Gizli Düşman: Flutter'da Performansı Tüketen Yeniden Oluşturmalar

60 FPS akıcılığını korumak ve kullanıcı deneyimini zirveye taşımak için bilinçli optimizasyon teknikleri.

NotebookLM

Flutter Hızlıdır. Ama Kodunuz Bu Hızı Sabote Edebilir.

Flutter framework'ü, widget ağacını verimli bir şekilde yönetmek ve arayüzü hızla oluşturmak için tasarlanmıştır. Ancak, en verimli sistem bile 'kötü kod' pratikleriyle yavaşlatılabilir.

Hedefimiz her zaman saniyede 60 kare (60 FPS) akıcılığını korumak olmalıdır. Kötü yazılmış kod, bu hedefin önündeki en büyük engeldir.

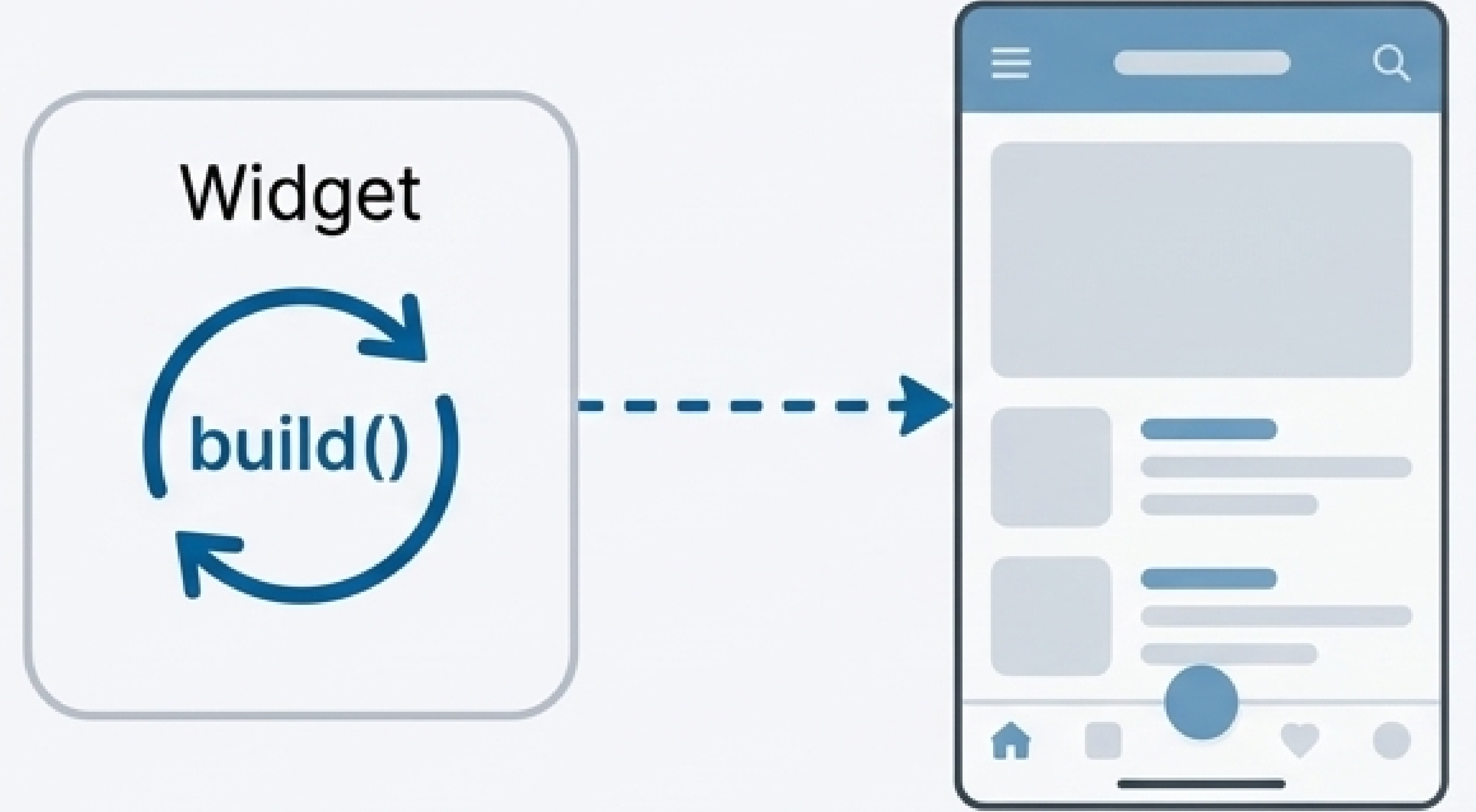
NotebookLM

'Yeniden Oluşturma' (Rebuild) Nedir?

