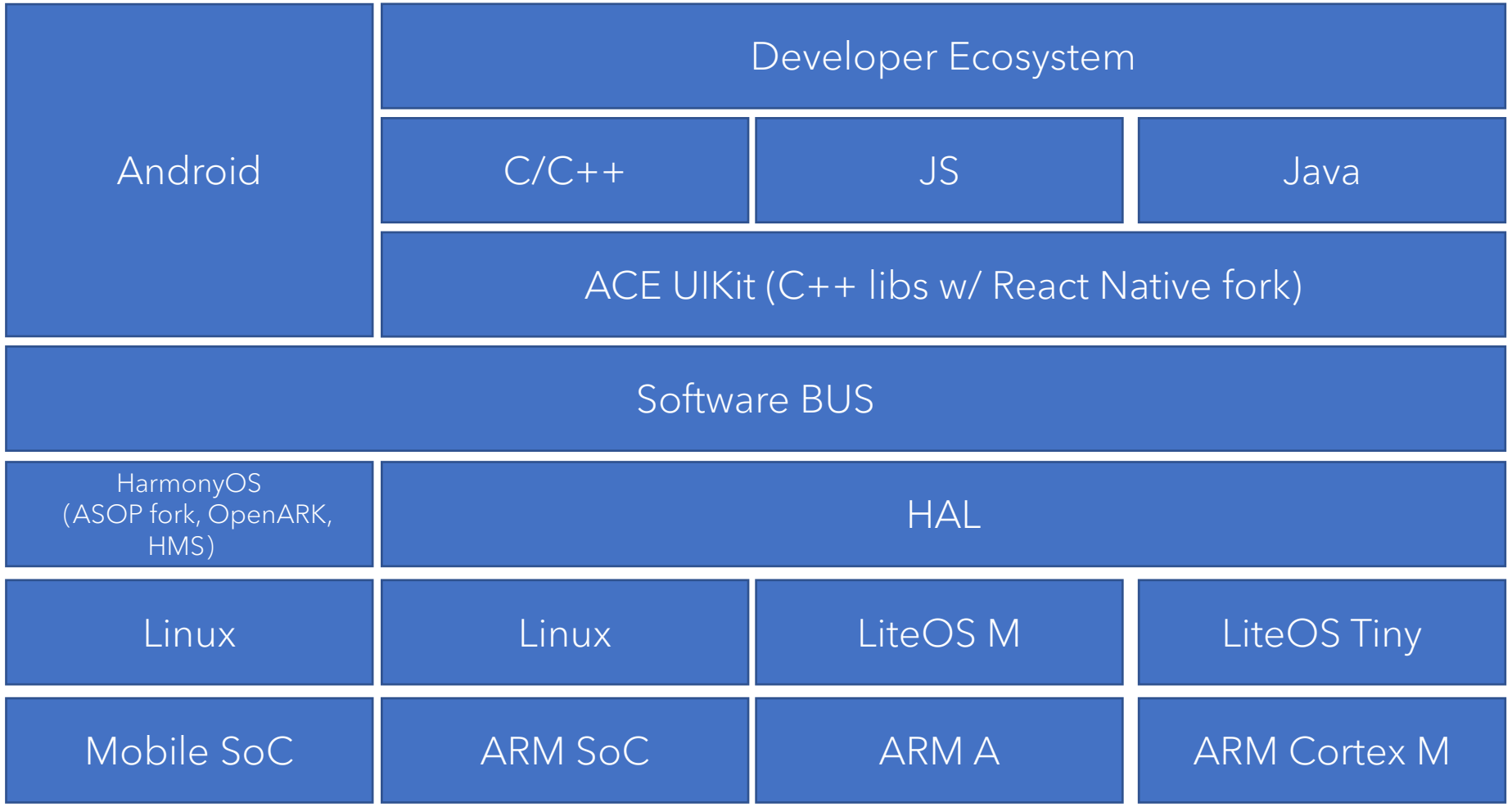
Next Gen App Architecture Trends

- Disaggregated architecture for the cloud and device, capabilities served by the DSA/chiplet and orchestrated by the hardware and software “BUS”
- Industry APIs exposed by the software/hardware capsules
- The traditional OS functionalities are offloaded to the capsules and the OS is becoming the management and glue layer
- The application framework is orchestrating the underline capsules with the industry APIs through the traditional APIs or WASM FFI integration

