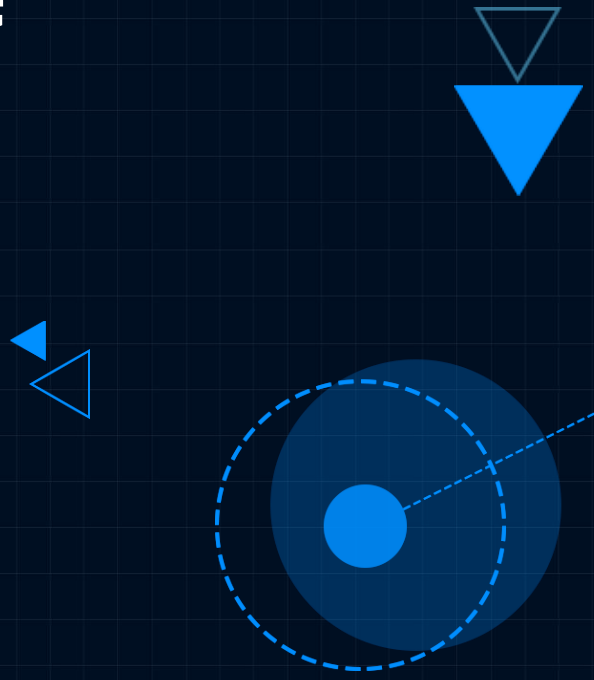
seL4 for Secure Edge Cloud Infrastructure

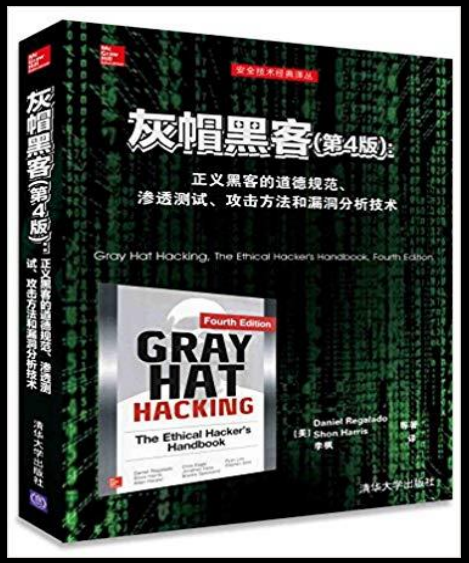
Feng Li(李枫)

Indie developer



Who Am I

- The main translator of the book «Gray Hat Hacking The Ethical Hacker's Handbook, Fourth Edition» (ISBN：9787302428671) & «Linux Hardening in Hostile Networks, First Edition» (ISBN: 9787115544384)



- Pure software developmement for ~15 years
- Actively participate in various activities of the open source community
- https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/Conf
  https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/LTS
- Recently, focus on infrastructure of Cloud/Edge Computing, AI, Virtualization, Program Runtimes, Network, 5G, RISC-V, EDA…

Agenda

I. Tech Stack

- Edge Computing
- Microkernel
- Unikernel
- Wasm
- Rust
- Userland
- Virtualization & Security on ARM
- Testbed

II. Virtualization on seL4

- Overview

III. seL4 on ARM

- Overview
- RPi4

IV. Unikernel on seL4

- Rumprun Unikernel
- Tittle-tattle
- Considering a Unikernel more suitable for seL4

V. Wasm on seL4

- WasmEdge
- Our Approach

VI. seL4 in Automotive

- SOAFEE

VII. Architecture and design

- WasmEdge
- Our Approach

VIII. Wrap-up

I. Tech Stack

1) Edge Cloud

1.1 Edge Computing

- https://en.wikipedia.org/wiki/Edge_computing

  a distributed computing paradigm which brings computation and data storage closer to the location where it is needed, to improve response times and save bandwidth.…

MEC

- https://en.wikipedia.org/wiki/Multi-access_edge_computing

Multi-access edge computing (**MEC**), formerly **mobile edge computing**, is an ETSI-defined[1] network architecture concept that enables cloud computing capabilities and an IT service environment at the edge of the cellular network[2][3] and, more in general at the edge of any network. The basic idea behind MEC is that by running applications and performing related processing tasks closer to the cellular customer, network congestion is reduced and applications perform better. MEC technology is designed to be implemented at the cellular base stations or other edge nodes, and enables flexible and rapid deployment of new applications and services for customers. Combining elements of information technology and telecommunications networking, MEC also allows cellular operators to open their radio access network (RAN) to authorized third parties, such as application developers and content providers.

Technical standards for MEC are being developed by the European Telecommunications Standards Institute, which has produced a technical white paper about the concept.[4]

Distributed computing in the RAN [edit]

MEC provides a distributed computing environment for application and service hosting. It also has the ability to store and process content close to cellular subscribers, for faster response time.[5] Applications can also be exposed to real-time radio access network (RAN) information.[6]

The key element is the MEC application server, which is integrated at the RAN element. This server provides computing resources, storage capacity, connectivity and access to RAN information. It supports a multitenancy run-time and hosting environment for applications. The virtual appliance applications are delivered as packaged operating system virtual machine (VM) images or containers incorporating operating systems and applications. The platform also provides a set of middleware application and infrastructure services. Application software can be provided from equipment vendors, service providers and third-parties.

Key points:

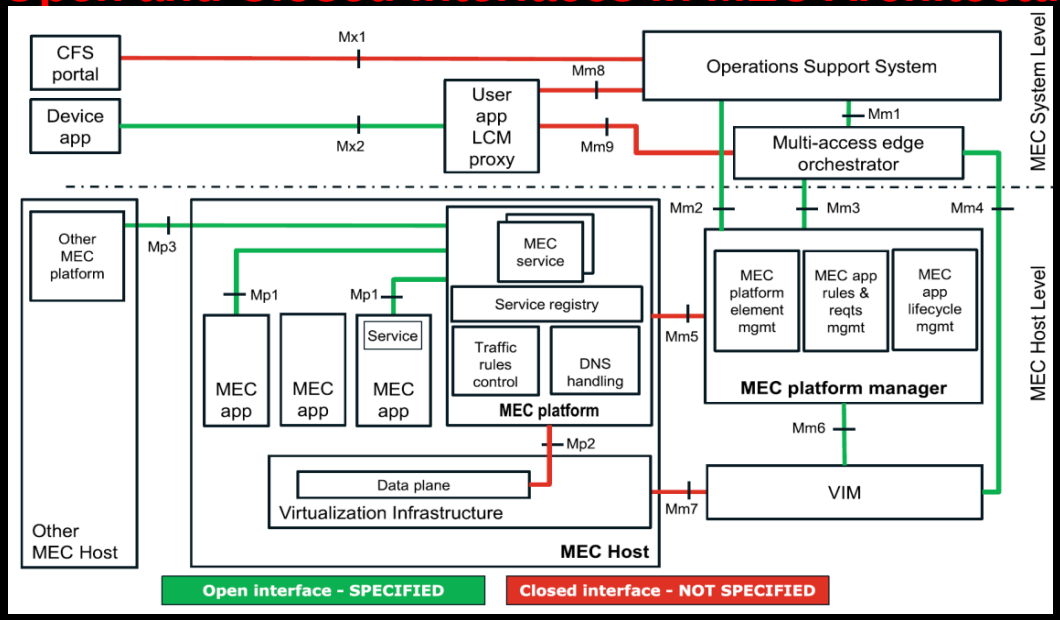
1) distributed computing
2) security
3) virtualization

- https://www.etsi.org/technologies/multi-access-edge-computing

ETSI

- **Use Cases**
  - video analytics
  - location services
  - Internet-of-Things (IoT)
  - augmented reality
  - optimized local content distribution and
  - data caching

  ...
- **Open and Closed interfaces in MEC Architecture**

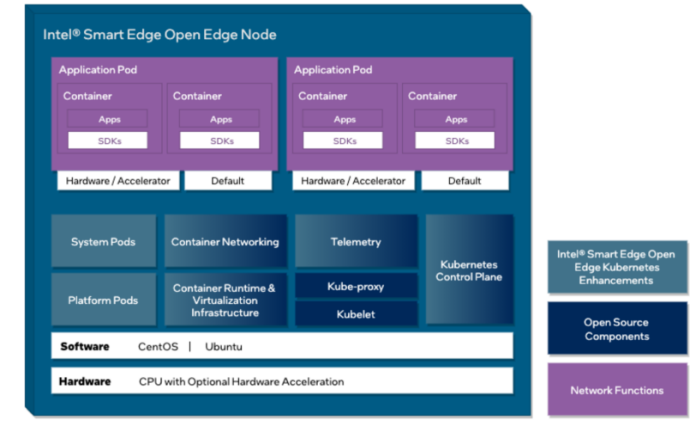


Source: "Multi-access Edge Computing: Software Development at the Network Edge", Dario Sabella, Springer 2021.

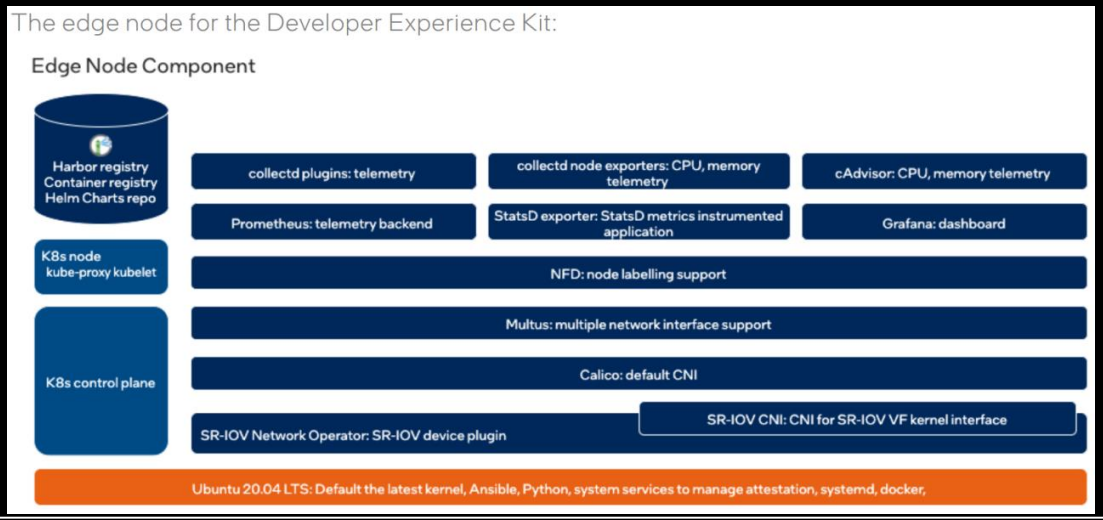
- https://www.etsi.org/technologies/multi-access-edge-computing/mec-poc

OpenNESS

- https://www.openness.org/
  https://smart-edge-open.github.io/docs/product-overview/

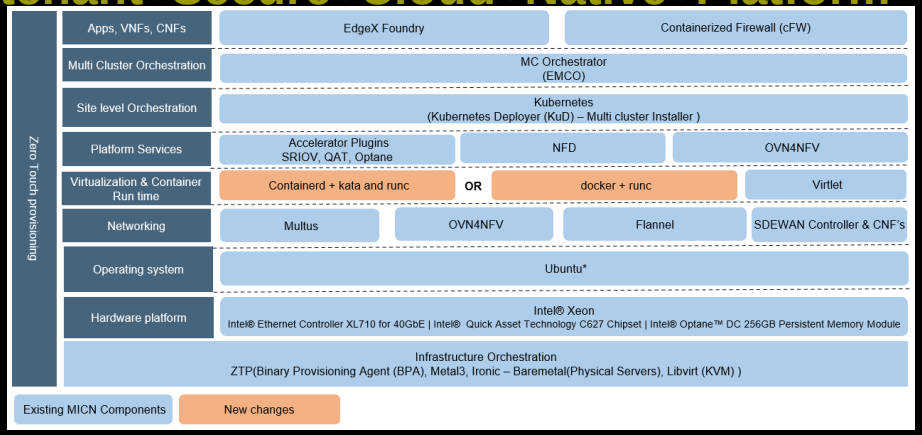


The Intel® Smart Edge Open node architecture is specialized for each experience kit, to enable developers to create solutions for specific use cases at a given edge location.

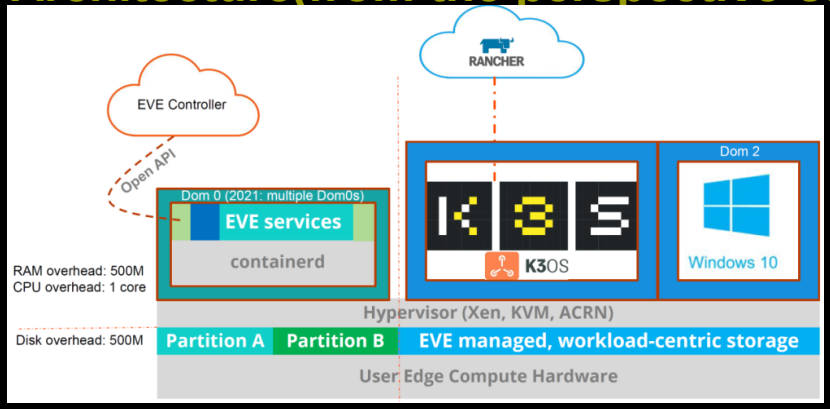


LF Edge

- https://www.lfedge.org/
  IoT + Telecom + Cloud + Enterprise + Industrial
- https://wiki.akraino.org/display/AK/Architecture+Document+-+Multi-tenant+Secure+Cloud+Native+Platform



- Architecture(from the perspective of Virtualization)



Source: https://wiki.lfedge.org/display/EVE/Slide+presentations

2) Microkernel

- https://en.wikipedia.org/wiki/Microkernel

In computer science, a **microkernel** (often abbreviated as **μ-kernel**) is the near-minimum amount of software that can provide the mechanisms needed to implement an operating system (OS). These mechanisms include low-level address space management, thread management, and inter-process communication (IPC).

If the hardware provides multiple rings or CPU modes, the microkernel may be the only software executing at the most privileged level, which is generally referred to as supervisor or kernel mode. Traditional operating system functions, such as device drivers, protocol stacks and file systems, are typically removed from the microkernel itself and are instead run in user space.[1]

In terms of the source code size, microkernels are often smaller than monolithic kernels. The MINIX 3 microkernel, for example, has only approximately 12,000 lines of code.[2]

- **Difference Between Microkernel and Monolithic Kernel**

Parameters	Monolithic kernel	MicroKernel
Basic	It is a large process running in a single address space	It can be broken down into separate processes called servers.
Code	In order to write a monolithic kernel, less code is required.	In order to write a microkernel, more code is required
Security	If a service crashes, the whole system collapses in a monolithic kernel.	If a service crashes, it never affects the working of a microkernel.
Communication	It is a single static binary file	Servers communicate through IPC.
Example	Linux, BSDs, Microsoft Windows (95,98, Me), Solaris, OS-9, AIX, DOS, XTS-400, etc.	L4Linux, QNX, SymbianK42, Mac OS X, Integrity, etc.

Source: https://www.guru99.com/microkernel-in-operating-systems.html

Security

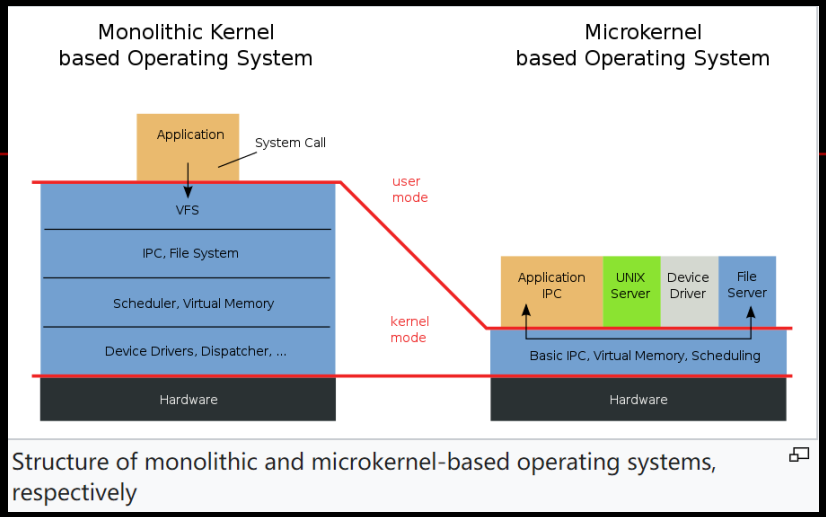
- The security benefits of microkernels have been frequently discussed.[29][30] In the context of security the minimality principle of microkernels is, some have argued, a direct consequence of the principle of least privilege, according to which all code should have only the privileges needed to provide required functionality. Minimality requires that a system's trusted computing base (TCB) should be kept minimal. As the kernel (the code that executes in the privileged mode of the hardware) has unvetted access to any data and can thus violate its integrity or confidentiality, the kernel is always part of the TCB. Minimizing it is natural in a security-driven design.

  Consequently, microkernel designs have been used for systems designed for high-security applications, including KeyKOS, EROS and military systems. In fact common criteria (CC) at the highest assurance level (Evaluation Assurance Level (EAL) 7) has an explicit requirement that the target of evaluation be "simple", an acknowledgment of the practical impossibility of establishing true trustworthiness for a complex system. Again, the term "simple" is misleading and ill-defined. At least the Department of Defense Trusted Computer System Evaluation Criteria introduced somewhat more precise verbiage at the B3/A1 classes:

  > "The TCB shall [implement] complete, conceptually simple protection mechanisms with precisely defined semantics. Significant system engineering shall be directed toward minimizing the complexity of the TCB, as well as excluding from the TCB those modules that are not protection-critical."
  >
  > — Department of Defense Trusted Computer System Evaluation Criteria

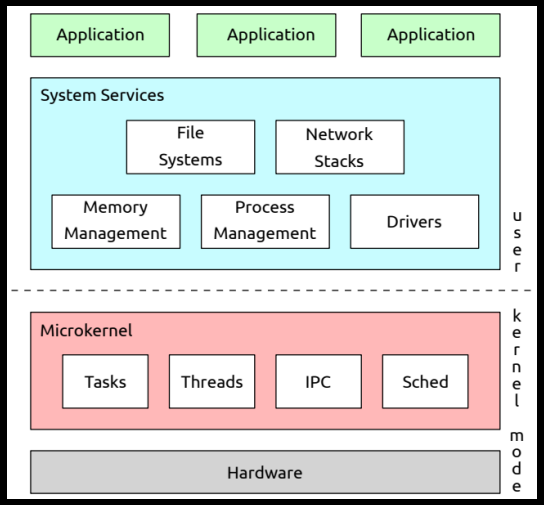
  In 2018, a paper presented at the Asia-Pacific Systems Conference claimed that microkernels were demonstrably safer than monolithic kernels by investigating all published critical CVEs for the Linux kernel at the time. The study concluded that 40% of the issues could not occur at all in a formally verified microkernel, and only 4% of the issues would remain entirely unmitigated in such a system.[31]

Architecture & Design



Structure of monolithic and microkernel-based operating systems, respectively

System Design



Source: https://os.inf.tu-dresden.de/Studium/MkK//SS2021/01_intro.pdf

2.1 L4

Overview

- https://en.wikipedia.org/wiki/L4_microkernel_family

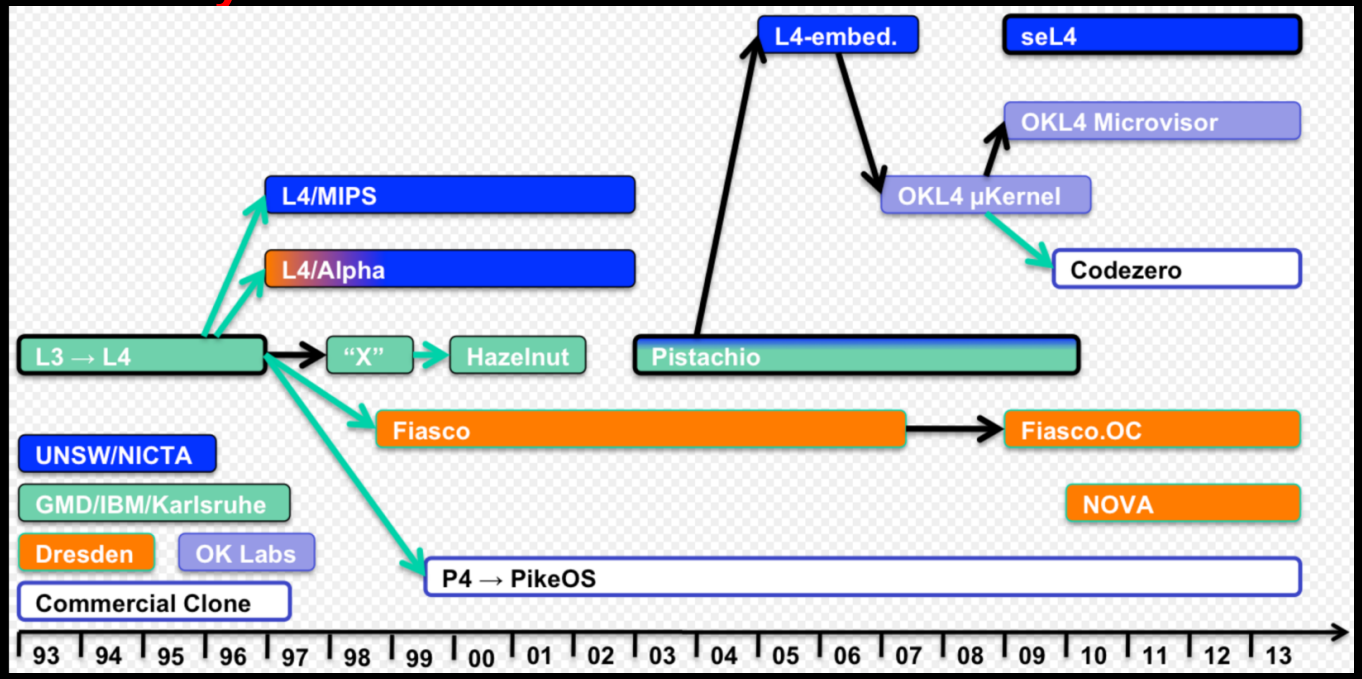
L4 is a family of second-generation microkernels, used to implement a variety of types of operating systems (OS), though mostly for Unix-like, *Portable Operating System Interface* (POSIX) compliant types.

