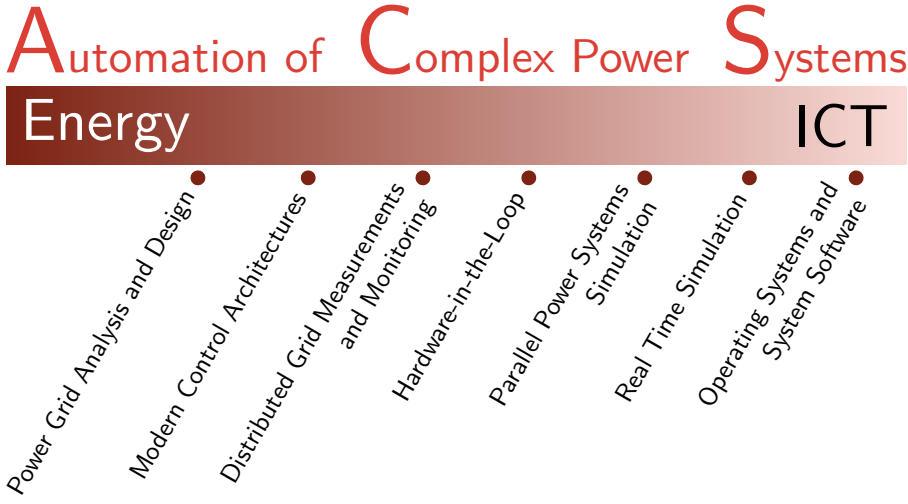
Sichere Systemsoftware in Rust?

J. Klimt, S. Lankes, J. Breitbart, S. Pickartz

E.ON Energy Research Center | RWTH AACHEN UNIVERSITY


Automation of Complex Power Systems
Energy
ICT
Power Grid Analysis and Design
Modern Control Architectures
Distributed Grid Measurements and Monitoring
Hardware-in-the-Loop
Parallel Power Systems Simulation
Real Time Simulation
Operating Systems and System Software

Motivation

Bugs

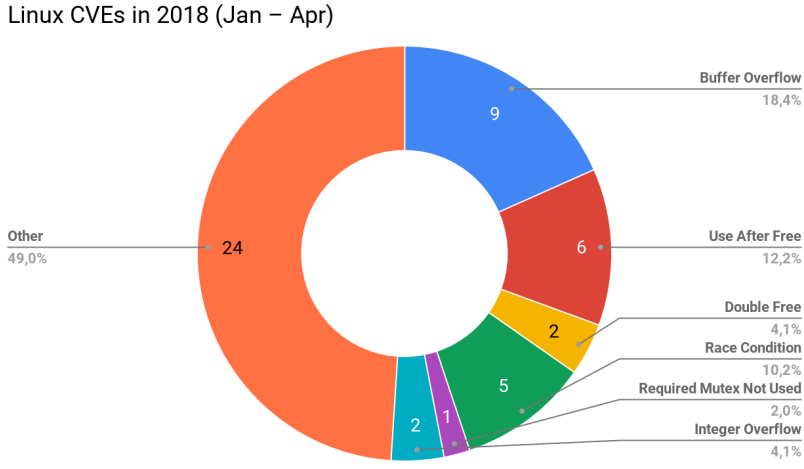
Microsoft: A Proactive Approach to more Secure Code:

- *70% of the vulnerabilities Microsoft assigns a CVE each year continue to be memory safety issues.*

Source: msrc-blog.microsoft.com

Source: Martin Maciaszek - CC BY 2.0

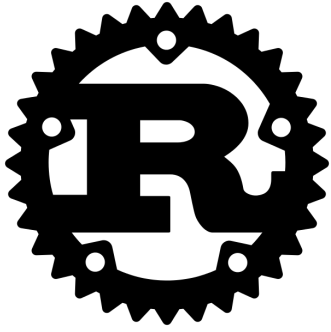
Bugs



Source: cvedetails.com via phil-opp.github.io

Rust

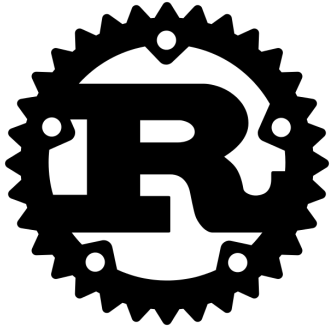
- Leistung ähnlich zu C/C++



Source: Rust-Lang - CC BY

Rust

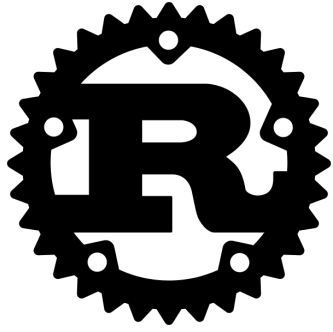
- Leistung ähnlich zu C/C++
- Umfassende Compilerchecks
  - Speichersicherheit ohne Garbage Collection
  - Keine Data Races



