

# An Empirical Study of Rust-for-Linux: The Success, Dissatisfaction, and Compromise

(\*)Hongyu Li<sup>1</sup>, <sup>(\*)</sup>Liwei Guo<sup>2</sup>, Yexuan Yang<sup>1</sup>, Shangguang Wang<sup>1</sup>, Mengwei Xu<sup>1</sup>

(\*) = co-primary

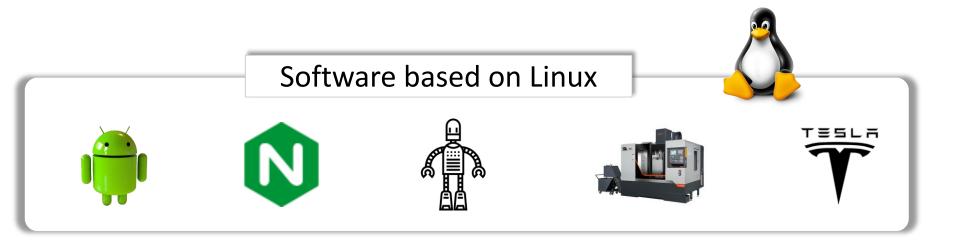
<sup>1</sup>Beijing University of Posts and Telecommunications



<sup>2</sup>University of Electronic Science and Technology of China



#### Linux suffers from bugs

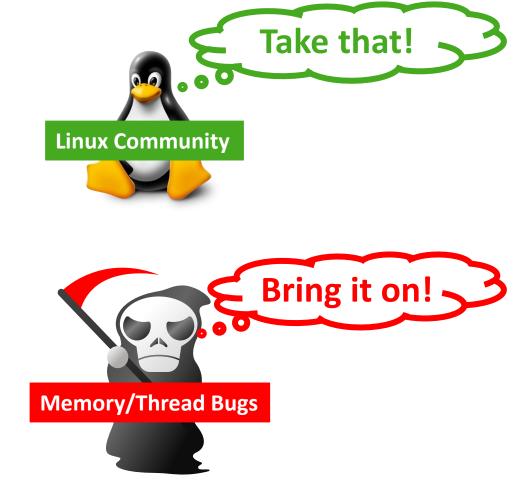


[	0.300014]	Kernel BUG at 0xffff80001019ed8c [verbose debug info unavailable]
[	0.300034]	Internal error: Oops - BUG: 0 [#1] PREEMPT SMP
[	0.300071]	Modules linked in:
[	0.300113]	CPU: 0 PID: 0 Comm: swapper/0 Not tainted 5.13.0-g7a293f65565c-dirty #616
[	0.300132]	Hardware name: linux,dummy-virt (DT)
[	0.300148]	IRQ stage: rros
[	0.300164]	pstate: 600000c5 (nZCv daIF -PAN -UAO -TCO BTYPE=)
I	0.300181]	pc : 0xffff80001019ed8c
[	0.300197]	lr : 0xffff80001019ecf8
[	0.300212]	sp : ffff8000105efd10
[	0.300228]	x29: ffff8000105efd10 x28: ffff800010488040 x27: 0000000000000000
[	0.300295]	x26: ffff8000105f0000 x25: ffff0000c0671580 x24: ffff8000105efdc0
I	0.300359]	x23: 0000000000000027 x22: ffff800010503340 x21: ffff0000ffa8ea90
I	0.300421]	x20: ffff0000ffa8eab0 x19: ffff0000ffa8eab0 x18: 0000000000000198
I	0.300484]	x17: 000000000000b67e x16: 00000000000000a4 x15: ffff8000101e7950
I	• <b>-</b>	x14: ffff80001033d4c0 x13: 0000000000000004b x12: 000000000000002b
[	0.300609]	x11: 000000000000000 x10: ffff80001031fb88 x9 : ffff800010321ed0
Γ	0.3006731	x8 : ffff80001031fb88 x7 : 000000000000000 x6 : ffff0000ffa81699

Projects	Туре	Mem bugs percentage
Chromium	User program	69%
Mozilla	User program	74%
Ubuntu	Kernel program	65%
Microsoft	Kernel Program	70%
Android	Kernel program	65%-90%
IOS/macOS	Kernel program	66.3%/71.5%

### Efforts to counter Memory/Thread bugs

- Static analysis<sup>[1]</sup>
  - Gcc Wanalyze\*
  - Clang
  - cppcheck
  - Codechecker
- Runtime detection<sup>[2]</sup>
  - Kernel Memory Sanitizer (KMSAN)
  - Kernel Concurrency Sanitizer (KCSAN)
  - Undefined Behavior Sanitizer (UBSAN)
- ➢ Kernel testing<sup>[2]</sup>
  - KUnit/Kselftest/LTP/Kernel CI/Fuzz



