Fuchsia 简介

韩冰天 2021 0702

Fuchsia 是什么

- 一个智能设备OS
- google 出品，目前已开放并接受社区贡献
  - A production-grade operating system that is secure, updatable, inclusive, and pragmatic
- 一个全新的，从kernel重写的操作系统

google 开发fuchsia可能的原因

- Linux kernel的掣肘，Android和linux面向的使用场景并不一致，有些Android的改动在服务器场景上没有价值却增加了linux维护的复杂性，导致android的代码在linux的主分支上的合入有一定困难，双方在驱动的license方面也产生过不同的观点。
- Java 的掣肘，Android为了吸引开发者采用了Java作为UI框架的开发语言，Framework也大部分由Java编写，导致运行时开销较大，内存消耗也多于iOS，另外Java API版权也与Oracle产生过争议
- 一个完全掌控的，能够实现全部想法的（如解决碎片化问题），下一代更先进的操作系统（前提是解决性能问题），安卓还是有一些缺点
- 推测，Fuchsia 是谷歌试图使用单一操作系统去统一整个生态圈的一种尝试，Fuchsia 的目标是能够在谷歌的技术保护伞下，运行于智能手机、智能音响、笔记本电脑等任何合适的设备之上。

Fuchsia 目标

- 是Android的替代品吗：不确定
  - 安卓的最大问题是碎片化，在这一点上fuchsia有优势
  - 安卓已经如此成功，取代不是容易的事
- 是一个试验性的项目吗
  - NO，是一个商业项目，目前有产品落地 nest hub 一代
- 仍然在快速演进中

fuchsia的目标设备

- 64位cpu，x86-64 or arm64
- 较为快速的cpu和一定数量的内存
    - 大概率不是面向IOT设备
- 个人计算设备：phone，laptop。。。

