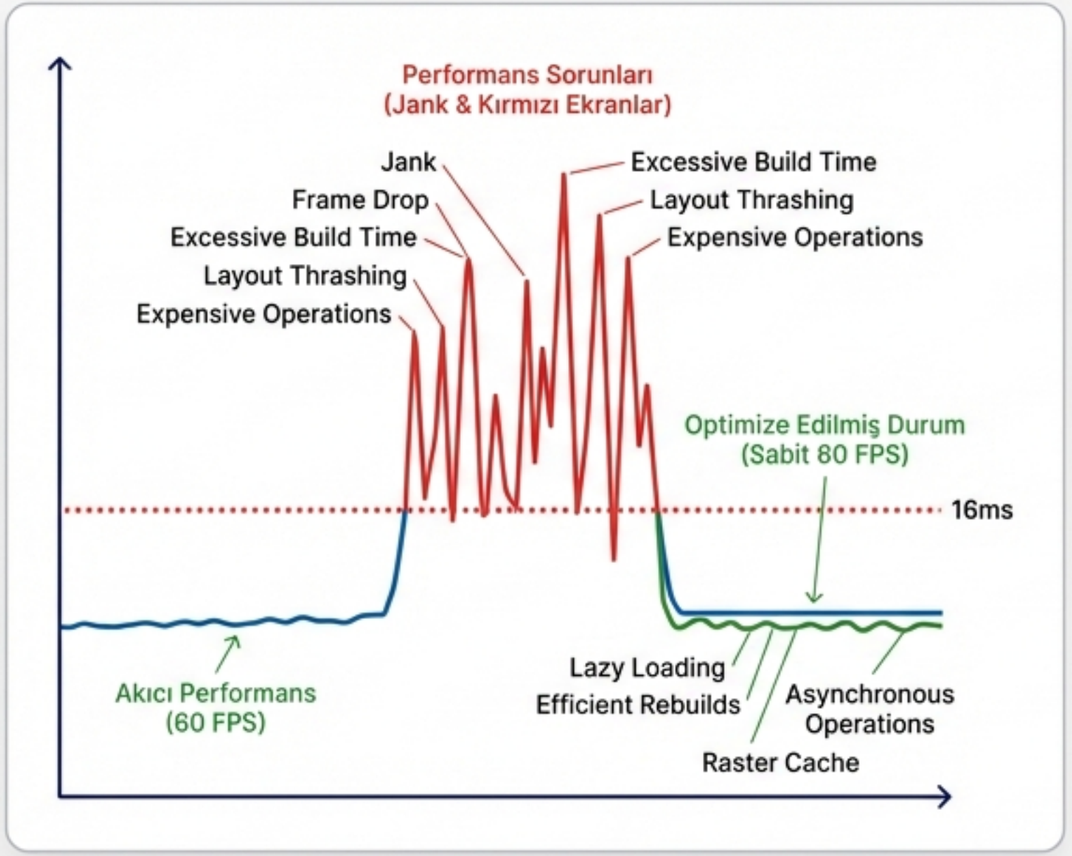
Flutter Performans Ustalığı

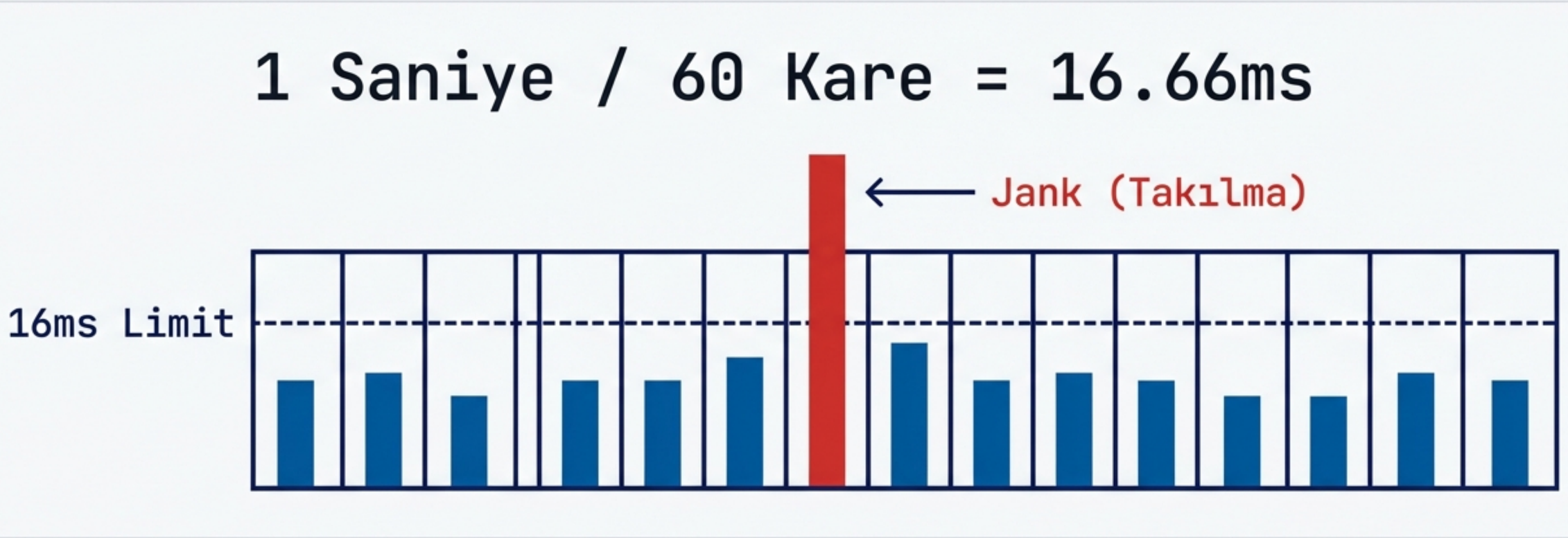
Profiling ve Optimizasyon

60 FPS'e Giden Yol ve DevTools Kullanımı



Kaynak: Chapter 16: Testing and profiling apps

Intter 16ms Mücadelesi ve 'Jank' Gerçeği



- **Flutter'ın hedefi: 60 FPS** (veya **120 FPS**).
- **Kural:** Her kare (frame) **16ms** içinde oluşturulup çizilmelidir.
- **Sorun:** 16ms'yi aşan kareler akıcılığı bozar ve "**Kırmızı Çubuk**" olarak işaretlenir.

