Hasso Plattner Institut
Digital Engineering · Universität Potsdam

WebAssembly im Kernel

Implementierung und Evaluierung einer WebAssembly-Laufzeitumgebung als Linux-Kernelmodul

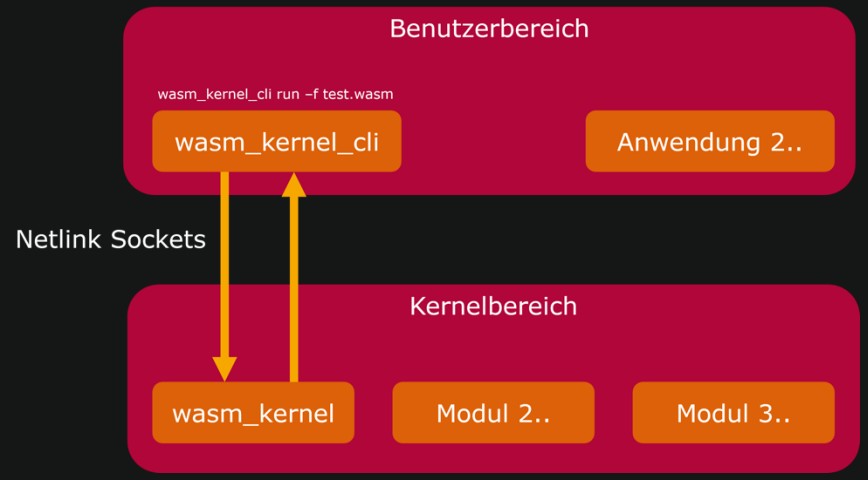
Sebastian Wilke

Betreuung: Clemens & Lukas

Design IT.
Create Knowledge.

www.hpi.de

Was in der letzten Episode geschah..



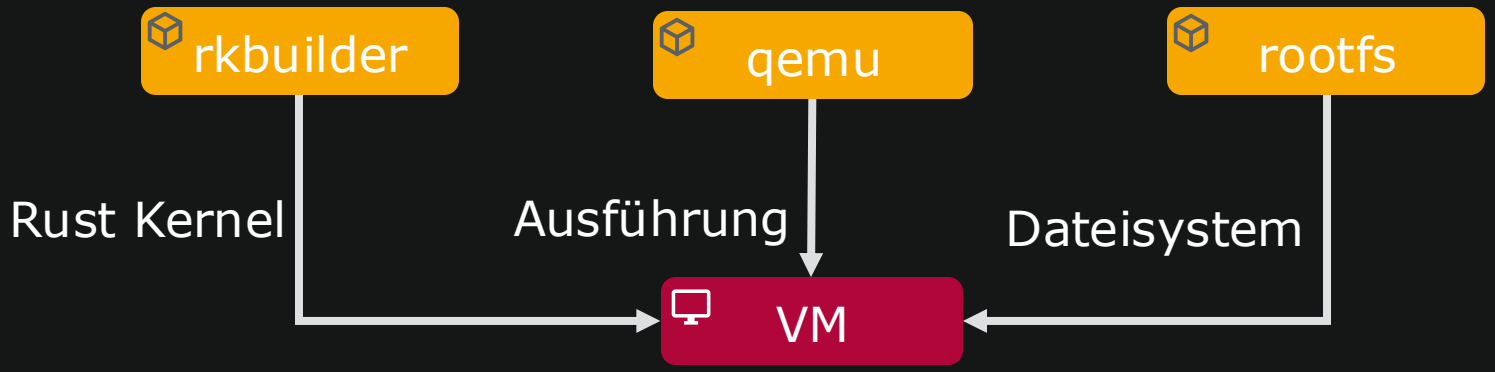

```
error[E0433]: failed to resolve: use of undeclared crate or module `wasmparser`
   --> /home/basti/kernel-wasm-runtime/src/dependencies/wasmi/module/utils.rs:149:25
    |
149 |     fn from(value_type: wasmparser::ValType) -> Self {
    |                         ^^^^^^^^^^ use of undeclared crate or module `wasmparser`

error: aborting due to 2076 previous errors

error: missing documentation for a module
 --> /home/basti/kernel-wasm-runtime/src/test/mod.rs:2:1
  |
2 | pub mod execution;
  | ^^^^^^^^^^^^^^^^^

error: missing documentation for a function
  --> /home/basti/kernel-wasm-runtime/src/test/execution.rs:10:1
   |
10 | pub fn test_execution() -> Result<bool, kernel::alloc::AllocError> {
   | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

error: aborting due to 83 previous errors
```

```rust
module! {
    type: WasmKernel,
    name: "wasm_kernel",
    author: "Sebastian Wilke",
    description: "WebAssembly runtime",
    license: "GPL",
}

struct WasmKernel {}

impl kernel::Module for WasmKernel {
    fn init(_module: &'static ThisModule) -> Result<Self> {
        pr_info!("Wasm Kernel (init)\n");
        test::execution::test_execution().unwrap();
        Ok(WasmKernel {})
    }
}

impl Drop for WasmKernel {
    fn drop(&mut self) {
        pr_info!("Wasm Kernel (exit)\n");
    }
}
```

```
[   44.097518] wasm_kernel: loading out-of-tree module taints kernel.
[   44.105890] wasm_kernel: Wasm Kernel (init)
[   44.110596] wasm_kernel: Testing i32_add:
[   44.111903] wasm_kernel: #1 2 + 3 = 5
[   44.113818] wasm_kernel: #2 10 + 5 = 15
[   44.114033] wasm_kernel: #3 1 + 1 = 2
[   44.115644] wasm_kernel: Testing i32_sub:
[   44.116092] wasm_kernel: #1 3 - 2 = 1
[   44.116225] wasm_kernel: #2 10 - 5 = 5
[   44.118144] wasm_kernel: #3 1 - 1 = 0
[   44.124222] wasm_kernel: Testing store_i32:
[   44.124669] wasm_kernel: Value at offset 16 = 1337
```

