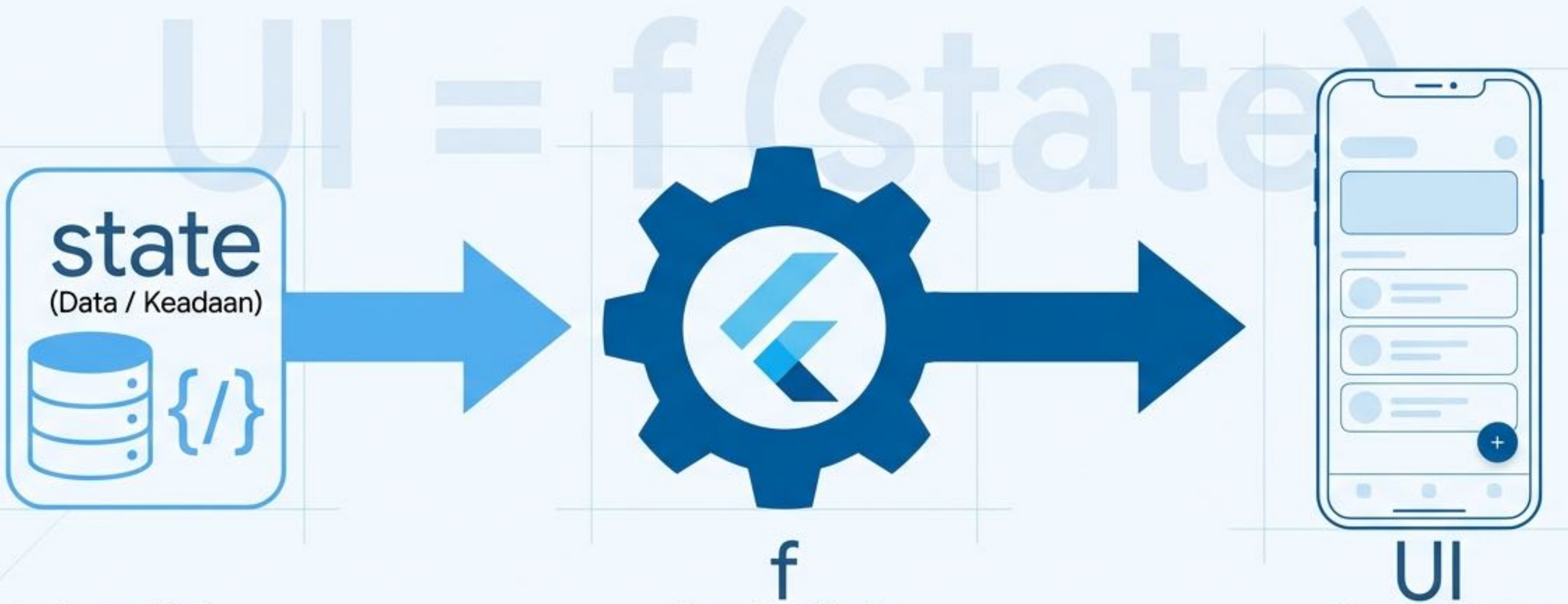
Aplikasi Mobile, Pertemuan #3

Manajemen State (Stateless & StatefulWIdget) & Activity Lifecycles

Dosen: Rizki Muliono, S.Kom, M.Kom
Prodi: Teknik Informatika
Universitas: Universitas Medan Area

Konsep Dasar State di Flutter

Dalam Flutter, antarmuka pengguna (UI) adalah refleksi langsung dari state (data/keadaan) aplikasi saat ini. Jika state berubah, UI akan otomatis dibangun ulang (rebuild) untuk menampilkan data terbaru.



Stateless Widget (UI Statis)

Statis: Tidak memiliki keadaan yang berubah setelah dibangun. Tidak ada perubahan data internal.

Sederhana & Cepat: Proses render sangat efisien dan ringan.

Pemicu Rebuild: Hanya dibangun ulang jika parent widget mengalami perubahan, bukan dari dirinya sendiri.

Fungsi Terbaik: Teks statis, Gambar, atau Ikon dasar.

Implementasi & Hasil: Stateless Widget

```dart
class MyStatelessWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Stateless Widget')),
      body: Center(
        child: Text('Hello, UMA! (Statis)'),
      ),
    );
  }
}
```

Hasil: Layar menampilkan teks 'Hello, UMA! (Statis)'. Tampilan ini permanen dan tidak akan merespons perubahan data internal meskipun pengguna berinteraksi.

Stateful Widget (UI Dinamis)

Dinamis: Memiliki siklus hidup mandiri (mirip Android/iOS) dan dapat merespons interaksi.

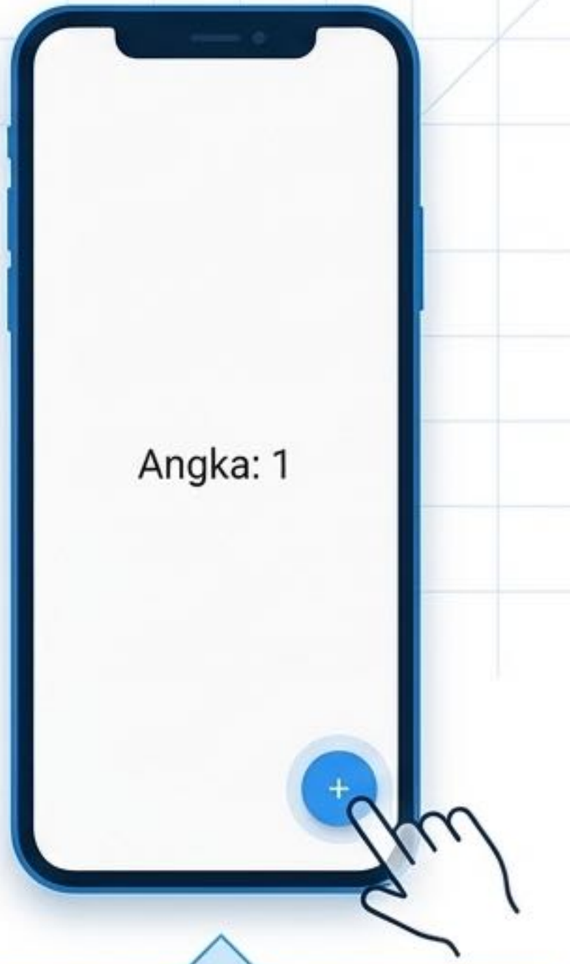
Manajemen State: Membutuhkan alokasi memori khusus untuk menyimpan data yang berubah-ubah.

Pembaruan Selektif: Hanya bagian UI spesifik yang berubah yang akan digambar ulang.

Fungsi Terbaik: Formulir input, penghitung (counter), animasi, atau interaksi UI kompleks.

Implementasi & Hasil: Stateful Widget

```dart
class MyStatefulWidget extends StatefulWidget {
  @override
  _MyStatefulWidgetState createState() => _MyStatefulWidgetState();
}
class _MyStatefulWidgetState extends State<MyStatefulWidget> {
  int _counter = 0;
  void _increment() {
    setState(() { _counter++; }); // Memicu rebuild UI
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(child: Text('Angka: $_counter')),
      floatingActionButton: FloatingActionButton(onPressed: _increment,
        child: Icon(Icons.add)),
    );
  }
}
```



Hasil: Angka di layar bertambah secara real-time setiap kali tombol ditekan. `setState` memerintahkan UI untuk merender ulang bagian teks saja.

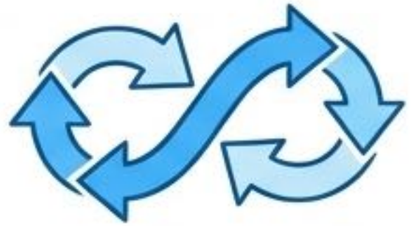
Analisis Komparasi: Stateless vs Stateful

	Stateless	Stateful
Sifat Data	Statis (Tidak Berubah)	Dinamis (Berubah-ubah)
Beban Performa	Sangat Cepat & Ringan	Membutuhkan Manajemen Memori
Metode Utama		
Pemicu Rebuild	Perubahan dari Parent	Perubahan State Internal
Contoh Ideal	Label teks, icon, layout dasar	Form login, animasi, fetch API

Anatomi Dinamis: initState vs setState

initState() - Inisialisasi Awal

- **Frekuensi**: Dipanggil hanya satu kali saat objek State pertama kali dibuat.
- **Tujuan**: Persiapan awal (sekali seumur hidup widget).
- **Contoh**: Mengatur listener, atau fetch data pertama kali dari server/API.

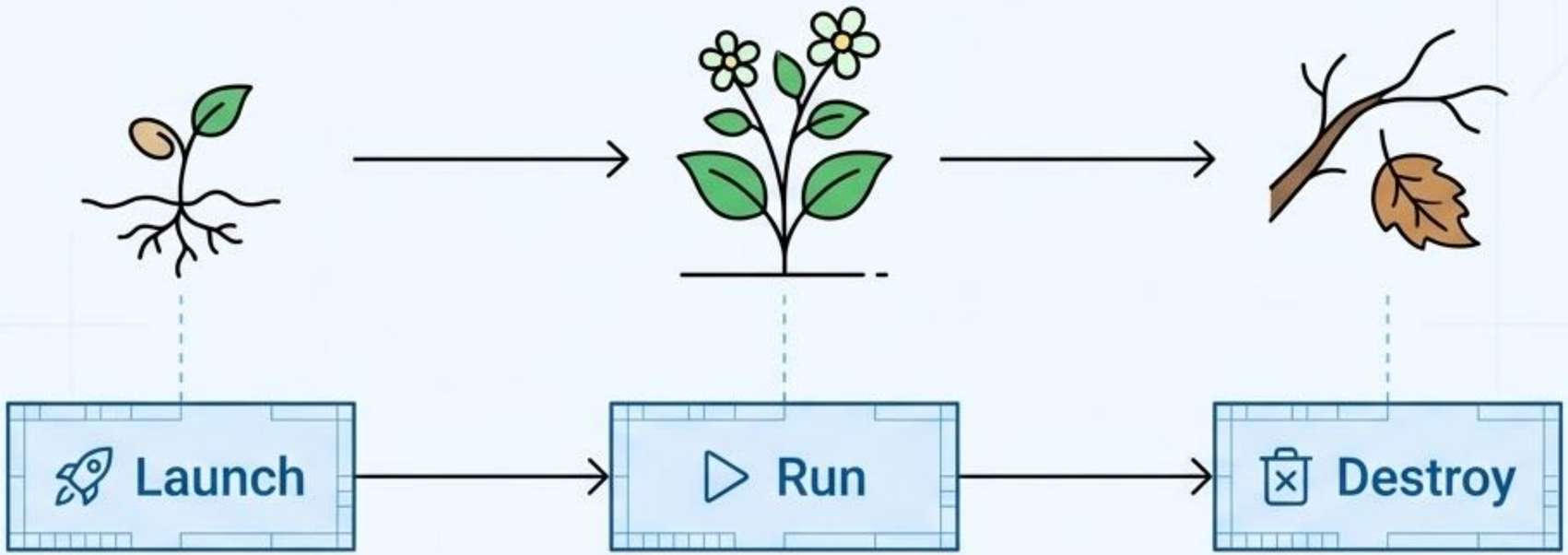


setState() - Pembaruan UI

- **Frekuensi**: Dipanggil berkali-kali saat terjadi perubahan data internal.
- **Tujuan**: Memberitahu framework untuk memicu build() ulang UI agar sinkron dengan data.
- **Contoh**: Memperbarui counter angka, mengubah teks saat tombol ditekan.

Di Balik Layar: Memahami Activity Lifecycle (Android)

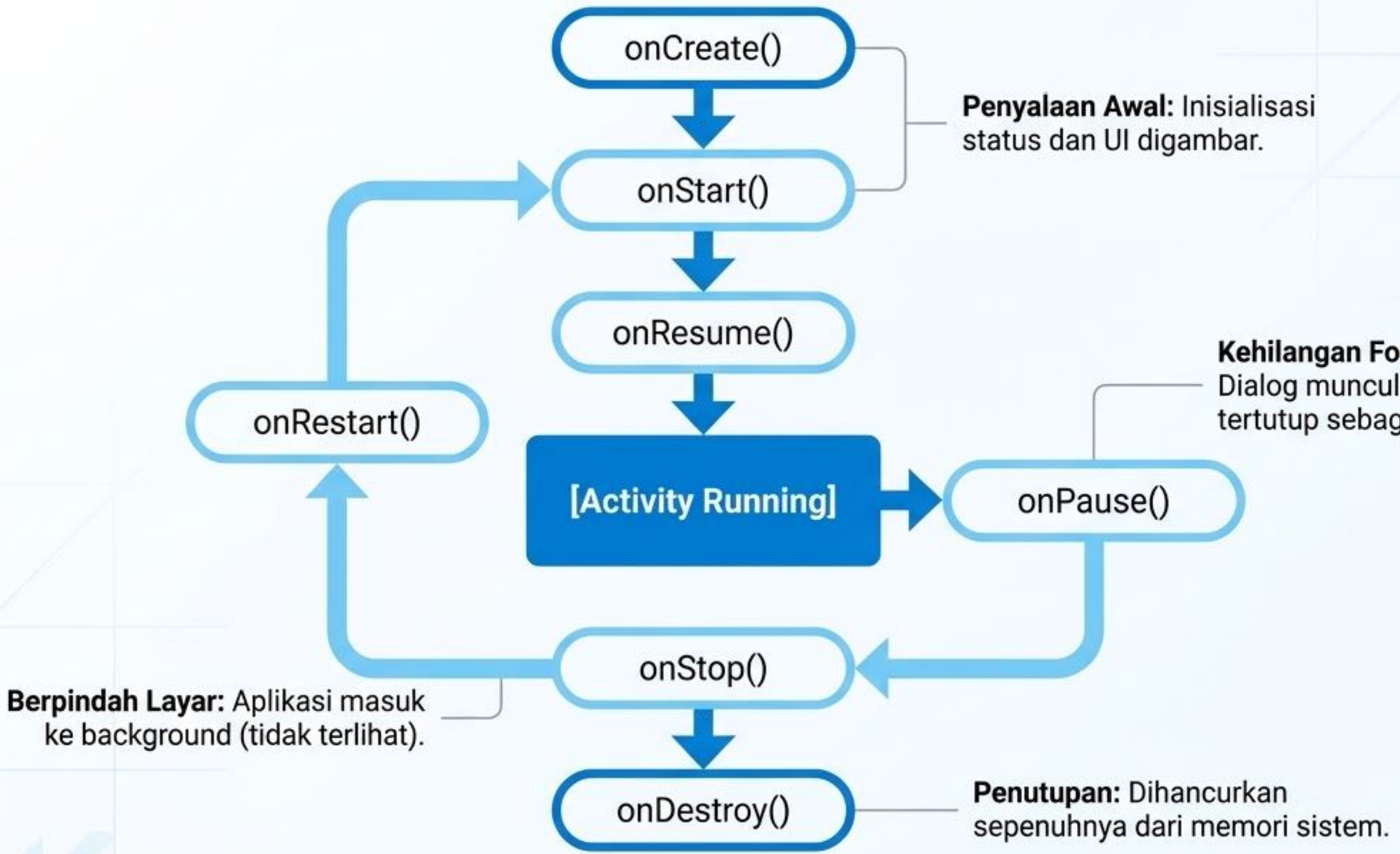
Setiap aplikasi memiliki "daur hidup" dari saat diciptakan hingga dihancurkan. Sistem callback oleh framework Android mengelola transisi status layar (activity).