Fuchsia的一些基本设计概念

- **secure**

  powered by new （Micro）kernel Zircon

- **updatable**

  powered by new （Micro）kernel Zircon

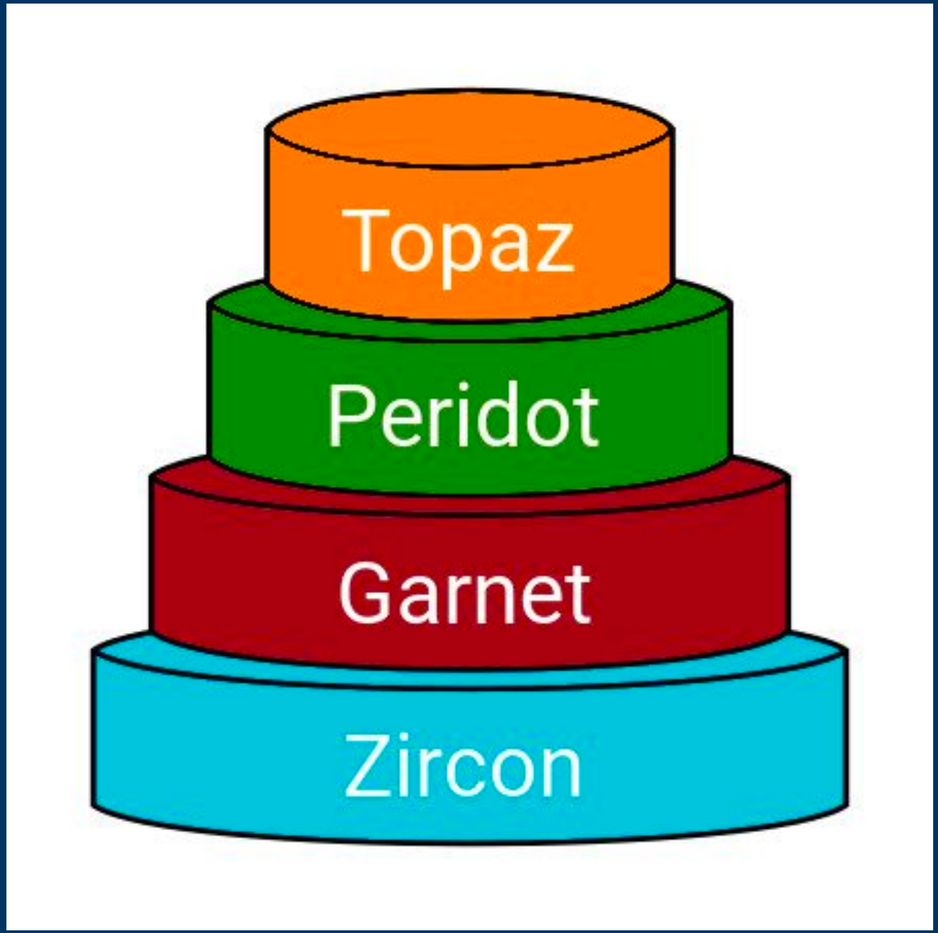
- **inclusive**

  Fuchsia is an open source project that currently supports a variety of languages and runtimes, including C++, Web, Rust, Go, Flutter, and Dart.

- **pragmatic**

  Fuchsia is not a science experiment, it’s a production-grade operating system that must adhere to fundamentals, like performance.

Fuchsia的结构



Fuchsia的四层蛋糕结构，调用关系自上而下，每层可以单独替换，概念上清晰完美，只是。。。

Deprecated！！

Why？ to be continued

- https://fuchsia-china.com/the-4-layers-of-fuchsia
- https://fuchsia.dev/fuchsia-src/contribute/open_projects/srcs/layer_cake_deprecation?hl=en

Fuchsia的 Updatable

- **Almost all software on Fuchsia is a component**

  Component 是fuchsia的软件执行单元

  A hermetic composable isolated unit of software

  https://fuchsia.dev/fuchsia-src/concepts/components/v2?hl=en

- **Software is interchangeable and reusable**

  Fuchsia Interface Definition Language (FIDL) enables loose coupling between components

- Push updates and security patches to all products on demand
- Update the system without modifying the driver

