BUILDING RELIABLE DISTRIBUTED EDGE-CLOUD APPLICATIONS WITH WEBASSEMBLY

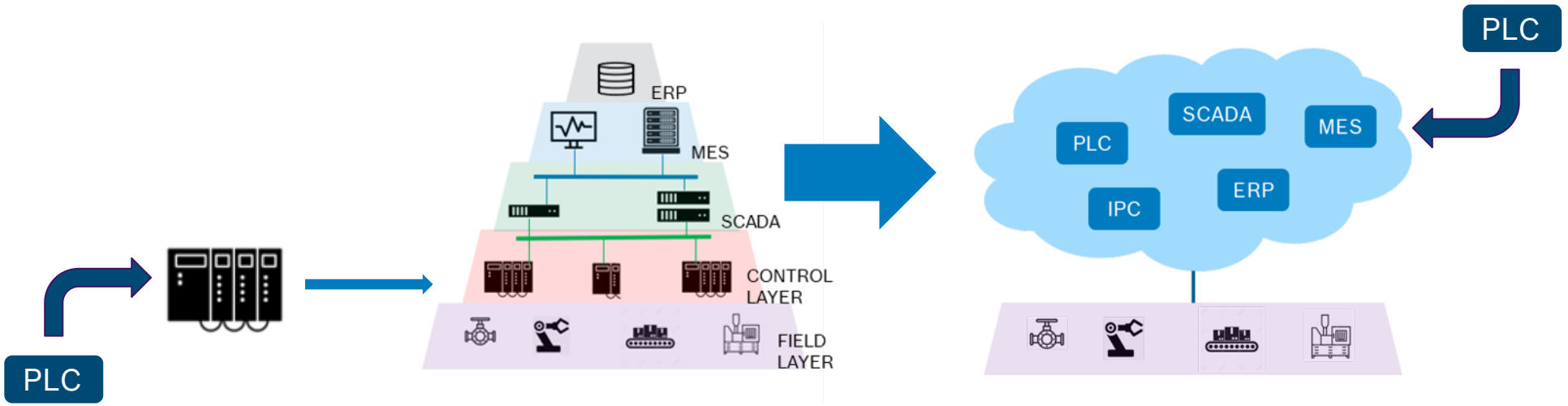
RT-CLOUD 2022

Franz-Josef Grosch, Dakshina Dasari,
Nuno Pereira, Anthony Rowe

BOSCH

Distributed Edge-Cloud Applications
Industrial production – software-defined manufacturing

Leverage **edge-cloud computing** for modular, scalable and reconfigurable real-time control

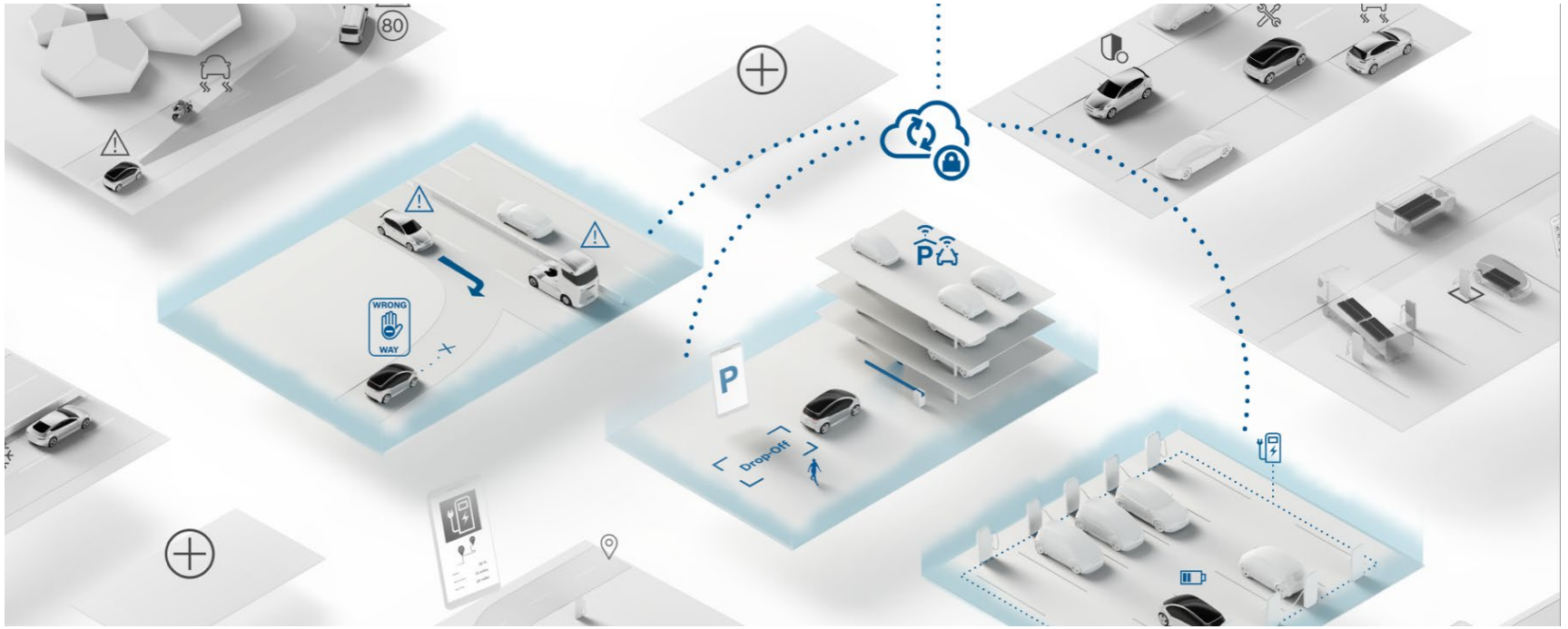


PLC: Programmable Logic Controller – usually hard real-time systems

SCADA: Supervisory Control and Data Acquisition – MES: Manufacturing Execution System – ERP: Enterprise Resouce Planning – IPC: Industrial Purposes Computer