Bir widget'ın `build()` metodunun yeniden çağrılmasıdır. Bu, Flutter'ın durum (state) değişikliklerine yanıt olarak arayüzü güncelleme mekanizmasıdır.

`build()` metodu ne zaman çağrılır?

- Arayüz ilk kez oluşturulduğunda.
- Uygulamanın yaşam döngüsü boyunca birçok kez.



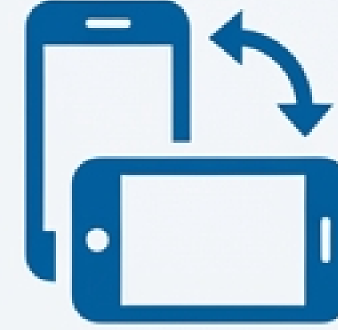
Önemli Not: `build()` metodunun kaç kez çağrılacağını tahmin edemezsiniz, çünkü birçok faktör yeniden oluşturmayı tetikleyebilir.

NotebookLM

Yeniden Oluşturmayı Ne Tetikler?

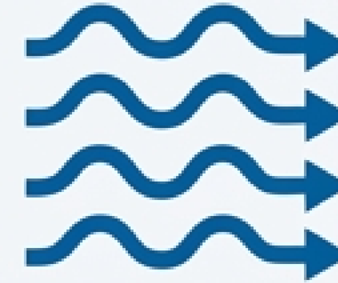
Aşağıdaki yaygın olaylar, bir widget ağacında yeniden oluşturma sürecini başlatabilir:

`setState` metodunu çağırmak.



Cihazın ekranını döndürmek.

Bir `Future` sonucunu beklemek (`await`).