L4, like its predecessor microkernel L3, was created by German computer scientist Jochen Liedtke as a response to the poor performance of earlier microkernel-based OSes. Liedtke felt that a system designed from the start for high performance, rather than other goals, could produce a microkernel of practical use. His original implementation in hand-coded Intel i386-specific assembly language code in 1993 sparked intense interest in the computer industry.[*citation needed*] Since its introduction, L4 has been developed to be cross-platform and to improve security, isolation, and robustness.

There have been various re-implementations of the original binary L4 kernel application binary interface (ABI) and its successors, including *L4Ka::Pistachio* (Karlsruhe Institute of Technology), *L4/MIPS* (University of New South Wales (UNSW)), *Fiasco* (Dresden University of Technology (TU Dresden)). For this reason, the name *L4* has been generalized and no longer refers to only Liedtke's original implementation. It now applies to the whole microkernel family including the L4 kernel interface and its different versions.

L4 is widely deployed. One variant, OKL4 from Open Kernel Labs, shipped in billions of mobile devices.[1][2]

- L4 family tree



2.2 seL4

- https://en.wikipedia.org/wiki/L4_microkernel_family#High_assurance:_seL4

In 2006, the NICTA group commenced a from-scratch design of a third-generation microkernel, named seL4, with the aim of providing a basis for highly secure and reliable systems, suitable for satisfying security requirements such as those of Common Criteria and beyond. From the beginning, development aimed for formal verification of the kernel. To ease meeting the sometimes conflicting requirements of performance and verification, the team used a middle-out software process starting from an executable specification written in Haskell.[16] seL4 uses capability-based security access control to enable formal reasoning about object accessibility.

A formal proof of functional correctness was completed in 2009.[17] The proof provides a guarantee that the kernel's implementation is correct against its specification, and implies that it is free of implementation bugs such as deadlocks, livelocks, buffer overflows, arithmetic exceptions or use of uninitialised variables. seL4 is claimed to be the first-ever general-purpose operating-system kernel that has been verified.[17]

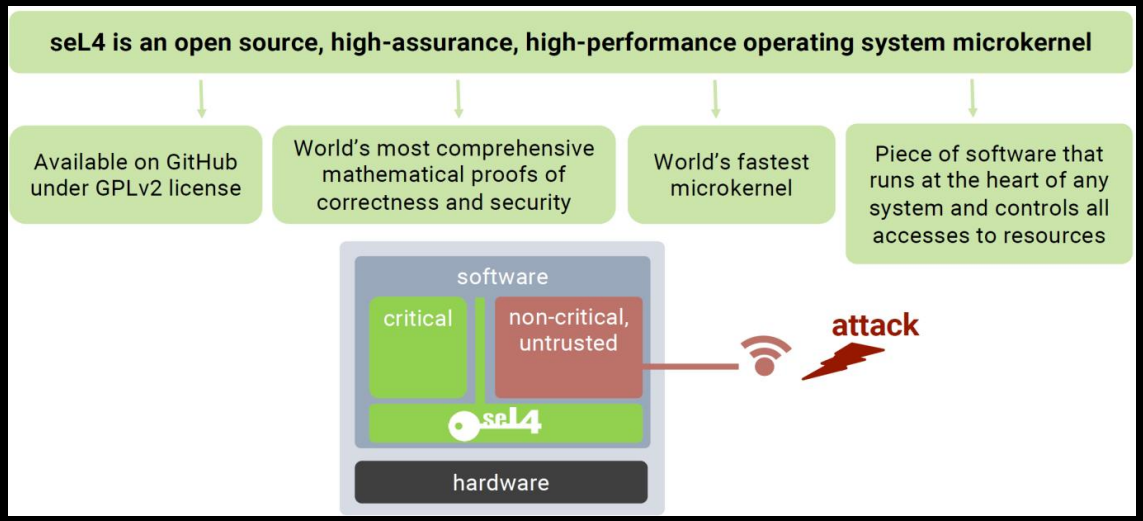
seL4 takes a novel approach to kernel resource management,[18] exporting the management of kernel resources to user level and subjects them to the same capability-based access control as user resources. This model, which was also adopted by Barrelfish, simplifies reasoning about isolation properties, and was an enabler for later proofs that seL4 enforces the core security properties of integrity and confidentiality.[19] The NICTA team also proved correctness of the translation from C to executable machine code, taking the compiler out of the trusted computing base of seL4.[20] This implies that the high-level security proofs hold for the kernel executable. seL4 is also the first published protected-mode OS kernel with a complete and sound worst-case execution-time (WCET) analysis, a prerequisite for its use in hard real-time systems.[19]

On 29 July 2014, NICTA and General Dynamics C4 Systems announced that seL4, with end to end proofs, was now released under open-source licenses.[21] The kernel source code and proofs are licensed under GNU General Public License version 2 (GPLv2), and most libraries and tools are under the BSD 2-clause. In April 2020, it was announced that the seL4 Foundation was created under the umbrella of the Linux Foundation to accelerate development and deployment of seL4.[22]

The researchers state that the cost of formal software verification is lower than the cost of engineering traditional "high-assurance" software despite providing much more reliable results.[23] Specifically, the cost of one line of code during the development of seL4 was estimated at around US$400, compared to US$1,000 for traditional high-assurance systems.[24]

Under the DARPA High-Assurance Cyber Military Systems (HACMS) program, NICTA together with project partners Rockwell Collins, Galois Inc, the University of Minnesota and Boeing developed a high-assurance drone based on seL4, along with other assurance tools and software, with planned technology transfer onto the optionally piloted autonomous Unmanned Little Bird helicopter under development by Boeing. Final demonstration of the HACMS technology took place in Sterling, VA in April 2017.[25] DARPA also funded several Small Business Innovative Research (SBIR) contracts related to seL4 under a program started by Dr. John Launchbury. Small businesses receiving an seL4-related SBIR included: DornerWorks, Techshot, Wearable Inc, Real Time Innovations, and Critical Technologies.[26]

- https://sel4.systems/
-



Source: "The seL4 Report", Gernot Heiser, Fosdem 2021

- https://github.com/seL4
- https://sel4.systems/Use/
- Data61, Linux Foundation launch seL4 open source foundation https://sel4.systems/Foundation/
- https://microkerneldude.wordpress.com/2020/03/11/sel4-design-principles/
- https://docs.sel4.systems/GettingStarted.html
- …

CAmkES

- https://docs.sel4.systems/projects/camkes/

CAmkES (component architecture for microkernel-based embedded systems) is a software development and runtime framework for quickly and reliably building microkernel-based multiserver (operating) systems. It follows a component-based software engineering approach to software design, resulting in a system that is modelled as a set of interacting software components. These software components have explicit interaction interfaces and a system design that explicitly details the connections between the components.

The development framework provides:

- a language to describe component interfaces, components, and whole component-based systems
- a tool that processes these descriptions to combine programmer-provided component code with generated scaffolding and glue code to build a complete, bootable, system image
- full integration in the seL4 environment and build system

- https://docs.sel4.systems/projects/camkes/terminology
- https://docs.sel4.systems/Tutorials/#camkes-tutorials
- https://github.com/seL4/camkes
- https://github.com/seL4/camkes-vm
- …

3) Unikernel

- https://en.wikipedia.org/wiki/Unikernel

A **unikernel** is a specialised, single address space machine image constructed by using library operating systems.[1] A developer selects, from a modular stack, the minimal set of libraries which correspond to the OS constructs required for the application to run. These libraries are then compiled with the application and configuration code to build sealed, fixed-purpose images (unikernels) which run directly on a hypervisor or hardware without an intervening OS such as Linux or Windows.

The first such systems were Exokernel and Nemesis in the late 1990s.

- https://en.wikipedia.org/wiki/Single_address_space_operating_system

In computer science, a **single address space operating system** (or **SASOS**) is an operating system that provides only one globally shared address space for all processes.

In a single address space operating system, numerically identical (virtual memory) logical addresses in different processes all refer to exactly the same byte of data.[1]

Single address-space operating systems offer many advantages. In a traditional OS with private per-process address space, memory protection is based on address space boundaries ("address space isolation"). Single address-space operating systems use a different approach for memory protection that is just as strong.[2][3]

One advantage is that the same virtual-to-physical map page table can be used with every process (and in some SASOS, the kernal as well). This makes context switches on a SASOS faster than on operating systems that must change the page table and flush the TLB caches on every context switch.

- https://en.wikipedia.org/wiki/Operating_system#Library
- https://en.wikipedia.org/wiki/Exokernel

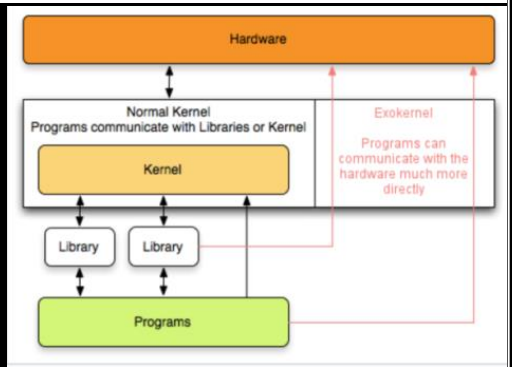
A library operating system is one in which the services that a typical operating system provides, such as networking, are provided in the form of libraries and composed with the application and configuration code to construct a unikernel: a specialized, single address space, machine image that can be deployed to cloud or embedded environments.

Exokernel is an operating system kernel developed by the MIT Parallel and Distributed Operating Systems group,[1] and also a class of similar operating systems.

Operating systems generally present hardware resources to applications through high-level abstractions such as (virtual) file systems. The idea behind exokernels is to force as few abstractions as possible on application developers, enabling them to make as many decisions as possible about hardware abstractions.[2] Exokernels are tiny, since functionality is limited to ensuring protection and multiplexing of resources, which is considerably simpler than conventional microkernels' implementation of message passing and monolithic kernels' implementation of high-level abstractions.

Implemented applications are called library operating systems; they may request specific memory addresses, disk blocks, etc. The kernel only ensures that the requested resource is free, and the application is allowed to access it. This low-level hardware access allows the programmer to implement custom abstractions, and omit unnecessary ones, most commonly to improve a program's performance. It also allows programmers to choose what level of abstraction they want, high, or low.

Exokernels can be seen as an application of the end-to-end principle to operating systems, in that they do not force an application program to layer its abstractions on top of other abstractions that were designed with different requirements in mind. For example, in the MIT Exokernel project, the Cheetah web server stores preformatted Internet Protocol packets on the disk, the kernel provides safe access to the disk by preventing unauthorized reading and writing, but how the disk is abstracted is up to the application or the libraries the application uses.



Graphic overview of Exokernel. Exokernels are much smaller than a normal kernel (monolithic kernel). They give more direct access to the hardware, thus removing most abstractions

Design

In a library operating system, protection boundaries are pushed to the lowest hardware layers, resulting in:

1. a set of libraries that implement mechanisms such as those needed to drive hardware or talk network protocols;
2. a set of policies that enforce access control and isolation in the application layer.

The library OS architecture has several advantages and disadvantages compared with conventional OS designs. One of the advantages is that since there is only a single address space, there is no need for repeated privilege transitions to move data between user space and kernel space. Therefore, a library OS can provide improved performance by allowing direct access to hardware without having to transition between user mode and kernel mode (on a traditional kernel this transition consists of a single TRAP instruction[2] and is not the same as a context switch[3]). Performance gains may be realised by elimination of the need to copy data between user space and kernel space, although this is also possible with Zero-copy device drivers in traditional operating systems.

A disadvantage is that because there is no separation, trying to run multiple applications side by side in a library OS, but with strong resource isolation, can become complex.[4] In addition, device drivers are required for the specific hardware the library OS runs on. Since hardware is rapidly changing this creates the burden of regularly rewriting drivers to remain up to date.

OS virtualization can overcome some of these drawbacks on commodity hardware. A modern hypervisor provides virtual machines with CPU time and strongly isolated virtual devices. A library OS running as a virtual machine only needs to implement drivers for these stable virtual hardware devices and can depend on the hypervisor to drive the real physical hardware. However, protocol libraries are still needed to replace the services of a traditional operating system. Creating these protocol libraries is where the bulk of the work lies when implementing a modern library OS.[1] Additionally, reliance on a hypervisor may reintroduce performance overheads when switching between the unikernel and hypervisor, and when passing data to and from hypervisor virtual devices.

By reducing the amount of code deployed, unikernels necessarily reduce the likely attack surface and therefore have improved security properties.[5][6]

An example unikernel-based messaging client has around 4% the size of the equivalent code bases using Linux.[7]

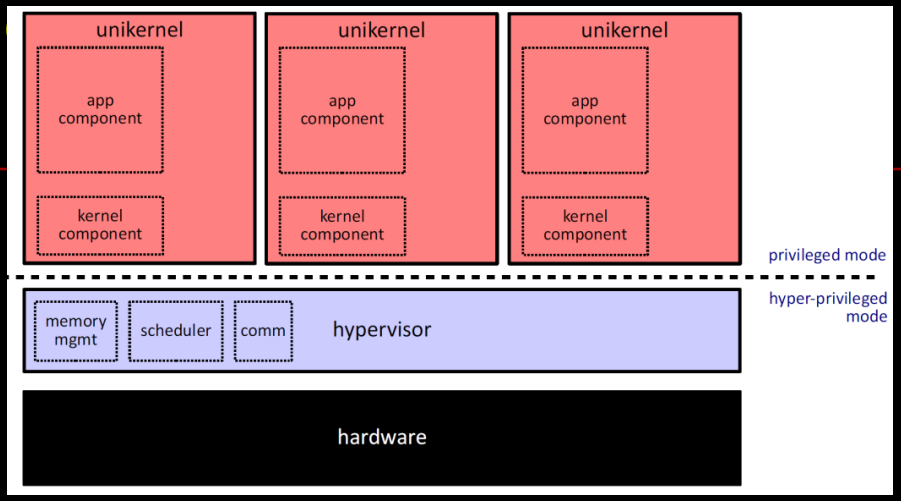
Due to the nature of their construction, it is possible to perform whole-system optimisation across device drivers and application logic, thus improving on the specialisation.[8][9] For example, off-the-shelf applications such as nginx, SQLite, and Redis running over a unikernel have shown a 1.7x-2.7x performance improvement.[10]

Unikernels have been regularly shown to boot extremely quickly, in time to respond to incoming requests before the requests time-out.[11][12][13]

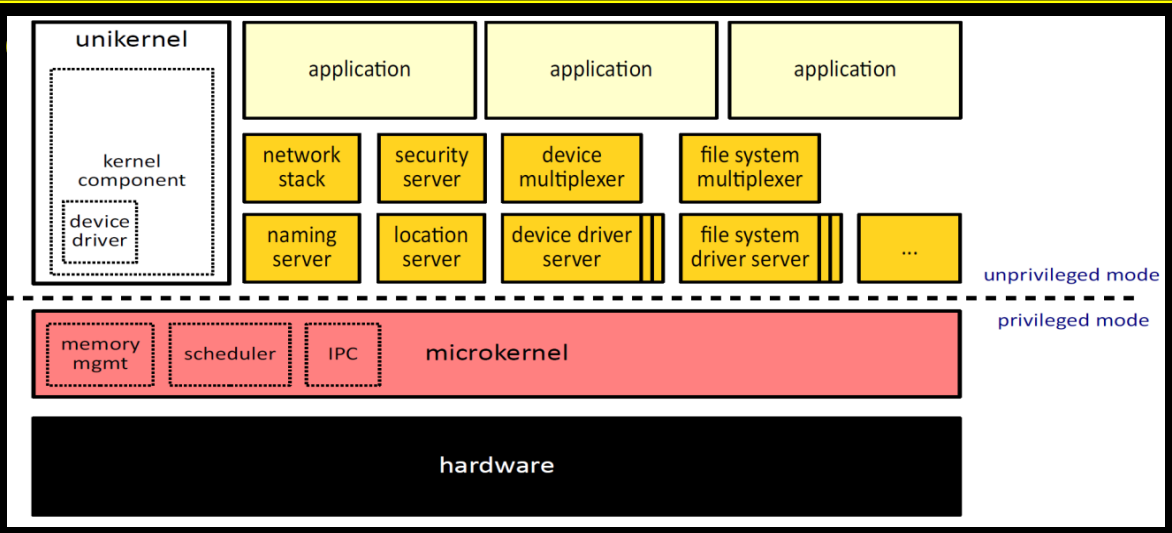
Unikernels lend themselves to creating systems that follow the service-oriented or microservices software architectures.

The high degree of specialisation means that unikernels are unsuitable for the kind of general purpose, multi-user computing that traditional operating systems are used for. Adding additional functionality or altering a compiled unikernel is generally not possible and instead the approach is to compile and deploy a new unikernel with the desired changes.

Hypervisor with Unikernels



Multiserver Microkernel with Unikernels for device drivers



Source: https://d3s.mff.cuni.cz/files/teaching/nswi161/martin-decky-microkernels-capabilities.pdf

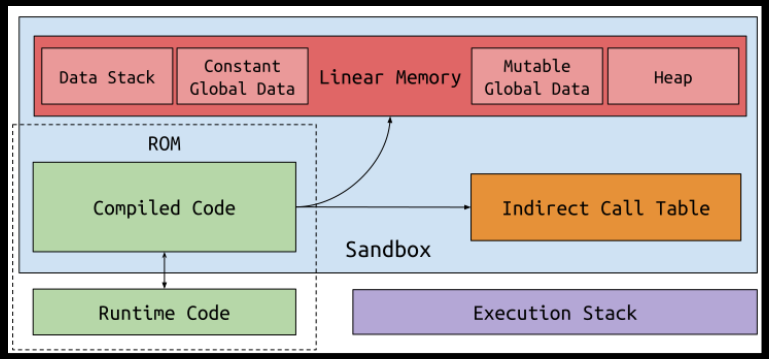
4) Wasm

- https://en.wikipedia.org/wiki/WebAssembly
- https://webassembly.org/

A binary instruction format for a stack-based VM. It is designed as a portable compilation target for programming languages…

4.1 Runtime

Sandboxing



Source: https://github.com/gwsystems/aWsm/blob/master/doc/design.md

Nanoprocesses

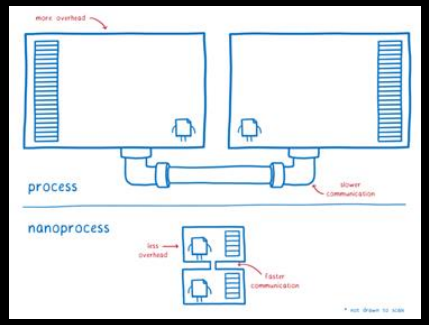
Tomorrow's solution: WebAssembly "nanoprocesses"

WebAssembly can provide the kind of isolation that makes it safe to run untrusted code. We can have an architecture that's like Unix's many small processes, or like containers and microservices.

But this isolation is much lighter weight, and the communication between them isn't much slower than a regular function call.

This means you can use them to wrap a single WebAssembly module instance, or a small collection of module instances that want to share things like memory among themselves.

Plus, you don't have to give up the nice programming language affordances—like function signatures and static type checking.