Entwicklungsumgebung für Rust im Kernel

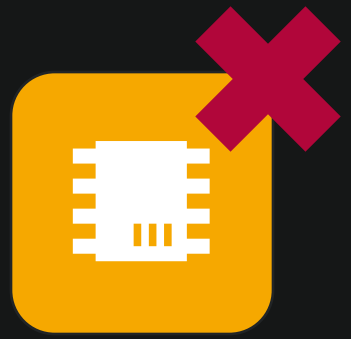
- Implementierung von *wasm_kernel* als Out-of-tree-Modul
- Zum konsistenten Testen: Verwendung von https://github.com/0xor0ne/RoustKit



- Seit über einem Jahr ungepflegt, daher:
- Aktualisierung des Kernels auf 6.12 und von Debian 11 auf 12 durch mich

Ausblick der Zwischenpräsentation

Fleißarbeit



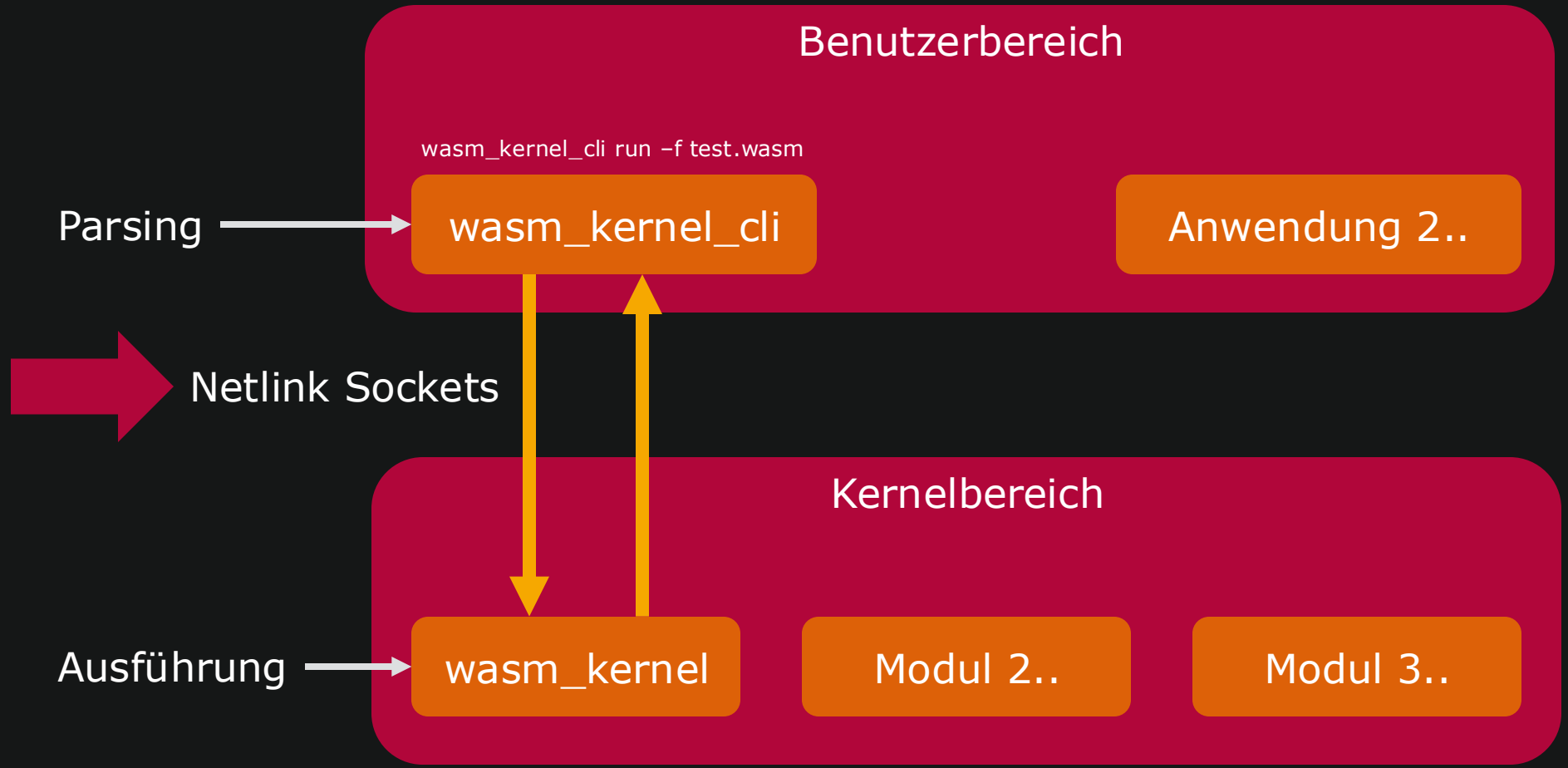
WebAssembly Core Instruktionssatz

WASM-Modulparser

Anbindung an Kernelschnittstellen

Test und Evaluierung

WASM-Modulparser

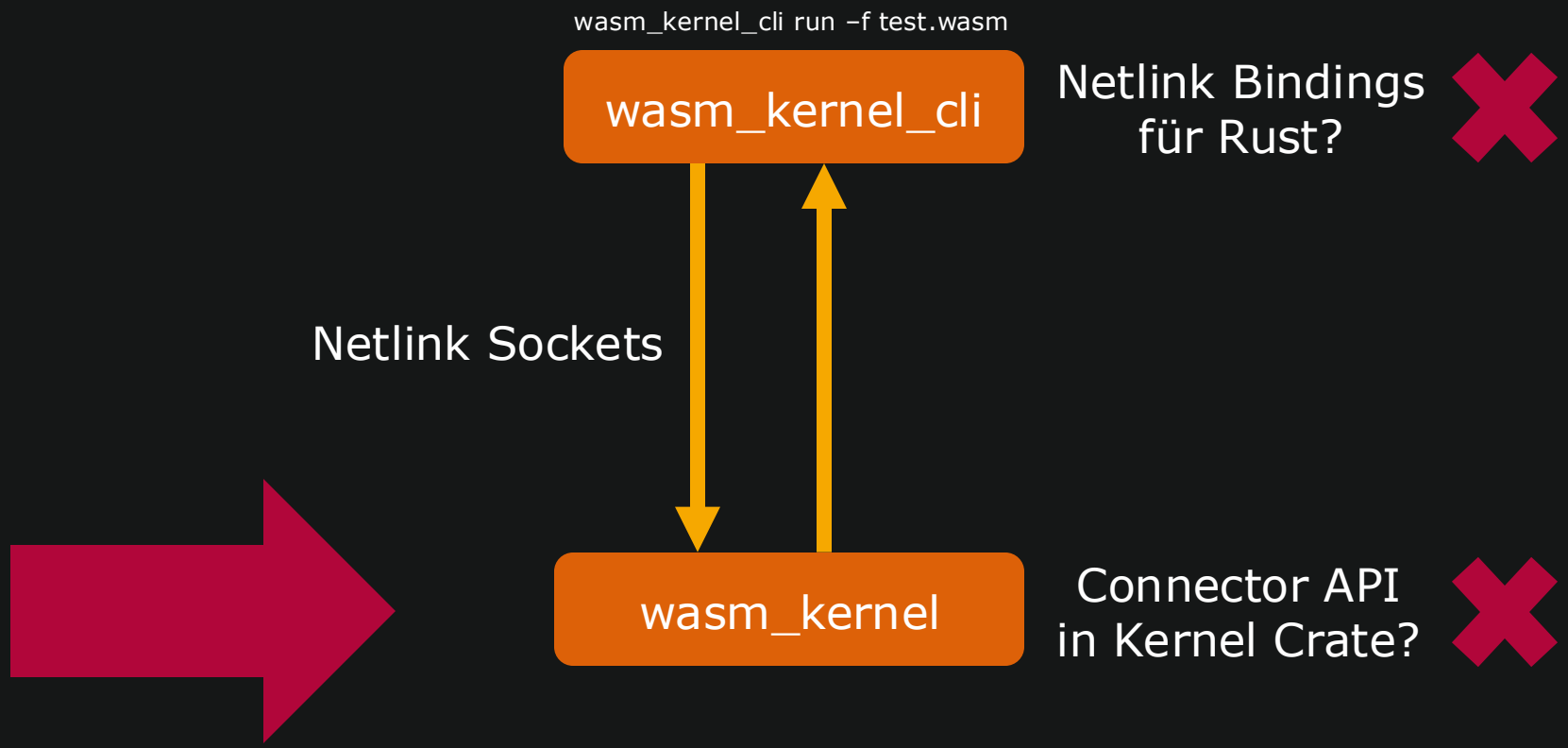


Kernelbereich ↔ Nutzerbereich: **Connector API** (Netlink Sockets)

- Ursprüngliche Idee: Netlink Sockets
- Neue Idee: Connector API (die wiederum Netlink Sockets verwendet..)

- Connector API bietet kernelseitig einfachere Implementierung durch
  - Callbacks (falls Nachricht aus Nutzerbereich eintrifft)
  - Funktion zum Senden von Connector-Nachrichten (erspart Socket-Handling)

Kernelbereich <-> Nutzerbereich:
Connector API (Netlink Sockets)

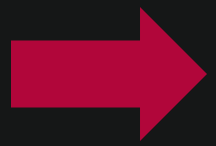


Exkurs

Rust Bindgen für *Foreign Function Interface* (FFI)

- Bindgen:
  - Generiert Rust-FFI-Code zur Interaktion mit C/C++ Bibliotheken

```c
typedef struct Doggo {
    int many;
    char wow;
} Doggo;

void eleven_out_of_ten_majestic_af(Doggo* pupper);
```



```rust
#[repr(C)]
pub struct Doggo {
    pub many: ::std::os::raw::c_int,
    pub wow: ::std::os::raw::c_char,
}

extern "C" {
    pub fn eleven_out_of_ten_majestic_af(pupper: *mut Doggo);