Precision Engineering Manual

Altın Kural: Doğru Ortam (Profil Modu)

YANLIŞ

Debug Mode (Simülatör)

- Performans ölçümü için değildir.
- Emülatörler gerçek donanım gücünü yansıtmaz.
- Sonuçlar yanıltıcıdır.

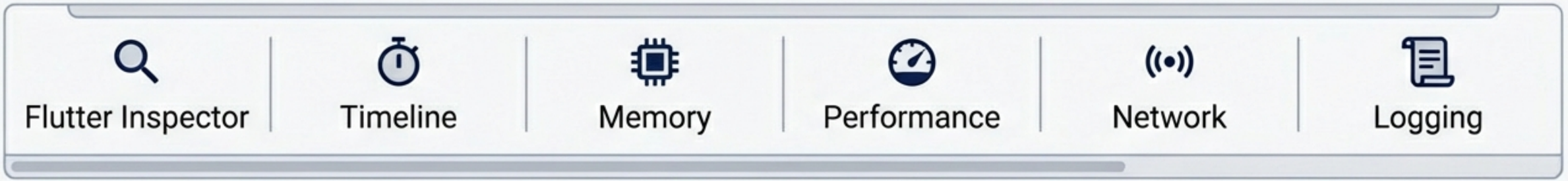
DOĞRU

Profile Mode (Fiziksel Cihaz)

- Release moduna en yakın performans.
- Analiz araçlarını (`DevTools`) açık tutar.
- Sadece fiziksel cihaza bağlıyken çalışır.

Komuta Merkezi: DevTools

Dart ve Flutter için özelleşmiş, tarayıcı tabanlı analiz paketi.