[1] <u>https://events.linuxfoundation.org/wp-content/uploads/2021/01/Static-Analysis-JSMoeller-LF-Live-Mentor-Series.pdf</u>
 [2] <u>https://www.kernel.org/doc/html/next/dev-tools/index.html</u>

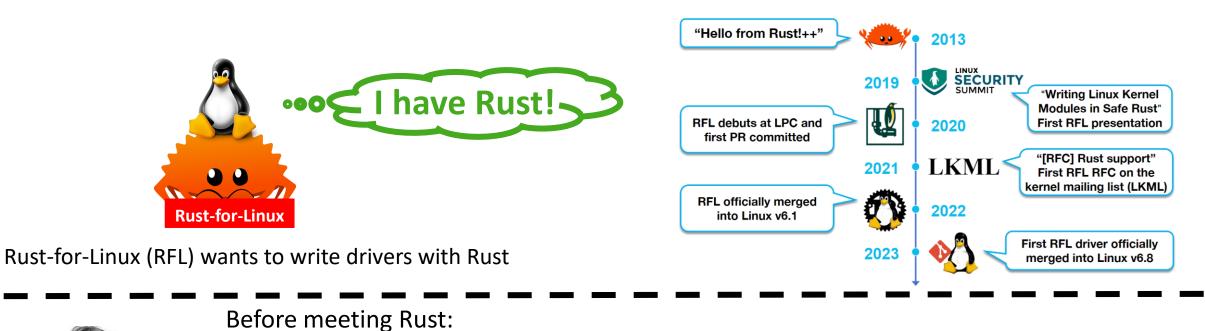
#### But Memory/Thread bugs still exist<sup>[1]</sup>

#### Vulnerabilities by types/categories

Year	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	File Inclusion	CSRF	XXE	SSRF	Open Redirect	Input Validation
2014	20	31	0	0	0	0	0	0	0	0	22
2015	13	17	0	0	0	0	0	0	0	0	5
2016	36	76	0	0	0	0	0	0	0	0	17
2017	64	86	0	0	1	0	0	0	0	0	20
2018	32	70	0	0	0	0	0	0	0	0	11
2019	30	124	0	0	1	0	0	0	0	0	4
2020	10	41	0	0	1	0	0	l'm <sup>°</sup> st	ill hor	0	2
2021	19	54	0	0	2				ill her	0	7
2022	41	149	0	0	0		0,00	0	0	0	2
2023	18	172	0	0	0		0	0	0	0	2
2024	30	452	0	0	<mark>ه Me</mark>	mory/Th	read Bug	<b>S</b> 0	0	0	0
Total	313	1272			5						92

[1] <u>https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor\_id=33</u>

#### Rust can help





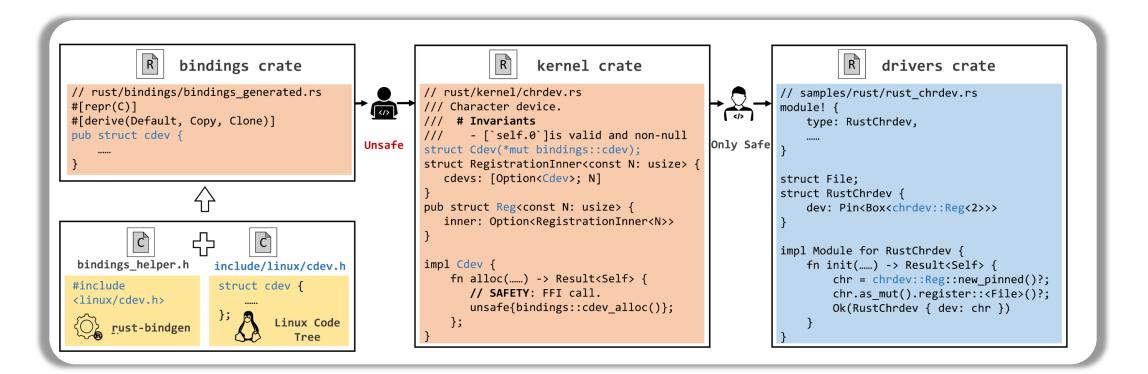
"I have yet to see a language that comes even close to **C**."

After meeting Rust:

"While I won't make any promises, I'd like to see **Rust** merging into the Linux kernel with the next release."

