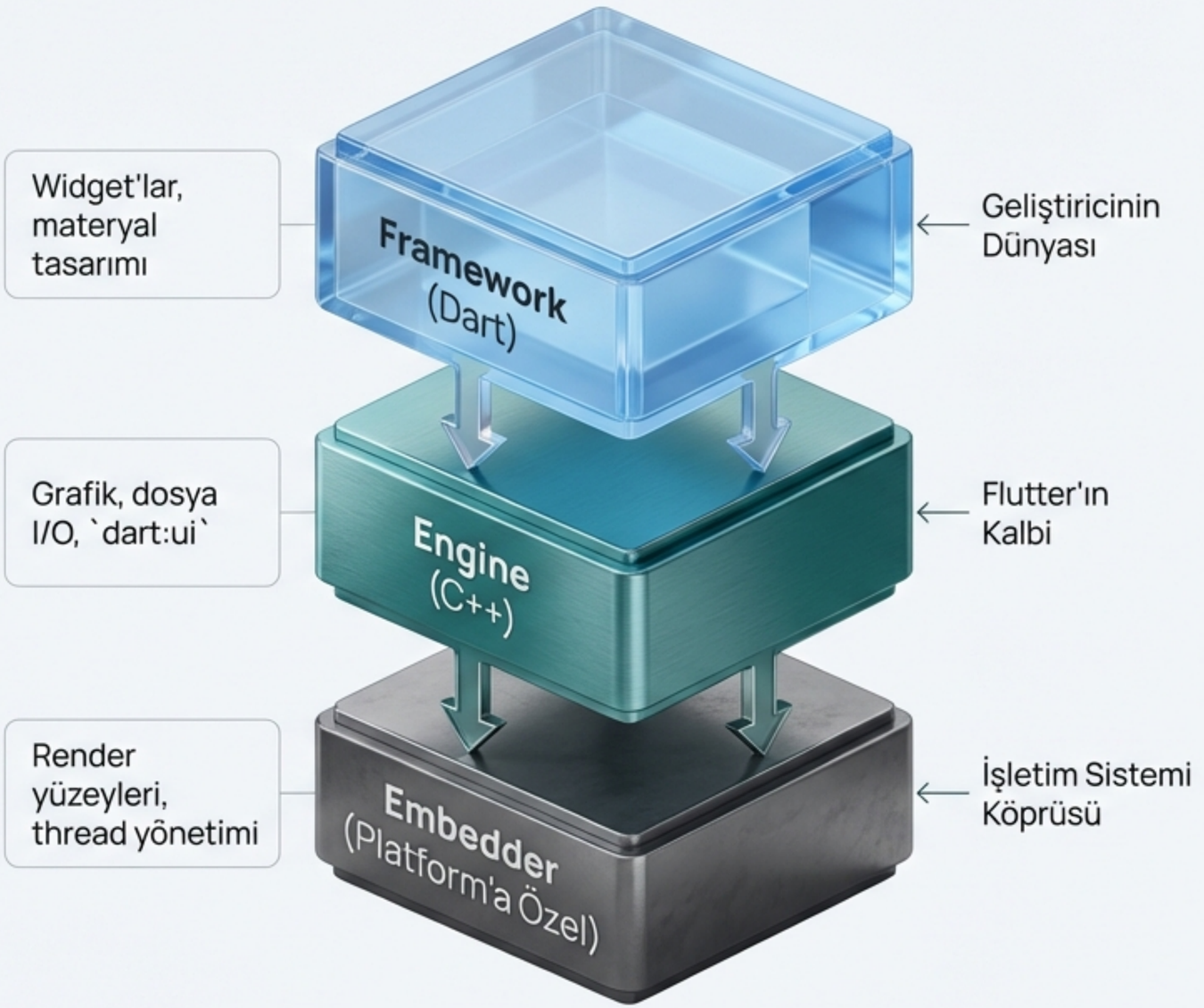
Flutter’ın Kalbine Yolculuk

Koddan Piksele: Flutter'ın Mimarisi ve Render Mekanizması

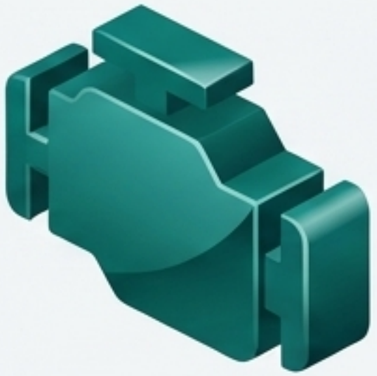
Flutter Bir Katmanlı Sistemdir

Flutter'ın gücü, her biri bir alttakine bağımlı olan üç temel katmandan oluşan mimarisinden gelir. Bu yapı, **esneklik ve platformlar arası tutarlılık** sağlar.
Geliştiriciler olarak biz en üst katmanda yaşarız, ancak sihrin gerçekleştiği yer daha derin katmanlardır.