Source: Rust-Lang - CC BY

Rust

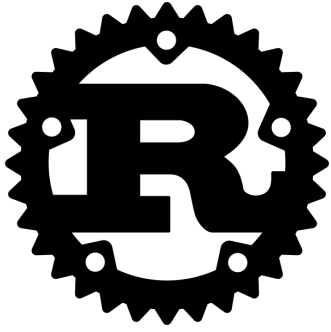
- Leistung ähnlich zu C/C++
- Umfassende Compilerchecks
  - Speichersicherheit ohne Garbage Collection
  - Keine Data Races
- Moderne Programmierkonzepte
  - Starkes Typsystem
  - Namespaces
  - Generic Types
  - Error Handling
  - Asynchrone Programmierung
  - Funktionale Programmierkonzepte (Closures, Iterators)
  - Automatische Bounds-Checks



Source: Rust-Lang - CC BY

Rust

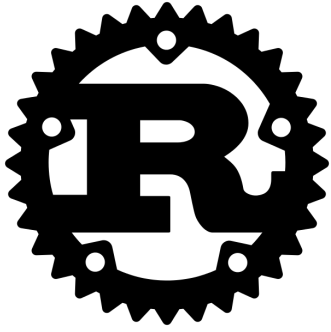
- Leistung ähnlich zu C/C++
- Umfassende Compilerchecks
  - Speichersicherheit ohne Garbage Collection
  - Keine Data Races
- Moderne Programmierkonzepte
  - Starkes Typsystem
  - Namespaces
  - Generic Types
  - Error Handling
  - Asynchrone Programmierung
  - Funktionale Programmierkonzepte (Closures, Iterators)
  - Automatische Bounds-Checks
- Hervorragendes Tooling



Source: Rust-Lang - CC BY

Rust

- Leistung ähnlich zu C/C++
- Umfassende Compilerchecks
  - Speichersicherheit ohne Garbage Collection
  - Keine Data Races
- Moderne Programmierkonzepte
  - Starkes Typsystem
  - Namespaces
  - Generic Types
  - Error Handling
  - Asynchrone Programmierung
  - Funktionale Programmierkonzepte (Closures, Iterators)
  - Automatische Bounds-Checks
- Hervorragendes Tooling
- C-interface



Source: Rust-Lang - CC BY

Missing Mutex

C++:

```
std::mutex mtx;
crit_var = 0;

//[...]

mtx.lock();
crit_var = 42;
mtx.unlock();
```

Missing Mutex

C++:

```
std::mutex mtx;
crit_var = 0;

//[...]

// mtx.lock();
crit_var = 42; // Data Race
// mtx.unlock();
```

Missing Mutex

C++:

```
std::mutex mtx;
crit_var = 0;

//[...]

// mtx.lock();
crit_var = 42; // Data Race
// mtx.unlock();
```

Rust:

```
let crit_var = Mutex::new(0);

// [...]

*crit_var.lock().unwrap() = 42;

// out of scope -> auto-lock of mtx
```

Ownership:

- Speicher hat immer *einen* “Owner”

Ownership:

- Speicher hat immer *einen* “Owner”
- Destruktoren wenn Out-of-Scope (RAII)

Ownership:

- Speicher hat immer *einen* "Owner"
- Destruktoren wenn Out-of-Scope (RAII)
- Keine Aliase für veränderliche Variablen
  - $\Rightarrow$ Kein use-after-free
  - $\Rightarrow$ Thread-Safety/Keine Data Races

Use after free

C++:

```
std::vector<int> v { 10, 11 };
int *vptr = &v[1];
v.push_back(12);
std::cout « *vptr; // Bug
```

Use after free

C++:

```
std::vector<int> v { 10, 11 };
int *vptr = &v[1];
v.push_back(12);
std::cout « *vptr; // Bug
```

Rust:

```
let mut v = vec![10, 11];
let vptr = &mut v[1];
v.push(12);
println!("{}", *vptr);
```