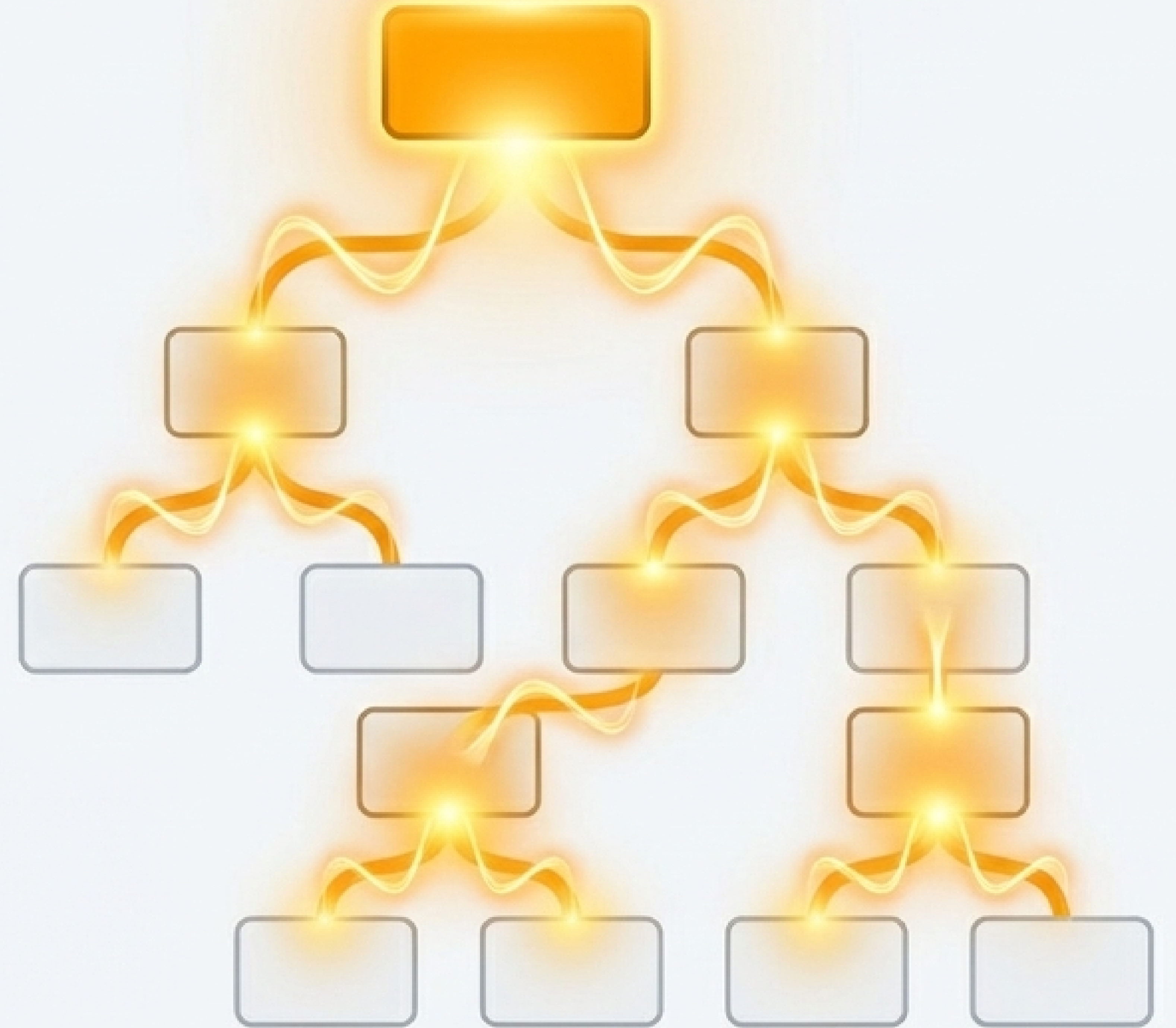
Bir `Stream`'den gelen olayları dinlemek.

NotebookLM

Asıl Sorun: Gereksiz Yeniden Oluşturma Fırtınası

Bir widget yeniden oluşturulduğunda, hiyerarşideki tutarlılığı korumak için tüm alt widget'ları da yeniden oluşturulur. Bu durum, kontrol edilmediğinde performansı tüketen bir zincirleme reaksiyona neden olabilir.

Altın Kural: Amacınız şu olmalı: 'Widget'ların yalnızca gerçekten gerektiğinde yeniden oluşturulmasına izin verin.'



NotebookLM

Cephaneliğiniz: Performansı Geri Kazanmanın İki Güçlü Yolu

Gereksiz yeniden oluşturmaları önlemek ve uygulamanızı akıcı tutmak için cephaneliğinize eklemeniz gereken iki temel optimizasyon tekniği vardır.

Silah 1: `const` Kurucular

Değişmeyen widget'ları "önbelleğe alarak" yeniden oluşturulmalarını tamamen engeller.