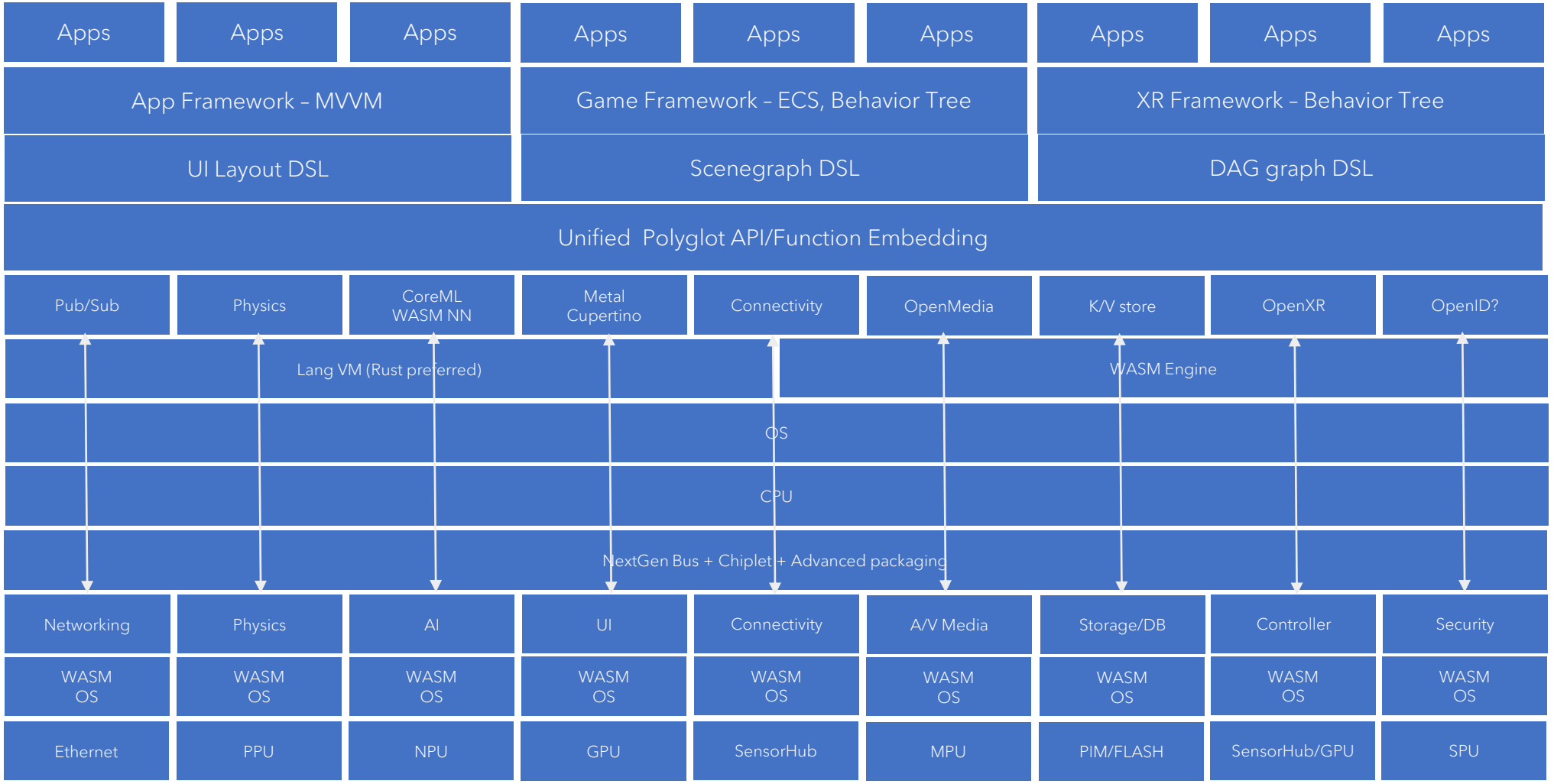
OpenHarmony



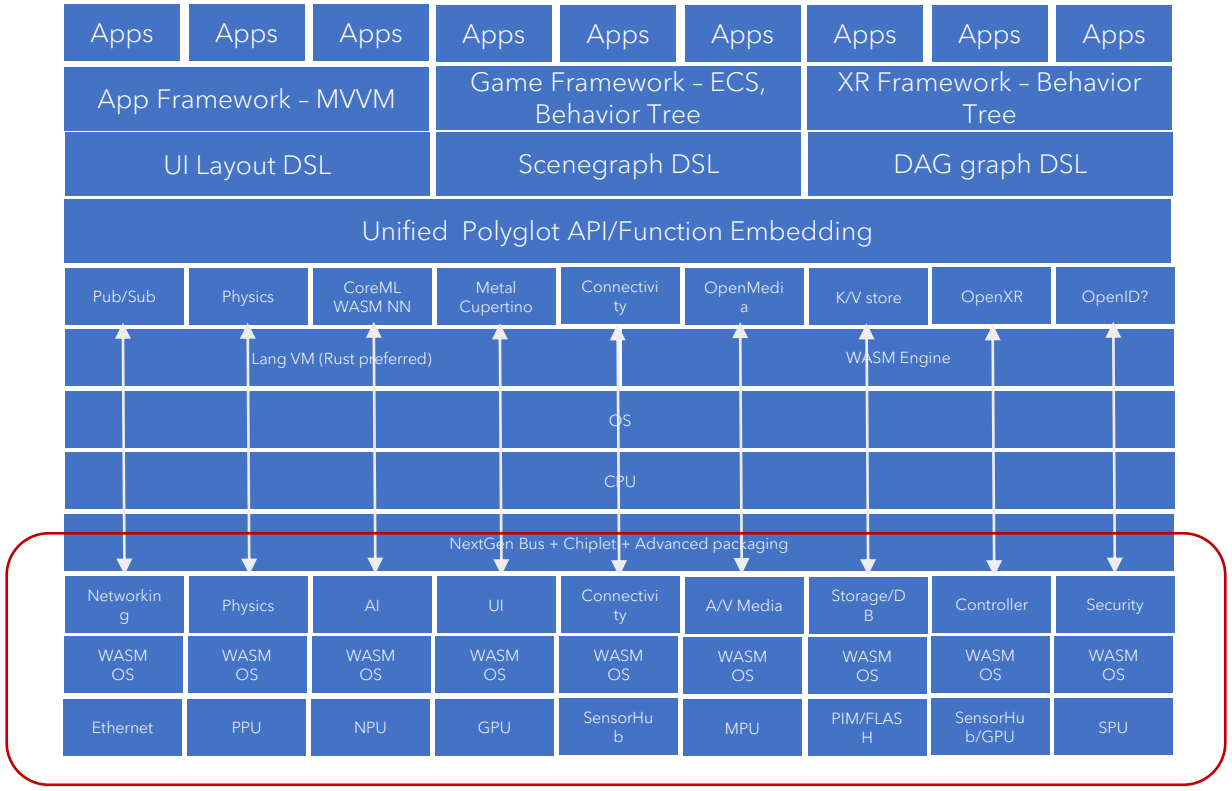
Key WASM technology planning areas

- WASM runtime/engine
  - Browser WASM as is – chrome, safari
  - Wasmtime – JIT, Rust, Cranelift
  - WasmEdge – AOT, C++, LLVM
  - WAMR – Interpreter, C/C++, LLVM
- WASM capabilities + Rust shim layer/Rust rewrite
  - NPU runtime, WASI NN
  - GPU, GLSL shader language, SPIR-V intermedia
  - OpenXR, WebXR
  - Sycl IoT, Khronos group?
  - Sensor Hub
- New WASM OS
  - Rust OS? Theseus OS+WASM
  - Mesalock, Linux userspace Rust rewrite
- WASM orchestration framework, WASM
  - Rust DSL for WASM orechestration – AssemblyScrpt, Rhai, Nushell
  - WASM Edge – zenoh, zenoh-flow, Anna storage, deployment, orchestration, ops, open telemetry,