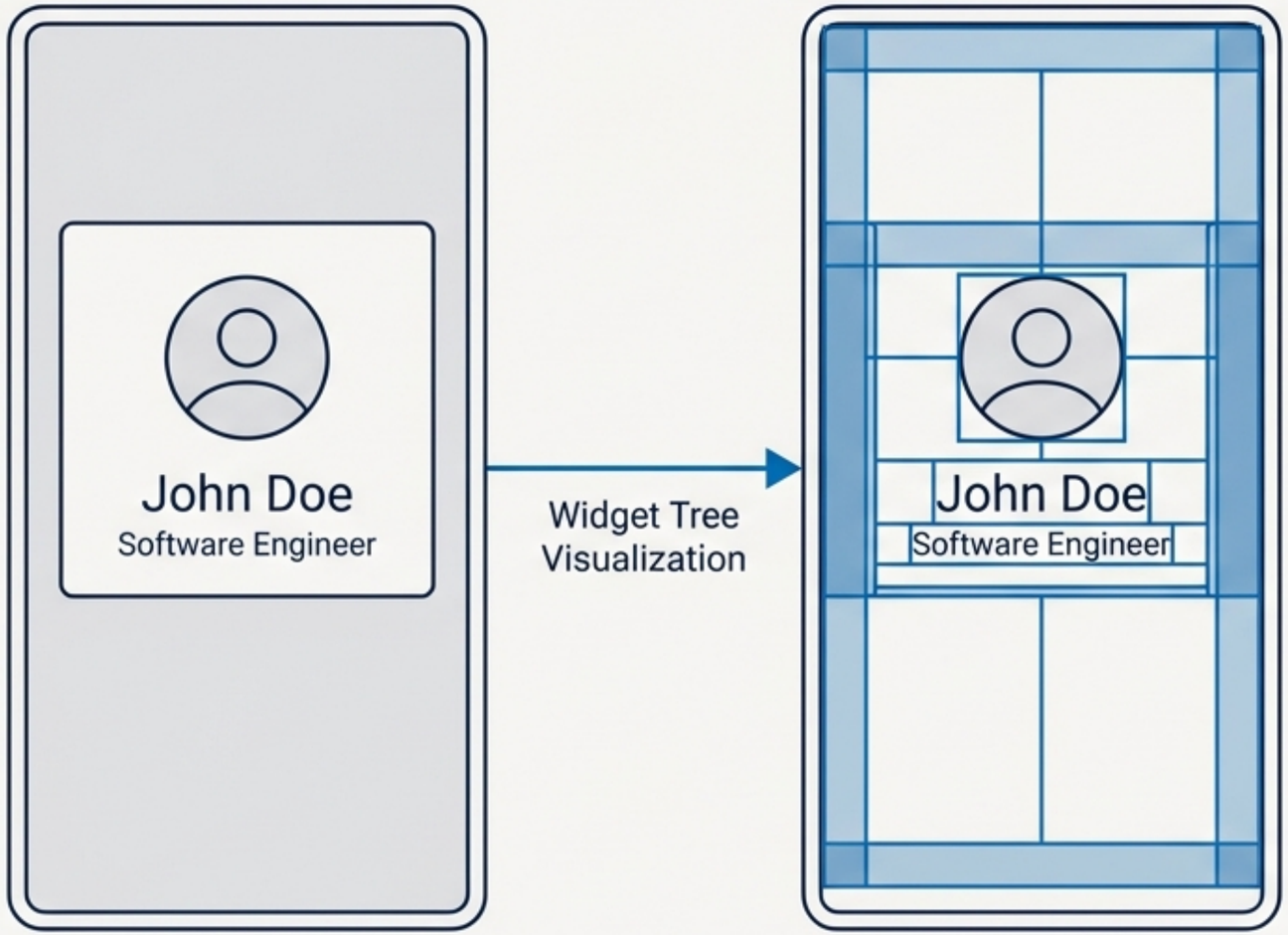
Etkileşim:
Uygulama ile doğrudan bağlantı.

Manipülasyon:
Yeniden inşaları tetikler ve animasyon hızını değiştirir.

Görselleştirme:
Widget ağacı ve bellek haritası.