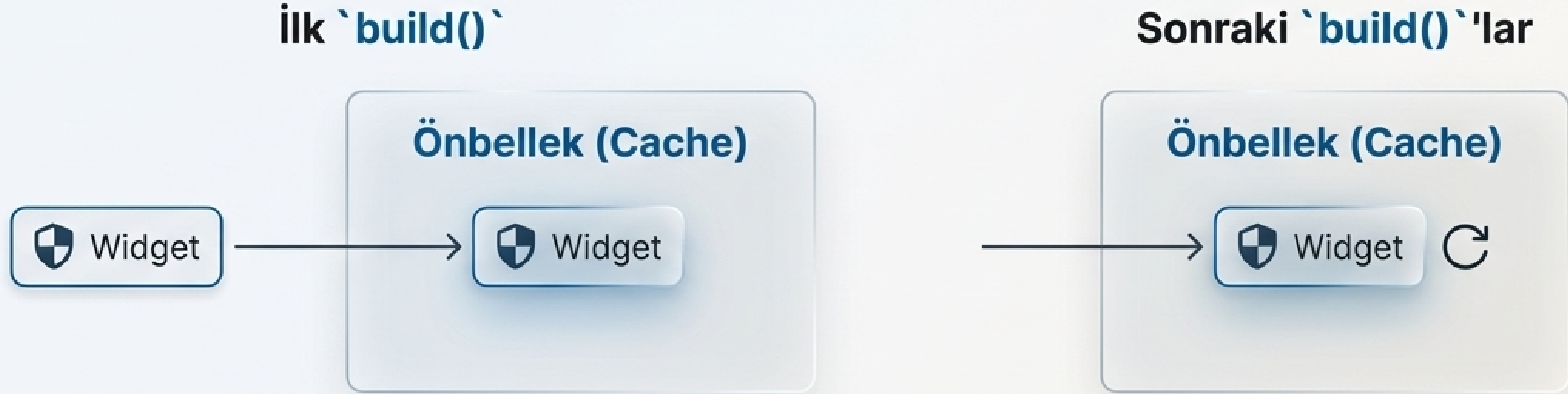
Silah 2: Widget Kompozisyonu

Widget döndüren fonksiyonlar yerine sınıfları kullanarak Flutter'ın optimizasyon mekanizmalarından tam olarak faydalanır.

NotebookLM

Silah 1: `const` Kurucuların Gücü

`const` ile işaretlenmiş bir widget, Flutter tarafından **yalnızca bir kez** oluşturulur.



`const` kullanmak, widget'ları önbelleğe almak (caching) gibidir. Bir kez oluşturulduktan sonra, asla yeniden oluşturulmazlar. Framework, sonraki `build` çağrılarında bu widget'ları görmezden gelir.

`const` kurucusu olan bir widget, 'değişmez' (immutable) demektir. Madem değişmeyecek, neden tekrar tekrar oluşturalım ki?

NotebookLM

Farkı Yaratan Tek Kelime: `const`

Aşağıdaki iki kod bloğu görsel olarak neredeyse aynıdır, ancak performans açısından aralarında devasa bir fark vardır.

VERİMSİZ

```
// OPTİMİZE EDİLMEMİŞ
ListView(
  children: [
    ExampleWidget(),
    ExampleWidget(),
    ExampleWidget(),
    // ...
  ]
);
```

VERİMLİ

```
// OPTİMİZE EDİLMİŞ
ListView(
  children: const [
    ExampleWidget(),
    ExampleWidget(),
    ExampleWidget(),
    // ...
  ]
);
```

`const` olmadan, `ExampleWidget` karmaşık bir `build()` metoduna sahipse, tüm liste her yeniden oluşturmada gereksiz yere baştan yaratılır. Bu, ciddi bir hesaplama israfıdır.