Use after free

C++:

```
std::vector<int> v { 10, 11 };
int *vptr = &v[1];
v.push_back(12);
std::cout « *vptr; // Bug
```

Rust:

```
let mut v = vec![10, 11];
let vptr = &mut v[1];
v.push(12);
println!("{}", *vptr);
```

Error Message:

```
error[E0499]: cannot borrow `v` as mutable more
  than once at a time
 -> src/main.rs:6:1
  |
5 | let vptr = &mut v[1];
  |                - first mut borrow occurs here
6 | v.push(12);
  | ^ second mutable borrow occurs here
7 | println!("{}", *vptr);
  |                --- borrow later used here
```

Limitierungen der Compilerchecks:

Was ist mit?

- Pointermanipulation
- Hardware Access
- Inline Assembler
- Aufruf von externen (C-)Funktionen
- Veränderliche globalen Variablen

Limitierungen der Compilerchecks:

Was ist mit?

- Pointermanipulation
- Hardware Access
- Inline Assembler
- Aufruf von externen (C-)Funktionen
- Veränderliche globalen Variablen

```rust
// Read raw memory
let ptr = 0x8000_0000 as *mut u64;
println!("Memory: {}", *ptr);
```

Limitierungen der Compilerchecks:

Was ist mit?

- Pointermanipulation
- Hardware Access
- Inline Assembler
- Aufruf von externen (C-)Funktionen
- Veränderliche globalen Variablen

```rust
// Read raw memory
let ptr = 0x8000_0000 as *mut u64;
println!("Memory: {}", *ptr);
```

Compiler Error:

```
error[E0133]: dereference of raw pointer is
unsafe and requires unsafe function or block
 -> src/main.rs:5:16
5 | println!("{}", *ptr);// Compiler error
  |                ^^^^ dereference of raw
  |                     pointer
```

Limitierungen der Compilerchecks:

Was ist mit?

- Pointermanipulation
- Hardware Access
- Inline Assembler
- Aufruf von externen (C-)Funktionen
- Veränderliche globalen Variablen

→ `unsafe { }`

(eher: "unchecked")

```
// Read raw memory
let ptr = 0x8000_0000 as *mut u64;
println!("Memory: {}", *ptr);
```

Compiler Error:

```
error[E0133]: dereference of raw pointer is
unsafe and requires unsafe function or block
 -> src/main.rs:5:16
5 | println!("{}", *ptr);// Compiler error
  |                ^^^^ dereference of raw
  |                     pointer
```

Limitierungen der Compilerchecks:

Was ist mit?

- Pointermanipulation
- Hardware Access
- Inline Assembler
- Aufruf von externen (C-)Funktionen
- Veränderliche globalen Variablen

→ `unsafe { }`

(eher: "unchecked")

```
// Read raw memory
let ptr = 0x8000_0000 as *mut u64;
println!("Memory: {}", unsafe{*ptr});
```

Limitierungen der Compilerchecks:

Was ist mit?

- Pointermanipulation
- Hardware Access
- Inline Assembler
- Aufruf von externen (C-)Funktionen
- Veränderliche globalen Variablen

→ `unsafe { }`

(eher: "unchecked")

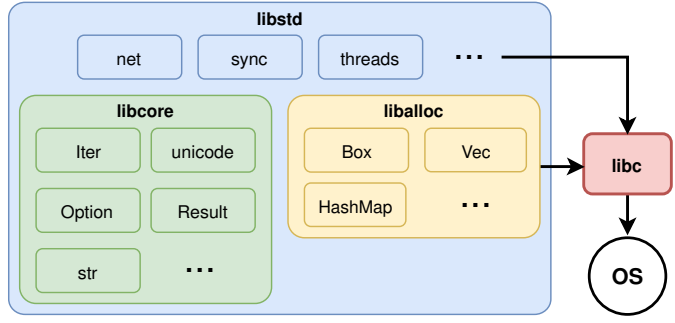
```
// Read raw memory
let ptr = 0x8000_0000 as *mut u64;
println!("Memory: {}", unsafe{*ptr});
```

Compiles and Runs:

```
> cargo run
     Running `target/debug/testprog`
[1]    56585 segmentation fault (core dumped)
```

Rust Standard Library

- `libcore`
  - Plattformunabhängig
- `liballoc`
  - Dynamische Datenstrukturen
  - Benötigt `global_alloc`
- `libstd`
  - BS-Abhängige Funktionen



Einschränkungen:

- Teilweise komplex (lifetimes, borrowing)

Einschränkungen:

- Teilweise komplex (lifetimes, borrowing)
- Einige Features noch “nightly”
  - inline assembly
  - `custom_test_frameworks`
  - `allocator_api`

Einschränkungen:

- Teilweise komplex (lifetimes, borrowing)
- Einige Features noch “nightly”
  - inline assembly
  - `custom_test_frameworks`
  - `allocator_api`
- `unsafe` für low-level Code benötigt → Speicherfehler

Einschränkungen:

- Teilweise komplex (lifetimes, borrowing)
- Einige Features noch “nightly”
  - inline assembly
  - `custom_test_frameworks`
  - `allocator_api`
- `unsafe` für low-level Code benötigt → Speicherfehler
- Längere Compilezeiten

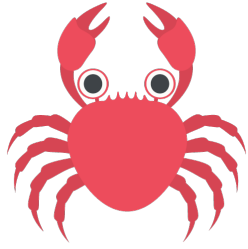
Einschränkungen:

- Teilweise komplex (lifetimes, borrowing)
- Einige Features noch “nightly”
  - inline assembly
  - `custom_test_frameworks`
  - `allocator_api`
- `unsafe` für low-level Code benötigt → Speicherfehler
- Längere Compilezeiten
- Tooling auf Open-Source ausgelegt

Betriebssysteme in Rust

Rust @ ACS

- Rusty-Hermit
  - Unikernel

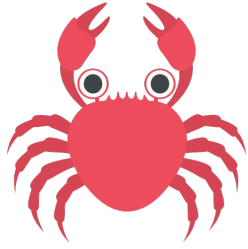


Source: EmojiOne

Rust @ ACS

- Rusty-Hermit
  - Unikernel
- eduOS
  - Mini Betriebssystem für die Lehre

(Beide Betriebssysteme haben einen C Vorgänger)



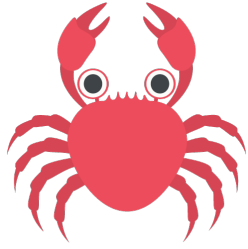
Source: EmojiOne

Rust @ ACS

- Rusty-Hermit
  - Unikernel
- eduOS
  - Mini Betriebssystem für die Lehre

(Beide Betriebssysteme haben einen C Vorgänger)

- uhyve
  - Schlanker Hypervisor für Rusty-Hermit



Source: EmojiOne

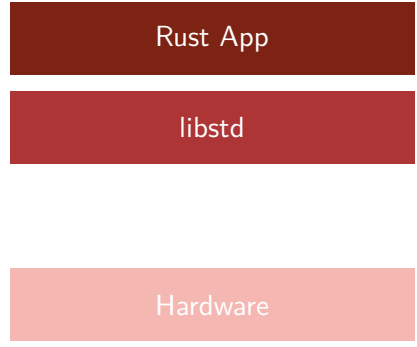
Rusty-Hermit

Rusty-Hermit

- Unikernel
- Smoltcp Netzwerk-Stack
- Linkt gegen C/C++/Go/Fortran

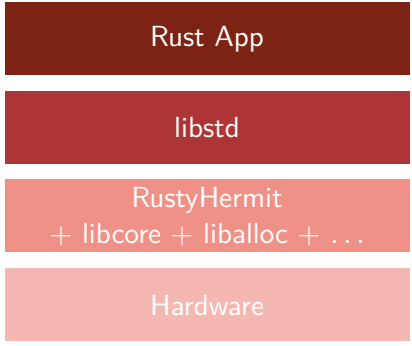
Rusty-Hermit

- Unikernel
- Smoltcp Netzwerk-Stack
- Linkt gegen C/C++/Go/Fortran
- Natives Rust Target



Rusty-Hermit

- Unikernel
- Smoltcp Netzwerk-Stack
- Linkt gegen C/C++/Go/Fortran
- Natives Rust Target



