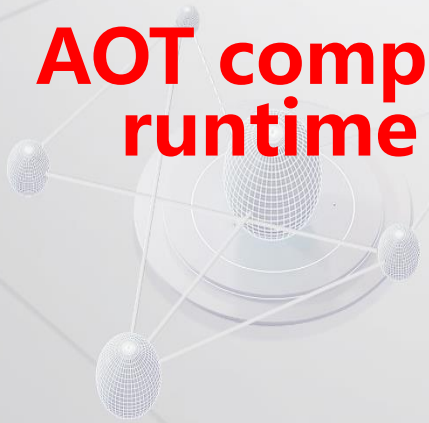
AOT compilation based Wasm compiler and runtime for Serverless Edge computing

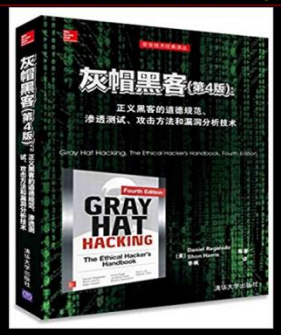


李枫 Koo

独立开发者(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

- Serverless Edge Computing
- LLVM
- WASM
- GraalVM
- Edge AI
- Testbed
- Summary

II. aWsm

- Overview
- aWsm on RPi4

III. SLEdge

- Overview
- SLEdge on RPi4

IV. Rethinking Serverless for IoT Edge

- Why Serverless?
- Usage scenarios
- Why aWsm/SLEdge?
- Possible improvements

V. Wrap-up

I. Tech Stack

1) Serverless Edge Computing

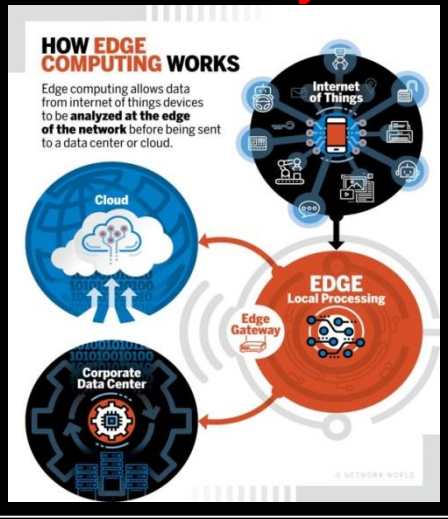
1.1 Edge Computing

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

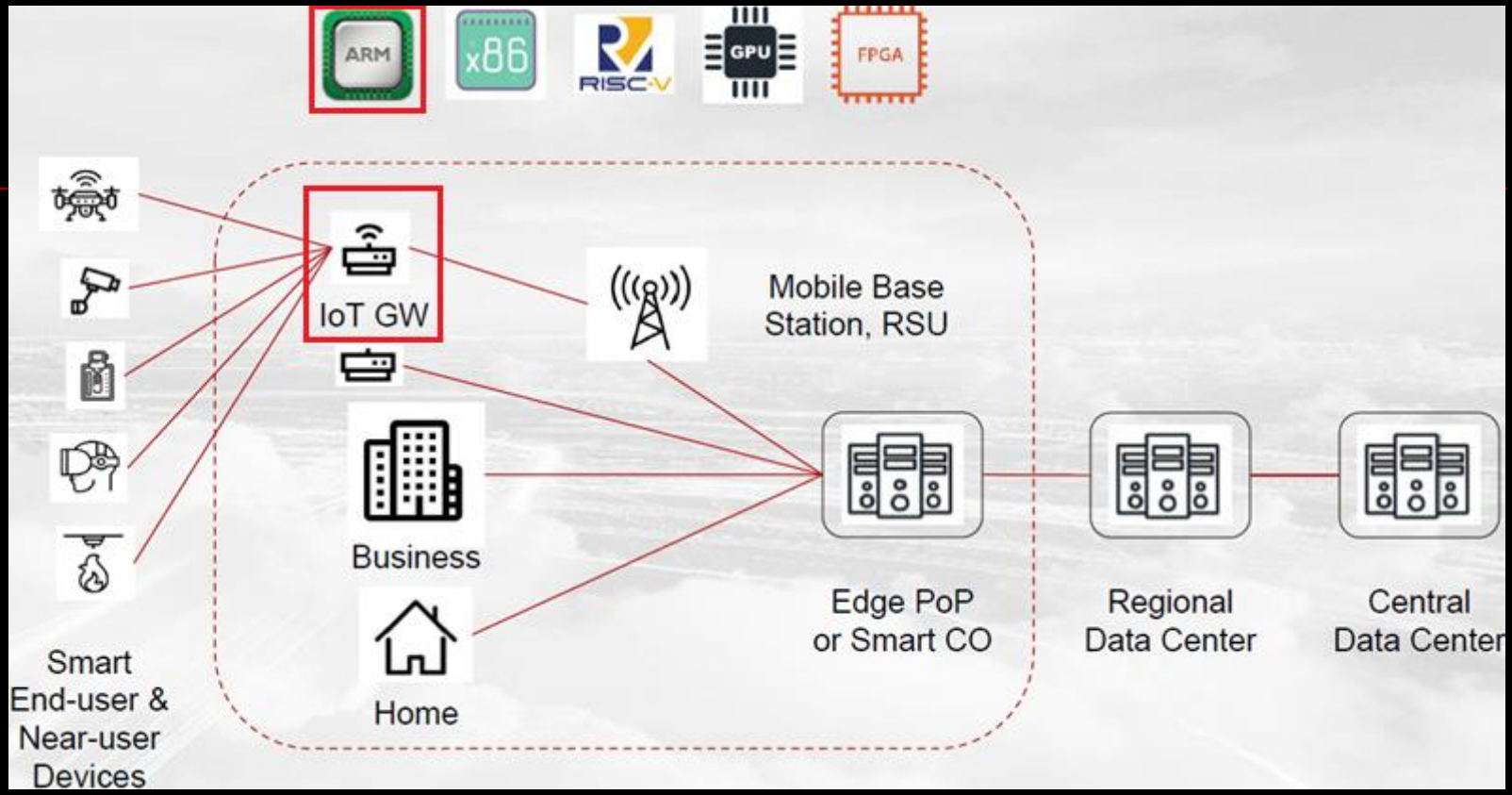
  a distributed computing paradigm which brings computation and data storage closer to the location where it is needed, to improve response times and save bandwidth.…

- https://www.networkworld.com/article/3224893/what-is-edge-computing-and-how-it-s-changing-the-network.html

  a way to streamline the flow of traffic from IoT devices and provide real-time local data analysis…

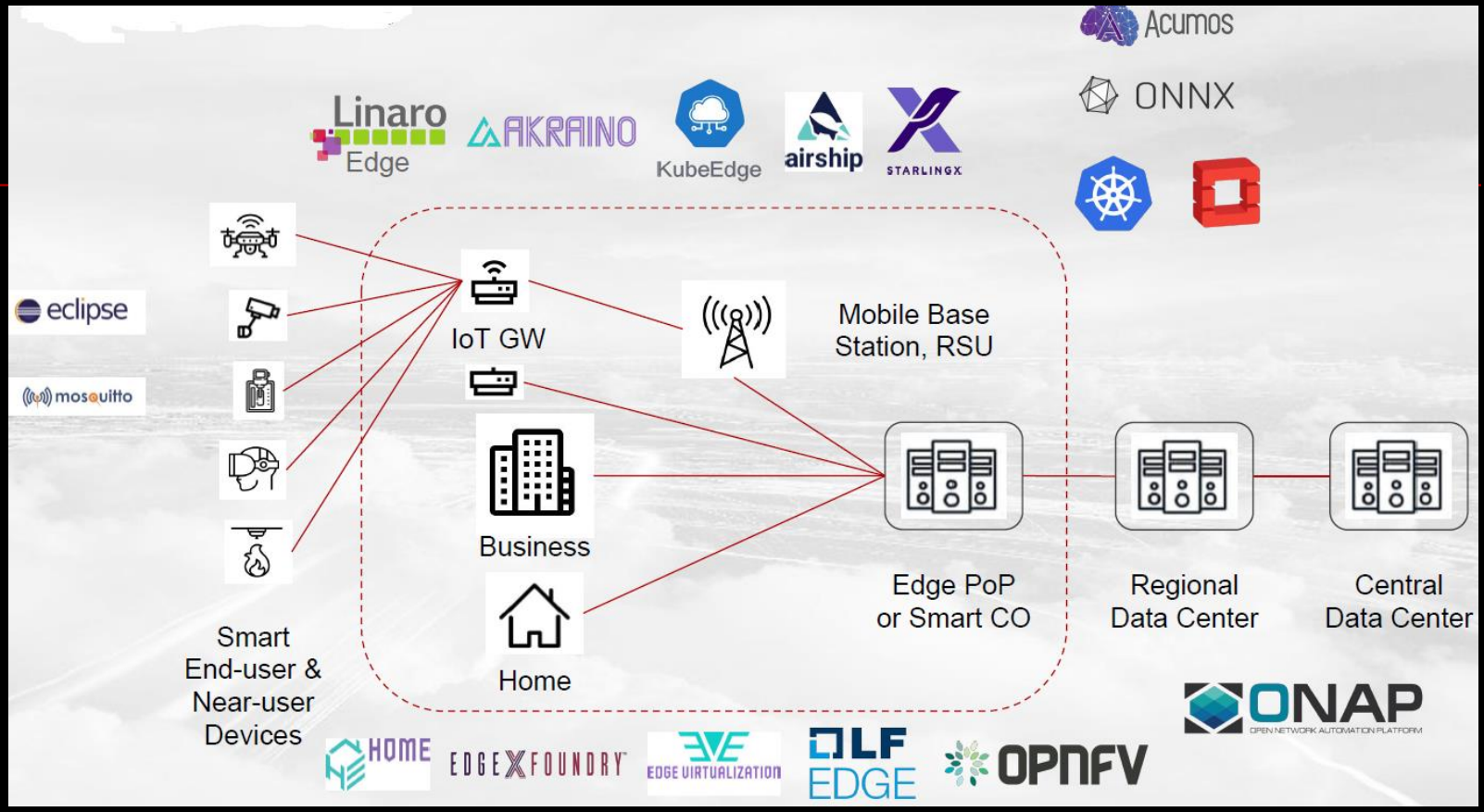


Hardware



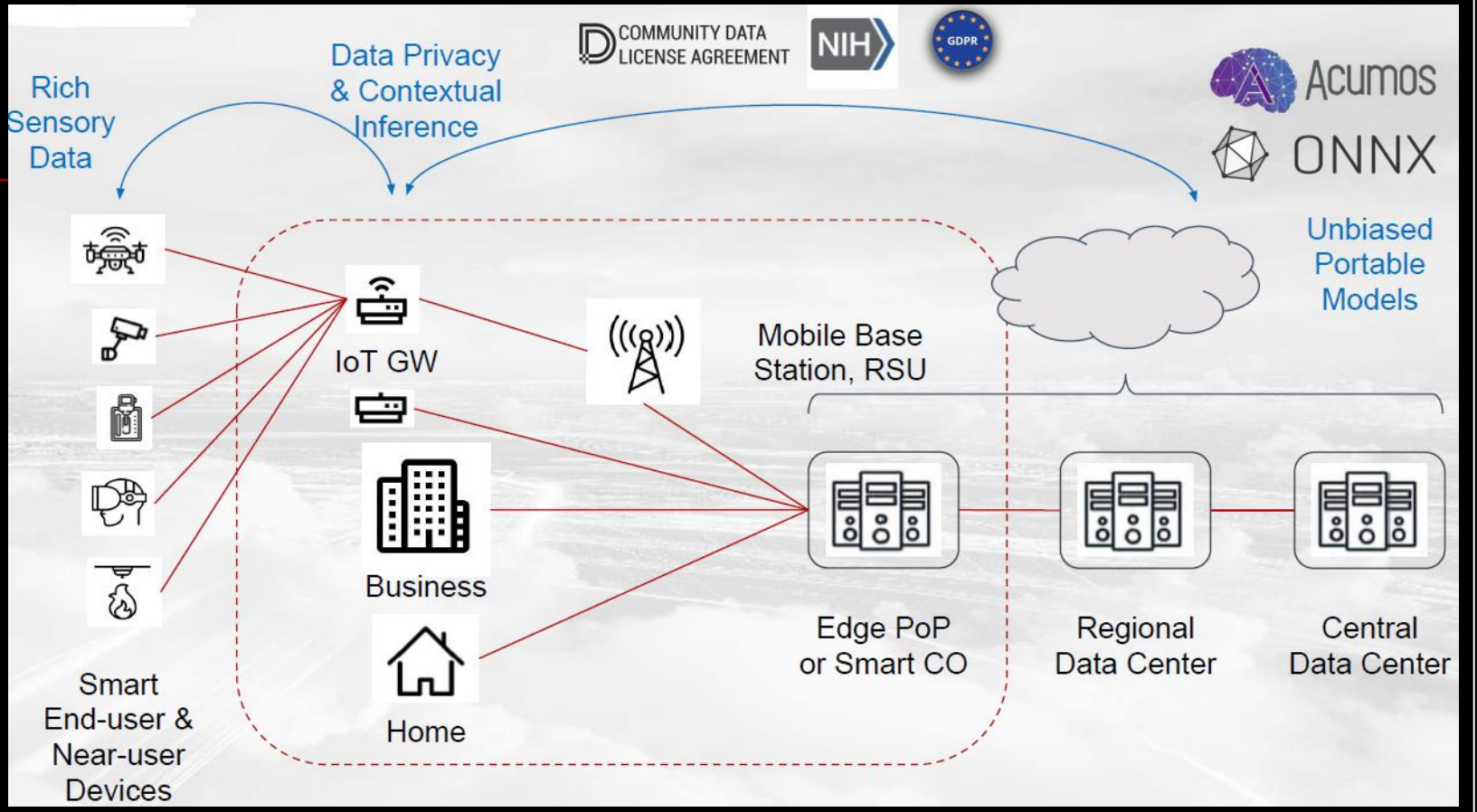
Source: "Deploying the New Intelligent Edge with Open Hardware, Software and Data", Wenjing Chu, ONS North America 2019

Software



Source: "Deploying the New Intelligent Edge with Open Hardware, Software and Data", Wenjing Chu, ONS North America 2019

Data



Source: "Deploying the New Intelligent Edge with Open Hardware, Software and Data", Wenjing Chu, ONS North America 2019

1.2 Serverless Computing

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

Serverless computing is a cloud computing execution model in which the cloud provider allocates machine resources on demand, taking care of the servers on behalf of their customers. Serverless computing does not hold resources in volatile memory; computing is rather done in short bursts with the results persisted to storage. When an app is not in use, there are no computing resources allocated to the app. Pricing is based on the actual amount of resources consumed by an application.[1] It can be a form of utility computing. "Serverless" is a misnomer in the sense that servers are still used by cloud service providers to execute code for developers. However, developers of serverless applications are not concerned with capacity planning, configuration, management, maintenance, fault tolerance, or scaling of containers, VMs, or physical servers.

Serverless computing can simplify the process of deploying code into production. Serverless code can be used in conjunction with code deployed in traditional styles, such as microservices or monoliths. Alternatively, applications can be written to be purely serverless and use no provisioned servers at all.[2] This should not be confused with computing or networking models that do not require an actual server to function, such as peer-to-peer (P2P).

Serverless runtimes

Serverless vendors offer compute runtimes, also known as Function as a Service (FaaS) platforms, which execute application logic but do not store data. Common languages supported by serverless runtimes are Java, Python and PHP. Generally, the functions run under isolation boundaries, such as, Linux containers.

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

Pros

Cost [edit]

Serverless can be more cost-effective than renting or purchasing a fixed quantity of servers,[18] which generally involves significant periods of underutilization or idle time.[1] It can even be more cost-efficient than provisioning an autoscaling group, due to more efficient bin-packing of the underlying machine resources.

This can be described as pay-as-you-go computing[18] or bare-code[18] as you are charged based solely upon the time and memory allocated to run your code; without associated fees for idle time.[18]

Immediate cost benefits are related to the lack of operating costs, including: licenses, installation, dependencies, and personnel cost for maintenance, support, or patching.[18] The lack of personnel cost is an advantage that applies broadly to cloud computing.

Elasticity versus scalability [edit]

See also: Scalability and Elasticity (cloud computing)

In addition, a serverless architecture means that developers and operators do not need to spend time setting up and tuning autoscaling policies or systems; the cloud provider is responsible for scaling the capacity to the demand.[1][12][18] As Google puts it: "from prototype to production to planet-scale."[18]

As cloud native systems inherently scale down as well as up, these systems are known as elastic rather than scalable.

Small teams of developers are able to run code themselves without the dependence upon teams of infrastructure and support engineers; more developers are becoming DevOps skilled and distinctions between being a software developer or hardware engineer are blurring.[18]

Productivity [edit]

With function as a service, the units of code exposed to the outside world are simple event driven functions. This means that typically, the programmer does not have to worry about multithreading or directly handling HTTP requests in their code, simplifying the task of back-end software development.

Cons

Performance [edit]

Infrequently-used serverless code may suffer from greater response latency than code that is continuously running on a dedicated server, virtual machine, or container. This is because, unlike with autoscaling, the cloud provider typically "spins down" the serverless code completely when not in use. This means that if the runtime (for example, the Java runtime) requires a significant amount of time to start up, it will create additional latency.[19]

Resource limits [edit]

Serverless computing is not suited to some computing workloads, such as high-performance computing, because of the resource limits imposed by cloud providers, and also because it would likely be cheaper to bulk-provision the number of servers believed to be required at any given point in time.[20]

Monitoring and debugging [edit]

Diagnosing performance or excessive resource usage problems with serverless code may be more difficult than with traditional server code, because although entire functions can be timed,[2] there is typically no ability to dig into more detail by attaching profilers, debuggers or APM tools.[21] Furthermore, the environment in which the code runs is typically not open source, so its performance characteristics cannot be precisely replicated in a local environment.

Security [edit]

Serverless is sometimes mistakenly considered as more secure than traditional architectures. While this is true to some extent because OS vulnerabilities are taken care of by the cloud provider, the total attack surface is significantly larger as there are many more components to the application compared to traditional architectures and each component is an entry point to the serverless application. Moreover, the security solutions customers used to have to protect their cloud workloads become irrelevant as customers cannot control and install anything on the endpoint and network level such as an intrusion detection/prevention system (IDS/IPS).[22]

This is intensified by the mono-culture properties of the entire server network. (A single flaw can be applied globally.) According to Protego, the "solution to secure serverless apps is close partnership between developers, DevOps, and AppSec, also known as DevSecOps. Find the balance where developers don't own security, but they aren't absolved from responsibility either. Take steps to make it everyone's problem. Create cross-functional teams and work towards tight integration between security specialists and development teams. Collaborate so your organization can resolve security risks at the speed of serverless."[23]

Privacy [edit]

Many serverless function environments are based on proprietary public cloud environments. Here, some privacy implications have to be considered, such as shared resources and access by external employees. However, serverless computing can also be done on private cloud environment or even on-premises, using for example the Kubernetes platform. This gives companies full control over privacy mechanisms, just as with hosting in traditional server setups.

Standards [edit]

Serverless computing is covered by International Data Center Authority (IDCA) in their Framework AE360.[24] However, the part related to portability can be an issue when moving business logic from one public cloud to another for which the Docker solution was created. Cloud Native Computing Foundation (CNCF) is also working on developing a specification with Oracle.[25]

Vendor lock-in [edit]

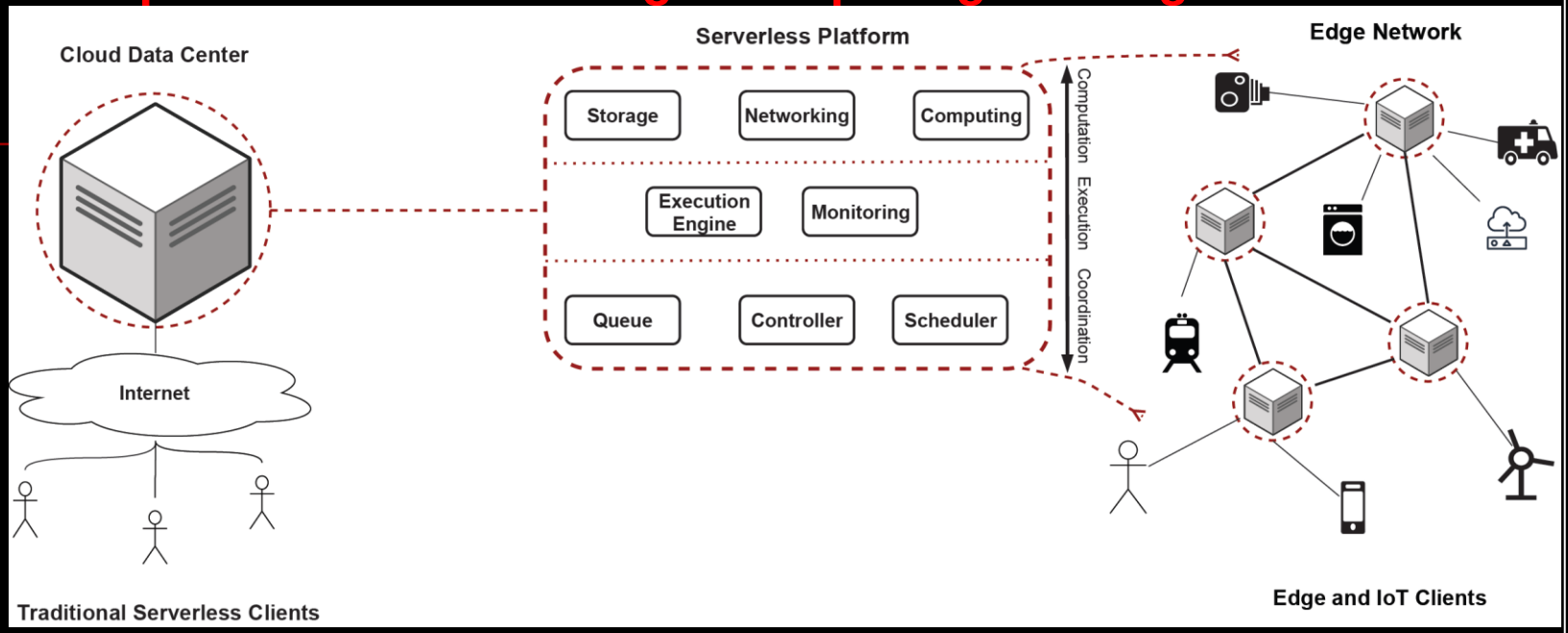
Serverless computing is provided as a third-party service. Applications and software that run in the serverless environment are by default locked to a specific cloud vendor.[26] Therefore, serverless can cause multiple issues during migration.[27]

1.3 Serverless for Edge

- https://medium.com/serverless-transformation/why-serverless-will-enable-the-edge-computing-revolution-4f52f3f8a7b0
- https://www.frontiersin.org/articles/10.3389/frsc.2021.690660/full
- https://www.section.io/blog/containers-vm-serverless-edge-computing/
- …

