATIVE

LEGACY WEB DASHBOARD – SECONDARY FOCUS

Target users — 18-35 year old women who want effortless social photos — live on their phones. Photos are taken on the phone, edited on the phone, and posted from the phone.

DEVICE DEPENDENCY: HIGH | WORKFLOW EFFICIENCY: OPTIMIZED



PRIMARY PLATFORM – MOBILE FIRST WORKFLOW

Why a Progressive Web App fails the ÉLAN core loop

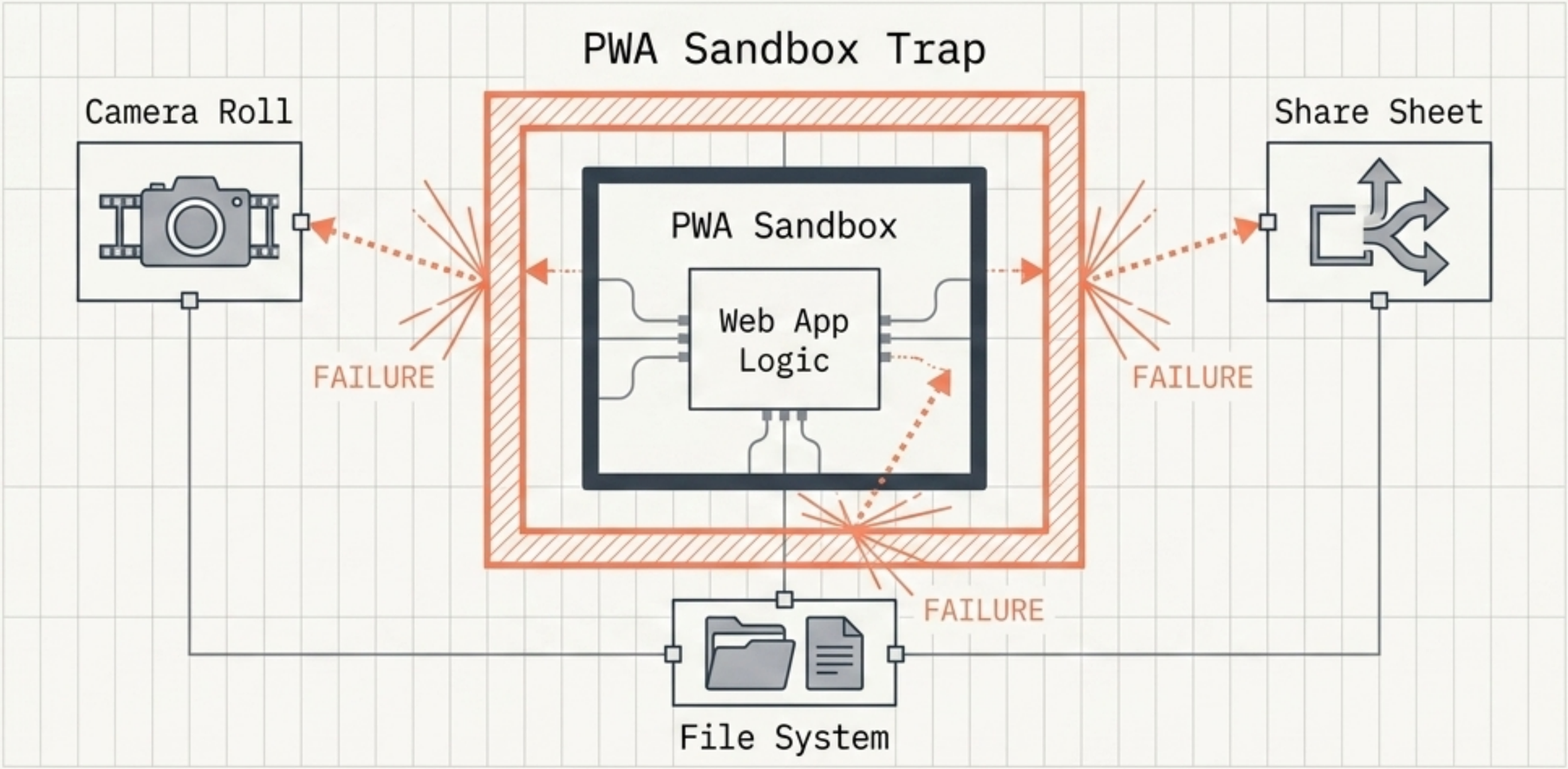


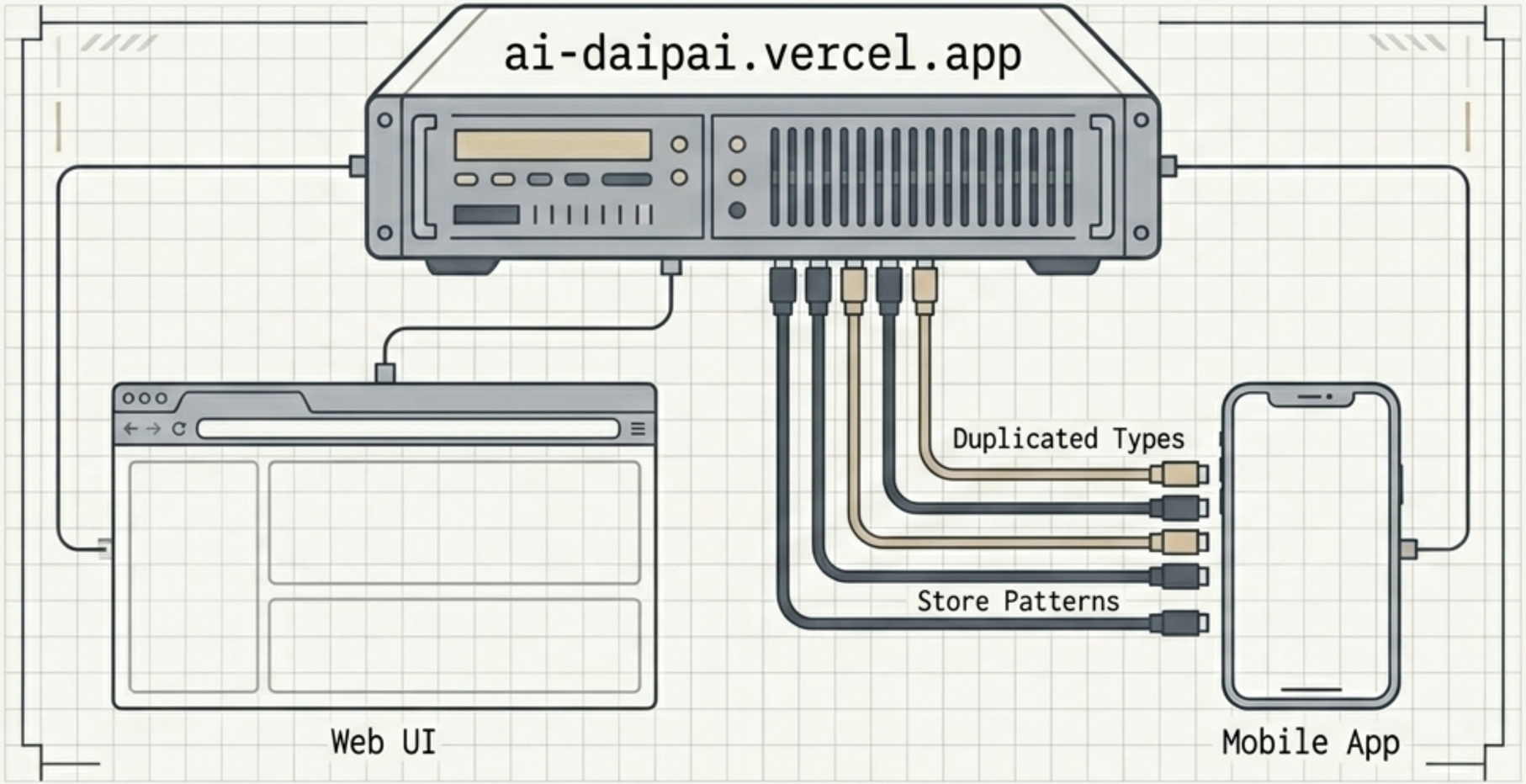
Photo Access	Persistence	Distribution
Sandboxed and clunky on iOS.	No reliable way to save to the user's album.	Native share sheet integration does not exist.

The stack decision matrix for a solo developer

Framework	Performance	Existing TS Codebase Reuse	Solo Developer Viability
Flutter	Great	Zero (Requires Dart rewrite)	Low
Native (Swift/Kotlin)	Best	None (3 separate codebases)	Zero
React Native (Expo SDK 55)	Excellent	Shared TypeScript & Zustand	High