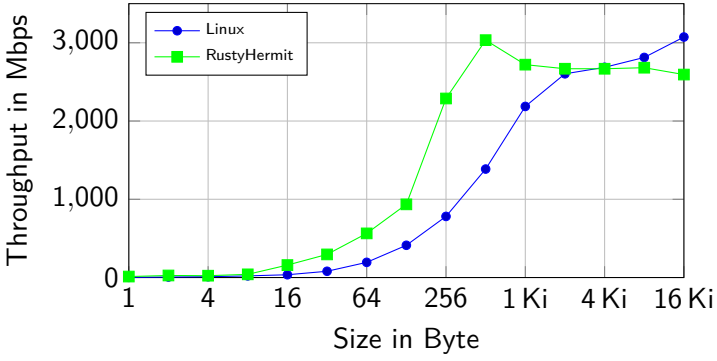
OS Micro-Benchmarks

System activity	**Linux**	**HermitCore** (C)	**RustyHermit** (Rust)
`getpid()`	1,962	13	36
`sched_yield()`	2,428	108	233
Thread creation	9,782	8,785	7,790
page fault handling	6,969	7,718	9,061

Test System:

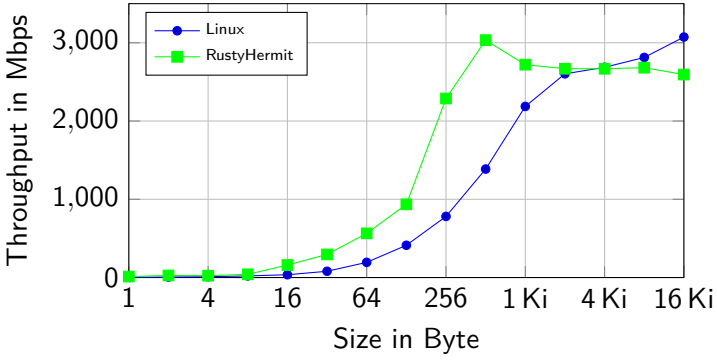
- Intel Xeon Gold 6132 with 14 physical cores
- 2.6 GHz, 376 GiB DDR4 RAM and 19.25 MiB L3 cache
- SpeedStep Technology, TurboMode, und Hyperthreading deaktiviert
- CentOS 7, 3.10.0 Linux kernel

Netzwerk Durchsatz



- RustyHermit in VM vs. Linux in VM

Netzwerk Durchsatz



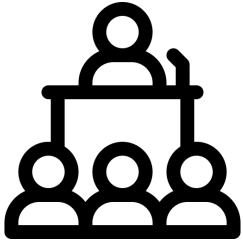
- RustyHermit in VM vs. Linux in VM
- Unikernel vs. Monolithischer Kernel

Related Work

- Redox OS (microkernel)
- blog OS (Lehre)
- Tock OS (embedded RTOS)
- Nebulet (Webasm, inactive)
- Rust im Linux-Kernel

Rust in der Lehre

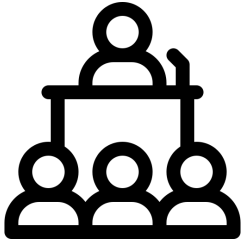
Rust in der Lehre



Icon by “Freepik” from Flaticon

Rust in der Lehre

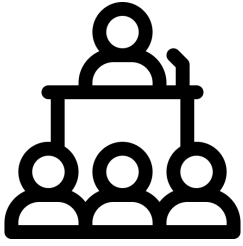
- Studierende haben keine Vorerfahrung



Icon by “Freepik” from Flaticon

Rust in der Lehre

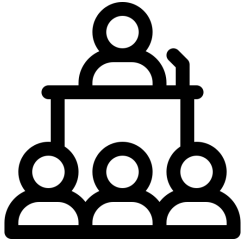
- Studierende haben keine Vorerfahrung
- Einige Features sind komplex



Icon by “Freepik” from Flaticon

Rust in der Lehre

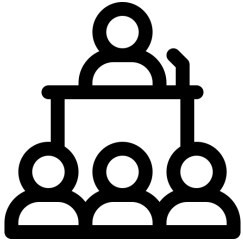
- Studierende haben keine Vorerfahrung
- Einige Features sind komplex
- Codebeispiele in *Betriebsysteme* VL in Rust



Icon by “Freepik” from Flaticon

Rust in der Lehre

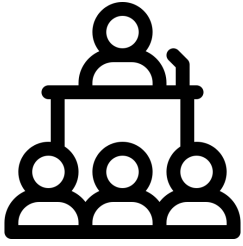
- Studierende haben keine Vorerfahrung
- Einige Features sind komplex
- Codebeispiele in *Betriebsysteme* VL in Rust
- Niederschwelliger Einstieg:
  - Einfaches Setup mit Cargo
  - Klare Projektstrukturen
  - Gutes Tooling (go-to-definition, Fehlermeldungen, . . .)