Distributed Edge-Cloud Applications
Car of the future – software-defined vehicle

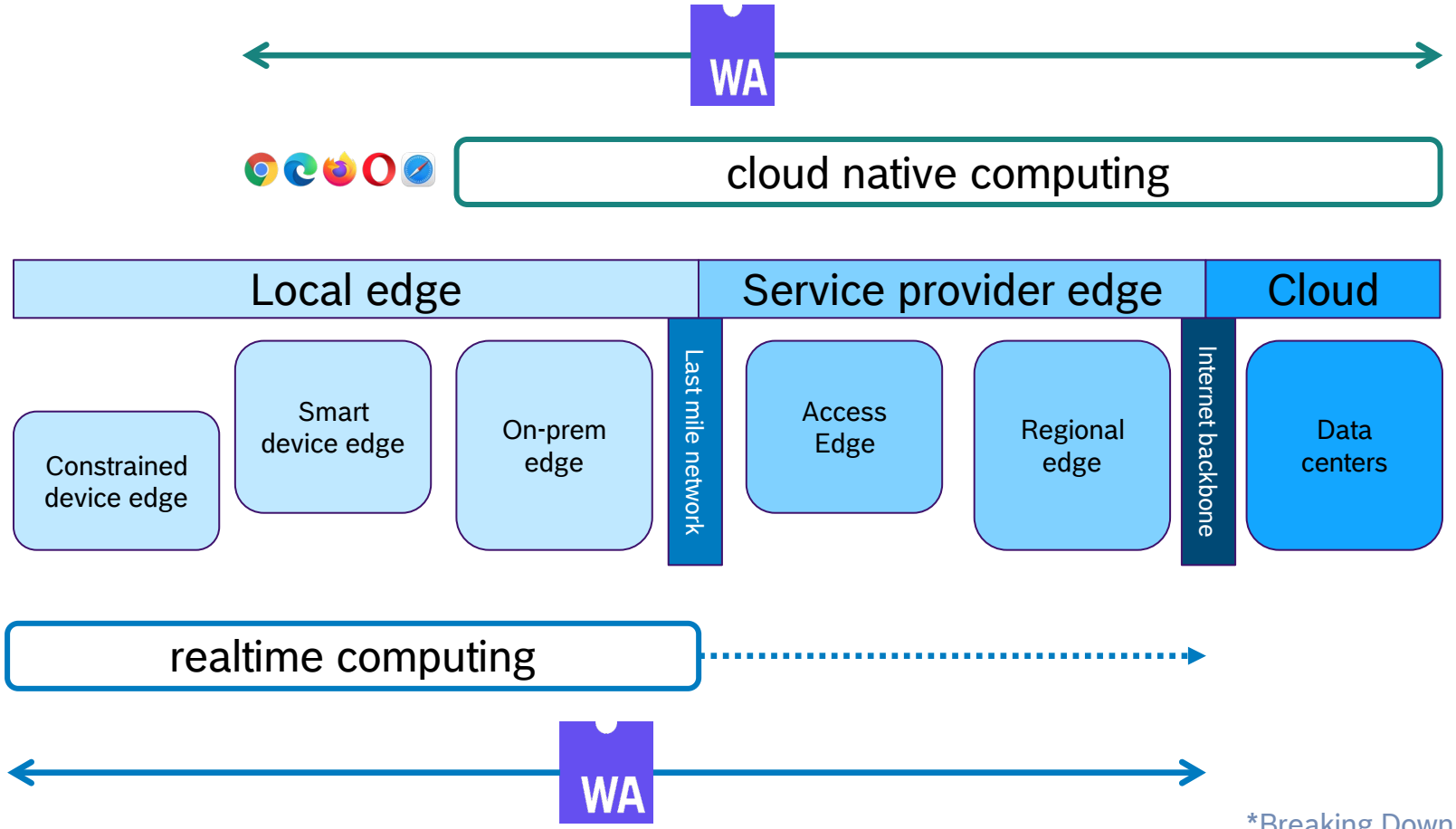
Leverage **edge-cloud computing** for infrastructure-supported driver-assistance