Ultimate architecture vision

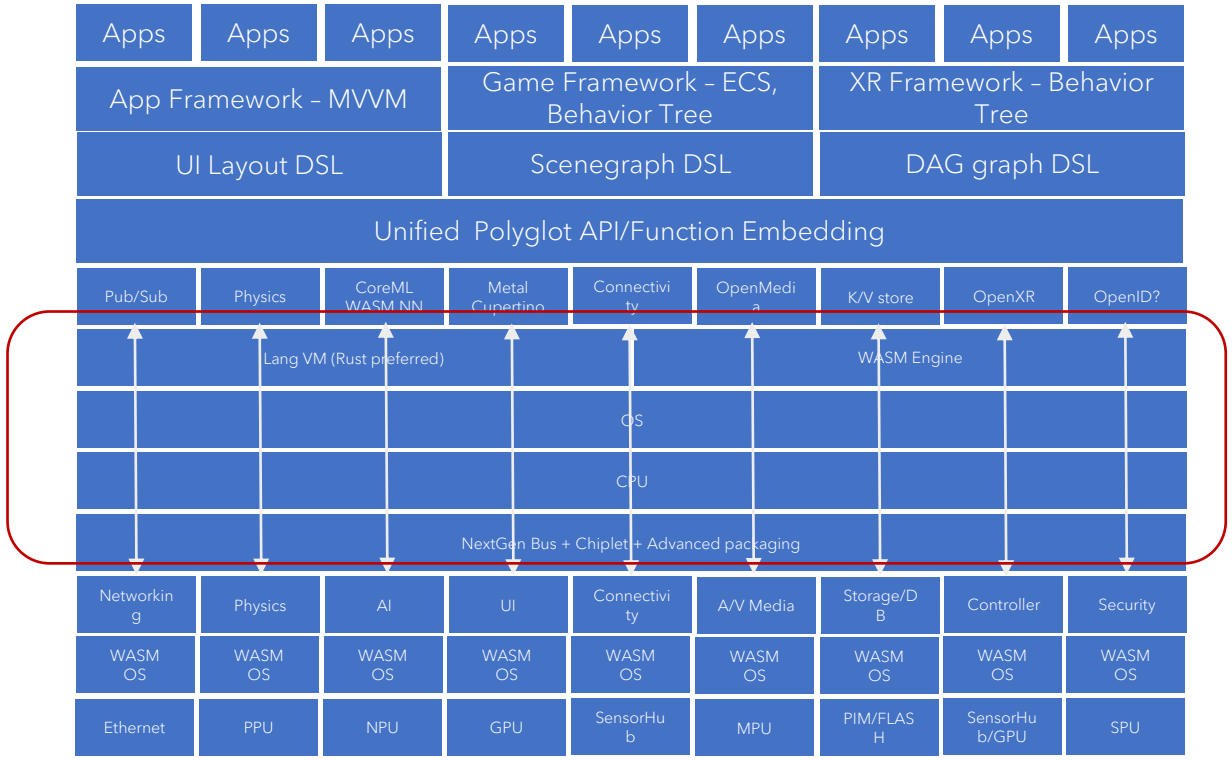


集成能力容器



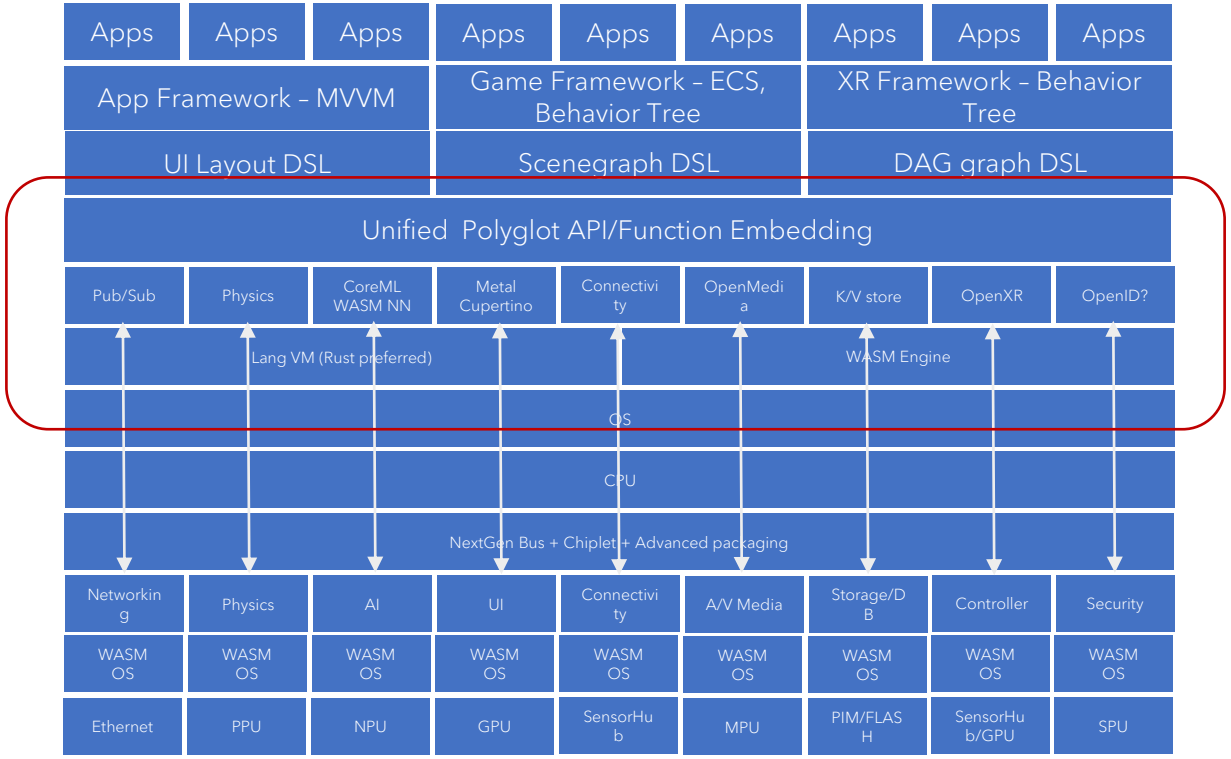
- Disaggrated 架构的优势
  - 软件硬件协同设计，优化性能和功耗（AWS nitro 卸载）
  - 更好的供应链管理，拜托CPU/OS生态控制（手机sensorhub，智能手表）
  - 符合能力发展趋势，AI，多模，图像为代表的异构计算模式
  - 符合后摩尔时代半导体技术的趋势，异构集成，优化技术效能
- Capability based OS趋势
  - Rust为代表的Tock OS的capsul，theseus OS的cell，是OS的micro kernel趋势的延续
  - WASM+Rust是未来能力化OS的基础技术
- 灵活的能力访问
  - 可以是API调用
  - 也可以在能力硬件中动态部署WASM代码