}
```

  - Ggf. sind Funktionen dann allerdings als `unsafe` markiert
    - z. B. bei *Raw Pointers*
- Nutzung durch Rust for Linux für Kernel-APIs
- Aber: bisher nur kleine Teilmenge der APIs generiert!

Rust for Linux

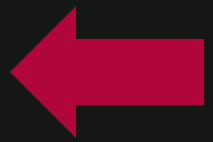
linux/rust/bindings/bindings_helper.h

```
/*
 * Header that contains the code (mostly headers) for which Rust bindings
 * will be automatically generated by `bindgen`.
 *
 * Sorted alphabetically.
 */

#include <kunit/test.h>
#include <linux/blk-mq.h>
#include <linux/blk_types.h>
#include <linux/blkdev.h>
#include <linux/errname.h>
#include <linux/ethtool.h>
#include <linux/firmware.h>
#include <linux/jiffies.h>
#include <linux/mdio.h>
#include <linux/phy.h>
#include <linux/refcount.h>
#include <linux/sched.h>
#include <linux/slab.h>
#include <linux/wait.h>
#include <linux/workqueue.h>
```

Rust for Linux

linux/rust/bindings/bindings_helper.h

```
/*
 * Header that contains the code (mostly headers) for which Rust bindings
 * will be automatically generated by `bindgen`.
 *
 * Sorted alphabetically.
 */

#include <kunit/test.h>
#include <linux/blk-mq.h>
#include <linux/blk_types.h>
#include <linux/blkdev.h>
#include <linux/connector.h>
#include <linux/errname.h>
#include <linux/ethtool.h>
#include <linux/firmware.h>
#include <linux/jiffies.h>
#include <linux/mdio.h>
#include <linux/phy.h>
#include <linux/refcount.h>
#include <linux/sched.h>
#include <linux/slab.h>
#include <linux/wait.h>
#include <linux/workqueue.h>
```