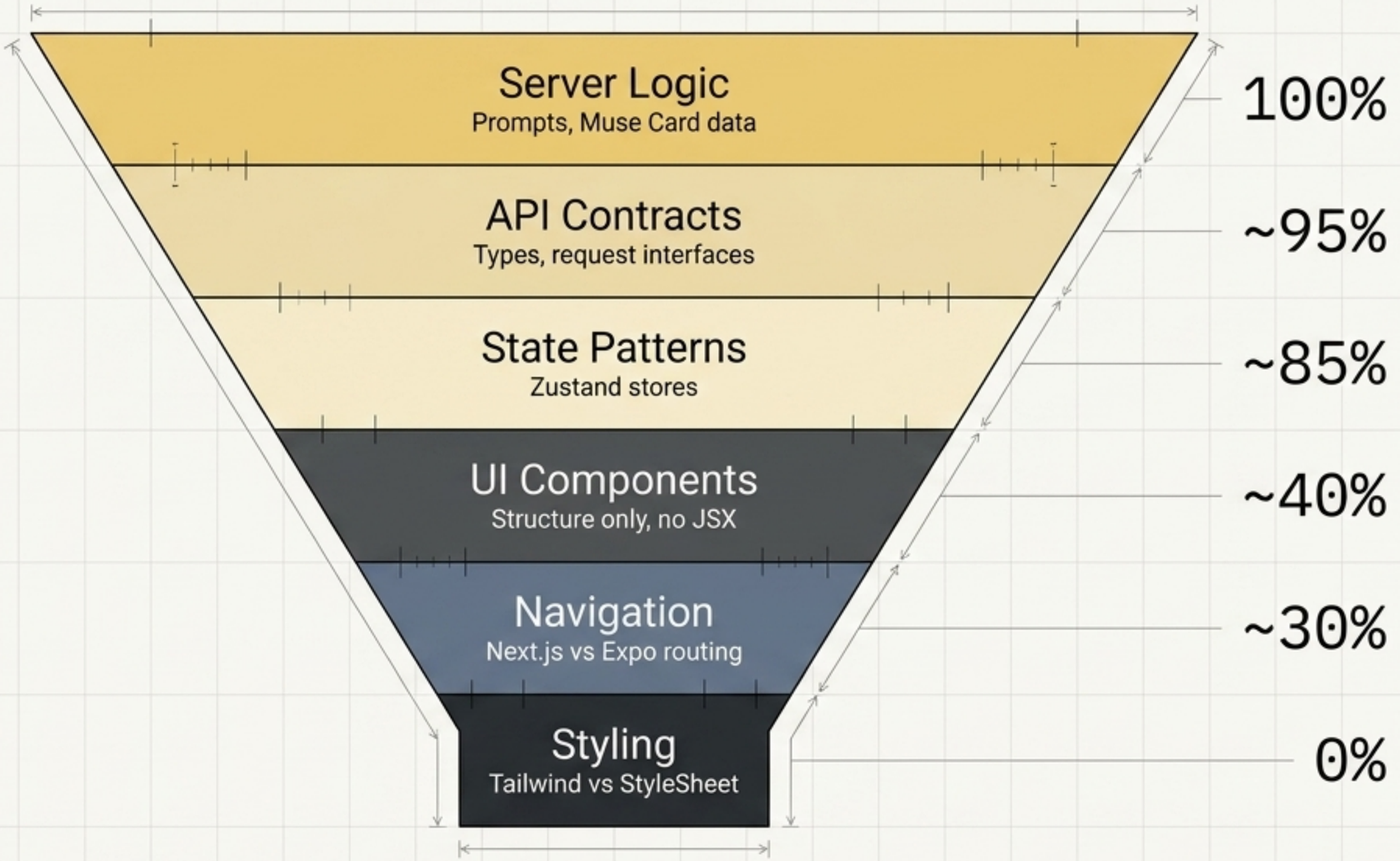
DECISION NOTE: EXPO SDK 55 PROVIDES OPTIMAL BALANCE OF PERFORMANCE AND CODE REUSE FOR SOLO TS DEVELOPER.