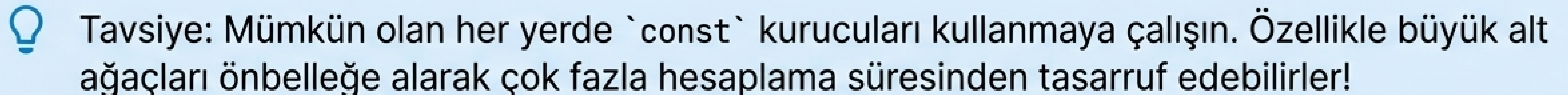
NotebookLM

Sadece `StatelessWidget` İçin Değil

`const` kurucular, durumu değişmeyen `StatefulWidget`'lar için de kullanılabilir ve kullanılmalıdır. Widget'ın kendisi değişmezken, durumu (`State` nesnesi) değişebilir.

```
class Example extends StatefulWidget {
  const Example(); // Bu widget yeniden oluşturulmayacak.

  @override
  _ExampleState createState() => _ExampleState();
}
```

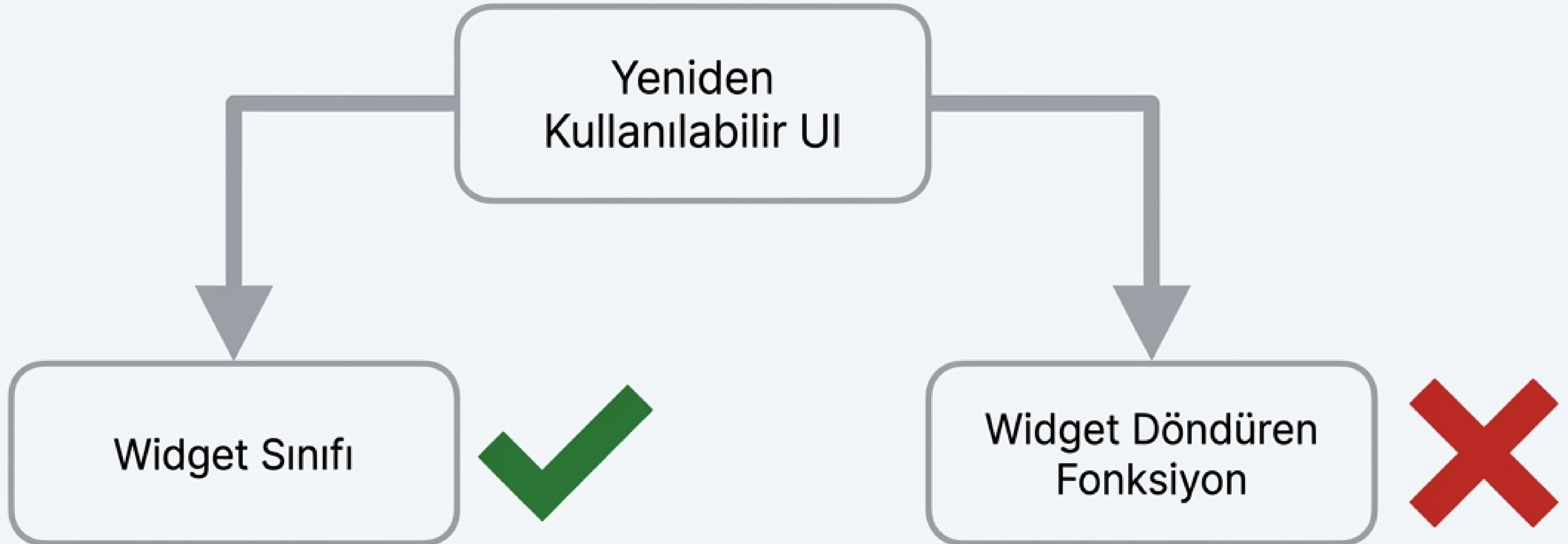


NotebookLM

Silah 2: Fonksiyonlar Yerine Widget Kompozisyonu

Kod tekrarı kötüdür, bu yüzden hepimiz yeniden **kullanılabilir UI bileşenleri** yaratırız. Flutter'da bunu yapmanın doğru ve yanlış bir yolu vardır.

Birçok sayfada kullanacağımız, telif hakkı metni ve ikonlar içeren bir '**footer**' bileşeni oluşturmak istediğimizi varsayalım.