Driver-assistance: Usually safety-critical, real-time systems

*Source: Bosch software-defined vehicle

Distributed Edge-Cloud Applications
Cloud native meets realtime computing



*Breaking Down the Edge Continuum - LF Edge

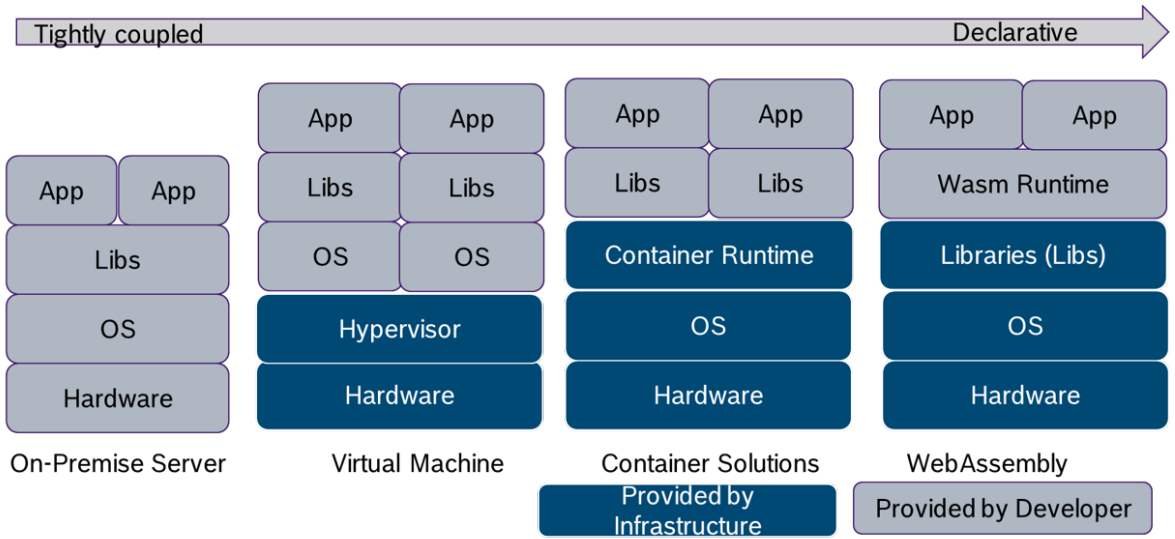
Grosch | Dasari | Pereira | Rowe | 2022-07-05

Distributed Edge-Cloud applications
WebAssembly virtualisation

	Wasm	Containers	Virtual machines
Memory efficiency	Good (KB)	Medium (MB)	Poor (GB)
Cold startup time	Good (usec)	Medium (msec)	Poor (sec)
Live migration	Good (seconds)	Poor (n/a)	Medium (minutes)
Targets	Cloud, edge, device	Cloud, edge	Cloud, edge

WebAssembly is lightweight

WebAssembly is loosely coupled



WHAT IS WEBASSEMBLY AKA WASM?

What is WebAssembly aka Wasm?
Byte code for a stack-based virtual machine

A **binary instruction format** for a **stack-based virtual machine**.
A portable compilation target for arbitrary programming languages.
Enabling application deployment on any modern hardware*

*Why WebAssembly? | by Andreas Rossberg | Medium

- Fast, safe and portable semantics
  - Fast to execute
  - Safe to execute
  - Well-defined – easy to reason about
  - Hardware-independent
  - Language-independent
  - Platform-independent
  - Open – to interoperate

- Efficient and portable representation
  - Compact
  - Modular
  - Efficient – decode, validate, compile
  - Streamable
  - Parallelizable
  - Portable

What is WebAssembly aka Wasm?
Wasm is a typed programming language*

- Basics
  - A binary is a **module** defining functions, globals, tables and memory
  - Definitions can be **imported** and **exported**
- Memory
  - One linear memory per module
  - Memory grows by pages (64KiB)
  - Out-of memory access traps
- Control flow
  - Block with return, loop, if
  - No unstructured control flow
- Function calls
  - Direct
  - Indirect via table, dynamically validated
  - Foreign calls for **imported host functions**

*Bringing the web up to speed with WebAssembly | PLDI 2017

What is WebAssembly aka Wasm?
Wasm is a stack-machine with typed instructions

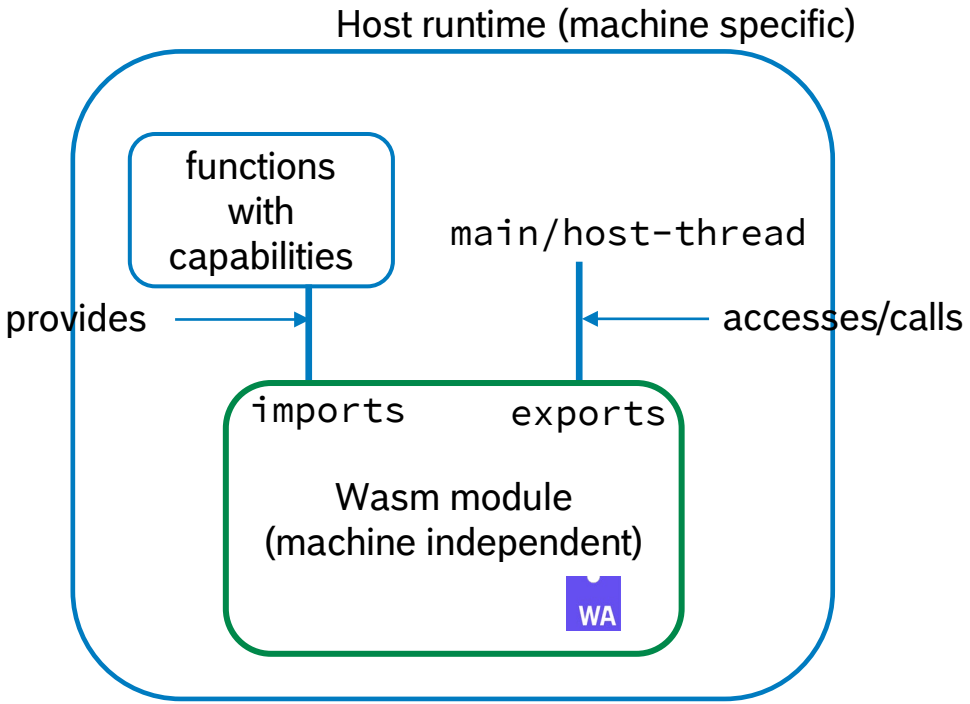
- Fibonacci function in C

```c
int fib (int n)
{
    if (n < 2) {
        return 1;
    }
    return fib(n-2) + fib(n-1);
}
```