  （On the roadmap）

https://fuchsia.dev/fuchsia-src/concepts/principles/updatable?hl=en

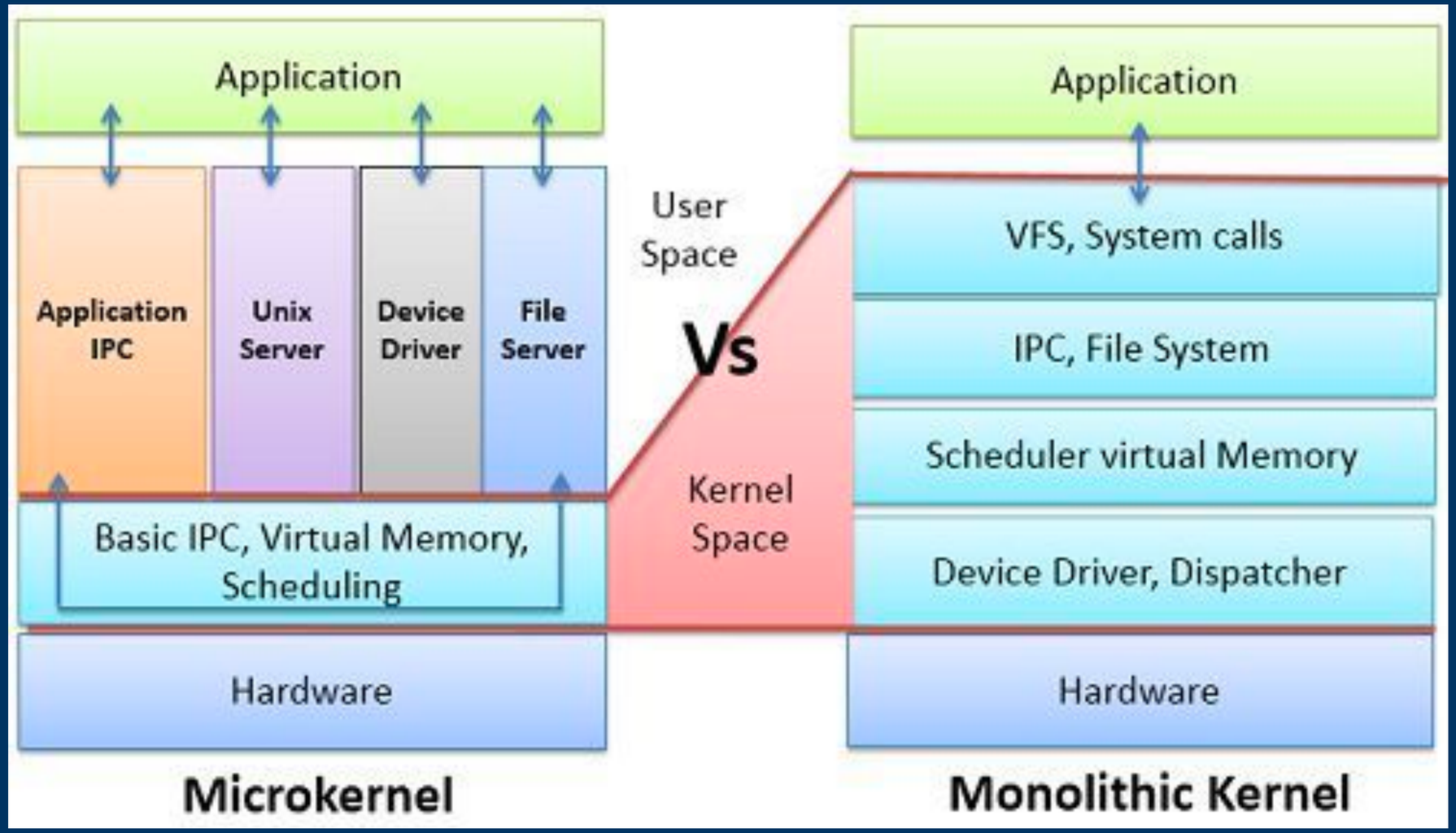
The Power of Zircon

- 解耦，模块化，解决升级问题，减少碎片化（微内核的优点）
  - Sys Call能否足够稳定,driver 的API/FIDL 能否足够稳定？像HAL一样版本化?
  - 解耦是系统工程
- 安全性（微内核的优点）
- 性能是否有短板
  - 已经不宣扬是微内核了，driver host 减少IPC，哪里有性能问题哪里集中?

The Problem of Micro kernel

- **Performance**
  - 过多的IPC，Context 切换会导致性能瓶颈
- **不成熟，缺乏业界认可所以缺乏业界的支持**
  - 目前是google自己的工程师支持硬件，何时芯片厂商开始支持Fuchsia是一个可能的关键转折点
  - 驱动生态缺失

Micro kernel & monolithic Kernel



Micro kernel & monolithic Kernel

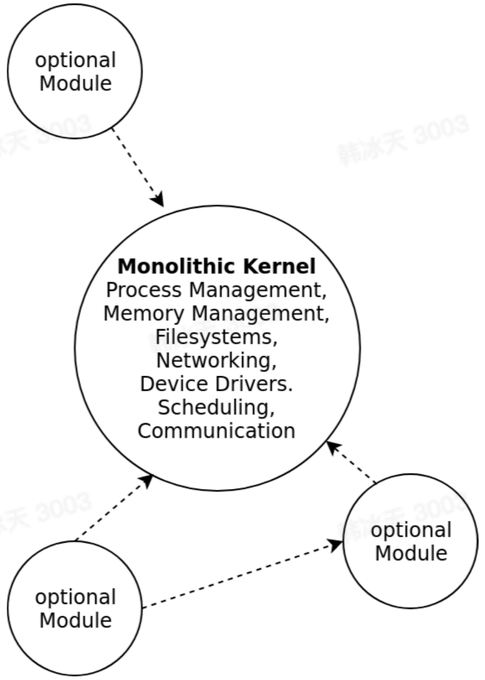


Figure 2.3: A Monolithic Kernel Architecture according to [26]