#### Background

- 1. Rust's safety rules cause limited expressiveness (Double linked list)
- 2. Code in the Unsafe block can break the safety rules
  - > Calling function from foreign function interface (FFI) needs unsafe blocks
- 3. It's proven possible to wrap unsafe blocks under safe APIs

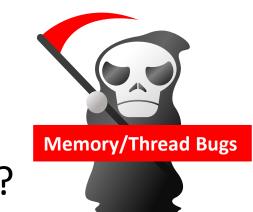


#### Motivation

#### While we know about Rust, RFL is rarely studied



RQ1: what is the status quo of RFL? RQ2: does RFL live up to the hype? RQ3: what are the lessons learned from RFL?

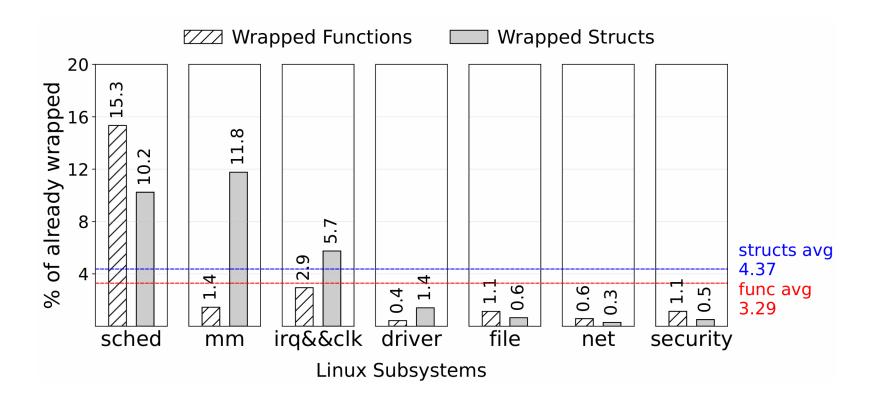


#### RQ1: what is the status quo of RFL?

Q1: RFL development statusQ2: How to construct safety abstractionQ3: How to rustify device drivers

#### Q1: RFL development status (RQ1)

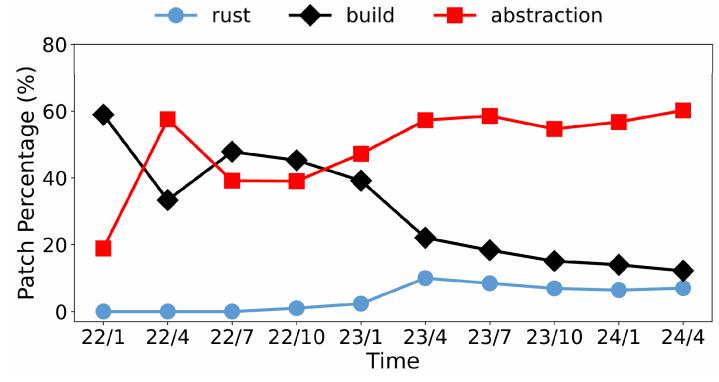
1. Development progress



Insight 1: drivers, netdev, and file systems are the long tail of RFL code.

### Q1: RFL development status (RQ1)

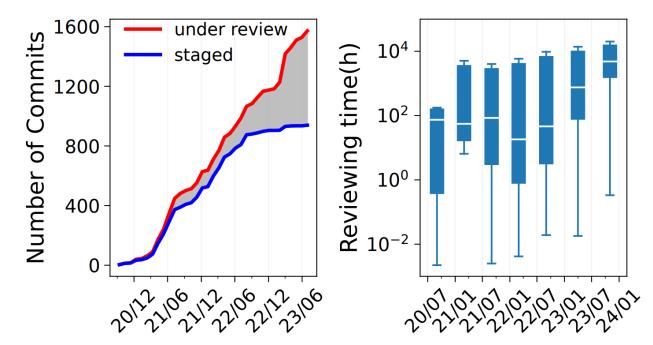
- 1. Development progress
- 2. Patch distribution



Insight 2: RFL infrastructure has matured, with safe abstraction and drivers being the next focus.

### Q1: RFL development status (RQ1)

- 1. Development progress
- 2. Patch distribution
- 3. The trend



Insight 3: RFL is bottlenecked by code review but not by code development.