Mengapa ini krusial?

- Mencegah aplikasi crash secara tiba-tiba.
- Menghemat baterai dengan mematikan proses yang tidak terlihat.
- Memastikan data/status pengguna tidak hilang saat berpindah aplikasi.

Flowchart: Android Activity Lifecycle



Studi Kasus: Lifecycle di Dunia Nyata

1. Membuka Aplikasi Pertama Kali

Sistem Memanggil:
onCreate() -> onStart()
-> onResume()

2. Tiba-tiba Ada Telepon Masuk

Aplikasi ke Background.

Sistem Memanggil:
onPause() -> onStop().

Waktu yang tepat untuk otomatis menghentikan video/animasi agar hemat baterai.

3. Kembali ke Aplikasi Utama

Selesai Telepon.

Sistem Memanggil:
onRestart() -> onStart()
-> onResume().

Aplikasi siap digunakan lagi.

4. Menekan Tombol Back

Selesai Menggunakan.

Sistem Memanggil:
onPause() -> onStop()
-> onDestroy().

Memori dibebaskan kembali.

Siklus Hidup Flutter (Flutter Lifecycle)

Dalam Flutter, siklus hidup dikelola melalui StatefulWIdget. Meskipun arsitektur berbeda dari Android, konsep manajemen sumber dayanya sangat serupa.

Inisialisasi awal (Setara dengan onCreate di Android).

Dipanggil segera setelah initState selesai.

Membangun UI (Paling sering dipanggil, dipicu terus menerus oleh setState).

Membersihkan sumber daya dan memori sebelum widget dihancurkan (Setara onDestroy).

Code & Implementasi: Siklus Hidup Flutter

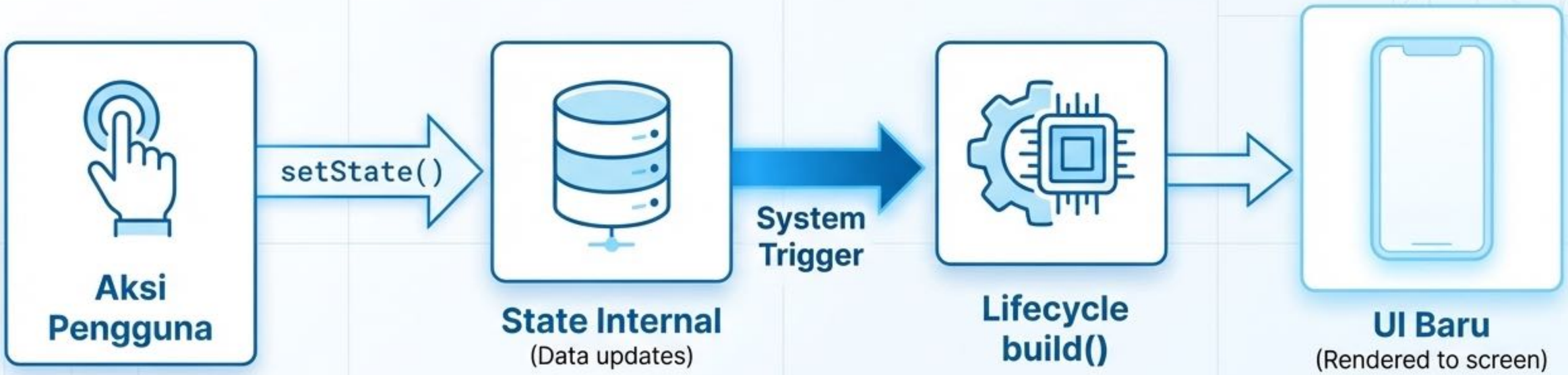
```dart
class MyLifecycleWidget extends StatefulWidget {
  @override
  _MyLifecycleState createState() => _MyLifecycleState();
}

class _MyLifecycleState extends State<MyLifecycleWidget> {

  @override
  void initState() {
    super.initState();
  }

  @override
  Widget build build(BuildContext context) {
    return Center(child: Text('Lifecycle Demo'));
  }

  @override
  void dispose() {
    super.dispose();
  }
}
```

Eksekusi **SEKALI** di awal
(Setup listener / Fetch API)

Eksekusi **BERKALI-KALI**
(Membangun dan memperbarui UI)

Eksekusi **TERAKHIR**
(Bersihkan memori/controller untuk cegah kebocoran)

Sintesis: Simfoni State dan Lifecycle



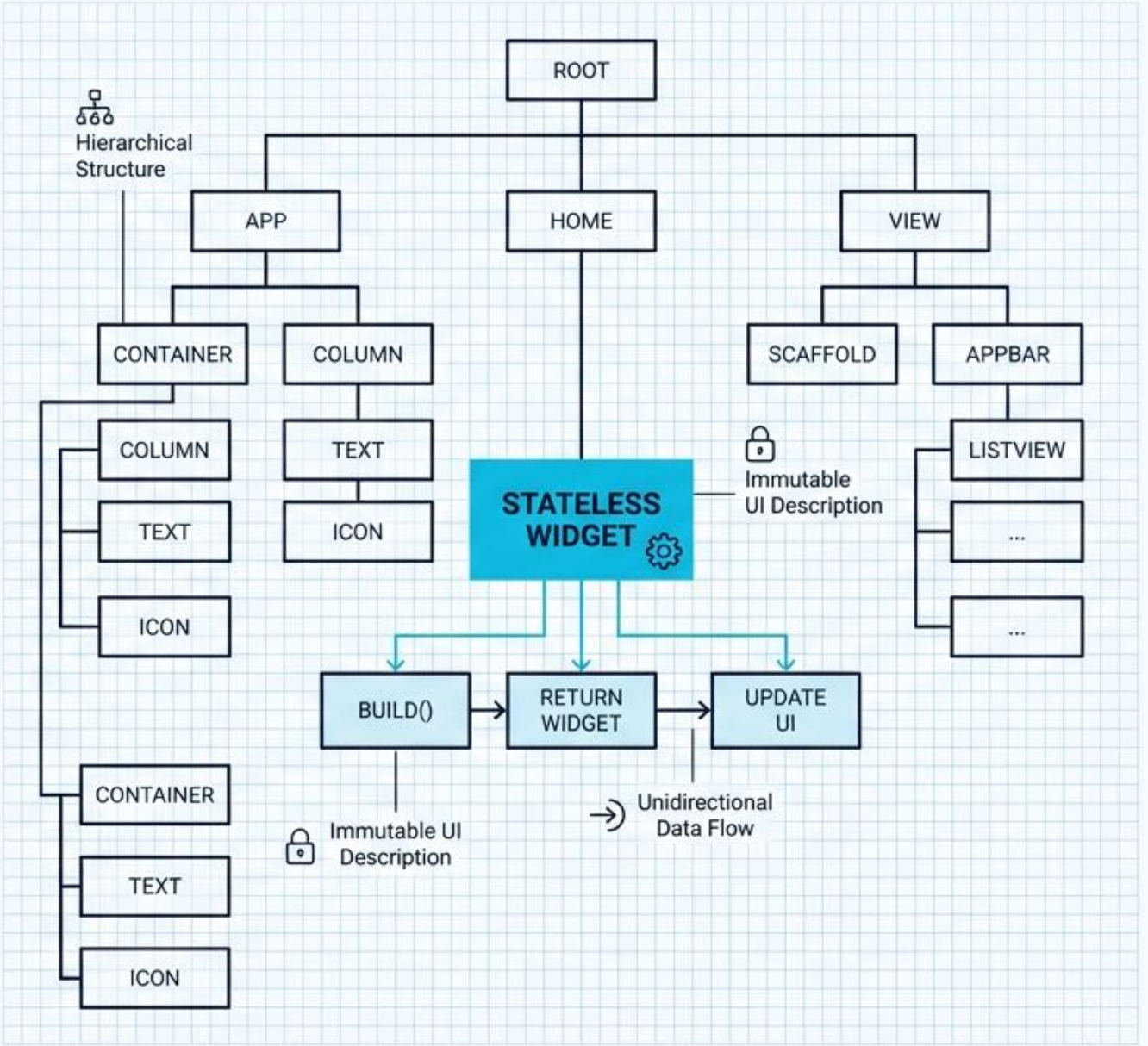
Interaksi Utama: State dan Lifecycle bukanlah dua hal yang terpisah, melainkan satu mesin penggerak UI yang saling bergantung.

Sinkronisasi: Setiap kali data (State) berubah melalui setState(), sistem akan memanggil fungsi build() pada Lifecycle untuk memastikan layar selalu sinkron dengan data mutakhir.

Keamanan Memori: Manajemen Lifecycle yang disiplin (initState & dispose) memastikan State berjalan mulus tanpa Memory Leak.

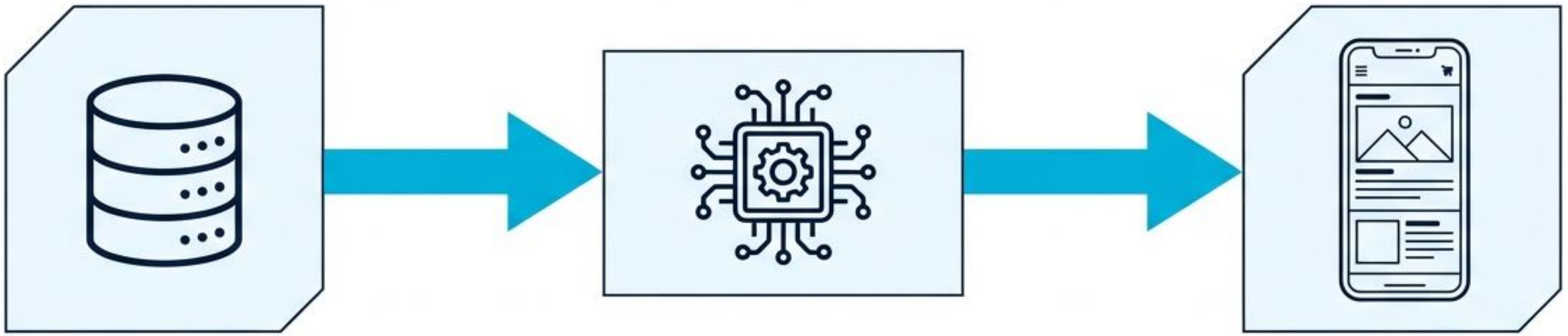
Arsitektur Widget & Sinkronisasi State di Flutter

Memahami Anatomi Stateless Widget dan Mekanisme Siklus build()



Konsep Fundamental Declarative UI

$$UI = f(state)$$



Data (State)
Kondisi data aplikasi saat ini.

Fungsi
Pemrosesan logika melalui metode build().

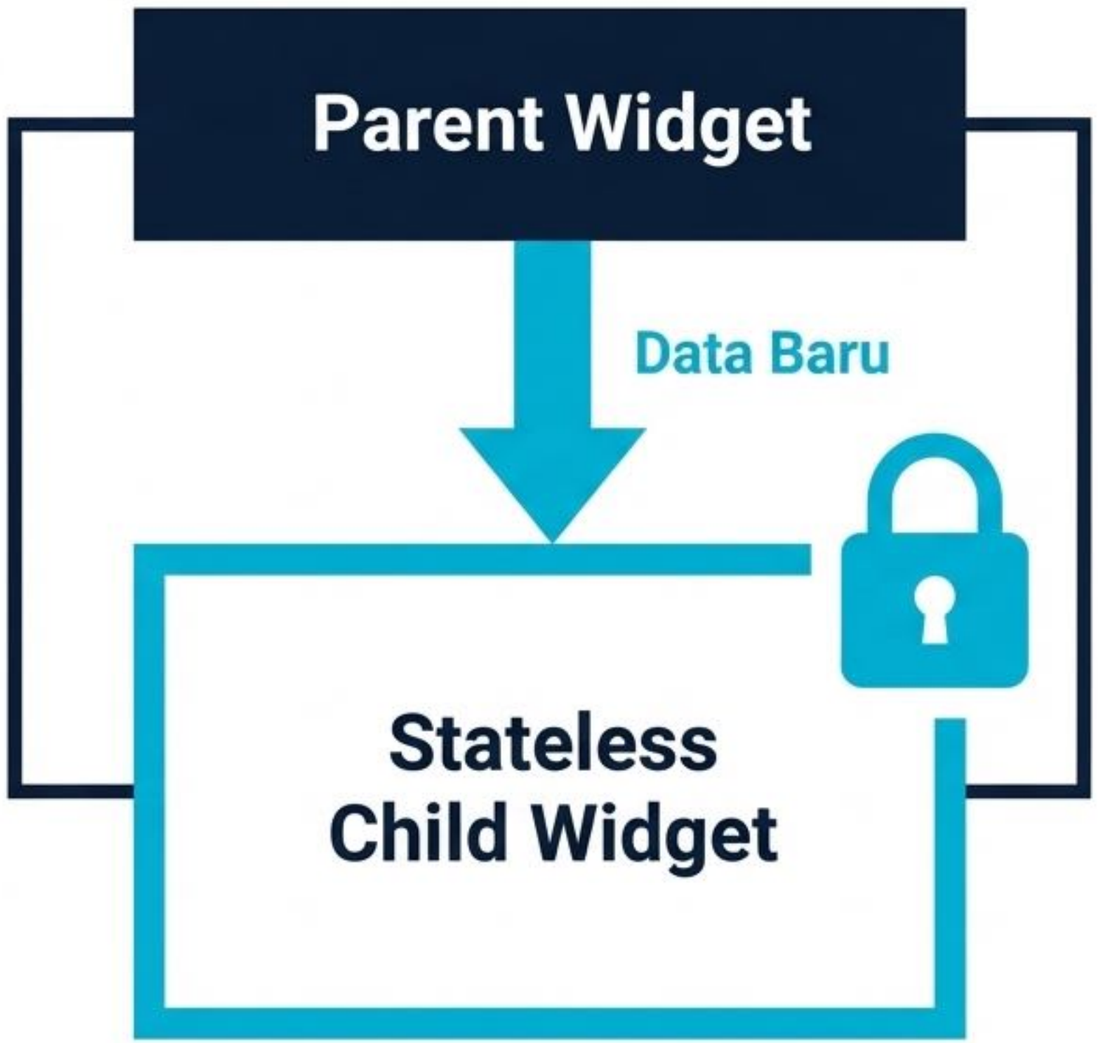
Tampilan (UI)
Tampilan akhir di layar pengguna.

Anatomi Stateless Widget

Elemen antarmuka bersifat statis.