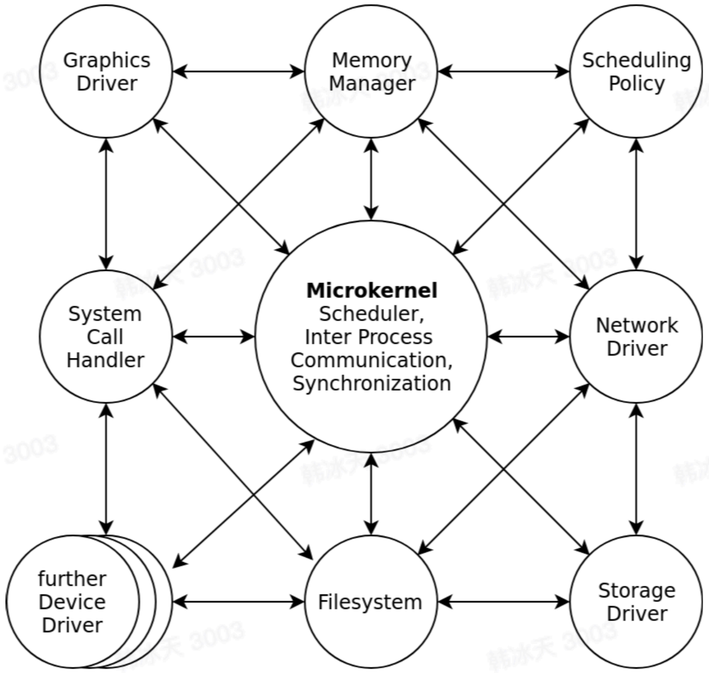


Figure 2.4: A Microkernel Architecture according to [26]

Zircon 是微内核吗?

- **Zircon 官方说不是**

  Zircon is a pragmatic, message-passing kernel—not a microkernel

  Although Fuchsia applies many of the concepts popularized by microkernels, Fuchsia does not strive for minimality. For example, Fuchsia has over 170 syscalls, which is vastly more than that of a typical microkernel. Instead of minimality, the system architecture is guided by practical concerns about security, privacy, and performance.

  Fuchsia is a new open source, micro-kernel-like operating system from Google. （官网）

- **VFS，Driver，Networking 都不在kernel**
  - **首先不是宏内核**
  - **其次。。。**
  - **个人觉得是**

Zircon 一些概念

Performance is a priority

Asynchronous communication reduces latency

Fuchsia makes heavy use of asynchronous communication, which reduces latency by letting the sender proceed without waiting for the receiver. This is important for delivering software that can come and go on a device as needed, to account for network latency.

Fuchsia does not yet achieve its performance goals, but this is an area under active development. For example, performance related storage enhancements are on the project roadmap。