- Fibonacci function in Wasm

```
(module
  (type (;0;) (func (param i32) (result i32)))
  (func $fib (type 0) (param $n i32) (result i32)
    local.get $n
    i32.const 2
    i32.lt_s
    if                ;; n < 2
      i32.const 1
      return
    end
    local.get $n
    i32.const 2
    i32.sub
    call $fib         ;; fib(n-2)
    local.get $n
    i32.const 1
    i32.sub
    call $fib         ;; fib(n-1)
    i32.add
    return)
  (export "fib" (func $fib)))
```

What is WebAssembly aka Wasm?
Wasm execution requires a host runtime



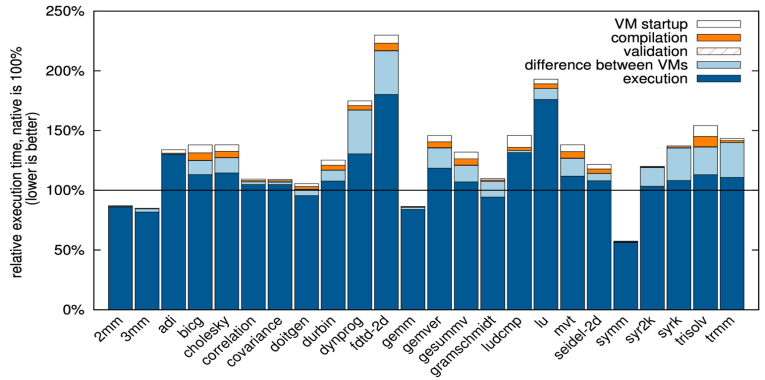
- Wasm **modules** provide
  - Sandboxing
    - A module can only interact with its environment through its **imports** which are **provided** by a client, so that the client has full control over the **capabilities** given to a module.
    - A module **without imports** cannot do anything, besides burning computation cycles.
  - Encapsulation
    - A module's client can only **access** the **exports** of a module, other internals are protected from tampering.
    - A module **without exports** cannot do anything, besides allocating its initial memory.

HOW WASM CHANGES THE GAME FOR SAFETY-CRITICAL REAL-TIME APPLICATIONS

BOSCH

Changing the game
Wasm is consistently fast

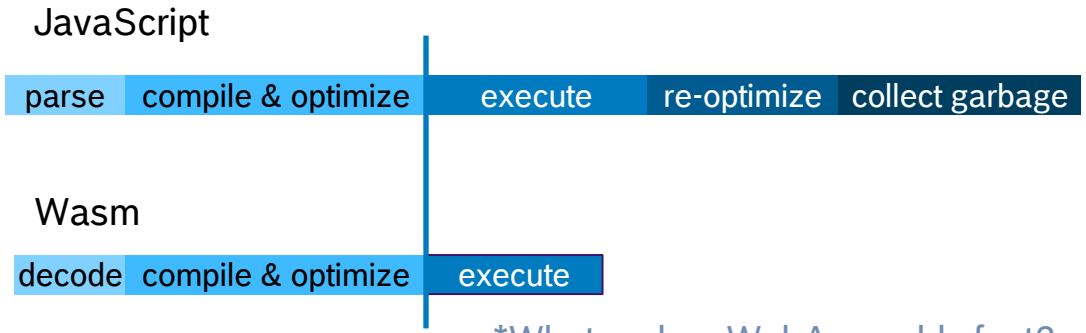
- JavaScript, C++, Wasm – all are fast



*Bringing the Web Up to Speed with WebAssembly

- Wasm execution
  - Interpreted
  - Just-in-time compiled
  - Ahead-of-time compiled