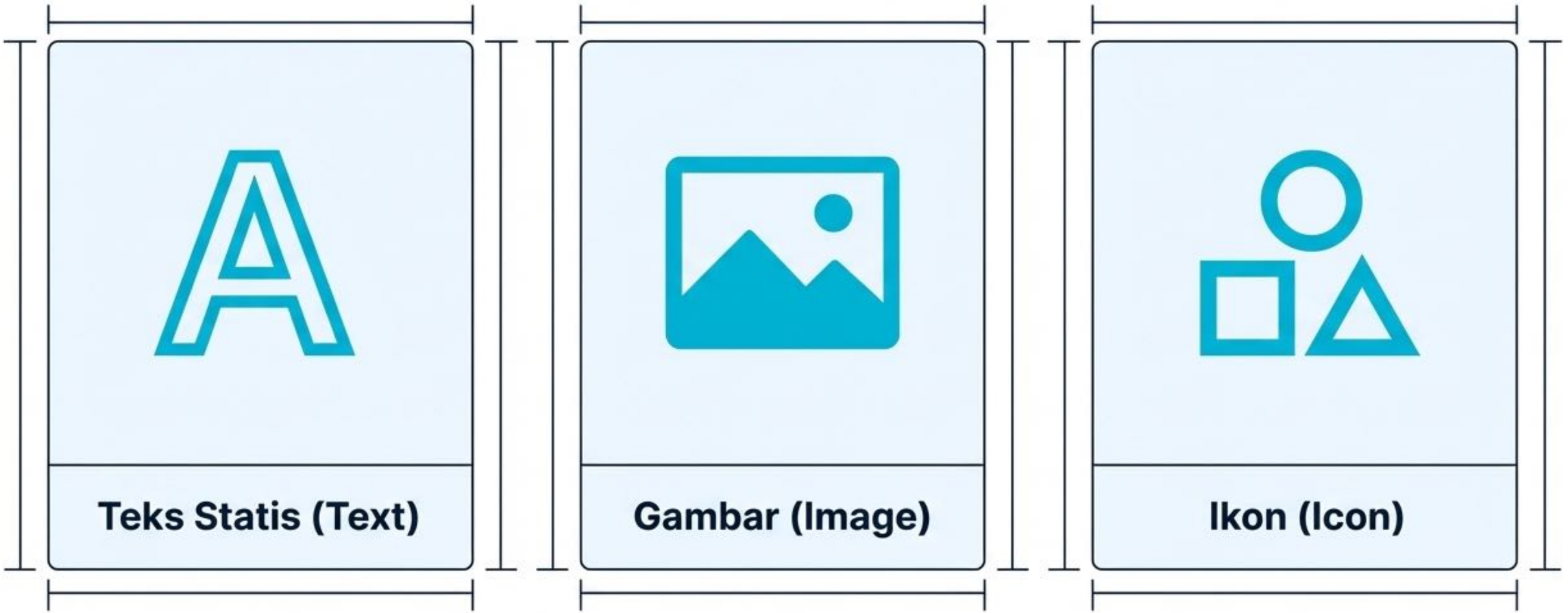
Tidak memiliki state internal yang dapat berubah setelah dibangun.

Hanya dibangun ulang dari awal jika terdapat perubahan pada parent widget.



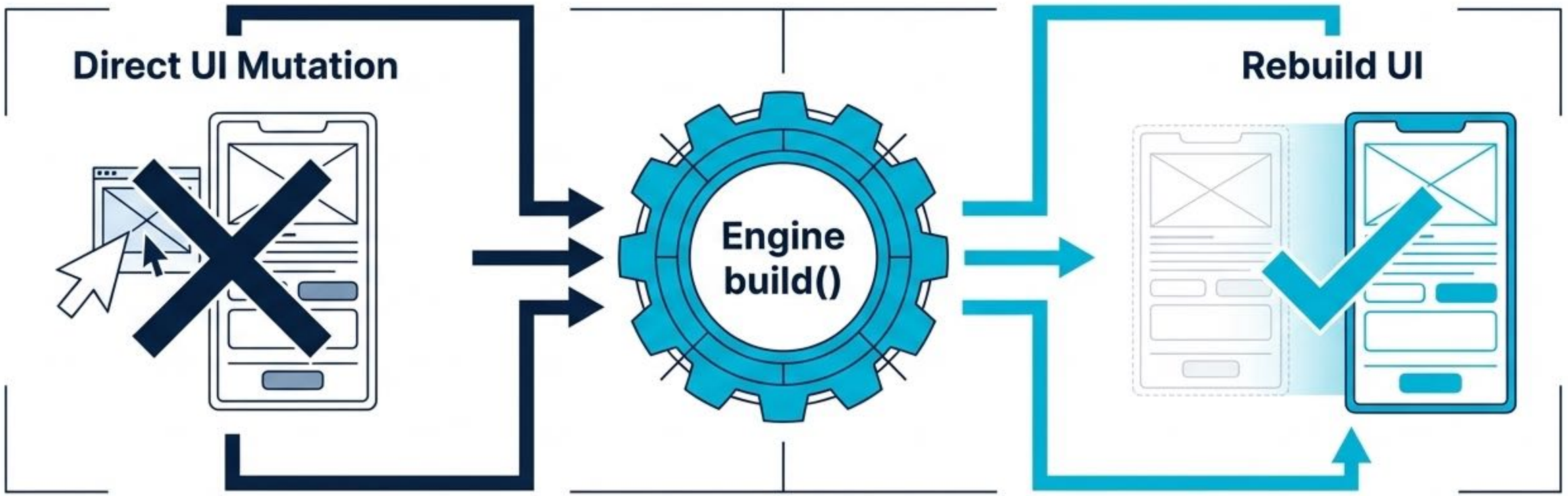
Katalog Stateless Widget Bawaan

Widget yang berfungsi murni menampilkan informasi tanpa interaksi data internal secara real-time.

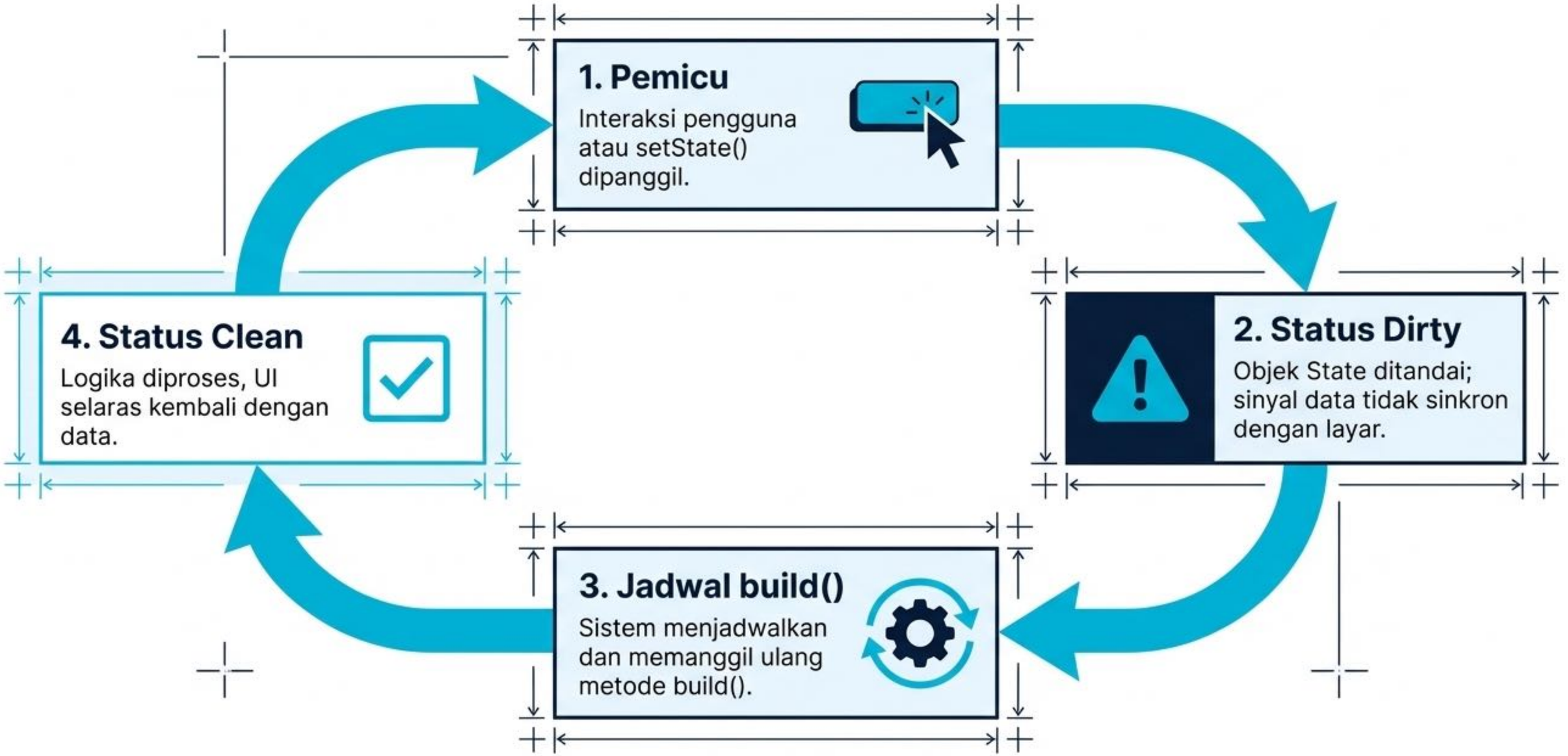


Mekanisme Pembaruan UI: Peran Sentral build()

Flutter tidak mengubah elemen UI secara langsung. Sistem membangun ulang (**rebuild**) widget untuk menyelaraskan tampilan dengan data terbaru. build() adalah metode yang paling sering dipanggil dalam siklus hidup widget.



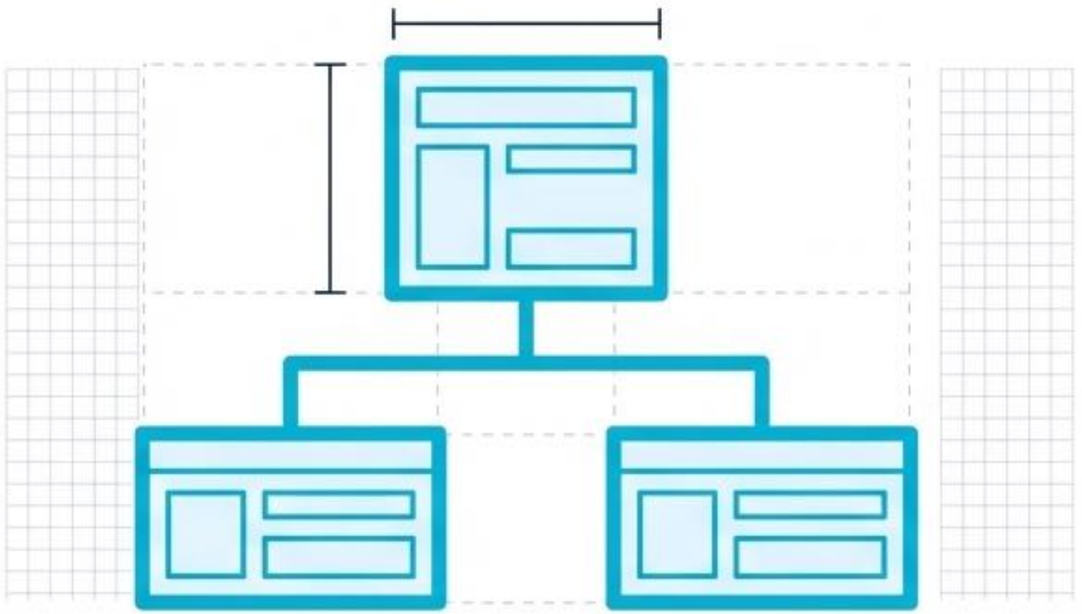
Siklus Sinkronisasi State



Efisiensi Rendering: Stateless vs. Stateful

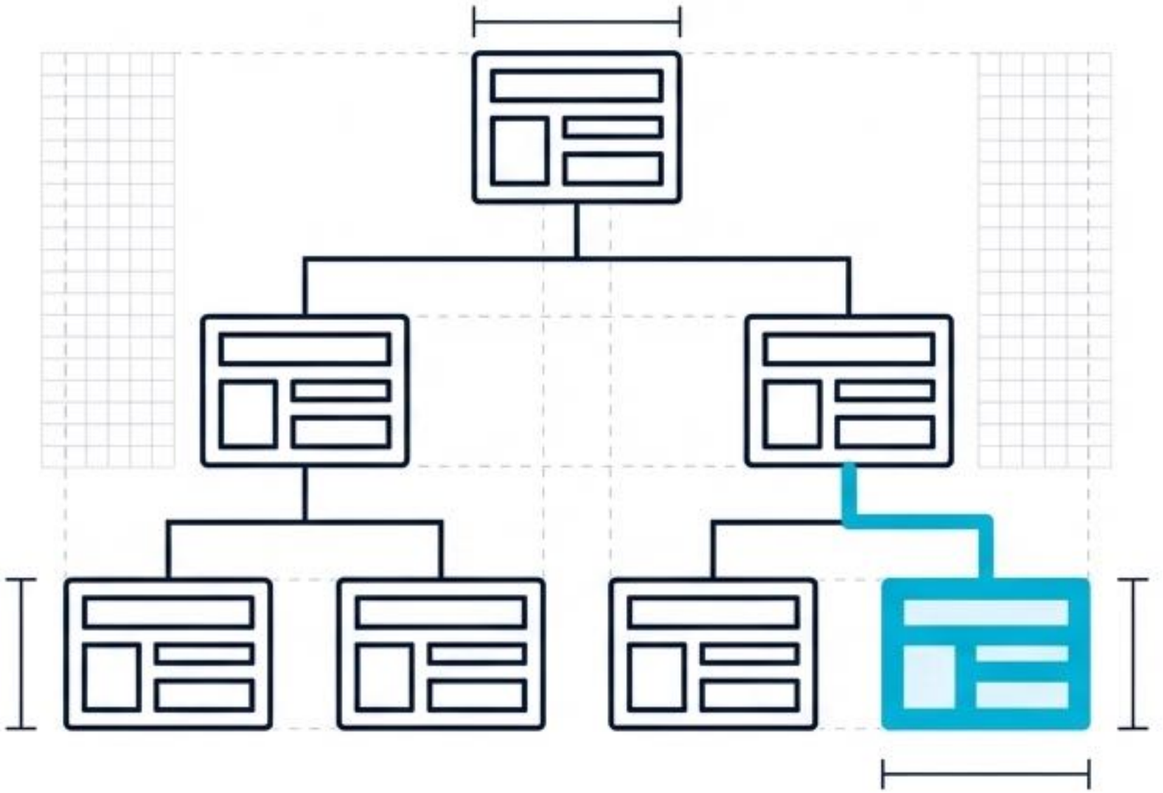
Stateless Widget

Dibangun ulang secara keseluruhan dari awal.



Stateful Widget

Sistem hanya memperbarui bagian UI spesifik yang berubah.