- 1. Structs safety abstraction
  - rust-bindgen: same layout in memory

```
struct llist_head {
    struct llist_node *first;
};
```

```
#[repr(C)]
#[derive(Copy, Clone)]
pub struct llist_head {
    pub first: *mut llist_node,
}
```

```
impl Default for llist_head {
    fn default() -> Self {
        unsafe { ::core::mem::zeroed() }
    }
}
```

- 1. Structs safety abstraction
  - rust-bindgen: same layout in memory
    - Bit field/union

```
struct fs_parameter {
    const char *key; /* Parameter name */
    enum fs_value_type type:8; /* The type of value here
    */
    union {
        char *string;
        void *blob;
        struct filename *name;
        struct file *file;
    };
};
```

- 1. Structs safety abstraction
  - rust-bindgen: same layout in memory
    - Bit field/union

```
pub struct __BindgenBitfieldUnit<Storage, Align> {
    storage: Storage,
    align: [Align; 0],
}
```

impl<Storage, Align> \_\_BindgenBitfieldUnit<Storage, Align> {
 pub fn get(&self, bit\_offset: usize, bit\_width: u8) -> u64
 pub fn set(&mut self, bit\_offset: usize, bit\_width: u8, val: u64)

- 1. Structs safety abstraction
  - rust-bindgen: same layout in memory
  - Abstraction
    - Deref is valid: \*ptr -> foo<\*mut ptr>

```
impl File {
    /// Creates a reference to a [`File`] from a valid pointer.
    /// # Safety
    /// The caller must ensure that `ptr` is valid and remains valid for the lifetime of
    /// the returned [`File`] instance.
    pub(crate) unsafe fn from_ptr<'a>(ptr: *const bindings::file) -> &'a File {
        // SAFETY: The safety requirements guarantee the validity of the dereference,
        // while the `File` type being transparent makes the cast ok.
        unsafe { &*ptr.cast() }
    }
}
```

- 1. Structs safety abstraction
- 2. Functions safety abstraction
  - Functions as the members of the struct
    - /// # Invariants
    - /// # Safety
    - > // SAFETY:

```
impl File {
    /// Returns the flags associated with the file.
    ///
    /// The flags are a combination of the constants in [`flags`].
    pub fn flags(&self) -> u32 {
        // SAFETY: The file is valid because the shared reference guarantees a nonzero
    refcount.
        unsafe { core::ptr::addr_of!((*self.0.get()).f_flags).read() }
}
```

- 1. Structs safety abstraction
- 2. Functions safety abstraction
  - Functions as the members of the struct
  - Function pointers as *trait*

```
impl<T: Operations> OpsTable<T> {
    const VTABLE: bindings::dev_pm_ops = bindings::dev_pm_ops {
        suspend: Some(suspend_callback::<T>),
        resume: Some(resume_callback::<T>),
        freeze: Some(freeze_callback::<T>),
        restore: Some(restore_callback::<T>),
        };
    ......
}
```

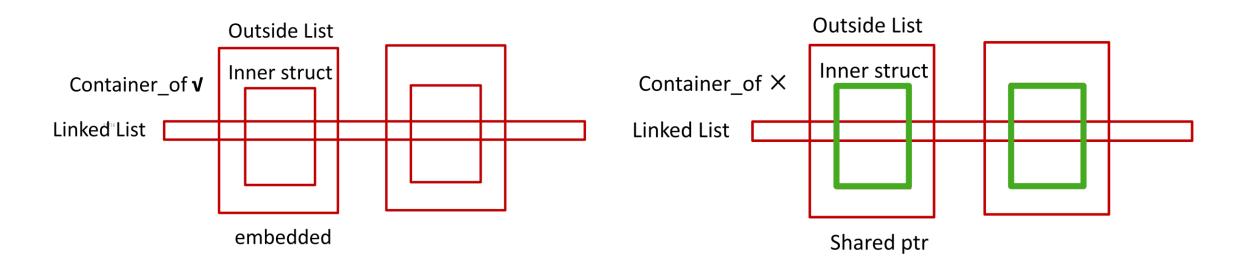
### Q3: How to rustify device drivers (RQ1)

- 1. Workflow
  - Device probe
  - > Driver logic
  - Device cleanup
- 2. Rust/RFL abstraction influences programing inflexibility

```
// In C
struct elements { int len; void* inner; };
struct factory {
         struct elements inner;
};
// In Rust with Fixed N
struct elements<const N: usize> {
         inner: [foo; N],
struct factory { inner: elements<256/8> }
// In Rust with Dynamic change N
struct thread/proxy<const N: usize>{
         thread/proxy elements: elements< N >,
impl dyn_num for thread<256>/proxy<8> {}
trait dyn_num { // fn use_elements(&self); }
struct factory { inner: &'static dyn dyn_n }
```

### Q3: How to rustify device drivers (RQ1)

- 1. Workflow
- 2. Rust/RFL abstraction influences programing flexibility
  - Container\_of



Insight 4: The major difficulty of writing safe drivers in Rust is to reconcile the inflexibility of Rust versus kernel programming conventions.