Rust for Linux

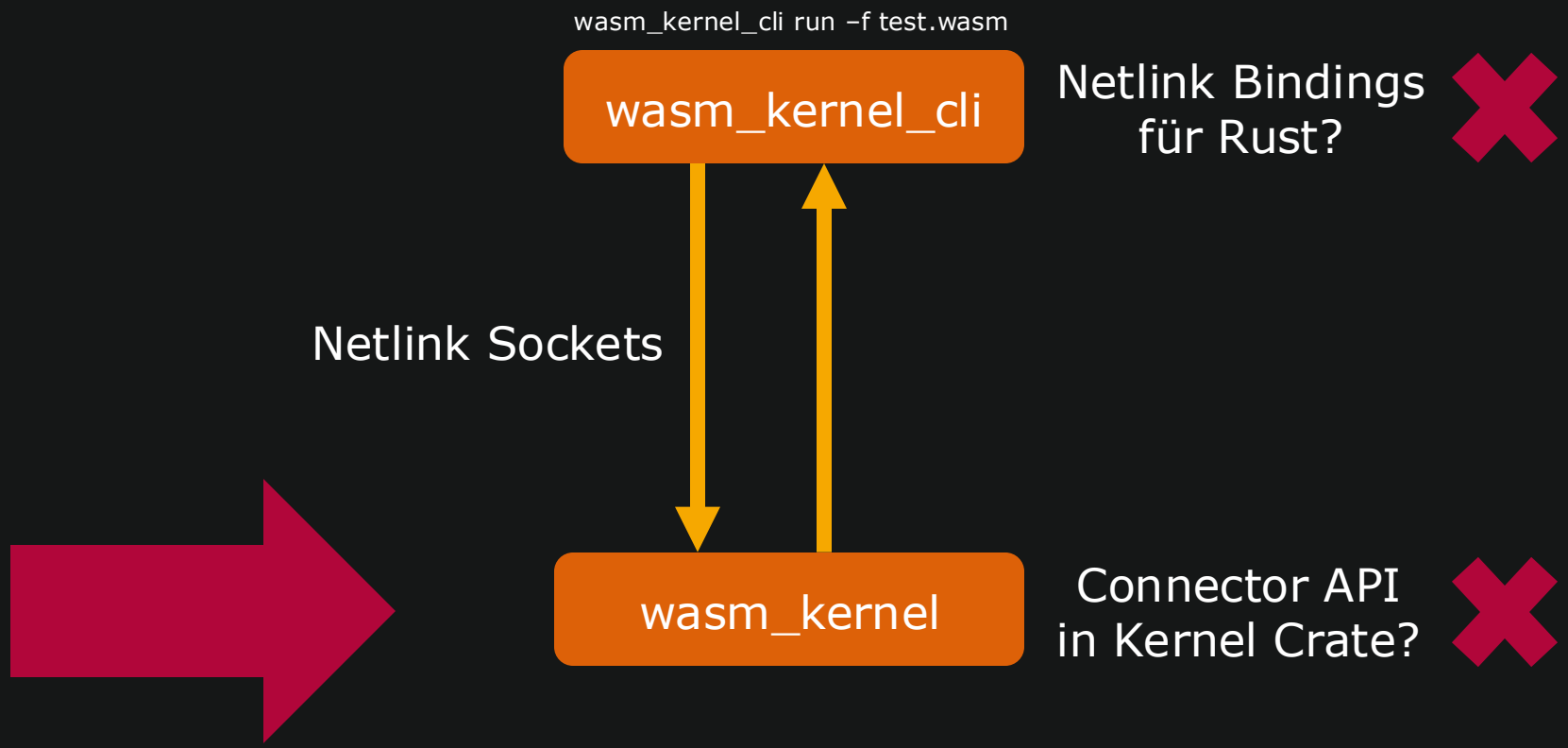
`unsafe` -> `safe`

- Unsafe-Code muss in sicheren Methoden verpackt werden
  - Dieser sollte dann z. B. Raw Pointer auf Korrektheit überprüfen, usw.
- Rust for Linux: Bündelung dieser abgesicherten APIs in *kernel*-Crate

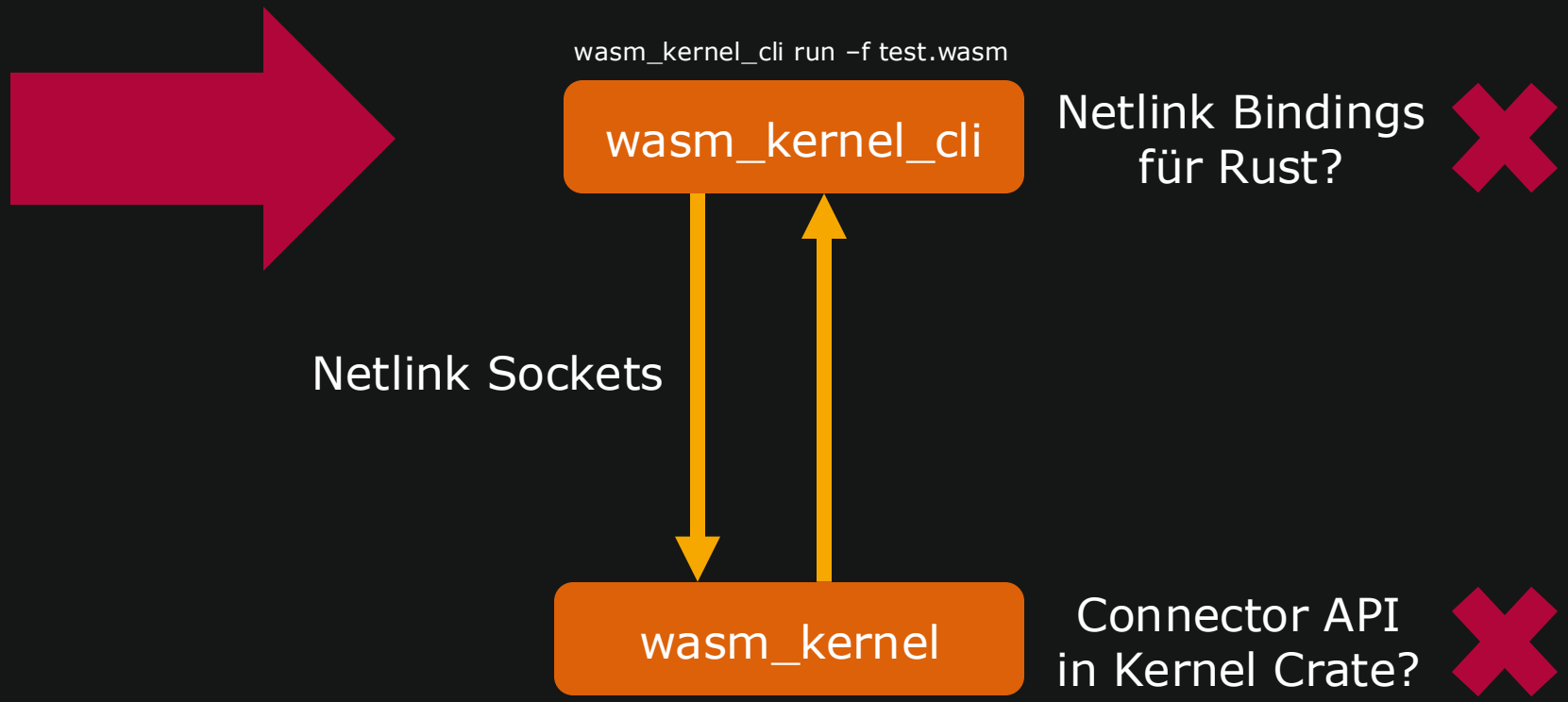
> *If you need a kernel C API that is not ported or wrapped yet here, then do so first instead of bypassing this crate.*
>
> *Kernel*-Crate Dokumentation