Makine Dairesi: Engine ve Embedder

The Engine - Taşınabilir Çekirdek



Flutter'ın kalbidir ve çoğunlukla C++ ile yazılmıştır. Uygulamalarınızla birlikte paketlenen taşınabilir bir çalışma zamanıdır (runtime). Skia grafik motoru, ağ ve dosya I/O gibi temel kütüphaneleri içerir. `dart:ui` kütüphanesi aracılığıyla Dart dünyasına açılır.

Önemli Not: Web için bu yapı farklıdır: C++ motoru yerine, Dart kodu `dart2js` ile JavaScript'e derlenir ve HTML/CSS veya WebGL (CanvasKit) ile render edilir.

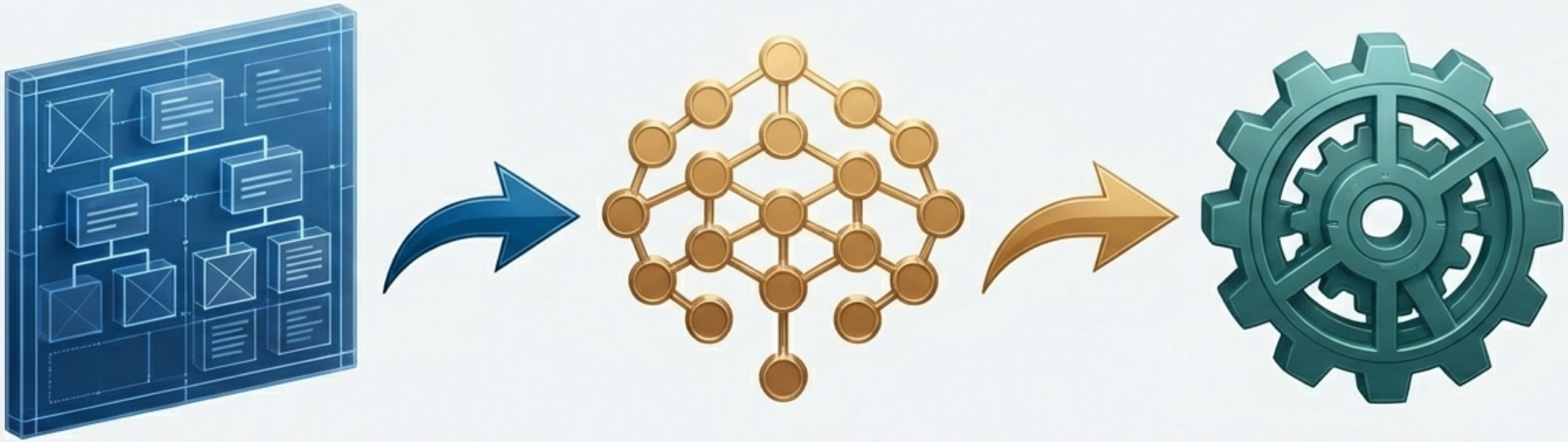
The Embedder - Platform'a Özel Ev Sahibi

Flutter içeriğinizi işletim sisteminde "barındıran" native uygulamadır. Uygulama başlatıldığında giriş noktasını (entrypoint) sağlar, UI ve render için thread'leri alır ve Flutter motorunu başlatır.

- iOS/macOS: Objective-C++
- Android: Java / C++
- Windows/Linux: C++

Bir Widget’tan Fazlası: Flutter’ın Üç Ağaç Mimarisi

Geliştirici olarak yazdığımız Widget ağacı, buzdağının sadece görünen kısmıdır. Flutter, performansı optimize etmek için perde arkasında paralel olarak iki ağaç daha yönetir. Bu üçlü yapı, Flutter’ın verimli render yeteneğinin sırrıdır.