Zircon 一些概念

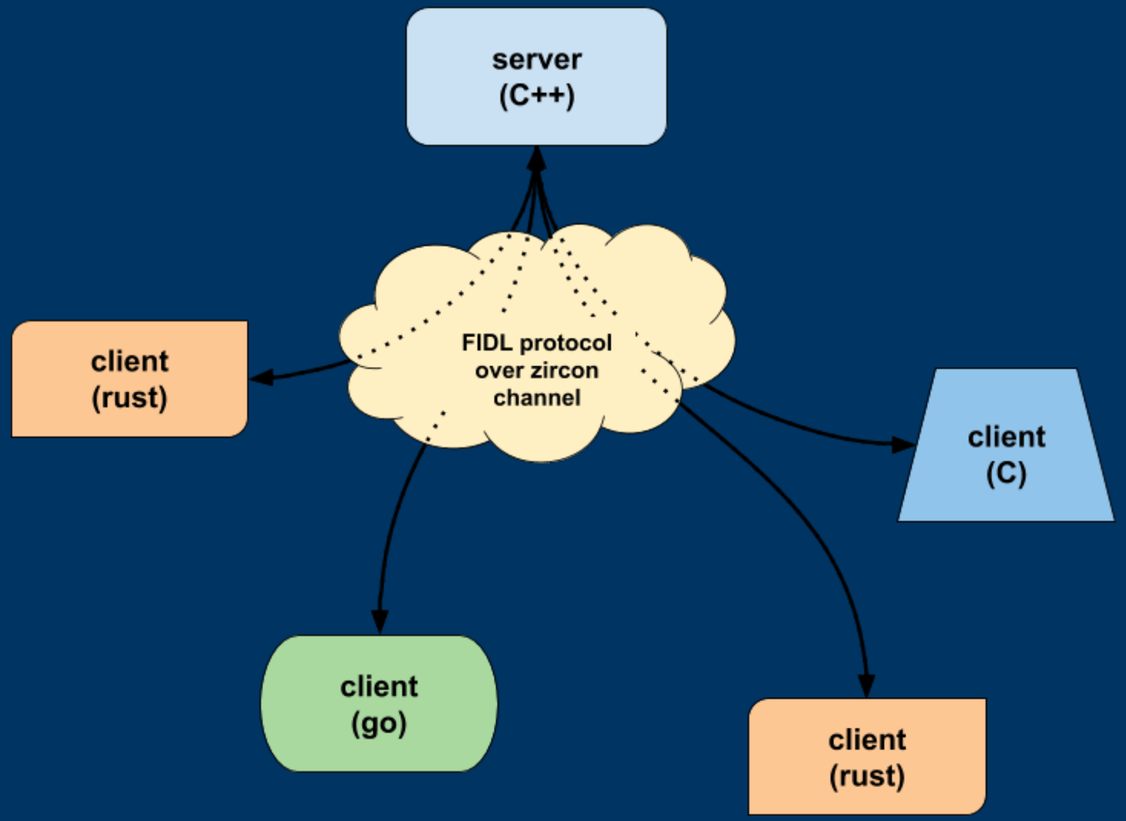
More safety system calls

Write in C++

with Object concepts

https://fuchsia.dev/fuchsia-src/concepts/kernel/concepts?hl=en&authuser=1

IPC protocol in Fuchsia



efficiency

determinism

robustness

ease of use

FIDL/Channel是最主要的RPC/IPC方式，就像AIDL/Binder，但范围大得多，包括driver

https://fuchsia.dev/fuchsia-src/concepts/fidl/overview?hl=en#why_use_fidl

Component

- (几乎）所有的都是component，包括driver，services
- 由manifest描述
- 通过Capability（能力）描述，管理，发现，协作
- 有生命周期
- 由Component Manager管理
- 在sandbox中运行
- FIDL是component的通讯方式