CPU+OS 瘦身，可信计算



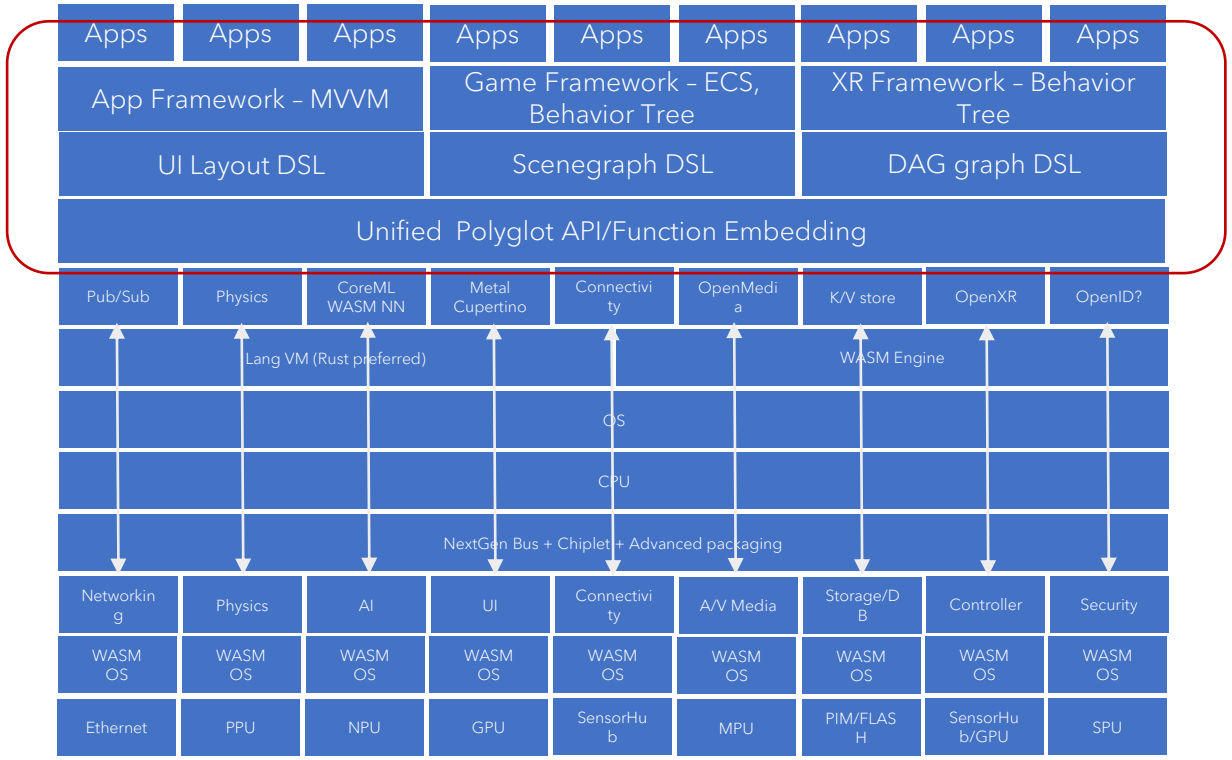
- 传统CPU+OS职能转换
  - 提供用户和能力的连接
  - 提供逻辑和数据胶水能力
  - 提供Rust和WASM的运行引擎
- 基于Rust和WASM的可信OS
  - 能力微内核（capsule, cell）
  - 可信访问
  - 单地址空间
  - Ring0 空间运行App

基于工业标准的能力开放



- 基础能力开放
  - 拉通工业标准能力API和底层能力
  - 北向提供多语言、多样化的集成模式
- 桥接API
  - 类Rest的RPC调用
- WASM embedding
  - 归一化语言的优化
  - 无数据串行化代价
  - 无数据类型转换代价
  - 用户代码和能力函数通过共享内存集成

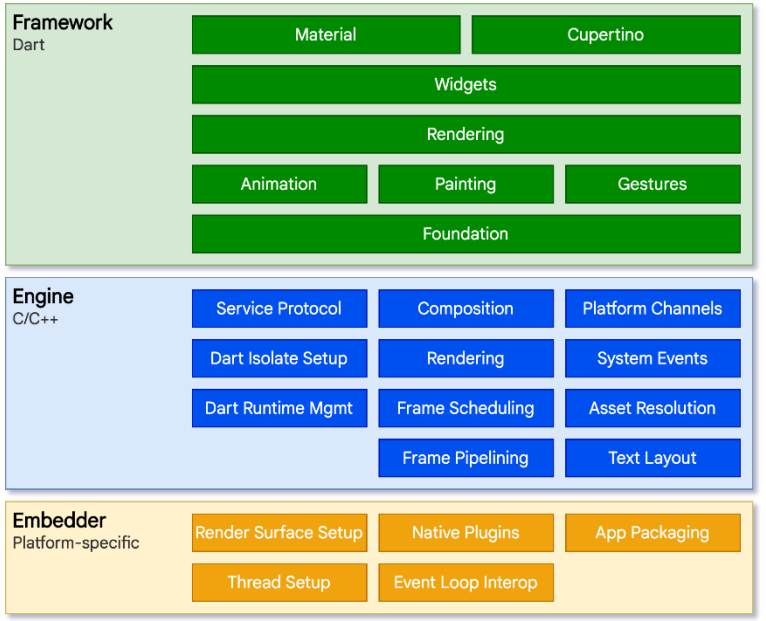
基于基础能力的应用框架



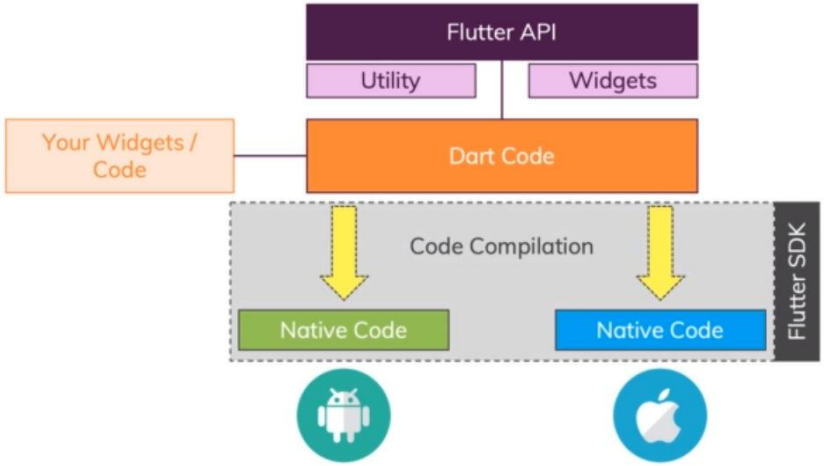
- 基于基础能力支持不同应用场景，或者共存
  - 应用框架提供设计模式和用户应用开发环境
  - 应用开发环境需要用户DSL编程语言
- 前端框架微服务化
  - 用户代码对基础能力做orchestration
  - 能力基于WASM封装

Flutter 战略

- 通过跨平台的UIKit 获取开发者，可以寄生在既有平台，也可以和Fuschia会师构成完整平台
  - 一套代码，跨平台部署
    - iOS, Android, Web, Fuschia
  - 原生App性能
    - Skia图形引擎和自绘原生支持iOS和Andriod的组件库，消除了底层使用webview等异构引擎的开销
    - DART用于引擎和用户侧语言，相比其他hybrid框架，降低数据bridge带来的开销
  - DART语言和平台适配层实现和底层平台的高性能集成
    - DART应用和平台适配层编译为支持iOS和Android的C/C++ ABI，Flutter作为iOS/Android library集成
- DART语言双模设计
  - 在开放阶段支持JIT模式，hot reloading，加快开放迭代速度
  - 在产品发布阶段支持AOT模式，提供性能和安全