API-first architecture over premature monorepo overhead

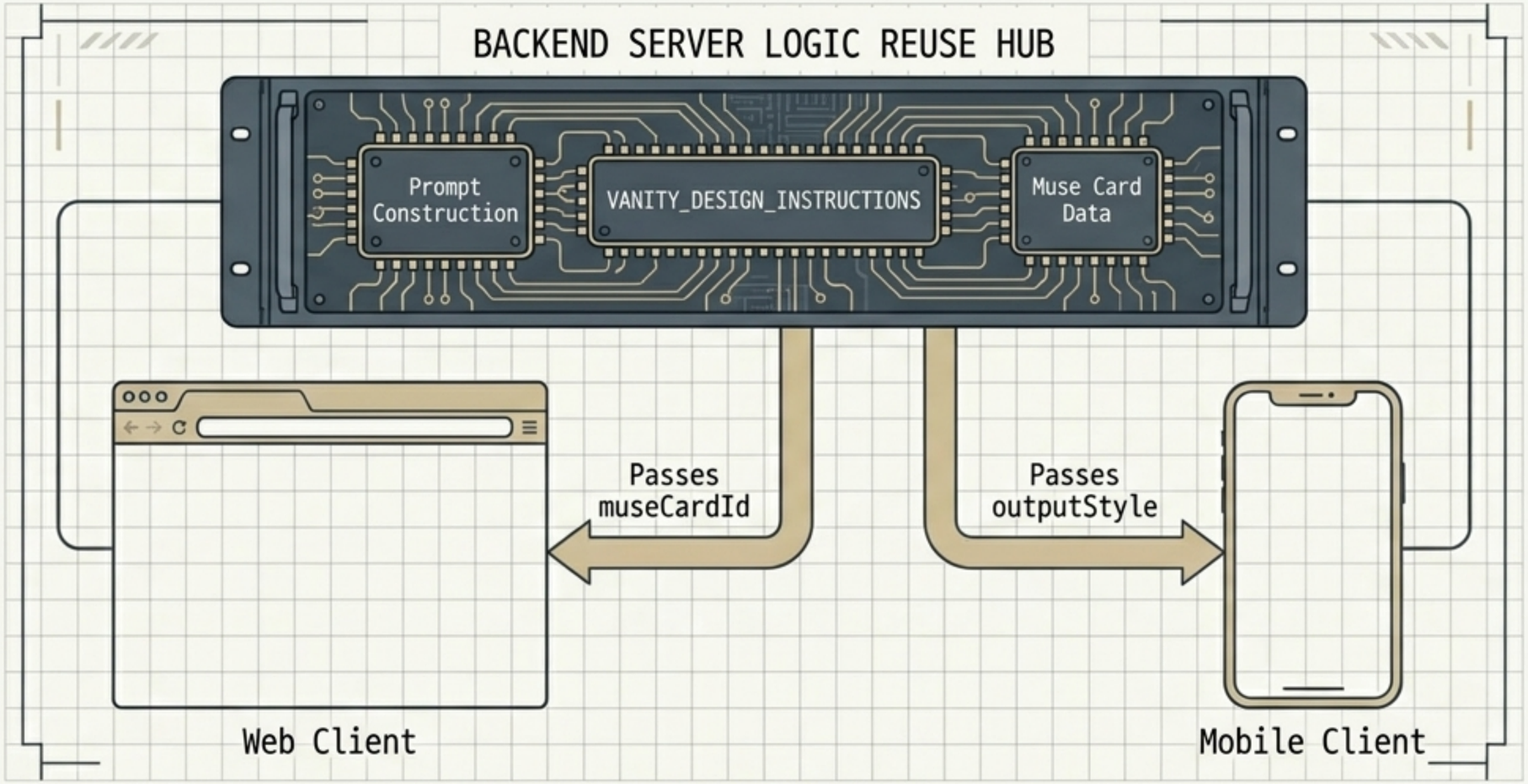


For a single mobile consumer, maintaining cross-package imports and tsconfig aliases was deferred in favor of pragmatic type duplication. The mobile app simply consumes the web app's API.