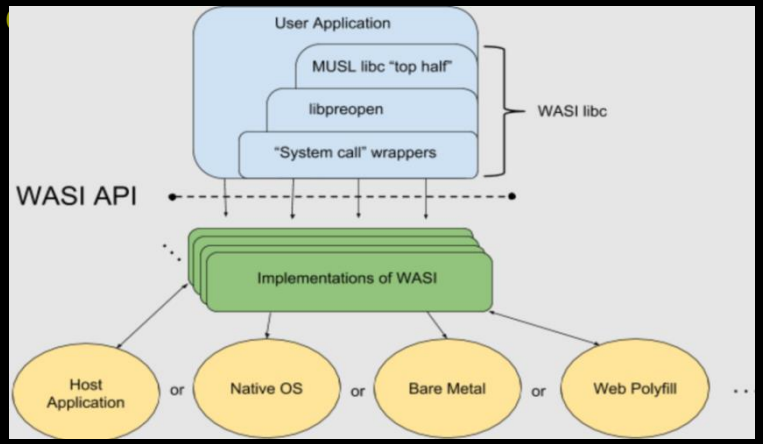
Source: https://hacks.mozilla.org/2019/11/announcing-the-bytecode-alliance/

4.2 WASI (WebAssembly System Interface)

- https://en.wikipedia.org/wiki/WebAssembly#WASI

> WebAssembly System Interface (WASI) is a simple interface (ABI and API) designed by Mozilla intended to be portable to any platform.[77] It provides POSIX-like features like file I/O constrained by capability-based security.[78][79] There are also a few other proposed ABI/APIs.[80][81]
>
> WASI is influenced by CloudABI and Capsicum.
>
> Solomon Hykes, a co-founder of Docker, wrote in 2019, "If WASM+WASI existed in 2008, we wouldn't have needed to create Docker. That's how important it is. WebAssembly on the server is the future of computing."[82] Wasmer, out in version 1.0, provides "software containerization, we create universal binaries that work anywhere without modification, including operating systems like Linux, macOS, Windows, and web browsers. Wasm automatically sandboxes applications by default for secure execution".[82]

- https://wasi.dev/
- https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/
- https://github.com/bytecodealliance/wasmtime/blob/main/docs/WASI-tutorial.md
-



Source: https://raw.githubusercontent.com/bytecodealliance/wasmtime/main/docs/wasi-software-architecture.png

- https://plus.qconferences.com/plus2021/presentation/wasi-new-kind-system-interface

5) Rust

- https://en.wikipedia.org/wiki/Rust_(programming_language)

Rust is a multi-paradigm, high-level, general-purpose programming language designed for performance and safety, especially safe concurrency.[12][13] Rust is syntactically similar to C++,[14] but can guarantee memory safety by using a *borrow checker* to validate references.[15] Rust achieves memory safety without garbage collection, and reference counting is optional.[16][17]

Rust was originally designed by Graydon Hoare at Mozilla Research, with contributions from Dave Herman, Brendan Eich, and others.[18][19] The designers refined the language while writing the Servo layout or browser engine,[20] and the Rust compiler. It has gained increasing use in industry, and Microsoft has been experimenting with the language for secure and safety-critical software components.[21][22]

Rust has been voted the "most loved programming language" in the Stack Overflow Developer Survey every year since 2016.[23]

- https://www.rust-lang.org/

- https://www.memorysafety.org/
- ...

Rust heads into Linux Kernel

- **What's happening!**

https://lwn.net/Articles/871283/ //Rust and GCC, two different ways
https://lwn.net/Articles/869428/ //More Rust concepts for the kernel
https://lwn.net/Articles/869317/ //Key Rust concepts for the kernel
https://lwn.net/Articles/853423/ //Rust heads into the kernel?
https://lwn.net/Articles/852704/ //Rust in the Linux kernel
https://lwn.net/Articles/849849/ //Rust support hits linux-next
https://lwn.net/Articles/829858/ //Supporting Linux kernel development in Rust
https://blogs.gartner.com/manjunath-bhat/2021/01/03/why-2021-will-be-a-rusty-year-for-system-programmers/
https://developers.slashdot.org/story/21/04/17/009241/linus-torvalds-says-rust-closer-for-linux-kernel-development-calls-c-a-crap-language
https://www.infoq.com/news/2021/04/rust-linux-kernel-development/
https://itwire.com/open-source/rust-support-in-linux-may-be-possible-by-5-14-release-torvalds.html

...

- **https://github.com/Rust-for-Linux**

The goal of this project is to add support for the Rust language to the Linux kernel. This repository contains the work that will be eventually submitted for review to the LKML.

Feel free to contribute! To start, take a look at `Documentation/rust`.

Rust for Cloud Native

- https://rust-cloud-native.github.io/

Applications and Services

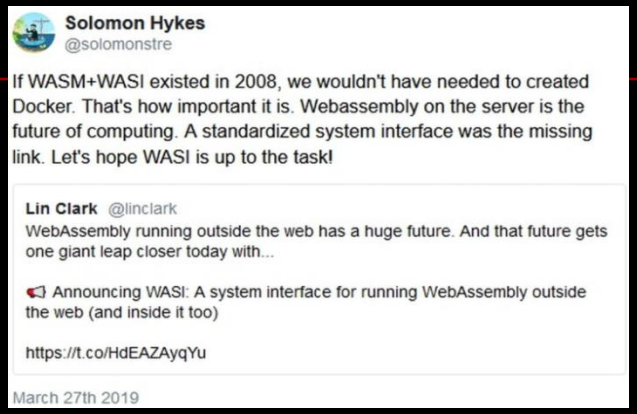
- apache/incubator-teaclave: open source universal secure computing platform, making computation on privacy-sensitive data safe and simple
- bottlerocket-os/bottlerocket: an operating system designed for hosting containers
- containers/krunvm: manage lightweight VMs created from OCI images
- containers/youki: a container runtime written in Rust
- datafuselabs/datafuse: A Modern Real-Time Data Processing & Analytics DBMS with Cloud-Native Architecture, built to make the Data Cloud easy
- firecracker-microvm/firecracker: secure and fast microVMs for serverless computing
- infinyon/fluvio: Cloud-native real-time data streaming platform with in-line computation capabilities
- krustlet/krustlet: Kubernetes Rust Kubelet
- kube-rs/controller-rs: a Kubernetes example controller
- kube-rs/version-rs: example Kubernetes reflector and web server
- kubewarden/policy-server: webhook server that evaluates WebAssembly policies to validate Kubernetes requests
- linkerd/linkerd2-proxy: a purpose-built proxy for the Linkerd service mesh
- openebs/mayastor: A cloud native declarative data plane in containers for containers
- rancher-sandbox/lockc: eBPF-based MAC security audit for container workloads
- tikv/tikv: distributed transactional key-value database
- tremor-rs/tremor-runtime: an event processing system that supporting complex workflows such as aggregation, rollups, an ETL language, and a query language
- valeriansaliou/sonic: fast, lightweight & schema-less search backend
- WasmEdge/WasmEdge: WasmEdge is a high-performance WebAssembly (Wasm) Virtual Machine (VM) runtime, which enables serverless functions to be embedded into any software platform; from cloud's edge to SaaS to automobiles

Libraries

- CNI Plugins: crate/framework to write CNI (container networking) plugins in Rust (includes a few custom plugins as well)
- containers/libkrun: a dynamic library providing Virtualization-based process isolation capabilities
- kube-rs/kube-rs: Kubernetes Rust client and async controller runtime
- qovery/engine: Qovery Engine is an open-source abstraction layer library that turns easy apps deployment on AWS, GCP, Azure, and other Cloud providers in just a few minutes
- open-telemetry/opentelemetry-rust: OpenTelemetry is a set of APIs, SDKs, tooling and integrations that are designed for the creation and management of telemetry data such as traces, metrics, and logs.

6) Userland

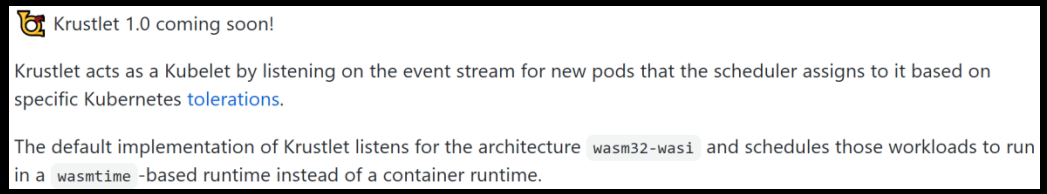
6.1 Replace Docker with WASM?



- Kubernetes is deprecating Docker as a container runtime after v1.20

6.3.1 Krustlet

- https://krustlet.dev/
- https://github.com/krustlet/krustlet
- Kubernetes Kubelet in Rust for running WASM.



- For Krustlet on RPi4, you may refer to my previous talk as below: "Cloud-Hypervisor on ARM" at the Rust China Meetup 2021(Hangzhou)

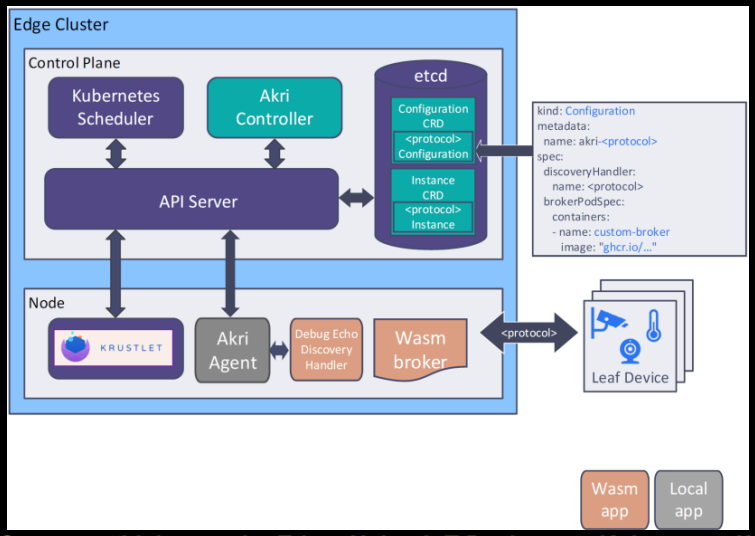
6.3.2 Akri

- https://docs.akri.sh/
- https://github.com/deislabs/akri

A Kubernetes Resource Interface for the Edge.

Akri lets you easily expose heterogeneous leaf devices (such as IP cameras and USB devices) as resources in a Kubernetes cluster, while also supporting the exposure of embedded hardware resources such as GPUs and FPGAs. Akri continually detects nodes that have access to these devices and schedules workloads based on them.

- https://github.com/deislabs/akri-on-krustlet
- Convert Akri from a containerized application to Wasm modules



Source: "Living on the Edge: Using IoT Devices on Kubernetes WebAssembly Applications", Kate Goldenring(Microsoft) and Rodrigo Rodrigues Lemos (Facebook), Cloud Native Wasm Day North America 2021.

- https://docs.akri.sh/demos/usb-camera-demo-rpi4
- https://github.com/deislabs/hippo //The WebAssembly PaaS

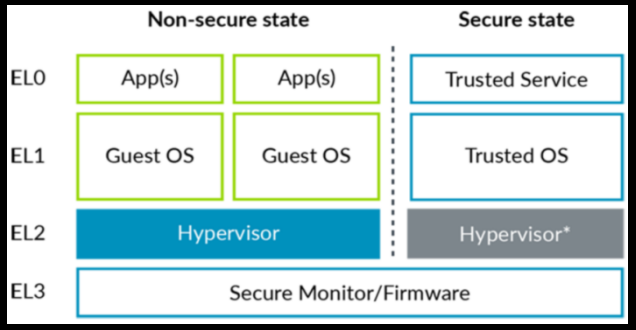
7) Virtualization & Security on ARM

- https://en.wikipedia.org/wiki/ARM_architecture
- https://www.arm.com/

6.1 Virtualization

Cortex-A

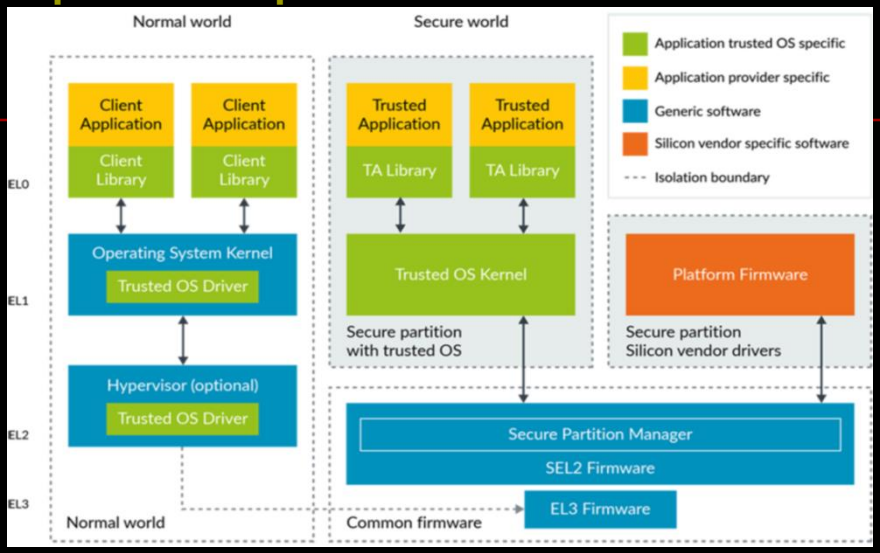
- https://developer.arm.com/documentation/100942/0100/



- https://developer.arm.com/documentation/102142/0100/Introduction-to-virtualization
- https://developer.arm.com/documentation/102142/0100/Virtualization-Host-Extensions
- https://developer.arm.com/documentation/102142/0100/Nested-virtualization
- https://developer.arm.com/ip-products/system-ip/system-controllers/interrupt-controllers

- **Secure Virtualization**

https://developer.arm.com/documentation/102142/0100/Secure-virtualization



- **System MMU(SMMU)**

https://en.wikipedia.org/wiki/Input-output_memory_management_unit
https://developer.arm.com/ip-products/system-ip/system-controllers/system-memory-management-unit

Segments	Use cases	Hypervisors	Whitepaper year and related ARM CoreLink System IP
Mobile	BYOD	VMware MVP	2011 - SMMU Automotive
Automotive	ADAS/IVI ECU consolidation	Green Hills (INTEGRITY) VirtualLogix	
Server	Live migration Rapid deployment Sandboxing	Xen / KVM	2017 - SMMU & GIC

7.2 Security

- https://en.wikipedia.org/wiki/ARM_architecture#Security_extensions

7.2.1 Trustzone

- https://developer.arm.com/ip-products/security-ip/trustzone
- https://developer.arm.com/ip-products/security-ip/trustzone/trustzone-system-ip
- https://globalplatform.org/specs-library/?filter-committee=tee

Cortex-A

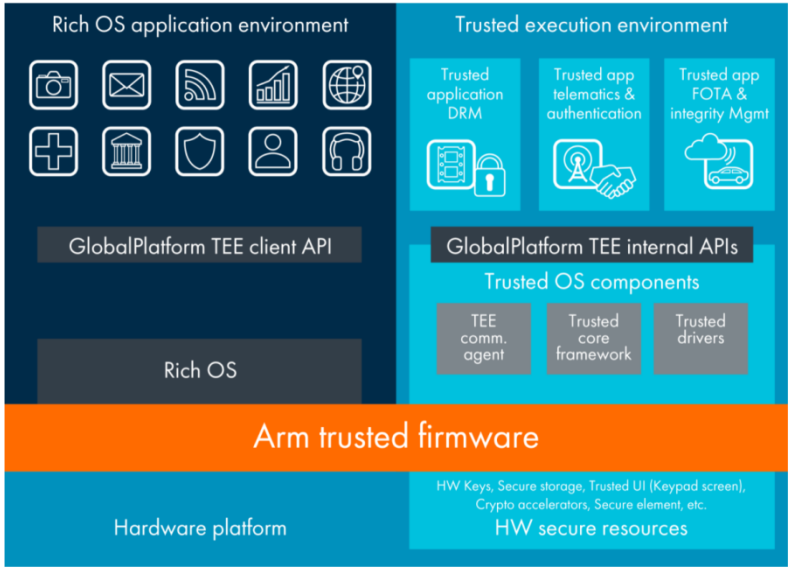
- https://developer.arm.com/ip-products/security-ip/trustzone/trustzone-for-cortex-a

Arm TrustZone is used on billions of applications' processors to protect high-value code and data. It is frequently used to provide a security boundary for a GlobalPlatform Trusted Execution Environment.

TrustZone is built on Secure and Non-secure worlds that are hardware separated. The partitioning of the two worlds is achieved by hardware logic present in the AMBA bus fabric, peripherals and processors.

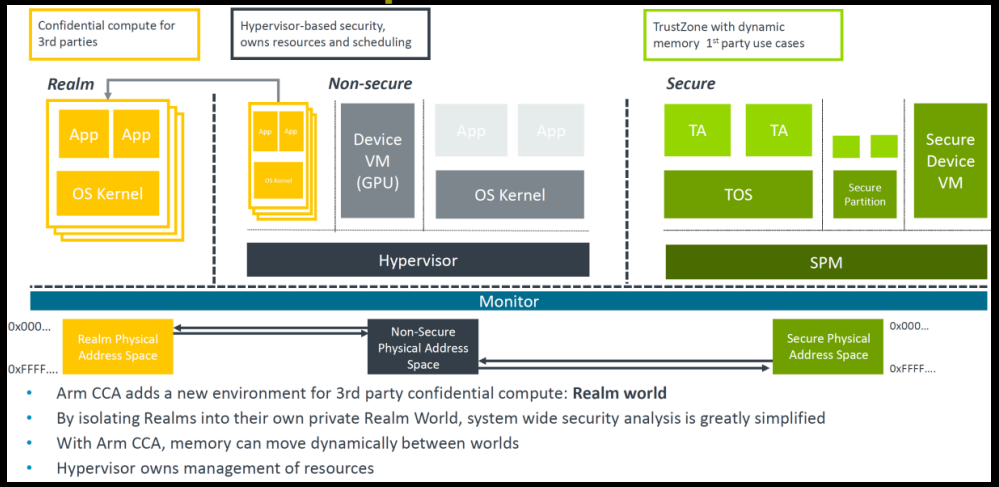
In order to implement a Secure state in the SoC, trusted software (Trusted OS) needs to be developed to make use of the protected assets. This code typically implements trusted boot, the Secure world switch monitor, a small trusted OS and trusted apps. The combination of TrustZone based hardware isolation, trusted boot and a trusted OS make up a Trusted Execution Environment (TEE), which can be used alongside other security technology.

Learn more about the GlobalPlatform TEE.