#### RQ2: does RFL live up to the hype?

Q1: Does Rust help Linux become safer?Q2: Does Rust bring additional overhead?Q3: How does Rust improve Linux development?

Fields	Goal	
Safety	Memory-safe and thread-safe drivers.	
Performance	Zero overhead on abstraction.	
Tools	Better documents and CI test quality.	
Efficiency	Higher development efficiency.	
Community	More developers in the kernel.	

#### Q1: Does Rust help Linux become safer? (RQ2)

- 1. There exist *soundness bugs* in the safety abstractions
  - Wrapping unsafe APIs needs manually review
  - Bugs may not disappear, just hide deeper<sup>[1]</sup>

Source	Compilation bug	Soundness bug
GitHub [22]	4(1/3)	7(3/4)
Intel LKP [41]	8(6/2)	0
Mailing List [44]	4(4/0)	2(1/1)

### Q1: Does Rust help Linux become safer? (RQ2)

- 1. There exist soundness bugs in the safety abstractions
- 2. The RFL drivers use *unsafe* blocks
  - > The driver itself still needs *unsafe* due to complex logic
  - > The safety abstraction is hard to maintain pure safe<sup>[1]</sup>

Driver	Number of Unsafe usage				
Diivei	Driver logic	Safety abstractions			
GPU [ <mark>67</mark> ]	107	7			
NVME [69]	44	16			
Null block [68]	0	0			
E1000 [62]	4	2			
Binder [59]	45	9			
Gpio_pl061 [64]	0	3			
Semaphore [70]	0	4			

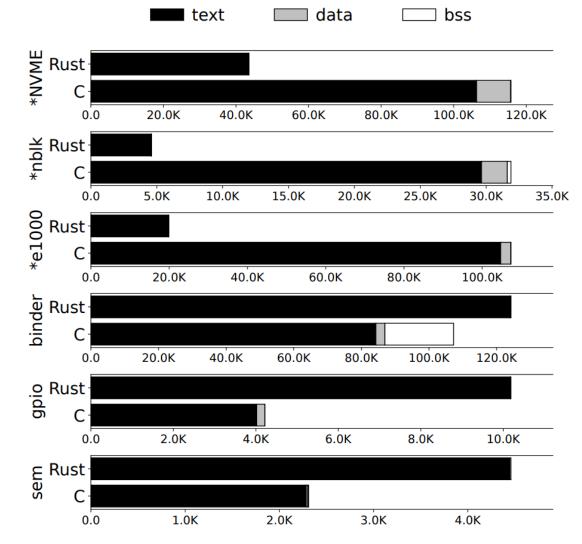
Insight 5: with RFL, Linux becomes more "securable" but still cannot be fully secure.

[1] <u>https://github.com/Rust-for-Linux/linux/commit/90e53c5e70a69159ec255fec361f7dcf9cf36eae</u>

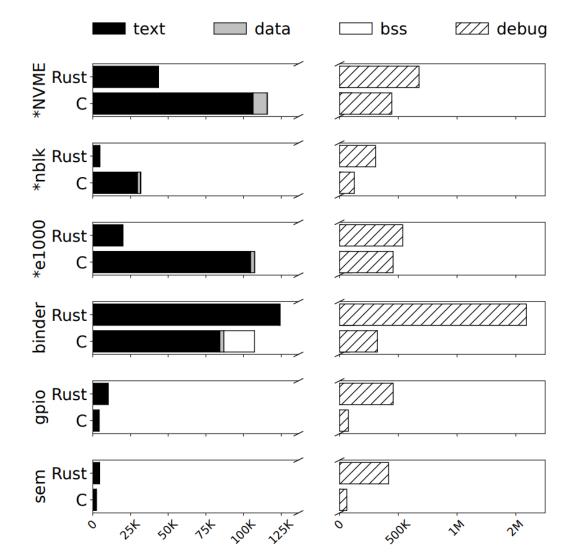
- 1. Setup
  - NVME and binder are considered the first batch of drivers to be merged in the Linux mainline
  - Others: File system/Network/Driver

Driver	Benchmark	Metrics		Device
NVME	fio		throughput	PC
Null Block	fio		throughput	PC
E1000	ping	driver	latency	PC
Binder	ping	size	latency	Raspi4b
Gpio_pl061	-		-	-
Semaphore	_		_	_