NotebookLM

Doğru Yaklaşım: Yeniden Kullanılabilir Widget Sınıfları

Widget'lar, Flutter ekosisteminin temel yapı taşlarıdır. Yeniden kullanılabilir bir bileşen oluşturmanın en doğru yolu, onu bir `StatelessWidget` veya `StatefulWidget` sınıfı olarak tanımlamaktır.

İYİ PRATİK

```
class FooterWidget extends StatelessWidget {
  const FooterWidget();

  @override
  Widget build(BuildContext context) {
    return Column(
      // ...
      children: const [
        Icon(Icons.email),
        Icon(Icons.tablet_mac),
      ],
    );
  }
}
```

Avantajları:

Bu sınıf, widget ağacının bir parçasıdır, kendi `BuildContext`'ine sahiptir ve `const` kurucu sayesinde önbelleğe alınabilir.

NotebookLM

Kaçınılması Gereken Hata: Widget Döndüren Fonksiyonlar

Aynı 'footer' bileşenini bir fonksiyon olarak oluşturmak cazip gelebilir. Ancak bu, performansa zarar veren ve kesinlikle kaçınmanız gereken bir yaklaşımdır.

```
// BUNU ASLA YAPMAYIN!
Widget footerWidget(BuildContext context) =>
  Column(
    // ...
    children: const [
      Icon(Icons.email),
      Icon(Icons.tablet_mac),
    ],
  );
}
```

Bu kod, bir sınıf değil, sadece bir widget döndüren bir fonksiyondur.

NotebookLM

Fonksiyonlar Neden Performans Düşmanıdır?

Flutter framework’ü, sınıflar ve fonksiyonlar arasında önemli bir ayrım yapar. Fonksiyonlar, widget optimizasyon mekanizmalarının dışında kalır.

`const` **Kurucuları Yoktur:** Fonksiyonlar önbelleğe alınamaz. Her çağrıldıklarında baştan sona yeniden çalıştırılırlar.

Framework Tarafından Tanınmazlar: Flutter, bir fonksiyondan dönen widget hakkında hiçbir şey bilmez (`BuildContext’i yoktur). Bu yüzden onu her seferinde yeniden oluşturmak zorunda kalır.

Ağacın Parçası Değiller: Sınıflar widget ağacının yapraklarıdır, fonksiyonlar ise değildir. Bu yüzden framework’ün optimizasyonlarından faydalanamazlar.

Alt Çizgi: Widget sınıfları önbelleğe alınabilir, fonksiyonlar **alınamaz**.

NotebookLM

Ustalığa Giden Yol: Bilinçli Kodlama

Philosophy
Performans optimizasyonu, sadece birkaç kuralı ezberlemek değildir. Bu, kodunuzun Flutter framework'ü tarafından nasıl yorumlandığını ve yürütüldüğünü anlama ve her satırda bilinçli kararlar alma alışkanlığıdır.

Metaphor
Kodunuzu, framework'ün verimliliğini en üst düzeye çıkaracak şekilde yazın. Sisteme karşı değil, sistemle birlikte çalışın.

NotebookLM

Performans İçin Kontrol Listeniz

Bir sonraki Flutter projenize başlarken veya mevcut kodunuzu gözden geçirirken bu iki kuralı aklınızda tutun:

`const` Kullanımı

Değişmeyecek bir widget veya widget koleksiyonu mu tanımlıyorsunuz? Önüne `const` ekleyin. Bu, en kolay ve en etkili optimizasyondur.

Sınıf Tercihi

Yeniden kullanılabilir bir UI parçasına mı ihtiyacınız var? Asla bir fonksiyon oluşturmayın. Her zaman `StatelessWidget` veya `StatefulWidget` sınıfı kullanın.

Nihai Hedef: Gereksiz `build()` çağrılarını ortadan kaldırarak uygulamanızın her zaman 60 FPS'de çalışmasını sağlamak.

NotebookLM