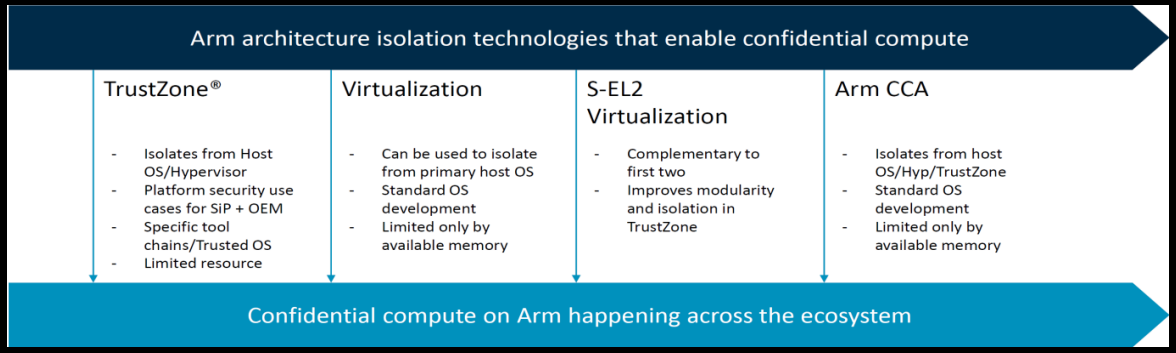


7.2.2 CCA(Confidential Compute Architecture)

- Since ARM v9
- https://www.arm.com/ja/why-arm/architecture/security-features/arm-confidential-compute-architecture



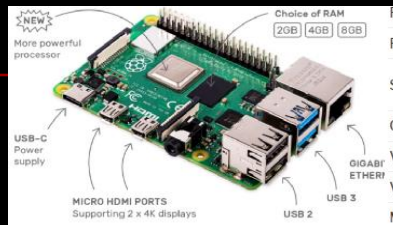
Confidential compute in ARM

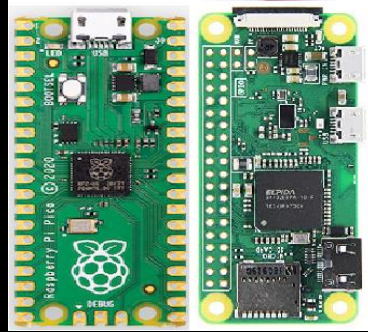


Source: https://connect.linaro.org/resources/armcca/introduction-to-the-arm-confidential-compute-architecture/

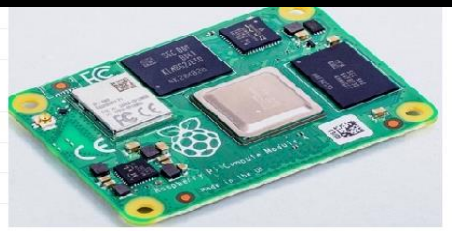
8) Testbed

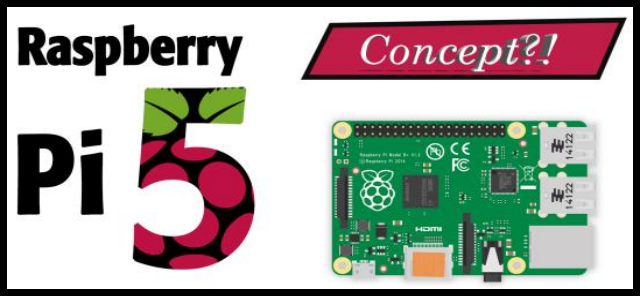
8.1 Raspberry Pi





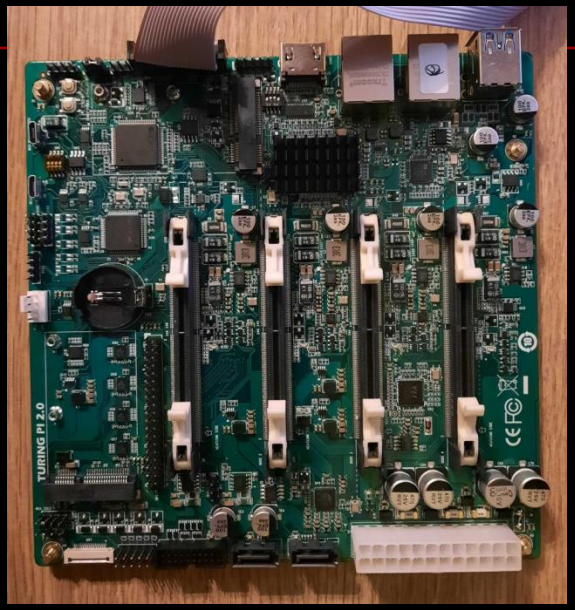
Features/Specs	Raspberry Pi 4B	Raspberry Pi 3 B+
Release date	24th June 2019	14th March 2018
SoC	Broadcom BCM2711 quad-core Cortex-A72 @ 1.5 GHz	Broadcom BCM2837B0 quad-core Cortex-A53 @ 1.4 GHz
GPU	VideoCore VI with OpenGL ES 1.1, 2.0, 3.0	VideoCore IV with OpenGL ES 1.1, 2.0
Video Decode	H.265 4Kp60, H.264 1080p60	H.264 & MPEG-4 1080p30
Video Encode	H.264 1080p30	
Memory	1GB, 2GB, or 4GB LPDDR4	1GB LPDDR2
Storage	microSD card	
Video & Audio Output	2x micro HDMI ports up to 4Kp60 3.5mm AV port (composite + audio) MIPI DSI connector	1x HDMI 1.4 port up to 1080p60 3.5mm AV port (composite + audio) MIPI DSI connector
Camera	MIPI CSI connector	
Ethernet	Native Gigabit Ethernet	Gigabit Ethernet over USB (300 Mbps max.)
WiFi	Dual band 802.11 b/g/n/ac	
Bluetooth	Bluetooth 5.0 + BLE	Bluetooth 4.2 + BLE
USB	2x USB 3.0 + 2x USB 2.0	4x USB 2.0
Expansion	40-pin GPIO header	
Power Supply	5V via USB type-C up to 3A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT	5V via micro USB up to 2.5A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT
Dimensions	85×56 mm	
Default OS	Raspbian (after June 24, 2019)	Raspbian (after March 2018)
Price	$35 (1GB RAM), $45 (2GB RAM), $55 (4GB RAM)	$35 (1GB RAM)

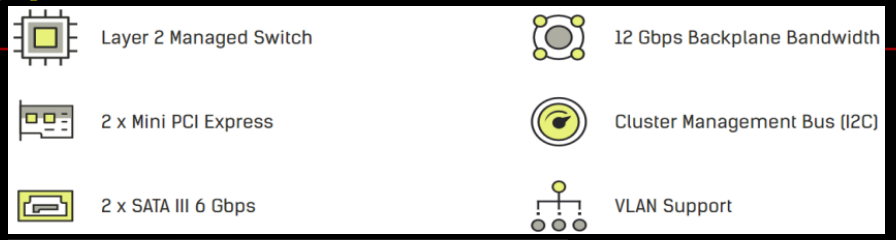


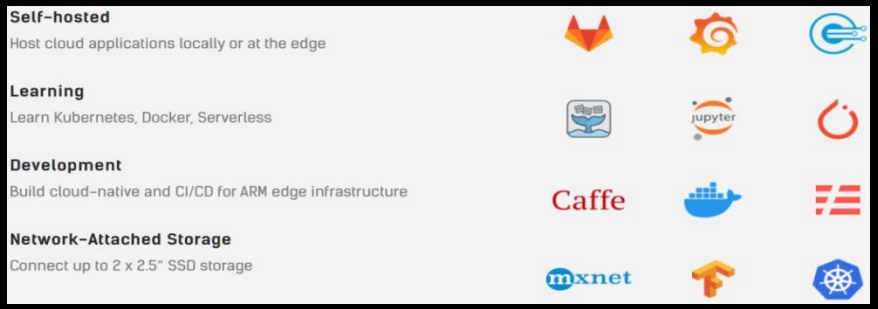


Turing Pi 2

- https://turingpi.com/v2/
- https://turingpi.com/turing-pi-v2-is-here/

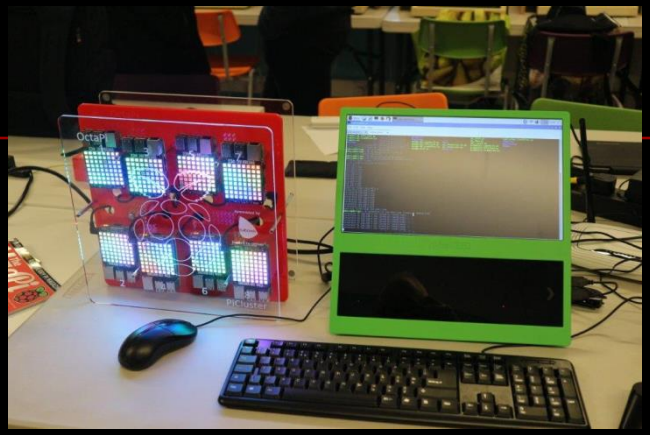


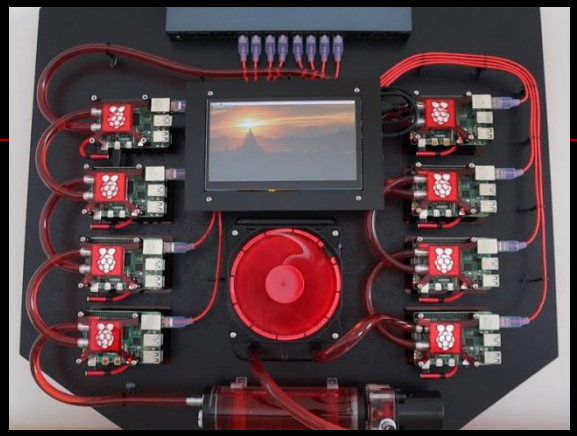




- The Turing Pi V2 now supports Raspberry Pi CM4, and Nvidia Jetson compute modules.

"Supercomputing" with RPi





Pico 20 Raspberry PI4 8GB

~~$ 758.00~~ $ 689.00 Sale

8.2 System

Fedora

- A Linux distribution developed by the community-supported Fedora Project which is sponsored primarily by Red Hat
- https://en.wikipedia.org/wiki/Fedora_(operating_system)
- https://getfedora.org/
- https://alt.fedoraproject.org/alt/
- https://spins.fedoraproject.org/
- https://fedoraproject.org/wiki/Architectures/ARM
- https://fedoramagazine.org/
- https://silverblue.fedoraproject.org/
- Developer friendly!

Current Dev Env

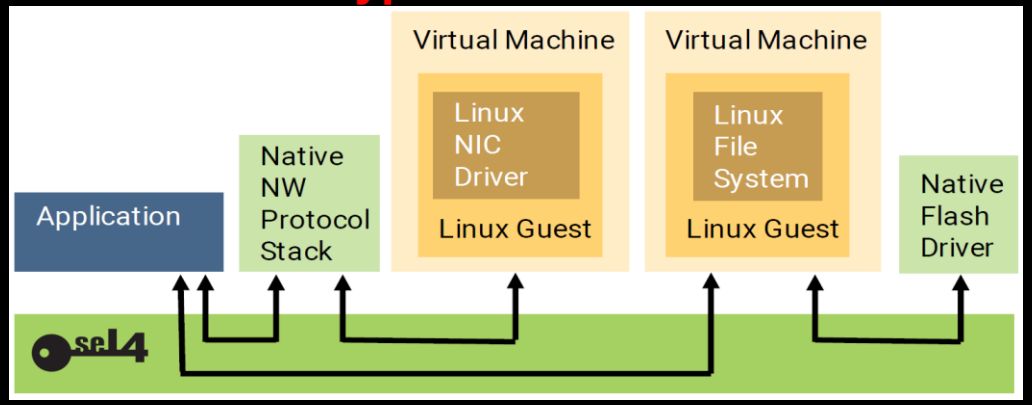
```
[mydev@fedora /]$ uname -a
Linux fedora 5.14.12-200.fc34.aarch64 #1 SMP Wed Oct 13 13:58:41 UTC 2021 aarch64 aarch64 aarch64 GNU/Linux
[mydev@fedora /]$
[mydev@fedora /]$ gcc -v
Using built-in specs.
COLLECT_GCC=/usr/bin/gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/aarch64-redhat-linux/11/lto-wrapper
Target: aarch64-redhat-linux
Configured with: ../configure --enable-bootstrap --enable-languages=c,c++,fortran,objc,obj-c++,ada,go,lto --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-shared --enable-threads=posix --enable-checking=release --enable-multilib --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-linker-build-id --with-gcc-major-version-only --with-linker-hash-style=gnu --enable-plugin --enable-initfini-array --with-isl=/builddir/build/BUILD/gcc-11.2.1-20210728/obj-aarch64-redhat-linux/isl-install --enable-gnu-indirect-function --build=aarch64-redhat-linux
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 11.2.1 20210728 (Red Hat 11.2.1-1) (GCC)
[mydev@fedora /]$
[mydev@fedora /]$ clang -v
clang version 12.0.1 (Fedora 12.0.1-1.fc34)
Target: aarch64-unknown-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
Found candidate GCC installation: /usr/bin/../lib/gcc/aarch64-redhat-linux/11
Found candidate GCC installation: /usr/lib/gcc/aarch64-redhat-linux/11
Selected GCC installation: /usr/lib/gcc/aarch64-redhat-linux/11
Candidate multilib: .;@m64
Selected multilib: .;@m64
[mydev@fedora /]$
[mydev@fedora /]$ rustup -V
rustup 1.24.3 (ce5817a94 2021-05-31)
info: This is the version for the rustup toolchain manager, not the rustc compiler.
info: The currently active `rustc` version is `rustc 1.55.0 (c8dfcfe04 2021-09-06)`
[mydev@fedora /]$
[mydev@fedora /]$ go version
go version go1.17.2 linux/arm64
[mydev@fedora /]$
[mydev@fedora /]$ java -version
openjdk version "17.0.1" 2021-10-19
OpenJDK Runtime Environment GraalVM CE 21.3.0 (build 17.0.1+12-jvmci-21.3-b05)
OpenJDK 64-Bit Server VM GraalVM CE 21.3.0 (build 17.0.1+12-jvmci-21.3-b05, mixed mode, sharing)
[mydev@fedora /]$
```

II. Virtualization on seL4

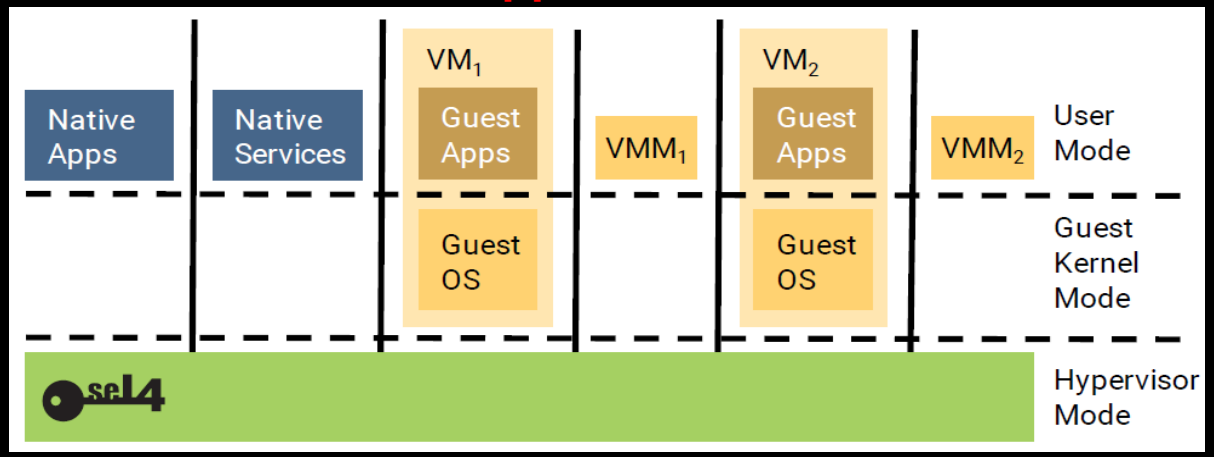
1) Overview

- https://sel4.systems/About/seL4-whitepaper.pdf

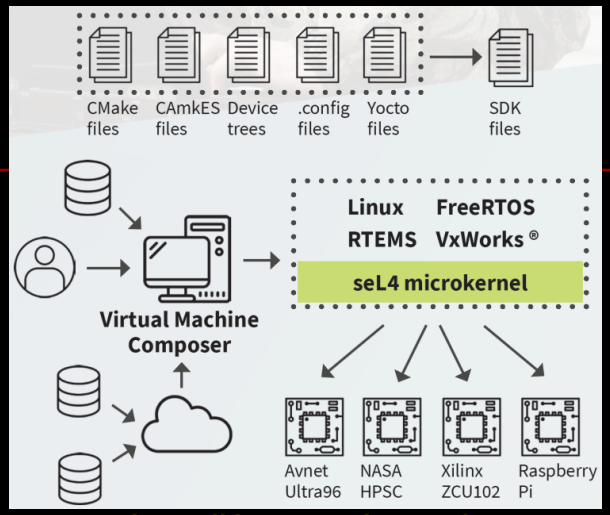
seL4 is also a hypervisor



seL4 virtualisation support with usermode VMMs



DornerWorks Virtual Machine Composer



Source: https://dornerworks.com/wp-content/uploads/2020/06/DornerWorks-Virtual-Machine-Composer-flyer.pdf

1.2 Implementation

- https://docs.sel4.systems/projects/virtualization/

Virtualisation Libraries

Virtualisation support on seL4 is underpinned by two libraries. These being:

- libsel4vm: A guest hardware virtualisation library for x86 (ia32) and ARM (ARMv7/w virtualization extensions & ARMv8)
- libsel4vmmplatsupport: A library containing various VMM utilities and drivers that can be used to construct a guest VM on a supported platform

These libraries can be utilized to construct VMM servers through providing useful interfaces to create VM(+VCPU) instances, manage *guest* physical address spaces and provide virtual device support (among multiple other things). Refer to each library to see further documentation regarding their features and APIs.

1.2.1 Libs

libsel4vm

- https://docs.sel4.systems/projects/virtualization/libsel4vm
- **Features**

- Hardware virtualisation support for the following architectures:
  - ARM
    - ARMv7 (+ Virtualisation Extensions)
    - ARMv8
  - X86
    - ia32 (Intel VTX)
- IRQ Controller emulation
  - GICv2 (aarch32, aarch64)
  - PIC & LAPIC (ia32)
- Guest VM Memory and RAM Management
- Guest VCPU Fault and Context Management
- VM Runtime Management

Architecture Specific Features

ARM

- SMP support for GICv2 (ARM) platforms

X86

- IOPort fault registration handler
- VMCall handler registration interface

- **Potential future features (yet to be implemented)**

Architecture Specific Features

ARM

- Virtual GICv3 support (aarch32 & aarch64)
  - SMP support for GICv3 platforms

X86

- x86-64 support (Intel VTX)
- SMP support on x86 platforms (ia32 & x86_64)

libsel4vmmplatsupport

- **https://docs.sel4.systems/projects/virtualization/libsel4vmmplatsupport**

- Virtio Support/Drivers
  - Virtio PCI
  - Virtio Console
  - Virtio Net
- Cross VM connection/communication driver
- Guest image loading utilities (e.g. kernel, initramfs)
- `libsel4vm` memory helpers and utilities
- IOPorts management interface

Architecture Specific Features

ARM

- VCPU fault handler module
  - HSR/Exception decoding
  - HSR/Exception handler dispatching
  - SMC decoding + handling
- PSCI handlers - through VCPU fault (+SMC handler) interface
- Generic virtual device interfaces
  - Generic access controlled devices - control read/write privileges to specific devices
  - Generic forwarding devices - interfaces for dispatching faults to external handlers
- Guest reboot utilities
- Virtual USB driver
- Guest OS Boot interfaces (for Linux VM's) : Boot VCPU initialisation

X86

- ACPI table generation
- PCI device passthrough helpers
- Guest OS Boot interfaces (for Linux VM's) : Boot VCPU initialisation, BIOS boot info structure generation, E820 map generation, VESA initialisation

Platform Specific Features

Exynos5422

- Virtual Clock device driver (Access Controlled Device)
- Virtual IRQ combiner device driver
- Virtual GPIO device driver
- Virtual Power device driver
- Virtual MCT device driver
- Virtual UART UART device driver

TK1

- USB Reboot Hooks

Potential future features (yet to be implemented)

- Additional Virtio Driver Support
  - e.g Virtio Blk, Virtio RNG, Virtio Balloon
- Block Driver support

Architecture Specific Features

ARM

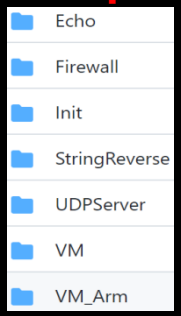
- Additional virtual devices for various supported platforms
- Additional platform/board support

x86

- Generic virtual device interface (as supported on ARM platforms)
- Additional virtual device support

2.2.1 CAmkES VM

- https://docs.sel4.systems/projects/camkes-vm
- Virtual Machine build as a CAmkES component.
- Components



CAmkES ARM VMM

- https://docs.sel4.systems/projects/camkes-arm-vm/
- Features

Plaform	Guest Mode	SMP Support	Multiple VM Support	Virtio PCI	Virtio Console	Virtio Net	Cross VM Connector Support	*Notes*
exynos5422	32-bit	Unsupported	Supported	Supported	Supported	Supported	Supported	SMP configurations are unsupported due to: * No exynos5422 kernel SMP support * No virtual power device interface to manage VCPU's at runtime (e.g. core startup)
TK1	32-bit	Unsupported	Unsupported	Unsupported	Unsupported	Unsupported	Unsupported	SMP configurations are unsupported due to: * No TK1 kernel SMP support * No virtual power device interface to manage VCPU's at runtime (e.g. core startup) Virtio PCI, Console, Net, Cross VM connector support & Multi-VM are untested
TX1	64-bit	Supported	Unsupported	Unsupported	Unsupported	Unsupported	Unsupported	Virtio PCI, Console, Net, Cross VM connector support & Multi-VM are untested
TX2	64-bit	Supported	Supported	Supported	Supported	Supported	Unsupported	Cross VM connector support is untested
QEMU Virt	64-bit	Supported	Unsupported	Supported	Supported	Supported	Supported	Multi-VM support depends on porting the TimeServer to QEMU (See https://github.com/seL4/global-components/tree/master/components/TimeServer)

Potential future features (yet to be implemented)

Architecture Specific Features

ARM

- Virtual GICv3 support (aarch32 & aarch64)
  - SMP support for GICv3 platforms

X86

- x86-64 support (Intel VTX)
- SMP support on x86 platforms (ia32 & x86_64)

III. seL4 on ARM

1) Overview

- https://docs.sel4.systems/Hardware/

seL4 has support for select ARMv6, ARMv7 and ARMv8 Platforms.