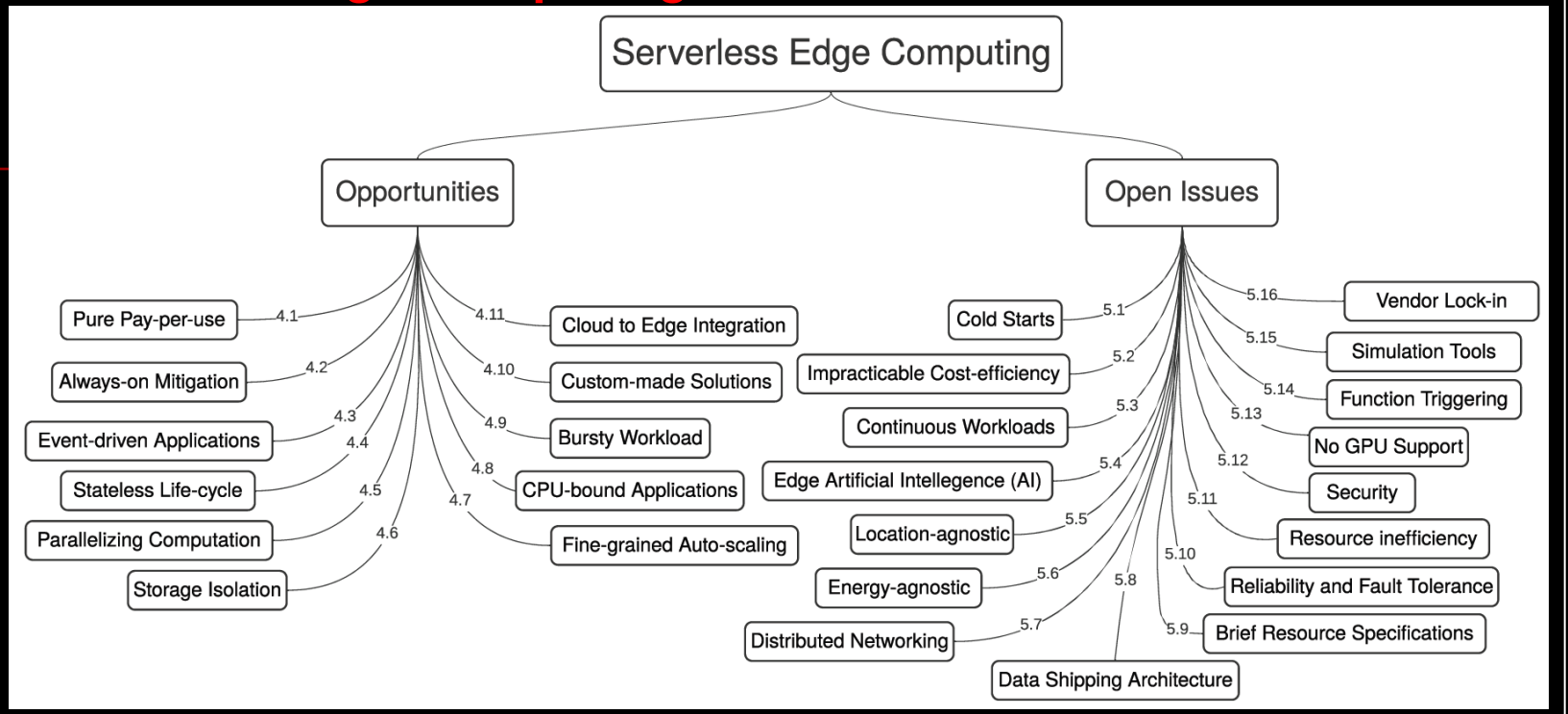
Vision and Challenges

- A simplified Serverless Edge Computing Paradigm



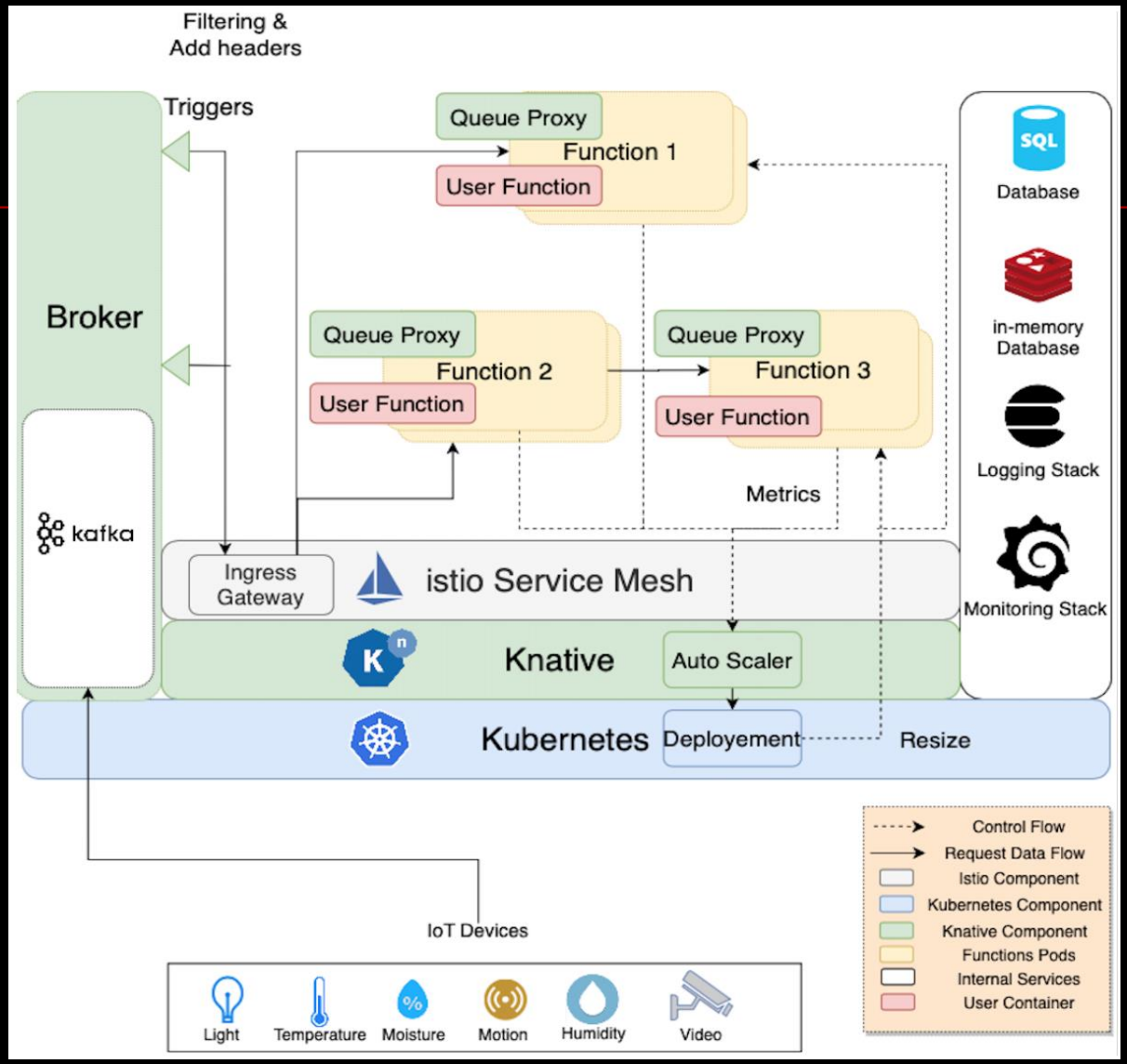
Source: https://dsg.tuwien.ac.at/team/sd/papers/AusPDC_2021_SD_Serverless.pdf

A Taxonomy of identified opportunities and open issues for Serverless Edge Computing



Source: https://dsg.tuwien.ac.at/team/sd/papers/AusPDC_2021_SD_Serverless.pdf

A sample of Serverless architecture for IoT Service



Source: https://cloudnet2020.ieee-cloudnet.org/wp-content/uploads/sites/237/2020/11/SP3-P2.pdf

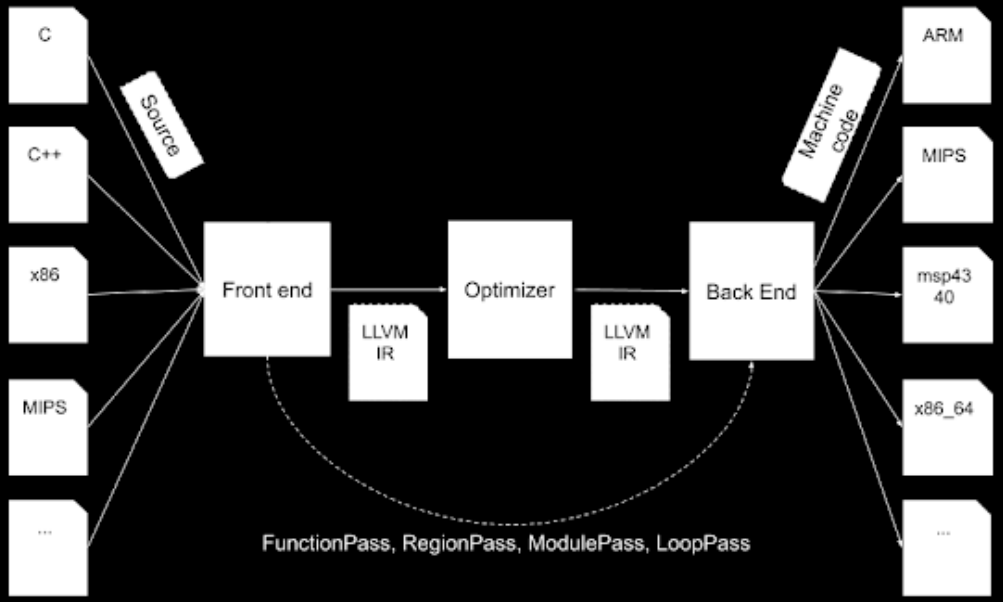
CHINA OpenInfra Days

2) LLVM

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

LLVM is a set of compiler and toolchain technologies,[5] which can be used to develop a front end for any programming language and a back end for any instruction set architecture. LLVM is designed around a language-independent intermediate representation (IR) that serves as a portable, high-level assembly language that can be optimized with a variety of transformations over multiple passes.[6]

- https://llvm.org
- http://clang.llvm.org/
-



Source: http://blog.k3170makan.com/2020/04/learning-llvm-i-introdution-to-llvm.html

- https://llvm.org/docs/
- https://www.llvm.org/ProjectsWithLLVM/

IR

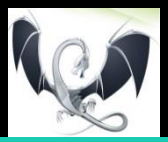
The core of LLVM is the intermediate representation (IR), a low-level programming language similar to assembly. IR is a strongly typed reduced instruction set computing (RISC) instruction set which abstracts away most details of the target. For example, the calling convention is abstracted through *call* and *ret* instructions with explicit arguments. Also, instead of a fixed set of registers, IR uses an infinite set of temporaries of the form %0, %1, etc. LLVM supports three equivalent forms of IR: a human-readable assembly format, an in-memory format suitable for frontends, and a dense bitcode format for serializing. A simple "Hello, world!" program in the IR format:[32]

```
@.str = internal constant [14 x i8] c"hello, world\0A\00"

declare i32 @printf(i8*, ...)

define i32 @main(i32 %argc, i8** %argv) nounwind {
entry:
    %tmp1 = getelementptr [14 x i8], [14 x i8]* @.str, i32 0, i32 0
    %tmp2 = call i32 (i8*, ...) @printf( i8* %tmp1 ) nounwind
    ret i32 0
}
```

- https://llvm.org/docs/LangRef.html
- MLIR:
  https://mlir.llvm.org/
  A hybrid IR which can support multiple different requirements in a unified infrastructure.

- Bitcode:
  https://llvm.org/docs/BitCodeFormat.html
  https://llvm.org/docs/CommandGuide/llvm-bcanalyzer.html
  https://zhuanlan.zhihu.com/p/308201373

LLVM vs GCC



GPL v3	UIUC, MIT
Front-end: CC1 / CPP	Front-end: Clang
ld.bfd / ld.gold	lld / mclinker
gdb	lldb
as / objdump	MC layer
glibc	llvm-libc?
libstdc++	libc++
libsupc++	libc++abi
libgcc	libcompiler-rt
libgccjit	libLLVMMCJIT
. . .	ORC JIT, Coroutines, Clangd, libclc, Falcon...

- http://lld.llvm.org/
- https://llvm.org/docs/Proposals/LLVMLibC.html

3) WASM

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

C source code and corresponding WebAssembly

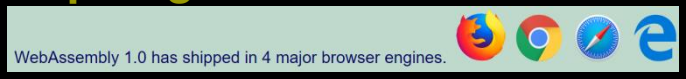
C source code	WebAssembly `.wat` text format	WebAssembly `.wasm` binary format

`int factorial(int n) {`
`  if (n == 0)`
`  return 1;`
`  else`
`  return n * factorial(n-1);`
`}`
	`(func (param i64) (result i64)`
	`  local.get 0`
	`  i64.eqz`
	`  if (result i64)`
	`  i64.const 1`
	`  else`
	`  local.get 0`
	`  local.get 0`
	`  i64.const 1`
	`  i64.sub`
	`  call 0`
	`  i64.mul`
	`  end)`
		`00 61 73 6D 01 00 00 00`
		`01 00 01 60 01 73 01 73 06`
		`03 00 01 00 02`
		`0A 00 01`
		`00 00`
		`20 00`
		`50`
		`04 7E`
		`42 01`
		`05`
		`20 00`
		`20 00`
		`42 01`
		`7D`
		`10 00`
		`7E`
		`0B`
		`0B 15 17`

- https://webassembly.org/

  WebAssembly(abbr. *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.

- https://github.com/WebAssembly/design

- 

- https://webassembly.org/specs/

Limitations

-

1. In general, WebAssembly does not allow direct interaction with the DOM. All interaction must flow through JavaScript interop.
2. Multithreading (although there are plans to address this.)
3. Garbage collection (although there are plans to address this.)
4. Security considerations (discussed below)

WebAssembly is supported on desktops, and mobile, but on the latter, in practice (for non-small memory allocations, such as with Unity game engine) there are "grave limitations that make many applications infeasible to be *reliably* deployed on mobile browsers [..] Currently allocating more than ~300MB of memory is not reliable on Chrome on Android without resorting to Chrome-specific workarounds, nor in Safari on iOS."[62]

All major web browsers allow WebAssembly if Content-Security-Policy is not specified, or if "unsafe-eval" is used, but otherwise the major web browsers behave differently.[63] In practice WebAssembly can't be used on Chrome without "unsafe-eval",[64][65] while a worker thread workaround is available.[66]

Runtime Implementations

- While WebAssembly was initially designed to enable near-native code execution speed in the web browser, it has been considered valuable outside of such, in more generalized contexts.[37][38] Since WebAssembly's runtime environments (RE) are low level virtual stack machines (akin to JVM or Flash VM) that can be embedded into host applications, some of them have found a way to standalone runtime environments like Wasmtime and Wasmer.[8][13]

  Web browsers [edit]

  In November 2017, Mozilla declared support "in all major browsers",[39] after WebAssembly was enabled by default in Edge 16.[40] The support includes mobile web browsers for iOS and Android. As of July 2021, 94% of installed browsers support WebAssembly.[41] But for older browsers, Wasm can be compiled into asm.js by a JavaScript polyfill.[42]

Compilers:

Because WebAssembly executables are precompiled, it is possible to use a variety of programming languages to make them.[43] This is achieved either through direct compilation to Wasm, or through implementation of the corresponding virtual machines in Wasm. There have been around 40 programming languages reported to support Wasm as a compilation target.[44]

Emscripten compiles C and C++ to Wasm[32] using the Binaryen and LLVM as backend.[45]

As of version 8, a standalone Clang can compile C and C++ to Wasm.[46]

Its initial aim is to support compilation from C and C++,[47] though support for other source languages such as Rust, .NET languages[48][49][44] and AssemblyScript[50] (TypeScript-like) is also emerging. After the MVP release, there are plans to support multithreading and garbage collection[51][52] which would make WebAssembly a compilation target for garbage-collected programming languages like C# (supported via Blazor), F# (supported via Bolero[53] with help of Blazor), Python, and even JavaScript where the browser's just-in-time compilation speed is considered too slow. A number of other languages have some support including Python,[54] Java,[55] Julia,[56][57][58] Zig,[59] and Ruby,[60] as well as Go.[61]

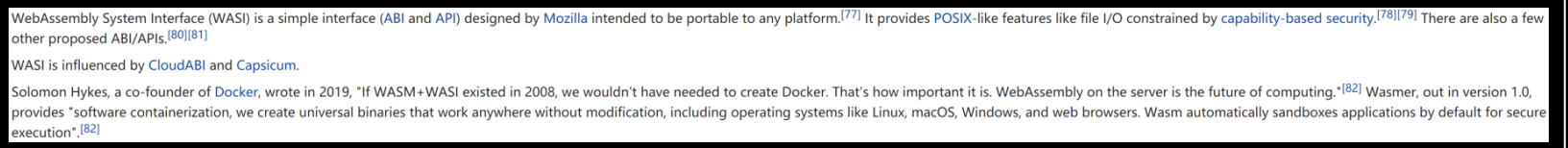
- https://github.com/appcypher/awesome-wasm-langs
- https://github.com/appcypher/awesome-wasm-runtimes

- Wasm & Rust
  https://www.rust-lang.org/what/wasm
  https://rustwasm.github.io/book
  https://github.com/rustwasm

Beyond the browser

- https://webassembly.org/docs/non-web/
- https://www.zdnet.com/article/microsoft-google-back-bytecode-alliance-to-move-webassembly-beyond-the-browser/

- **WASI (WebAssembly System Interface)**



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]

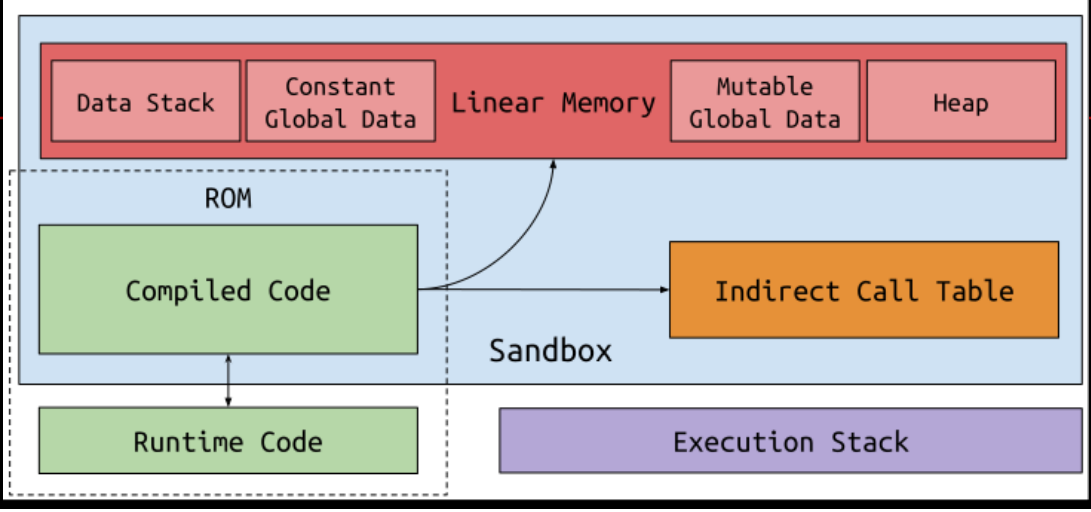
WASI is influenced by CloudABI and Capsicum.

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://github.com/WebAssembly/WASI
https://github.com/bytecodealliance/wasmtime/blob/main/docs/WASI-documents.md
https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/
https://training.linuxfoundation.org/announcements/wasi-bringing-webassembly-way-beyond-browsers

1.1 Runtime

Sandboxing



Wasm uses a co-design between the compiler, and the dynamic checks of the runtime system to provide the sandbox that isolates the surrounding system from the logic of the contained code. The figure depicts the main aspects of the sandbox. These include:

- *Linear memory* that holds all memory accessed by the sandbox. The compiler emits code that checks that all loads and stores remain within the linear memory, thus preventing errant accesses outside the sandbox. Linear memory is expandable much like a traditional heap.
- The *indirect function call table* that facilitates function pointer calls. To ensure that function pointer invocations are safe (to code generated by the compiler), function pointers reference an *offset* into the table. Each entry includes the type of the function, and ensures that function invocations are well-typed.
- The separation of the *execution stack* -- used to track function calls -- and the *data stack* -- used to contain stack-allocated data that can be referenced, thus must be in linear memory.

The first of these ensures the proper memory isolation of the sandbox, while the latter two provide control-flow integrity of the sandbox.

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

Lightweight Virtualization

- **Nanoprocesses**

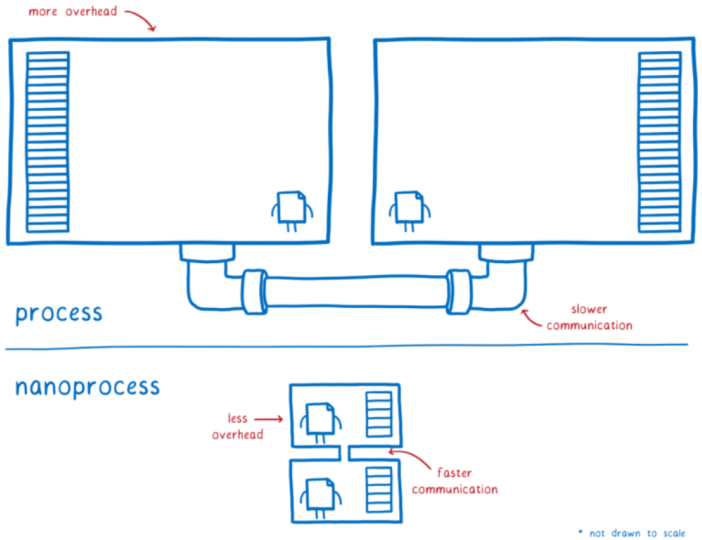
Tomorrow's solution: WebAssembly "nanoprocesses"

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.

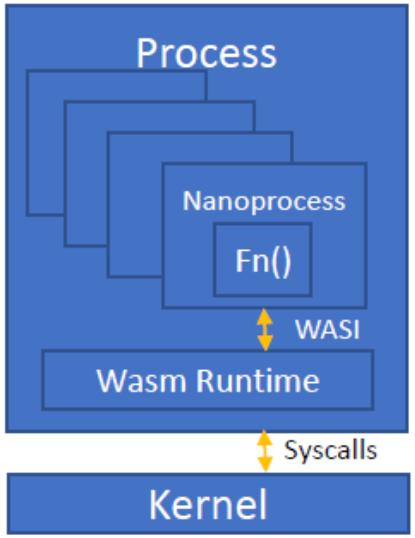


Source: https://hacks.mozilla.org/2019/11/announcing-the-bytecode-alliance/

Architecture

High-level reference architecture for running multiple WebAssembly sandboxes within a single native OS process. A userspace runtime schedules sandbox execution and provides system services.

Processes



Nanoprocesses

Source: https://conix.io/wp-content/uploads/pubs/3878/CONIX-Sledge-Poster.pdf

1.2 WASM & LLVM

LLVM-based toolchain for WASM

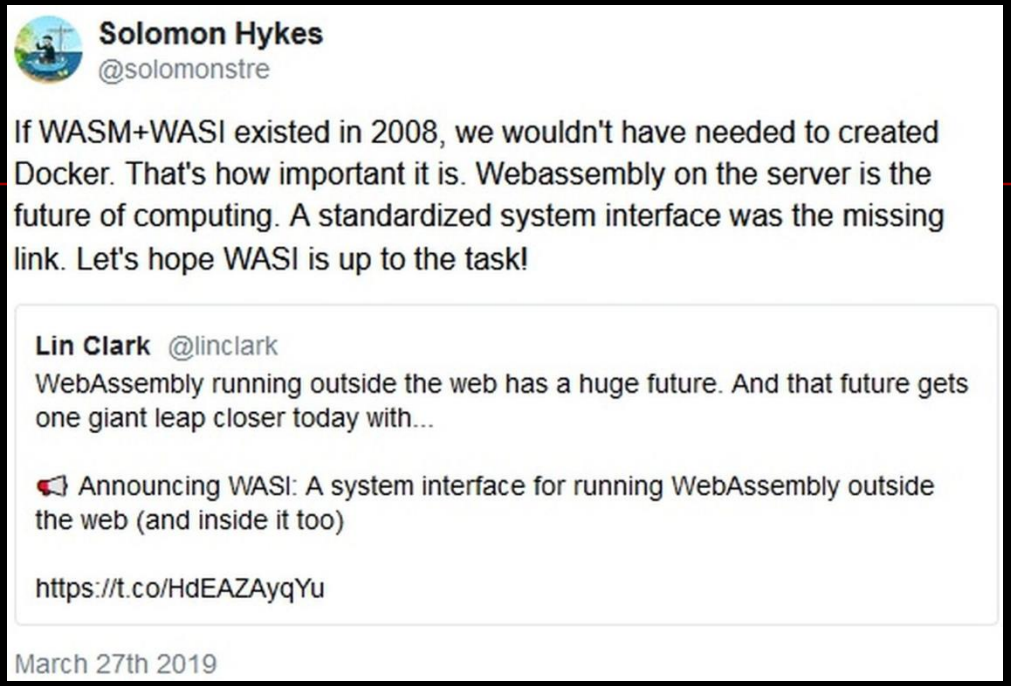
- https://emscripten.org/
- https://wavm.github.io/
- https://github.com/WebAssembly/wasi-sdk
- ...

LLVM WebAssembly backend

- since LLVM version 8
  https://releases.llvm.org/8.0.0/docs/ReleaseNotes.html#changes-to-the-webassembly-target
- $SRC_LLVM/lib/Target

```
[mydev@fedora llvm]$ tree -L 1 lib/Target
lib/Target
├── AArch64
├── AMDGPU
├── ARC
├── ARM
├── AVR
├── BPF
├── CMakeLists.txt
├── CSKY
├── Hexagon
├── Lanai
├── M68k
├── Mips
├── MSP430
├── NVPTX
├── PowerPC
├── README.txt
├── RISCV
├── Sparc
├── SystemZ
├── Target.cpp
├── TargetIntrinsicInfo.cpp
├── TargetLoweringObjectFile.cpp
├── TargetMachineC.cpp
├── TargetMachine.cpp
├── VE
├── WebAssembly
├── X86
└── XCore
```

1.3 Replace Docker with WASM?



Solomon Hykes
@solomonstre

If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!

Lin Clark @linclark
WebAssembly running outside the web has a huge future. And that future gets one giant leap closer today with...