Icon by “Freepik” from Flaticon

Rust in der Lehre

- Studierende haben keine Vorerfahrung
- Einige Features sind komplex
- Codebeispiele in *Betriebsysteme* VL in Rust
- Niederschwelliger Einstieg:
  - Einfaches Setup mit Cargo
  - Klare Projektstrukturen
  - Gutes Tooling (go-to-definition, Fehlermeldungen, ...)
- Bringt interessierte Studierende ans Institut



Icon by "Freepik" from Flaticon

Zusammenfassung

Zusammenfassung:

- Rust ist eine moderne Programmiersprache

Zusammenfassung:

- Rust ist eine moderne Programmiersprache
- Rust KANN kritische Sicherheitslücken verhindern

Zusammenfassung:

- Rust ist eine moderne Programmiersprache
- Rust KANN kritische Sicherheitslücken verhindern
- Rust hat eine sehr vergleichbare Leistung zu C/C++

Zusammenfassung:

- Rust ist eine moderne Programmiersprache
- Rust KANN kritische Sicherheitslücken verhindern
- Rust hat eine sehr vergleichbare Leistung zu C/C++
- Soll ich morgen mein Betriebssystem in Rust neuschreiben?

Zusammenfassung:

- Rust ist eine moderne Programmiersprache
- Rust KANN kritische Sicherheitslücken verhindern
- Rust hat eine sehr vergleichbare Leistung zu C/C++
- Soll ich morgen mein Betriebssystem in Rust neuschreiben?
  - **UM HIMMELS WILLEN NEIN!**

Zusammenfassung:

- Rust ist eine moderne Programmiersprache
- Rust KANN kritische Sicherheitslücken verhindern
- Rust hat eine sehr vergleichbare Leistung zu C/C++
- Soll ich morgen mein Betriebssystem in Rust neuschreiben?
  - ≡ **UM HIMMELS WILLEN NEIN!**
- Soll ich neue (System-)Software in Rust schreiben?

Zusammenfassung:

- Rust ist eine moderne Programmiersprache
- Rust KANN kritische Sicherheitslücken verhindern
- Rust hat eine sehr vergleichbare Leistung zu C/C++
- Soll ich morgen mein Betriebssystem in Rust neuschreiben?
  - **UM HIMMELS WILLEN NEIN!**
- Soll ich neue (System-)Software in Rust schreiben?
  - Ich empfehle es!

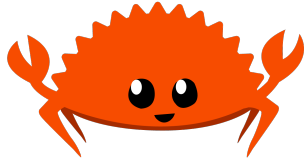
Zusammenfassung:

- Rust ist eine moderne Programmiersprache
- Rust KANN kritische Sicherheitslücken verhindern
- Rust hat eine sehr vergleichbare Leistung zu C/C++
- Soll ich morgen mein Betriebssystem in Rust neuschreiben?
  - **UM HIMMELS WILLEN NEIN!**
- Soll ich neue (System-)Software in Rust schreiben?
  - Ich empfehle es!
- Studierende sind an Rust interessiert

Zusammenfassung:

- Rust ist eine moderne Programmiersprache
- Rust KANN kritische Sicherheitslücken verhindern
- Rust hat eine sehr vergleichbare Leistung zu C/C++
- Soll ich morgen mein Betriebssystem in Rust neuschreiben?
  - **UM HIMMELS WILLEN NEIN!**
- Soll ich neue (System-)Software in Rust schreiben?
  - Ich empfehle es!
- Studierende sind an Rust interessiert

Vielen Dank! Fragen?



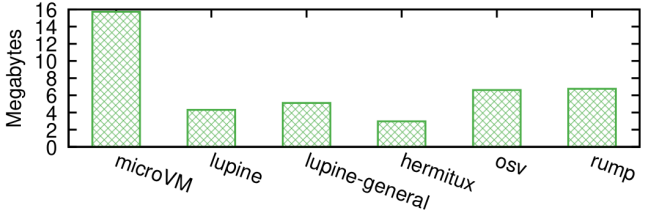
Thank you for your kind attention!

J. Klimt, S. Lankes, J. Breitbart, S. Pickartz – jonathan.klimt@eonerc.rwth-aachen.de

Institute for Automation of Complex Power Systems
E.ON Energy Research Center, RWTH Aachen University
Mathieustraße 10
52074 Aachen

www.eonerc.rwth-aachen.de

Size Comparision



From “A Linux in Unikernel Clothing” - Kuo et al.
Rusty-Hermit is $\approx 1.5MB$ large