FLUTTER 3.44 / ANDROID RENDERING

Hybrid Composition++ (HCPP) in Flutter

A visual deep dive into the new Android Platform View pipeline

WHY THIS MATTERS

HCPP lets Flutter and native Android views render into separate native surfaces, then hands final composition to Android. The goal: strong native fidelity without the same performance penalty of traditional Hybrid Composition.

API 34+ | Vulkan | Impeller | Experimental

Inside this issue

Architecture • SurfaceControl • SurfaceFlinger • Setup • Benchmarking • Use cases

The Platform View problem

Flutter renders most UI through its own graphics pipeline, while components such as WebView, Google Maps, camera previews, video surfaces and SDK-provided Android controls are real native Android Views. C me h

TEXTURE LAYER

Native View -> texture -> Flutter scene

Good Flutter transforms, but fast scrolling and some SurfaceView/accessibility scenarios can suffer.

HYBRID COMPOSITION

Native View stays in Android view hierarchy

Excellent native fidelity, but tighter platform/raster synchronization can reduce Flutter frame performance.

HCPP

Flutter surface + native surface -> Android compositor

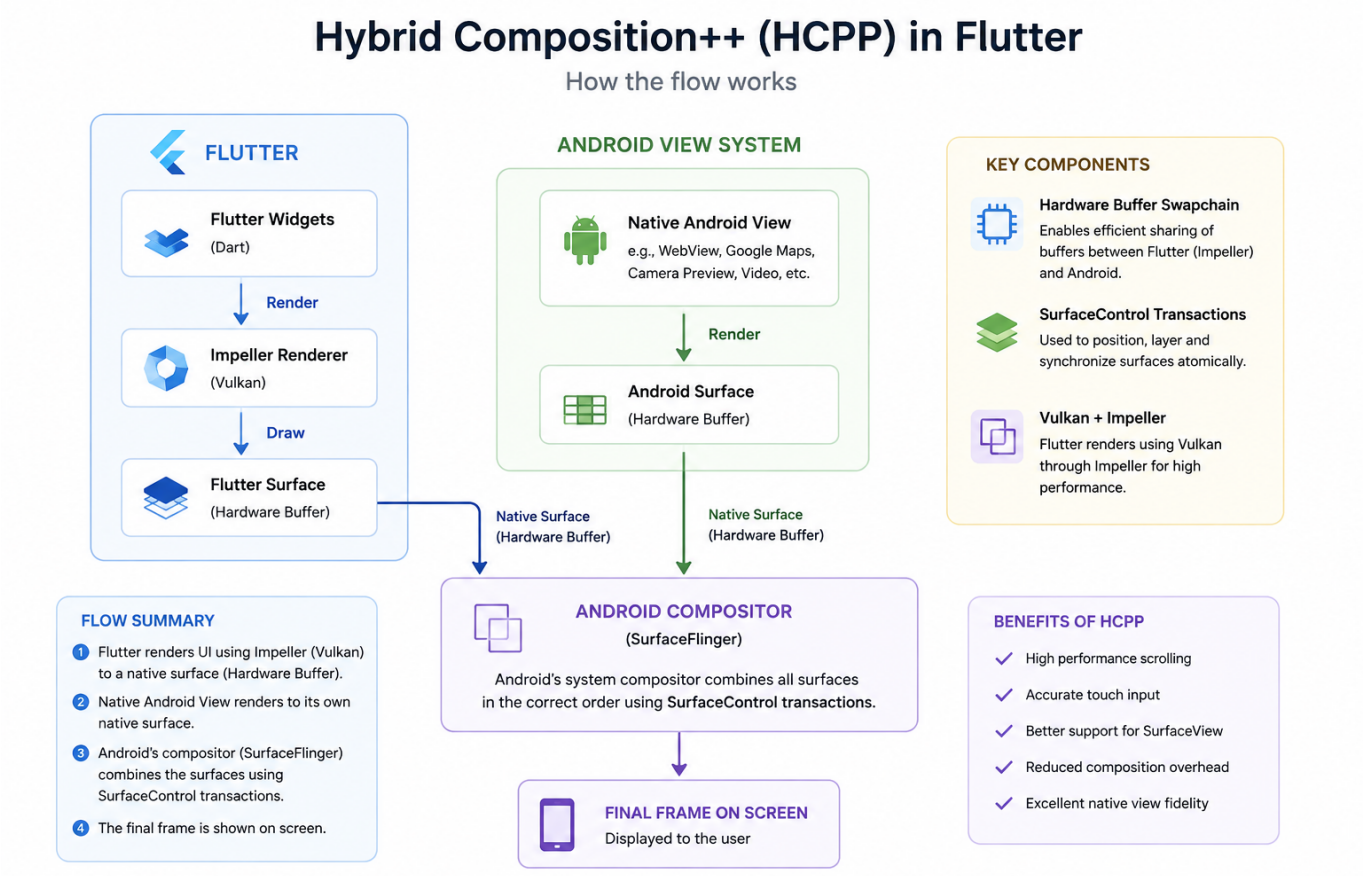
Designed to preserve native fidelity while reducing composition overhead on supported devices.