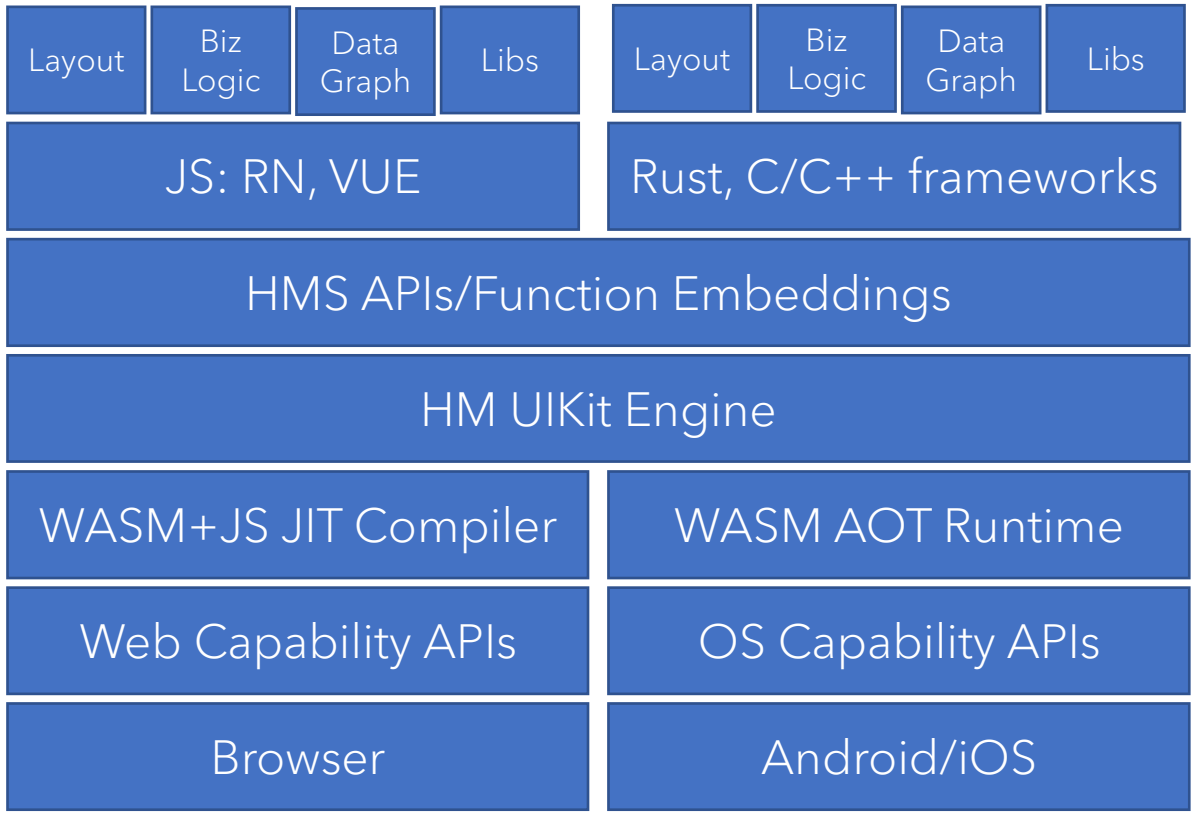


How is Flutter/ Dart "transformed" to a Native App?