📢 Announcing WASI: A system interface for running WebAssembly outside the web (and inside it too)

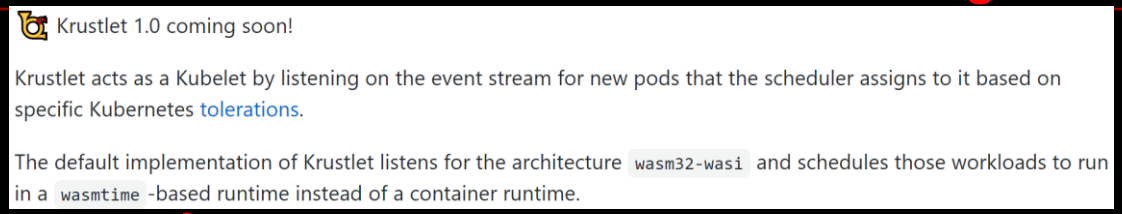
https://t.co/HdEAZAyqYu

March 27th 2019

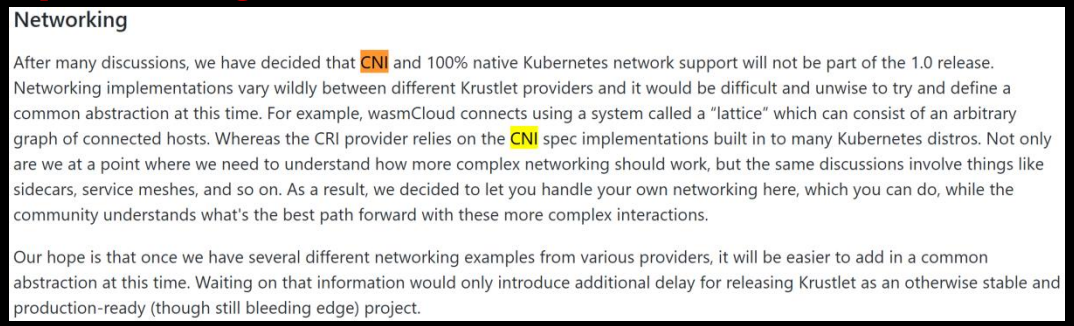
- **Kubernetes is deprecating Docker as a container runtime after v1.20…**
  https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/
  https://cloud.redhat.com/blog/kubernetes-is-removing-docker-support-kubernetes-is-not-removing-docker-support
- **...**

1.3.1 Krustlet

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



- Upcoming v1.0



- For Krustlet on RPi4, you may refer to my previous talk as below:

  "Cloud-Hypervisor on ARM" at the Rust China Meetup (Hangzhou 2021)

- …

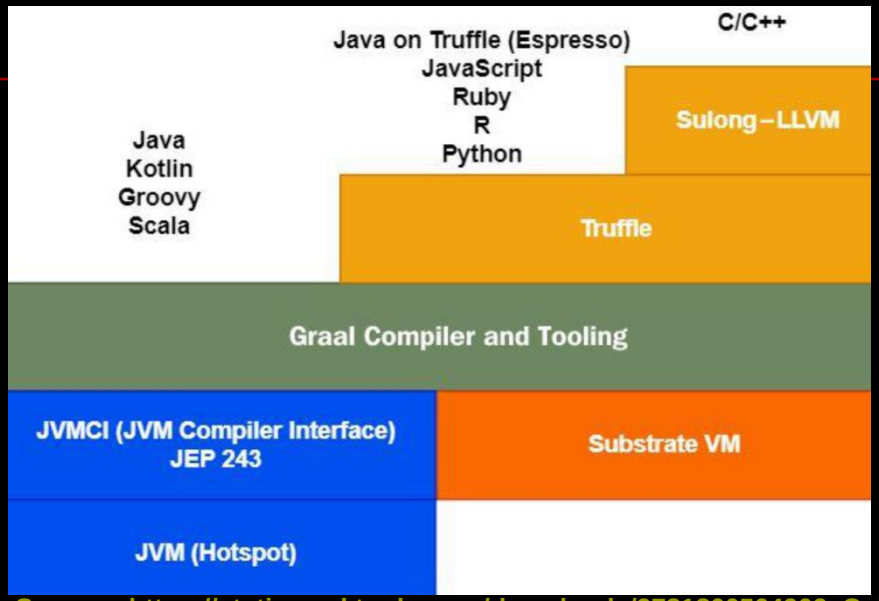
New Approches

- Integrate lightweight Kubenetes distributions with Krustlet
- Even more lighter weight designs

Our new solution will come to the surface before the end of the year…

4) GraalVM

- https://en.wikipedia.org/wiki/GraalVM
- https://www.graalvm.org/
-

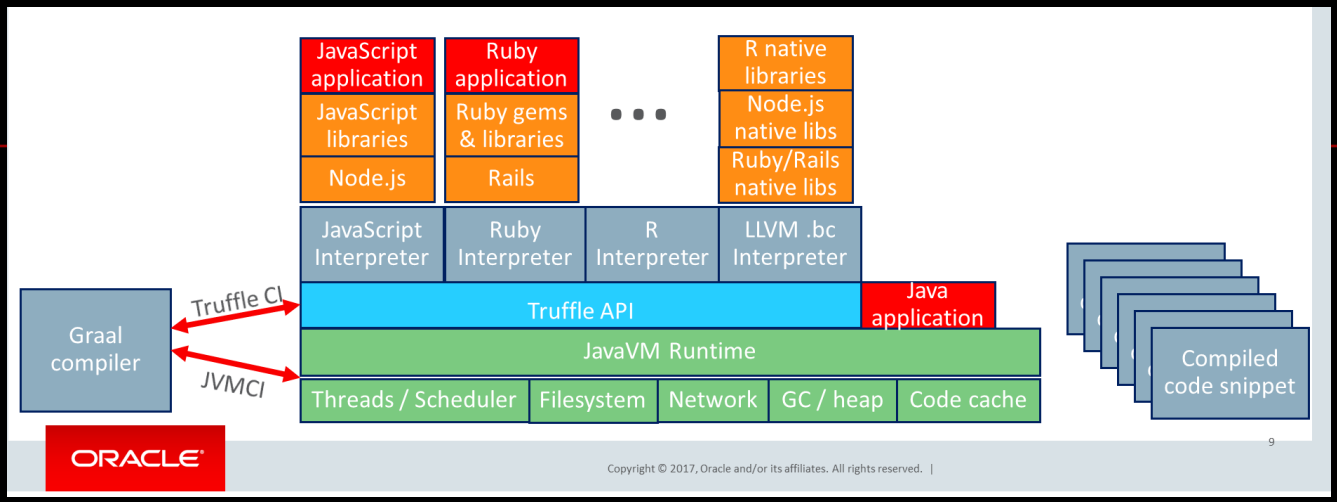


Source: https://static.packt-cdn.com/downloads/9781800564909_ColorImages.pdf

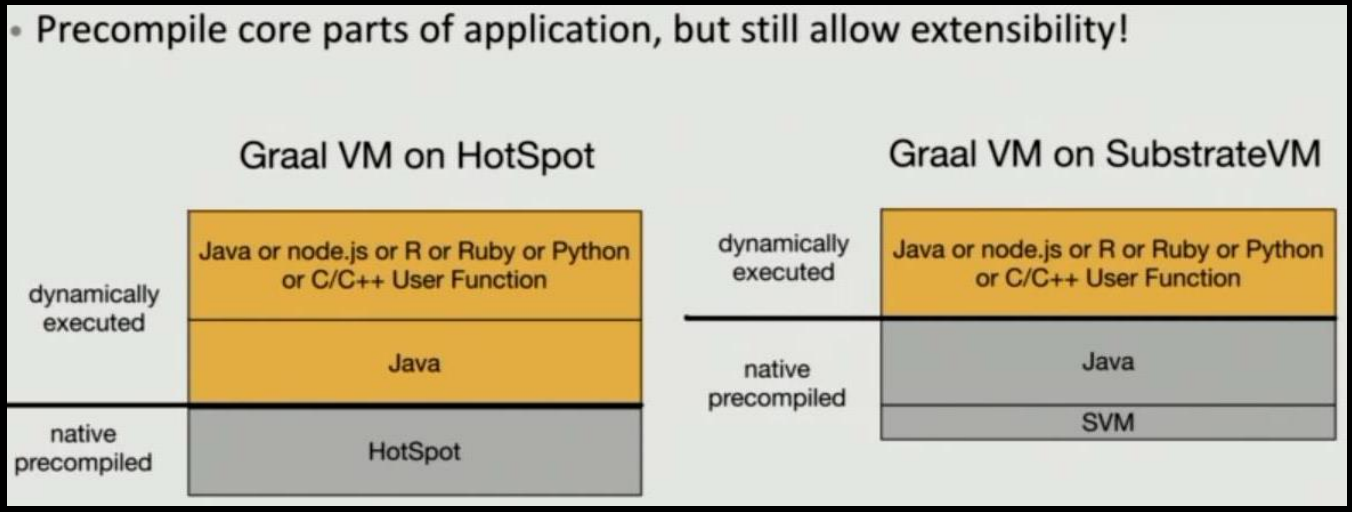
- A Universal High-Performance Polyglot VM
- A meta-runtime for Language-Level Virtualization
- Currently base an Oracle Labs JDK 8, 11, 16 with JVMCI support
- https://www.graalvm.org/docs/introduction/
- https://github.com/oracle/graal/blob/master/docs/

Architecture

- A hybrid of static & dynamic runtimes



Source: https://ics.psu.edu/wp-content/uploads/2017/02/GraalVM-PSU.pptx



Source: "Adopting Java for the Serverless world", Vadym Kazulkin, JUG London 2020.

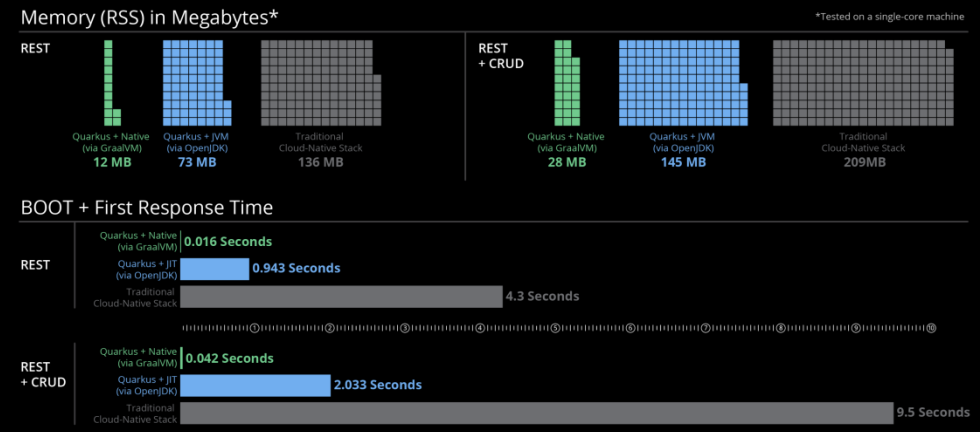
4.1 Quarkus

- https://quarkus.io/

  A Kubernetes Native Java stack tailored for OpenJDK HotSpot and GraalVM, crafted from the best of breed Java libraries and standards.

- **Amazingly fast boot time, incredibly low RSS memory!**

```
$ ./my-native-java-rest-app
Quarkus started in 0.008s
```



- https://github.com/quarkusio/quarkus

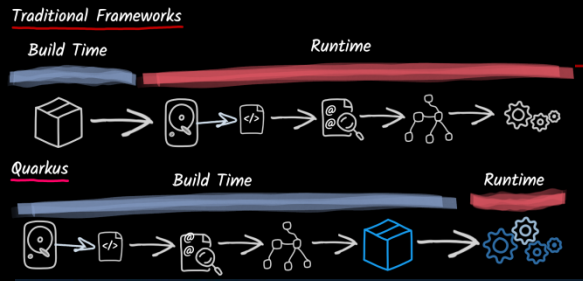
Quarkus is a Cloud Native, (Linux) Container First framework for writing Java applications.

- **Container First**: Minimal footprint Java applications optimal for running in containers.
- **Cloud Native**: Embraces 12 factor architecture in environments like Kubernetes.
- **Unify imperative and reactive**: Brings under one programming model non-blocking and imperative styles of development.
- **Standards-based**: Based on the standards and frameworks you love and use (RESTEasy and JAX-RS, Hibernate ORM and JPA, Netty, Eclipse Vert.x, Eclipse MicroProfile, Apache Camel...).
- **Microservice First**: Brings lightning fast startup time and code turn around to Java apps.
- **Developer Joy**: Development centric experience without compromise to bring your amazing apps to life in no time.

All under ONE framework.

- ## https://quarkus.io/vision/container-first

From the outset, Quarkus has been designed around a container-first philosophy. What this means in concrete terms is that Quarkus applications are optimised for low memory usage and fast startup times in the following ways:



Build Time Processing

The central idea behind Quarkus is to do at build-time what traditional frameworks do at runtime: configuration parsing, classpath scanning, feature toggle based on classloading, and so on.

As much processing as possible is done at build time; thus, your application only contains the classes used at runtime. In traditional frameworks, all the classes required to perform the initial application deployment hang around for the application's life, even though they are only used once. With Quarkus, they are not even loaded into the production JVM! Quarkus does not stop here. During the build-time processing, it prepares the initialization of all components used by your application. It results in less memory usage and faster startup time as all metadata processing has already been done.

Reduction in Reflection Usage

As much as possible Quarkus tries to avoid reflection, reducing startup time and memory usage. During the built-time processing, extensions can analyze the application code and the classes available on the classpath and replace reflection calls with regular invocations. The usage of dynamic proxies is also prevented by using generating custom proxy at build time.

Arc, the dependency injection framework used by Quarkus, eliminates all the reflection calls and deduces the injection graph at build time. So, when the application starts, no expensive lookups; it's done already!

First-Class Support for GraalVM Native Images

GraalVM Native Executable support has been an essential part of the design for Quarkus from the beginning. When an application is compiled down to a native executable, it starts much faster and can run with a much smaller heap than a standard JVM. The native compiler uses aggressive dead-code elimination techniques to only embed the parts of the JVM and classes that are absolutely required by your application. Quarkus makes building optimized native executables plain easy. The build-time approach allows Quarkus to collect enough metadata on your application to fine-tune the compilation. No `-H:+ReportUnsupportedElementsAtRuntime` flag, no fallback, no compromise!

Native Image Pre-Boot

We pre-boot as many of the frameworks as possible during the native compilation of a Quarkus application. It means that the resulting native executable has already run most of the startup code and serialized the result into the executable: even faster startup!

Kubernetes, but also bare metal

All the techniques allowing reducing the memory usage and provide faster startup times are not only advantageous in containers. Even on bare metal, it would reduce your memory pressure, and it's always pleasant to not have to wait 10 seconds to see your application running.

When Quarkus was designed, we didn't focus only on containers but also on deploying Quarkus applications on container orchestrators such as Kubernetes. Quarkus build-time processing also generates the Kubernetes metadata, so your application is ready to be deployed on Kubernetes. Runtime capabilities such as health checks and metrics are exposed out of the box. Quarkus collects all the required metadata at build time to create the Kubernetes deployment descriptor and produce a container image. A single command line can deploy your application onto your Kubernetes cluster.

5) Edge AI

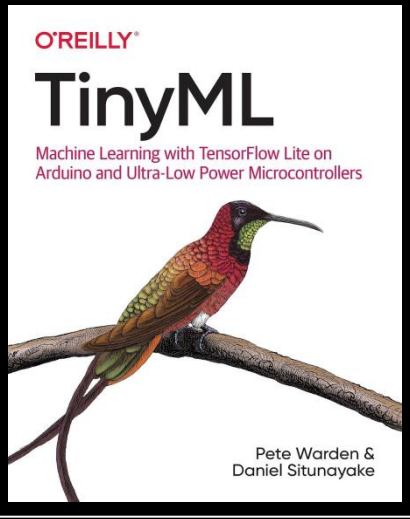
- https://www.itbusinessedge.com/data-center/developments-edge-ai/
- https://www.seeedstudio.com/blog/2021/04/02/edge-ai-what-is-it-and-what-can-it-do-for-edge-iot/
- https://www.vectoritcgroup.com/en/tech-magazine-en/artificial-intelligence-en/edge-ai-el-futuro-de-la-inteligencia-artificial/
- https://towardsdatascience.com/edge-ai-is-the-next-wave-of-ai-a3e98b77c2d7
- https://www.semi.org/en/blogs/semi-news/deploying-artificial-intelligence-at-the-edge-key-takeaways-from-semi-cto-forum

5.1 TinyML

- https://www.tinyml.org/

> Tiny machine learning is broadly defined as a fast growing field of machine learning technologies and applications including hardware, algorithms and software capable of performing on-device sensor data analytics at extremely low power, typically in the mW range and below, and hence enabling a variety of always-on use-cases and targeting battery operated devices.

- https://semiengineering.com/why-tinyml-is-such-a-big-deal/
- TinyML Summit, TinyML Asia, TinyML Research Symposium…
- https://www.meetup.com/en-AU/pro/tinyml/
-



CHINA OpenInfra Days

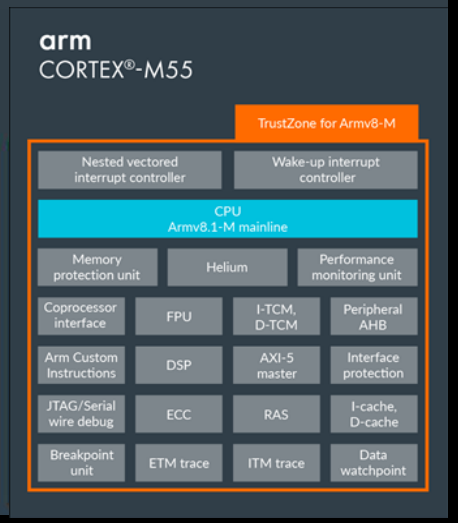
TinyML on ARM

- https://www.arm.com/blogs/blueprint/tinyml
- Cortex-M55
  https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m55

First Helium and Custom Instruction capable CPU core

Cortex-M55: The Most AI-capable Cortex-M Processor

- ✓ First CPU based on Arm Helium technology
  - Energy-efficient and configurable with vector processing capabilities
  - Delivers up to 5x DSP performance and up to 15x ML performance*
  - Versatile capability for both classical ML and NN inference
- ✓ Advanced memory interfaces for fast access to ML data and weights
- ✓ Arm TrustZone security, accelerating the route to PSA Certified



- https://www.arm.com/why-arm/technologies/helium
- https://www.arm.com/why-arm/technologies/custom-instructions
- https://www.zhihu.com/question/371097288/answer/1010694311
- https://www.arm.com/blogs/blueprint/cambridge-consultants

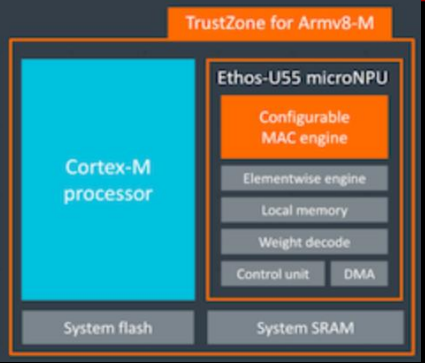
Ethos: MicroNPU for ARM

https://www.arm.com/product-filter?families=ethos%20npus&showall=true

Ethos-U55:

Ethos-U55: The First microNPU for Cortex-M

- ✓ Highest efficiency and small memory footprint
- ✓ 32, 64, 128, or 256 unit multiply-accumulate (MAC) engine
- ✓ Weight decoder and DMA for on-the-fly weight decompression
- ✓ Tooling available for offline optimization
- ✓ Works with a range of Cortex-M processors:
  - Cortex-M55
  - Cortex-M7
  - Cortex-M33
  - Cortex-M4



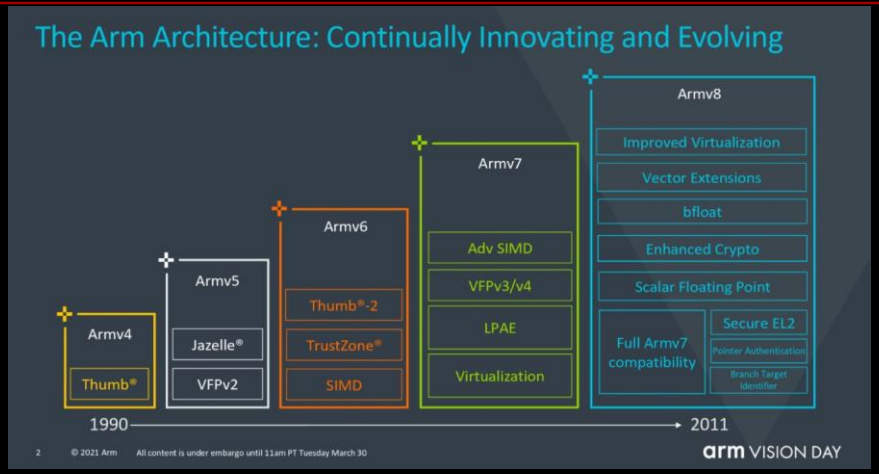
Key Features	Performance (At 1GHz)	64 to 512 GOP/s
	MACs (8x8)	32, 64, 128, 256
	Utilization on popular networks	Up to 85%
	Data Types	Int-8 and Int-16
	Network Support	CNN and RNN/LSTM
	Winograd Support	No
	Sparsity	Yes
Memory System	Internal SRAM	18 to 50 KB
	External On Chip SRAM	KB to Multi-MB
	Compression	Weights only
	Memory Optimizations	Extended compression, layer/operator fusion
Development Platform	Neural Frameworks	TensorFlow Lite Micro
	Operating Systems	RTOS or bare-metal
	Software Components	TensorFlow Lite Micro Runtime, CMSIS-NN, Optimizer, Driver
	Debug and Profile	Layer-by-layer visibility with PMUs
	Evaluation and Early Prototyping	Performance Model, Cycle Accurate Model, or FPGA Evaluations

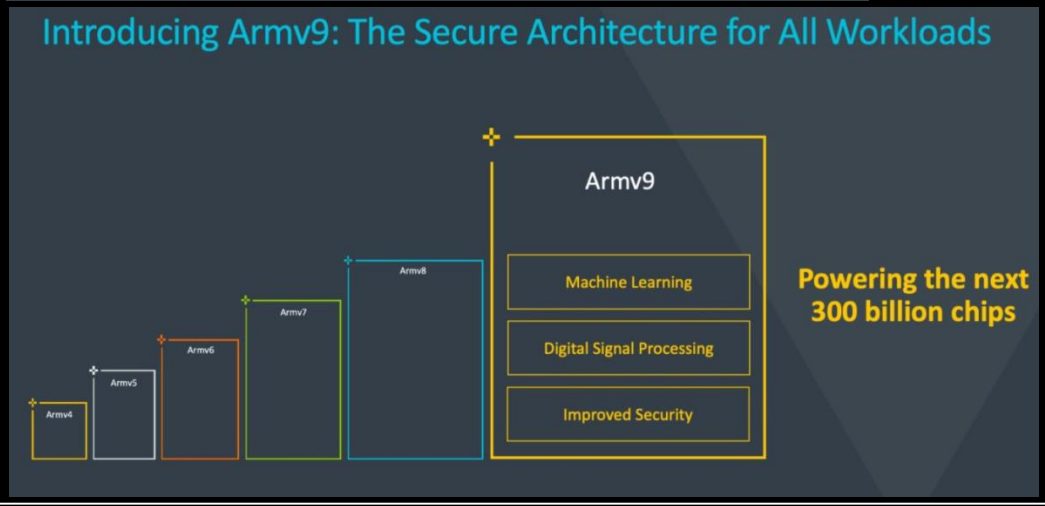
6) Testbed

6.1 ARM Ecosystem for Edge Computing in 2021

ARM v9

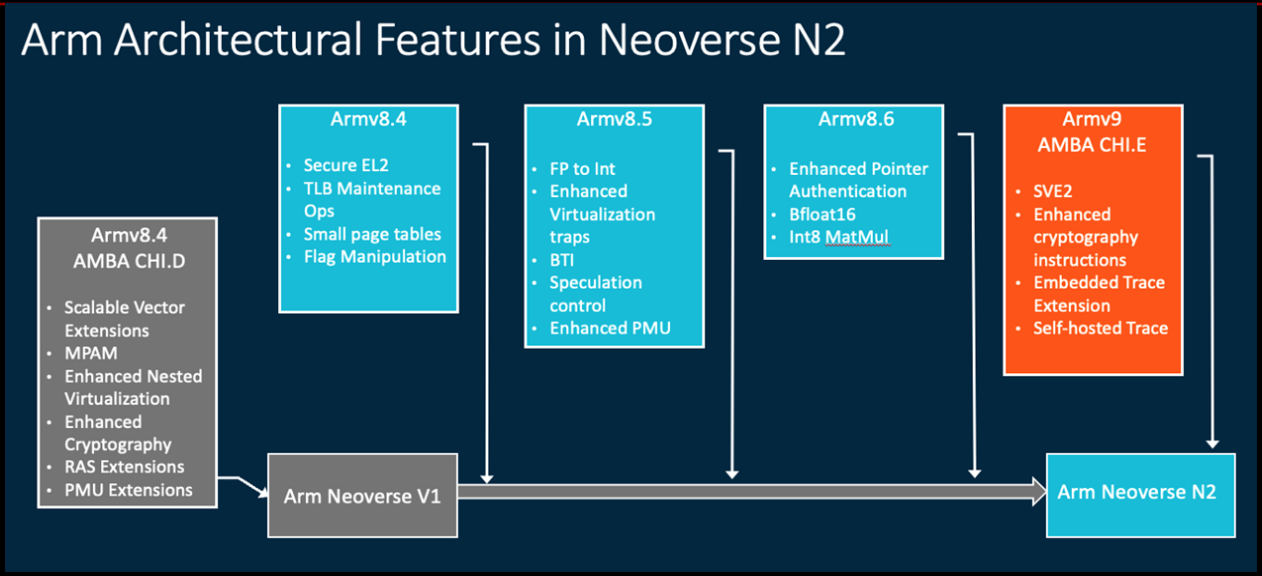
- https://www.arm.com/campaigns/arm-vision



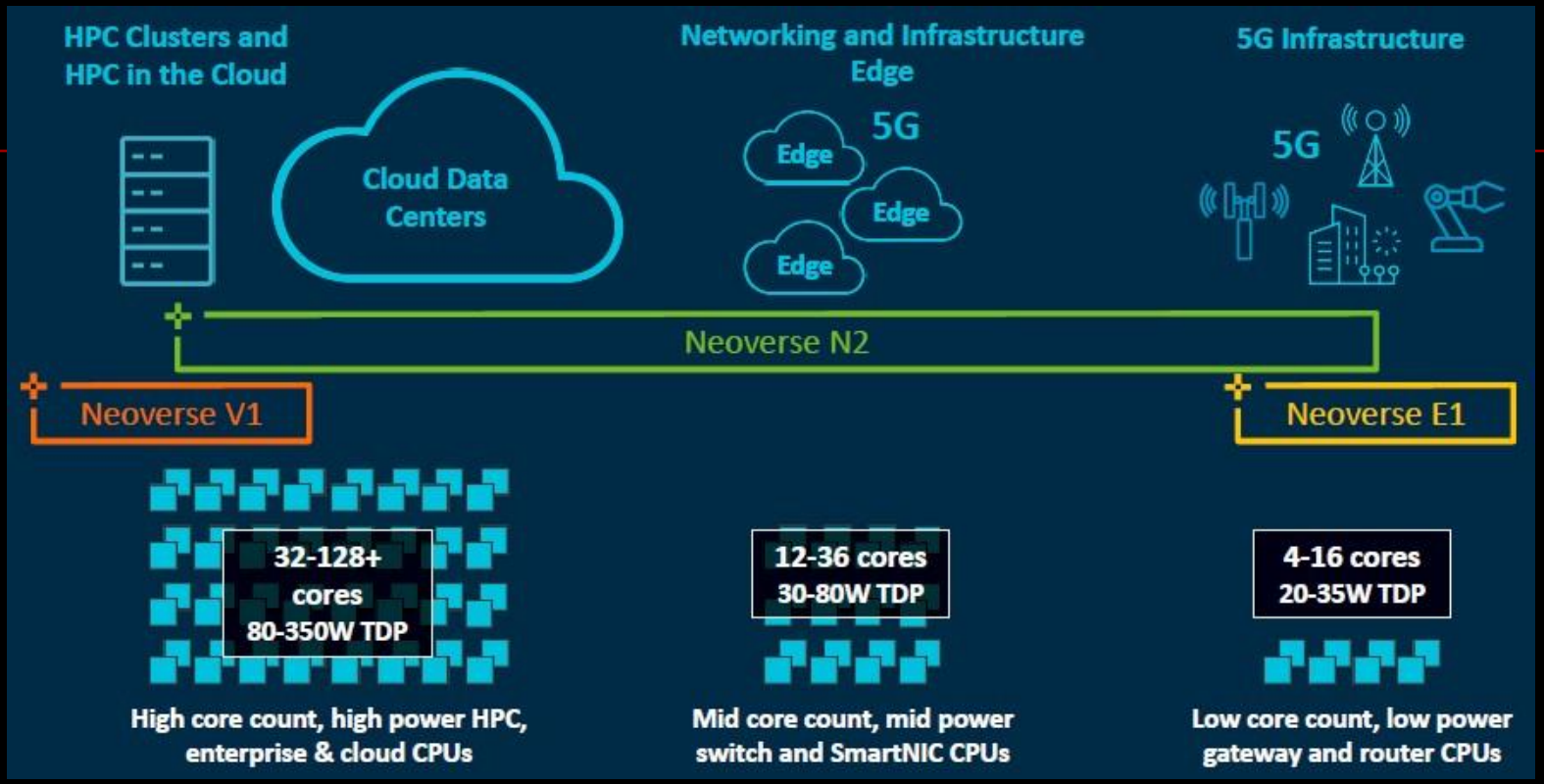


Neoverse N2

- **https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/arm-neoverse-n2-industry-leading-performance-efficiency**