The true code reuse stratification funnel

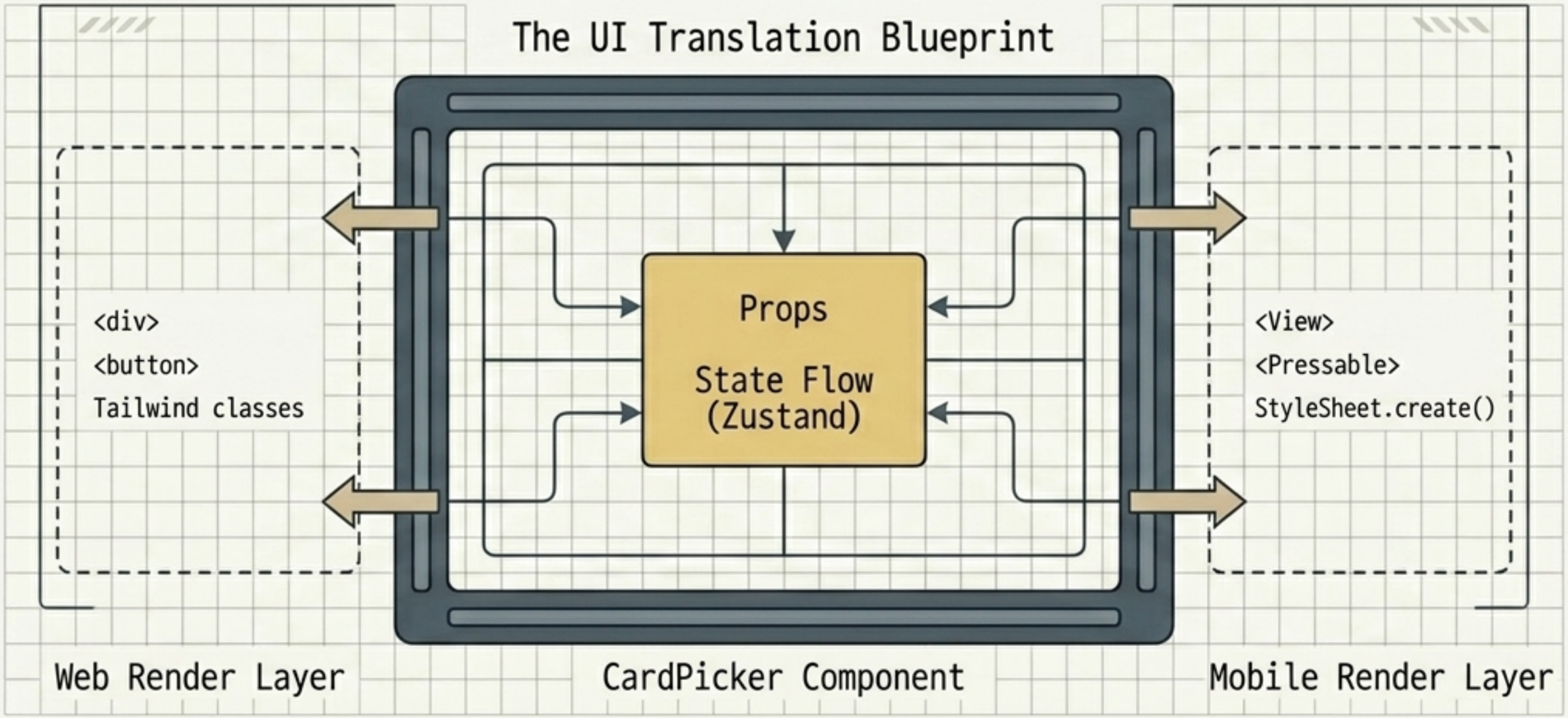


Hoarding complexity on the server yields 100% logic reuse



The mobile app does not construct prompts. It passes IDs to the API. Every update to the generation pipeline is instantly deployed to both platforms for free.

Component structure transfers, rendering does not



Not a single line of JSX is shared, but the component decomposition maps perfectly 1:1.

The cross-platform friction heatmap

Just Works

Expo Native APIs

expo-image-picker, expo-sharing. Zero Xcode config.

Zustand

Identical React Native API; exact same create() pattern.

EAS Build

15-minute cloud builds for both .ipa and .apk without touching IDEs.