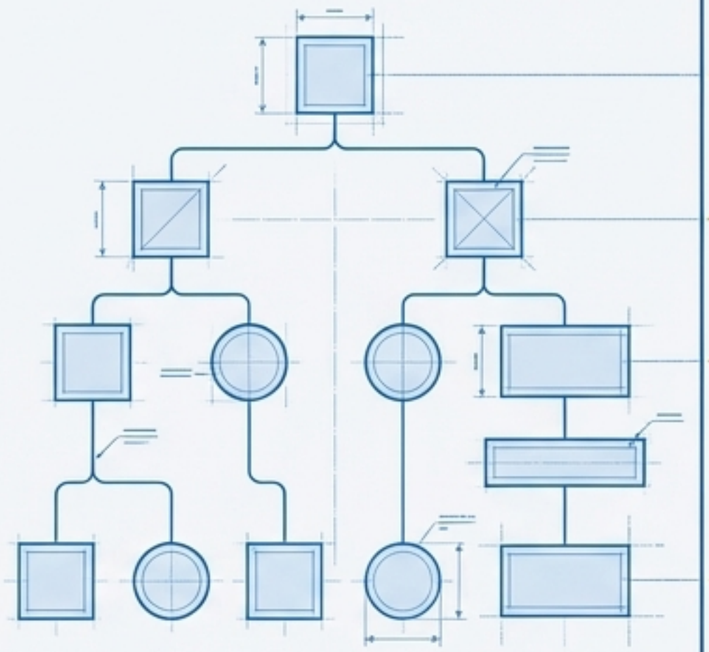
Widget Ağacı

Element Ağacı

Render Ağacı

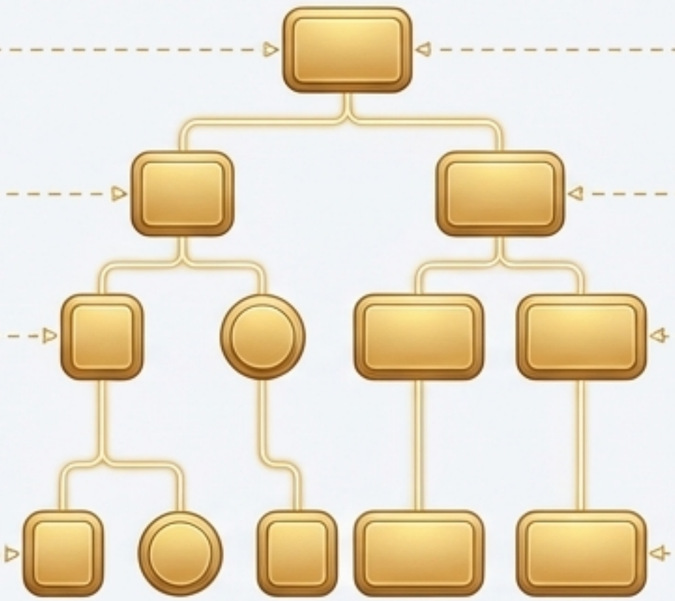
Üç Ağaç, Üç Farklı Rol

Widget Tree (Planlar)



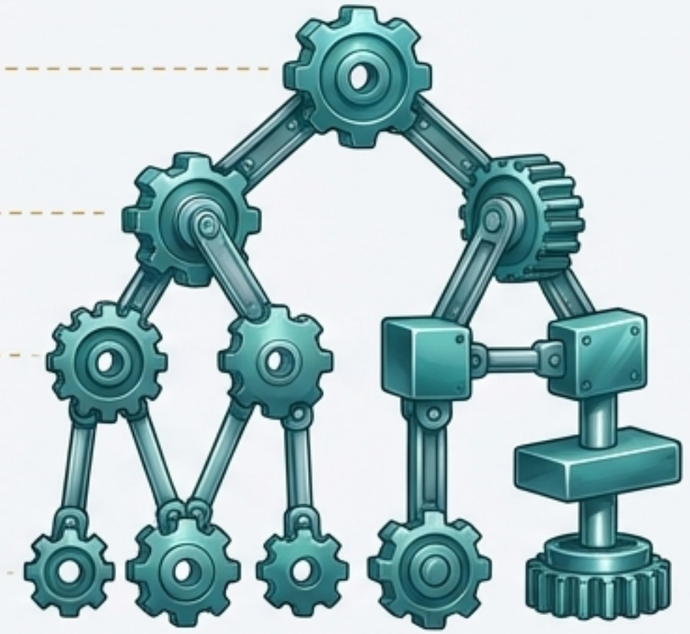
Geliştiricinin oluşturduğu, UI'ın yapılandırmasını tanımlayan ağaç. Hafif (lightweight), değişmez (immutable) ve oluşturulması ucuzdur. Bunlar sizin yazdığınız kodlardır.