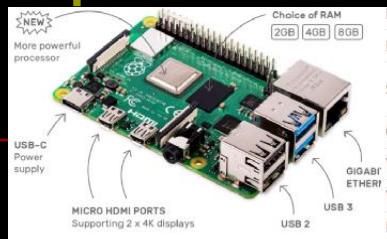


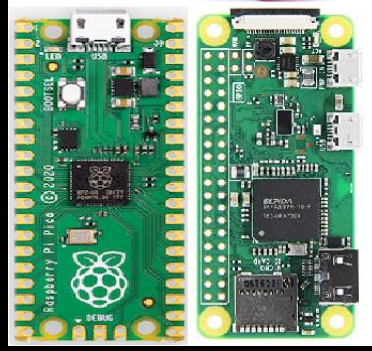
- https://www.nextplatform.com/2021/04/27/arm-puts-some-muscle-into-future-neoverse-server-cpu-designs/



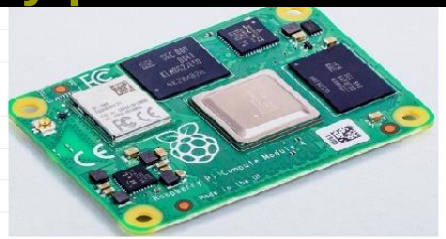
6.2 Raspberry Pi

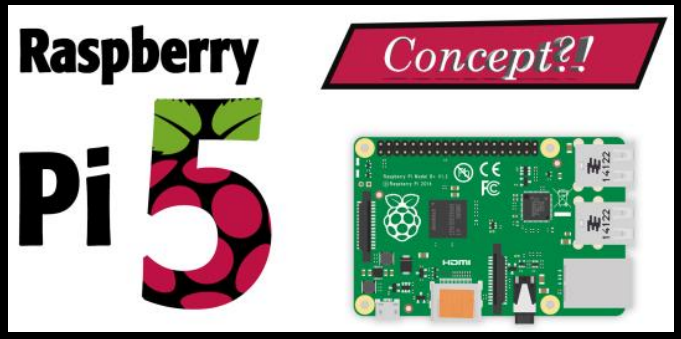
- https://www.raspberrypi.org/products/raspberry-pi-4-model-b/





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)





6.2.1 System

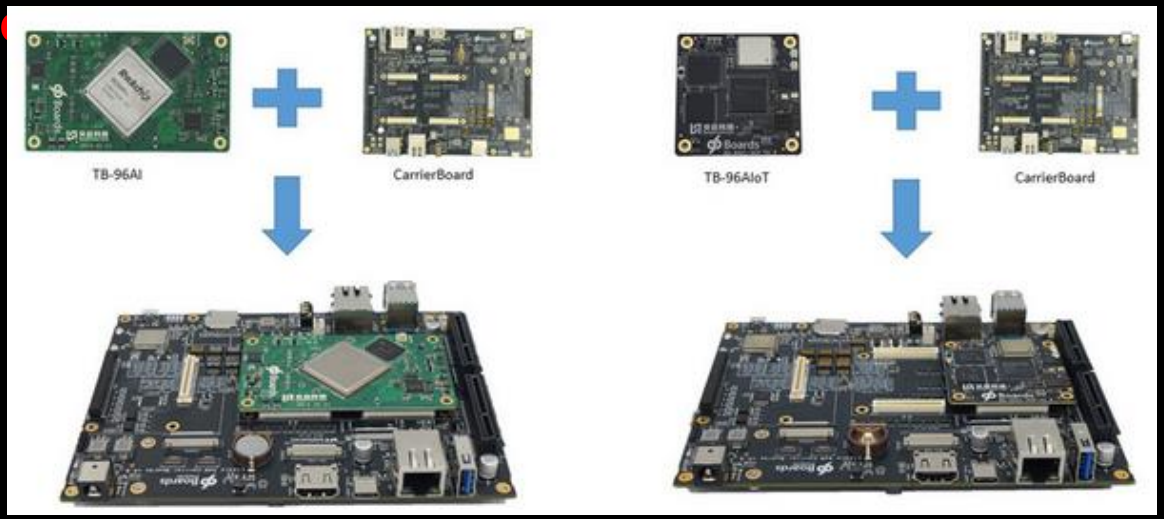
Fedora

- 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!

6.2.2 Clustering

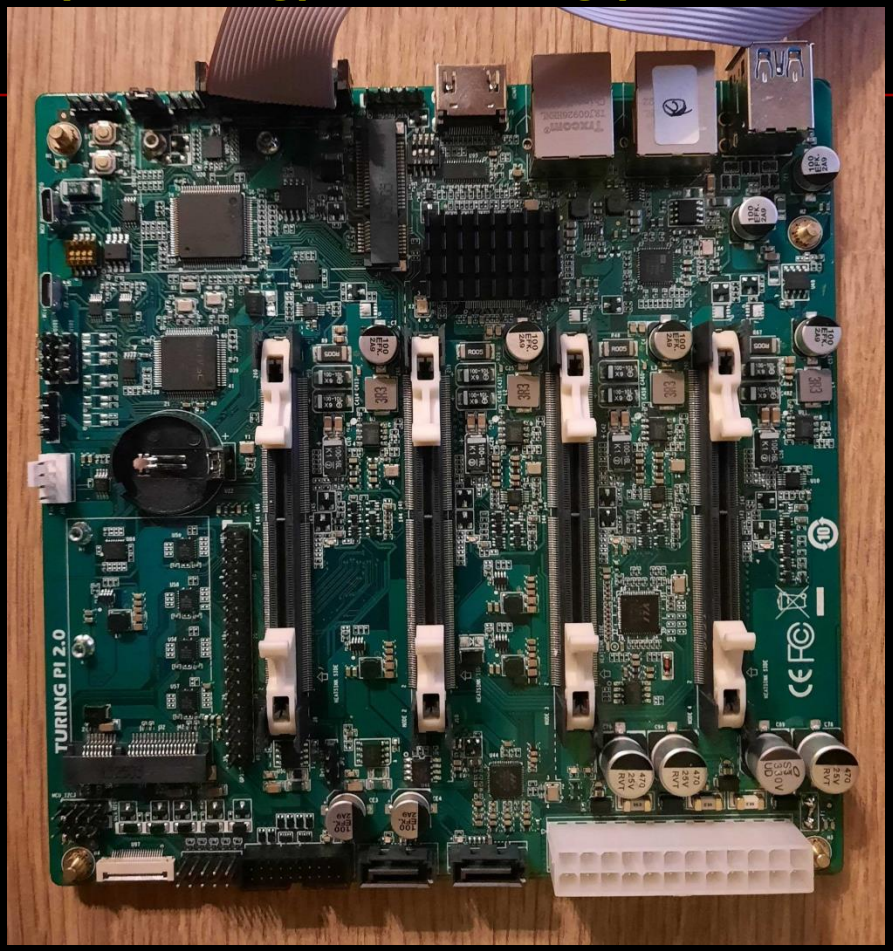
SOM/COM

- https://www.linaro.org/news/linaro-announces-launch-of-96boards-system-on-module-som-specification/
- http://linuxgizmos.com/linaro-launches-two-96boards-som-specifications/
- http://static.linaro.org/assets/specifications/96BoardsComputeSoMSpecificationV1.0.pdf
- https://static.linaro.org/assets/specifications/96BoardsWirelessSoMSpecificationV1.0.pdf
-



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.

Reliable, Scalable, Cloud-Native Infrastructure for the Edge

- Layer 2 Managed Switch
- 12 Gbps Backplane Bandwidth
- 2 x Mini PCI Express
- Cluster Management Bus (I2C)
- 2 x SATA III 6 Gbps
- VLAN Support

Self-hosted

Host cloud applications locally or at the edge

Learning

Learn Kubernetes, Docker, Serverless

Development

Build cloud-native and CI/CD for ARM edge infrastructure

Network-Attached Storage

Connect up to 2 x 2.5" SSD storage

6.2.3 RPi in IoT

RevPi

- https://revolutionpi.com/
  Open source, modular, cost-effective. Your tool of choice for implementing your IIoT & automation projects.
- https://www.kunbus.com/
- https://revolutionpi.com/revolution-pi-series/
  The base module RevPi Core 3 dismantled into its components:

Available modules:

Base modules


RevPi Connect+/Connect


RevPi Connect+ feat. CODESYS


RevPi Core 3+/Core 3/Core


RevPi Compact


RevPi Flat

Expansion modules (only for RevPi Connect & RevPi Core base modules)


Digital IO modules


Analog IO module

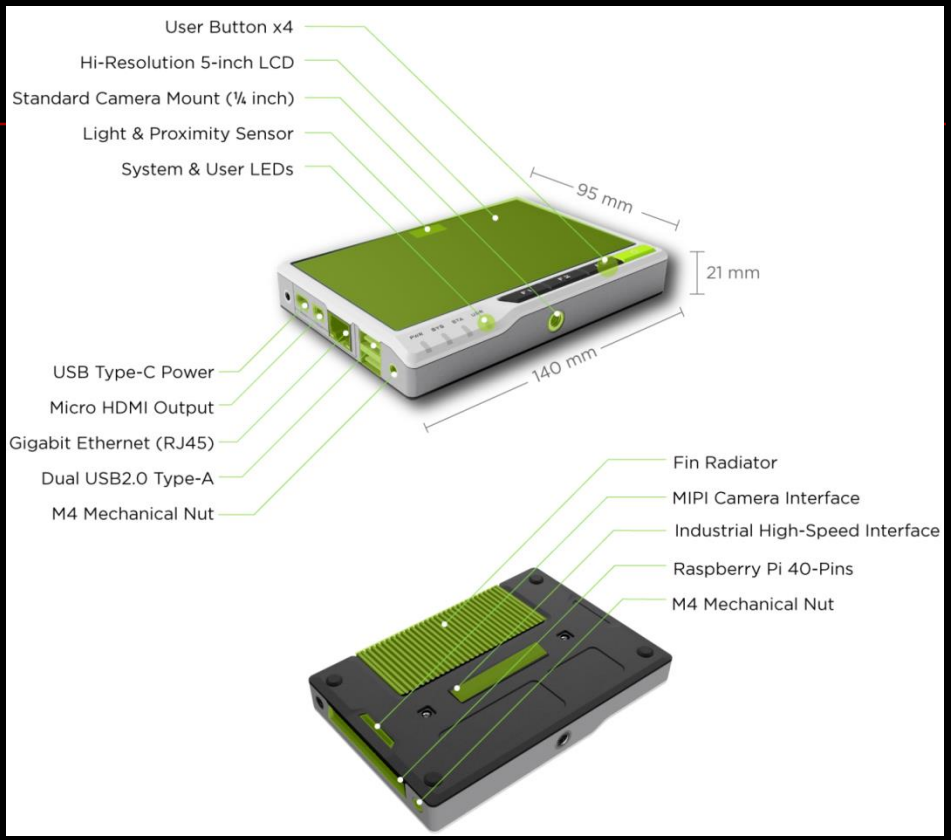

Analog & digital IO module

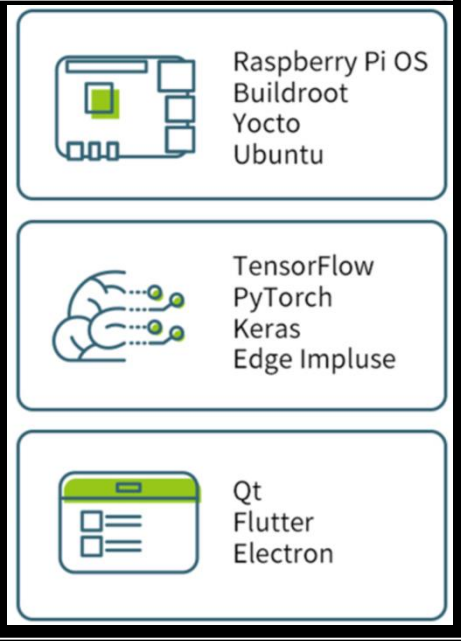

CON expansion modules


Gateways

reTerminal

- https://wiki.seeedstudio.com/reTerminal/

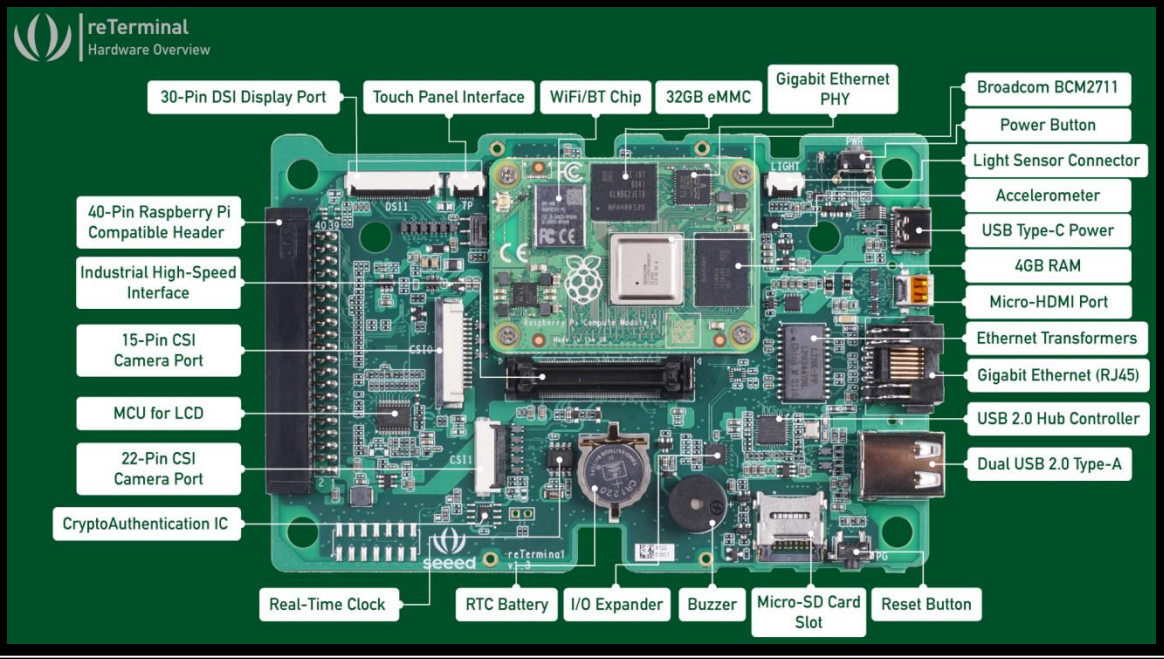




Features

- Integrated modular design with high stability and expandability
- Powered by Raspberry Pi Computer Module 4 with 4GB RAM & 32GB eMMC
- 5-Inch IPS capacitive multi-touch screen at 1280 x 720 and 293 PPI
- Wireless connectivity with dual-band 2.4GHz/5GHz Wi-Fi and Bluetooth 5.0 BLE
- High-speed expansion interface and rich I/O for more expandability
- Cryptographic co-processor with secure hardware-based key storage
- Built-in modules such as accelerometer, light sensor and RTC
- Gigabit Ethernet Port and Dual USB 2.0 Type-A ports
- 40-Pin Raspberry Pi compatible header for IoT applications

Block Diagram



6.3 Dev Env

RPi4 with Fedora

```
[mydev@fedora /]$ uname -a
Linux fedora 5.14.9-200.fc34.aarch64 #1 SMP Thu Sep 30 11:39:46 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-l
inker-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 /]$ java --version
openjdk 17 2021-09-14
OpenJDK Runtime Environment GraalVM CE 21.3.0-dev (build 17+35-jvmci-21.3-b03)
OpenJDK 64-Bit Server VM GraalVM CE 21.3.0-dev (build 17+35-jvmci-21.3-b03, mixed mode, sharing)
[mydev@fedora /]$
[mydev@fedora /]$ rustc -v
error: no input filename given

[mydev@fedora /]$ rustc -V
rustc 1.55.0 (Fedora 1.55.0-1.fc34)
[mydev@fedora /]$
[mydev@fedora /]$ cargo -V
cargo 1.55.0
[mydev@fedora /]$
[mydev@fedora /]$ which rustup
/usr/bin/which: no rustup in (/opt/MyWorkSpace/DevSW/SCM/Git/ACC/FGit/bin:/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java17-21.3.0-dev/bin:/opt/MyWorkSpace/DevSW/Build/Bazel/4.x:/opt/MyWorkSpace/DevSW/Build
/Gradle/7.x/bin:/opt/MyWorkSpace/DevSW/Go/Std/1.x/bin:/opt/MyWorkSpace/DevSW/DotNet/SDK/Std/6.x:/home/mydev/.dotnet/tools:/opt/MyWorkSpace/DevSW/DotNet/PowerShell/7.x:/home/mydev/.local/bin:/home/mydev/bin:
/usr/condabin:/usr/lib64/ccache:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/usr/libexec/sdcc:/home/mydev/.local/bin:/opt/MyWorkSpace/DevSW/Editor/Sublime/4.x:/opt/MyWorkSpace/DevSW/Go/Std/1.x/pkg/too
l/linux_arm64:/home/mydev/.arkade/bin:/home/mydev/.miscs/bin)
[mydev@fedora /]$
```

Rust

```
[mydev@fedora /]$ cat ~/.cargo/config
[source.crates-io]
registry = "https://github.com/rust-lang/crates.io-index"
replace-with = 'ustc'

[source.ustc]
registry = "https://mirrors.ustc.edu.cn/crates.io-index"

[source.sjtu]
registry = "https://mirrors.sjtug.sjtu.edu.cn/git/crates.io-index/"

[source.tuna]
registry = "https://mirrors.tuna.tsinghua.edu.cn/git/crates.io-index.git"

[source.rustcc]
registry = "https://code.aliyun.com/rustcc/crates.io-index.git"
[mydev@fedora /]$ _
```

7) Summary

You may refer to my previous talks as below for related content and more resources:

- "GraalVM-based unified runtime for eBPF-WASM" at GOTC (Shenzhen 2021)
- "Kata Containers for Edge Computing" at Kata Containers Meetup (Shanghai 2021)
- …

CHINA OpenInfra Days

II. aWsm

1) Overview

- **https://github.com/gwsystems/aWsm**

aWsm is a compiler and runtime for compiling WebAssembly (Wasm) code into llvm bytecode, then into sandboxed binaries you can run on various platforms. It focuses on generating very fast code (best of breed for WebAssembly), having a simple and extensible code-base, and on portability.

- **Motivation**

We're interested in taking Wasm out of the browser, and into all different types of computers (from servers to microcontrollers). Wasm provides a number of benefits outside of the web including:

- *Sandboxing.* Regardless which language is compiled to Wasm, the Wasm runtime can execute them *safely* (including those languages with footguns like C). This uses Software Fault Isolation (SFI) -- to ensure that loads and stores are only within the sandbox-- and Control-Flow Integrity (CFI) -- only allowing the execution of code intentionally generated by the compiler -- to provide safe execution. Restricting SFI protects the surrounding code from faulty or malicious code, and CFI prevents a compromise from hijacking the flow of execution (see buffer overflows and ROP chains).

  Sandboxing can act as a process replacement, especially on systems that don't have hardware support for processes, or where high-density is required.

- *Ubiquity.* Wasm is a high-level assembly, independent from any specific hardware. This has the potential to provide a universal specification of computation that can execute in the cloud, on the edge, or in an embedded device.

Features

- *Performance.* aWsm is an ahead-of-time compiler that leverages the LLVM compiler to optimize code, and target different architectural backends. We have evaluated aWsm on x86-64, aarch64 (Raspberry Pi), and thumb (ARM Cortex-M4 and M7), and performance on the microprocessors is within 10% of native, and within 40% on the microcontrollers on Polybench benchmarks.
- *Simplicity.* The entire code base for the compiler and runtime is relatively small. The compiler is <3.5K lines of Rust, and the runtime (for *all* platforms) is <5K lines of C. It is nearly trivial to implement different means of sandboxing memory accesses. We've implemented *seven* different mechanisms for this!
- *Portability.* Both the compiler and runtime are mostly platform-independent code, and porting to a new platform only really requires additional work if you need to tweak stack sizes (microcontrollers), or use architectural features (e.g., MPX, segmentation, etc...). aWsm only links what is needed, so it's possible to avoid microcontroller-expensive operations such as f64, f32, and even dynamic memory.
- *Composability.* The final output of aWsm is simple `*.o` elf objects that can be linked into larger systems. This enables the trivial composition of sandboxes together, and sandboxes into larger programs.

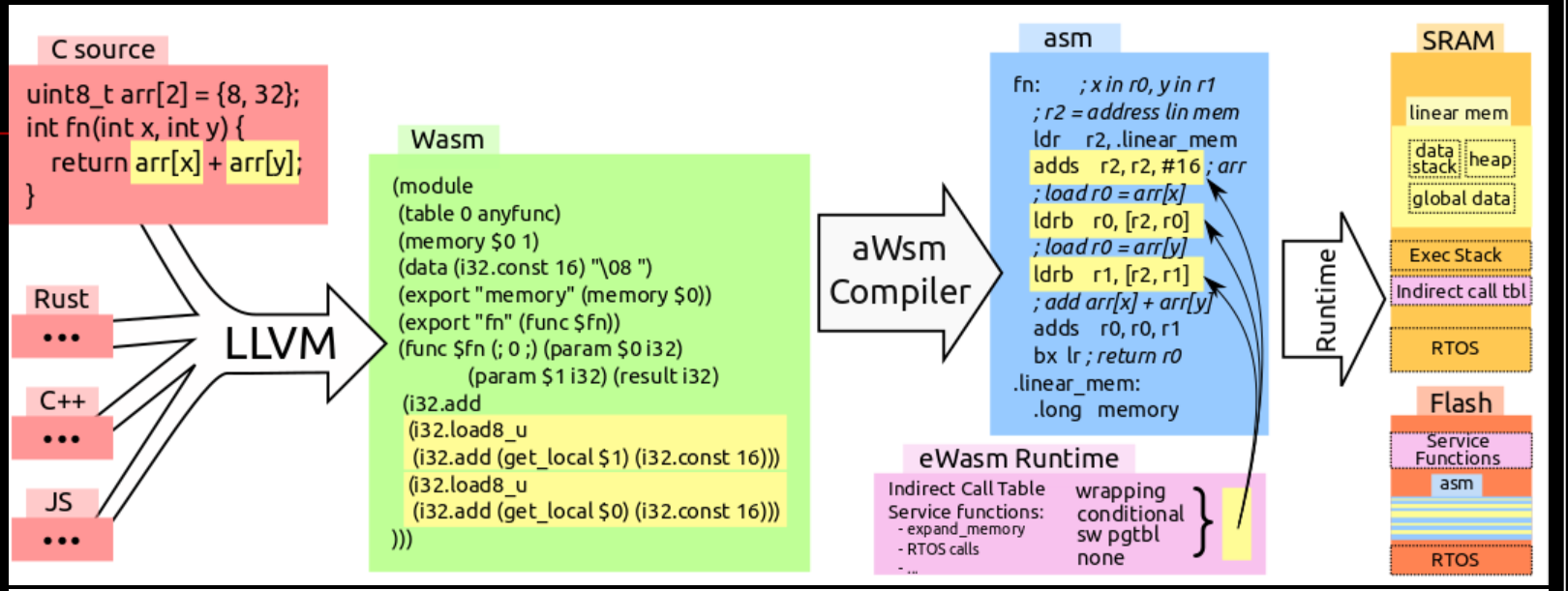
Comparison

Following table is from the best of our understanding of each system in July 2020:

Runtime	Method	x86_64	x86	aarch64	thumb	URL
aWsm	AoT	✓	✓	✓	✓	You are here
Wasmtime	AoT	✓	✓	✓		https://github.com/bytecodealliance/wasmtime
Wasmer	AoT	✓	✓	✓		https://github.com/wasmerio/wasmer
WAMR	Pseudo-AoT/Int	✓	✓	✓	✓	https://github.com/bytecodealliance/wasm-micro-runtime
Wasm3	Int	✓	✓	✓	✓	https://github.com/wasm3/wasm3
Wasmi	Int	✓	✓	✓	✓	https://github.com/paritytech/wasmi

How it works

Processing Pipeline



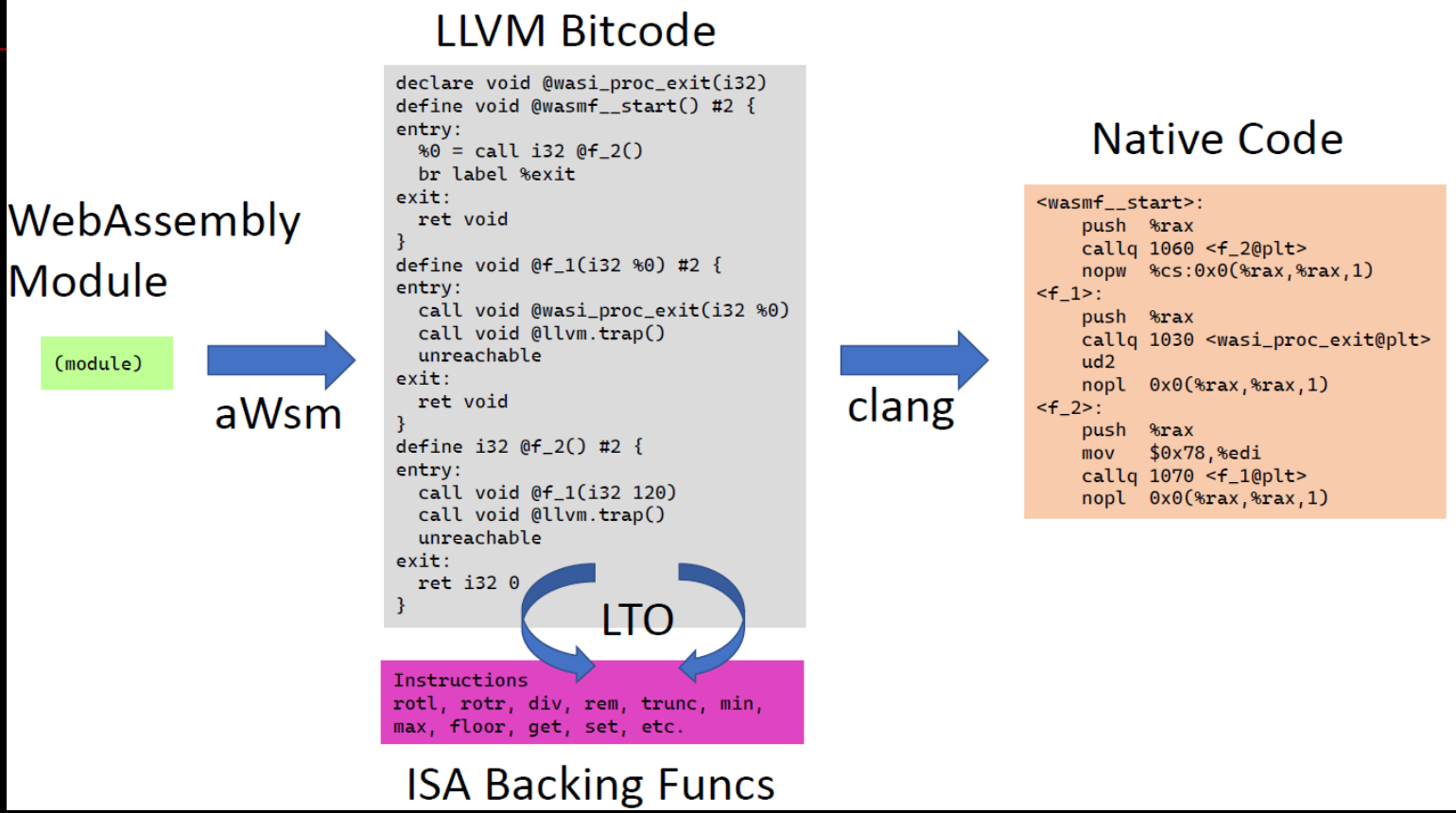
- Programming languages are compiled into Wasm, for example, using LLVM.
- Wasm has a binary representation and (as depicted) a s-expr representation.
- The aWsm compiler inputs binary Wasm, generates LLVM IR corresponding to the Wasm.
- This IR is compiled with the runtime to generate the final object that exports `wasm_main` to execute in the broader application.