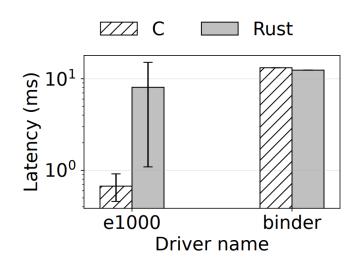
- 1. Setup
- 2. Binary size overhead
  - 1.2× for binder, 2.4× for gpio, and 1.9× for sem (\* means full feature in Rust)



- 1. Setup
- 2. Binary size overhead
  - 1.2× for binder, 2.4× for gpio, and 1.9× for sem (\* means full feature in Rust)
  - Rust brings overhead especially in the *debug* segmentation:
     3.9×-6.6× larger



- 1. Setup
- 2. Binary size overhead
- 3. Runtime overhead
  - Latency
    - Rust e1000 driver lacks features, e.g., prefetch



- 1. Setup
- 2. Binary size overhead
- 3. Runtime overhead
  - Latency
  - Throughput (poor)
    - Rust has higher cache miss rate due to smart pointer
- Rust: 78,280,692 L1-icache-load-misses 2,240,713 dTLB-load-misses
- C: 52,976,908 L1-icache-load-misses 1,312,452 dTLB-load-misses

Bs (KiB)	4         28         12         21         21           4         28         12         15         6           4         27         12         15         4           16         24         13         15         4           20         24         14         28         2           12         21         11         21           24         24         14         28         2           12         20         19         2         2           24         25         14         30         24           12         28         14         30         24           14         28         14         30         24           15         16         13         4         30         34           20         32         35         15         5         2           24         28         22         1         0         34           32         35         15         5         2           40         28         20         0         1           24         28         20         0         2 </th <th>34         15         17         33           32         11         19         0           30         19         20         .5           33         .16         .5         .10           34         .16         .11         .12           35         .16         .11         .12           35         .13         .11         .12           36         .1         .12         .13           36         .1         .22         .3         .21           37         .3         .4         .15           37         .4         .15         .12           39         .4         .15         .14           10         .15         .13         .14           10         .15         .3         .14           10         .14         .15         .14         .15           .7         .7         .8         .7           .1         .2         .3         .4         .15</th> <th>30       26       16       21         30       14       20       5         28       23       15       -2         26       60       -11       -2         28       27       -6       -11         23       28       -8       -11         24       1       1       -1         3       -3       2       8         4       1       1       -1         -1       -2       -7       -6         -4       10       -9       -9         -7       -11       -11       -10         -5       -6       -4       -4         -7       2.5       3       4         -7       -13       -14       -10         -8       -4       -4       -4         -7       -7       -7       -7         -7       -7       -7       -7         -7       -7       -7       -7         -7       -7       -7       -7         -7       -7       -7       -7         -7       -7       -7       -7      <tr tr="">        -7</tr></th> <th>261424281111311571027142552014422517422617423793332361310103220883213262821542038423123434</th> <th>40 0 0 0 - 0 0</th>	34         15         17         33           32         11         19         0           30         19         20         .5           33         .16         .5         .10           34         .16         .11         .12           35         .16         .11         .12           35         .13         .11         .12           36         .1         .12         .13           36         .1         .22         .3         .21           37         .3         .4         .15           37         .4         .15         .12           39         .4         .15         .14           10         .15         .13         .14           10         .15         .3         .14           10         .14         .15         .14         .15           .7         .7         .8         .7           .1         .2         .3         .4         .15	30       26       16       21         30       14       20       5         28       23       15       -2         26       60       -11       -2         28       27       -6       -11         23       28       -8       -11         24       1       1       -1         3       -3       2       8         4       1       1       -1         -1       -2       -7       -6         -4       10       -9       -9         -7       -11       -11       -10         -5       -6       -4       -4         -7       2.5       3       4         -7       -13       -14       -10         -8       -4       -4       -4         -7       -7       -7       -7         -7       -7       -7       -7         -7       -7       -7       -7         -7       -7       -7       -7         -7       -7       -7       -7         -7       -7       -7       -7 <tr tr="">        -7</tr>	261424281111311571027142552014422517422617423793332361310103220883213262821542038423123434	40 0 0 0 - 0 0