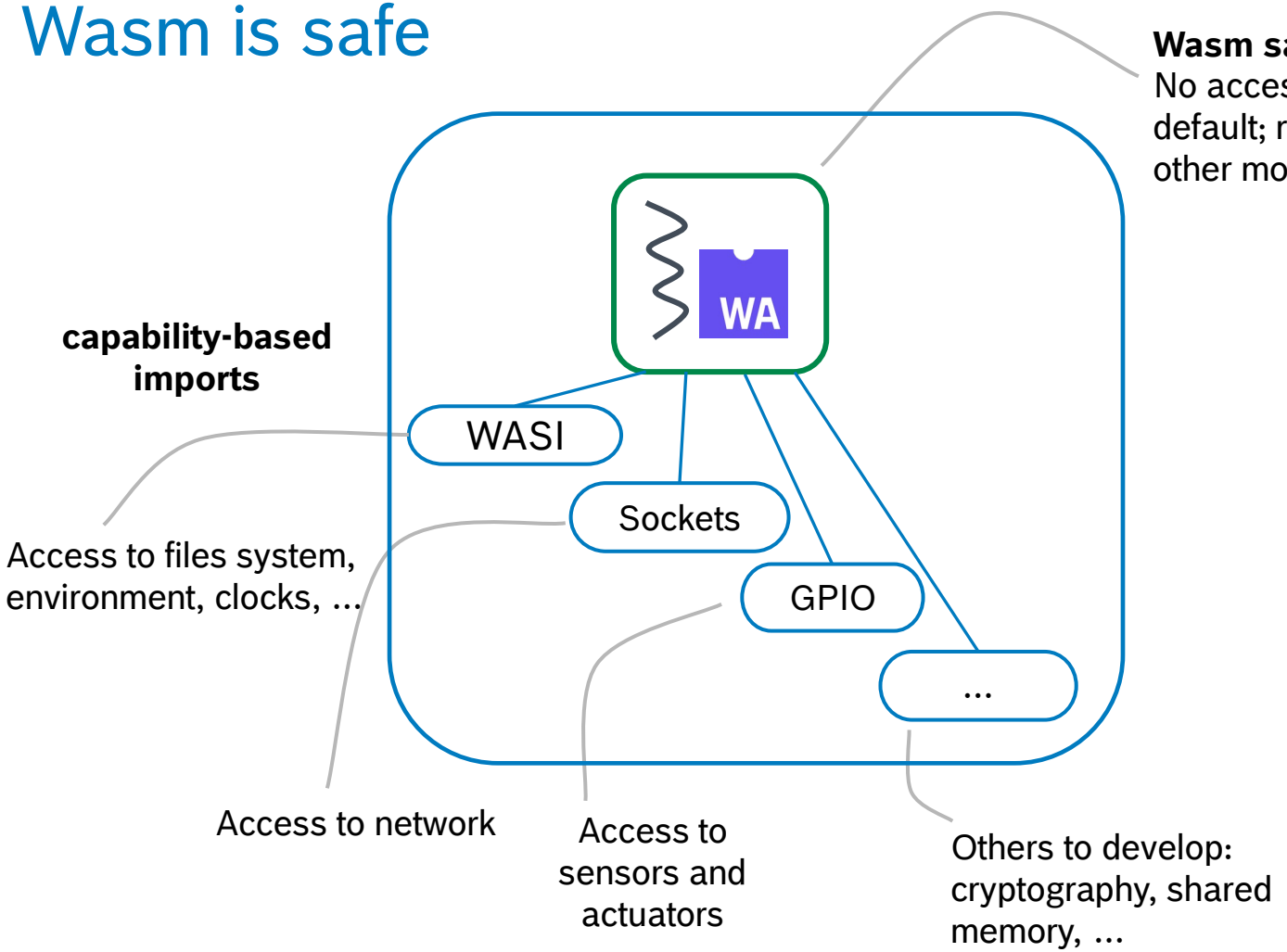
- Wasm is consistently fast



*What makes WebAssembly fast?

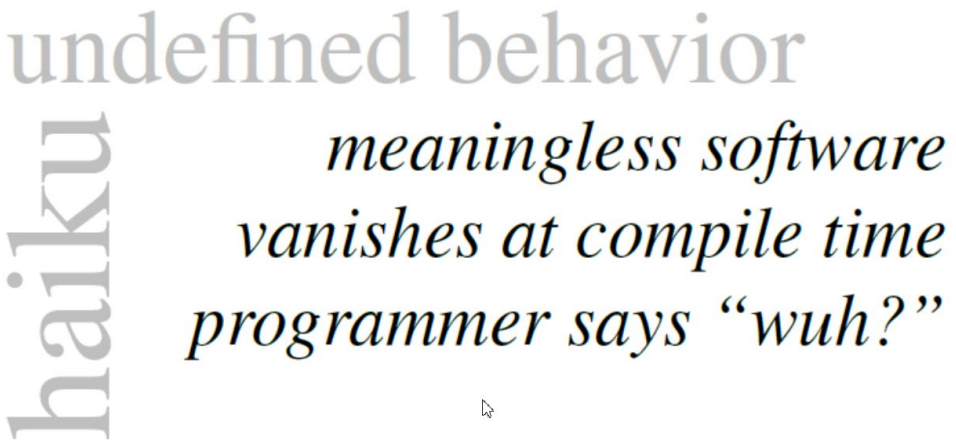
  - Decode, compile + optimize at startup
  - No re-optimize
  - Garbage collection will be optional
  - Fits well to real-time applications

Changing the game
Wasm is safe



- Wasm sandbox
  - Memory safety
  - Control flow integrity
  - Fault isolation
  - No access to code addresses and the call stack
- Capability-based imports
  - Standardized Wasm system interface (WASI)
  - System safety
  - Isolate untrusted or buggy code

Changing the game
Wasm is well-defined and deterministic



Schrödinger's Code - ACM Queue

- Wasm is well-defined
  - No undefined behaviour
  - No implementation-defined behavior
  - No machine-dependent behaviour
  - No unspecified behavior
  - Well-defined traps – e.g. division by zero
  - No invalid calls
  - No illegal access to data
- Wasm is deterministic
  - Any program, on any machine
  - NaN representation needs normalisation
  - Threads will be optional
    - Well-defined memory model on the way*

*Weakening WebAssembly | OOPSLA 19

Changing the game
Wasm is polyglot

- Compile safety-critical, real-time applications from low-level languages
- Compile best-effort applications from high-level languages
- Run prototyped applications from dynamic languages