In the Figure, we target Arm Cortex-M, and the yellow boxes emphasize how linear memory bounds checks transition throughout the process.

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

Compilation

aWsm transforms WebAssembly into LLVM Bitcode, which is then linked with runtime code implementing the core WebAssembly instructions and WASI interface and then link-time optimized into native code.



Source: https://conix.io/wp-content/uploads/pubs/3878/CONIX-Sledge-Poster.pdf

Runtime

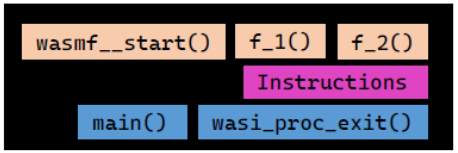
aWsm implements all safety checks in the runtime (in C). This maximizes the portability and extensibility of the system, and we've used this to prototype multiple bounds check implementations (see the discussion of three of these in the paper). To enable this, the compiler generates LLVM IR that calls the runtime for common operations including loading and storing in linear memory. We rely on the Link-Time Optimization (LTO) of LLVM to remove the boundaries between executable and runtime. Similarly, indirect function calls (function pointer invocations) are implemented within the C of the runtime.

aWsm uses a `musl` libc implementation (by default), and we interpose on the system calls by instead converting them to calls to the runtime. In this way, system calls can be sanitized, constrained, or transformed by the runtime. WASI support is of significant interest, but is not yet enabled.

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

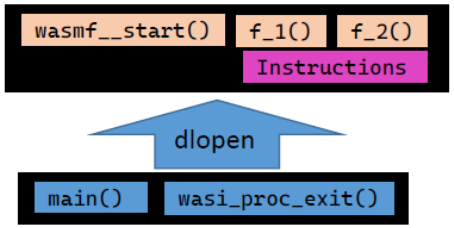
Deployment

Standalone Executable



Statically links application code with a minimal runtime and WASI backing functions.

*.so Shared Library

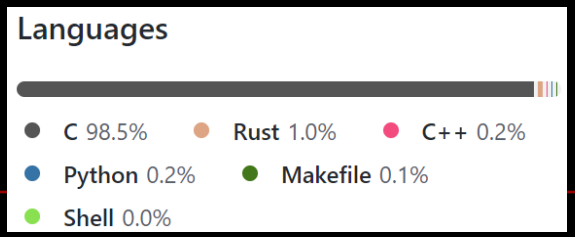


A runtime such as SLEdge resolves the required WASI imports and calls the exposed **`wasmf__start`** entrypoint.

Source: https://conix.io/wp-content/uploads/pubs/3878/CONIX-Sledge-Poster.pdf

Source Code

- Languages



- Directory Hierarchical

.github/workflows	refactor: Clang and Cargo Auto-formatting and CI Checks (#31)	2 months ago
code_benches	chore: trailing newline	2 months ago
doc	added video links in design.md	8 months ago
example_code	refactor: Clang and Cargo Auto-formatting and CI Checks (#31)	2 months ago
runtime	refactor: Clang and Cargo Auto-formatting and CI Checks (#31)	2 months ago
src	refactor: Clang and Cargo Auto-formatting and CI Checks (#31)	2 months ago
tests	refactor: Clang and Cargo Auto-formatting and CI Checks (#31)	2 months ago
wasmception @ 30bdf71	Readme cleanup + install script cleanup (#24)	7 months ago
.clang-format	refactor: Clang and Cargo Auto-formatting and CI Checks (#31)	2 months ago
.editorconfig	refactor: Clang and Cargo Auto-formatting and CI Checks (#31)	2 months ago
.gitignore	chore: run **@geky** one-liner from #21 (#28)	6 months ago
.gitmodules	change to use gwsystems repositories	2 years ago
.rustfmt.toml	refactor: Clang and Cargo Auto-formatting and CI Checks (#31)	2 months ago
Cargo.lock	chore: run **@geky** one-liner from #21 (#28)	6 months ago
Cargo.toml	chore: run **@geky** one-liner from #21 (#28)	6 months ago
LICENSE	Create LICENSE	2 years ago
README.md	chore: run **@geky** one-liner from #21 (#28)	6 months ago
format.sh	refactor: Clang and Cargo Auto-formatting and CI Checks (#31)	2 months ago
install_deb.sh	chore: run **@geky** one-liner from #21 (#28)	6 months ago
install_mac.sh	chore: add cmake install to Mac script	6 months ago
sections.ld	Add Cortex-M glue/support code	2 years ago

- **AOT compiler ($SRC_AWSM/src)**

```
[mydev@fedora awsm-master]$ tree -L 2 src
src
├── codegen
│   ├── block.rs
│   ├── breakout.rs
│   ├── function.rs
│   ├── globals.rs
│   ├── memory.rs
│   ├── mod.rs
│   ├── runtime_stubs.rs
│   ├── table.rs
│   └── type_conversions.rs
├── llvm_externs.rs
├── main.rs
└── wasm.rs
```

```
[mydev@fedora awsm-master]$ tokei ./src
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Rust                   12         4287         3622          156          509
 |- Markdown             1            6            0            6            0
 (Total)                            4293         3622          162          509
===============================================================================
 Total                  12         4293         3622          162          509
===============================================================================
```

- **Submodules**

  https://github.com/gwsystems/wasmception

  Minimal toolset for building wasm files

  (may be completely replaced by WASI SDK in the future...)

- **Dependencies**

  $SRC_AWSM/Cargo.toml:

```
1  [package]
2  name = "awsm"
3  version = "0.1.0"
4  authors = ["Gregor Peach <gregorpeach@gmail.com>"]
5  edition = "2018"
6
7  [dependencies]
8  flexi_logger = "0.14.4"
9  llvm-alt = { git = "https://github.com/gwsystems/llvm-rs", branch = "sfbase"}
10 log = "0.4.8"
11 structopt = "0.3.2"
12 wasmparser = "0.39.2"
13
14 [profile.release]
15 debug = true
```

  https://github.com/gwsystems/llvm-rs

  a library that wraps LLVM using Rust idioms and the cbox library

  FYI，current official LLVM bindings ($SRC_LLVM/bindings)

```
[mydev@fedora llvm]$ tree -L 1 bindings/
bindings/
├── go
├── ocaml
├── python
└── README.txt
```

Runtime ($SRC_AWSM/runtime)

```
[mydev@fedora awsm-master]$ tree -L 2 runtime/
runtime/
├── cortex_m_glue
│   ├── cortexm_linkup.py
│   ├── hello.ld
│   ├── math2.c
│   ├── math3.c
│   ├── math4.c
│   ├── math.c
│   ├── printf.c
│   ├── printf.h
│   ├── qemu_pid.c
│   ├── qemu_putchar.c
│   ├── sections.ld
│   └── start.s
├── libc
│   ├── cortex_m_backing.c
│   ├── env.c
│   ├── wasi_sdk_backing.c
│   ├── wasi_sdk_backing.h
│   └── wasmception_backing.c
├── memory
│   ├── 64bit_nix.c
│   ├── cortex_m.c
│   ├── cortex_m_no_protection.c
│   ├── cortex_m_spt.c
│   ├── cortex_m_wrapping.c
│   ├── generic.c
│   ├── mpx.c
│   ├── no_protection.c
│   └── segmented.c
├── runtime.c
└── runtime.h
```

```
[mydev@fedora awsm-master]$ tokei ./runtime
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 GNU Style Assembly      1           40           35            0            5
 C                      21         7788         5667         1053         1068
 C Header                3          632          359          167          106
 Python                  1          205          136           27           42
===============================================================================
 Total                  26         8665         6197         1247         1221
===============================================================================
```

Code ($SRC_AWSM/code_benches)


```
[mydev@fedora awsm-master]$ tree -L 1 code_benches/
code_benches/
├── app_nn
├── app_pid
├── app_tinycrypt
├── app_tiny_ekf
├── app_v9
├── custom_binarytrees
├── custom_function_pointers
├── custom_libjpeg
├── custom_matrix_multiply
├── custom_memcmp
├── custom_sqlite
├── dummy.c
├── dummy.cpp
├── example.mk
├── mi_adpcm
├── mi_basic_math
├── mi_bitcount
├── mi_bitcount_cm
├── mi_blowfish
├── mi_crc
├── mi_dijkstra
├── mi_dijkstra_cm
├── mi_fft
├── mi_fft_cm
├── mi_gsm
├── mi_mandelbrot
├── mi_mandelbrot_cm
├── mi_patricia
├── mi_patricia_cm
├── mi_pgp
├── mi_qsort
├── mi_qsort_cm
├── mi_rsynth
├── mi_sha
├── mi_stringsearch
├── mi_susan
├── pb_datamining_correlation
├── pb_datamining_covariance
├── pb_la_blas_gemm
├── pb_la_blas_gemver
├── pb_la_blas_gesummv
├── pb_la_blas_symm
├── pb_la_blas_syr2k
├── pb_la_blas_syrk
├── pb_la_blas_trmm
├── pb_la_kernels_2mm
├── pb_la_kernels_3mm
├── pb_la_kernels_atax
├── pb_la_kernels_bicg
├── pb_la_kernels_doitgen
├── pb_la_kernels_mvt
├── pb_la_solvers_cholesky
├── pb_la_solvers_durbin
├── pb_la_solvers_gramschmidt
├── pb_la_solvers_lu
├── pb_la_solvers_ludcmp
├── pb_la_solvers_trisolv
├── pb_medely_deriche
├── pb_medely_floyd_warshall
├── pb_medely_nussinov
├── pb_stencils_adi
├── pb_stencils_fdtd_2d
├── pb_stencils_heat_3d
├── pb_stencils_jacobi_1d
├── pb_stencils_jacobi_2d
├── pb_stencils_seidel_2d
├── run.py
└── _test
```

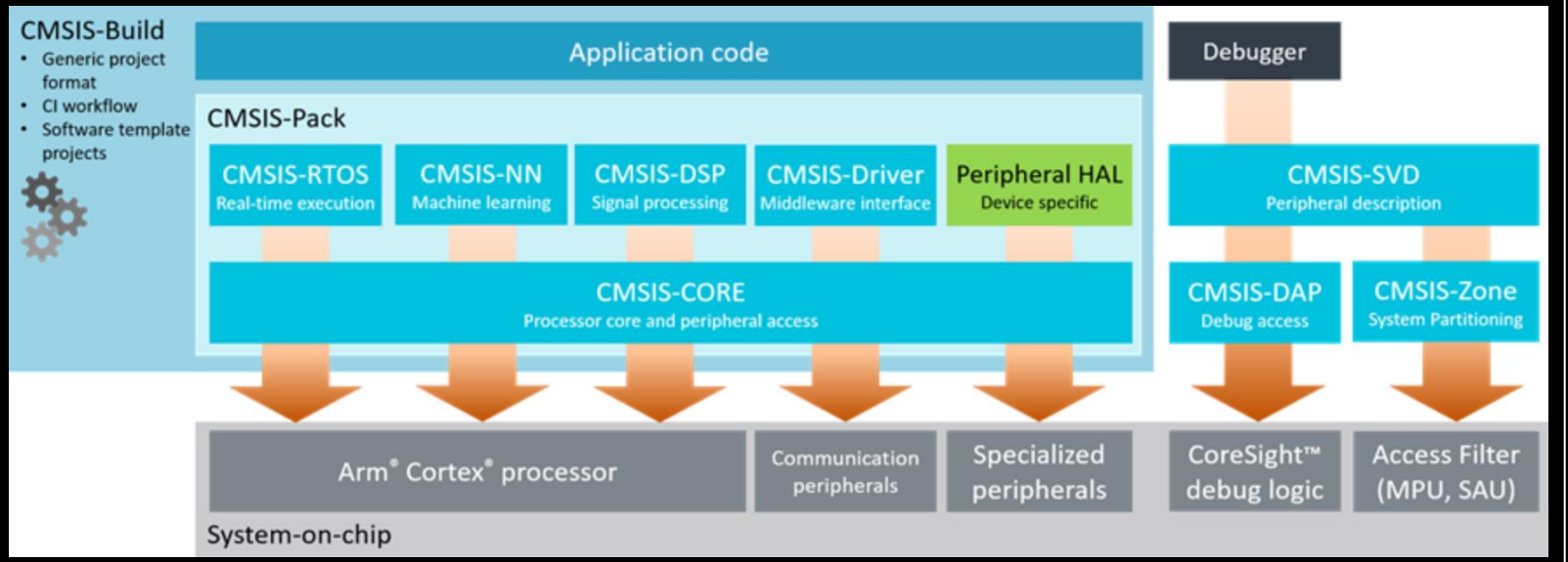
```
[mydev@fedora awsm-master]$ tokei ./code_benches
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 C                     510       413898       274581       101655        37662
 C Header              196        34554        13863        17100         3591
 C++                     5          779          523          127          129
 Makefile                2          146          105           15           26
 Module-Definition       5         1876         1803            0           73
 Python                  1          444          292           82           70
 Shell                   2           67           44           12           11
 Plain Text              4          174            0          165            9
===============================================================================
 Total                 725       451938       291211       119156        41571
===============================================================================
```

CMSIS:

https://github.com/ARM-software/CMSIS_5

https://arm-software.github.io/CMSIS_5/General/html/index.html

a set of tools, APIs, frameworks, and work flows that help to simplify software re-use, reduce the learning curve for microcontroller developers, speed-up project build and debug, and thus reduce the time to market for new applications.



https://github.com/ARM-software/CMSIS-Driver

…

CHINA OpenInfra Days

Performance

- **https://github.com/gwsystems/aWsm**

Belowing numbers are from May 2020:

PolyBench/C benchmarks for **x86-64** (slowdown over native C):

	Wasmer	WAVM	Node.js + Emscripten	Lucet	aWsm
Avg. Slowdown	149.8%	28.1%	84.0%	92.8%	13.4%
Stdev. in Slowdown	194.09	53.09	107.84	117.25	34.65

PolyBench/C benchmarks for **Arm aarch64** (slowdown over native C):

	aWsm
Avg. Slowdown	6.7%
Stdev. of Slowdown	19.38

Polybench/C benchmarks for **Arm Cortex-M** microcontrollers (slowdown over native C):

Architectures	aWsm
Cortex-M7 Avg. slowdown	40.2%
Cortex-M4 Avg. slowdown	24.9%

eWASM

- https://www2.seas.gwu.edu/~gparmer/publications/emsoft20wasm.pdf

Practical Software Fault Isolation for Reliable Embedded Devices.

The design of microcontroller-specific aWsm's runtime in details.

The eWASM runtime consists of a number of functions that are directly invoked from the Wasm sandbox. These include:

- linear memory accesses, that include the bounds-checking logic,
- function pointer invocation indirection tables and type-checking,
- the implementations of select system calls by pulling in a modified musl libc for backwards compatibility, and
- service functions to expose the underlying RTOS's functionality, where appropriate (*e.g.*, message passing based communication, or APIs for accessing I/O devices).

2) aWsm on RPi4

building from source

- cd $SRC_AWSM

```
[mydev@fedora awsm-master]$ cargo build --release
```

```
Compiling libc v0.2.91
Compiling version_check v0.9.3
Compiling autocfg v1.0.1
Compiling proc-macro2 v1.0.24
Compiling unicode-xid v0.2.1
Compiling memchr v2.3.4
Compiling syn v1.0.64
Compiling bitflags v1.2.1
Compiling gcc v0.3.55
Compiling semver v0.1.20
Compiling unicode-width v0.1.8
Compiling unicode-segmentation v1.7.1
Compiling log v0.4.14
Compiling ansi_term v0.11.0
Compiling regex-syntax v0.6.23
Compiling cfg-if v1.0.0
Compiling bitflags v0.5.0
Compiling llvm-alt v0.5.0 (https://github.com/gwsystems/llvm-rs?branch=sfbase#97ab94e6)
Compiling strsim v0.8.0
Compiling vec_map v0.8.2
Compiling glob v0.3.0
Compiling lazy_static v1.4.0
Compiling yansi v0.5.0
Compiling wasmparser v0.39.3
Compiling proc-macro-error-attr v1.0.4
Compiling proc-macro-error v1.0.4
Compiling num-traits v0.2.14
Compiling num-integer v0.1.44
Compiling textwrap v0.11.0
Compiling heck v0.3.2
Compiling llvm-sys v0.3.0 (https://github.com/gwsystems/llvm-sys.rs?branch=sfbase#f4d6b752)
Compiling quote v1.0.9
Compiling time v0.1.44
Compiling atty v0.2.14
Compiling cbox v0.3.0
Compiling aho-corasick v0.7.15
Compiling clap v2.33.3
Compiling regex v1.4.5
Compiling chrono v0.4.19
Compiling structopt-derive v0.4.14
Compiling flexi_logger v0.14.8
Compiling structopt v0.3.21
Compiling awsm v0.1.0 (/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master)
 Finished release [optimized + debuginfo] target(s) in 5m 45s
```

```
[mydev@fedora awsm-master]$ find . -name "awsm"
./target/release/awsm
[mydev@fedora awsm-master]$ file ./target/release/awsm
./target/release/awsm: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-aarch64.so.1, BuildID[sha1]=7e85d16604898e5a1ba87ca2af96d10bb4506561
, for GNU/Linux 3.7.0, with debug_info, not stripped
[mydev@fedora awsm-master]$
[mydev@fedora awsm-master]$ ldd ./target/release/awsm
        linux-vdso.so.1 (0x0000ffffbad33000)
        /usr/local/jemalloc/lib/libjemalloc.so.2 (0x0000ffffba818000)
        libLLVM-12.so => /lib64/libLLVM-12.so (0x0000ffffb510b000)
        libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x0000ffffb50da000)
        libpthread.so.0 => /lib64/libpthread.so.0 (0x0000ffffb50a5000)
        libdl.so.2 => /lib64/libdl.so.2 (0x0000ffffb5084000)
        libc.so.6 => /lib64/libc.so.6 (0x0000ffffb4ef9000)
        libm.so.6 => /lib64/libm.so.6 (0x0000ffffb4e48000)
        libstdc++.so.6 => /lib64/libstdc++.so.6 (0x0000ffffb4c12000)
        libffi.so.6 => /lib64/libffi.so.6 (0x0000ffffb4bf1000)
        libedit.so.0 => /lib64/libedit.so.0 (0x0000ffffb4b9d000)
        libz.so.1 => /lib64/libz.so.1 (0x0000ffffb4b6c000)
        libtinfo.so.6 => /lib64/libtinfo.so.6 (0x0000ffffb4b2b000)
        /lib/ld-linux-aarch64.so.1 (0x0000ffffbacf6000)
[mydev@fedora awsm-master]$
```

Executing and Testing

- `[mydev@fedora awsm-master]$ cd code_benches; ./run.py`

but:

```
/bin/sh: line 1: /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/dist/bin/clang: No such file or directory
Compiling custom_binarytrees 1/37
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/dist/bin/clang -Wl,--allow-undefined,-z,stack-size=16384,--no-threads,--stack-first,--no-entry,--export-all,--export=main,--export=dummy --target=wasm32-unknown-unknown-wasm -nostartfiles -O3 -flto --sysroot=/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/sysroot  -O3 -flto ../dummy.c binarytrees.c -o bin/custom_binarytrees.wasm
Traceback (most recent call last):
  File "/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/code_benches/./run.py", line 374, in <module>
    compile_to_wasm(p)
  File "/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/code_benches/./run.py", line 288, in compile_to_wasm
    sp.check_call(command, shell=True, cwd=program.name)
  File "/usr/lib64/python3.9/subprocess.py", line 373, in check_call
    raise CalledProcessError(retcode, cmd)
subprocess.CalledProcessError: Command '/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/dist/bin/clang -Wl,--allow-undefined,-z,stack-size=16384,--no-threads,--stack-first,--no-entry,--export-all,--export=main,--export=dummy --target=wasm32-unknown-unknown-wasm -nostartfiles -O3 -flto --sysroot=/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/sysroot  -O3 -flto ../dummy.c binarytrees.c -o bin/custom_binarytrees.wasm' returned non-zero exit status 127.
```

because there is no available Wasmception binary, so we have to build it from source:

`[mydev@fedora awsm-master]$ cd wasmception;make`

```
2021-10-02 09:01:00  File 'dCloning into 'llvm'...
fatal: unable to access 'https://github.com/gwsystems/wasmception-llvm.git/': HTTP/2 stream 1 was not closed cleanly before end of the underlying stream
make: *** [Makefile:26: src/llvm.CLONED] Error 128
efault' does not exist.
2021-10-02 09:01:00   Considering target file 'build'.
2021-10-02 09:01:00    File 'build' does not exist.
2021-10-02 09:01:00     Considering target file 'build/llvm.BUILT'.
2021-10-02 09:01:00      File 'build/llvm.BUILT' does not exist.
2021-10-02 09:01:00       Considering target file 'src/llvm.CLONED'.
2021-10-02 09:01:00        File 'src/llvm.CLONED' does not exist.
2021-10-02 09:01:00        Finished prerequisites of target file 'src/llvm.CLONED'.
2021-10-02 09:01:00       Must remake target 'src/llvm.CLONED'.
2021-10-02 09:01:00 Makefile:25: target 'src/llvm.CLONED' does not exist
2021-10-02 09:01:00 mkdir -p src/
2021-10-02 09:01:00 Putting child 0xffffa36423c0 (src/llvm.CLONED) PID 75135 on the chain.
2021-10-02 09:01:00 Live child 0xffffa36423c0 (src/llvm.CLONED) PID 75135
2021-10-02 09:01:00 Reaping winning child 0xffffa36423c0 PID 75135
2021-10-02 09:01:00 cd src/; git clone https://github.com/gwsystems/wasmception-llvm.git llvm
2021-10-02 09:01:00 Live child 0xffffa36423c0 (src/llvm.CLONED) PID 75136
2021-10-02 09:03:00 Reaping losing child 0xffffa36423c0 PID 75136
2021-10-02 09:03:00 Removing child 0xffffa36423c0 PID 75136 from chain.
```

may be failed due to poor network conditions…

- **pre-download**

check $SRC_AWSM/wasmception/Makefile (https://github.com/gwsystems/wasmception/blob/30bdf71fb169830c7ac87a28c4f337f30fde4626/Makefile):