- D.h. Modifikation des *kernel*-Crates in Rust-for-Linux-Projekt war notwendig!
- Nur teilweise und eher unsauber implementiert; daher bisher kein PR in RFL-Repo

Kernelbereich <-> Nutzerbereich:
Connector API (Netlink Sockets)



Kernelbereich <-> Nutzerbereich:
Connector API (Netlink Sockets)



Nutzung von Netlink Sockets

Rust im Nutzerbereich

- Zugriff über FFI auf C-Standardbibliothek
  - Notwendig, um Sockets mit AF_NETLINK zu erstellen
- Verwendung der *rust-netlink* Crates

Netlink

This project aims at providing building blocks for netlink (see `man 7 netlink`).

If you seeking crates to communication with linux netlink, please try:

- The `rtnetlink` crate provides higher level abstraction for the route protocol
- The `audit` crate provides higher level abstractions for the audit protocol.
- The `genetlink` crate provides higher level abstraction for the generic netlink protocol
- The `ethtool` crate provides higher level abstraction for ethtool netlink protocol
- The `mptcp-pm` crate provides MPTCP path manager netlink protocol
- The `wl-nl80211` crate provides wireless nl80211 netlink protocol

If you seeking crates to parsing or emitting netlink packet, please try:

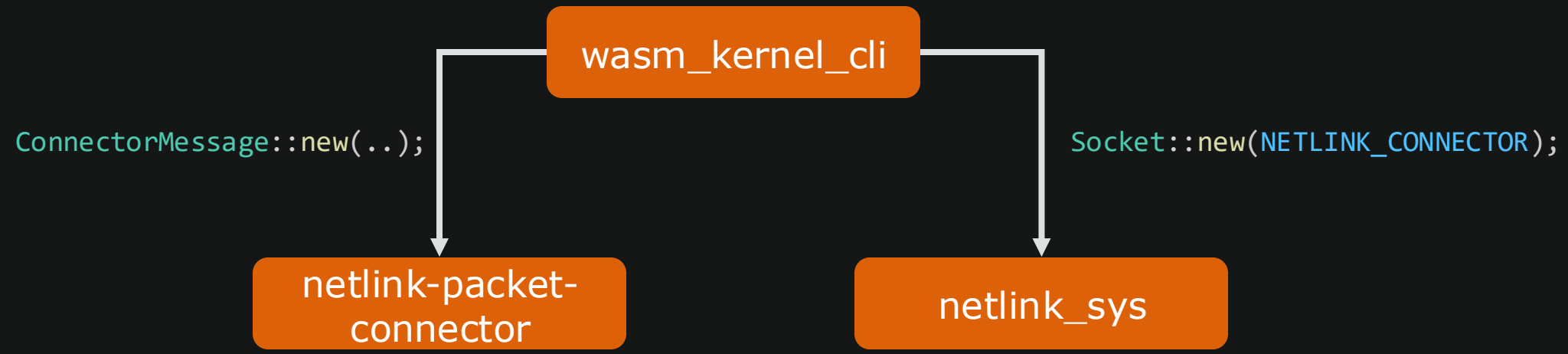
- Each netlink protocol has a `netlink-packet-<protocol_name>` crate that provides the packets for this protocol:
  - The `netlink-packet-wireguard` crate provide netlink message for wireguard.
  - `netlink-packet-route` provides messages for the route protocol
  - `netlink-packet-audit` provides messages for the audit protocol
  - `netlink-packet-sock-diag` provides messages for the sock-diag protocol
  - `netlink-packet-generic` provides message for the generic netlink protocol
  - `netlink-packet-netfilter` provides message for the `NETLINK_NETFILTER` protocol
  - `netlink-packet-xfrm` provides message for IPsec

netlink-packet-connector? :(

Nutzung von Netlink Sockets

Rust im Nutzerbereich

- Implementierung von Connector-Protokoll durch mich
- Neues Open Source Crate: netlink-packet-connector
  - Basiert dennoch auf *rust-netlink* Crates:
    - netlink_sys für sichere Netlink Socket FFI-API
    - netlink-packet-core für Netlink-Protokoll



Nutzung von Netlink Sockets

Rust im Nutzerbereich

- Nutzer→ Kernel: ✓
- Nutzer ← Kernel: ❌

- ..Viel Debugging später:

- Connector API

  “zweckentfremdet” Netlink

  Nachrichtentyp *NLMSG_DONE*
- Klassische Netlink-

  Bibliotheken verwirrt dies

```c
int cn_netlink_send_mult(struct cn_msg *msg, u16 len, u32 portid, u32 __group, gfp_t
gfp_mask, netlink_filter_fn filter,
void *filter_data)
{
    /* ... */
    unsigned int size;
    struct nlmsghdr *nlh;
    struct cn_msg *data;

    nlh = nlmsg_put(skb, 0, msg->seq, NLMSG_DONE, size, 0);
    if (!nlh) {
        kfree_skb(skb);
        return -EMSGSIZE;
    }

    data = nlmsg_data(nlh);

    memcpy(data, msg, size);

    /* ... */
}
```

Nutzung von Netlink Sockets

Rust im Nutzerbereich

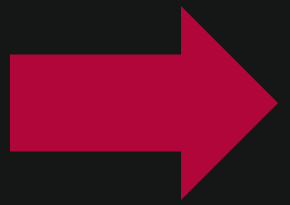
`NLMSG_DONE` is simpler, the request is never echoed but the extended ACK attributes may be present:

```
-----------------------------------------
| struct nlmsghdr - response header     |
-----------------------------------------
|    int error                          |
-----------------------------------------
| ** optionally extended ACK            |
-----------------------------------------
```

Note that some implementations may issue custom `NLMSG_DONE` messages in reply to `do` action requests. In that case the payload is implementation-specific and may also be absent.

u32 idx
u32 value
u32 seq
u32 ack
u16 len
u16 flags
u8[] data

```
struct cn_msg {
    struct cb_id id;

    __u32 seq;
    __u32 ack;

    __u16 len;      /* Length of the following data */
    __u16 flags;
    __u8 data[];
};
```



error ≠ idx

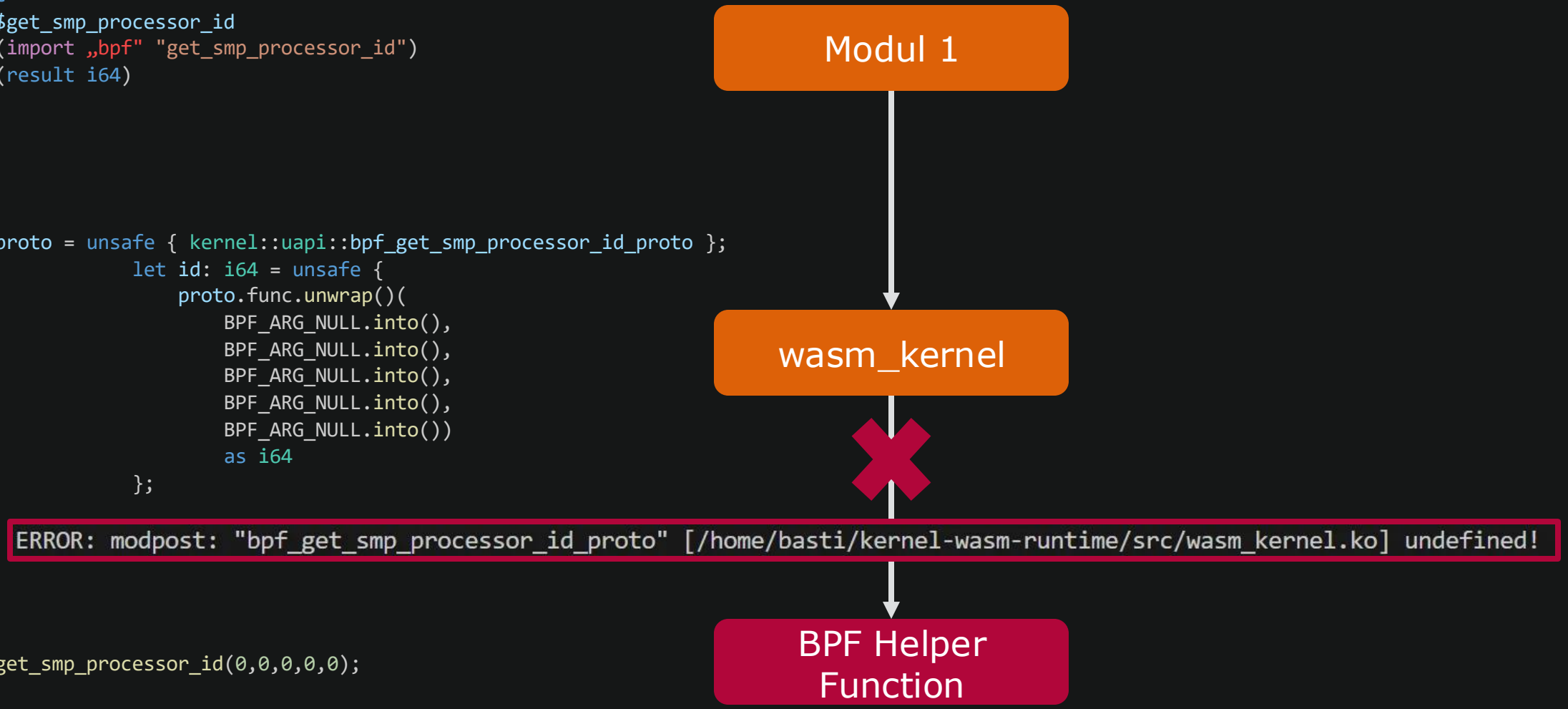
WASM-Modulparser

- Voraussetzungen sind jetzt erfüllt
- Aber: bisher nur so wenig Instruktionen unterstützt, sodass die aller meisten WASM-Binaries sowieso nicht geladen werden könnten
- Daher: Fokus auf Host-Function-Anbindung für Kernel APIs

Kernel APIs in WebAssembly Modulen

```
(func
    $get_smp_processor_id
    (import „bpf" "get_smp_processor_id")
    (result i64)
)
```