- General info on ARM Platforms

Platform	System-on-chip	Core	Arch	Virtualisation	IOMMU	Status	Contributed by	Maintained by
KZM	i.MX31	ARM1136J	ARMv6A	No	No	Unverified	Data61	Data61
BeagleBoard	OMAP3	Cortex-A8	ARMv7A	No	No	Unverified	Data61	Data61
BeagleBone Black / Blue	AM335x	Cortex-A8	ARMv7A	No	No	Unverified	Community	Data61
Inforce IFC6410	Snapdragon S4 Pro APQ8064	Krait (Cortex-A15 like)	ARMv7A	No	No	Unverified	Data61	No
OdroidXU	Exynos5	Cortex-A15	ARMv7A	ARM HYP	limited System MMU	Unverified	Data61	Data61
OdroidXU4	Exynos5	Cortex-A15	ARMv7A	ARM HYP	limited System MMU	Unverified	Data61	Data61
Zynq-7000 ZC706 Evaluation Kit	Zynq7000	Cortex-A9	ARMv7A	No	No	Unverified	Data61	Data61
Arndale	Exynos5	Cortex-A15	ARMv7A	ARM Hyp	No	Unverified	Data61	No
TK1-SOM	NVIDIA Tegra K1	Cortex-A15	ARMv7A	ARM HYP	System MMU	FC (without MMU)	Data61	Data61
TK1	NVIDIA Tegra K1	Cortex-A15	ARMv7A	ARM HYP	System MMU	Unverified	Data61	Data61
OdroidX	Exynos4412	Cortex-A9	ARMv7A	No	No	Unverified	Data61	Data61
Sabre Lite	i.MX6	Cortex-A9	ARMv7A	No	No	Verified	Data61	Data61
Raspberry Pi 3-b	BCM2837	Cortex-A53	ARMv8A	ARM HYP	No	Unverified	Data61	Data61
Zynq ZCU102 Evaluation Kit	Zynq UltraScale+ MPSoC	Cortex-A53	ARMv8A	ARM HYP	System MMU	Unverified	DornerWorks	DornerWorks
imx8mq	MCIMX8M-EVKB	Cortex-A53 Quad 1.5 GHz	ARMv8A	No	No	Unverified	Data61	Data61
HiKey	Kirin 620	Cortex-A53	ARMv8A	ARM HYP	No	Unverified	Data61	Data61
Imx8mm	IMX8MM-EVK	Cortex-A53 Quad 1.8 GHz	ARMv8A, AArch64	No	No	Unverified	Data61	Data61
Rockpro64	RK3399 hexa-core	Cortex-A53 Quad 1.8 GHz	ARMv8A, AArch64	No	No	Unverified	Data61	Data61
TX2	NVIDIA Tegra X2	Cortex-A57 Quad, Dual NVIDIA Denver	ARMv8A, AArch64 only	No	No	Unverified	Data61	Data61
Odroid-C2	Amlogic S905	Cortex-A53	ARMv8A, AArch64 only	No	No	Unverified	Data61	Data61
TX1	NVIDIA Tegra X1	Cortex-A57 Quad	ARMv8A, AArch64 only	ARM HYP	System MMU	Unverified	Data61	Data61

- https://docs.sel4.systems/Hardware/GeneralARM
- e.g., https://retout.co.uk/2020/05/11/sel4-on-rpi3-aarch64/...

- **there is no official guide for running seL4 on RPi4 or any successful demo case from the Internet, RPi4 support was said to be added(but it still seems in development stage):**

https://github.com/seL4/seL4/blob/master/CHANGES

12.1.0 2021-06-10: SOURCE COMPATIBLE

```
124     * Updated device definitions for the exynos5.
125     * Added Raspberry Pi 4 support.
126     * Updated Ethernet interrupts in the ZynqMP.
```

```
[mydev@fedora seL4]$ tree -L 2 src/plat/bcm2711
src/plat/bcm2711
├── config.cmake
└── overlay-rpi4.dts

0 directories, 2 files
[mydev@fedora seL4]$
[mydev@fedora seL4]$ tree -L 2 ./tools/dts
./tools/dts
├── allwinnerA20.dts
├── am335x-boneblack.dts
├── am335x-boneblue.dts
├── apq8064.dts
├── ariane.dts
├── exynos4.dts
├── exynos5250.dts
├── exynos5410.dts
├── exynos5422.dts
├── fvp.dts
├── hifive.dts
├── hikey.dts
├── imx7sabre.dts
├── imx8mm-evk.dts
├── imx8mq-evk.dts
├── mpfs_icicle.dts
├── nitrogen6sx.dts
├── odroidc2.dts
├── odroidc4.dts
├── omap3.dts
├── rocketchip.dts
├── rockpro64.dts
├── rpi3.dts
├── rpi4.dts
├── sabre.dts
├── spike32.dts
├── spike.dts
├── tk1.dts
├── tqma8xqp1gb.dts
├── tx1.dts
├── tx2.dts
├── ultra96.dts
├── update-dts.sh
├── wandq.dts
├── zynq7000.dts
└── zynqmp.dts
```

```
[mydev@fedora seL4]$ grep -ir "rpi3" .
grep: ./.git/index: binary file matches
./.github/workflows/sel4test-deploy.yml:          # - rpi3
./.github/workflows/sel4test-hw.yml:          # - rpi3
./src/plat/bcm2837/config.cmake:declare_platform(bcm2837 KernelPlatformRpi3 PLAT_BCM2837 KernelArchARM)
./src/plat/bcm2837/config.cmake:if(KernelPlatformRpi3)
./src/plat/bcm2837/config.cmake:    config_set(KernelARMPlatform ARM_PLAT rpi3)
./src/plat/bcm2837/config.cmake:    list(APPEND KernelDTSList "tools/dts/rpi3.dts")
./src/plat/bcm2837/config.cmake:    list(APPEND KernelDTSList "src/plat/bcm2837/overlay-rpi3.dts")
./src/plat/bcm2837/config.cmake:    DEP "KernelPlatformRpi3"
./tools/dts/update-dts.sh:bcm2837-rpi-3-b=rpi3
[mydev@fedora seL4]$
[mydev@fedora seL4]$ grep -ir "rpi4" .
grep: ./.git/objects/pack/pack-2e9e0538bf93a2b67cadb2c4abc52399adffec8d.pack: binary file matches
grep: ./.git/index: binary file matches
./src/plat/bcm2711/config.cmake:declare_platform(bcm2711 KernelPlatformRpi4 PLAT_BCM2711 KernelArchARM)
./src/plat/bcm2711/config.cmake:if(KernelPlatformRpi4)
./src/plat/bcm2711/config.cmake:        # set aarch64 as default for RPi4
./src/plat/bcm2711/config.cmake:    config_set(KernelARMPlatform ARM_PLAT rpi4)
./src/plat/bcm2711/config.cmake:    list(APPEND KernelDTSList "tools/dts/rpi4.dts")
./src/plat/bcm2711/config.cmake:    list(APPEND KernelDTSList "src/plat/bcm2711/overlay-rpi4.dts")
./src/plat/bcm2711/config.cmake:    DEP "KernelPlatformRpi4"
./tools/dts/update-dts.sh:broadcom/bcm2711-rpi-4-b=rpi4
[mydev@fedora seL4]$
```

https://github.com/seL4/ci-actions/blob/master/seL4-platforms/platforms.yml

```
201     RPI3:
202       arch: arm
203       modes: [32]
204       platform: rpi3
205       req: [rpi3]
206       image_platform: bcm2837
207       march: armv8a
208
209     RPI4:
210       arch: arm
211       modes: [64]
212       platform: rpi4
213       req: [rpi4]
214       image_platform: bcm2711
215       march: armv8a
216       no_hw_build: true
```

2) seL4 on RPi4

- Work in process

Please look forward to our upcoming follow-up of this talk like "Revisiting seL4 for secure Edge Cloud infrastructure"…

IV. Unikernel on seL4

1) The rumprun-sel4-demoapps

- https://github.com/seL4/rumprun
- https://github.com/seL4/rumprun-sel4-demoapps

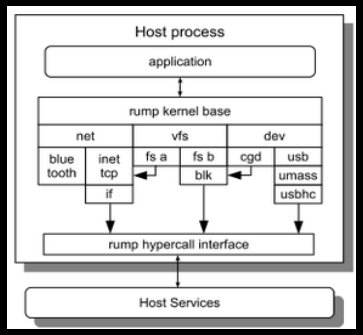
Rumprun

- https://en.wikipedia.org/wiki/Rump_kernel

The NetBSD **rump kernel** is the first implementation of the "anykernel" concept where drivers either can be compiled into or run in the monolithic kernel or in user space on top of a light-weight kernel.[1][2][3][4] The NetBSD drivers can be used on top of the rump kernel on a wide range of POSIX operating systems, such as the Hurd,[5] Linux, NetBSD, DragonFly BSD, Solaris kernels and even Cygwin, along with the file system utilities[6] built with the rump libraries. The rump kernels can also run without POSIX directly on top of the Xen hypervisor, an L4 microkernel using the Genode OS Framework[7] or even on "OS-less" bare metal.

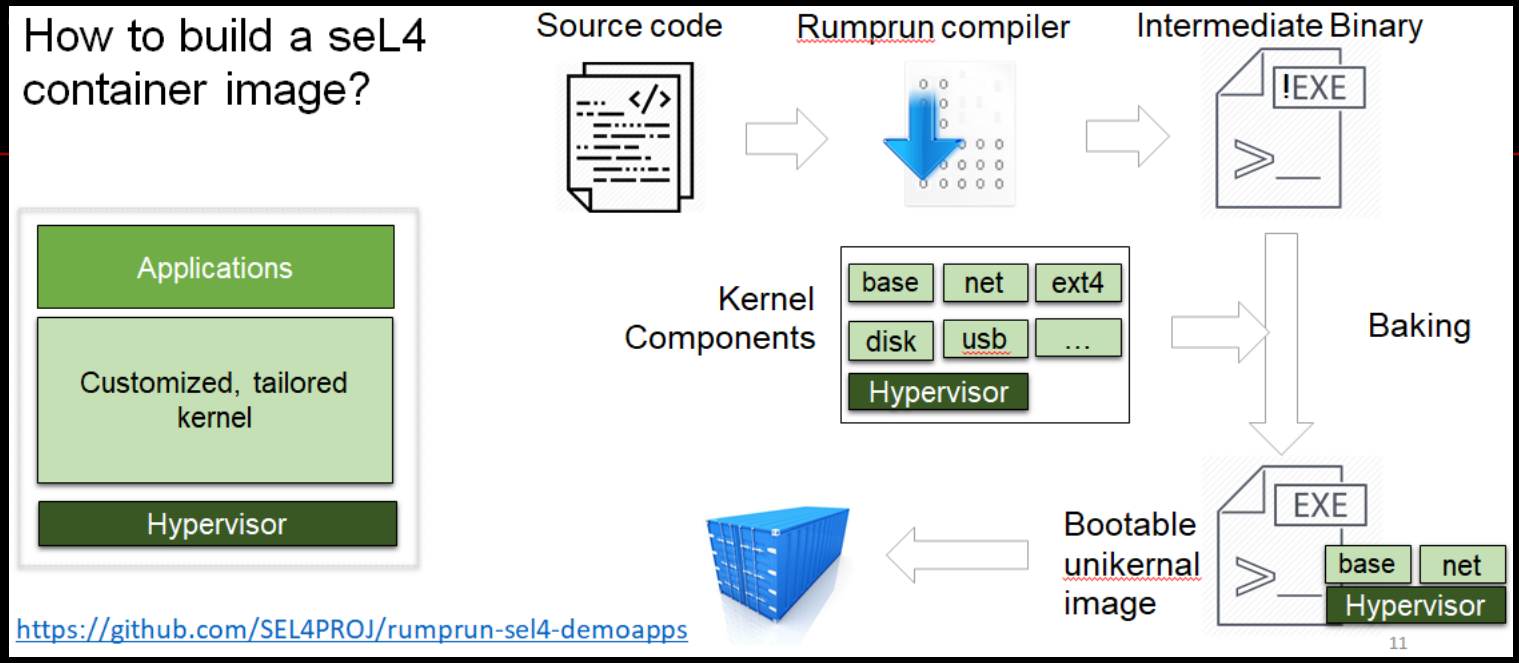
Anykernel [edit]

An anykernel is different in concept from microkernels, exokernels, partitioned kernels or hybrid kernels in that it tries to preserve the advantages of a monolithic kernel, while still enabling the faster driver development and added security in user space.[8] The "anykernel" concept refers to an architecture-agnostic approach to drivers where drivers can either be compiled into the monolithic kernel or be run as a userspace process, microkernel-style, without code changes.[9] With drivers, a wider concept is considered where not only device drivers are included but also file systems and the networking stack.



- https://rumpkernel.org/
- https://github.com/rumpkernel/rumprun

A Rumprun Unikernel Implementation



Source: "Enabling seL4 ContainerstoSupport Legacy Applications", Hui Lu, seL4 Summit 2019.

- # Test it on RPi4(work in process)

Installation steps

```
repo init -u $(GIT URL TO MANIFEST)
repo sync
(cd projects/rumprun && ./init-sources.sh)
mkdir build-hello && cd build-hello
../init-build.sh -DPLATFORM=x86_64 -DSIMULATION=TRUE -DAPP=hello
ninja
```

Running the image

QEMU

You can run by running a generated simulate script:

```
./simulate
# quit with Ctrl-A X
```

To add QEMU networking you will also need: `-net nic,model=e1000 -net tap,script=no,ifname=tap0` which requires a tap network interface configured.

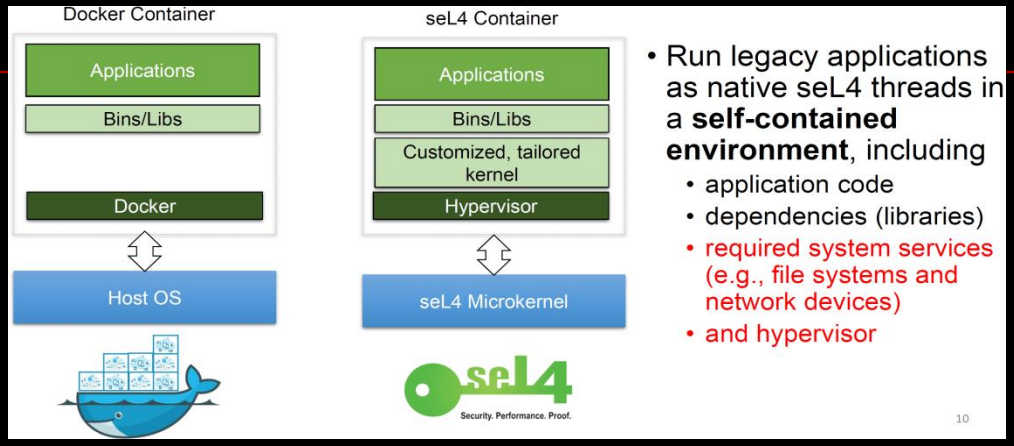
See https://docs.sel4.systems/Hardware/IA32 for running on actual hardware

https://github.com/seL4/rumprun-sel4-demoapps/blob/master/settings.cmake

```
16  include(${project_dir}/tools/seL4/cmake-tool/helpers/application_settings.cmake)
17
18  # Define our top level settings. Whilst they have doc strings for readability here
19  # Users should initialize a build directory by doing something like
20  # mkdir build
21  # cd build
22  # ../init-build.sh -DPLATFORM=x86_64 -DSIMULATION=TRUE -DAPP=Hello
23  set(SIMULATION OFF CACHE BOOL "Include only simulation compatible tests")
24  set(RELEASE OFF CACHE BOOL "Performance optimized build")
25  set(PLATFORM "x86_64" CACHE STRING "Platform to test")
26  set(supported_platforms x86_64 ia32)
27  set_property(CACHE PLATFORM PROPERTY STRINGS ${supported_platforms})
28  set(APP "hello" CACHE STRING "Application to build")
29
30  # Determine the platform/architecture
31  if(${PLATFORM} IN_LIST supported_platforms)
32      set(KernelArch x86 CACHE STRING "" FORCE)
33      set(KernelSel4Arch ${PLATFORM} CACHE STRING "" FORCE)
34      set(KernelPlatform pc99 CACHE STRING "" FORCE)
35  else()
36      message(FATAL_ERROR "Unknown PLATFORM. Initial configuration may not work")
37  endif()
```

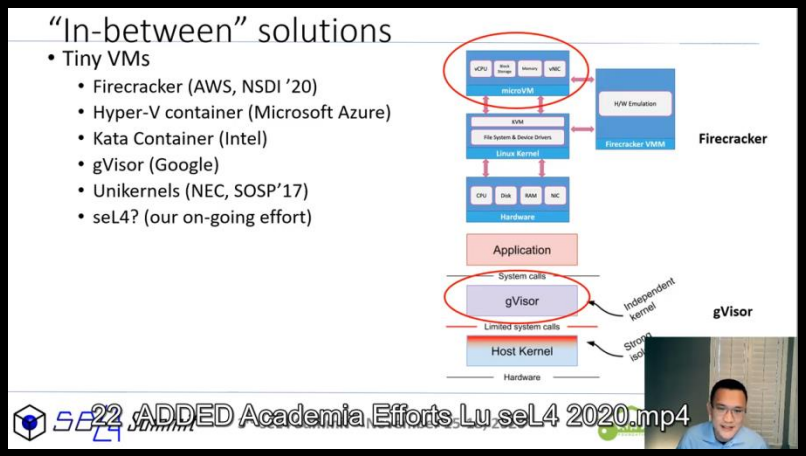
Trying some workarounds…

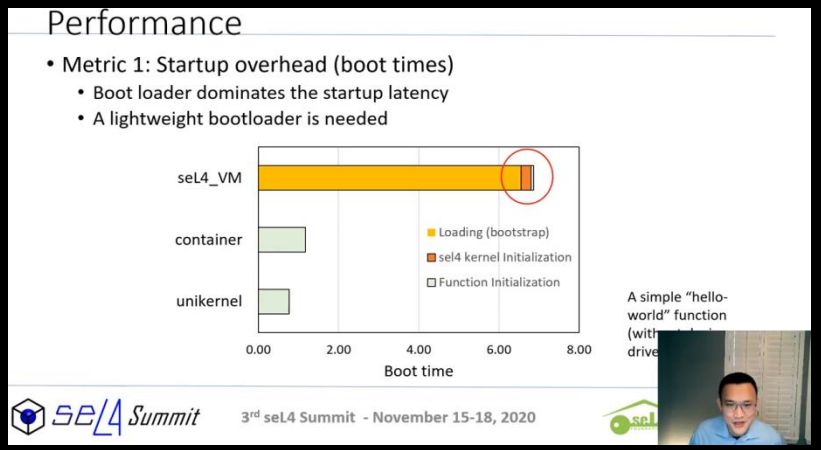
2) Tittle-tattle
2.1 seL4 Container



Source: "Enabling seL4 ContainerstoSupport Legacy Applications", Hui Lu, seL4 Summit 2019.

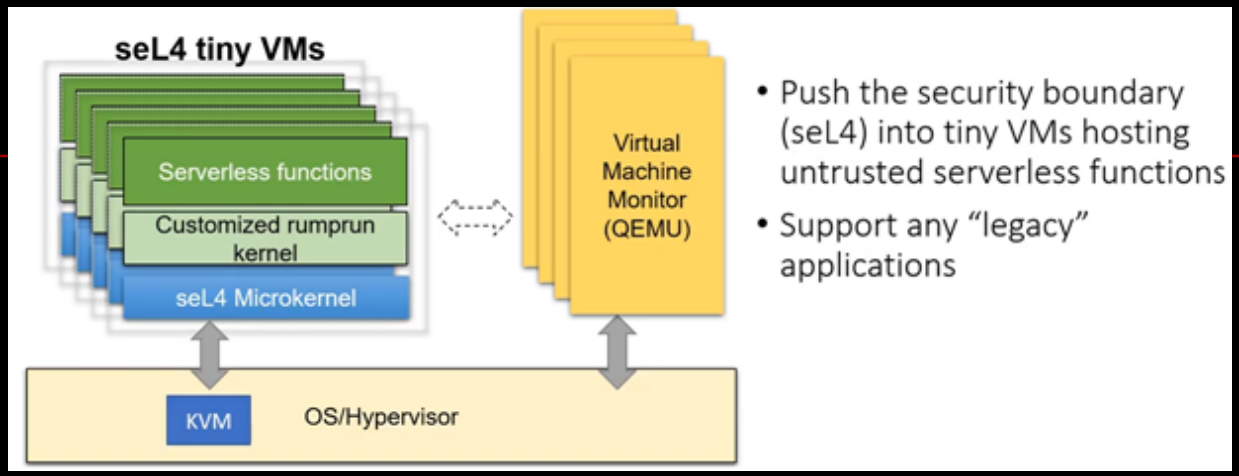
"Tiny VM"





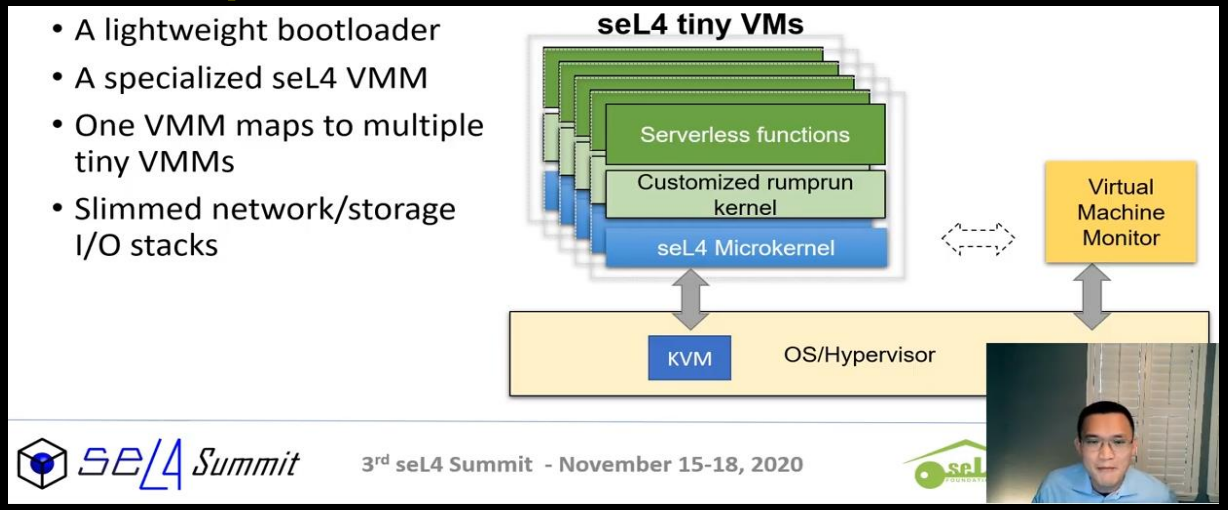
Source: "Securing Cloud Serverless Infrastructure with seL4", Hui Lu, seL4 Summit 2020.

seL4 Container for Serverless



Source: "Securing Cloud Serverless Infrastructure with seL4", Hui Lu, seL4 Summit 2020.