Element Tree (Yöneticiler)



Widget ve RenderObject arasında bir köprü görevi görür. Değişiklikleri karşılaştırmada çok etkilidir ve UI'ın anlık yapısını (state) tutar. `BuildContext` aslında bir Element'tir.

Render Tree (İş Makineleri)



Gerçek işi yapan ağaç. Layout (yerleşim), painting (çizim) ve hit testing (dokunma testi) gibi tüm render mantığını içerir. Oluşturulması pahalıdır, bu yüzden Flutter onu mümkün olduğunca yeniden kullanmaya çalışır.

Yolculuk Başlıyor: "Ucuz" Güncelleme (Özellik Değişikliği)

Bir statefulWIdget'ın setState çağrısı ile sadece bir metnin değiştiğini düşünelim.

Kod Örneği (Önce & Sonra)

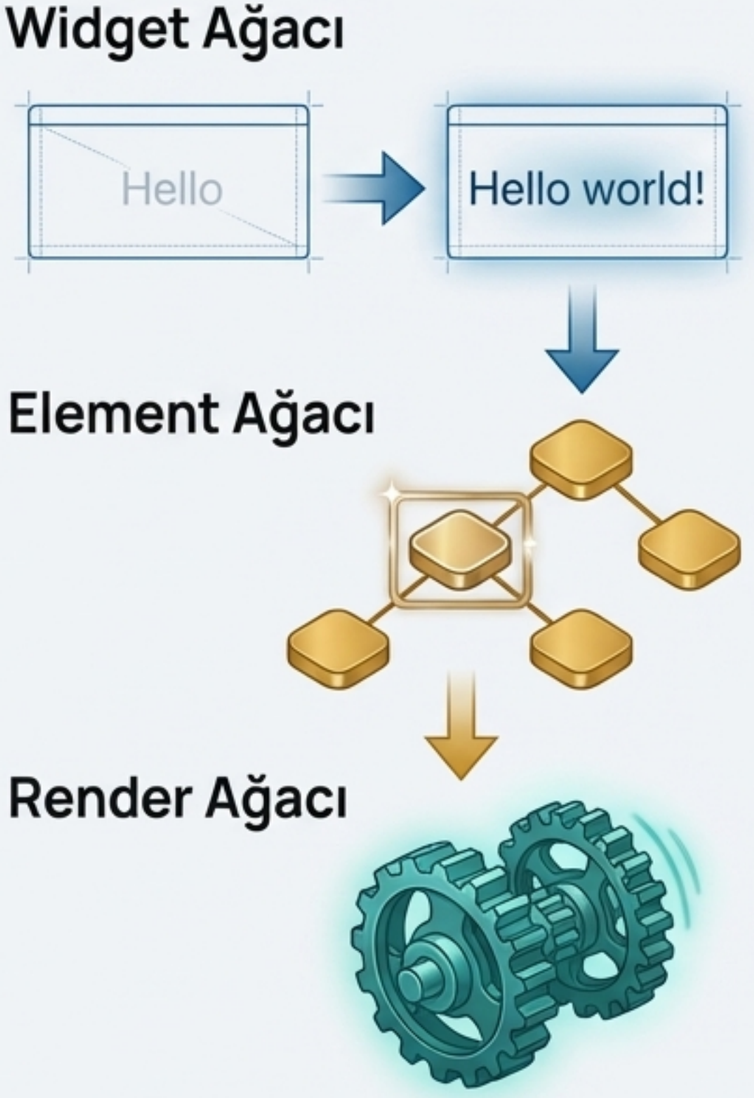
```
   // ÖNCE
1  SomeText(text: "Hello")
```

```
   // SONRA
1  SomeText(text: "Hello world!")
```

Görsel Açıklama

1. Flutter ağacı gezer ve `SomeText` widget'ının **tipinin aynı** kaldığını görür.
2. `Element` ağacı bu karşılaştırmayı yapar ve mevcut `Element`'i korur.
3. Element, ilişkili olduğu `RenderObject`'e yeni widget'ın `text` özelliğini ileterek sadece bir **güncelleme** tetikler.