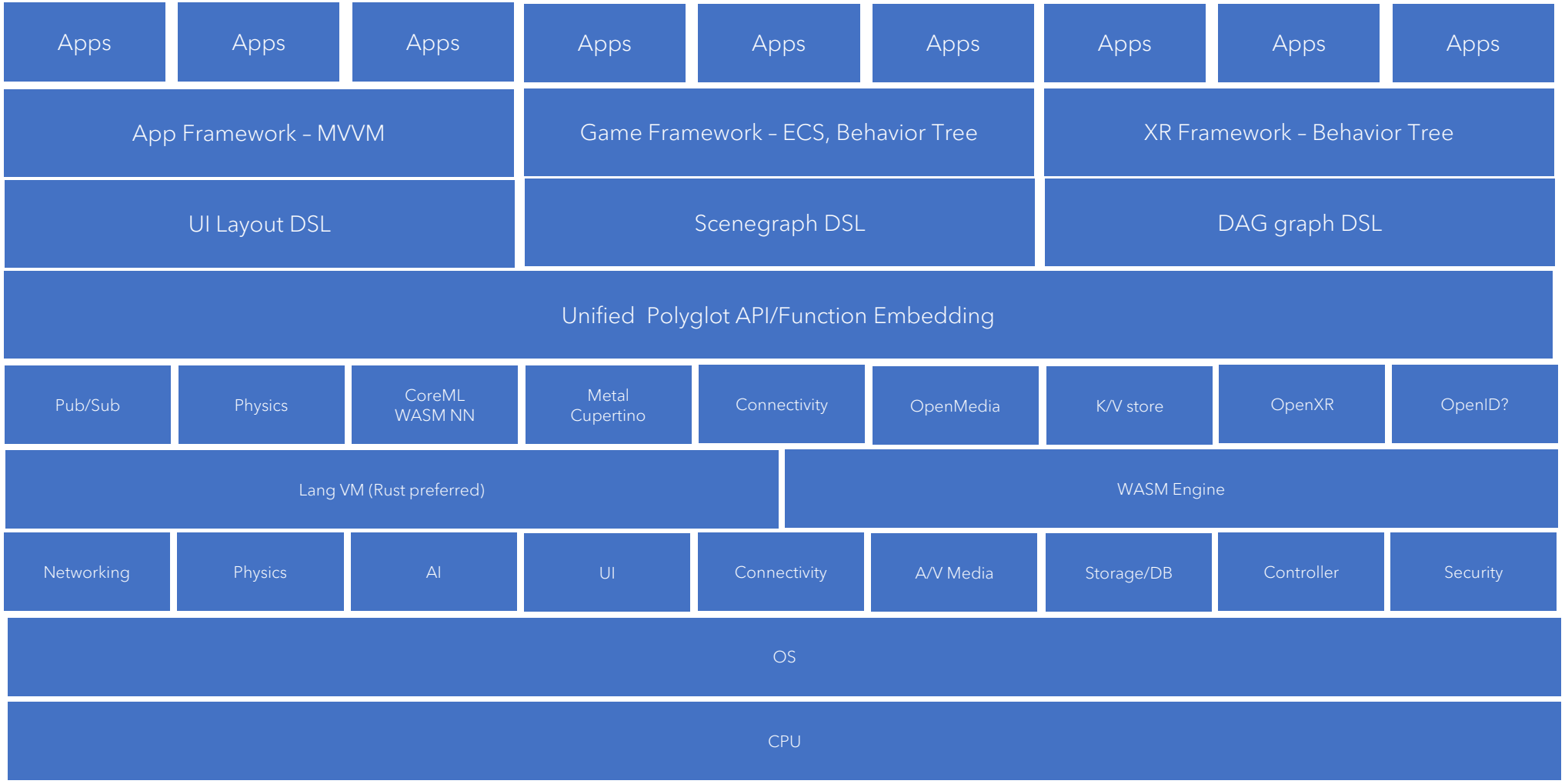


HM 战略，通过WASM+浏览器实现跨平台

- 构建HM跨平台UIKit层
  - C/C++，Rust 的UI library =》WASM
  - Skia 已经可编译为WASM Canvaskit
- WASM 编程语言
  - C/C++, Rust平滑编译为WASM，WASM UIKit编译为libraries实现和flutter类似的和iOS/Android集成
  - Interface type提供多语言集成，用户态语言和UIKit公共WASM集成
  - WASM sandbox实现安全集成
- 基于WASM的平台适配层
  - WASI 提供对OS的抽象
  - WASM AOT =》平台Library
- WASM vs DART
  - WASM支持多语言集成
  - WASM没有开发者的learning curve
  - 提供和DART一样的JIT和AOT
  - WASM AOT 性能存在安全带来的性能代价



通过WASM以软能力胶囊寄生在浏览器生态



App新语言的趋势

- 趋势判断
  - Low coding, no coding是前端发展的趋势，设计和实现一体的IDE（figma, webflow），声明式编程的普遍采纳
  - Micro-frontend，大厂内部实现了App能力组件化，云端和前端主要工作是粘贴和组合，相应的SaaS工具（https://bit.dev/，https://www.framer.com/）
  - 针对对domain的DSL配合compiler优化domain问题，例如：JS之上发展出来的DSL，伯克利hydro-flow项目基于Rust构建的多个DSL
- 新语言的战略
  - 底座语言做能力：系统语言，性能，安全，跨平台，多语言集成，值得长期投资
  - 胶水语言粘用户：拉通设计和工程，low coding，no coding，组件化，服务化，胶水DSL语言
  - 投资在基础底座语言上构建DSL的能力，通过开放社区快速迭代（微软的TS战略）
  - 透明更换底座：把JS的DSL变为Rust DSL，提升性能和安全性

App前端语言的机会

- 前端语言的多样场景，导致已经在JS上长出不同的DSL
  - Layout 设计，声明式编程主导
    - – React JSX，RAX, 基于JS的DSL，通过Babel transcompile到JS
    - SwiftUI，Jetpack
  - 业务逻辑
    - TypeScript，强类型，通过Babel transcompile到JS
    - Dart，强类型，通过dart2js transcompile到JS
  - 计算类型
    - C/C++，Rust，通过asm.js compile 到JS
  - 数据ORM
    - GraphQL，拉通前后台的数据图描述

- 基于Rust的DSL对标前端JS DSL
  - 排版布局
    - Makepad 的 shader DSL，兼容CSS，借鉴sharder DSL，充分利用GPU（https://github.com/makepad/makepad_docs/blob/main/Makepad%20Whitepaper%202020.pdf）
  - 业务逻辑
    - TS, JS compiler in Rust, https://github.com/swc-project/swc
    - 通过WASM支持多种语言编写业务逻辑
  - 计算类型
    - WASM的能力库
  - 数据ORM
    - 支持GraphQL in Rust

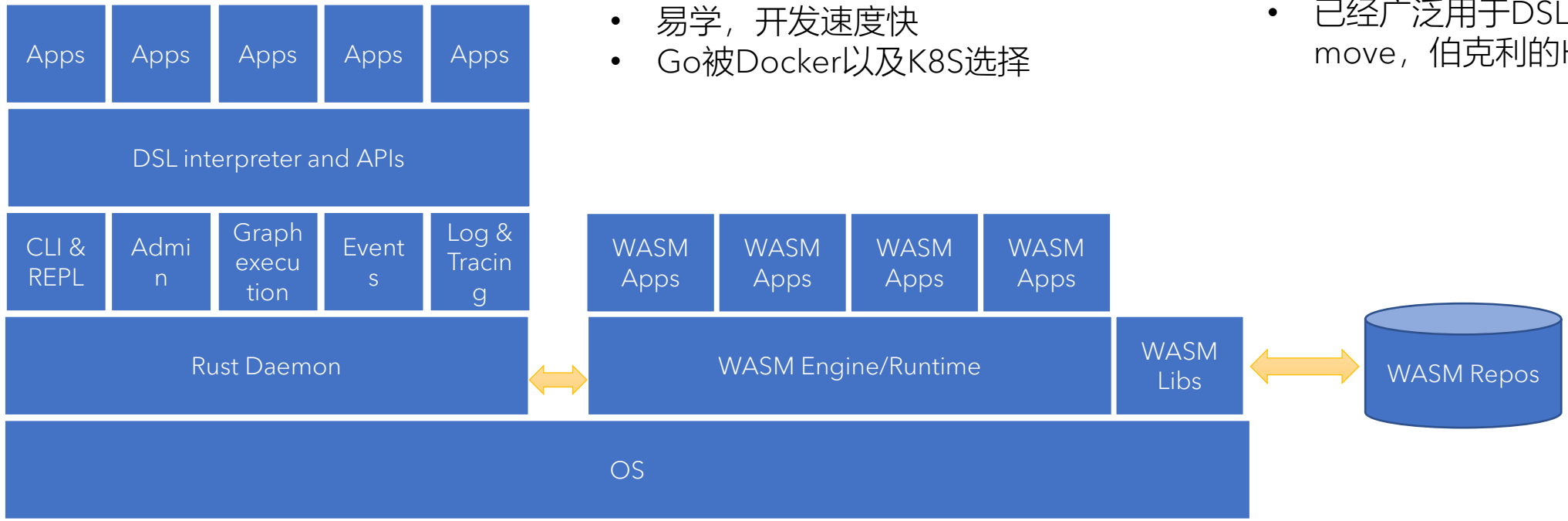
WASM orchestration DSL 的机会

WASM生态需要新的语言

- 低开销部署，否则破坏了WASM的优点
- 系统语言，可以和OS和WASM紧密整合
- 系统生态丰富
- 易学，开发速度快
- Go被Docker以及K8S选择