Work in process



Source: "Securing Cloud Serverless Infrastructure with seL4", Hui Lu, seL4 Summit 2020.

3) Considering a Unikernel more suitable for seL4

- Please look forward to our upcoming follow-up of this talk like "Revisiting seL4 for secure Edge Cloud infrastructure"…

VI. Wasm on seL4

1) WasmEdge

- https://github.com/second-state/wasmedge-seL4

1. Guest Linux OS on seL4: This is the controller of WasmEdge runtime, which will send wasm program to WasmEdge runner that is a agent on seL4 to execute.
2. WasmEdge runner on seL4: This is the wasm program runtime, which will execute the given wasm program from Guest Linux OS.

This demo is based on the seL4 simulator on Linux.

- https://www.youtube.com/watch?v=2Qu-Trtkspk
- Boot wasmedge-seL4

```
./simulate: qemu-system-aarch64 -machine virt,virtualization=on,highmem=off,secure=off -cpu cortex-a53 -nograp
ELF-loader started on CPU: ARM Ltd. Cortex-A53 r0p4
  paddr=[6abd8000..750cf0af]
No DTB passed in from boot loader.
Looking for DTB in CPIO archive...found at 6ad18f58.
Loaded DTB from 6ad18f58.
```

...

build.sh

- https://github.com/second-state/wasmedge-seL4/blob/main/build.sh

Executable File | 57 lines (47 sloc) | 1.93 KB

```
#!/bin/bash
# SPDX-License-Identifier: Apache-2.0
# This scripts checkout sel4 and wasmedge, and create an sel4 example app for running WebAssembly in sel4.
# If you see error like "undefined reference `__getauxval'", your aarch64-linux-gcc's version is 10 or greater.
# You can add this line to 'projects/musllibc/src/misc/getauxval.c'
# ```
# weak_alias(getauxval, __getauxval);
# ```

set -xuo pipefail
# Install dependency
# See https://docs.sel4.systems/projects/buildsystem/host-dependencies.html
sudo apt install \
  cmake ccache ninja-build cmake-curses-gui \
  python3-dev python3-pip \
  libxml2-utils ncurses-dev \
  curl git doxygen device-tree-compiler \
  u-boot-tools \
  protobuf-compiler python-protobuf \
  gcc-aarch64-linux-gnu g++-aarch64-linux-gnu \
  qemu-system-aarch64 libc6-dev-arm64-cross \
  haskell-stack

pip3 install --user setuptools sel4-deps camkes-deps
```

```
25
26  # Create root directory
27  mkdir -pv sel4_wasmedge
28  cd sel4_wasmedge
29
30  # Setup project files
31  git clone https://gerrit.googlesource.com/git-repo .repo/repo
32  .repo/repo/repo init -u https://github.com/second-state/wasmedge-seL4.git
33
34  # Update and checkout files
35  .repo/repo/repo sync
36
37  # Apply patches
38
39  patch -p1 -d projects/camkes-tool < .repo/manifests/patches/01-camkes-tool.patch
40  patch -p1 -d projects/llvm < .repo/manifests/patches/02-llvm.patch
41  patch -p1 -d projects/seL4_libs < .repo/manifests/patches/03-seL4_libs.patch
42  patch -p1 -d projects/seL4_projects_libs < .repo/manifests/patches/04-seL4_projects_libs.patch
43  patch -p1 -d projects/vm-examples < .repo/manifests/patches/05-vm-examples.patch
44  patch -p1 -d projects/vm-linux < .repo/manifests/patches/06-vm-linux.patch
45  patch -p1 -d projects/wasmedge < .repo/manifests/patches/07-wasmedge.patch
46
47  # Copy wasm examples
48  cp .repo/manifests/wasm-examples/*.wasm projects/vm-examples/apps/Arm/wasmedge/overlay_files/
49
50  # Configure seL4
51  mkdir -p build
52  cd build
53  ../init-build.sh -DCAMKES_VM_APP=wasmedge -DPLATFORM=qemu-arm-virt
54
55  # Build image
56  ninja
57  ninja
```

1.1 Attempt to run manually on RPi4

- sudo dnf install ghc cabal-install ghc-stack ghc-call-stack…
- pip3 install --user setuptools sel4-deps camkes-deps
- …
- apply the patches

```
[mydev@fedora sel4_wasmedge]$ patch -p1 -d projects/camkes-tool < .repo/manifests/patches/01-camkes-tool.patch
patching file camkes.cmake
patching file libsel4camkes/src/sys_clock.c
[mydev@fedora sel4_wasmedge]$
[mydev@fedora sel4_wasmedge]$ patch -p1 -d projects/llvm < .repo/manifests/patches/02-llvm.patch
patching file CMakeLists.txt
patching file Findlibcxx.cmake
patching file libcxxabi/include/features.h
[mydev@fedora sel4_wasmedge]$
[mydev@fedora sel4_wasmedge]$ patch -p1 -d projects/seL4_libs < .repo/manifests/patches/03-seL4_libs.patch
patching file libsel4debug/src/backtrace.c
patching file libsel4debug/src/sel4_arch/aarch64/stack_trace.c
patching file libsel4muslcsys/src/sys_morecore.c
patching file libsel4muslcsys/src/syscalls.h
patching file libsel4muslcsys/src/vsyscall.c
[mydev@fedora sel4_wasmedge]$
[mydev@fedora sel4_wasmedge]$ patch -p1 -d projects/seL4_projects_libs < .repo/manifests/patches/04-seL4_projects_libs.patch
patching file libsel4vm/src/arch/arm/vgic/vgic.h
[mydev@fedora sel4_wasmedge]$
[mydev@fedora sel4_wasmedge]$ patch -p1 -d projects/vm-examples < .repo/manifests/patches/05-vm-examples.patch
patching file apps/Arm/wasmedge/CMakeLists.txt
patching file apps/Arm/wasmedge/overlay_files/init_scripts/cross_vm_module_init
patching file apps/Arm/wasmedge/overlay_files/init_scripts/cross_vm_test
patching file apps/Arm/wasmedge/qemu-arm-virt/devices.camkes
patching file apps/Arm/wasmedge/settings.cmake
patching file apps/Arm/wasmedge/src/cross_vm_connections.c
patching file apps/Arm/wasmedge/src/wasmedge.cpp
patching file apps/Arm/wasmedge/wasmedge.camkes
[mydev@fedora sel4_wasmedge]$
[mydev@fedora sel4_wasmedge]$ patch -p1 -d projects/vm-linux < .repo/manifests/patches/06-vm-linux.patch
patching file camkes-linux-artifacts/camkes-linux-apps/wasmedge-emit/CMakeLists.txt
patching file camkes-linux-artifacts/camkes-linux-apps/wasmedge-emit/wasmedge_emit.c
[mydev@fedora sel4_wasmedge]$
[mydev@fedora sel4_wasmedge]$ patch -p1 -d projects/wasmedge < .repo/manifests/patches/07-wasmedge.patch
patching file CMakeLists.txt
patching file FindWasmEdge.cmake
patching file cmake/Helper.cmake
patching file include/common/defines.h
patching file include/common/timer.h
patching file include/common/variant.h
patching file include/host/wasi/environ.h
patching file include/host/wasi/inode.h
patching file lib/aot/compiler.cpp
patching file lib/ast/section.cpp
patching file lib/host/wasi/CMakeLists.txt
patching file lib/host/wasi/clock-linux.cpp
patching file lib/host/wasi/environ-linux.cpp
patching file lib/host/wasi/inode-linux.cpp
patching file lib/host/wasi/linux.h
patching file lib/host/wasmedge_process/processfunc.cpp
patching file lib/loader/CMakeLists.txt
patching file lib/loader/loader.cpp
patching file lib/loader/shared_library.cpp
patching file lib/system/allocator.cpp
patching file test/aot/AOTcoreTest.cpp
patching file test/host/wasi/wasi.cpp
[mydev@fedora sel4_wasmedge]$
```

- ...
- configure the build for ninja

```
[mydev@fedora sel4_wasmedge]$ ll
total 20
drwxr-xr-x. 1 mydev mydev 210 Oct 20 02:04 ./
drwxr-xr-x. 1 mydev mydev 304 Oct 20 12:06 ../
drwxr-xr-x. 1 mydev mydev   0 Oct 21 05:04 build/
lrwxrwxrwx. 1 mydev mydev  34 Oct 19 11:15 camkes_README.md -> projects/camkes-tool/docs/index.md
lrwxrwxrwx. 1 mydev mydev  40 Oct 19 11:18 easy-settings.cmake -> projects/vm-examples/easy-settings.cmake
drwxr-xr-x. 1 mydev mydev 986 Oct 19 10:31 git-repo.bak/
lrwxrwxrwx. 1 mydev mydev  29 Oct 19 11:18 griddle -> tools/seL4/cmake-tool/griddle*
lrwxrwxrwx. 1 mydev mydev  35 Oct 19 11:18 init-build.sh -> tools/seL4/cmake-tool/init-build.sh*
drwxr-xr-x. 1 mydev mydev 570 Oct 20 12:22 kernel/
drwxr-xr-x. 1 mydev mydev 334 Oct 19 11:18 projects/
lrwxrwxrwx. 1 mydev mydev  21 Oct 19 11:18 README.md -> projects/vm/README.md
drwxr-xr-x. 1 mydev mydev 228 Oct 19 11:15 .repo/
drwxr-xr-x. 1 mydev mydev  18 Oct 19 11:18 tools/
[mydev@fedora sel4_wasmedge]$
```

```
[mydev@fedora sel4_wasmedge]$ cat init-build.sh
#!/bin/sh
#
# Copyright 2020, Data61, CSIRO (ABN 41 687 119 230)
#
# SPDX-License-Identifier: BSD-2-Clause
#

# This script is intended to be symlinked into the same location as your root
# CMakeLists.txt file and then invoked from a clean build directory.

set -eu

# Determine path to this script (fast, cheap "dirname").
SCRIPT_PATH=${0%/*}
# Save script name for diagnostic messages (fast, cheap "basename").
SCRIPT_NAME=${0##*/}

# Ensure script path and current working directory are not the same.
if [ "$PWD" = "$SCRIPT_PATH" ]
then
    echo "\"$SCRIPT_NAME\" should not be invoked from top-level directory" >&2
    exit 1
fi

# Try and make sure we weren't invoked from a source directory by checking for a
# CMakeLists.txt file.
if [ -e CMakeLists.txt ]
then
    echo "\"$SCRIPT_NAME\" should be invoked from a build directory and not" \
         "source directories containing a CMakeLists.txt file" >&2
    exit 1
fi

if [ -d "$HOME/.sel4_cache" ]
then
    CACHE_DIR="$HOME/.sel4_cache"
else
    CACHE_DIR="$SCRIPT_PATH/.sel4_cache"
fi

if [ -e "$SCRIPT_PATH/CMakeLists.txt" ]
then
    # If we have a CMakeLists.txt in the top level project directory,
    # initialize CMake.
    cmake -DCMAKE_TOOLCHAIN_FILE="$SCRIPT_PATH"/kernel/gcc.cmake -G Ninja "$@" \
        -DSEL4_CACHE_DIR="$CACHE_DIR" -C "$SCRIPT_PATH/settings.cmake" "$SCRIPT_PATH"
else
    # If we don't have a CMakeLists.txt in the top level project directory then
    # assume we use the project's directory tied to easy-settings.cmake and resolve
    # that to use as the CMake source directory.
    real_easy_settings="$(realpath $SCRIPT_PATH/easy-settings.cmake)"
    project_dir="$(dirname $real_easy_settings)"
    # Initialize CMake.
    cmake -G Ninja "$@" -DSEL4_CACHE_DIR="$CACHE_DIR" -C "$project_dir/settings.cmake" "$project_dir"
fi
[mydev@fedora sel4_wasmedge]$
```

```
[mydev@fedora build]$ ../init-build.sh -DCAMKES_VM_APP=wasmedge -DPLATFORM=qemu-arm-virt
loading initial cache file /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/vm-examples/settings.cmake
-- Set platform details from PLATFORM=qemu-arm-virt
--   KernelPlatform: qemu-arm-virt
-- Found seL4: /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/kernel
Warning: no cpu specified for virt board, fallback on cortex-a53
QEMU MEMORY is: 2048
qemu-system-aarch64: info: dtb dumped to /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/build/virt.dtb. Exiting.
Warning: no cpu specified for virt board, fallback on cortex-a53
QEMU MEMORY is: 2048
qemu-system-aarch64: info: dtb dumped to /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/build/virt.dtb. Exiting.
-- The C compiler identification is unknown
-- The CXX compiler identification is unknown
-- The ASM compiler identification is unknown
-- Found assembler: aarch64-linux-gnu-gcc
CMake Error at CMakeLists.txt:9 (project):
  The CMAKE_C_COMPILER:

    aarch64-linux-gnu-gcc

  is not a full path and was not found in the PATH.

  Tell CMake where to find the compiler by setting either the environment
  variable "CC" or the CMake cache entry CMAKE_C_COMPILER to the full path to
  the compiler, or to the compiler name if it is in the PATH.


CMake Error at CMakeLists.txt:9 (project):
  The CMAKE_CXX_COMPILER:

    aarch64-linux-gnu-g++

  is not a full path and was not found in the PATH.

  Tell CMake where to find the compiler by setting either the environment
  variable "CXX" or the CMake cache entry CMAKE_CXX_COMPILER to the full path
  to the compiler, or to the compiler name if it is in the PATH.


CMake Error at CMakeLists.txt:9 (project):
  The CMAKE_ASM_COMPILER:

    aarch64-linux-gnu-gcc

  is not a full path and was not found in the PATH.

  Tell CMake where to find the compiler by setting either the environment
  variable "ASM" or the CMake cache entry CMAKE_ASM_COMPILER to the full path
  to the compiler, or to the compiler name if it is in the PATH.


-- Warning: Did not find file Compiler/-ASM
-- Configuring incomplete, errors occurred!
See also "/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/build/CMakeFiles/CMakeOutput.log".
See also "/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/build/CMakeFiles/CMakeError.log".
[mydev@fedora build]$
```

a workaround

```
[mydev@fedora kernel]$ git status
Not currently on any branch.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   gcc.cmake

no changes added to commit (use "git add" and/or "git commit -a")
[mydev@fedora kernel]$ git diff
diff --git a/gcc.cmake b/gcc.cmake
index bc3d33a0f..0f55a9dce 100644
--- a/gcc.cmake
+++ b/gcc.cmake
@@ -58,7 +58,7 @@ if("${CROSS_COMPILER_PREFIX}" STREQUAL "")
         if(${sel4_arch} STREQUAL "aarch32" OR ${sel4_arch} STREQUAL "arm_hyp")
             FindPrefixedGCC(CROSS_COMPILER_PREFIX "arm-linux-gnueabi-" "arm-linux-gnu-")
         elseif(${sel4_arch} STREQUAL "aarch64")
-            set(CROSS_COMPILER_PREFIX "aarch64-linux-gnu-")
+            set(CROSS_COMPILER_PREFIX "")
         elseif(${arch} STREQUAL "riscv")
             FindPrefixedGCC(
                 CROSS_COMPILER_PREFIX
[mydev@fedora kernel]$
```

Pthread issue

```
-- Adding Benchmark: vector_operations.bench.cpp
-- Found libcxx: /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/llvm
-- Looking for pthread.h
-- Looking for pthread.h - not found
CMake Error at /usr/share/cmake/Modules/FindPackageHandleStandardArgs.cmake:230 (message):
  Could NOT find Threads (missing: Threads_FOUND)
Call Stack (most recent call first):
  /usr/share/cmake/Modules/FindPackageHandleStandardArgs.cmake:594 (_FPHSA_FAILURE_MESSAGE)
  /usr/share/cmake/Modules/FindThreads.cmake:238 (FIND_PACKAGE_HANDLE_STANDARD_ARGS)
  /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/wasmedge/CMakeLists.txt:30 (find_package)


-- Configuring incomplete, errors occurred!
See also "/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/build/CMakeFiles/CMakeOutput.log".
See also "/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/build/CMakeFiles/CMakeError.log".
```

a fix:

```
[mydev@fedora wasmedge]$ git diff CMakeLists.txt
diff --git a/CMakeLists.txt b/CMakeLists.txt
index 97dc83d..47706ef 100644
--- a/CMakeLists.txt
+++ b/CMakeLists.txt
@@ -25,11 +25,14 @@ endif()

 include(FetchContent)

+set(CMAKE_THREAD_LIBS_INIT "-lpthread")
+set(CMAKE_HAVE_THREADS_LIBRARY 1)
+set(CMAKE_USE_WIN32_THREADS_INIT 0)
+set(CMAKE_USE_PTHREADS_INIT 1)
 set(THREADS_PREFER_PTHREAD_FLAG ON)
 find_package(Filesystem REQUIRED Final Experimental)
 find_package(Threads REQUIRED)
```

- # generated the build files

```
-- Found WasmEdge: /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/wasmedge
CMake Warning (dev) at /usr/share/cmake/Modules/ExternalProject.cmake:2218 (message):
  Policy CMP0114 is not set: ExternalProject step targets fully adopt their
  steps.  Run "cmake --help-policy CMP0114" for policy details.  Use the
  cmake_policy command to set the policy and suppress this warning.

  ExternalProject target 'wasmedge-emit' would depend on the targets for
  step(s) 'install' under policy CMP0114, but this is being left out for
  compatibility since the policy is not set.
Call Stack (most recent call first):
  /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/tools/seL4/cmake-tool/helpers/external-project-helpers.cmake:34 (ExternalProject_Add_StepTargets)
  /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/vm-linux/vm-linux-helpers.cmake:172 (DeclareExternalProjObjectFiles)
  apps/Arm/wasmedge/CMakeLists.txt:124 (AddExternalProjFilesToOverlay)
This warning is for project developers.  Use -Wno-dev to suppress it.

-- /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/build/ast.pickle is out of date. Regenerating...
-- /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/build/camkes-gen.cmake is out of date. Regenerating...
-- Found extra flags for fserv.instance.bin: $<TARGET_PROPERTY:CAmkESComponent_FileServer,COMPONENT_C_FLAGS>;$<TARGET_PROPERTY:CAmkESComponent_FileServer_instance_fserv,COMPONENT_C_FLAGS>
-- Found extra flags for vm0.instance.bin: $<TARGET_PROPERTY:CAmkESComponent_VM,COMPONENT_C_FLAGS>;$<TARGET_PROPERTY:CAmkESComponent_VM_instance_vm0,COMPONENT_C_FLAGS>
-- Found extra flags for wasmedge.instance.bin: $<TARGET_PROPERTY:CAmkESComponent_WasmEdge,COMPONENT_C_FLAGS>;$<TARGET_PROPERTY:CAmkESComponent_WasmEdge_instance_wasmedge,COMPONENT_C_FLAGS>
-- Configuring done
-- Generating done
-- Build files have been written to: /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/build
[mydev@fedora build]$
```