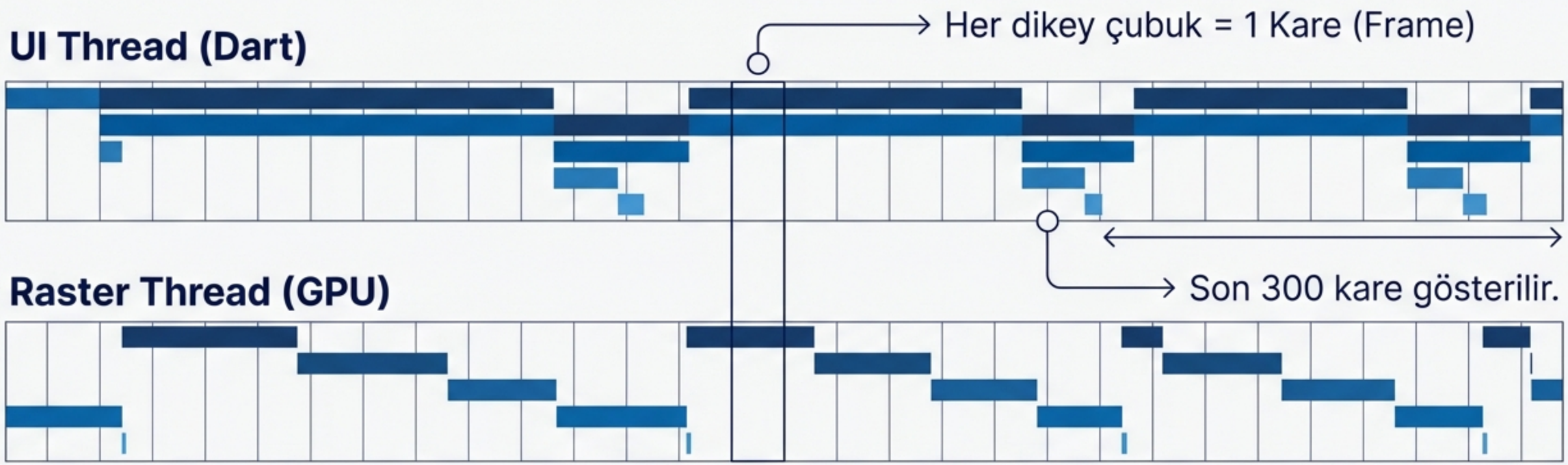
Analiz:
CPU ve GPU darboğaz tespiti.

Görsel Denetim: Flutter Inspector



- **Select Widget Mode:** Cihaz üzerinde ögeye tıklayarak boyutları ve kısıtlamaları (constraints) görün.
- **Slow Animations:** Animasyon hızını düşürerek görsel hataları yakalayın.
- **Debug Paint:** Hizalamaları ve sınırları (margins/padding) ekrana çizer.

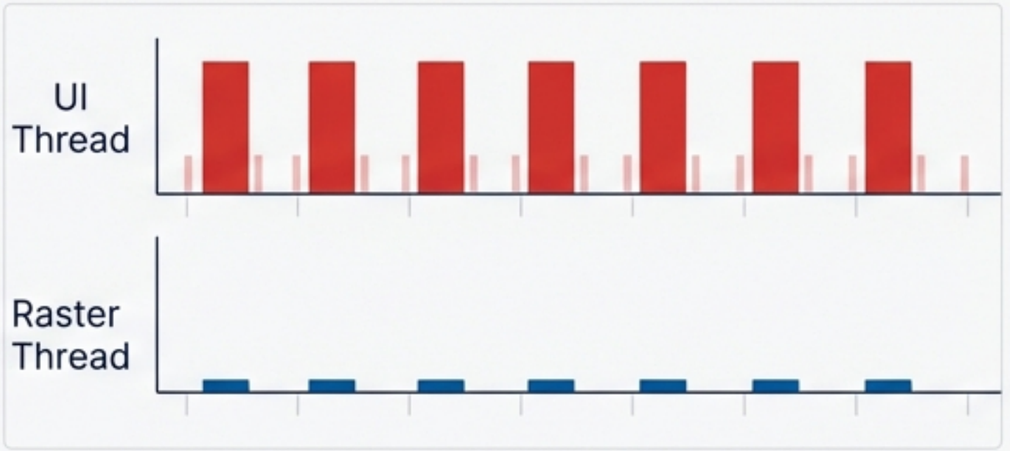
Uygulamanın Nabzı: Timeline (Zaman Çizelgesi)



1. **UI Thread:** Sizin kodunuz (Dart VM). `build()` metodları burada çalışır.
2. **Raster Thread:** Flutter motoru (Skia). UI thread'den gelen emirleri çizer.