KEY IDEA

HCPP is not a new Flutter widget API. It is a lower-level backing upgrade for Android Platform Views. Existing Platform View integrations can use the newer composition mechanism when HCPP is enabled and the device is eligible.

How the HCPP flow works

The important architectural change is who performs final composition. Flutter and native Android content keep separate native surfaces; Android then combines them into the final frame.



4-STEP SUMMARY

1. Flutter widgets render through Impeller using Vulkan into a native Flutter surface.
2. The Android Platform View renders into its own native Android surface.
3. SurfaceControl transactions coordinate position, order and synchronization.
4. Android's compositor (SurfaceFlinger) combines the surfaces and presents the final frame.

Enable HCPP - then measure it

HCPP is experimental in Flutter 3.44. Treat adoption as a performance experiment: enable it, verify compatibility, benchmark real hardware and test fallback behavior.

Local testing

```
flutter run --profile --enable-hcpp
```

Production opt-in

```
<meta-data
  android:name="io.flutter.embedding.android.Enable
  android:value="true" />
```

Current eligibility

- ✓ Android API 34 or newer
- ✓ Vulkan-capable device
- ✓ Impeller rendering path
- ✓ Fallback available on unsupported devices

FRAME PACING

- Dropped frames
- Raster/UI duration
- Scroll smoothness

INTERACTION

- Touch accuracy
- Keyboard/focus
- Gesture conflicts

PLATFORM BEHAVIOR

- TalkBack
- Lifecycle
- Transparent overlays

DEVICE COVERAGE

- Android 14+
- Different GPUs
- Mid-range hardware

Where HCPP can matter most

WebView

Fast scrolling, forms, authentication and embedded web modules.

Google Maps

Panning, zooming and native gestures alongside Flutter overlays.

Camera & video

High-frequency native surfaces with Flutter controls on top.

SDK UI

Payments, ads and enterprise SDKs that expose native Android Views.

Quick FAQ

Is HCPP enabled by default?
No. In Flutter 3.44 it is experimental and opt-in.

Do I need new Platform View APIs?
Usually no. HCPP changes the backing composition strategy.

Will older Android devices break?
No. Flutter can fall back to the existing Platform View strategy.

Should I enable it blindly?
No. Benchmark on physical devices and test overlays, accessibility and lifecycle behavior.

Official references

Flutter - What's new in Flutter 3.44: flutter.dev/blog/whats-new-in-flutter-3-44

Flutter Docs - Android Platform Views: docs.flutter.dev/platform-integration/android/platform-views

Note: HCPP is experimental. Verify the latest Flutter documentation before production rollout.