Matriks Perbandingan Arsitektur Widget

Dimensi	Stateless Widget	Stateful Widget
Sifat Dasar	**Statis**, tanpa state internal	**Dinamis**, memiliki state internal
Reaksi Data	Menampilkan info tanpa perubahan real-time	Merespons perubahan data real-time
Pemicu Rebuild	Perubahan pada Parent Widget	Pemanggilan setState() atau interaksi internal
Cakupan Update	Dibangun ulang secara **keseluruhan**	Hanya **bagian UI yang berubah** yang diperbarui

EDISI BLUEPRINT

Dynamic UI Management

Menguasai StatefulWIdget dalam Framework Flutter

Menghidupkan Antarmuka Pengguna

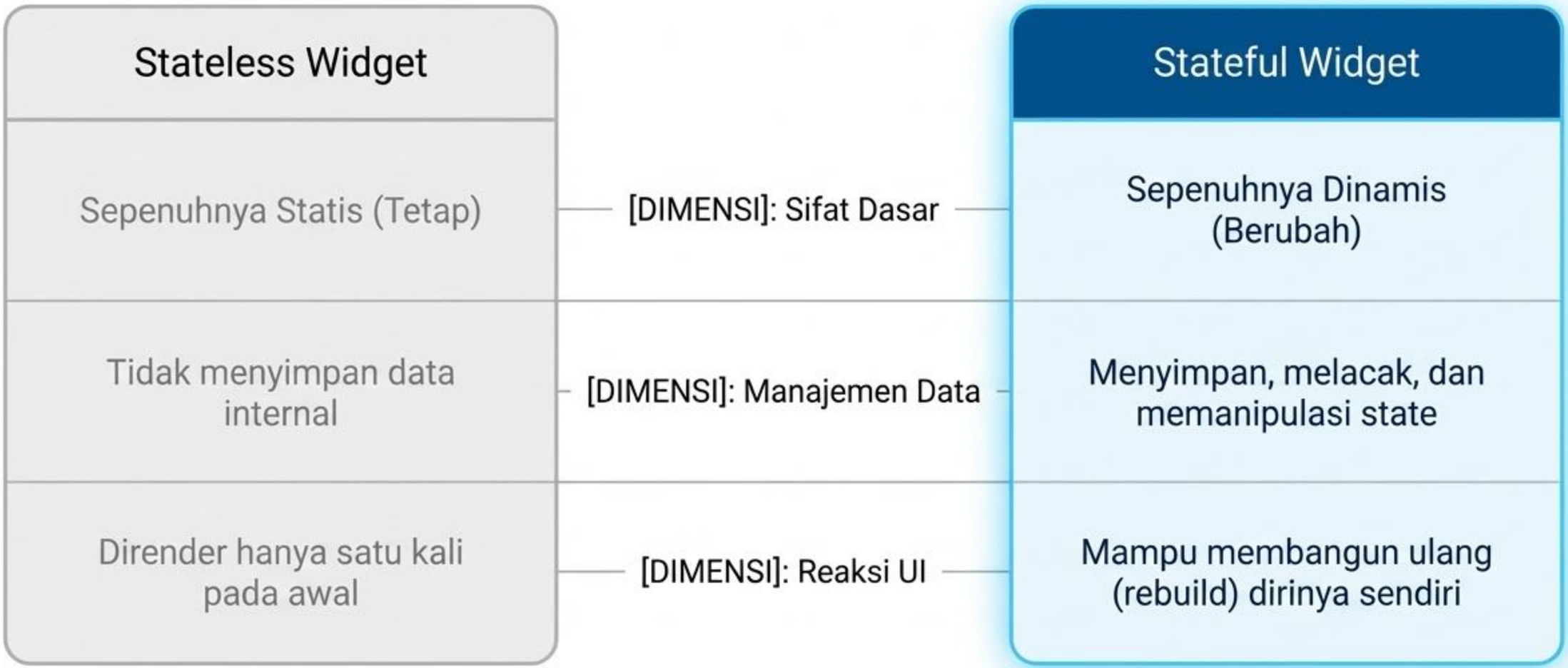
StatefulWidget adalah elemen antarmuka yang memiliki keadaan (state) yang dapat berubah secara dinamis selama aplikasi berjalan.

STATIC LAYOUT (Stateless)

DYNAMIC LAYOUT (Stateful)

1. **Keadaan Berubah (Mutable State)** – Data dan tampilan yang terikat dapat diperbarui di tengah runtime.
2. **Sangat Adaptif** – UI bereaksi secara real-time terhadap interaksi pengguna tanpa harus memuat ulang seluruh layar.

Memilih Pondasi Arsitektur



Mesin Siklus Rebuild

Bagaimana StatefulWIdget tetap sinkron dengan data terbaru.

1. Deteksi (Event/Input)

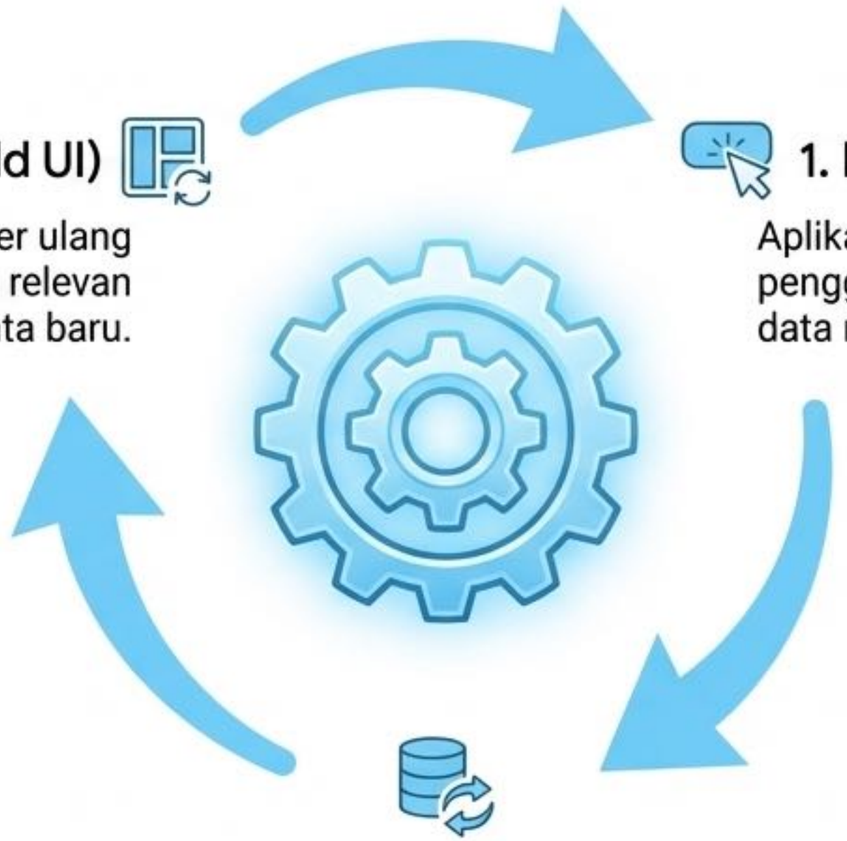
Aplikasi menangkap interaksi pengguna atau perubahan aliran data masuk.

2. Sinkronisasi (Update State)

Data terbaru disimpan di latar belakang, memicu pemberitahuan bahwa antarmuka sudah usang.

3. Bangun Ulang (Rebuild UI)

Flutter secara otomatis merender ulang hanya pada bagian UI yang relevan agar sesuai dengan data baru.

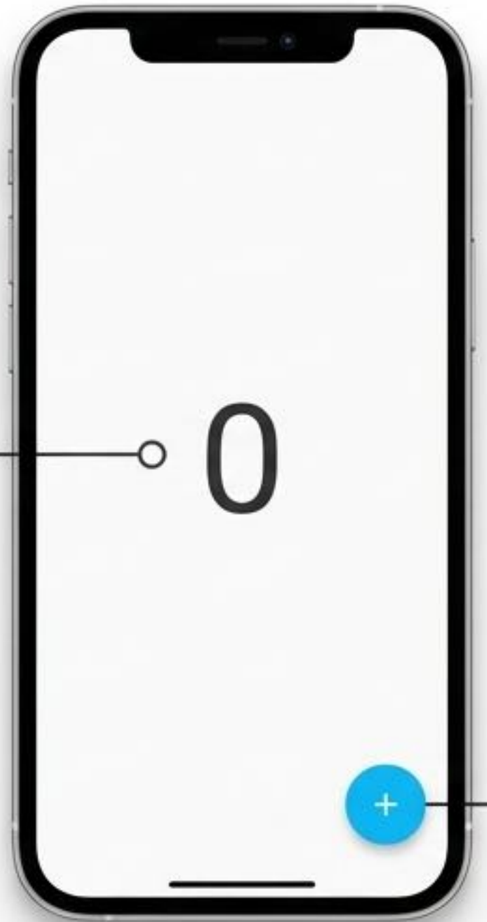


Skenario Praktis: Aplikasi Penghitung

Membedah anatomi state pada interaksi dasar.

State (Data Utama)

Variabel internal yang terus dipantau. Saat nilainya berubah, angka di layar ini akan di-rebuild secara otomatis.



Pemicu (Trigger)

Komponen interaktif yang, setiap kali ditekan, mengeksekusi fungsi untuk menambah angka dan menginstruksikan pembaruan state.

```
onPressed: () {
  setState(() {
    counter++;
  });
}
```

Keseimbangan Ekosistem Flutter

Performa Ringan

Interaktivitas Dinamis

Gunakan Statefulwdget hanya ketika elemen antarmuka Anda secara eksplisit perlu bereaksi dan merender ulang dirinya sendiri secara dinamis. Jika UI murni informasional, gunakan Stateless untuk menjamin performa yang optimal.

Dynamic when needed, static when possible.

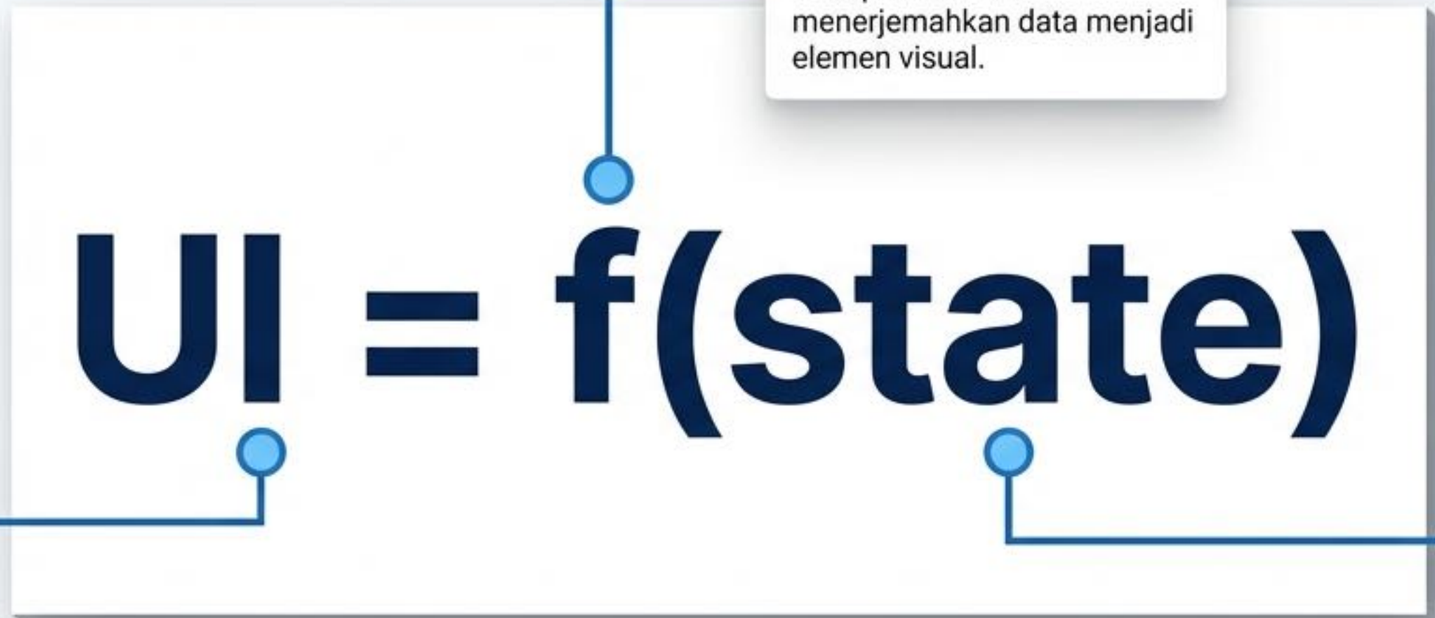
Mastering Dynamic UI: Panduan Esensial Flutter StatefulWidget

Memahami arsitektur, siklus hidup, dan efisiensi manajemen antarmuka dinamis.

Formula Fundamental Antarmuka Dinamis

Fungsi Build

Metode build() yang memproses dan menerjemahkan data menjadi elemen visual.



Tampilan Layar

Antarmuka pengguna yang direnderer secara visual.

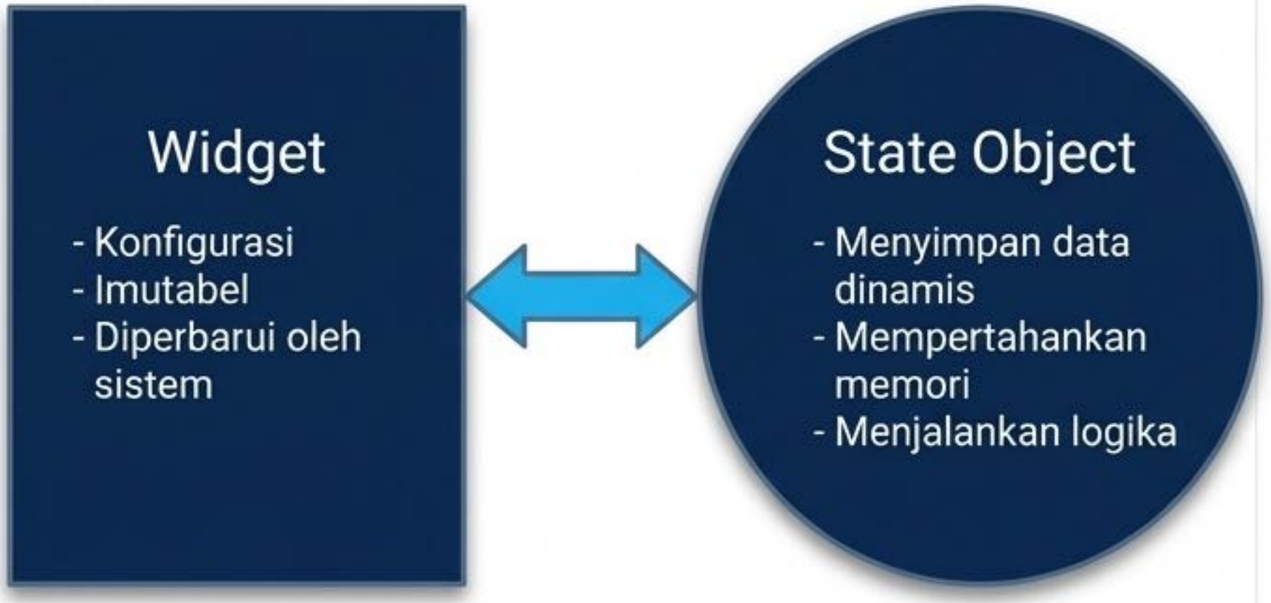
Data Dinamis

Variabel dan status yang dapat berubah selama aplikasi berjalan.