blocksize (KiB)	2         29         1         5         2         1         0           4         -3         -1         -1         1         3         0           4         -3         -1         -1         1         3         0           8         -12         -2         5         -7         -6         -11           90         -1         -9         -4         -3         -2         -9           16         -0         -9         -4         -3         -2         -9           2         -0         -5         -2         1         6         1           4         -5         2         3         0         -2         0           4         -5         2         3         5         -2         6         1           5         -0         -1         2         3         -0         -2         0           16         -12         -8         -8         -6         -1         5         -2         4         1         -0         -0         -0         -0         -0         -0         -0         1         0         -0         -0         -0	B         -1         3         -1         4         2         9         -1         3         -1         4         2         -1         5         -2         1         6         2         -1	v         v         v         v           0         0         0         0         0         0           11         10         12         10         9         8           2         3         -2         11         9         4           -0         -3         2         23         7         33           -7         -20         -0         0         -28         1           -13         -12         -12         -10         10         12         12           -14         7         10         12         12         10         11           12         -3         20         21         27         10         12           14         6         13         -1         2         -1         10           14         6         13         12         11         13         12         11           10         -3         10         15         14         7         13         12         11           10         -3         10         15         14         7         14         1         1         1         1         1 <td< th=""><th>-2         23         -2         23         -29         5           5         -37         28         -20         -12         -12           0         -22         -35         -44         -19         -3           2         -23         -12         -11         -52         -13           2         -23         -12         -11         -52         -13           2         -17         -27         -20         -9         18           -16         -16         -28         46         -62         -4           -3         -35         -27         -20         -17         -16           -3         -16         -3         -22         -15         -4           -3         -16         -3         -22         -11         24           -1         -14         -21         -26         -11         24           -24         -27         -29         -29         36         -8           -3         -19         -30         25         -2         -22           0         -24         42         -9         55         -27           -1         -17</th><th>60 (%) 40 00 40 00 20 0 20 0 20 0 20 0 20 0 20</th></td<>	-2         23         -2         23         -29         5           5         -37         28         -20         -12         -12           0         -22         -35         -44         -19         -3           2         -23         -12         -11         -52         -13           2         -23         -12         -11         -52         -13           2         -17         -27         -20         -9         18           -16         -16         -28         46         -62         -4           -3         -35         -27         -20         -17         -16           -3         -16         -3         -22         -15         -4           -3         -16         -3         -22         -11         24           -1         -14         -21         -26         -11         24           -24         -27         -29         -29         36         -8           -3         -19         -30         25         -2         -22           0         -24         42         -9         55         -27           -1         -17	60 (%) 40 00 40 00 20 0 20 0 20 0 20 0 20 0 20