Full Custom Hack Required

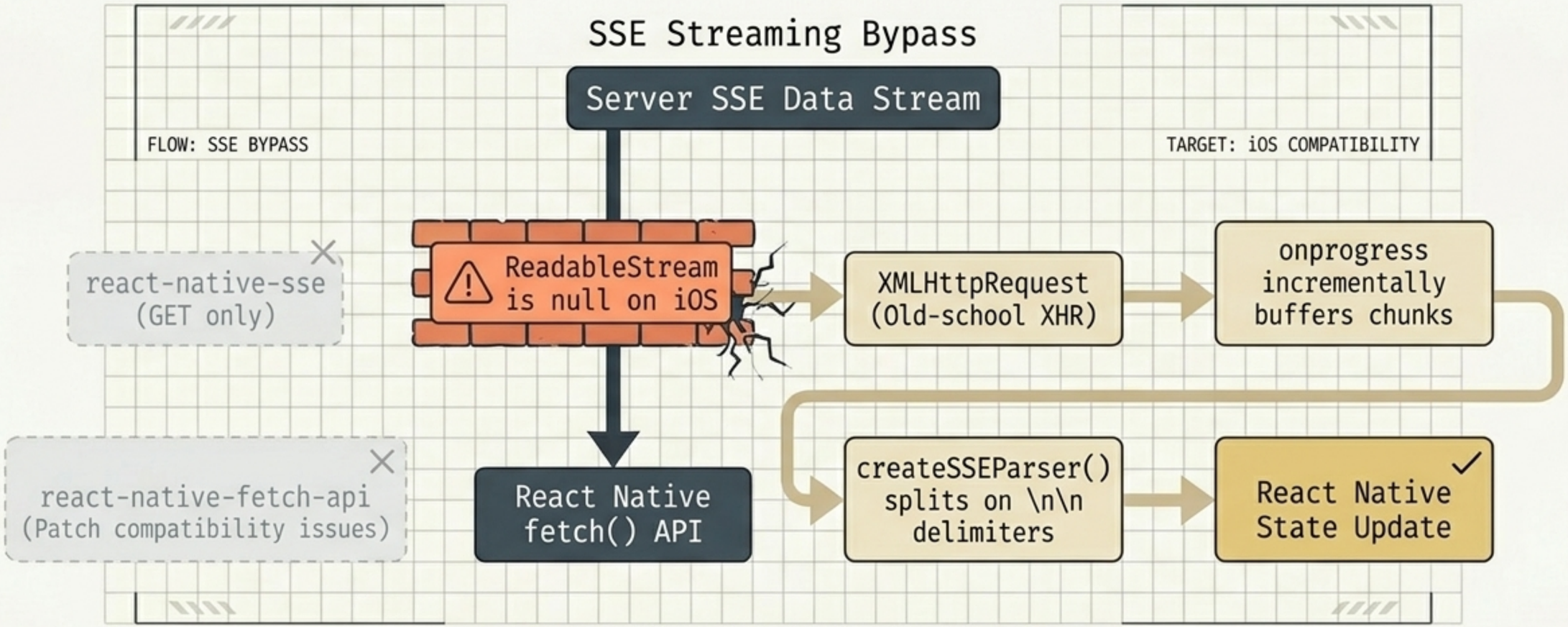
Touch Targets

Apple HIG demands 44px minimums vs web's py-1.5 (~28px). Requires tedious component-wide adjustments.

Safe Area Insets

Status bar, notch, and home indicator padding required manual