- Link applications from components written in different languages

Say goodbye to the C/C++ stranglehold

Changing the game
Wasm is open

- Open design
  - Designed cooperatively by the 4 major browser vendors (Google, Mozilla, Apple, Microsoft)
  - Design process open to the public
  - Defined by an open standard
  - Anybody can use it, implement it, contribute to it
  - Avoids licensing, copyrighting, or patenting problems

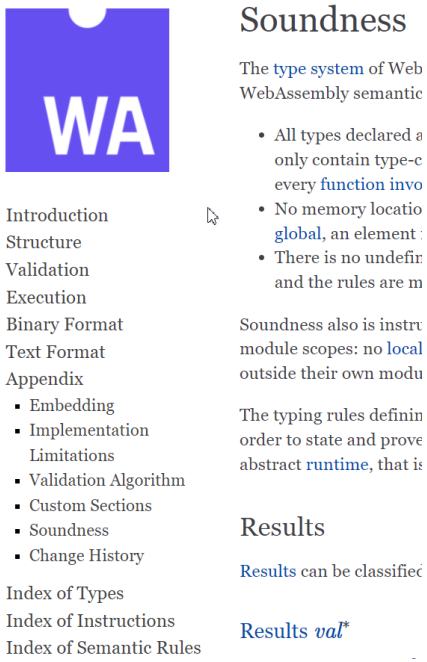
W3C WebAssembly Working Group

- Open interfacing
  - Useful for any environment, not only the Web
  - More WASI features
    - system access, networking, tensor flow
  - Upcoming Wasm features
    - threads, garbage collection, exceptions, ...
  - Use consistent Wasm feature sets for your domain
  - Build/use the runtime for your domain

How WASI Makes Containerization More Efficient

Open for realtime requirements ?

Changing the game
Wasm is formally defined and provably correct



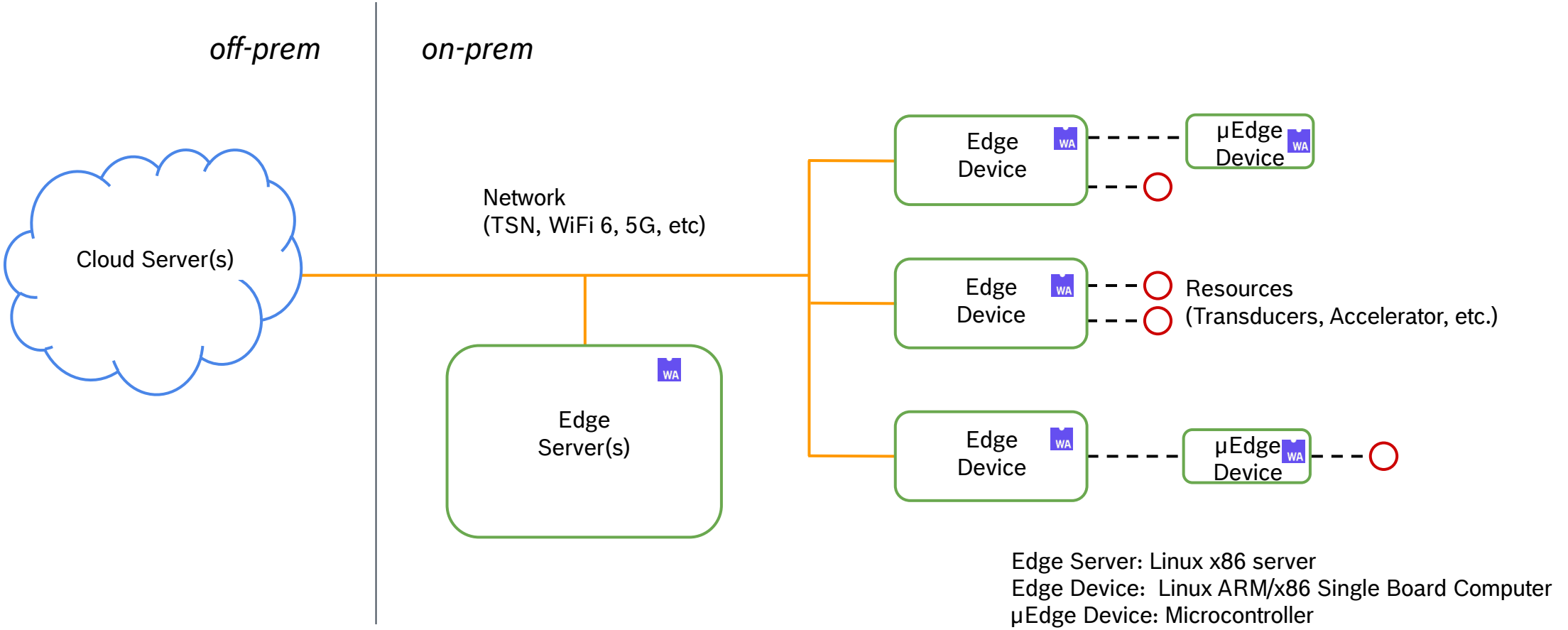
- WebAssembly Specification 2.0 (Draft)
- Mechanising and verifying the WebAssembly specification

- Opens the door for certified compilers that translate to Wasm
- Raises the bar for programming language design in general
- Upgrades the state-of-the-art for certification

A computation either runs forever, traps, or terminates with a result that has the expected type.
It cannot “crash” or otherwise (mis)behave in ways not covered by the execution semantics.