Prinsip dasar Flutter: Tampilan tidak digambar manual secara statis, melainkan direkonstruksi secara otomatis setiap kali state berubah.

Anatomi StatefulWidget: Konfigurasi vs Eksekusi

StatefulWidget adalah jenis widget yang memegang objek Status (State), mendeteksi perubahan data, dan menampilkan kembali (rebuild) layar agar antarmuka pengguna selalu selaras dengan data terbaru.



Memisahkan definisi antarmuka (Widget) dari data yang berubah-ubah (State).

4 Pilar Karakteristik Utama

Dinamis

Tampilan dapat diperbarui secara real-time berdasarkan interaksi pengguna atau perubahan data internal.

Efisiensi Pembaruan

Sistem dirancang presisi; hanya bagian UI yang berubah saja yang diperbarui, bukan membangun ulang seluruh layar dari nol.

Kompleksitas

Memiliki siklus hidup (lifecycle) bertahap dan membutuhkan logika manajemen state yang terstruktur.

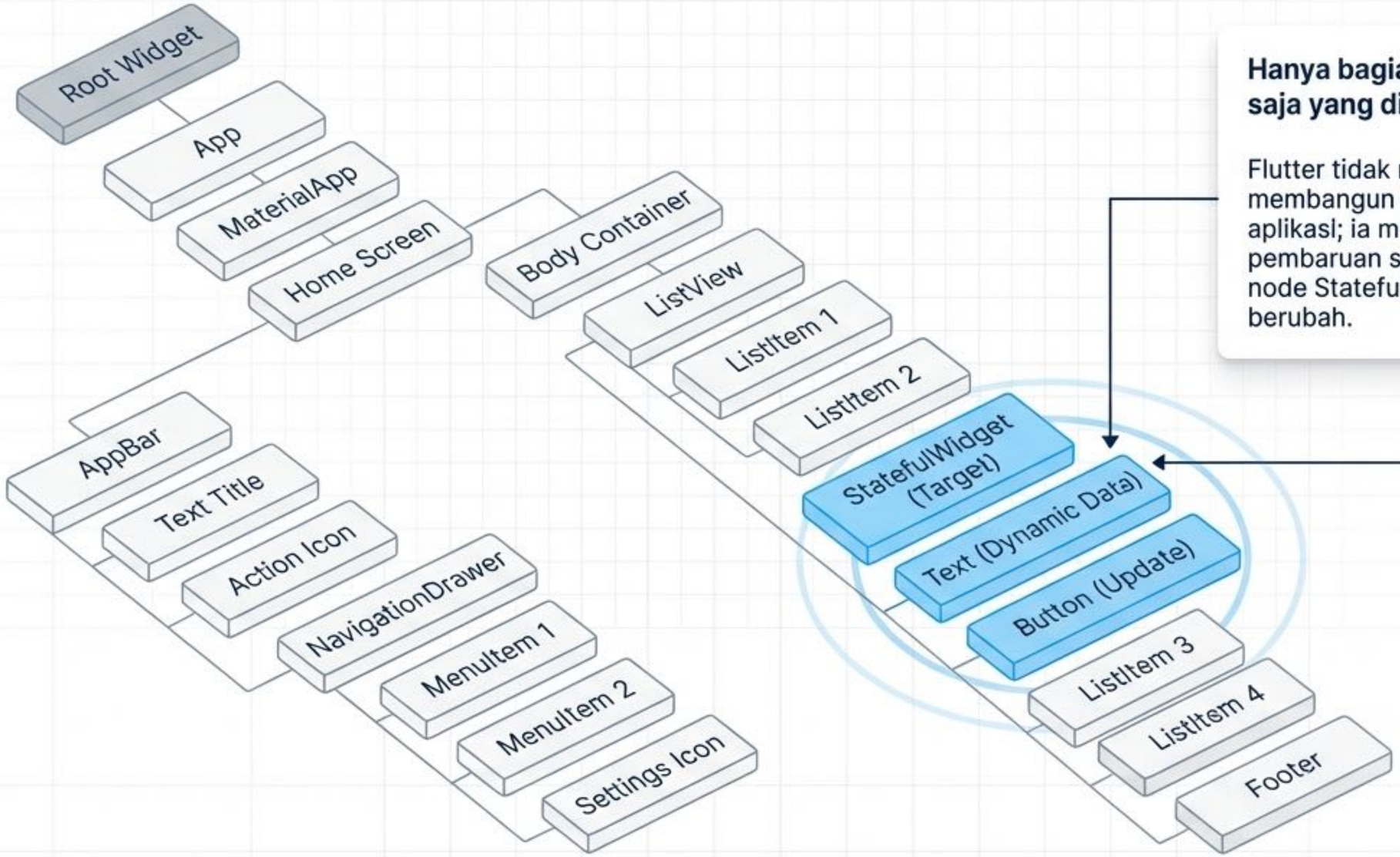
Fokus Penggunaan

Ideal untuk elemen interaktif tinggi seperti formulir input, penghitung (counter), atau daftar belanja dinamis.

Matriks Diagnostik: Kapan Menggunakan Stateful?

	Stateless Widget	**Stateful Widget**
Sifat	Statis (Tidak berubah setelah dirender)	**Dinamis (Bereaksi terhadap interaksi)**
Siklus Hidup	Sederhana (Hanya fungsi build)	**Kompleks (Memiliki banyak fase)**
Kinerja & Memori	Ringan & Sangat Cepat	**Membutuhkan alokasi memori lebih untuk objek State**
Kasus Penggunaan	Teks label, Ikon statis, Tata letak dasar	**Formulir login, Animasi interaktif, Daftar data real-time**

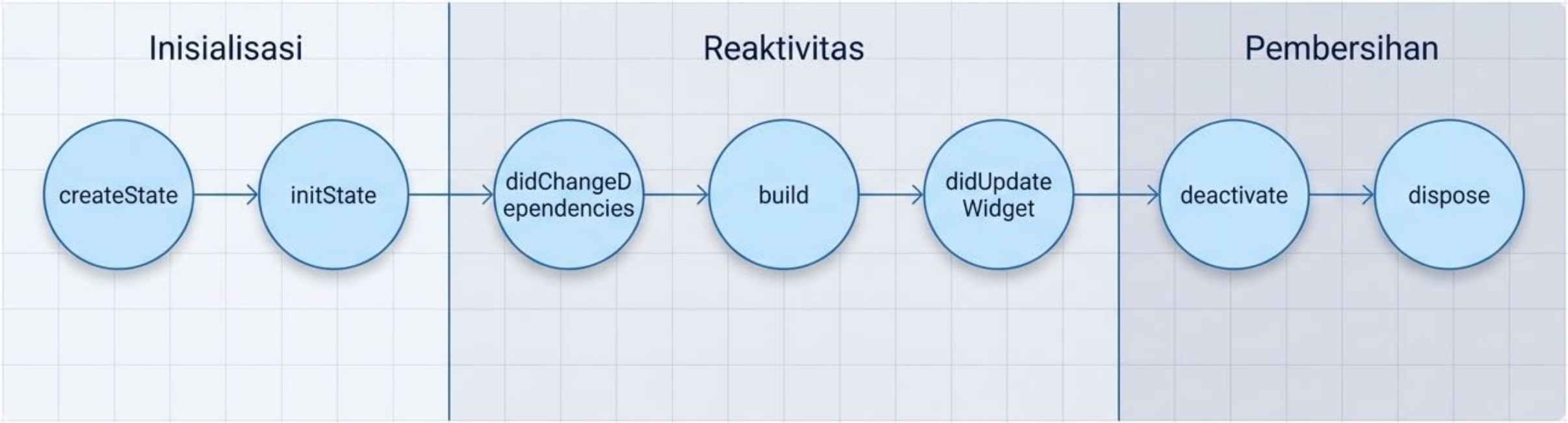
Efisiensi Terpusat: Isolasi Pembaruan Antarmuka



Hanya bagian UI yang berubah saja yang diperbarui.

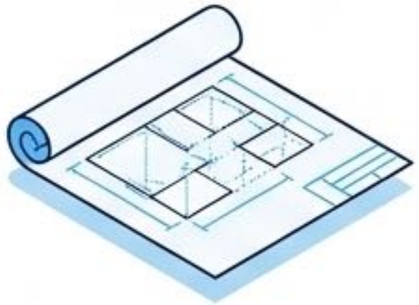
Flutter tidak menghancurkan dan membangun ulang seluruh struktur aplikasi; ia melokalisasi pembaruan secara presisi pada node StatefulWidget yang datanya berubah.

Peta Perjalanan: Siklus Hidup (Lifecycle) Widget



Memahami urutan ketujuh fase ini adalah kunci untuk mengelola sumber daya secara efisien dan mencegah kebocoran memori.

Fase 1: Inisialisasi & Pondasi



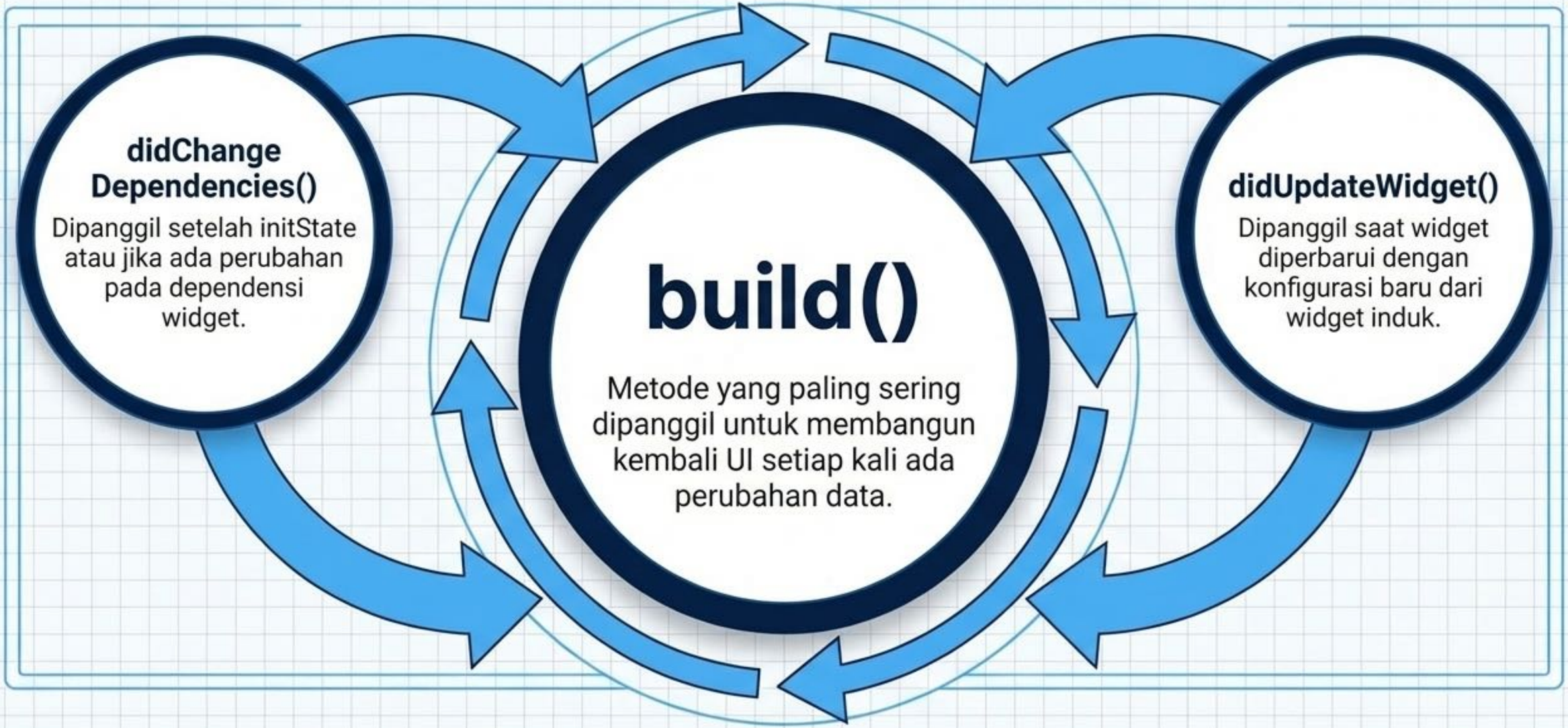
createState()

Membuat objek State yang akan menampung logika widget.

InitState()

- Dipanggil hanya satu kali di awal masa hidup widget.
- Tujuan Utama: Melakukan inisialisasi awal, mengatur listener, atau mengambil aliran data eksternal (API calls).
- Aturan Emas: Tidak boleh digunakan untuk memicu pembaruan antarmuka secara langsung.

Fase 2: Siklus Reaktivitas Utama



Fase 3: Pembersihan Sumber Daya

deactivate()

Dipanggil saat widget dikeluarkan sementara dari pohon widget (widget tree).

dispose()

Tahap akhir kehidupan widget.

Fase Kritis! Gunakan **dispose()** untuk membersihkan sumber daya secara permanen (seperti menghentikan listener, timer, atau controller). Kegagalan melakukan ini akan menyebabkan kebocoran memori (memory leak) yang menurunkan performa aplikasi.

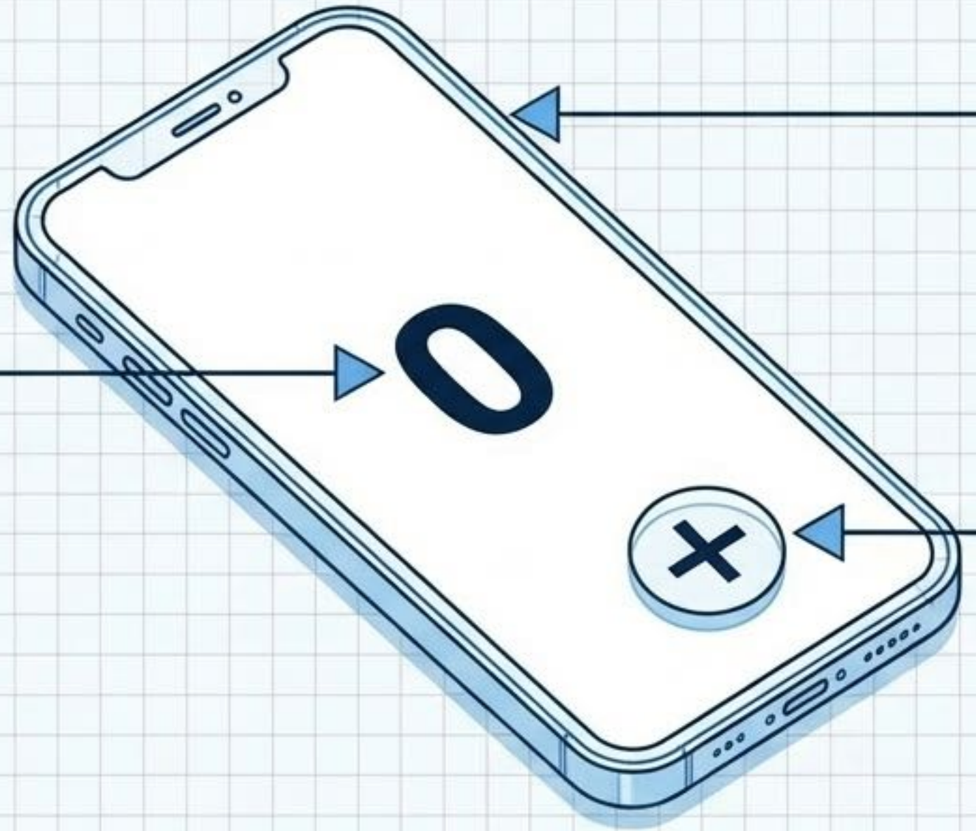
Sintesis Fungsional: Anatomi Aplikasi Penghitung

Metode build()

Hasil render visual dari nilai state saat ini.