模块化（Modular）—deprecated in favor of Session framework

- **何为模块：于用户进行交互的component**

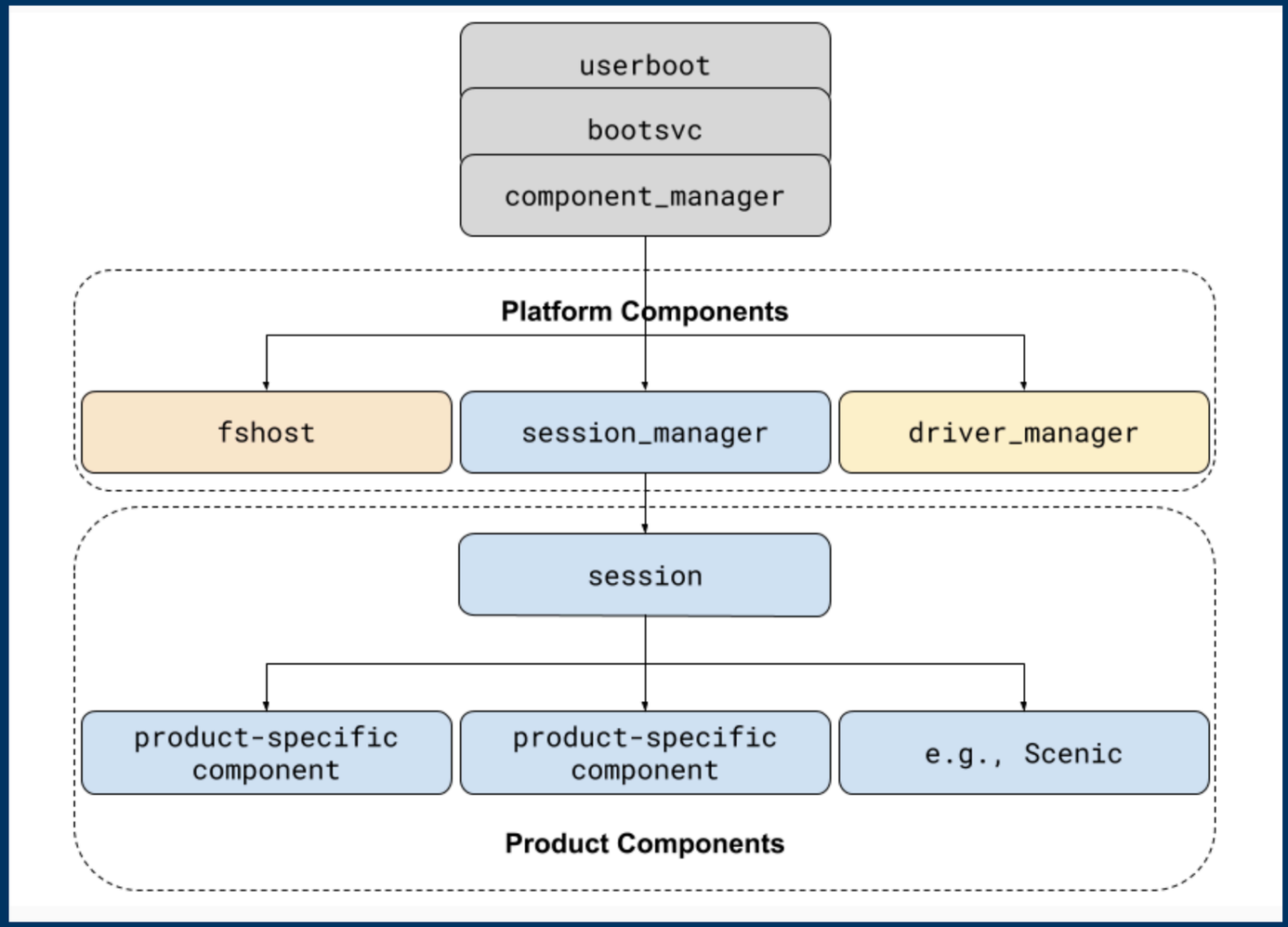
  Module：UI的基本单位

  Agent：provide services，No UI View

  Story：一组Module，定义交互的路径，管理lifecycle

- **Modular session**

Session Manager



https://fuchsia.dev/fuchsia-src/concepts/session/introduction?authuser=1

Session And Element

- **Session：组织，封装，形成产品的用户体验**

  第一个产品相关的component，只有一个根，sub-session形成树

  系统的各方面能力，服务的组织者，elements的管理者：

  提供通用能力，管理input，配置场景，管理附件等

- **Element：相当于其他平台的App，提供直接的用户功能和体验**

- Defines UI：
  - The mechanism by which it presents itself (graphically using Scenic, through audio, and so on).
  - The mechanism by which it receives user input (finger touches, gestures, keyboard, and so on).
  - The semantic meaning associated with its various annotations.
- Elements 可跨产品移植