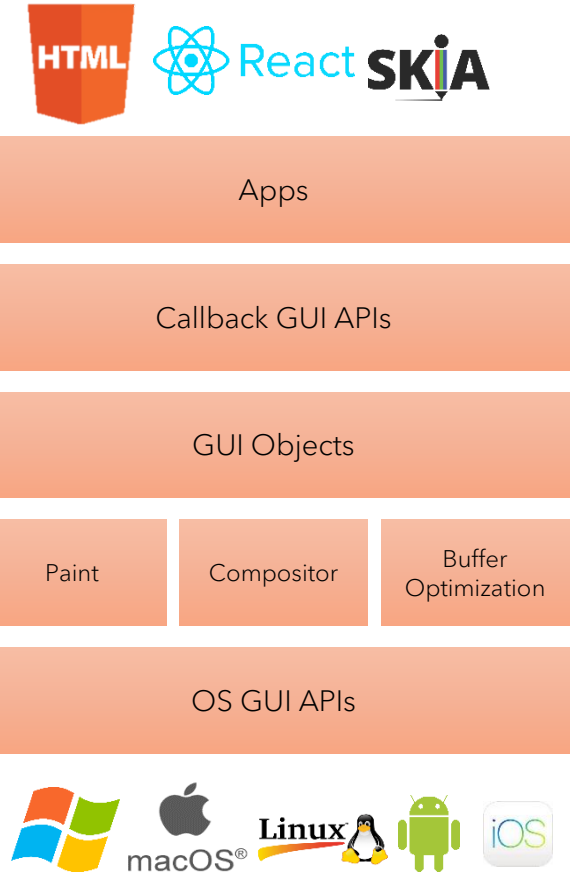
Rust 具备支持DSL设计的特点

- Macro 展开
- 泛型 polymorphism
- 强大的parser
- 也可以用于做编译器
- 已经广泛用于DSL设计，FB的move，伯克利的Hydroflow

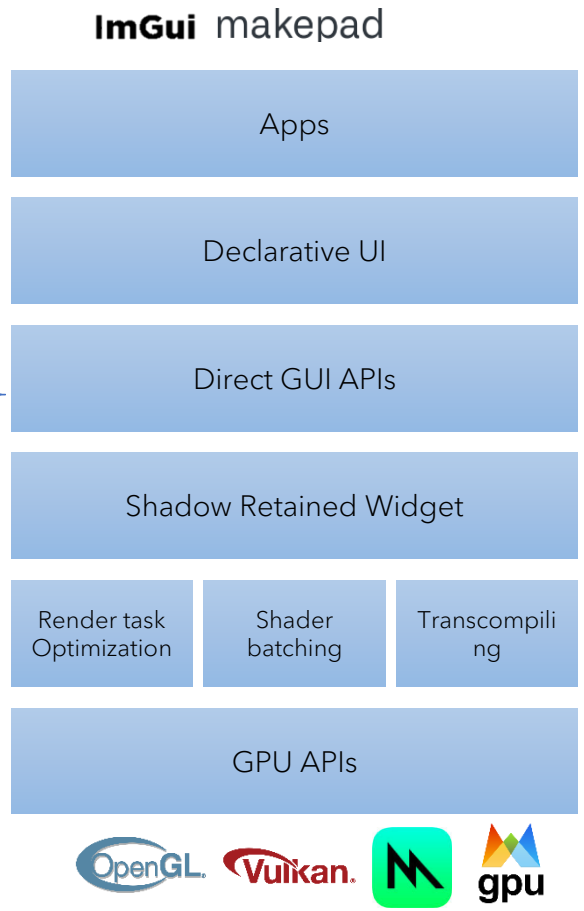


GUI 发展趋势

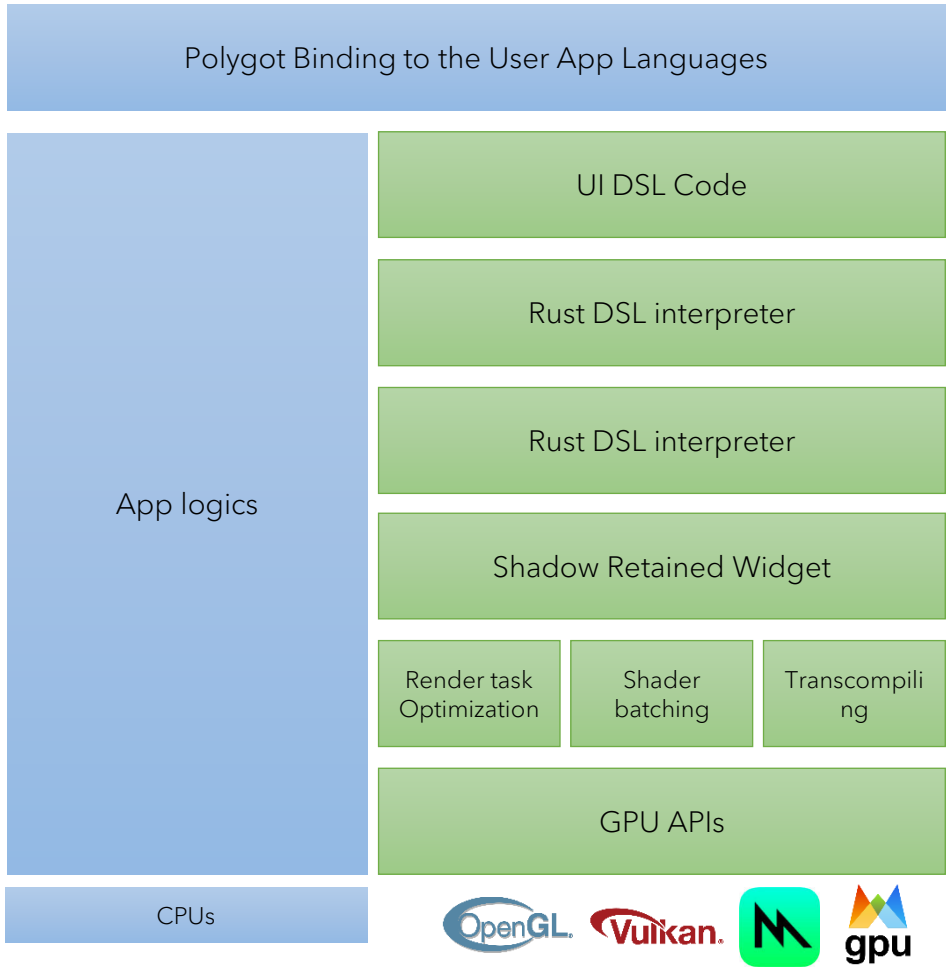
- App和GUI的状态需要同步，用户程序复杂
- 只能优化2D显示，3D和动画需要单独stack
- CPU为核心的渲染机制，没有充分利用GPU的并行计算模式