192 lines (170 sloc) | 6.78 KB

```
1   # Any copyright is dedicated to the Public Domain.
2   # http://creativecommons.org/publicdomain/zero/1.0/
3
4   # use one less job than number of cores
5   NTHDS=$(shell expr $(shell getconf _NPROCESSORS_ONLN) - 1)
6   ifeq ($(NTHDS), 0)
7   NTHDS=1
8   endif
9
10  ROOT_DIR=${CURDIR}
11  LLVM_SHA=eee32a02db4cedf5ddce806dfffaad21fbef17fe
12  CLANG_SHA=b362b05b29b0d5bf897d5f3e9d99eb60c0025d5d
13  LLD_SHA=5fb37bb34735f7006f2c22ff10e3e6081a9ce33a
14  COMPILER_RT_SHA=250580a9aee433b34c9e187a72b8dda9ac75c4ec
15  LIBCXX_SHA=02b189877a38a4fd583d3d4770afa29bd4f4dde1
16  LIBCXXABI_SHA=d66bcda1e1d200e707d33eb204d9f89eb0c3eb77
17  MUSL_SHA=d9e28df3d85c0bb3b53c8f2e5a16f69dc74162a3
18
19  default: build
20
21  clean:
22          rm -rf build src dist sysroot wasmception-*-bin.tar.gz
23
24  src/llvm.CLONED:
25          mkdir -p src/
26          cd src/; git clone https://github.com/gwsystems/wasmception-llvm.git llvm
27  ifdef LLVM_SHA
28          cd src/llvm; git checkout $(LLVM_SHA)
29  endif
30          cd src/llvm/tools; git clone https://github.com/gwsystems/wasmception-clang.git clang
31  ifdef CLANG_SHA
32          cd src/llvm/tools/clang; git checkout $(CLANG_SHA)
33  endif
34          cd src/llvm/tools; git clone https://github.com/gwsystems/wasmception-lld.git lld
35  ifdef LLD_SHA
36          cd src/llvm/tools/lld; git checkout $(LLD_SHA)
37  endif
38          touch src/llvm.CLONED
```

```
40  src/musl.CLONED:
41          mkdir -p src/
42          cd src/; git clone https://github.com/gwsystems/wasmception-musl.git musl
43  ifdef MUSL_SHA
44          cd src/musl; git checkout $(MUSL_SHA)
45  endif
46          touch src/musl.CLONED
47
48  src/compiler-rt.CLONED:
49          mkdir -p src/
50          cd src/; git clone https://github.com/gwsystems/wasmception-compiler-rt.git compiler-rt
51  ifdef COMPILER_RT_SHA
52          cd src/compiler-rt; git checkout $(COMPILER_RT_SHA)
53  endif
54          touch src/compiler-rt.CLONED
55
56  src/libcxx.CLONED:
57          mkdir -p src/
58          cd src/; git clone https://github.com/gwsystems/wasmception-libcxx.git libcxx
59  ifdef LIBCXX_SHA
60          cd src/libcxx; git checkout $(LIBCXX_SHA)
61  endif
62          cd src/libcxx; patch -p 1 < $(ROOT_DIR)/patches/libcxx.patch
63          touch src/libcxx.CLONED
64
65  src/libcxxabi.CLONED:
66          mkdir -p src/
67          cd src/; git clone https://github.com/gwsystems/wasmception-libcxxabi.git libcxxabi
68  ifdef LIBCXXABI_SHA
69          cd src/libcxxabi; git checkout $(LIBCXXABI_SHA)
70  endif
71          touch src/libcxxabi.CLONED
```

in our case:

```
[mydev@fedora awsm-master]$ cd wasmception;tree -L 1 src
src
├── compiler-rt
├── libcxx
├── libcxxabi
├── llvm
└── musl
```

```
[mydev@fedora awsm-master]$ cd wasmception;git diff
diff --git a/Makefile b/Makefile
index f4da57d..93cda86 100644
--- a/Makefile
+++ b/Makefile
@@ -23,15 +23,12 @@ clean:

 src/llvm.CLONED:
        mkdir -p src/
-       cd src/; git clone https://github.com/gwsystems/wasmception-llvm.git llvm
 ifdef LLVM_SHA
        cd src/llvm; git checkout $(LLVM_SHA)
 endif
-       cd src/llvm/tools; git clone https://github.com/gwsystems/wasmception-clang.git clang
 ifdef CLANG_SHA
        cd src/llvm/tools/clang; git checkout $(CLANG_SHA)
 endif
-       cd src/llvm/tools; git clone https://github.com/gwsystems/wasmception-lld.git lld
 ifdef LLD_SHA
        cd src/llvm/tools/lld; git checkout $(LLD_SHA)
 endif
@@ -39,7 +36,6 @@ endif

 src/musl.CLONED:
        mkdir -p src/
-       cd src/; git clone https://github.com/gwsystems/wasmception-musl.git musl
 ifdef MUSL_SHA
        cd src/musl; git checkout $(MUSL_SHA)
 endif
@@ -47,7 +43,6 @@ endif

 src/compiler-rt.CLONED:
        mkdir -p src/
-       cd src/; git clone https://github.com/gwsystems/wasmception-compiler-rt.git compiler-rt
 ifdef COMPILER_RT_SHA
        cd src/compiler-rt; git checkout $(COMPILER_RT_SHA)
 endif
@@ -55,7 +50,6 @@ endif

 src/libcxx.CLONED:
        mkdir -p src/
-       cd src/; git clone https://github.com/gwsystems/wasmception-libcxx.git libcxx
 ifdef LIBCXX_SHA
        cd src/libcxx; git checkout $(LIBCXX_SHA)
 endif
@@ -64,7 +58,6 @@ endif

 src/libcxxabi.CLONED:
        mkdir -p src/
-       cd src/; git clone https://github.com/gwsystems/wasmception-libcxxabi.git libcxxabi
 ifdef LIBCXXABI_SHA
        cd src/libcxxabi; git checkout $(LIBCXXABI_SHA)
 endif
[mydev@fedora wasmception]$
```

rebuild with pre-download source, but as a result, you are especially likely to encounter the following errors:

```
2021-10-05 06:11:30     Must remake target 'lib/Support/CMakeFiles/LLVIn file included from /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/De
mangle/MicrosoftDemangle.h:14,
                 from /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/lib/Demangle/MicrosoftDemangle.cpp:17:
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h:14:17: error: found ':' in nested-name-specifier, expected '
::'
   14 | enum Qualifiers : uint8_t {
      |                 ^
      |                 ::
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h:14:6: error: 'Qualifiers' has not been declared
   14 | enum Qualifiers : uint8_t {
      |      ^~~~~~~~~~
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h:14:27: error: expected unqualified-id before '{' token
   14 | enum Qualifiers : uint8_t {
      |                           ^
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h:25:6: warning: elaborated-type-specifier for a scoped enum m
ust not use the 'class' keyword
   25 | enum class StorageClass : uint8_t {
      | ~~~~ ^~~~~
      |      -----
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h:25:25: error: found ':' in nested-name-specifier, expected '
::'
   25 | enum class StorageClass : uint8_t {
      |                         ^
      |                         ::
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h:25:12: error: 'StorageClass' has not been declared
   25 | enum class StorageClass : uint8_t {
      |            ^~~~~~~~~~~~
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h:25:35: error: expected unqualified-id before '{' token
   25 | enum class StorageClass : uint8_t {
      |                                   ^
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h:38:6: warning: elaborated-type-specifier for a scoped enum m
ust not use the 'class' keyword
   38 | enum class CallingConv : uint8_t {
      | ~~~~ ^~~~~
      |      -----
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h:38:24: error: found ':' in nested-name-specifier, expected '
::'
   38 | enum class CallingConv : uint8_t {
      |                        ^
      |                        ::
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h:38:12: error: 'CallingConv' has not been declared
   38 | enum class CallingConv : uint8_t {
      |            ^~~~~~~~~~~
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h:38:34: error: expected unqualified-id before '{' token
   38 | enum class CallingConv : uint8_t {
      |                                  ^
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h:51:6: warning: elaborated-type-specifier for a scoped enum m
ust not use the 'class' keyword
   51 | enum class ReferenceKind : uint8_t { None, LValueRef, RValueRef };
      | ~~~~ ^~~~~
      |      -----
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/src/llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h:51:26: error: found ':' in nested-name-specifier, expected '
::'
   51 | enum class ReferenceKind : uint8_t { None, LValueRef, RValueRef };
```

...

you may refer to https://bugs.gentoo.org/732094 for the root cause (llvm/include/llvm/Demangle/MicrosoftDemangleNodes.h).

prepare a new patch to deal with this issue:

```
[mydev@fedora awsm-master]$ tree wasmception/patches
wasmception/patches
├── libcxx.patch
├── llvm.demangle.patch
└── musl.1.patch
```

```
[mydev@fedora awsm-master]$ cd wasmception;git status
HEAD detached at 30bdf71
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:   Makefile

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        patches/llvm.demangle.patch

no changes added to commit (use "git add" and/or "git commit -a")
[mydev@fedora wasmception]$
[mydev@fedora wasmception]$ cat patches/llvm.demangle.patch
diff --git a/include/llvm/Demangle/MicrosoftDemangleNodes.h b/include/llvm/Demangle/MicrosoftDemangleNodes.h
index 1d0b66a7bf4..30321b15656 100644
--- a/include/llvm/Demangle/MicrosoftDemangleNodes.h
+++ b/include/llvm/Demangle/MicrosoftDemangleNodes.h
@@ -4,6 +4,8 @@
 #include "llvm/Demangle/Compiler.h"
 #include "llvm/Demangle/StringView.h"
 #include <array>
+#include <cstdint>
+#include <string>

 class OutputStream;

@@ -596,4 +598,4 @@ struct FunctionSymbolNode : public SymbolNode {
 } // namespace ms_demangle
 } // namespace llvm

-#endif
\ No newline at end of file
+#endif
[mydev@fedora wasmception]$
```

```
[mydev@fedora wasmception]$ git diff
diff --git a/Makefile b/Makefile
index f4da57d..d2652e1 100644
--- a/Makefile
+++ b/Makefile
@@ -23,15 +23,13 @@ clean:

 src/llvm.CLONED:
        mkdir -p src/
-       cd src/; git clone https://github.com/gwsystems/wasmception-llvm.git llvm
 ifdef LLVM_SHA
        cd src/llvm; git checkout $(LLVM_SHA)
 endif
-       cd src/llvm/tools; git clone https://github.com/gwsystems/wasmception-clang.git clang
+       cd src/llvm; patch -p 1 < $(ROOT_DIR)/patches/llvm.demangle.patch
 ifdef CLANG_SHA
        cd src/llvm/tools/clang; git checkout $(CLANG_SHA)
 endif
-       cd src/llvm/tools; git clone https://github.com/gwsystems/wasmception-lld.git lld
 ifdef LLD_SHA
        cd src/llvm/tools/lld; git checkout $(LLD_SHA)
 endif
@@ -39,7 +37,6 @@ endif

 src/musl.CLONED:
        mkdir -p src/
-       cd src/; git clone https://github.com/gwsystems/wasmception-musl.git musl
 ifdef MUSL_SHA
        cd src/musl; git checkout $(MUSL_SHA)
 endif
@@ -47,7 +44,6 @@ endif

 src/compiler-rt.CLONED:
        mkdir -p src/
-       cd src/; git clone https://github.com/gwsystems/wasmception-compiler-rt.git compiler-rt
 ifdef COMPILER_RT_SHA
        cd src/compiler-rt; git checkout $(COMPILER_RT_SHA)
 endif
@@ -55,7 +51,6 @@ endif

 src/libcxx.CLONED:
        mkdir -p src/
-       cd src/; git clone https://github.com/gwsystems/wasmception-libcxx.git libcxx
 ifdef LIBCXX_SHA
        cd src/libcxx; git checkout $(LIBCXX_SHA)
 endif
@@ -64,7 +59,6 @@ endif

 src/libcxxabi.CLONED:
        mkdir -p src/
-       cd src/; git clone https://github.com/gwsystems/wasmception-libcxxabi.git libcxxabi
 ifdef LIBCXXABI_SHA
        cd src/libcxxabi; git checkout $(LIBCXXABI_SHA)
 endif
[mydev@fedora wasmception]$
```

pass the build for Wasmception this time, and the result of run.py would be got:

...

```
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/dist/bin/clang -Wl,--allow-undefined,-z,stack-size=32768,--no-threads,--stack-first,--no-entry,--export-all,--export
=main,--export=dummy --target=wasm32-unknown-unknown-wasm -nostartfiles -O3 -flto --sysroot=/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/wasmception/sysroot  -O3 -flto ../du
mmy.c polybench.c seidel-2d.c -o bin/pb_stencils_seidel_2d.wasm
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/target/release/awsm  bin/pb_stencils_seidel_2d.wasm -o bin/pb_stencils_seidel_2d.bc
/opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/target/release/awsm  -u bin/pb_stencils_seidel_2d.wasm -o bin/pb_stencils_seidel_2d_us.bc
clang  -lm -O3 -flto -g polybench.c seidel-2d.c -o bin/pb_stencils_seidel_2d
clang -lm  -O3 -flto -g bin/pb_stencils_seidel_2d.bc /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/runtime/runtime.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWs
m-Sledge/awsm-master/runtime/libc/wasmception_backing.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/runtime/libc/env.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge
/aWsm-Sledge/awsm-master/runtime/memory/cortex_m.c -o bin/pb_stencils_seidel_2d_cm
clang -lm  -O3 -flto -g bin/pb_stencils_seidel_2d_us.bc /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/runtime/runtime.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/
aWsm-Sledge/awsm-master/runtime/libc/wasmception_backing.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/runtime/libc/env.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-E
dge/aWsm-Sledge/awsm-master/runtime/memory/no_protection.c -o bin/pb_stencils_seidel_2d_np_us
clang -lm  -O3 -flto -g bin/pb_stencils_seidel_2d.bc /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/runtime/runtime.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWs
m-Sledge/awsm-master/runtime/libc/wasmception_backing.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/runtime/libc/env.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge
/aWsm-Sledge/awsm-master/runtime/memory/no_protection.c -o bin/pb_stencils_seidel_2d_np
clang -lm  -O3 -flto -g bin/pb_stencils_seidel_2d.bc /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/runtime/runtime.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWs
m-Sledge/awsm-master/runtime/libc/wasmception_backing.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/runtime/libc/env.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge
/aWsm-Sledge/awsm-master/runtime/memory/generic.c -o bin/pb_stencils_seidel_2d_bc
clang -lm  -O3 -flto -g bin/pb_stencils_seidel_2d.bc /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/runtime/runtime.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWs
m-Sledge/awsm-master/runtime/libc/wasmception_backing.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/runtime/libc/env.c /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge
/aWsm-Sledge/awsm-master/runtime/memory/64bit_nix.c -o bin/pb_stencils_seidel_2d_vm

Outputting to /opt/MyWorkSpace/MyProjs/Runtime/WASM/Serverless-Edge/aWsm-Sledge/awsm-master/code_benches/benchmarks.csv

Testing custom_binarytrees(16) native
1.0435
Testing custom_binarytrees(16) wasm no protection unsafe
1.9024 (82.31% slower)
Testing custom_binarytrees(16) wasm no protection
1.8570 (77.96% slower)
Testing custom_binarytrees(16) wasm bounds checked
2.5985 (149.02% slower)
Testing custom_binarytrees(16) wasm virtual memory
1.8147 (73.91% slower)

Testing custom_function_pointers() native
33.0623
Testing custom_function_pointers() wasm no protection unsafe
33.3797 (0.96% slower)
Testing custom_function_pointers() wasm no protection
33.3944 (1.00% slower)
Testing custom_function_pointers() wasm bounds checked
39.4822 (19.42% slower)
Testing custom_function_pointers() wasm virtual memory
40.0498 (21.13% slower)
```

...

benchmarks.csv:

Program	native	wasm no protection unsafe	wasm no protection	wasm bounds checked	wasm virtual memory
custom_binarytrees	1	1.82313594	1.779554284	2.490221526	1.739035869
custom_function_pointers	1	1.00959763	1.010042971	1.19417293	1.21134275
custom_matrix_multiply	1	1.058323408	1.074500994	1.176706667	1.060965563
custom_memcmp	1	1.015159917	1.014172417	1.00871647	1.021905265
app_pid	1	0.989361388	0.99148168	0.920395385	0.988236944
app_tiny_ekf	1	1.112203197	1.041231471	1.156712284	1.077062559
app_tinycrypt	1	1.204216357	1.20376527	1.329178371	1.203640506
pb_datamining_correlation	1	1.008730807	1.015218126	0.983584063	1.010484923
pb_datamining_covariance	1	1.026634766	1.035028986	1.031405147	1.016741487
pb_la_blas_gemm	1	1.003019228	1.027168048	1.025815048	0.99314023
pb_la_blas_gemver	1	1.035924169	1.02799317	1.022854669	1.019497127
pb_la_blas_gesummv	1	0.983821866	0.940385501	1.000634112	0.983952298
pb_la_blas_symm	1	1.047591747	1.057533129	1.05890979	0.916093602
pb_la_blas_syr2k	1	1.035131217	0.992700801	1.052405372	1.008271216
pb_la_blas_syrk	1	1.026230244	1.031939084	1.035836605	1.015084532
pb_la_blas_trmm	1	1.050867558	1.0016642	1.016303949	1.014307277
pb_la_kernels_2mm	1	1.015442951	1.016951893	1.02111721	1.004923225
pb_la_kernels_3mm	1	1.034588762	1.019164108	1.044562028	1.029960175
pb_la_kernels_atax	1	1.013412188	1.010322154	0.994839389	1.010131436
pb_la_kernels_bicg	1	1.000093867	0.986551205	0.977184195	0.98398325
pb_la_kernels_doitgen	1	1.026285871	1.029269275	1.026176677	1.049241329
pb_la_kernels_mvt	1	1.014775807	1.020103665	1.017170848	1.00734135
pb_la_solvers_cholesky	1	1.023555329	1.03005393	1.038038145	1.02525782
pb_la_solvers_durbin	1	1.022040068	1.074049902	1.029136444	1.016541793
pb_la_solvers_gramschmidt	1	1.017064636	1.021675429	0.965347374	1.028203848
pb_la_solvers_lu	1	1.023186512	1.025372574	1.049506594	1.005461405
pb_la_solvers_ludcmp	1	1.029224321	1.02506151	1.04330814	1.028110776
pb_la_solvers_trisolv	1	1.029425309	1.02252735	0.962972912	1.004436151
pb_medely_deriche	1	1.01612142	1.014492522	1.01983848	1.003834959
pb_medely_floyd_warshall	1	1.027075987	1.039128878	1.057033421	1.02318889
pb_medely_nussinov	1	1.014286133	1.038311883	1.043499859	1.019353099
pb_stencils_adi	1	1.031382854	1.034445217	1.030406241	0.997020166
pb_stencils_fdtd_2d	1	0.979981114	1.006199647	1.019247536	0.997225888
pb_stencils_heat_3d	1	1.02603293	1.025815432	1.039595179	1.02275844
pb_stencils_jacobi_1d	1	1.010862993	1.025703292	1.03656428	1.000826451
pb_stencils_jacobi_2d	1	1.060705041	1.02414134	1.040205021	1.021527712
pb_stencils_seidel_2d	1	1.028325378	1.023166988	1.022085747	1.015861939

III. SLEdge

1) Overview

- https://github.com/gwsystems/sledge-serverless-framework

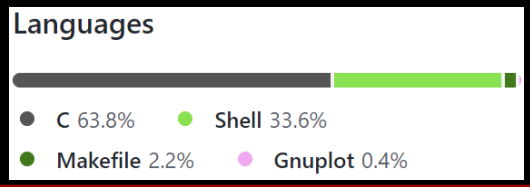
A lightweight serverless solution suitable for edge computing.
It builds on WASM sandboxing provided by the aWsm compiler.

- **Features**

- *Light-weight function isolation. Sledge* executes functions in light-weight Wasm sandboxes, removing the high overheads of traditional runtimes (*e.g.*, VMs and containers), while still providing strong memory isolation.
- *Optimized function startup for high densities.* Given the repeated execution of the same functions across many tenants, *Sledge* optimizes function startup by decoupling the processing (linking and loading) of function binaries from function instantiation.
- *Decoupling work-distribution from temporal isolation. Sledge* leverages the short-lived execution properties of serverless to specialize system scheduling by decoupling both the work-distribution and load balancing across cores for scalability, from the scheduling logic to maintain fairness and temporal isolation.
- *Serverless execution performance evaluation.* We perform an extensive evaluation of *Sledge* serverless runtime using various Edge workloads to show that *Sledge* provides up to 4 times better latencies and throughputs compared to *Nuclio* [57], one of the fastest open-source serverless frameworks. We also evaluate our *Sledge* Wasm compiler and its runtime on x86_64 and AArch64 architectures to show an average performance overhead within 13% of the native code execution for PolyBench/C [62] benchmarks and compare its efficiency with various existing LLVM- and Cranelift-based [19] Wasm compilers and runtimes.

Source: https://conix.io/wp-content/uploads/pubs/3878/CONIX-Sledge-Poster.pdf

Source Code



Directory Hierarchical

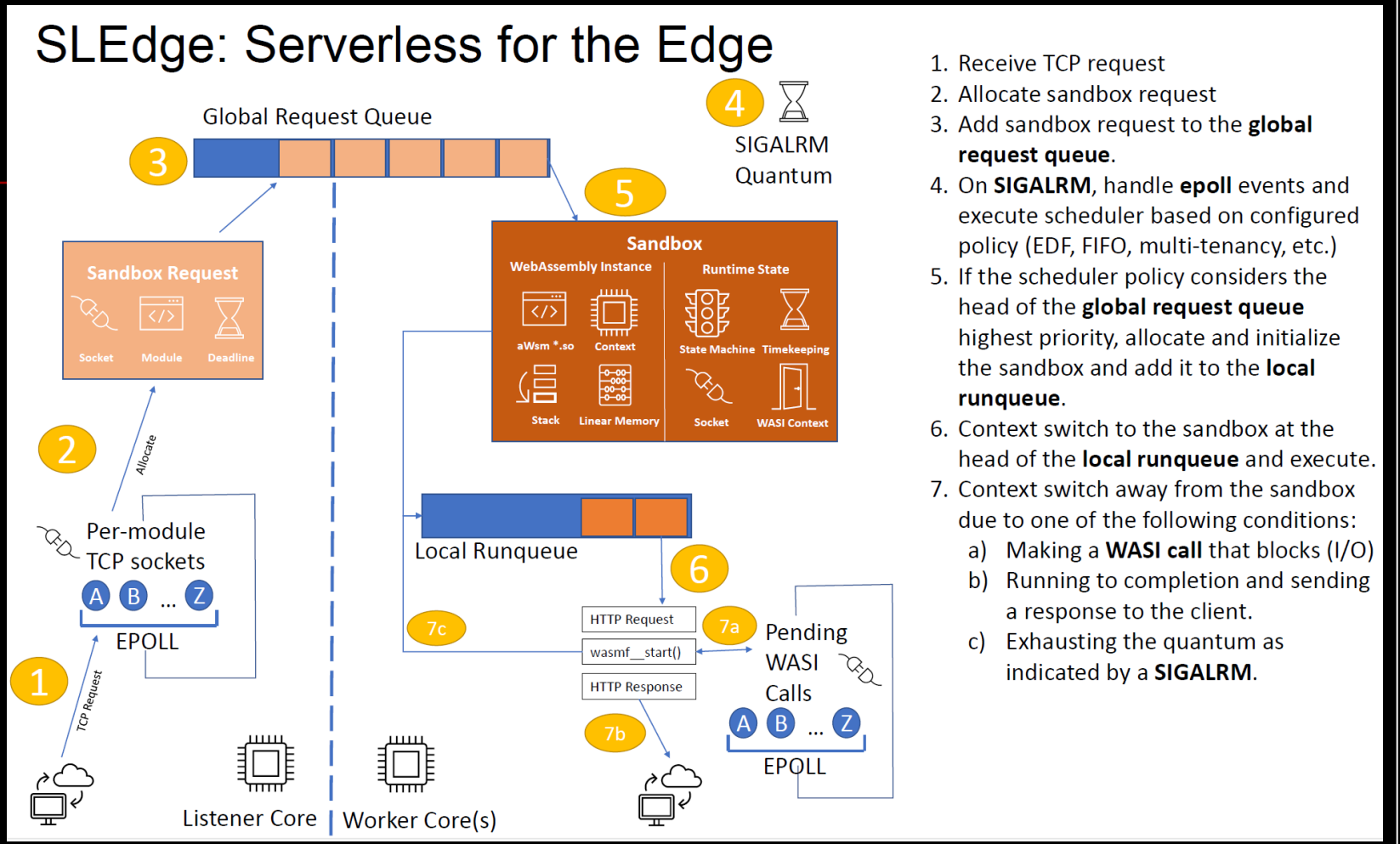
.devcontainer	chore: formatting nits	6 months ago
.github/workflows	test: invalidate test caches on compiletime change	2 days ago
.vscode	chore: trailing JSON newlines	2 days ago
awsm @ 708401d	chore: Update submodules	2 days ago
docs/sledge-states	chore: graphviz file formatting	3 hours ago
runtime	chore: Add trailing newline to JSON	15 hours ago
.clang-format	chore: align consecutive macros	2 years ago
.editorconfig	chore: graphviz file formatting	3 hours ago
.env	refactor: assorted bash cleanup	4 months ago
.gitignore	feat: VSCode build container and visual debugging	6 months ago
.gitmodules	chore: add speechtotext submodule	7 months ago
Dockerfile.aarch64	chore: formatting nits	6 months ago
Dockerfile.x86_64	Fix gocr hey (#256)	4 months ago
LICENSE	chore: first rename pass	14 months ago
Makefile	chore: fix paths for CI	4 months ago
README.md	doc: Remove extra newlines	2 days ago
devenv.sh	chore: clean up bind mount	4 months ago
fix_root.sh	chore: improve fix_root script	6 months ago
format.sh	chore: Apply shfmt to shell scripts	6 months ago
install.sh	chore: Apply shfmt to shell scripts	6 months ago
install_llvm.sh	chore: Install clang-format-11 as minimum	6 months ago
install_perf.sh	refactor: assorted bash cleanup	4 months ago
test.sh	refactor: assorted bash cleanup	4 months ago

Submodules

$SRC_SLEDGE/.gitmodules:

```
[submodule "awsm"]
path = awsm
url = https://github.com/gwsystems/aWsm
ignore = dirty
[submodule "http-parser"]
path = runtime/thirdparty/http-parser
url = https://github.com/gwsystems/http-parser.git
[submodule "ck"]
path = runtime/thirdparty/ck
url = https://github.com/gwsystems/ck.git
[submodule "jsmn"]
path = runtime/thirdparty/jsmn
url = https://github.com/gwsystems/jsmn.git
[submodule "runtime/tests/gocr"]
path = runtime/tests/gocr
url = https://github.com/gwsystems/gocr.git
branch = sledge
[submodule "runtime/tests/TinyEKF"]
path = runtime/tests/TinyEKF
url = https://github.com/gwsystems/TinyEKF.git
branch = sledge
[submodule "runtime/tests/CMSIS_5_NN"]
path = runtime/tests/CMSIS_5_NN
url = https://github.com/gwsystems/CMSIS_5_NN.git
branch = sledge
[submodule "runtime/tests/sod"]
path = runtime/tests/sod
url = https://github.com/gwsystems/sod.git
branch = sledge
[submodule "runtime/tests/speechtotext"]
path = runtime/tests/speechtotext
url = https://github.com/gwsystems/speechtotext.git
branch = sledge
```

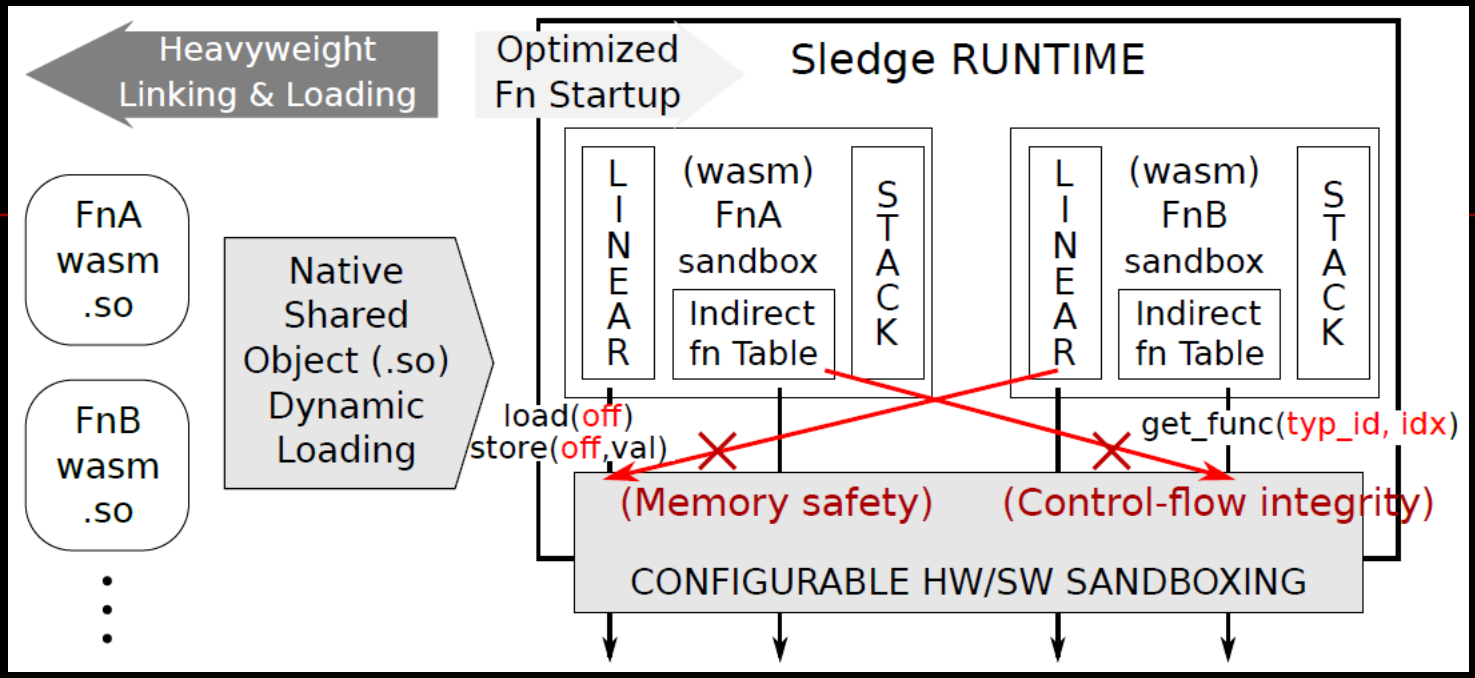
Workflow



1. Receive TCP request
2. Allocate sandbox request
3. Add sandbox request to the **global request queue**.
4. On **SIGALRM**, handle **epoll** events and execute scheduler based on configured policy (EDF, FIFO, multi-tenancy, etc.)
5. If the scheduler policy considers the head of the **global request queue** highest priority, allocate and initialize the sandbox and add it to the **local runqueue**.
6. Context switch to the sandbox at the head of the **local runqueue** and execute.
7. Context switch away from the sandbox due to one of the following conditions:
   a) Making a **WASI call** that blocks (I/O)
   b) Running to completion and sending a response to the client.
   c) Exhausting the quantum as indicated by a **SIGALRM**.