RenderObject yeniden oluşturulmaz, sadece güncellenir. Bu çok hızlı bir işlemdir.



"Pahalı" Güncelleme: Tip Değişikliği

Şimdi `SomeText` widget'ını tamamen farklı bir widget olan Flutter'ın kendi `Text` widget'ı ile değiştirelim.

Kod Örneği (Önce & Sonra)

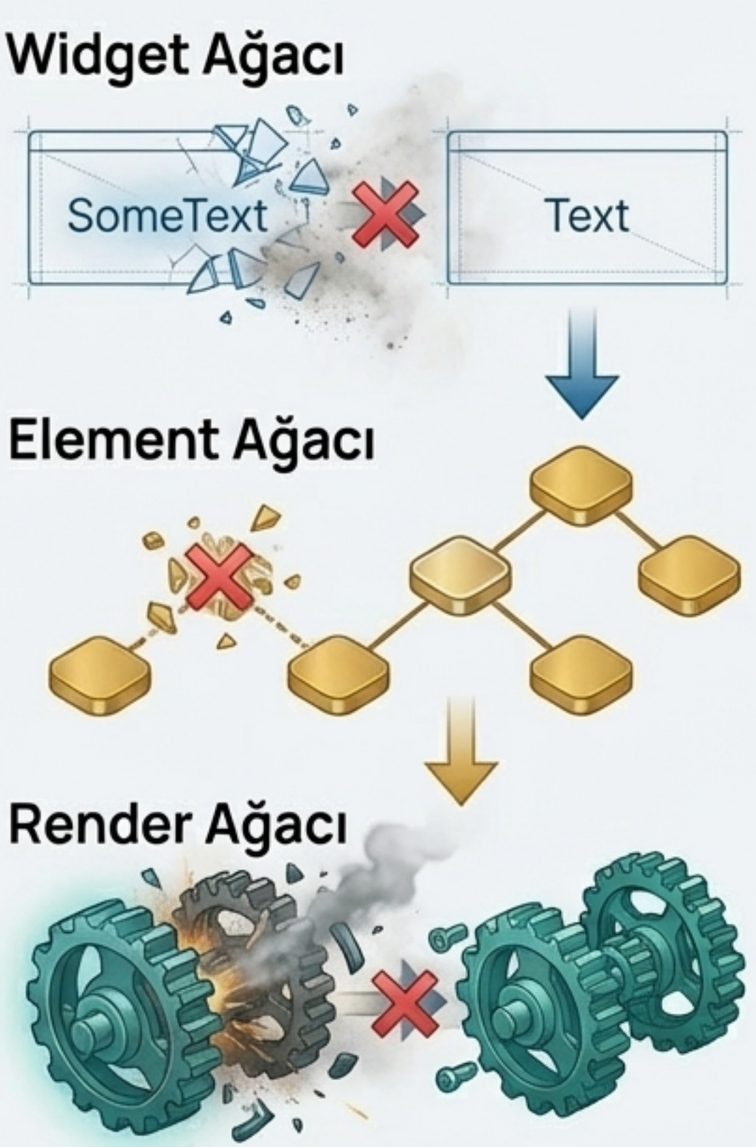
```
// ÖNCE
SomeText(text: "Hello")
```

```
// SONRA
Text("Hello world!")
```

Görsel Açıklama

1. Flutter ağacı gezer ve widget **tipinin değiştiğini** (`SomeText` → `Text`) fark eder.
2. `Element` ağacı bu uyumsuzluğu tespit eder. Eski `Element` ve `RenderObject` artık kullanılamaz.
3. Eski widget, element ve render nesnesinden oluşan tüm alt ağaç **yok edilir** ve yeni `Text` widget'ı için sıfırdan **yeniden oluşturulur**.

Bu işlem, pahalı olan RenderObject'in yeniden oluşturulmasını gerektirdiği için maliyetlidir.

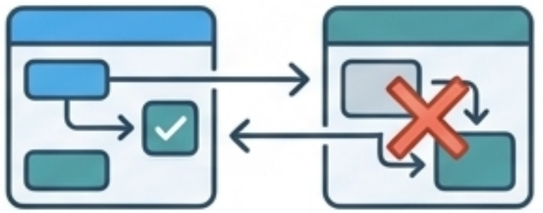


Geliştirici İçin Anlamı Ne? (Performans Çıkarımları)

Bu mimariyi anlamak, daha performanslı uygulamalar yazmanıza yardımcı olur.