useSafeAreaInsets() intervention on every single screen.

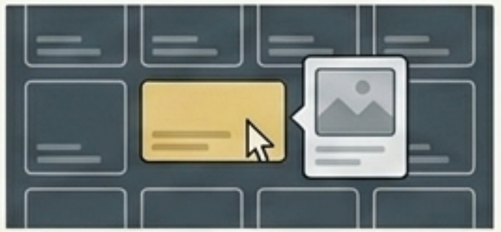
Overcoming the React Native streaming limitation



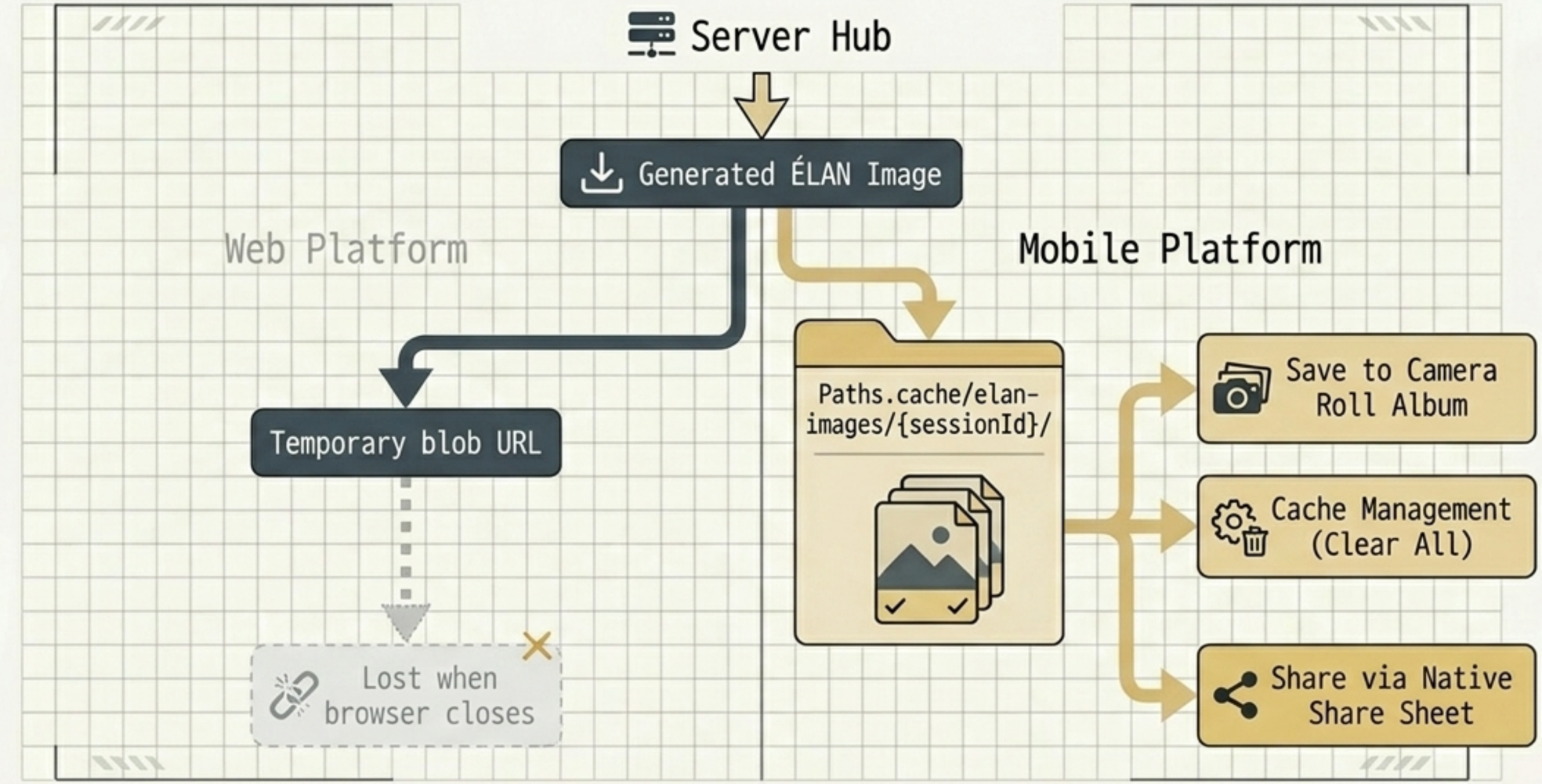
Not a single line of JSX is shared, but the component decomposition maps perfectly 1:1.