Source: https://conix.io/wp-content/uploads/pubs/3878/CONIX-Sledge-Poster.pdf

Runtime



The *Sledge* compiler, *aWsm*, and a runtime together enable the decoupling of process linking and loading, which is expensive, from the function instantiation, which is optimized for function startup times (Figure 2). *aWsm* compiles code for a function into an elf dynamic library (`.so`) that includes the necessary Wasm safety checks. To load in a new function, the *Sledge* runtime dynamically loads a function's `.so`. When a function execution is triggered, *Sledge* allocates and populates linear memory for the execution of the already linked and loaded Wasm module. By avoiding heavy-weight linking and loading, this approach supports faster function startup time.

Figure 2 helps to demonstrate the ability of *Sledge* compiler and runtime to provide configurable mechanisms for memory and function invocation bounds checks. Software conditional bounds checks for Wasm linear memory represent the most obvious overheads over native code. Such safety checks ("load" and "store" in Figure 2) transform native loads/stores into an addition (from the linear memory base), and a branch (for the bounds check) as shown in Figure 2. We implement these safety checks in the runtime, thus enabling their customization and implementation in C libraries.

Source: https://www2.seas.gwu.edu/~gparmer/publications/middleware20sledge.pdf

2) SLEdge on RPi4
2.1 Build by Docker

- cd $SRC_SLEDGE

```
[mydev@fedora sledge-serverless-framework-master]$ ./devenv.sh run
```

then you may encounter the following error message:

...

```
---[ Concurrency Kit has installed successfully.
mkdir -p /sledge/runtime/thirdparty/dist/
mkdir -p /sledge/runtime/thirdparty/dist//include/
cp jsmn/jsmn.h /sledge/runtime/thirdparty/dist//include/
mkdir -p /sledge/runtime/thirdparty/dist/
mkdir -p /sledge/runtime/thirdparty/dist//lib/
cd http-parser; cc  -I. -c http_parser.c
mv http-parser/http_parser.o /sledge/runtime/thirdparty/dist//lib/
mkdir -p /sledge/runtime/thirdparty/dist/
mkdir -p /sledge/runtime/thirdparty/dist//include/
cp http-parser/http_parser.h /sledge/runtime/thirdparty/dist//include/
Compiling runtime
clang -O3 -flto -g -pthread -D_GNU_SOURCE -std=c18 -Iinclude/ -Ithirdparty/dist/include/ -Daarch64 -Wl,--export-dynamic -ldl -lm -Lthirdparty/dist/lib/ -DJSMN_STATIC -DJSMN_STRICT -L/usr/lib/ src/gen
eric_thread.c src/http_parser_settings.c src/global_request_scheduler.c src/global_request_scheduler_deque.c src/worker_thread.c src/admissions_control.c src/local_runqueue_minheap.c src/env.c src/sc
heduler.c src/http_total.c src/sandbox_state.c src/sandbox_request.c src/module.c src/arch_context.c src/current_sandbox.c src/admissions_info.c src/main.c src/local_runqueue.c src/http_request.c src
/module_database.c src/local_runqueue_list.c src/listener_thread.c src/software_interrupt.c src/runtime.c src/global_request_scheduler_minheap.c src/local_completion_queue.c src/sandbox.c src/arch/aa
rch64/env.c src/libc/syscall.c src/memory/common.c src/memory/64bit_nix.c thirdparty/dist/lib/http_parser.o -o bin/sledgert
src/generic_thread.c:2:10: fatal error: 'threads.h' file not found
#include <threads.h>
         ^~~~~~~~~~~
1 error generated.
In file included from src/http_parser_settings.c:5:
In file included from include/sandbox_types.h:9:
In file included from include/arch/context.h:3:
In file included from include/arch/common.h:15:
include/software_interrupt.h:11:10: fatal error: 'threads.h' file not found
#include <threads.h>
         ^~~~~~~~~~~
1 error generated.
In file included from src/global_request_scheduler.c:3:
In file included from include/global_request_scheduler.h:5:
In file included from include/sandbox_request.h:12:
In file included from include/module.h:9:
In file included from include/admissions_info.h:3:
In file included from include/perf_window_t.h:5:
In file included from include/lock.h:7:
In file included from include/runtime.h:9:
include/types.h:5:10: fatal error: 'threads.h' file not found
#include <threads.h>
         ^~~~~~~~~~~
1 error generated.
In file included from src/global_request_scheduler_deque.c:1:
In file included from include/global_request_scheduler.h:5:
In file included from include/sandbox_request.h:12:
In file included from include/module.h:9:
In file included from include/admissions_info.h:3:
In file included from include/perf_window_t.h:5:
In file included from include/lock.h:7:
In file included from include/runtime.h:9:
include/types.h:5:10: fatal error: 'threads.h' file not found
#include <threads.h>
         ^~~~~~~~~~~
1 error generated.
src/worker_thread.c:7:10: fatal error: 'threads.h' file not found
```

...

Docker images of SLEdge:

```
[mydev@fedora /]$ sudo docker images
[sudo] password for mydev:
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
sledge                        latest    bf842c7aa9cf   33 hours ago     2.76GB
sledge-dev                    latest    82e1fa9f2096   36 hours ago     2.74GB
ubuntu                        bionic    9b6441fe3d04   6 days ago       56.6MB
alpine                        latest    ae607a46d002   2 months ago     5.33MB
busybox                       latest    d9d6c2bcb750   4 months ago     1.4MB
multiarch/qemu-user-static    latest    a217e72a8293   9 months ago     280MB
hello-world                   latest    a29f45ccde2a   21 months ago    9.14kB
[mydev@fedora /]$
```

$SRC_SLEDGE/devenv.sh:

Executable File | 199 lines (165 sloc) | 6.03 KB | Raw

```
#!/bin/sh

# This environment file and dockerfile are inspired by lucet's devenv_xxx.sh scripts

# Root directory of host
HOST_ROOT=${HOST_ROOT:-$(cd "$(dirname "${BASH_SOURCE:-$0}")" && pwd)}

# Name use to represent the SLEdge system
SYS_NAME='sledge'

# /sledge
HOST_SYS_MOUNT=${HOST_SYS_MOUNT:-"/${SYS_NAME}"}

# sledge
SYS_DOC_NAME=${SYS_NAME}

# sledge-dev
SYS_DOC_DEVNAME=${SYS_DOC_NAME}'-dev'

# Docker Tag we want to use
SYS_DOC_TAG='latest'

# The name of the non-dev Docker container that we want to build. sledge:latest
SYS_DOC_NAMETAG=${SYS_DOC_NAME}:${SYS_DOC_TAG}
SYS_DOC_DEVNAMETAG=${SYS_DOC_DEVNAME}:${SYS_DOC_TAG}

# An optional timeout that allows a user to terminate the script if sledge-dev is detected
SYS_BUILD_TIMEOUT=0

# Provides help to user on how to use this script
usage() {
    echo "usage $0 <setup||run||stop||rm||rma/>"
    echo "      setup   Build a sledge runtime container and sledge-dev, a build container with toolchain needed to compile your own functions"
    echo "      run     Start the sledge Docker image as an interactive container with this repository mounted"
    echo "      stop    Stop and remove the sledge Docker container after use"
    echo "      rm      Remove the sledge runtime container and image, but leaves the sledge-dev container in place"
    echo "      rma     Removes all the sledge and sledge-dev containers and images"
}
```

...

2.2 Build SLEdge natively(without Docker)

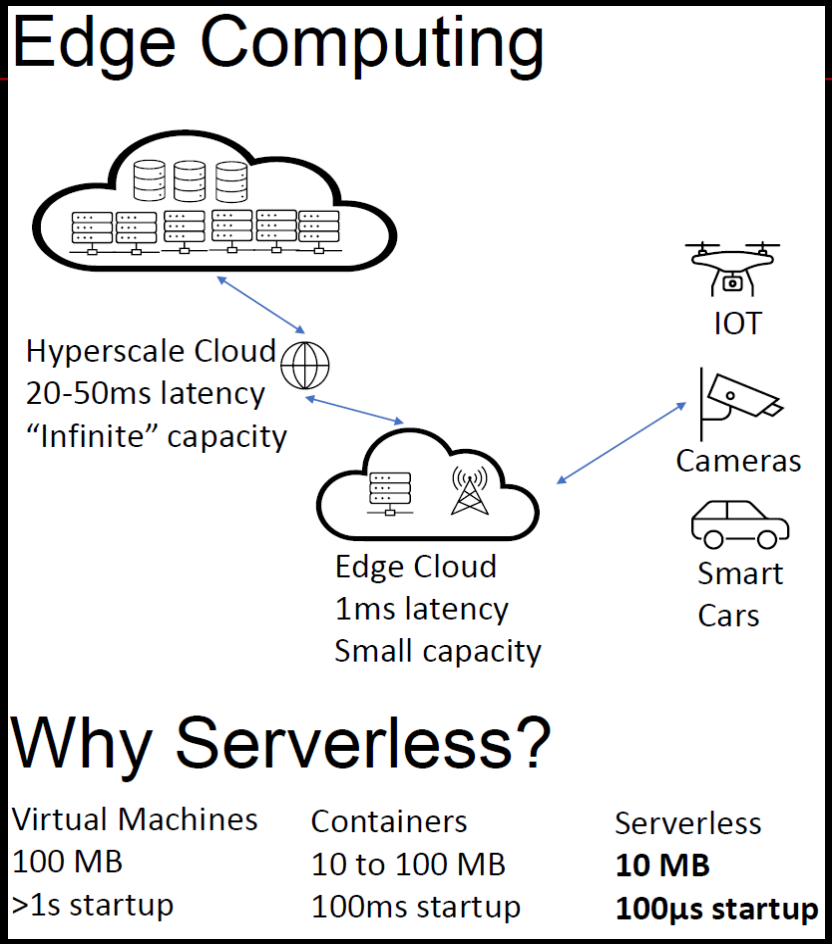
- After replace **Wasmception** with **WASI SDK…**
- Note the latest progress on the **wasi-abstract-interface** branch of aWsm