Objek State

Data dinamis yang disimpan di memori.



Pemicu Interaksi

Elemen yang akan memanggil mutasi data saat disentuh pengguna.

Mekanika Pembaruan Waktu Nyata (Action & Reaction Loop)

1. Tindakan Pengguna

Pengguna menekan tombol '+'.

2. Mutasi State

Fungsi setState() dipanggil, nilai variabel angka bertambah (0 → 1).

3. Pemicu Rebuild

Flutter mendeteksi perubahan State dan secara otomatis memanggil ulang fungsi build().

4. Pembaruan UI

Antarmuka baru dirender ke layar, menampilkan angka '1'.

Prinsip Utama Manajemen Antarmuka Dinamis

1. Dinamis Namun Presisi

Gunakan **StatefulWIdget** untuk elemen yang membutuhkan pembaruan real-time (formulir, penghitung), namun pertahankan efisiensi karena Flutter hanya merender ulang area yang terdampak.

2. Rumus Universal

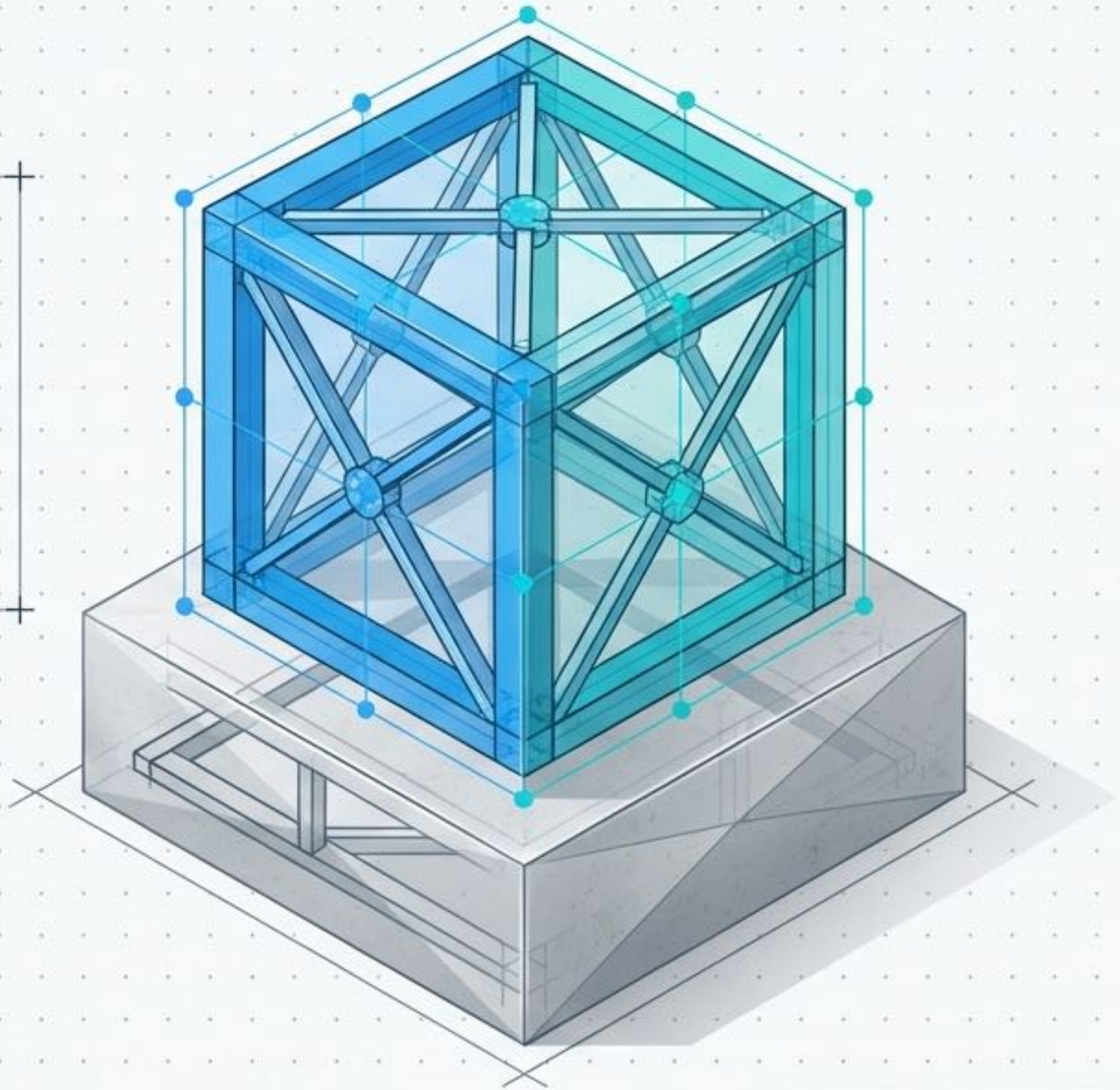
Selalu ingat bahwa UI = f(state). Tampilan hanyalah pantulan matematis dari data Anda pada detik tersebut.

3. Disiplin Siklus Hidup

Inisialisasi koneksi di initState() dan pastikan selalu membersihkan sumber daya di dispose() untuk menghindari kebocoran memori.

Architecting Mobile Experiences with Flutter & Dart

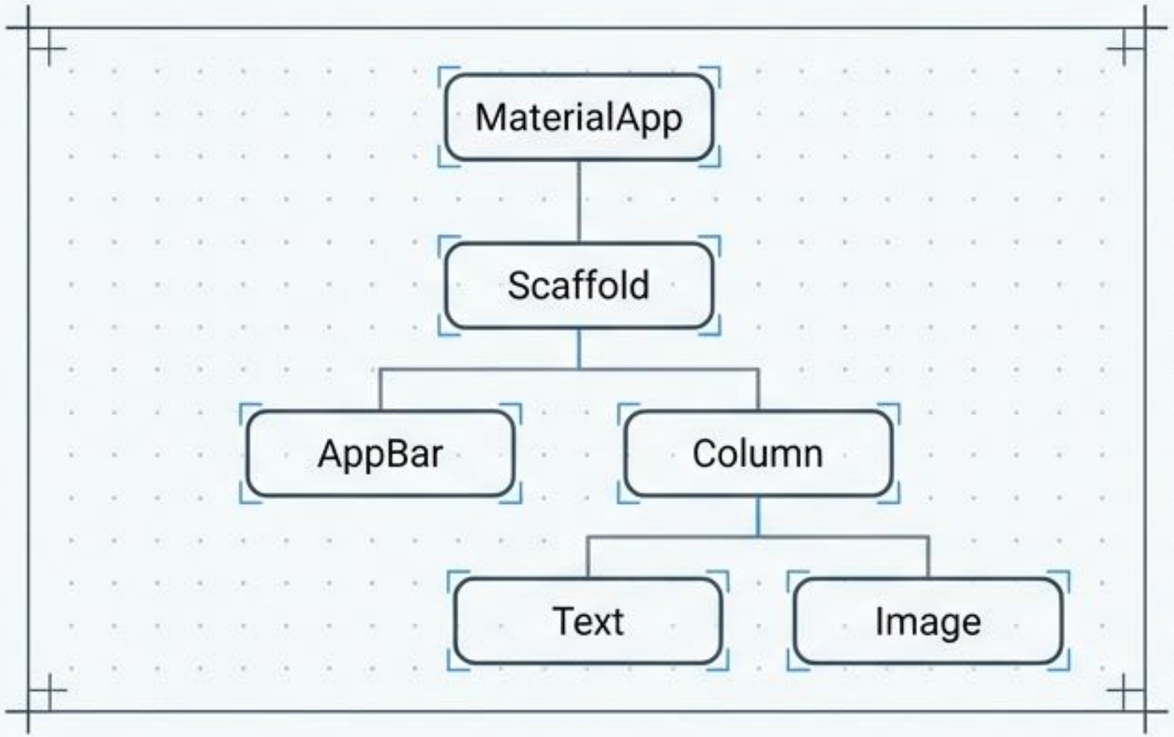
From foundational paradigms to production deployment.

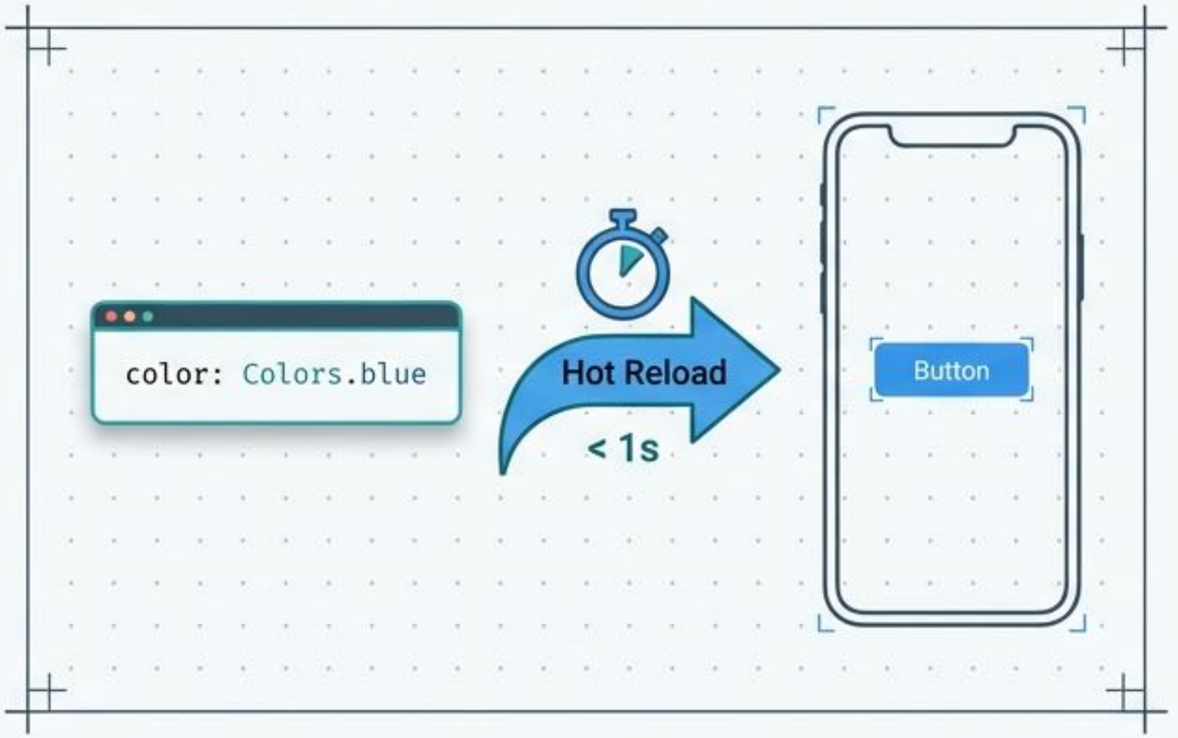


Framework Blueprint // v1.0

The Framework Paradigm: Everything is a Widget

Flutter constructs high-performance, cross-platform applications by composing small, modular UI components into a reactive tree.





Single Codebase

Native compilation for multiple platforms.

Fast Development

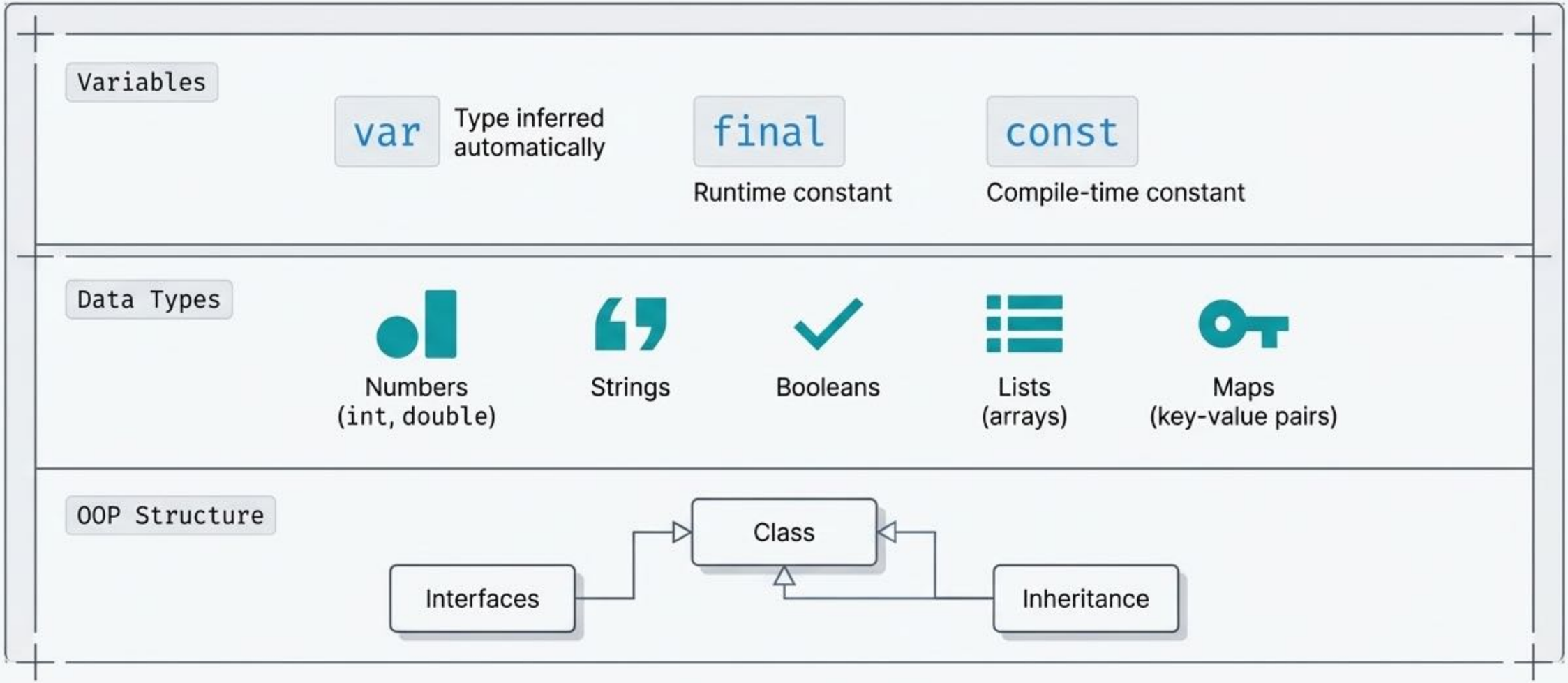
Sub-second stateful Hot Reload.