Adapting interaction mechanics to platform expectations

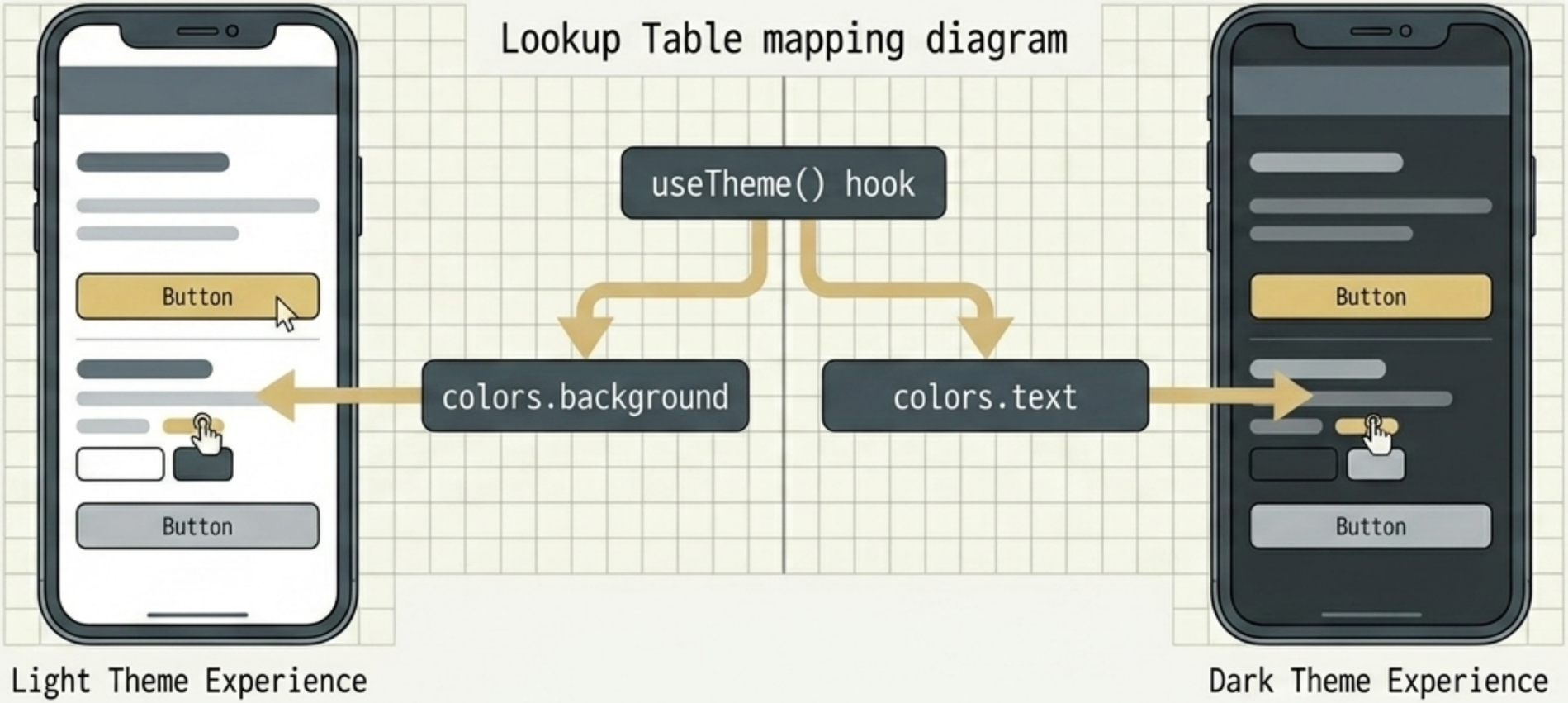
Platform Interaction Matrix

	Web Experience	Mobile Experience
Photo Selection	 Drag-and-drop zone	 expo-image-picker with compact, horizontal camera roll preview row
Muse Card Browsing	 Hover-to-preview responsive grid	 Scrollable grid with long-press-to-preview slide-up modal
Inspiration Matching	Requires manual selection tap after AI analysis.	Auto-selects matched card and collapses picker to a compact summary row.

Deepening the loop via native device persistence



The two-hour dark mode surprise



Expo SDK 55's userInterfaceStyle: 'automatic' and React Navigation's ThemeProvider bypassed all conditional className logic and CSS variables required on Web, establishing full dark mode parity in two hours.

Architecture pivot points: What to change if starting over

1

Sequence Pivot

Start mobile-first.

Mobile has deeper integration constraints (camera roll, caching). Build Expo first, add a web dashboard later.

2

Monorepo Pivot

Implement a shared types package.

Maintaining manual duplication across `src/types/` and `mobile/src/lib/types.ts` is manageable but creates friction. A `packages/types/` workspace pays for itself by week two.

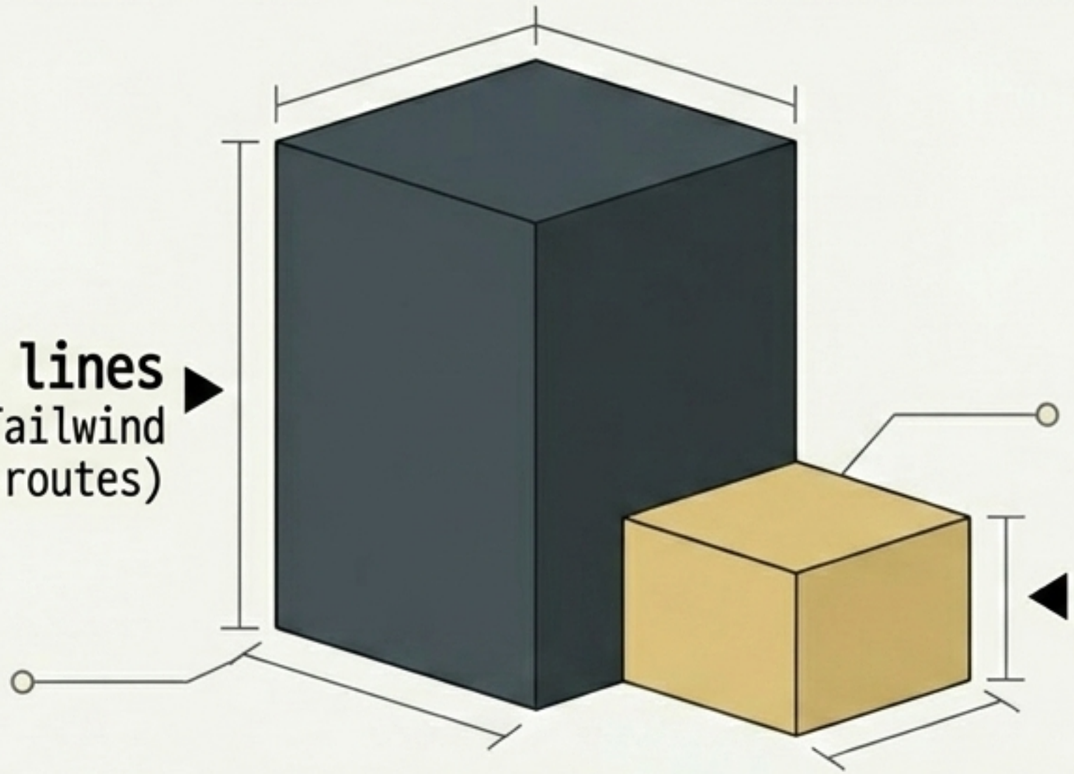
3

Styling Pivot

Skip explicit StyleSheet for Tamagui.

Manual touch-target scaling and padding is too much labor. A component library with built-in theme tokens and web+native parity is the right bet.

The 35% Reality: Clients are just windows



Synthesis overlay

Mobile is significantly simpler because it is a thin client. True cross-platform velocity isn't about sharing UI rendering code—it's about hoarding all the intelligence, prompt construction, and complexity on the server.