- 声明式编程，拉通设计和实现流程，简化UI设计
- GUI 无状态，App直接调用GPU APIs
- 同时支持2D，动画和3D
- 利用现代GPU的并行化计算能力，60fps



基于WASM的跨平台UIKit



- UI DSL 作为intrinsic嵌入用户语言，或在更高层面的声明式语言使用
- DSL 代码被编译为WASM，被Rust编写的解释器执行
- DSL代码WASM作为Library和app代码链接
- 基于Rust实现immediate mode的GPU GUI

游戏/XR 业务前端语言

- 游戏场景渲染
  - Scengraph
- 游戏策略
  - Behavior tree
- Scripting and plugins
  - Unity 支持C#, JS的pluging，已经支持WASM
  - Ulreal 支持C++，可以支持WASM

WASM in the Next Gen App

- WASM basically makes C/C++ secure and modular
  - Can be used to package the C/C++ code assets
  - Easy for the computing intensive, need WASI for interfacing with the OS/HW
- WASM is also natively integrated with Rust
  - Minimum performance loss, secure by static checking and runtime protection
  - Could have optimized integration, i.e. compiling process optimization
- Benefits
  - Universal byte code/package running everywhere, e.g. XPUs
  - Secure sandbox, allowing running the 3rd party code everywhere, libraries for the foreign languages, inside OS kernels, inside applications
  - Refactor the OS, eliminated the kernel-user space separation
  - Capability based security model in WASI

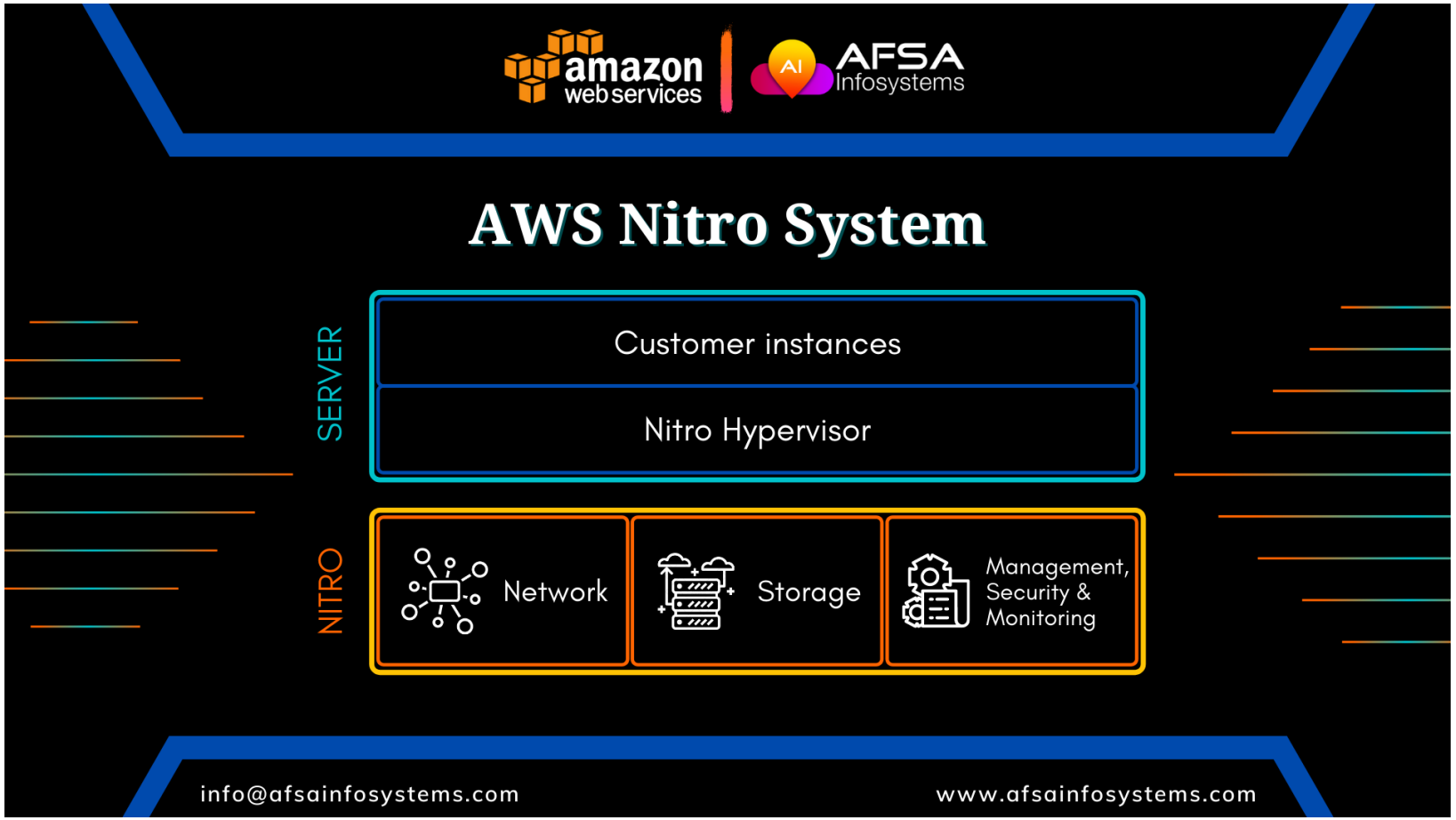
Webassembly design principles

- Integrated many language runtime/VM ideas from the past
  - JVM – WASI
  - LLVM, JavaScript, GraalVM/Truffle – language design
- Universal byte code/packaging format
- Secure sandbox
  - Control flow integrity
  - Single address space, boundary check
- Unified language integration mechanism
  - Interface type standard
  - As a shared library for the foreign languages, WASM blobs integration
- Abstraction of the OS
  - Almost OS agnostic, from Linux to RTOS or simple runtime
  - Low overhead
- Flexible linking mechanism, micro services, micro modules
- Benefit to the capability-based system

投资重点

- 充分利用Rust和WASM的产业机会，作为新的基于服务能力的计算架构的底座
  - 人才的获取，标准的制定，生态的影响力
- 重点构建服务能力底座
  - 短期借助既有的计算生态，软能力胶囊，寄生于既有生态，长期软件+芯片实现能力超越
  - 参与能力标准的制定和引领
- 利用DSL语言和编译技术获取前端生态
  - 顺应前端Low coding, no coding，微前端等趋势，重点投资能简化用户开发，降低开发成本的DSL和工具
  - 在新兴业务领域抢先布局生态，例如：开放游戏引擎，XR 引擎
  - 充分利用WASM带来的浏览器第二计算平台，构建寄生生态

AWS Nitro卸载CPU负载



Capsule based Trustworthy OS