Reactive UI

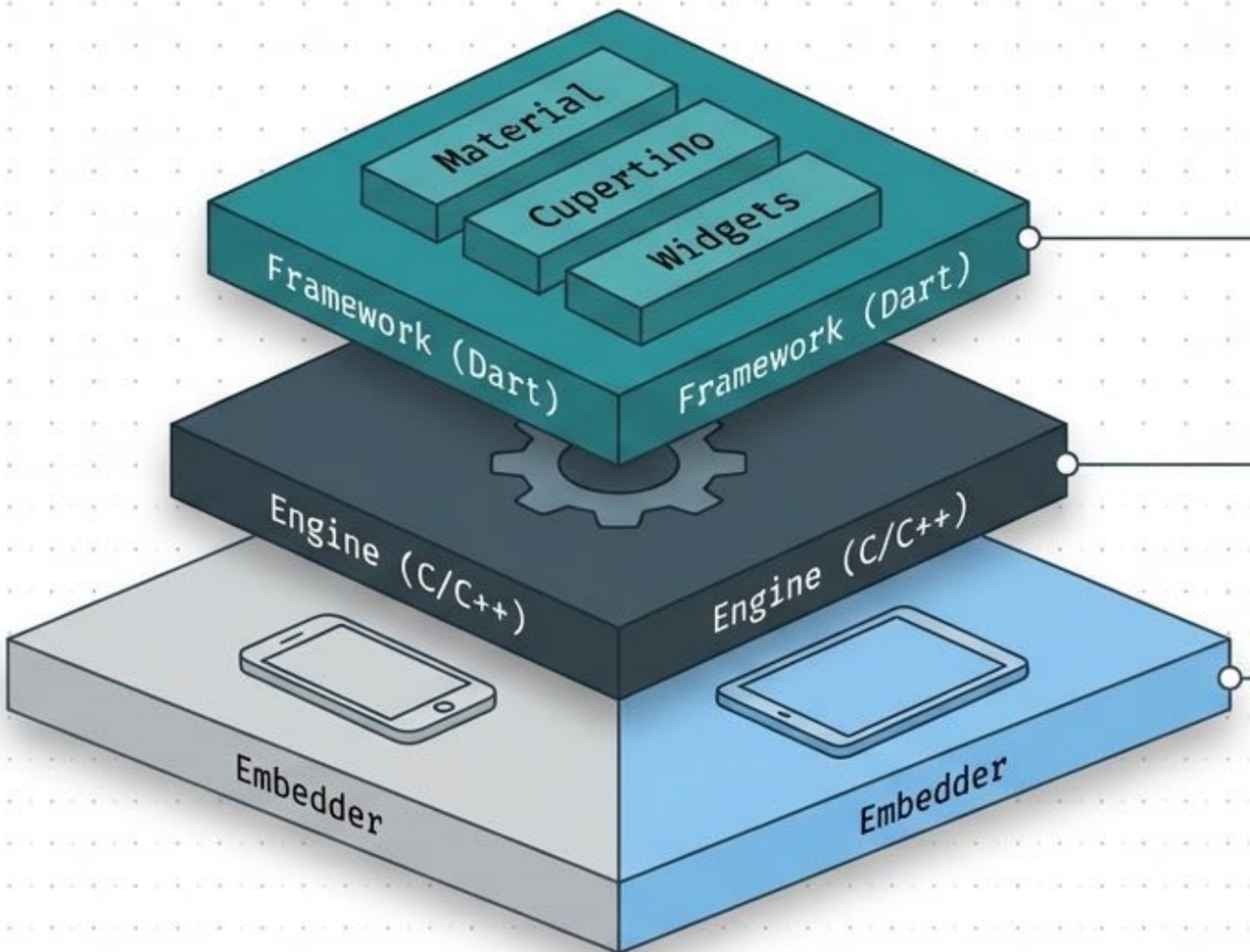
Declarative interface driven by data.

Powering the Engine: The Dart Language Foundation

Dart is a modern, object-oriented language with C-style syntax, optimized specifically for fast client-side UI rendering.



Bypassing OEM Abstractions: The Architectural Stack

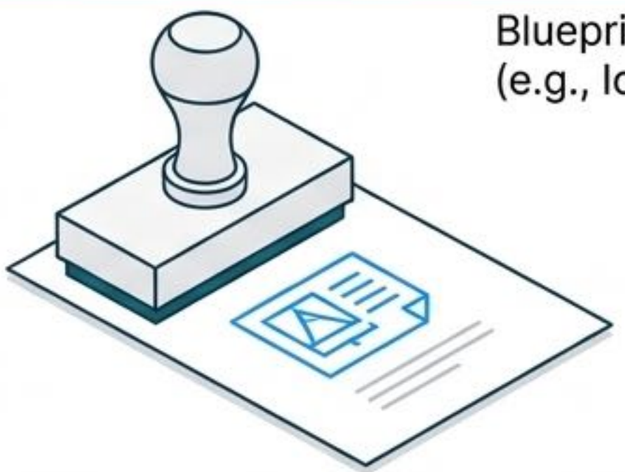
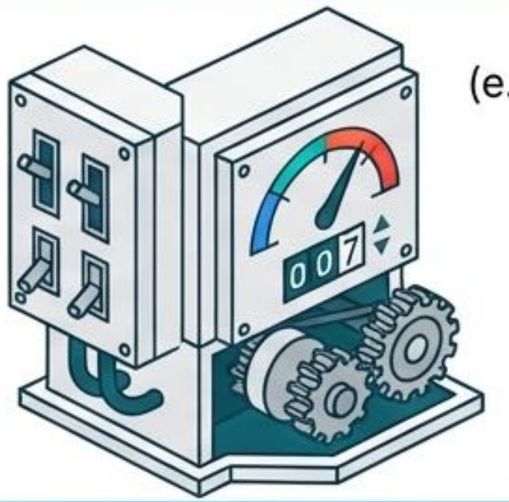


Handles widgets, animations, gestures, and layout logic.

Paints directly to the canvas, ensuring consistent 60+ FPS performance.

Platform-specific integration layer (Android, iOS).

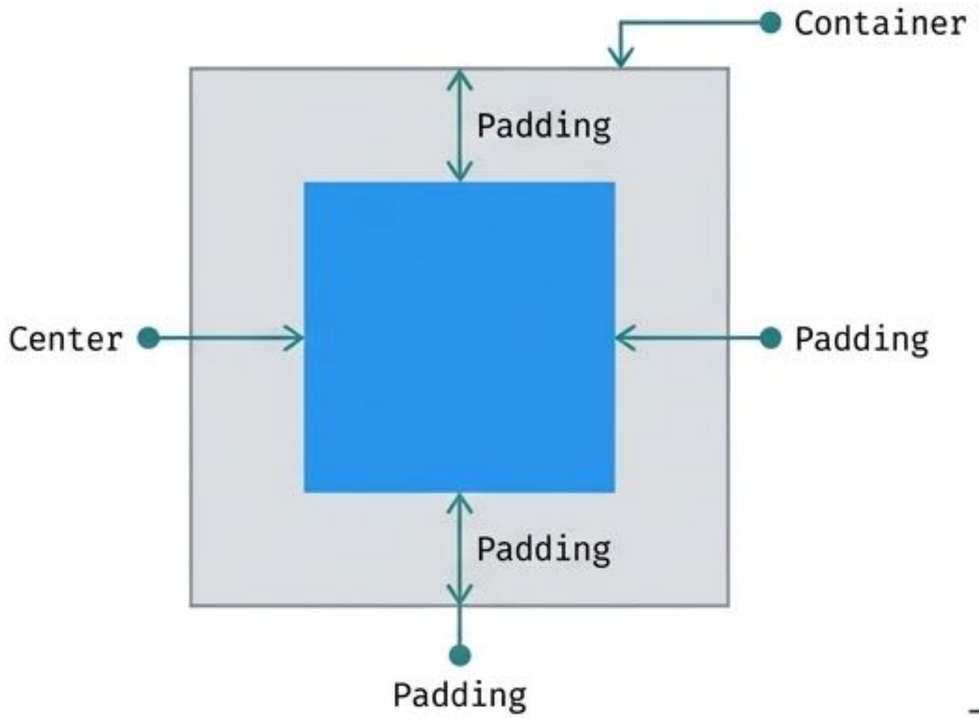
The UI Core Matrix: Managing Widget Lifecycle

	`StatelessWidget`	`StatefulWidget`
Definition	Static UI that never changes during runtime.	Dynamic UI with internal memory (state).
Mutability	Immutable.	Mutable based on interaction or data.
Conceptual Metaphor	 Blueprint Stamp (e.g., Icon, Text)	 Active Engine (e.g., Counter, Form)

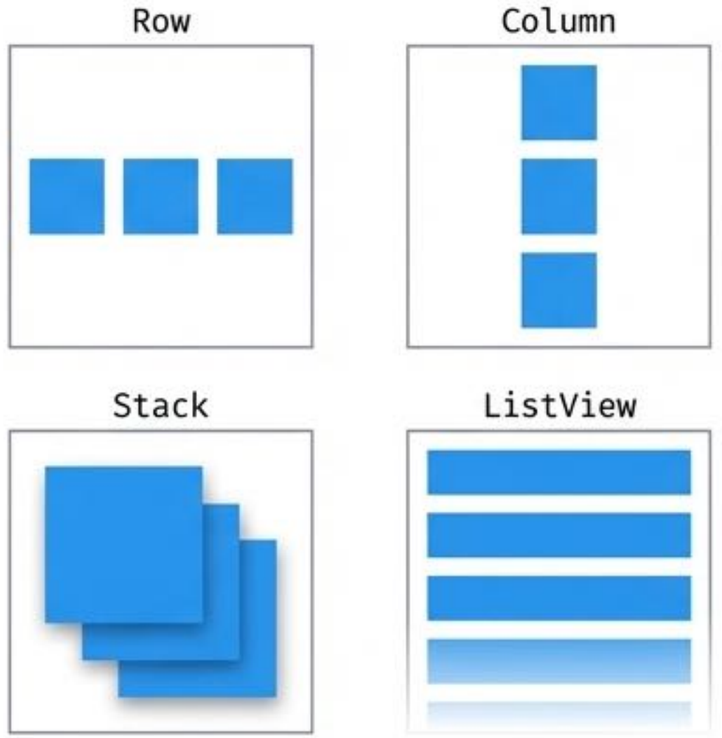
Structuring Space: The Layout Widget Matrix

Flutter layout dictates precise control over the visual canvas, divided by widget capacity.

Single-Child Constraints

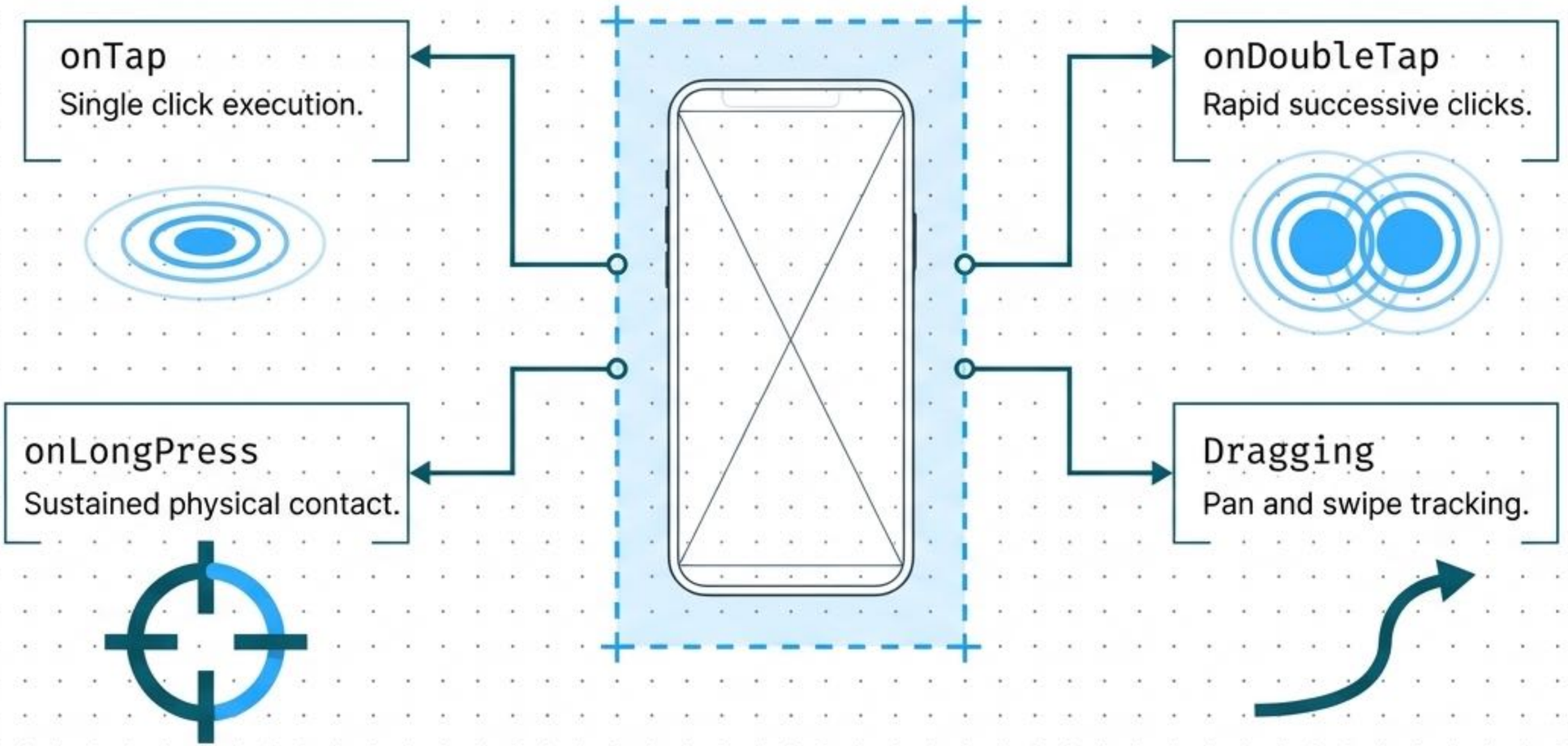


Multi-Child Arrays



Capturing Physical Intent: Gesture Recognition

The GestureDetector is an invisible wrapper that translates physical user interactions into logical responses.



Moving Through the Application: The Navigation Stack

Routes

In Flutter, individual screens are conceptualized as `Routes`.

The Navigator

The core widget managing the LIFO history stack.

Named Routes

Explicitly defined string paths registered at the `MaterialApp` level.