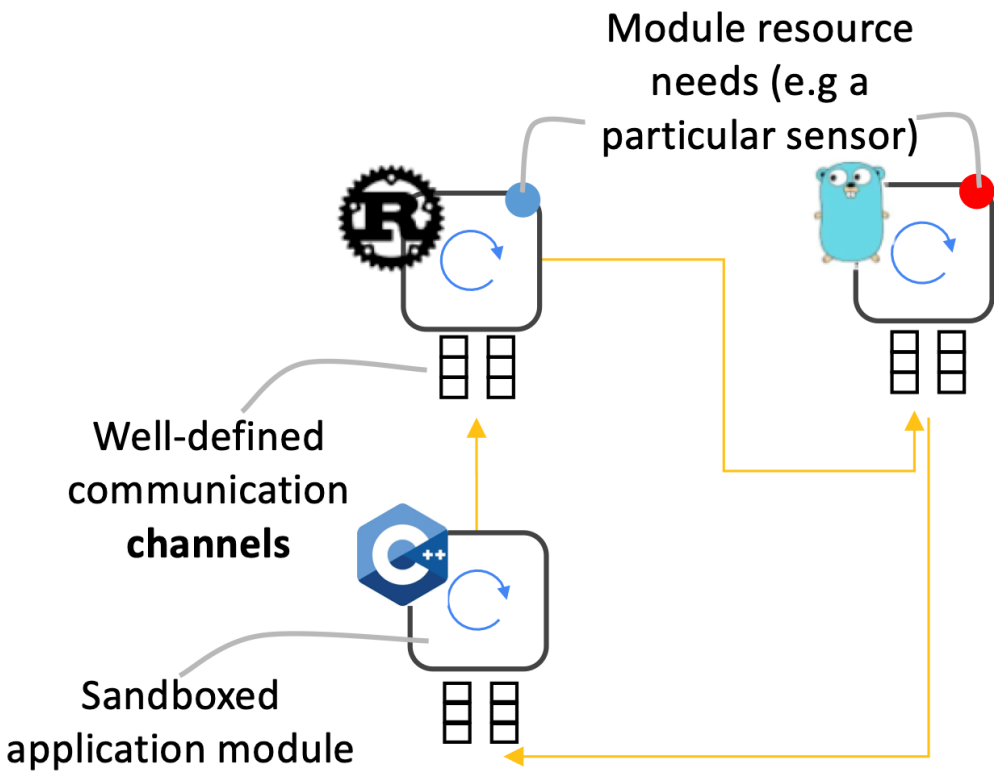
ORCHESTRATION AND DEPLOYMENT OF DISTRIBUTED WASM APPLICATIONS

Wasm orchestration and deployment
Heterogenous platforms and resource-constrained devices



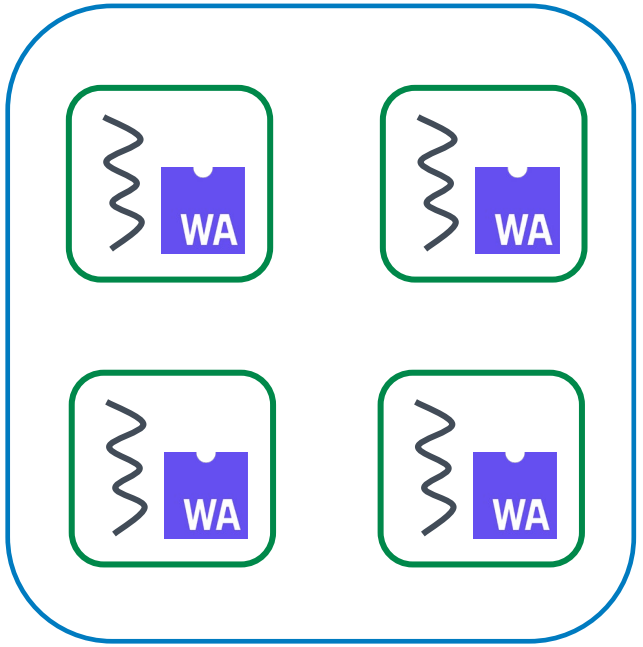
Edge Server: Linux x86 server
Edge Device: Linux ARM/x86 Single Board Computer
µEdge Device: Microcontroller

Wasm orchestration and deployment
Distributed Wasm applications



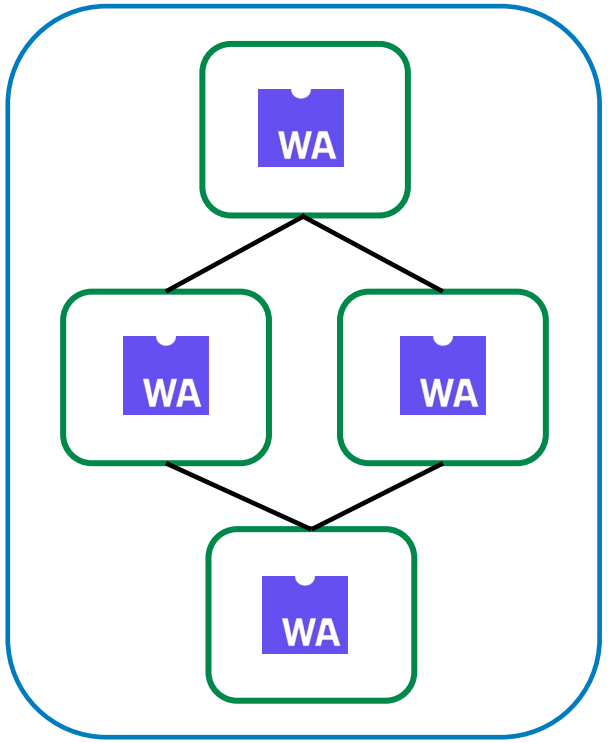
- Wasm modules
  - Platform-independent
  - Language-independent
  - Portable
  - Sandboxed
  - Migratable
- Host runtime
  - Platform-specific
  - Provides platform access and resources
    - Filesystem, network interfaces, sensors, accelerators
  - Generic for different applications