- # Build errors

```
[mydev@fedora build]$ ninja
[4/753] Building CXX object apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/Unwind-seh.cpp.obj
FAILED: apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/Unwind-seh.cpp.obj
/usr/bin/ccache /usr/lib64/ccache/g++ --sysroot=/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/build  -I/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/llvm/libunwind/include -isystem musllibc/build-temp/stage/include -march=armv8-a   -D__KERNEL_64__ -nostdinc++ -g   -U__linux -U__linux__ -Werror=return-type -W -Wall -Wchar-subscripts -Wconversion -Wmismatched-tags -Wmissing-braces -Wno-unused-function -Wshadow -Wsign-compare -Wsign-conversion -Wstrict-aliasing=2 -Wstrict-overflow=4 -Wunused-parameter -Wunused-variable -Wwrite-strings -Wundef -Wno-suggest-override -Wno-error -pedantic -funwind-tables -nostdinc++ -D_DEBUG -D_LIBUNWIND_IS_NATIVE_ONLY -nostdinc -fno-pic -fno-pie -fno-stack-protector -fno-asynchronous-unwind-tables -ftls-model=local-exec -mstrict-align -mno-outline-atomics -O2 -g -fno-rtti -std=c++11  -fstrict-aliasing -fno-exceptions -fno-rtti -MD -MT apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/Unwind-seh.cpp.obj -MF apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/Unwind-seh.cpp.obj.d -o apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/Unwind-seh.cpp.obj -c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/llvm/libunwind/src/Unwind-seh.cpp
In file included from /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/llvm/libunwind/src/Unwind-seh.cpp:13:
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/llvm/libunwind/src/config.h:16:10: fatal error: assert.h: No such file or directory
   16 | #include <assert.h>
      |          ^~~~~~~~~~
compilation terminated.
[5/753] Building CXX object apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/libunwind.cpp.obj
FAILED: apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/libunwind.cpp.obj
/usr/bin/ccache /usr/lib64/ccache/g++ --sysroot=/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/build  -I/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/llvm/libunwind/include -isystem musllibc/build-temp/stage/include -march=armv8-a   -D__KERNEL_64__ -nostdinc++ -g   -U__linux -U__linux__ -Werror=return-type -W -Wall -Wchar-subscripts -Wconversion -Wmismatched-tags -Wmissing-braces -Wno-unused-function -Wshadow -Wsign-compare -Wsign-conversion -Wstrict-aliasing=2 -Wstrict-overflow=4 -Wunused-parameter -Wunused-variable -Wwrite-strings -Wundef -Wno-suggest-override -Wno-error -pedantic -funwind-tables -nostdinc++ -D_DEBUG -D_LIBUNWIND_IS_NATIVE_ONLY -nostdinc -fno-pic -fno-pie -fno-stack-protector -fno-asynchronous-unwind-tables -ftls-model=local-exec -mstrict-align -mno-outline-atomics -O2 -g -fno-rtti -std=c++11  -fstrict-aliasing -fno-exceptions -fno-rtti -MD -MT apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/libunwind.cpp.obj -MF apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/libunwind.cpp.obj.d -o apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/libunwind.cpp.obj -c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/llvm/libunwind/src/libunwind.cpp
In file included from /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/llvm/libunwind/src/libunwind.cpp:12:
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/llvm/libunwind/include/libunwind.h:18:10: fatal error: stdint.h: No such file or directory
   18 | #include <stdint.h>
      |          ^~~~~~~~~~
compilation terminated.
[7/753] Building C object apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/UnwindLevel1.c.obj
FAILED: apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/UnwindLevel1.c.obj
/usr/bin/ccache /usr/lib64/ccache/gcc --sysroot=/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/build  -I/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/llvm/libunwind/include -isystem musllibc/build-temp/stage/include -march=armv8-a   -D__KERNEL_64__ -g   -U__linux -U__linux__ -Werror=return-type -W -Wall -Wchar-subscripts -Wconversion -Wmismatched-tags -Wmissing-braces -Wno-unused-function -Wshadow -Wsign-compare -Wsign-conversion -Wstrict-aliasing=2 -Wstrict-overflow=4 -Wunused-parameter -Wunused-variable -Wwrite-strings -Wundef -Wno-suggest-override -Wno-error -pedantic -funwind-tables -nostdinc++ -D_DEBUG -D_LIBUNWIND_IS_NATIVE_ONLY -nostdinc -fno-pic -fno-pie -fno-stack-protector -fno-asynchronous-unwind-tables -ftls-model=local-exec -mstrict-align -mno-outline-atomics -O2 -g -fno-rtti -std=gnu11 -std=c99 -MD -MT apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/UnwindLevel1.c.obj -MF apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/UnwindLevel1.c.obj.d -o apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/UnwindLevel1.c.obj -c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/llvm/libunwind/src/UnwindLevel1.c
cc1: warning: command-line option '-nostdinc++' is valid for C++/ObjC++ but not for C
cc1: warning: command-line option '-Wmismatched-tags' is valid for C++/ObjC++ but not for C
cc1: warning: command-line option '-Wno-suggest-override' is valid for C++/ObjC++ but not for C
cc1: warning: command-line option '-fno-rtti' is valid for C++/D/ObjC++ but not for C
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/WasmEdge/sel4_wasmedge/projects/llvm/libunwind/src/UnwindLevel1.c:21:10: fatal error: inttypes.h: No such file or directory
   21 | #include <inttypes.h>
      |          ^~~~~~~~~~~~
compilation terminated.
[9/753] Building CXX object apps/Arm/wasmedge/libcxx/libunwind/src/CMakeFiles/unwind_static.dir/Unwind-EHABI.cpp.obj
ninja: build stopped: subcommand failed.
[mydev@fedora build]$
```

2) Our Approach

Containers vs WASI on K8S

Containers	WASI
High startup cost.	Low startup cost.
Bigger binary files and highest memory consumption.	Smaller .wasm files and lower memory consumption.
Security requires high starting cost.	High security by default.
Active open-source community.	Active open-source community.
Over 10 years of active development for Docker.	WASI was only created 2 years ago.
Access to multithreading.	Only able to run in a single thread.
Access to Network.	Limited network access.
Any library can be compiled to.	Not every library can be compiled to.

Source: "Living on the Edge: Using IoT Devices on Kubernetes WebAssembly Applications", Kate Goldenring (Microsoft) and Rodrigo Rodrigues Lemos (Facebook), Cloud Native Wasm Day North America 2021

2.1 GaaS(GraalVM as a Service)

- For container-oriented workload
- For details, please refer to my previous talk "Revisiting GraalVM-based unified runtime for eBPF & WebAssembly" at OpenInfra Days China 2021(Beijing) and upcoming follow-ups.
- …

2.2 aWsm/SLEdge

- For Wasm-oriented workload
- For details, please refer to my previous talk "AOT compilation based Wasm compiler and runtime for Serverless Edge computing" at OpenInfra Days China 2021(Beijing) and upcoming follow-ups.

Build SLEdge natively(without Docker)

For aWsm:

- Replacing Wasmception with WASI SDK…
- Note the latest progress on the wasi-abstract-interface branch of aWsm
- Successfully build WASI SDK on RPi4

```
[mydev@fedora /]$ tree -L 2 /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Official/wasi-sdk-main/build/install/opt/wasi-sdk
/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Official/wasi-sdk-main/build/install/opt/wasi-sdk
├── bin
│   ├── ar -> llvm-ar
│   ├── c++filt -> llvm-cxxfilt
│   ├── clang -> clang-12
│   ├── clang++ -> clang
│   ├── clang-12
│   ├── clang-apply-replacements
│   ├── clang-cl -> clang
│   ├── clang-cpp -> clang
│   ├── clang-format
│   ├── clang-tidy
│   ├── git-clang-format
│   ├── ld64.lld -> lld
│   ├── ld64.lld.darwinnew -> lld
│   ├── ld.lld -> lld
│   ├── lld
│   ├── lld-link -> lld
│   ├── llvm-ar
│   ├── llvm-cxxfilt
│   ├── llvm-dwarfdump
│   ├── llvm-nm
│   ├── llvm-objcopy
│   ├── llvm-objdump
│   ├── llvm-ranlib -> llvm-ar
│   ├── llvm-size
│   ├── llvm-strings
│   ├── llvm-strip -> llvm-objcopy
│   ├── nm -> llvm-nm
│   ├── objcopy -> llvm-objcopy
│   ├── objdump -> llvm-objdump
│   ├── ranlib -> llvm-ar
│   ├── size -> llvm-size
│   ├── strings -> llvm-strings
│   ├── strip -> llvm-objcopy
│   └── wasm-ld -> lld
├── lib
│   └── clang
└── share
    ├── clang
    ├── cmake
    ├── misc
    └── wasi-sysroot
```

- …

VII. seL4 in Automotive

1) SOAFEE

- https://www.arm.com/en/solutions/automotive/software-defined-vehicles

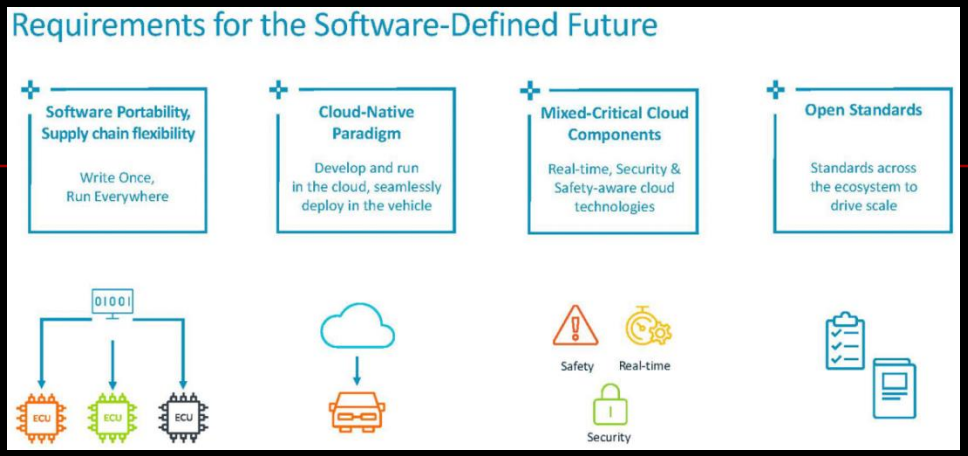
What it is

-

SOAFEE Summary		
	Key Information	**Other Information**
What is SOAFEE?	• Software framework for cloud & auto software • Open-source reference software framework • Reference hardware platform • Builds on Arm's Project Cassini • Leverages Arm SystemReady compliance	• Cloud-native: develop in cloud, deploy in car • To improve software-defined car • For early software development • Cloud-native ecosystem for edge apps • Hardwar-firmware certification standards
SOAFEE advantages	• Accelerates cloud-native tech to auto software • Increased software portability across platforms • Improved software quality and quantity • Software container advantages	• Expands current trends and capabilities • For Arm-based hardware and systems • Faster development and lower cost • Key to portable software modules
SOAFEE cloud-native technology	• Software containers: Portable & reusable apps • Microservice architecture: Lowers complexity • Orchestrator: Manages cloud microservices • DevOps: Cloud-native workflow aspects	• Virtualized app; tied to a specific OS • A service-oriented architecture (SOA) • Kubernetes is most common • Development & operational workflow
SOAFEE cloud-native enhancements	• Automotive orchestrator enhancements • Automotive container runtime enhancements • Automotive DevOps enhancements	• For safety and real-time functions • Proposal for virtualized container runtime • CI/CD testing and validations
SOAFEE availability	• Downloadable reference software stack • Reference hardware platforms for purchase	• https://gitlab.arm.com/soafee • 2 systems available from ADLINK
SOAFEE competition	• Will there be SOAFEE competitors? • Intel will need equivalent for automotive software • Nvidia can leverage SOAFEE	• For other processor platforms • The opportunity window will close quickly • Even if Arm acquisition does not happen
Summary	• Auto industry is moving to cloud-native software • SOAFEE accelerates this trend • SOAFEE is not a new Arm revenue stream	• Development, lifetime support & SaaS • Likely de facto standard for Arm systems • Grows the barrier for processor competitors
CI/CD=Continuous Integration/Continuous Development; OCI= Open Container Initiative		
Source: Egil Juliussen, September 2021		

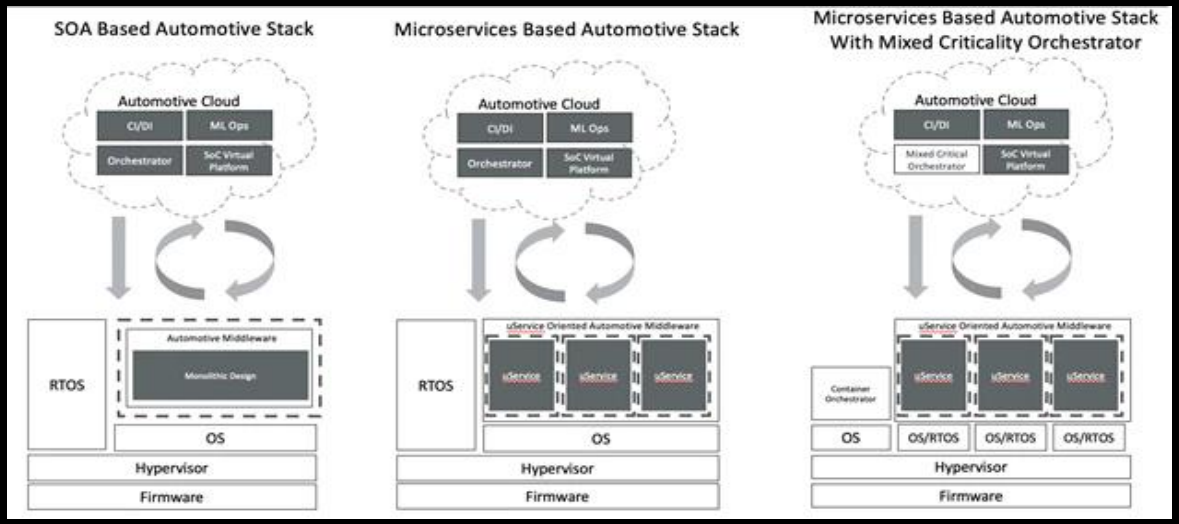
Source: https://www.eetimes.com/arms-soafee-brings-automotive-to-the-cloud/

Software defined



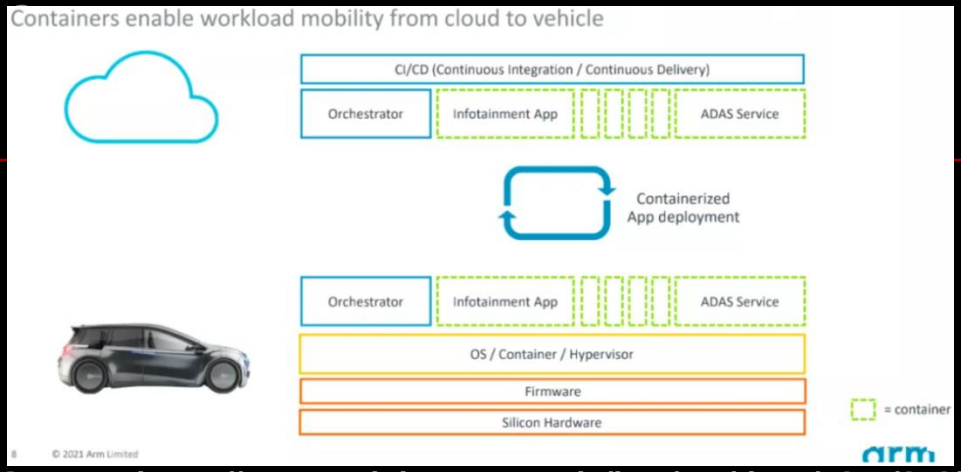
Source: https://car.watch.impress.co.jp/img/car/docs/1351/212/html/004.jpg.html

Cloud-native trend across the automotive stacks.



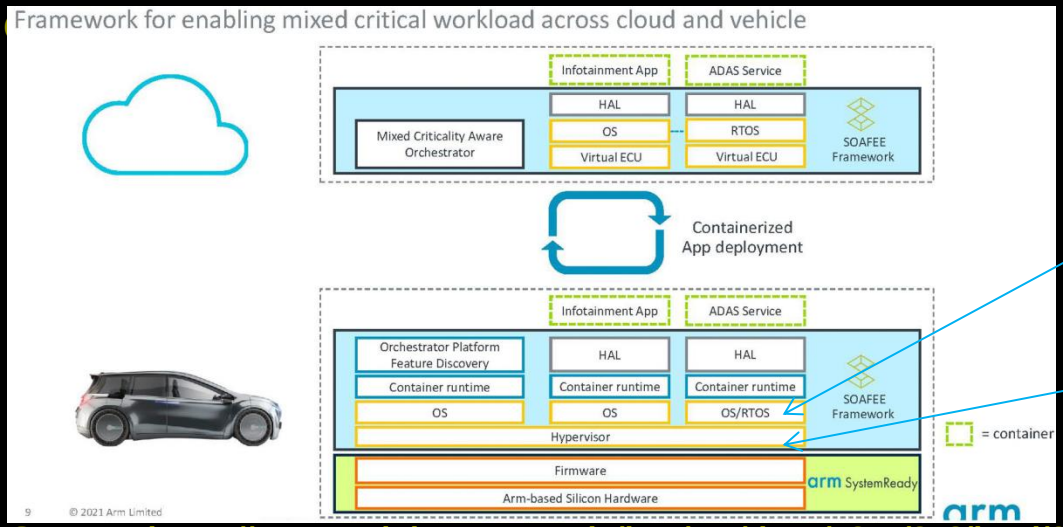
Source: https://community.arm.com/arm-community-blogs/b/embedded-blog/posts/cloud-native-approach-to-the-software-defined-car

A Software-Defined Future Requires a Cloud-Native Approach



Source: https://car.watch.impress.co.jp/img/car/docs/1351/212/html/004.jpg.html

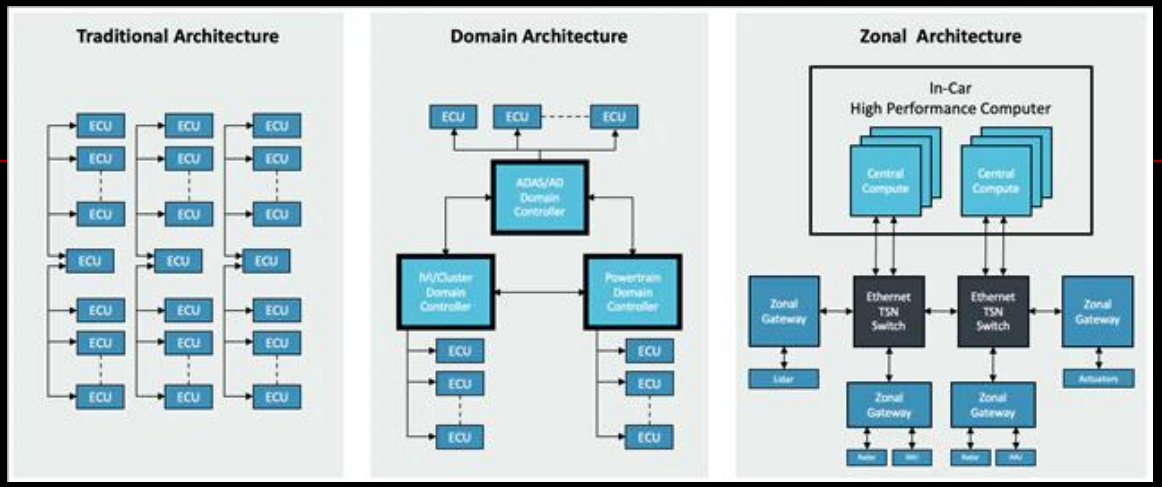
Accelerating Cloud Native Development with SOAFEE



seL4
or
seL4

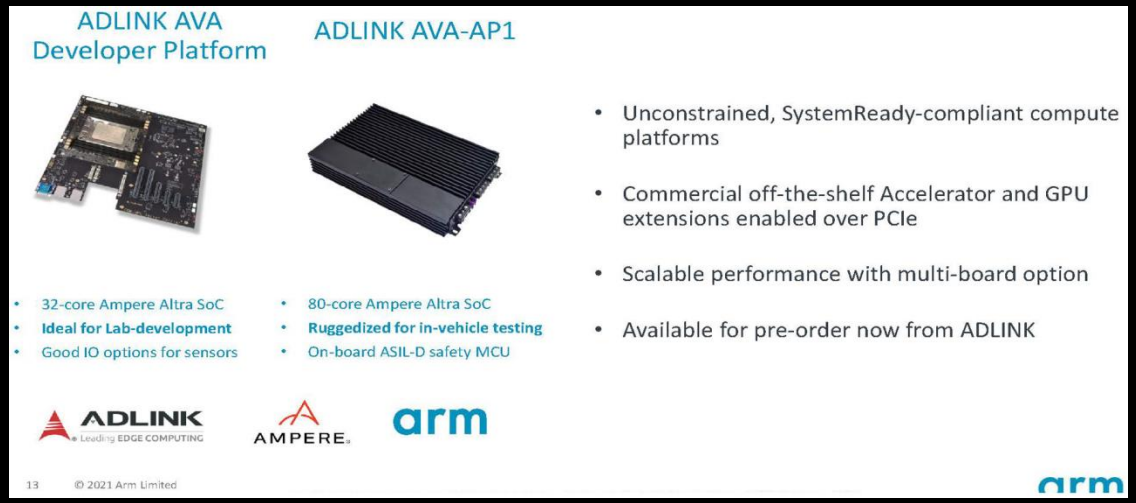
Source: https://car.watch.impress.co.jp/img/car/docs/1351/212/html/007.jpg.html

Automotive E/E Architecture Trend



Source: https://community.arm.com/arm-community-blogs/b/embedded-blog/posts/cloud-native-approach-to-the-software-defined-car

Reference Hardware Platforms



Source: https://car.watch.impress.co.jp/img/car/docs/1351/212/html/007.jpg.html

VII. Architecture and Design

1) WasmEdge

- https://wasmedge.org/
- https://www.secondstate.io/articles/wasmedge-joins-cncf/
- Key Features

WasmEdge is fully compatible with the W3C WebAssembly standard. Out of the box, it is supported by standard language and compiler tool chains, such as LLVM, Rustc and emscripten. WasmEdge is differentiated for its support for extensions, especially extensions for edge computing.