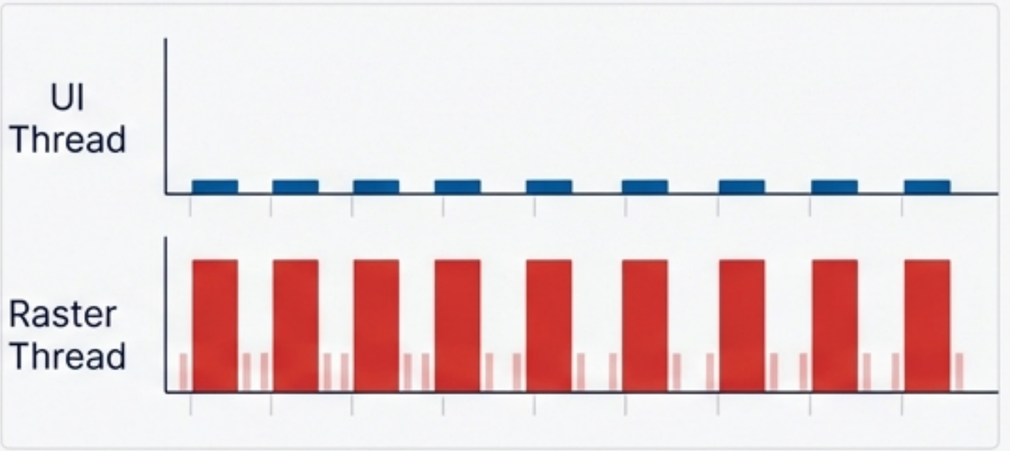
Bağlantı: UI Thread yavaşlarsa, Raster Thread beklemek zorunda kalır.

Teşhis: Kırmızı Çubukları Okumak



Senaryo A: UI Thread Kırmızı

Sebep: Dart kodu çok ağır veya `build()` uzun sürüyor.
Çözüm: Gereksiz rebuild'leri azaltın, ağır işlemleri asenkron yapın.



Senaryo B: Raster Thread Kırmızı

Sebep: Sahne çok karmaşık (gölgeler, `clipping`).

Durum: Kod hızlı çalıştı ama çizim motoru (GPU) zorlanıyor.

Optimizasyon: Kod Seviyesi (UI Thread)

```
child: const DecoratedBox(
  decoration: const BoxDecoration(color: Colors.blue),
  child: const Center(child: Text('Sabit Widget')),
)
```

1. Const Kullanımı:

Değişmeyen widget'lar için `const` kullanın. Flutter bu widget'ları önbelleğe alır ve tekrar oluşturmaz.

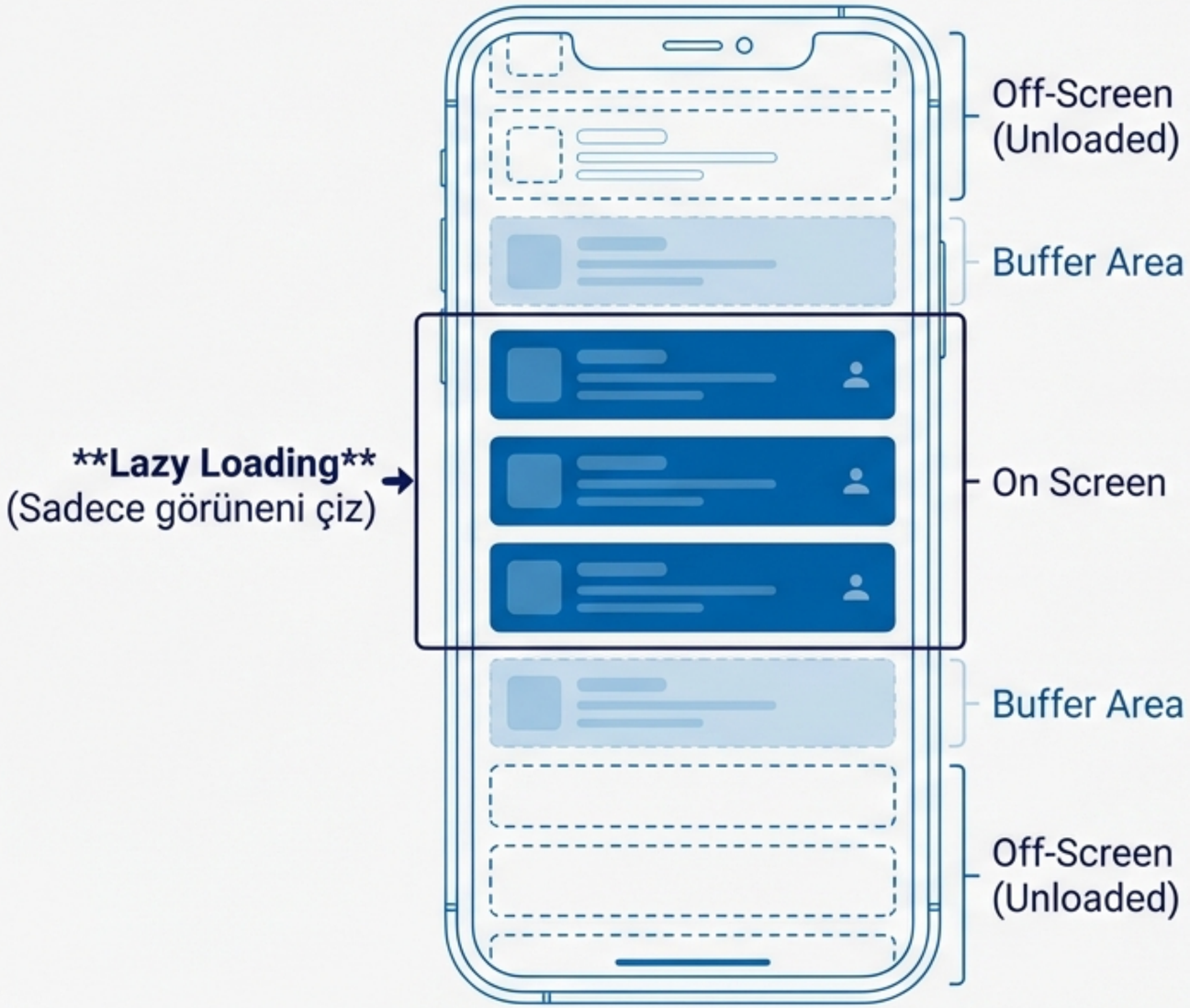
2. Async/Await:

UI thread'i bloklamamak için I/O işlemlerini asenkron yapın.

3. Manuel Caching:

Karmaşık widget dallarını bir `final` değişkende saklayın.

Listeler ve Izgaralar: Verimli Kaydırma



Builder Kullanımı

Her zaman `ListView.builder` kullanın. Tüm listeyi bir anda belleğe almaktan kaçının.

Pro İpucu: itemExtent

Eğer yükseklik sabitse `itemExtent: 75.0` gibi bir değer verin.
Flutter boyut hesaplamasını atlar ve kaydırma performansı artar.

Kritik Hata: FutureBuilder ve Rebuild Döngüsü

HATALI ❌

```
build(context) {
  return FutureBuilder(
    future: http.get('..'), // ⛔ Her build'de tekrar çalışır!
    builder: ...
  );
}
```

Her klavye açılışında veya ekran yenilenmesinde istek tekrar gönderilir.

DOĞRU ✅

```
initState() {
  _myFuture = http.get('..'); // ✅ Sadece bir kez çalışır.
}

build(context) {
  return FutureBuilder(
    future: _myFuture, // Saklanan değişken kullanılır.
    builder: ...
  );
}
```

Future nesnesi `initState` içinde başlatılmalı ve saklanmalıdır.

GPU Katilleri: Raster Thread Optimizasyonu

Eğer Raster Thread kırmızıysa, suçlular genellikle bunlardır.

Opacity

Animasyonlarda `Opacity` widget'ı kullanmak çok maliyetlidir. Mümkünse kaçının veya alternatif kullanın.