Breathing Life into the UI: Managing Application State

Ephemeral State

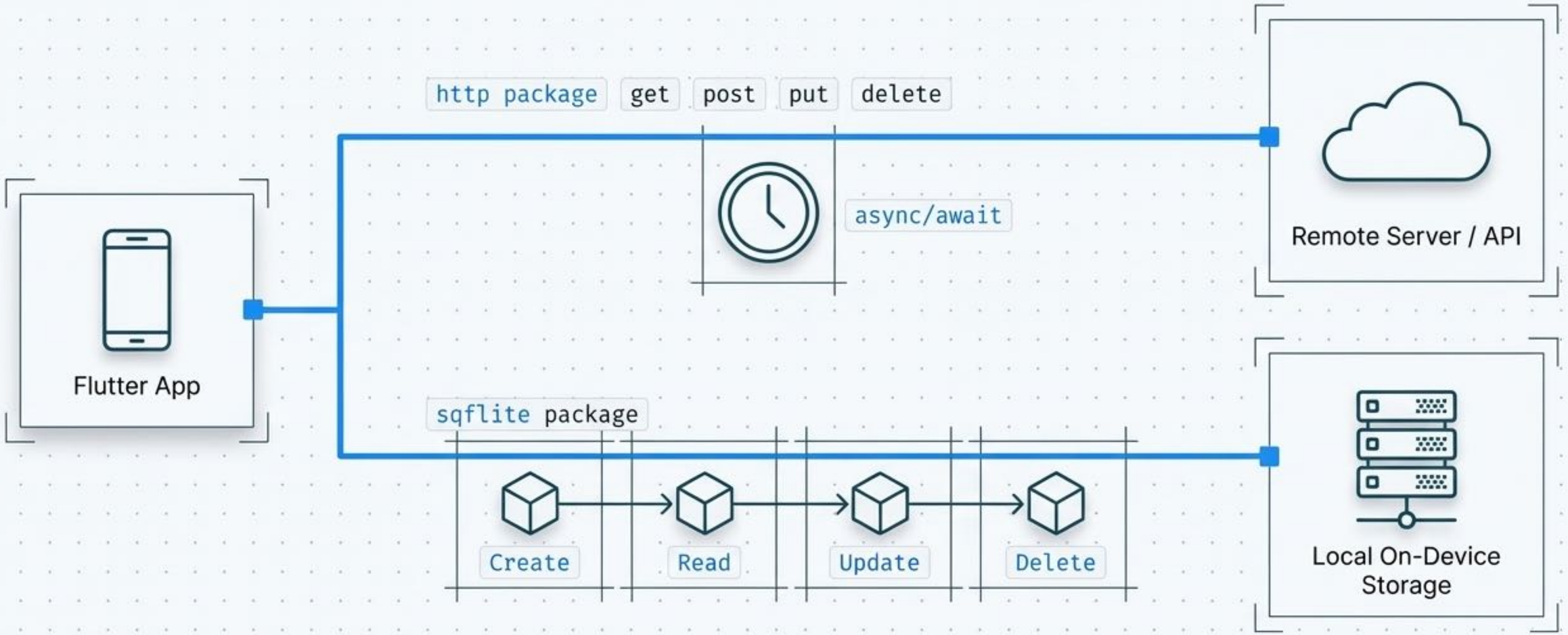
Localized, short-lived data isolated to a single route. Managed natively via setState.

App State

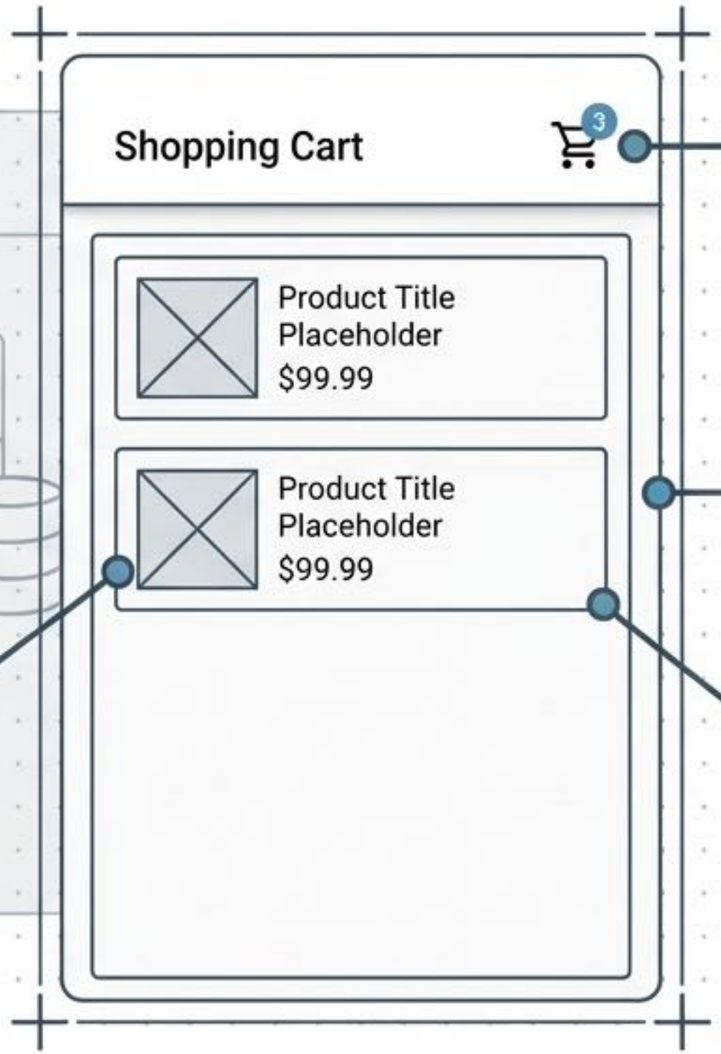
Shared data required across multiple parts of the application. Managed via tools like Provider or global variables.

Integrating the Outside World: Network and Local Data

Applications act as data synthesis layers, requiring seamless integration with remote servers and local on-device storage.



Synthesis: Architecting the Shopping Cart



Data Source: Product Class models data fetched via `http` and cached in `sqflite`.

Media: Product images registered in `pubspec.yaml` assets.

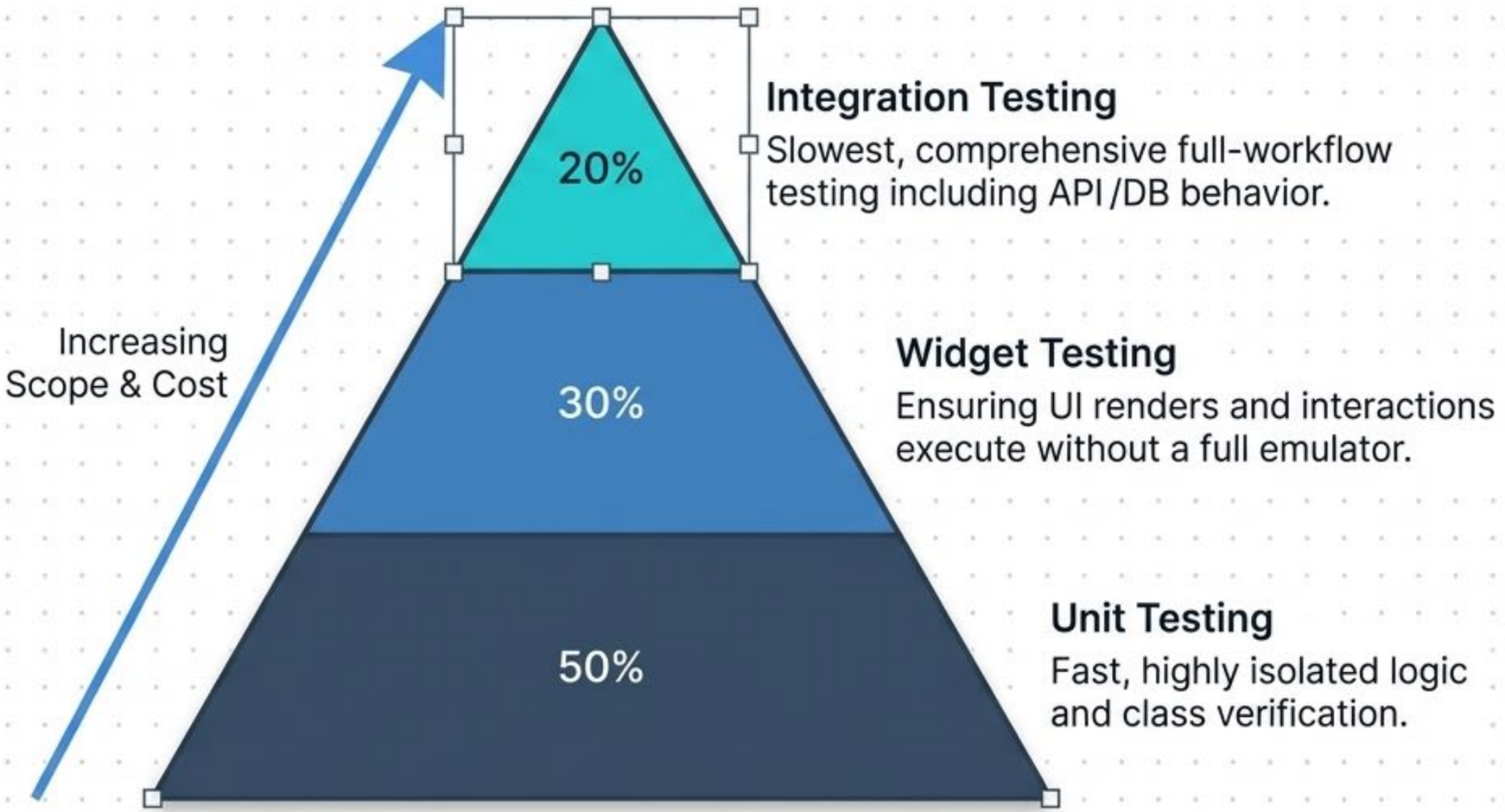
AppBar Badge: App State dynamically updates the Badge total.

Scrollable Body: `ListView.builder` renders multiple dynamic children.

Row Items: `ListTile` structures product leading image, title, and subtitle price.

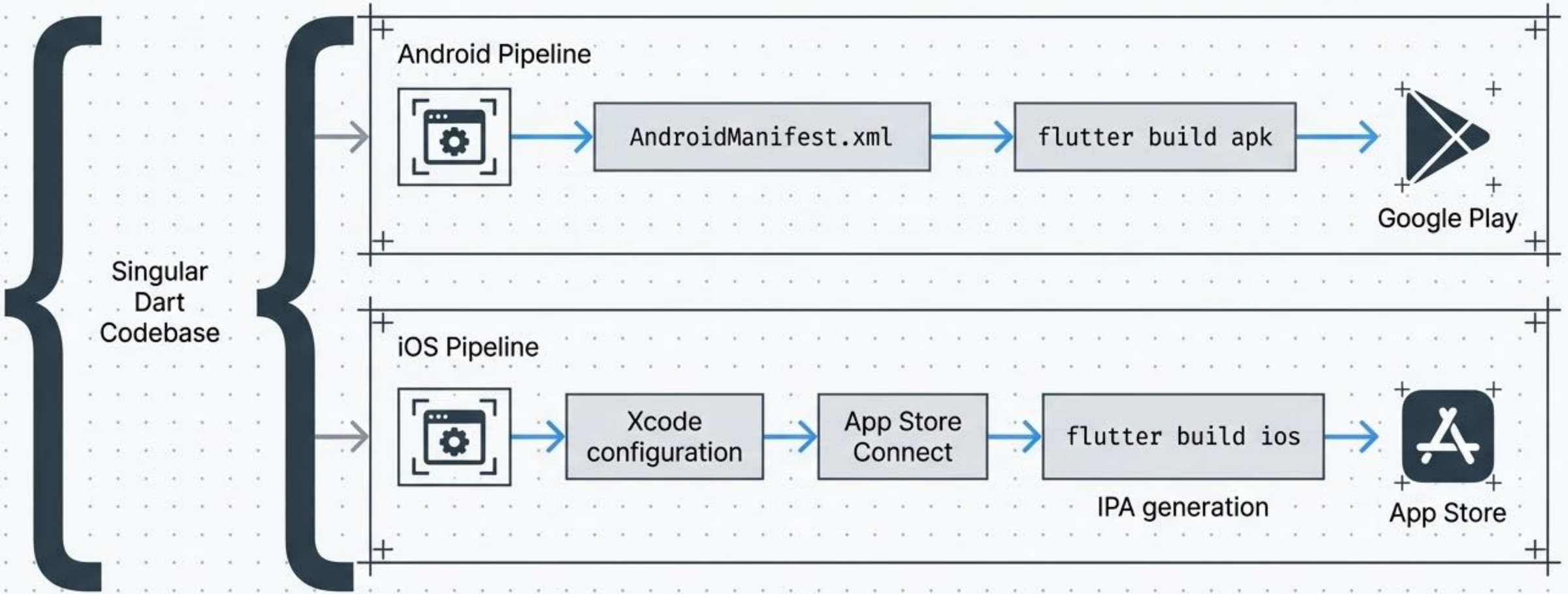
Ensuring Structural Integrity: The Testing Pyramid

Quality assurance in Flutter is layered, moving from isolated code functions up to full-scale user simulations.



From Code to Production: The Deployment Pipeline

While the Dart codebase is singular, the final build process diverges into platform-specific configurations and artifacts.



Blueprint Best Practices: Crafting Clean Code

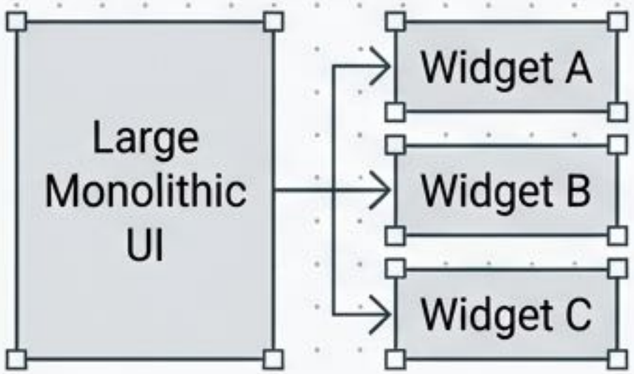
Optimize Mutability

Always utilize **const** constructors for static widgets to reduce memory footprint and bypass unnecessary rendering.

```
const Text('Hello')
```

Componentization

Deconstruct massive monolithic UI files into smaller, encapsulated, custom widgets across separate files for maintainability.



Resource Culling

Rigorously audit **pubspec.yaml** to remove unreferenced images, assets, or dead code before publication to minimize application payload size.

```
~~assets/unused_icon.png~~
```

Kesimpulan Pertemuan #3

Stateless vs Stateful: Gunakan Stateless untuk antarmuka statis yang ringan dan efisien. Gunakan Stateful untuk antarmuka dinamis yang membutuhkan interaksi dan penyimpanan data mandiri.

initState vs setState: Ingat aturannya: initState untuk setup awal (hanya sekali), dan setState untuk memicu rebuild UI (berkali-kali sesuai interaksi).

Lifecycle Awareness: Memahami siklus hidup layar (Android & Flutter) adalah syarat wajib untuk membangun aplikasi yang stabil, hemat baterai, dan terhindar dari crash tak terduga.