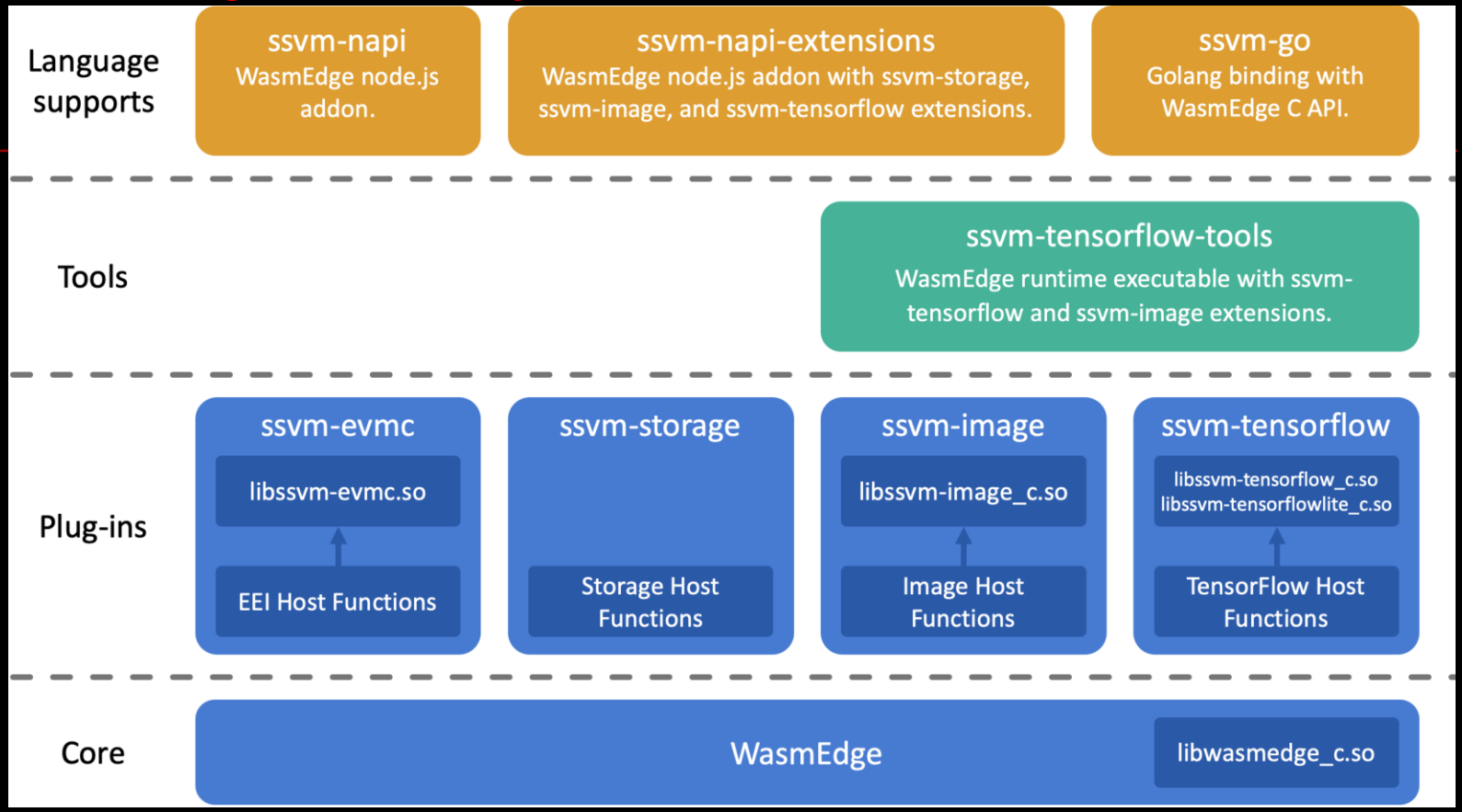
For starters, WasmEdge supports W3C optional WebAssembly features and proposals, such as WebAssembly System Interface(WASI) spec, Reference type, Bulk memory operations, and SIMD. We are also exploring the wasi-socket proposal to support network access in WebAssembly programs.

Furthermore, WasmEdge supports non-standard extensions designed for specific application scenarios.

- Tensorflow. Developers can write Tensorflow inference functions using a simple Rust API, and then run the function securely and at native speed inside WasmEdge. We are working on supporting other AI frameworks.
- Storage. The WasmEdge storage interface allows WebAssembly programs to read and write a key-value store.
- Command interface. WasmEdge enables webassembly functions to execute native commands in the host operating system. It supports passing arguments, environment variables, STDIN / STDOUT pipes, and security policies for host access.
- Ethereum. The WasmEdge Ewasm extension supports Ethereum smart contracts compiled to WebAssembly. It is a leading implementation for Ethereum flavored WebAssembly (Ewasm).
- Substrate. The Pallet allows WasmEdge to act as an Ethereum smart contract execution engine on any Substrate based blockchains.

Last but not least, WasmEdge is a “cloud-native” WebAssembly VM. It supports the OCI (Open Container Initiative)specification, which will allow WasmEdge instances to be managed by cloud-native orchestration tools such as Kubernetes.

WasmEdge is formerly called SSVM



2) Our Approach

- ARM-first.
- AI: PyTorch + TVM.
- Storage: Redis + Lua +…
- Unified runtime for eBPF and Wasm

  "GraalVM-based unified runtime for eBPF & Wasm" at GOTC 2021(Shenzhen).

  "Revisiting GraalVM-based unified runtime for eBPF & WebAssembly" at OpenInfra Days China 2021(Beijing).

  …
- Orchestration: follows Krustlet/Akri/Hippo,

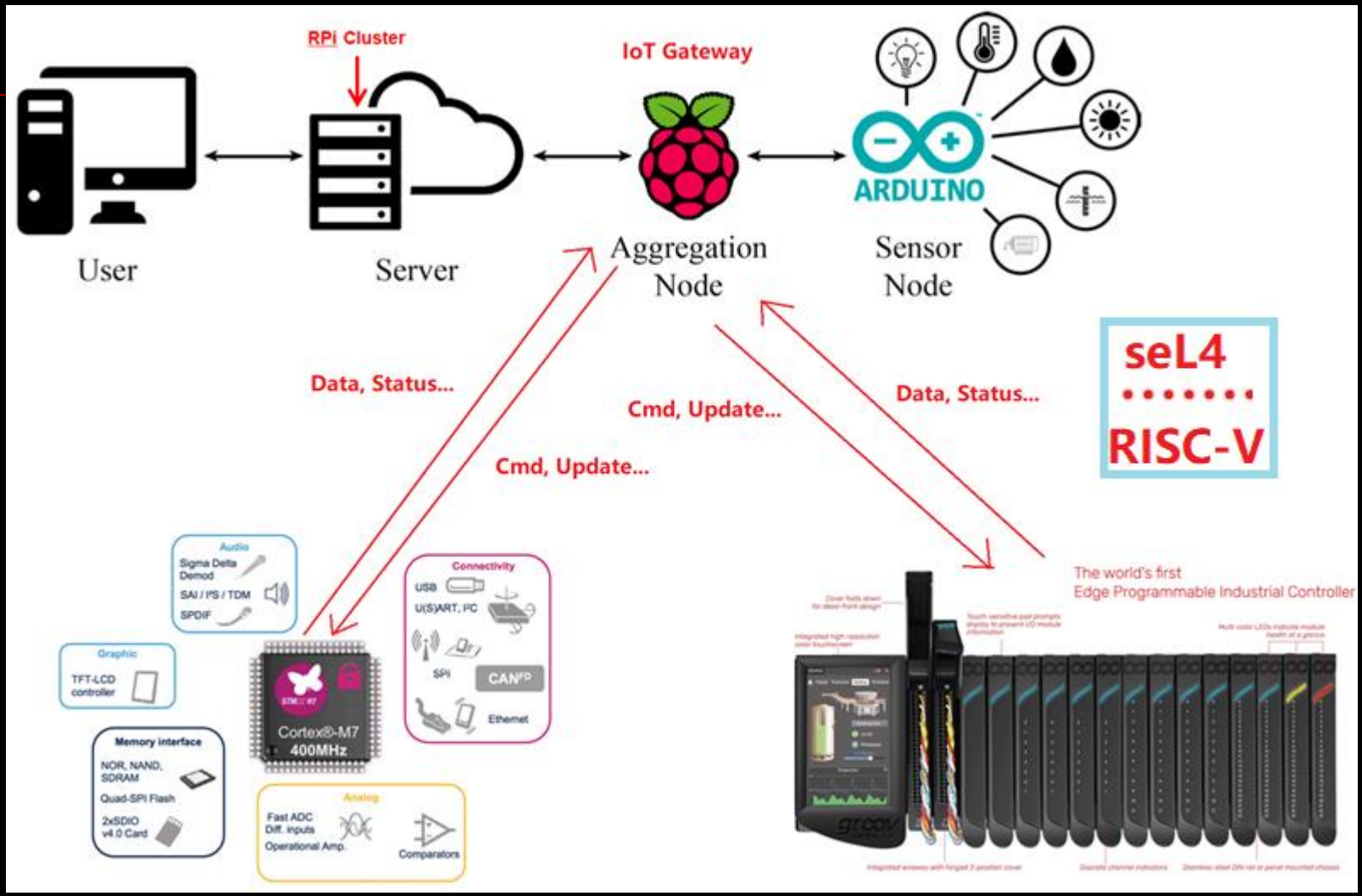
  customize lightweight solution for Wasm-specific workload.
- Serverless Architecture: GaaS(GraalVM as a Service) & SLEdge

  "AOT compilation based Wasm compiler and runtime for Serverless Edge computing" at OpenInfra Days China 2021(Beijing) and upcoming follow-ups "GraalVM for Heterogeneous Parallel Computing", "Revisiting AOT compilation based Wasm compiler and runtime for Serverless Edge computing"…
- MicroVM: Firecracker on ARM.
- Unikernel on seL4 for Serverless

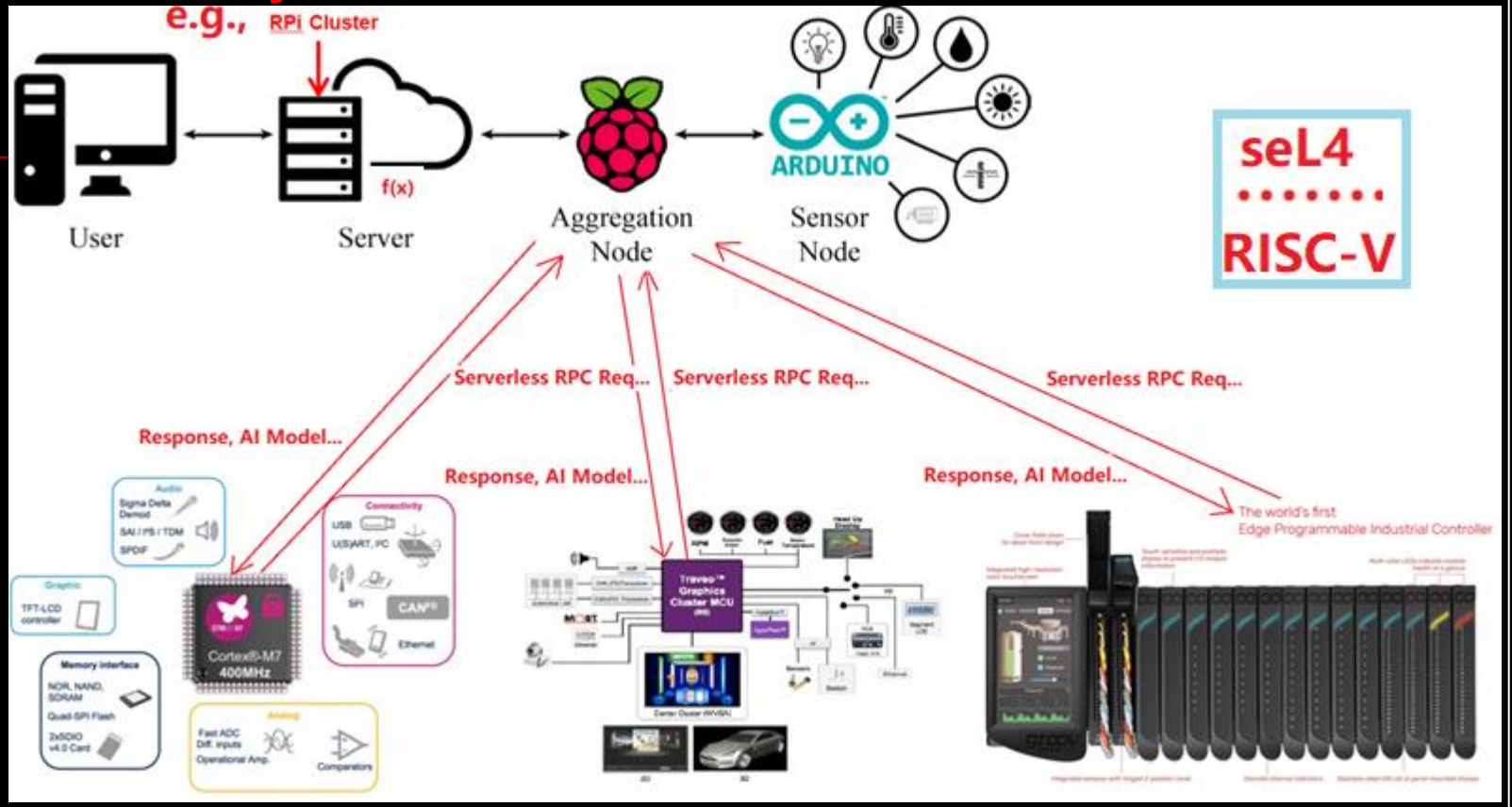
- OS: customized Linux distribution for IoT GW, Server…
  seL4 for IoT Node, Automotive…
- Evaluating RISC-V with seL4.
- HW ACC: RISC-V based SmartNIC for Edge.
- Please refer to some of the details in "Rethinking Hyper-Converged Infrastructure for Edge Computing" at OpenInfra Days China 2019(Shanghai) & "Revisiting the eBPF-centric new approach for Hyper-Converged Infrastructure & Edge Computing" at K+ Summit(Nov 20).
- …
- There is still a long way to go to perfect seL4 on ARM…

Usage scenarios

- General case

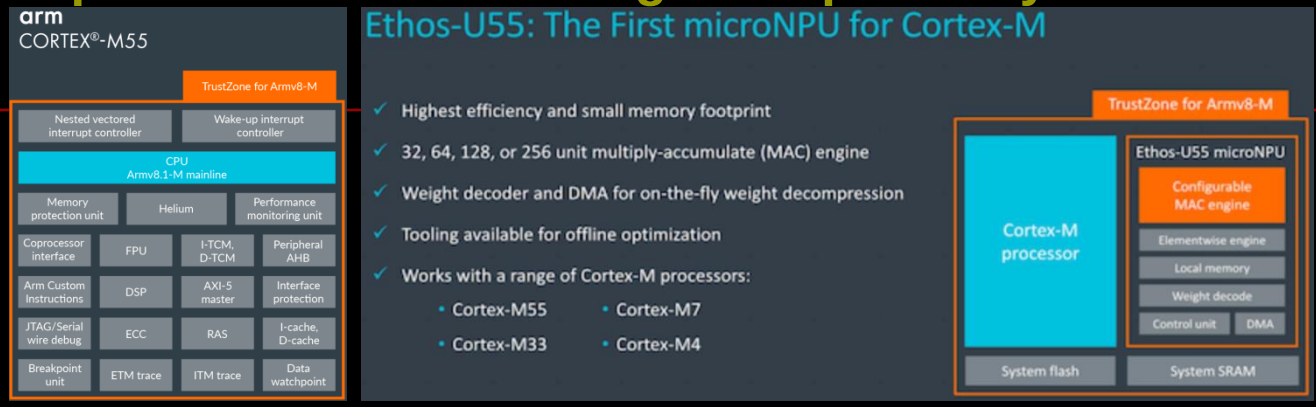


Case of TinyML

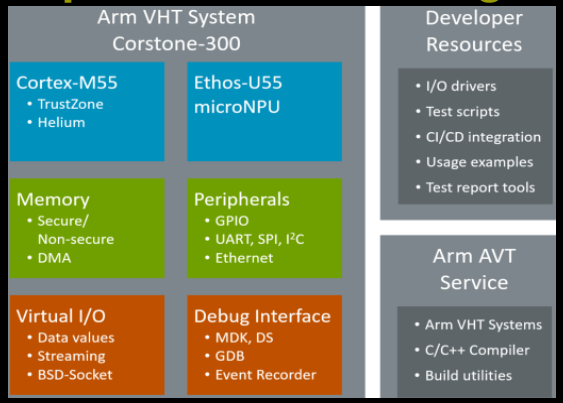


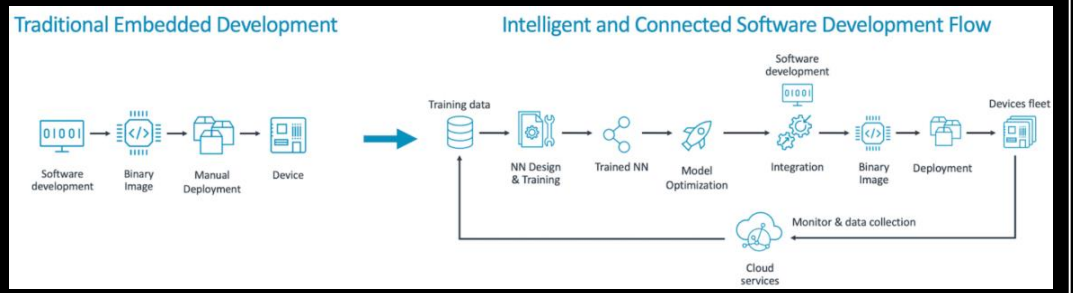
ARM Virtual Hardware for IoT

- https://www.arm.com/blogs/blueprint/tinyml



- https://arm-software.github.io/VHT/main/overview/html/index.html





- https://community.arm.com/arm-community-blogs/b/internet-of-things-blog/posts/introducing-arm-virtual-hardware-for-cloud-based-iot-development
- https://www.arm.com/en/products/development-tools/simulation/virtual-hardware
- https://devsummit.arm.com/en/sessions/539

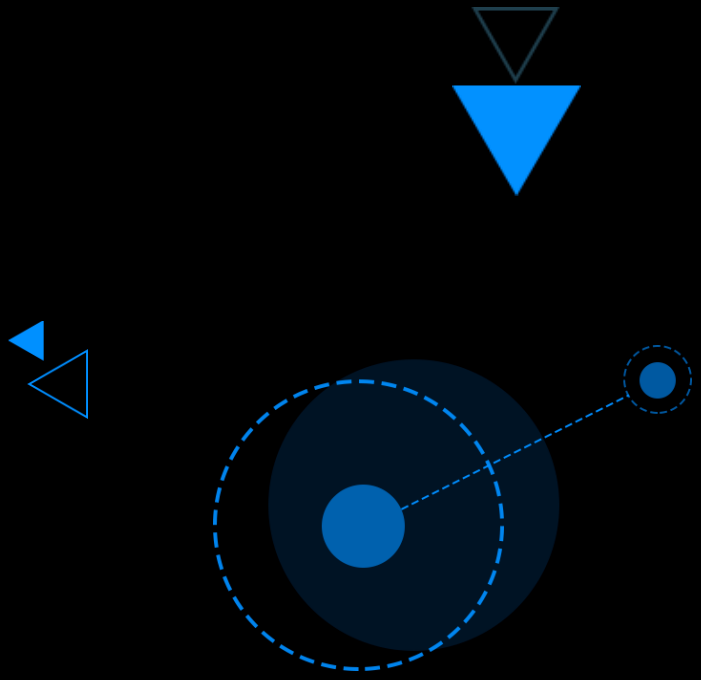
VIII. Wrap-up

Add Wasm here!

VM Cloud 1.0 → Container Cloud 2.0 → LightVM/MicroVM Cloud 2.5 → Unikernel Cloud 3.0?

- Serverless architecture is a good fit for IoT Edge.
- Security is the first citizen of the Edge Cloud world.
- seL4 will play a more and more important role in tomorrow's privacy computing.
- It's time to think seriously about how to run Wasm workload.
- We're embracing the ecosystem of Rust for Cloud Native.
- What shall we think about "beyond Kubernetes"!?

Thanks_



Reference

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/
- https://forge.etsi.org/
- https://arstechnica.com/gadgets/2021/03/arm-vision-day-outlined-upcoming-arm-v9-cpu-design/
- …