Wasm orchestration and deployment
High density and multi-tenancy



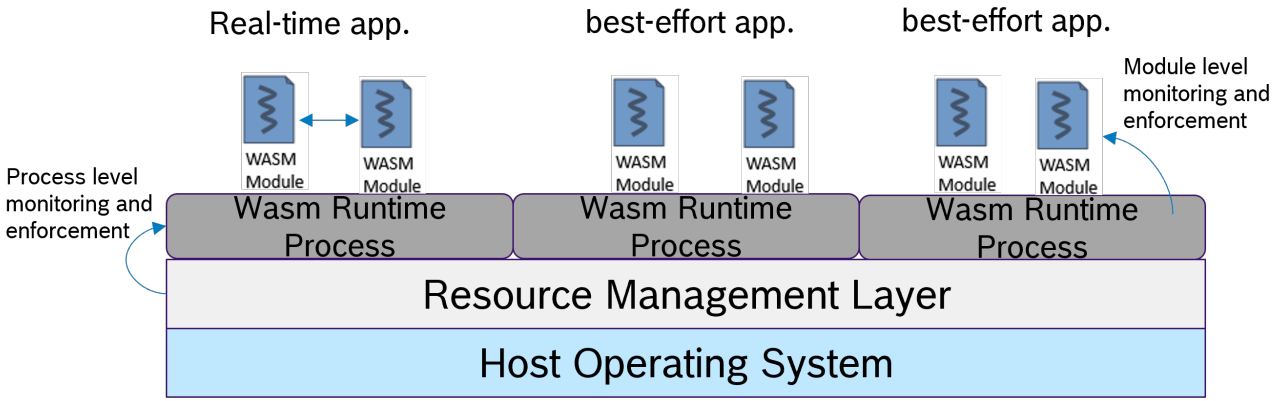
Nano processes

- 1 runtime, many modules
  - keep isolation
  - keep fine-grained capabilities
- 1 module, 1 thread
  - OS threads
  - green threads
  - runtime level scheduler
- Link modules as components
  - lightweight microservices
  - Fast function calls
  - Synchronous calls
  - Asynchronous calls



Module linking

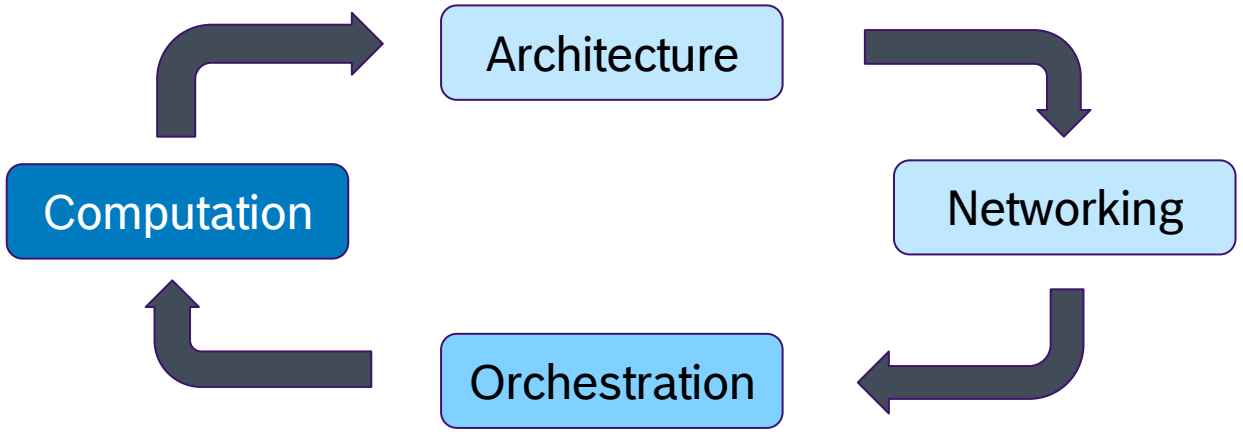
Wasm orchestration and deployment
Resource Monitoring and Enforcement



- Host runtime
  - Process-level resource mechanisms
    - Scheduling primitives
    - Control groups
  - Separate runtimes for different QoS requirements/criticalities
- Wasm modules
  - Fine-grained monitoring mechanisms
    - „Gas“ metering for computation time
    - Metering for linear memory

CONCLUSION AND FUTURE WORK

Conclusion and Future Work
Distributed reliable edge-cloud applications



- Wasm is promising – also for realtime
  - Performance
  - Determinism
  - Safety and security
  - Metering
  - Customizable host runtime
- Distributed applications require more
  - Real-time networking
  - Edge orchestration
  - Predictable distributed timing
  - A suitable distributed programming model
  - Functional determinism for safety

THANK YOU FOR LISTENING

QUESTION IT !

BOSCH