Widget'lar Ucuz, RenderObject'ler Pahalıdır

`build` metodunuzun sık sık çalışmasından korkmayın. Flutter bunu verimli bir şekilde yönetmek için tasarlanmıştır.



Özellikleri Değiştirin, Tipleri Değil

Mümkün olduğunca, bir widget'ın tipini değiştirmek yerine mevcut bir widget'ın parametrelerini güncelleyin.

`const` Widget'lar Altın Değerindedir

Bir widget'ı `const` olarak işaretlediğinizde, Flutter'a onun asla değişmeyeceğini söylersiniz. Flutter bu widget ve altındaki alt ağaç için tüm karşılaştırma (diffing) işlemini atlayabilir. Bu, en büyük performans kazanımlarından biridir.

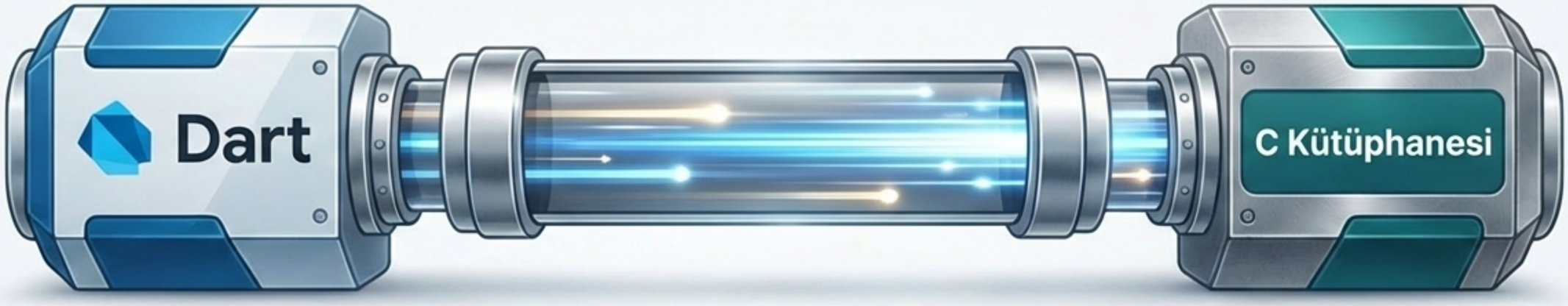
Anahtarlar (Key`) Önemlidir

Listelerde elemanların sırası değiştiğinde, `Key` kullanarak Flutter'a hangi elemanın hangisi olduğunu söylersiniz.
Bu, state'i korur ve gereksiz yeniden oluşturmaları önler.

Sınırları Aşmak: Flutter’ın Native Dünya ile İletişimi

Bir Flutter uygulaması nadiren tek başına yaşar. Cihazın özelliklerine erişmek veya mevcut native kodu kullanmak kritik öneme sahiptir. Flutter, bu iletişim için iki **güçlü mekanizma sunar**.

Hızlı Şerit: Foreign Function Interface (FFI)



`dart:ffi` kütüphanesi sayesinde Dart kodunuzun doğrudan C ile yazılmış native API'leri çağırmasını sağlar.

Serileştirme Yok, Gecikme Yok.
Veri dönüşümü gerekmediği için son derece hızlıdır. Çağrılar, dinamik olarak bağlı kütüphanelere (`.dll`, `.so`, `.dylib`) doğrudan yapılır.

Nasıl Çalışır (Basitleştirilmiş)

1. **İmzaları Tanımla:** C ve Dart fonksiyonları için iki `typedef` oluştur.
2. **Kütüphaneyi Yükle:** `DynamicLibrary.open()` ile ilgili kütüphaneyi aç.
3. **Fonksiyonu Çağır:** `lookup()` ile fonksiyonu bul ve `.asFunction()` ile dönüştürerek doğrudan çağır.

Diplomatik Köprü: Method Channels

Dart kodunun, uygulamanızı barındıran platforma özel kodları (Kotlin, Swift, Java vb.) çağırmasına olanak tanır.

Veriler, Dart'tan native tarafa gönderilirken **serileştirilir**, native taraftan cevap alınırken **deserileştirilir**. Bu işlem bir miktar ek yük (overhead) getirir.