Sucessfully 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
```

IV. Rethinking Serverless for IoT Edge

1) Why Serverless?



Source: https://conix.io/wp-content/uploads/pubs/3878/CONIX-Sledge-Poster.pdf

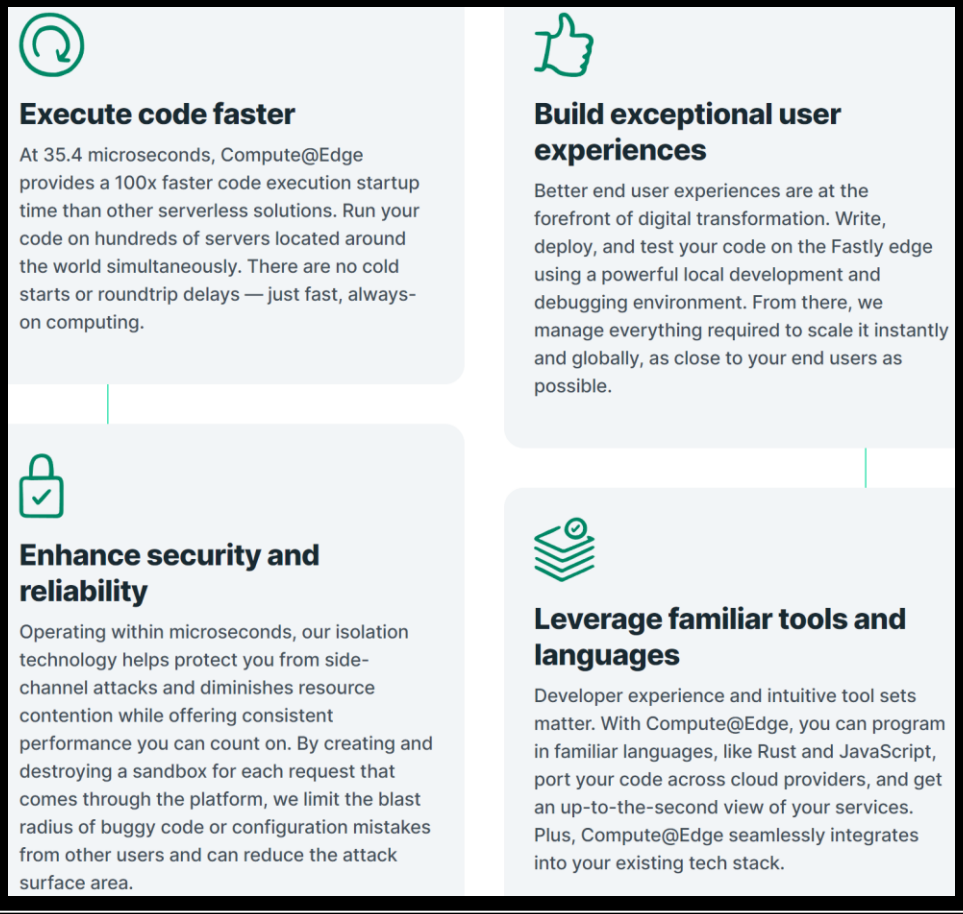
1.1 Fastly & Bytecode Alliance

Fastly Compute@Edge

- https://www.fastly.com/products/edge-compute/serverless

Faster, simpler, and more secure serverless code.

- Benefits



Execute code faster

At 35.4 microseconds, Compute@Edge provides a 100x faster code execution startup time than other serverless solutions. Run your code on hundreds of servers located around the world simultaneously. There are no cold starts or roundtrip delays — just fast, always-on computing.

Build exceptional user experiences

Better end user experiences are at the forefront of digital transformation. Write, deploy, and test your code on the Fastly edge using a powerful local development and debugging environment. From there, we manage everything required to scale it instantly and globally, as close to your end users as possible.

Enhance security and reliability

Operating within microseconds, our isolation technology helps protect you from side-channel attacks and diminishes resource contention while offering consistent performance you can count on. By creating and destroying a sandbox for each request that comes through the platform, we limit the blast radius of buggy code or configuration mistakes from other users and can reduce the attack surface area.

Leverage familiar tools and languages

Developer experience and intuitive tool sets matter. With Compute@Edge, you can program in familiar languages, like Rust and JavaScript, port your code across cloud providers, and get an up-to-the-second view of your services. Plus, Compute@Edge seamlessly integrates into your existing tech stack.

Key Features

Data

- Device detection
- Geolocation
- Edge database
- Powerful caching

Observability

- Real-time logging
- Customizable stats
- Log tailing
- Tracing support

Tooling

- Full command line control
- User Interface
- Intuitive Application Programming Interface
- Terraform API support
- Rust language
- AssemblyScript language (Beta)
- JavaScript language (Beta)

Develop in Rust (our most mature language) and compile to WebAssembly

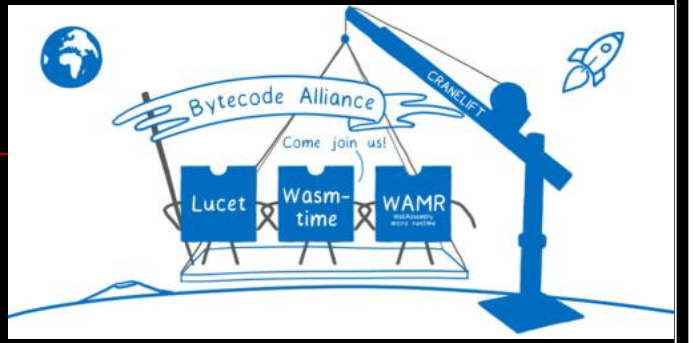
Bytecode Alliance

- https://bytecodealliance.org/

The Bytecode Alliance is a nonprofit organization dedicated to creating secure new software foundations, building on standards such as WebAssembly and WebAssembly System Interface (WASI).

The Bytecode Alliance is committed to establishing a capable, secure platform that allows application developers and service providers to confidently run untrusted code, on any infrastructure, for any operating system or device, leveraging decades of experience doing so inside web browsers.

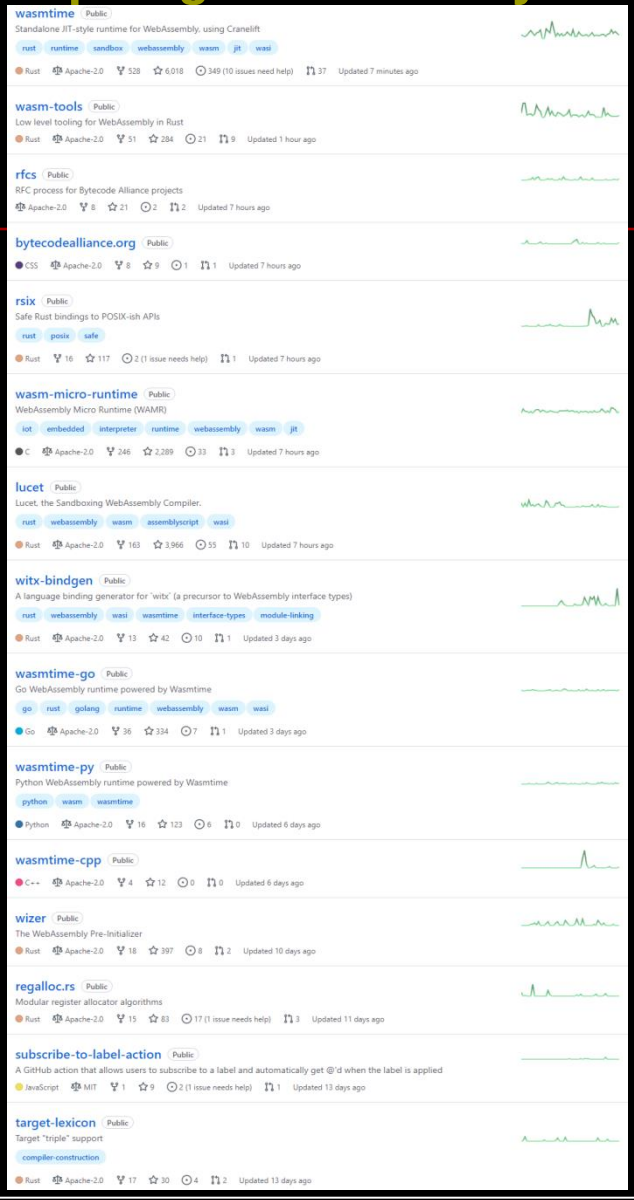
We have a vision for a secure-by-default WebAssembly ecosystem for all platforms.



- Members

arm, embark, fastly, Google, InfinyOn, igalia, intel, Microsoft, moz://a, Profian, SIEMENS, shopify, StackBlitz, SUBORBITAL

- https://github.com/bytecodealliance/



wasmtime Public
Standalone JIT-style runtime for WebAssembly, using Cranelift
rust runtime sandbox webassembly wasm jit wasi
Rust Apache-2.0 528 6,018 349 (10 issues need help) 37 Updated 7 minutes ago

wasm-tools Public
Low level tooling for WebAssembly in Rust
Rust Apache-2.0 51 284 21 9 Updated 1 hour ago

rfcs Public
RFC process for Bytecode Alliance projects
Apache-2.0 8 21 2 2 Updated 7 hours ago

bytecodealliance.org Public
CSS Apache-2.0 8 9 1 1 Updated 7 hours ago

rsix Public
Safe Rust bindings to POSIX-ish APIs
rust posix safe
Rust 16 117 2 (1 issue needs help) 1 Updated 7 hours ago

wasm-micro-runtime Public
WebAssembly Micro Runtime (WAMR)
iot embedded interpreter runtime webassembly wasm jit
C Apache-2.0 246 2,289 33 3 Updated 7 hours ago

lucet Public
Lucet, the Sandboxing WebAssembly Compiler.
rust webassembly wasm assemblyscript wasi
Rust Apache-2.0 163 3,966 55 10 Updated 7 hours ago

witx-bindgen Public
A language binding generator for 'witx' (a precursor to WebAssembly interface types)
rust webassembly wasi wasmtime interface-types module-linking
Rust Apache-2.0 13 42 10 1 Updated 3 days ago

wasmtime-go Public
Go WebAssembly runtime powered by Wasmtime
go rust golang runtime webassembly wasm wasi
Go Apache-2.0 36 334 7 1 Updated 3 days ago

wasmtime-py Public
Python WebAssembly runtime powered by Wasmtime
python wasm wasmtime
Python Apache-2.0 16 123 6 0 Updated 6 days ago

wasmtime-cpp Public
C++ Apache-2.0 4 12 0 0 Updated 6 days ago

wizer Public
The WebAssembly Pre-Initializer
Rust Apache-2.0 18 397 8 2 Updated 10 days ago

regalloc.rs Public
Modular register allocator algorithms
Rust Apache-2.0 15 83 17 (1 issue needs help) 3 Updated 11 days ago

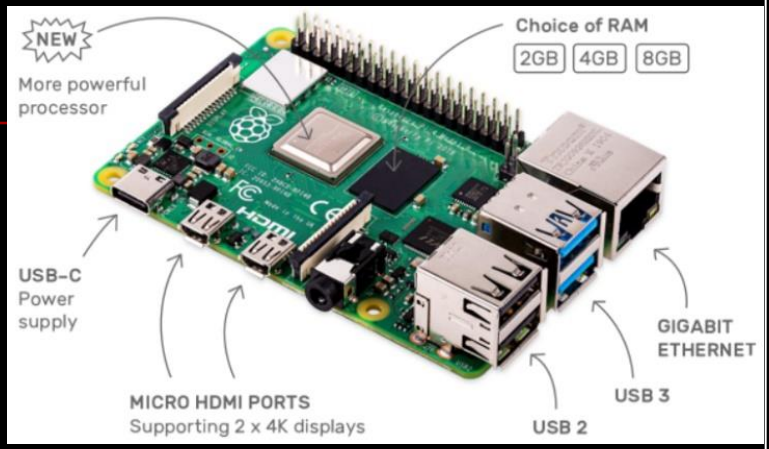
subscribe-to-label-action Public
A GitHub action that allows users to subscribe to a label and automatically get @'d when the label is applied
JavaScript MIT 1 9 2 (1 issue needs help) 1 Updated 13 days ago

target-lexicon Public
Target "triple" support
compiler-construction
Rust Apache-2.0 17 30 4 2 Updated 13 days ago

1.2 Serverless on RPi

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

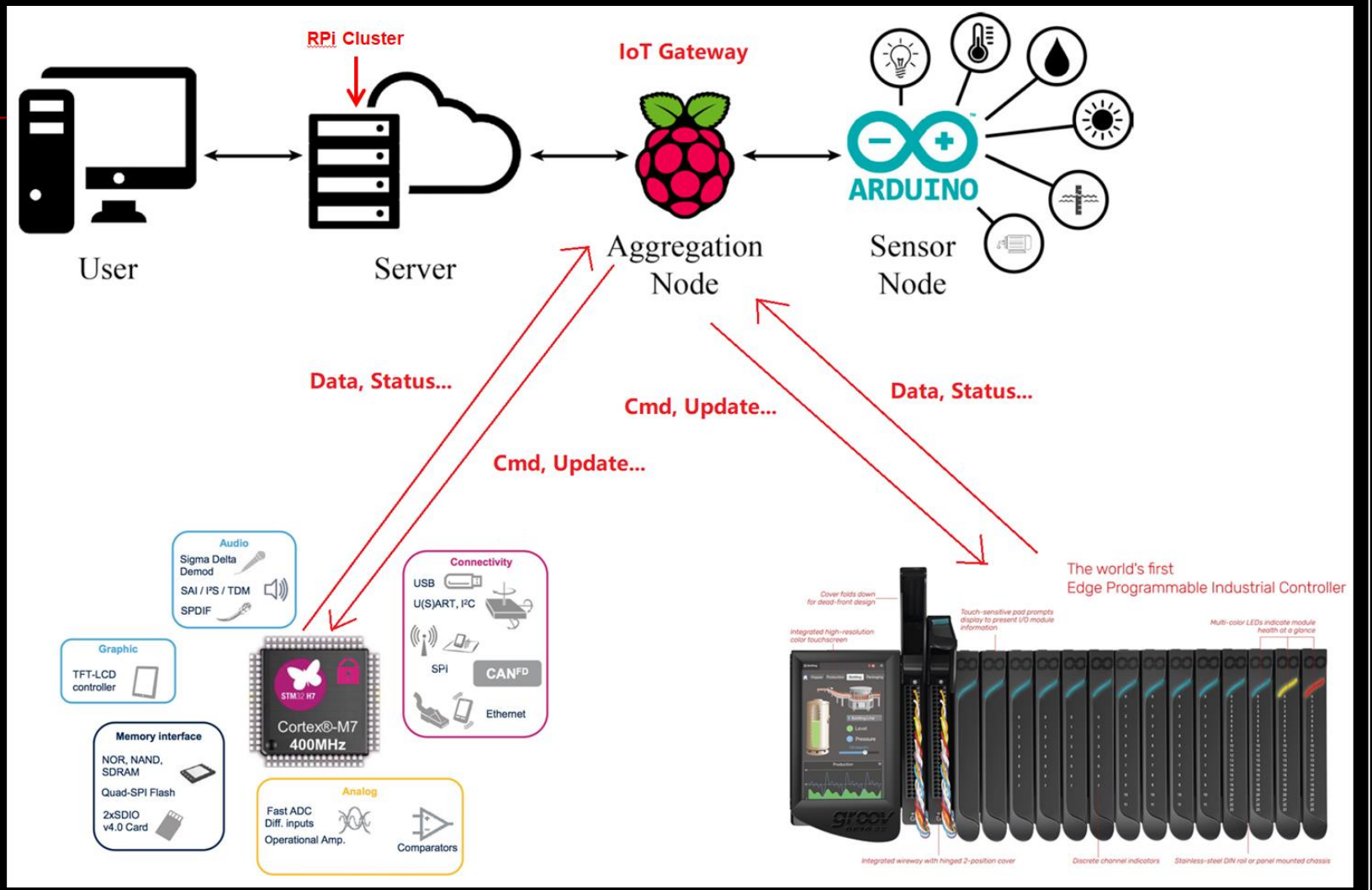


Good Resources

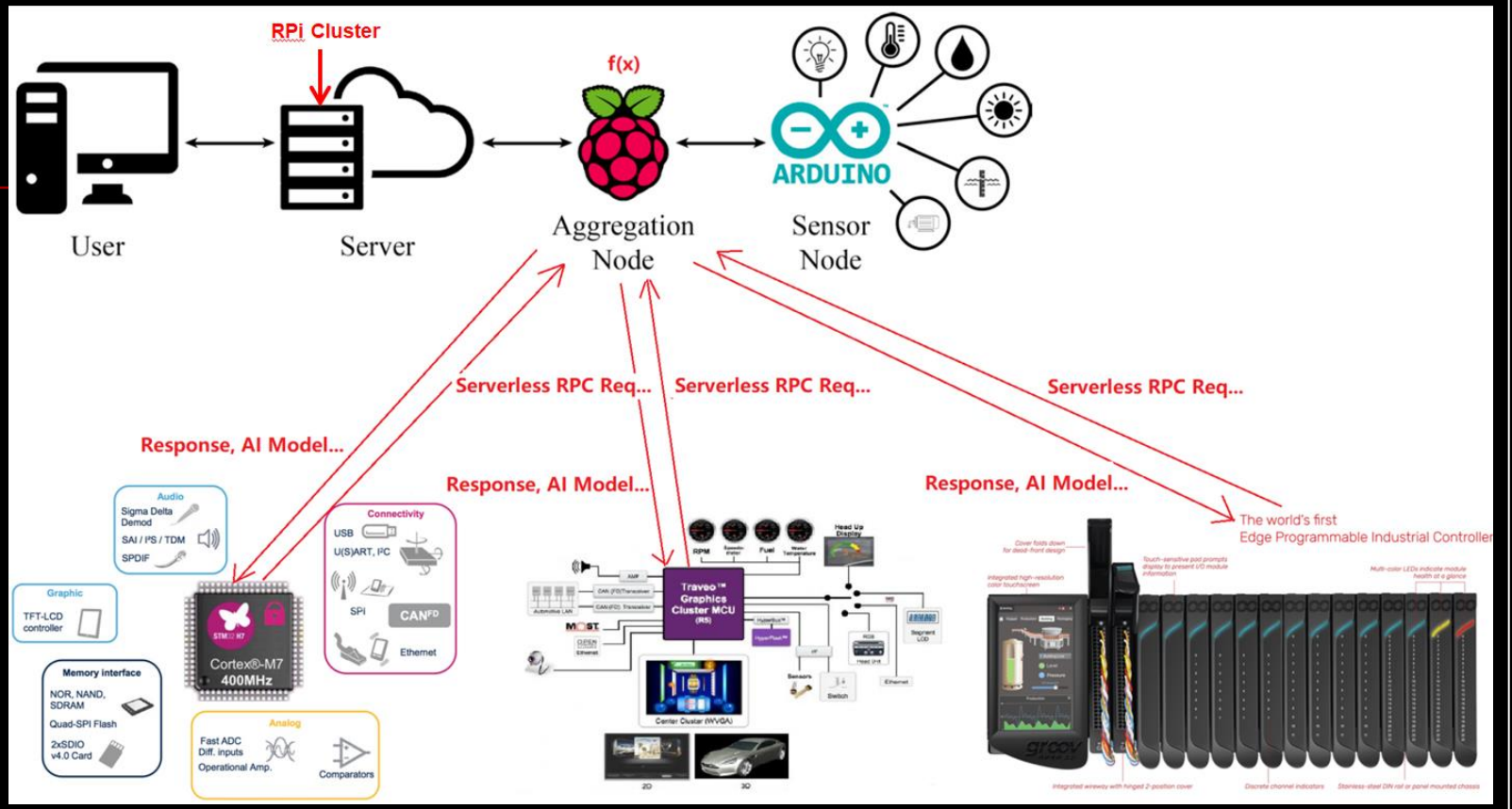
- https://johansiebens.dev/posts/2020/08/a-serverless-appliance-for-your-raspberry-pi-with-faasd/
- https://aws.amazon.com/cn/blogs/compute/building-a-raspberry-pi-telepresence-robot-using-serverless-part-1/
- https://aws.amazon.com/cn/blogs/compute/building-a-raspberry-pi-telepresence-robot-using-serverless-part-2/
- https://jussiroine.com/2021/06/building-a-kubernetes-cluster-using-raspberry-pi-4/
- …

2) Usage scenarios

- General case



Case of TinyML



3) Why aWsm/SLEdge?

Comparison

- **Implementation Methodology**

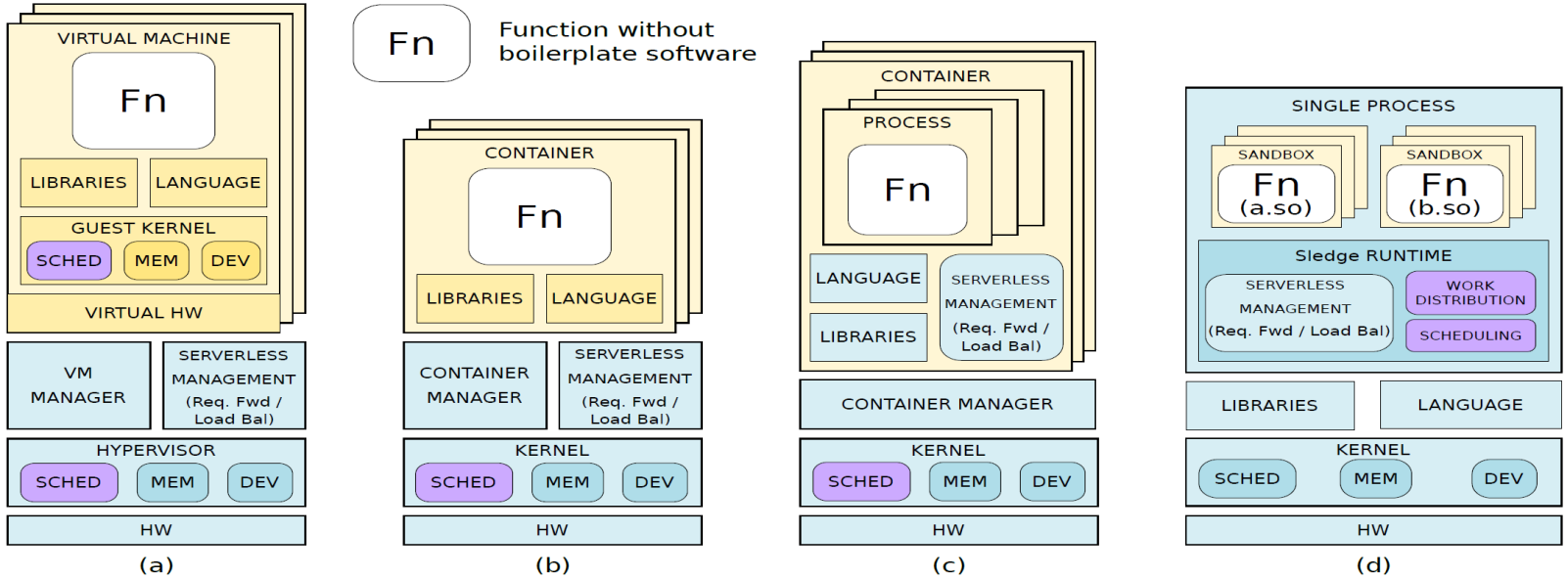


Figure **(a)** VM-based Serverless (e.g., AWS Lambda using Firecrackers, Microsoft Azure Functions using Hyper-V, etc.), **(b)** Container-based Serverless (e.g., OpenWhisk, Google Cloud Functions, etc.), **(c)** Container + Processes-based Serverless (e.g., *Nuclio*, OpenFaaS), **(d)** *Sledge* Approach for Serverless at the Edge.

Source: https://www2.seas.gwu.edu/~gparmer/publications/middleware20sledge.pdf

Support X64/ARM64/Cortex-M and have good performance

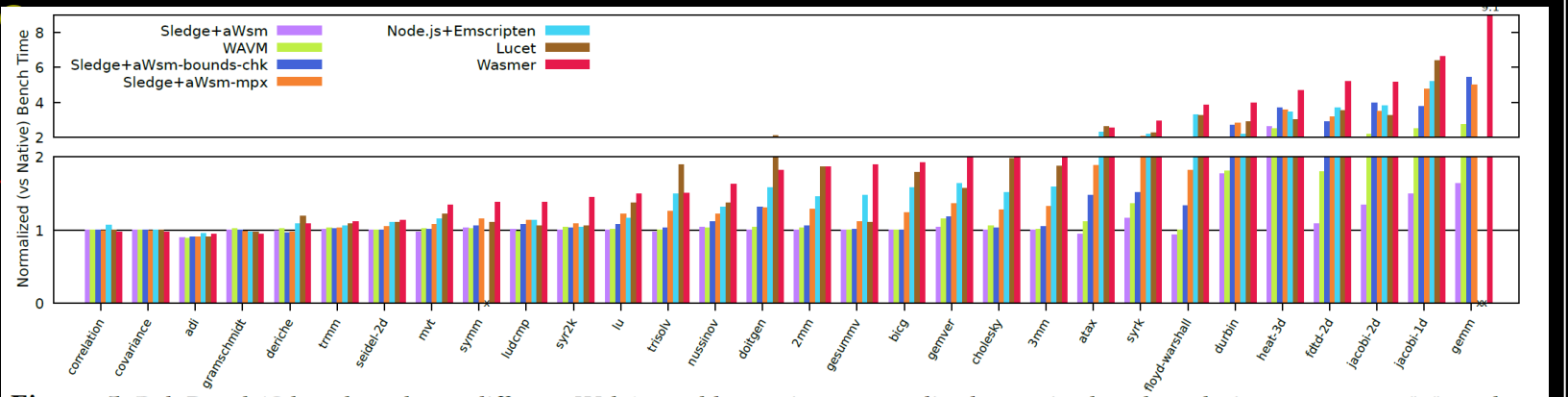


Figure 5. PolyBench/C benchmarks on different WebAssembly runtimes normalized to native benchmark time on x86_64. "x" mark on x-axis indicates failure to execute benchmark on a runtime. Node.js+Emscripten failed for `gemm` and `symm`. Lucet failed for `gemm`.

	x86_64							*Raspberry Pi (ARM v8)*	
Wasm Runtime	Wasmer	WAVM	Node.js+ Emscripten	Lucet	*Sledge+aWsm-bounds-chk*	*Sledge+ mpx*	*Sledge+ aWsm*	*Sledge+aWsm-bounds-chk-rpi*	*Sledge+aWsm-rpi*
Slowdown (AM)	149.8%	28.1%	84.0%	92.8%	62.7%	75.1%	13.4%	36.7%	6.74%
Slowdown (GM)	101.6%	20.5%	62.3%	68.9%	38.4%	51.6%	9.9%	26.86%	5.0%
SD	194.09	53.09	107.84	117.25	116.14	113.41	34.65	60.3	19.38

Table 1. Arithmetic mean (Slowdown (AM)) and geometric mean (Slowdown (GM)) of % slowdowns, and the standard deviations (SD) for arithmetic mean in different Wasm runtimes. The x86_64 results summarize the results from Figure 5. For brevity, we only include summarized results for AArch64 on Raspberry Pi in the right-most columns of the table (systems with -rpi suffix).

Source: https://www2.seas.gwu.edu/~gparmer/publications/middleware20sledge.pdf

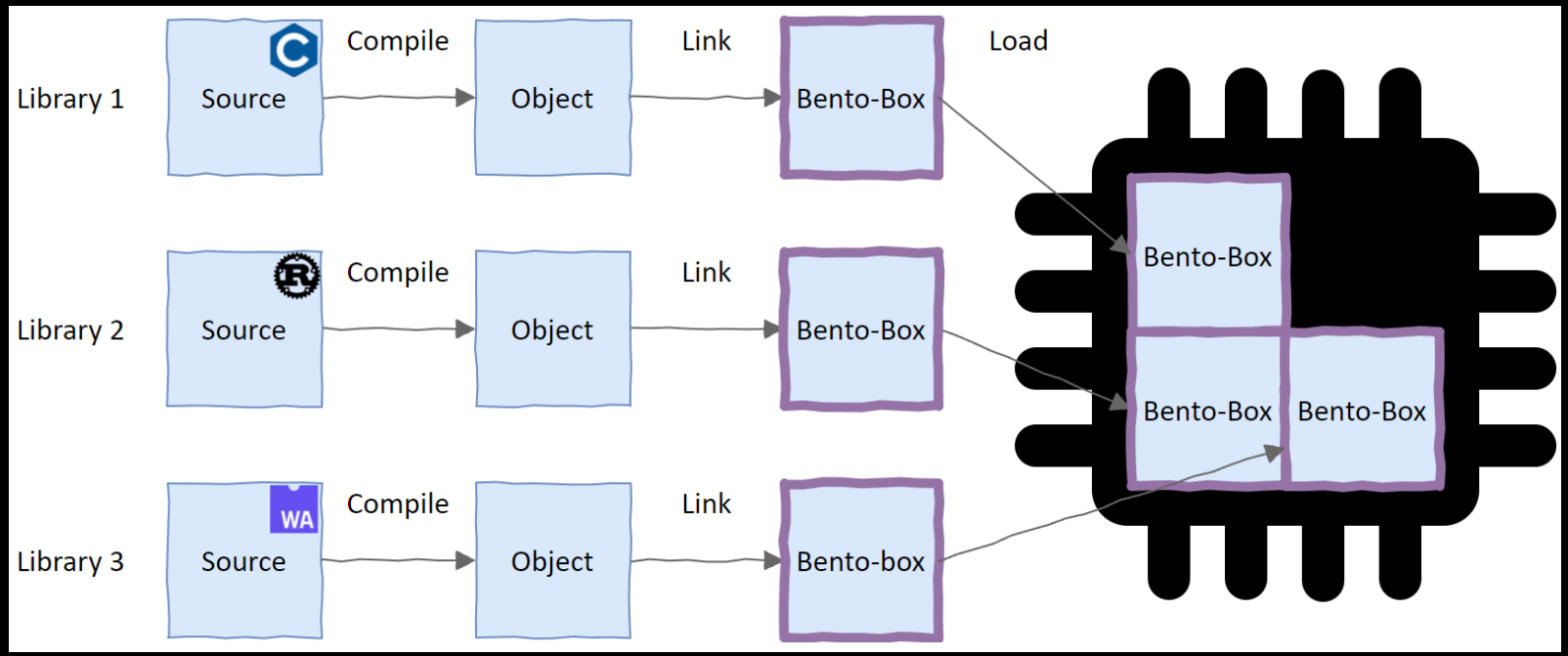
Official application--Project Bento from ARM

- https://github.com/ARM-software/bento-linker

  A light-weight alternative to processes for microcontrollers.

- Bento-boxes

  Independently-linked, memory-isolated pieces of code that are designed to work together. You can think of them as a light-weight alternative to processes for microcontrollers.



Features

Unlike processes, Bento-boxes don't require mutlithreading or virtual memory. Instead of files and pipes, boxes communicate using type-rich inter-box-communication (IBC) mechanisms that behave like familiar C functions with a few limitations.

A Bento-box is described by two things:

1. A set of memory regions + data
2. Function-like imports and exports

When provided by a config that describes these boxes, the bento-linker (this tool) can generate a variety of different glue that enables some pretty powerful features:

1. Automatic handling of box state bringing up/down boxes as needed. This enables RAM-sharing, compressed boxes, or even storing boxes on external storage.
2. Component-level firmware updates. With clearly defined memory regions and standard IBC mechanisms, it's easy to update a single part of the system. This reduces risk and bandwidth cost for rolling out bug-fixes or new features in the field.
3. Hardware or software enforced memory isolation. No more losing the entire device because your CBOR parser had a buffer overflow. If a rogue box tries to escape its allocated memory regions it is killed and an error is politely returned to the caller.

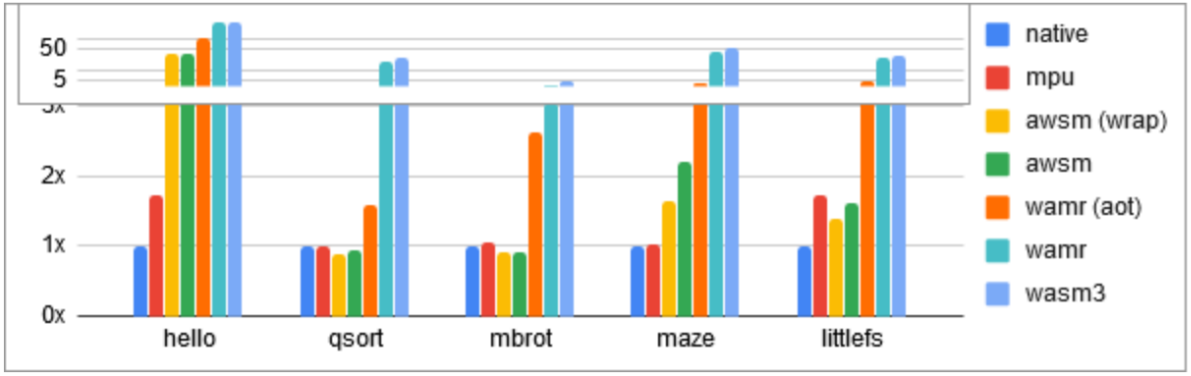
And some less powerful but nice features:

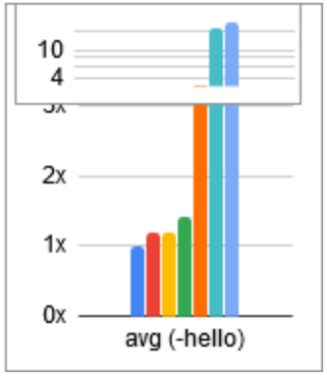
1. Programming language interoperability. The requirements for imports and exports are the same as the requirements for FFI and can be automatically generated.
2. A good target for LTO, balancing optimizations and build time.

Results

Runtime (in ns):

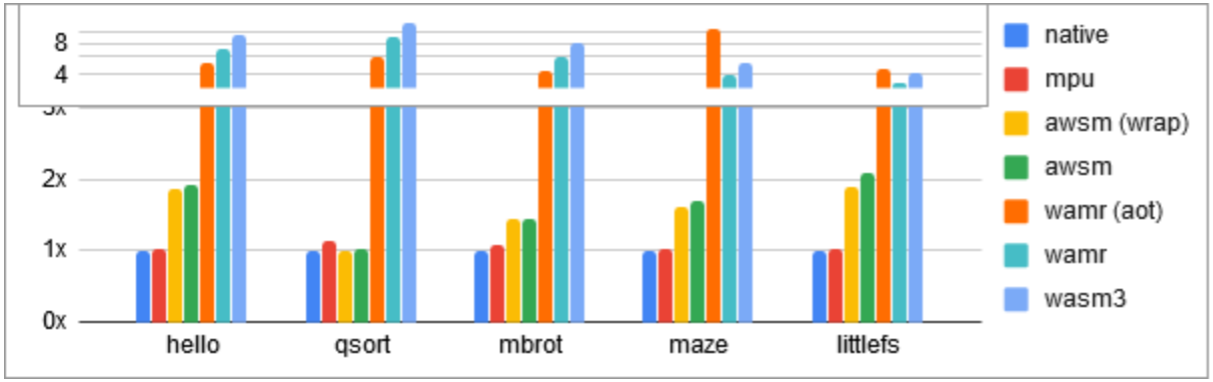
	hello	qsort	mbrot	maze	littlefs
native	271 ns	84287 ns	2186562 ns	3118464 ns	3850311 ns
mpu	463 ns	84851 ns	2275587 ns	3203449 ns	6654519 ns
awsm (wrap)	9527 ns	75256 ns	1979281 ns	5106498 ns	5320550 ns
awsm	8508 ns	78896 ns	1988887 ns	6823538 ns	6150274 ns
wamr (aot)	28440 ns	132688 ns	5733677 ns	12368716 ns	17249899 ns
wamr	93229 ns	1551582 ns	6608421 ns	121541011 ns	103575732 ns
wasm3	101678 ns	2079359 ns	8844360 ns	157946032 ns	114494192 ns

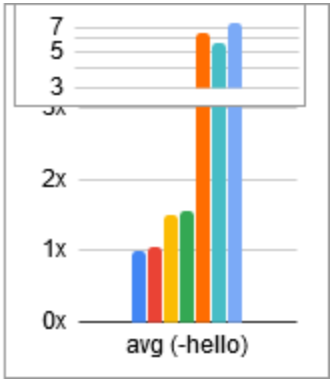




Code size (in bytes):

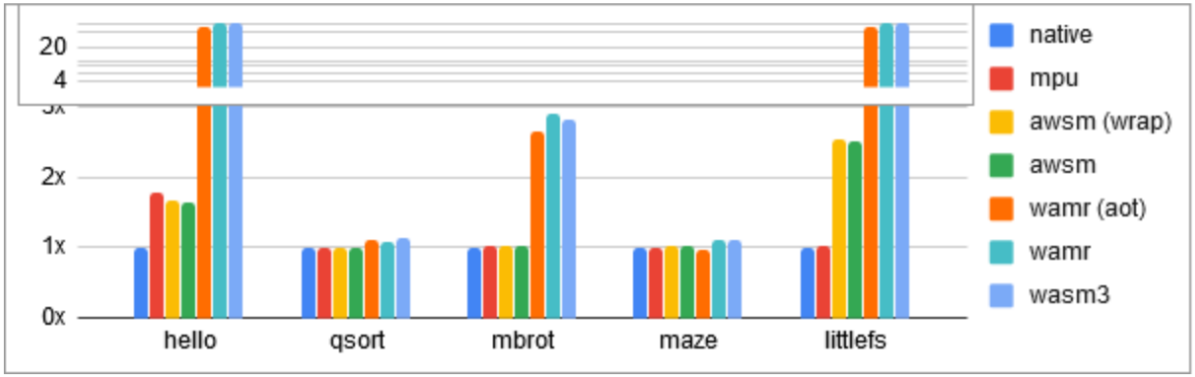
	hello	qsort	mbrot	maze	littlefs
native	7616 B	5228 B	8532 B	16052 B	22944 B
mpu	7900 B	5896 B	9216 B	16340 B	23228 B
awsm (wrap)	14152 B	5268 B	12368 B	25672 B	43428 B
awsm	14664 B	5280 B	12240 B	27208 B	47524 B
wamr (aot)	39772 B	30828 B	36816 B	176156 B	102848 B
wamr	53160 B	47567 B	50117 B	63582 B	77792 B
wasm3	72000 B	66143 B	68725 B	82382 B	96512 B

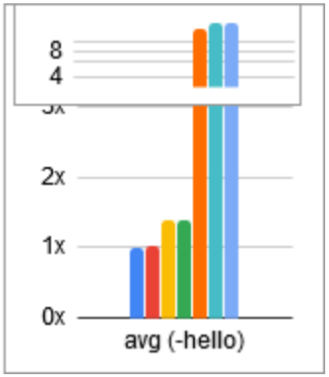




RAM lower-bound (in bytes):

	hello	qsort	mbrot	maze	littlefs
native	304 B	40772 B	6772 B	43940 B	1468 B
mpu	540 B	40860 B	6908 B	44240 B	1496 B
awsm (wrap)	504 B	40692 B	7000 B	44652 B	3712 B
awsm	500 B	40692 B	7000 B	44672 B	3664 B
wamr (aot)	15348 B	44700 B	17796 B	43084 B	77988 B
wamr	19028 B	43660 B	19564 B	48116 B	91668 B
wasm3	18588 B	45968 B	19048 B	49188 B	94176 B





4) Possible improvements

Considering potential improvements for aWsm/SLEdge

- Completely replace Wasmception with WASI SDK
- Porting aWsm to Cortex-M33/Cortex-M55 etc

- Porting Bento to Cortex-M33/Cortex-M55 etc
- Extending SLEdge for TinyML
- Implement a specific lightweight orchestrator for WASM
- Try to combine aWsm and GraaVM's Native Image
- Customize a new version of GraalVM & Quarkus for WASM
- Rewriting AOT compiler by some new languages
- …

Please look forward to our follow-ups like "Revisiting AOT compilation based Wasm compiler and runtime for Serverless Edge computing"…

V. 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.
- Wasm and eBPF are the two key points of the infrastructure for tomorrow's Edge Computing.
- What most preferred is a specific lightweight orchestrator that beyond Kubernetes for running WASM in the future…

CHINA
OpenInfra Days

Thanks!

Feng Li (李枫)
hkli2012@126.com

Reference

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/
- http://web.cse.ohio-state.edu/~pouchet.2/software/polybench/
- https://en.wikipedia.org/wiki/Execute_in_place
- …