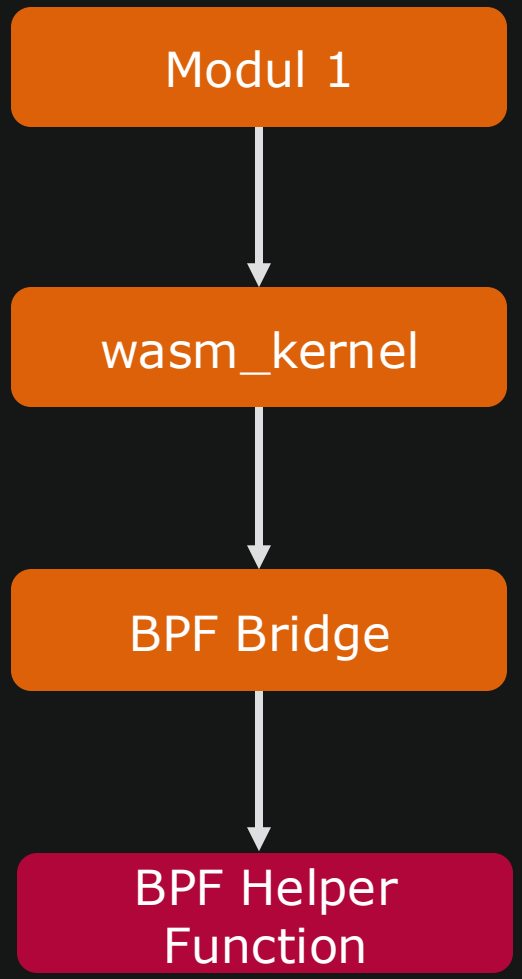
```
let proto = unsafe { kernel::uapi::bpf_get_smp_processor_id_proto };
                let id: i64 = unsafe {
                    proto.func.unwrap()(
                        BPF_ARG_NULL.into(),
                        BPF_ARG_NULL.into(),
                        BPF_ARG_NULL.into(),
                        BPF_ARG_NULL.into(),
                        BPF_ARG_NULL.into())
                        as i64
                };
```



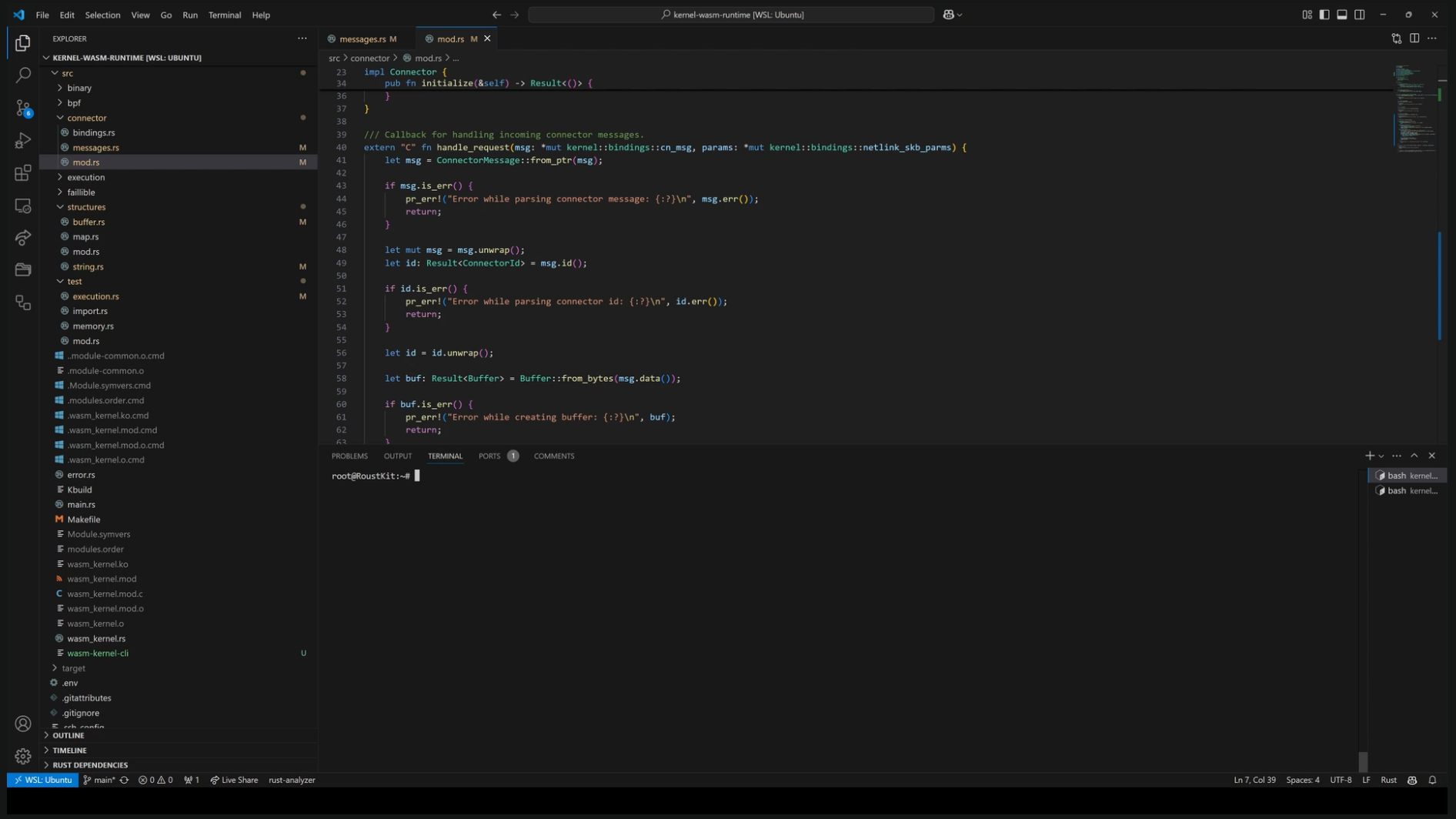

```
bpf_get_smp_processor_id(0,0,0,0,0);
```

Kernel APIs in WebAssembly Modulen

Mögliche Alternative:



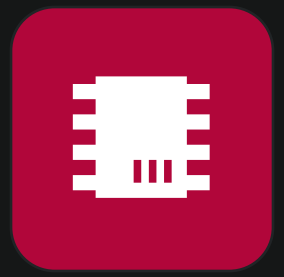
Demo


File Edit Selection View Go Run Terminal Help
kernel-wasm-runtime [WSL: Ubuntu]
EXPLORER
KERNEL-WASM-RUNTIME [WSL: UBUNTU]
src
binary
bpf
connector
bindings.rs
messages.rs M
mod.rs M
execution
faillible
structures
buffer.rs M
map.rs
mod.rs
string.rs M
test
execution.rs M
import.rs
memory.rs
mod.rs
error.rs
Kbuild
main.rs
Makefile
Module.symvers
modules.order
wasm_kernel.ko
wasm_kernel.mod
wasm_kernel.mod.c
wasm_kernel.mod.o
wasm_kernel.o
wasm_kernel.rs
wasm-kernel-cli U
target
.env
.gitattributes
.gitignore
messages.rs M
mod.rs M
src > connector > mod.rs > ...
impl Connector {
pub fn initialize(&self) -> Result<()> {
}
}
/// Callback for handling incoming connector messages.
extern "C" fn handle_request(msg: *mut kernel::bindings::cn_msg, params: *mut kernel::bindings::netlink_skb_parms) {
let msg = ConnectorMessage::from_ptr(msg);
if msg.is_err() {
pr_err!("Error while parsing connector message: {:?}\n", msg.err());
return;
}
let mut msg = msg.unwrap();
let id: Result<ConnectorId> = msg.id();
if id.is_err() {
pr_err!("Error while parsing connector id: {:?}\n", id.err());
return;
}
let id = id.unwrap();
let buf: Result<Buffer> = Buffer::from_bytes(msg.data());
if buf.is_err() {
pr_err!("Error while creating buffer: {:?}\n", buf);
return;
}
PROBLEMS OUTPUT TERMINAL PORTS 1 COMMENTS
root@RoustKit:~#
bash kernel...
bash kernel...
OUTLINE
TIMELINE
RUST DEPENDENCIES
WSL: Ubuntu main* Live Share rust-analyzer
Ln 7, Col 39 Spaces: 4 UTF-8 LF Rust

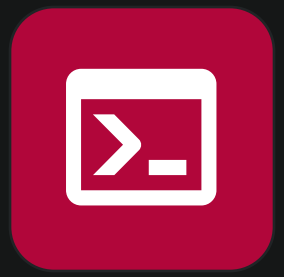
Fazit

- Seit Zwischenpräsentation eher Arbeit an Umgebung als an WASM-Instruktionen
- WASM-Parser fehlt weiterhin; Kommunikation mittels Connector brauchte deutlich mehr Zeit als geplant
- Kernel-APIs können integriert werden, jedoch keine BPF-Hilfsfunktionen

- Aus einem Projekt wurden vier:



wasm_kernel_runtime



wasm_kernel_cli

*bald

Safe Bindings für Connector API in *kernel*-Crate

netlink-packet-connector

Vielen Dank!

Fleißarbeit

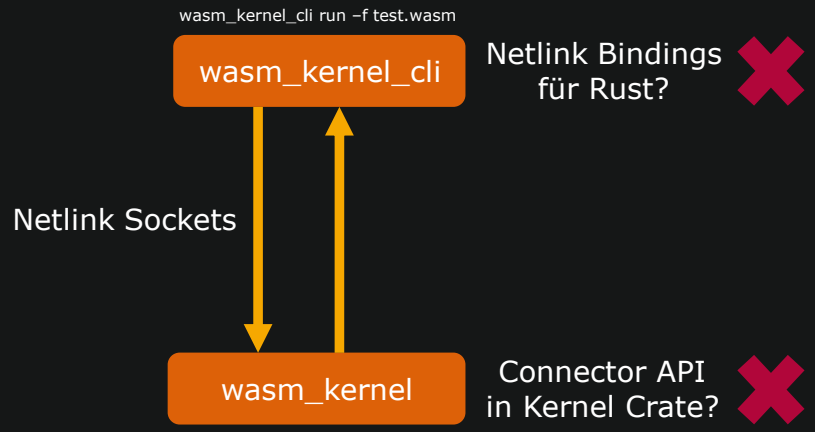
WebAssembly Core Instruktionssatz

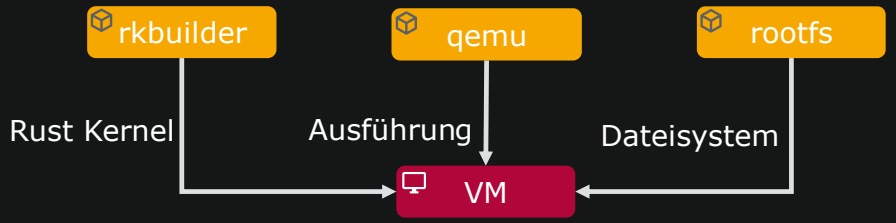
WASM-Modulparser

Anbindung an Kernelschnittstellen

Test und Evaluierung

```
[   44.097518] wasm_kernel: loading out-of-tree module taints kernel.
[   44.105890] wasm_kernel: Wasm Kernel (init)
[   44.110596] wasm_kernel: Testing i32_add:
[   44.111903] wasm_kernel: #1 2 + 3 = 5
[   44.113818] wasm_kernel: #2 10 + 5 = 15
[   44.114033] wasm_kernel: #3 1 + 1 = 2
[   44.115644] wasm_kernel: Testing i32_sub:
[   44.116092] wasm_kernel: #1 3 - 2 = 1
[   44.116225] wasm_kernel: #2 10 - 5 = 5
[   44.118144] wasm_kernel: #3 1 - 1 = 0
[   44.124222] wasm_kernel: Testing store_i32:
[   44.124669] wasm_kernel: Value at offset 16 = 1337
```





wasm_kernel_runtime

wasm_kernel_cli

*bald

Safe Bindings für Connector API in *kernel*-Crate

netlink-packet-connector