FFI vs. Method Channels: Ne Zaman Hangisini Kullanmalı?

Foreign Function Interface (FFI)

Kullanım Alanı

Mevcut C/C++ kütüphanelerini (örn: SQLite, TensorFlow Lite) kullanmak, yüksek performans gerektiren hesaplamalar, gerçek zamanlı ses/video işleme.

Avantajları

+ Maksimum performans, sıfır serileştirme yükü.

Dezavantajları

— Yalnızca C uyumlu API'ler ile çalışır, kurulumu daha karmaşıktır.

Method Channels

Kullanım Alanı

Platform servislerine erişim (kamera, GPS, batarya durumu), native UI bileşenlerini kullanmak, platforma özel SDK'ları (örn: Firebase) çağırmak.

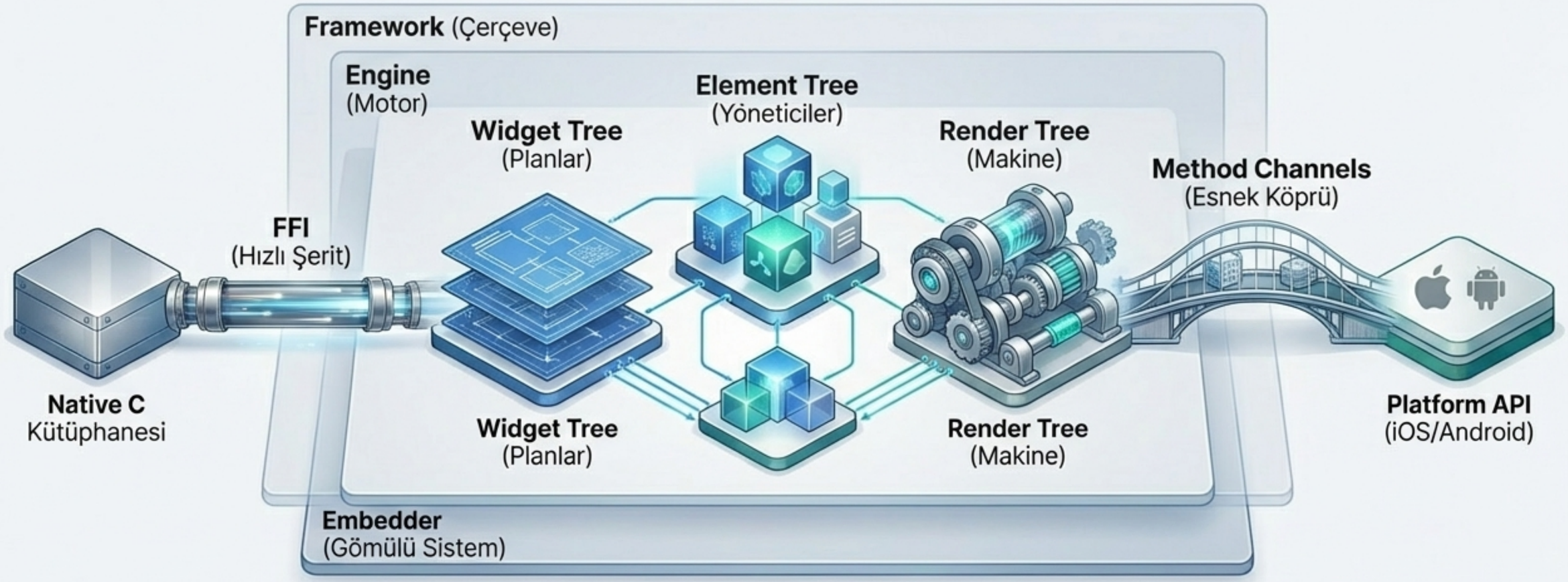
Avantajları

+ Kullanımı daha kolay, Kotlin/Swift gibi platform dilleriyle doğrudan entegrasyon.

Dezavantajları

— Serileştirme nedeniyle FFI'dan daha yavaş. Asenkron iletişim.

Bütün Resim: Flutter'ın Güçlü Mimarisi



Flutter'ın gücü; **esnek katmanlı mimarisi**, üç ağaçlı verimli render motoru ve güçlü native entegrasyon yeteneklerinin birleşiminden gelir. Artık bu makinenin kaputunun altında ne olduğunu ve yazdığınız kodun nasıl **piksele** dönüştüğünü biliyorsunuz.