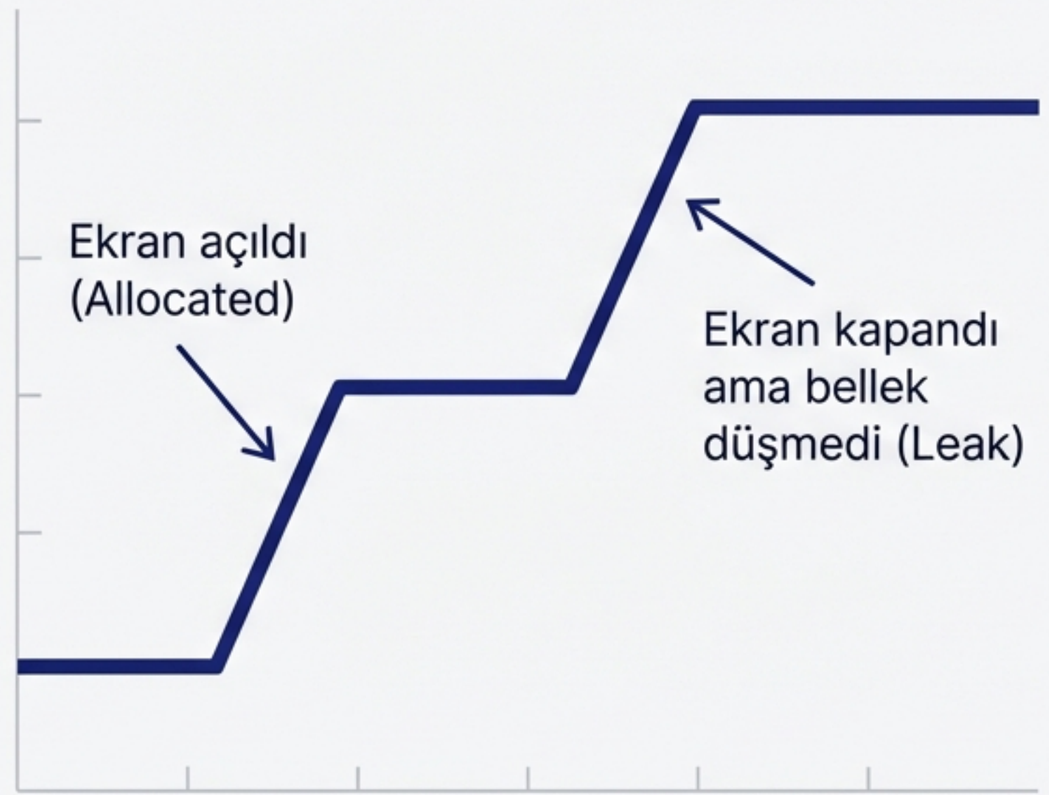
Clipping (Kırpma)

`ClipRect`, `ClipRRect`, `ClipPath` işlemleri GPU için pahalıdır. Aşırı kullanımdan sakının.

Shadows (Gölgeler)

BoxShadow render süresini uzatır. Fiziksel gölgeler (elevation) yerine basit border kullanmayı düşünün.

Bellek Yönetimi ve Sızıntılar (Leaks)



Snapshot Tekniği ile Tespit:

1. DevTools Memory sekmesinde 'Reset' butonuna basın.
2. Şüpheli işlemi yapın (sayfayı aç/kapat).
3. 'Snapshot' alın.
4. Sonuç: Kapanan sayfanın nesneleri hala bellekteyse sızıntı vardır.

Olağan Şüpheli: Dispose edilmemiş `AnimationController` veya dinleyiciler.
JetBrains Mono

Ağ Trafiği ve Akıllı Loglama

Network Inspection

DevTools Network

Method	Status	Type	Time
GET	200	json	150ms

HTTP trafiğini tahmin etmeyin, Network sekmesinden izleyin.

Logging Strategy

`print()` yerine `developer.log()` kullanın.

```
import 'dart:developer' as developer;

developer.log(
  'Veri yüklendi',
  name: 'my.app.network',
  error: jsonError,
);
```

↳ Tree Shaking sayesinde bu loglar Release modunda uygulamadan tamamen silinir.

Yeniden İnşaları İzlemek (Widget Rebuild Stats)

Widget Rebuild Stats (Android Studio)

Widget	Rebuild Count
`Scaffold`	1
`MyHeaderWidget`	1
`StaticIcon`	60

Analysis and Action

- **Araç:** Android Studio Performance sekmesi.
- **Mantık:** Sabit bir widget'ın sayacı '1' olmalıdır.
- **Hata:** Eğer statik bir widget **60 kez yeniden çiziliyorsa** (üstteki `StaticIcon` örneği), **`const` eksiktir.**
- **Aksiyon:** Gereksiz rebuild'leri durdurun.

Mühendisin Kontrol Listesi

- [x] 1. **Doğru Ortam:** Testleri Fiziksel Cihazda ve Profil Modunda yapın.
- [x] 2. **16ms Kuralı:** Kare sürelerini takip edin.
- [x] 3. **Future Yönetimi:** `initState` içinde cacheleyin, `build` içinde değil.
- [x] 4. **Bellek:** Controller'ları mutlaka `dispose` edin.
- [x] 5. **Const:** Sabit widget'larda `const` kullanın.

"Hata ayıklama (Debugging) geliştirirken; Profilleme (Profiling) yayınlamadan önce yapılır."