UI Framework - flutter跨平台框架

- flutter跨平台框架
  - 跨平台：
    - mobile：安卓，ios，fuchsia，鸿蒙，Tizen
    - Desktop， not stable：macos， windows（win32， UWP），linux
    - 其他：toyota车机，设计环境：AdobeXD 导出flutter
  - 性能问题？从google的能力上讲应该能解决，其是native的本身比java在性能和内存上有先天优势，框架层的优化是很先进的
  - 反应式编程框架：是否更适合UI，解耦更好?
  - dart语言

UI Framework - flutter跨平台框架

- 性能：理论上flutter应该比普通Andriod应用性能好
  - Sublinear 的build和layout，单方向遍历（通过限制的到速度？）
  - Aggressive composition，大量的控件组合，基础组件精简稳定，组合灵活，优化解决数量膨胀带来的性能下降
  - 优于Android的View系统?
    - 至少在遍历和layout方面优于。
  - 后台运行，dart isolate，基于事件calback（仅此一种？是否可以“自由运行”？）

UI Framework - flutter跨平台框架

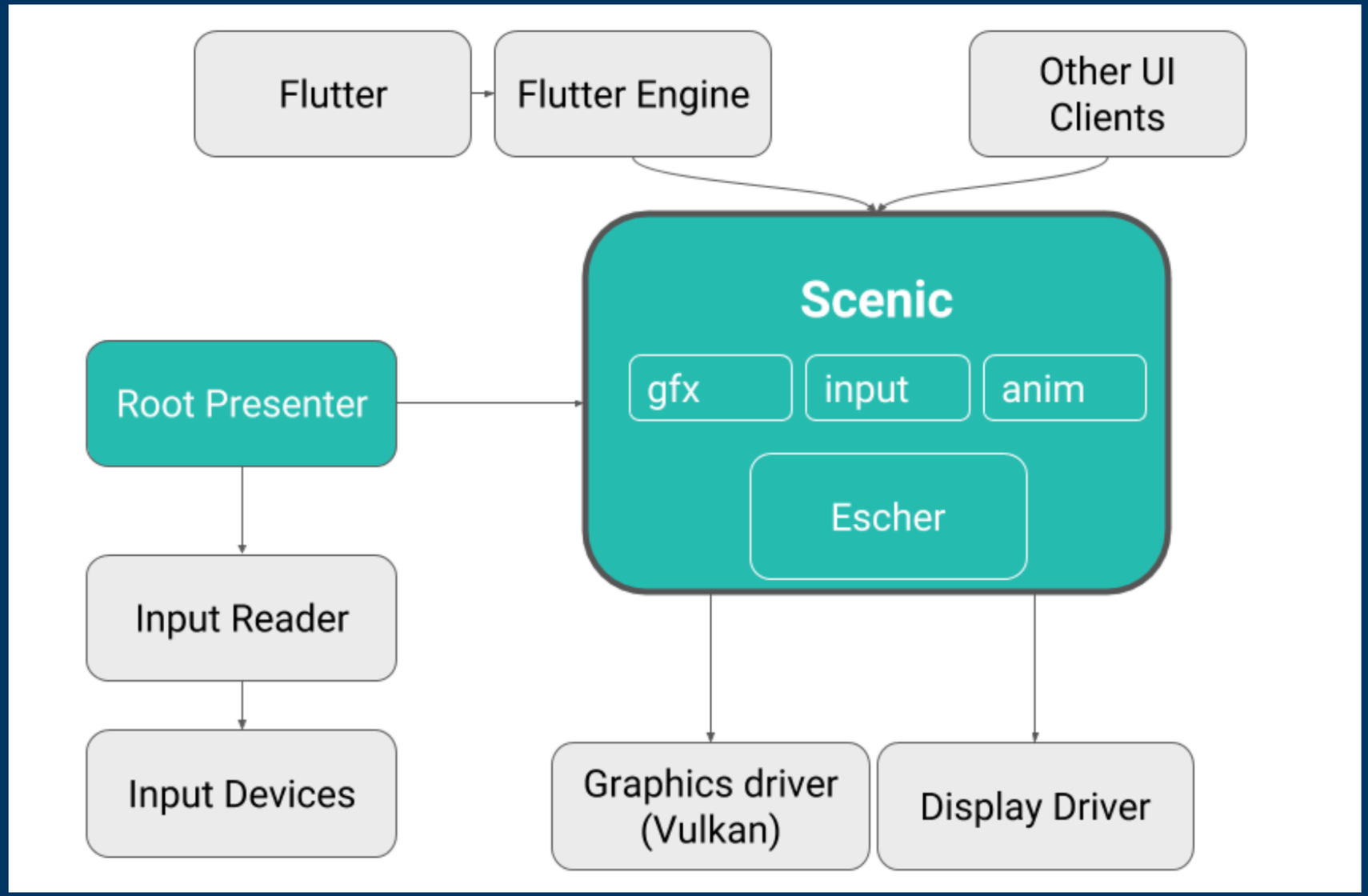
- API Ergonomics：API的人类工程学

  通过API设计减少错误，增加健壮性（同样多的失误，更多正确的结果）

  - Specializing APIs to match the developer’s mindset
  - Explicit arguments
  - Paving over pitfalls
  - Reporting error cases aggressively
  - Reactive paradigm
  - Interpolation

  https://flutter.dev/docs/resources/inside-flutter#api-ergonomics

Graphics


Flutter
Flutter Engine
Other UI Clients
Scenic
gfx
input
anim
Escher
Root Presenter
Input Reader
Input Devices
Graphics driver (Vulkan)
Display Driver

Graphics

Escher:

- 基于物理的渲染（Physically Based Rendering，PBR）是指使用基于物理原理和微平面理论建模的着色/光照模型，以及使用从现实中测量的表面参数来准确表示真实世界材质的渲染理念。https://zhuanlan.zhihu.com/p/53086060
- 内部的渲染引擎，未暴露到SDK

- Magma：Vulkan driver
- Scenic：System Compositor
  - better for 3D Scenes（light and shadow via Escher）and multi output target
    - https://fuchsia.dev/fuchsia-src/concepts/graphics

Graphics

Scenic (SurfaceFlinger & WindowManager?)

- Manage Sessions with clients
- manage resources added via Sessions from clients
- routes touch and mouse input from RootPresenter to UI Clients
- root presenter： provide system ui and route inputs to scenic
- similar to Android：
  - Units and Metrics
  - Views, Bounds, and Clipping
  - Scenic Views, view tree, and ViewRefs
  - Scenic Views, view focus, and focus chain

Sysmem — Ion

shared memory for use by multiple applications and hardware blocks.

数据存储变革/分布式文件存储

ledger：账本

Deleted！！

分布式数据存储的需求是合理的，为何删除?

—不清楚

fuchsia 支持的语言

Fuchsia的大多数UI都是用DART（一种被设计成让JavaScript和Java开发人员感到熟悉的语言）通过Frutter框架实现的。还支持Go语言。谷歌还通过引入了swift支持来引入苹果的开发者群体。不过，最重要的是，大多数这些语言都支持互操作。通过FIDL（Fuchisa接口定义语言）协议，DART UI代码可以直接与Go后端或任何其他语言代码互相调用。这使开发人员更具表现力，并为手头的工作使用最好的语言开发。

—吸引开发者是最重要的事