Number of IO jobs

- 1. Setup
- 2. Binary size overhead
- 3. Runtime overhead
  - > Latency
  - Throughput (poor)
    - Rust has higher cache miss rate due to smart pointer
    - Rust runtime checks/bit field translation

seq 4 28 12	write s	eq read	rand write	rand read 26 14 24 22	40 0 30 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
8 27 12	-15     6     32	11 <mark>-19</mark> 0	35     14     -20     5	38     11     -19     11	- 30
원 16 24 13 은 32 24 14	-15 4 30	<b>19</b> -20 -5	28 23 -15 -2	34 15 -7 10	
	-8 2 23	-16 -5 -10	26 -30 -5 -3	27 14 -5 5	-20 Sng
$64 \ 27 \ -22$	1 -2 15	-31 -11 -12	18         -27         -6         -11           23         -28         -8         -11	33         -21         4         2           25         -17         4         2	10 5
8 120	2 -2 22	-31 -11 -13			ority
Seq Seq S		eq read	rand write	rand read	o o
	30     24     -2       7     10     -2	1 -2 -2 -3 -4 -15	3 -3 2 8 4 1 1 -1	37         9         33         32           36         16         -13         10	-10 5
$\frac{1}{10}$ $\frac{1}{10}$ $\frac{1}{10}$ $\frac{1}{10}$ $\frac{1}{10}$ $\frac{1}{10}$ $\frac{1}{10}$	-13 4 -12	-3 -12 -14	-1 -2 -7 -6	32 20 -8 8	20 E
<del>0</del> 32 35 15	-5 2 -10	-15 -13 -14	-4 -10 -9 -9	32 13 2 6	20 E
au       8       23       17         Be       16       28       16         au       32       35       15         bu       64       28       -22	1 0 -10	-14 -14 -15	-7 -11 -11 -10	28 -21 5 4	20 em 30 em 30 em 40 em 40 em
<sup>2</sup> 128 <sup>23</sup> -20	0 -1 -7	-7 -8 -7	-5 -6 -4 -4	20 -18 4 2	e Lo
1 2	341	2 3 4	1 2 3 4	1 2 3 4	
		Number of	IO jobs		
seq w	vrite s	eq read	rand write	rand read	
2 <mark>29 1 5</mark>		3 -1 4 2	0 0 0 0 0	5 -2 23 65 -29 5	60 🛞
∥ 4 <mark>-3 -1 -1</mark> to 8 <mark>-12 -12 -5</mark>		2 1 6 2 -1 0 -10 -14 -6 -7	11 10 12 10 9 8 2 3 -2 11 9 4	5 -37 -28 -20 -12 -12 0 -22 -35 -44 -19 -3	U U
$\begin{bmatrix} 1 & 2 \\ -4 & -3 & -1 & -1 \\ +4 & 8 & -12 & -12 & -5 \\ -10 & -9 & -4 \\ -10 & -10 & -10 & -4 \\ -10 & -10 & -10 & -4 \\ -10 & -10 & -10 & -4 \\ -10 & -10 &$			2 3 -2 11 9 4 -0 -3 2 23 7 33	0 -22 -35 -44 -19 -3 2 -23 -12 -11 -52 -13	40
······································			<b>27</b> -20 -0 0 -28 1	2 -17 -27 -20 -9 18	st C
<u></u>					20 J
	3 -2 -0 2 0		13 -12 -12 -12 -10 -10	-16 -16 -28 -46 -62 -4	
C 2 4 5 2 3 3 2 2 2 4 2 2 2 2 4 2 2 2 2 2 2 2 2	0 -2 0 3 -1 -7 -3 -12 -5 -1		14 7 10 12 12 9 -6 -1 1 12 0 1	3       -35       -27       -20       -17       -16         2       -22       -54       -24       -27       -4	1 orit
Y p 16 -12 -8 -8	-6 -1 -5 -8 -7		1 2 -3 20 21 27	-3 -16 -3 -22 -15 -4	ber
0       0       25       2       3         0       0       25       2       3       3         10       10       10       10       10       10         10       10       25       10       10       10       10         10       10       10       10       10       10       10       10         10<	-2 6 1 1 -5	5 -4 -0 -0 -0	11 46 13 -1 2 -1	-1 -14 -21 -26 -11 24	Su Su
+ 2 29 0 -1 ↓ 4 -2 1 2	2 3 -0 5 0 -0 1 -0 9 -4		12 -12 -12 -12 -10 -10 9 6 13 13 12 11	-24 -24 -17 -41 -14 -6 5 -32 -39 -29 36 -8	- 60 - 40 - 40 - 40 - 20 - 20 - 20 - 20 - 20 - 20 - 20 - 2
<u> </u>	-4 -2 -9 -12 -8		0 -3 10 15 14 7	-3 -19 -30 -25 -2 -22	-40 <u>g</u>
e 16 -10 -9 -4	-4 -3 -5 -9 -7		-0 2 12 27 18 -15	0 -24 -42 -9 -55 -27	еЪ
32 -6 -3 -4	-2 4 -0 -5 -5		24 9 4 -1 1 3	-1 -17 -19 -30 8 26	-60 <u>–</u>
124	8 1632 1 2	4 8 1632	1 2 4 8 1632	1 2 4 8 1632	2
		Number	IO laba		_

Number of IO jobs

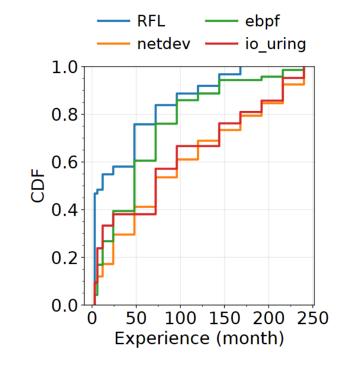
- 1. Setup
- 2. Binary size overhead
- 3. Runtime overhead
  - Latency
  - Throughput (better)
    - Rust use less cache lines (pahole)
    - Less code path

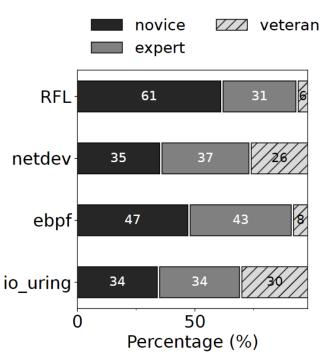
Image: First Substraint of the series of	Selection           34         15         17         33           32         11         19         0           30         19         20         .5           33         16         .5         .10           34         .16         .5         .10           35         .16         .5         .10           15         .30         .11         .12           22         .30         .11         .12           22         .30         .11         .12           -2         .1         .2         .2           -2         .1         .2         .2           -12         .3         .4         .15           -12         .3         .14         .15           -12         .3         .12         .14           -10         .15         .13         .14           .10         .15         .13         .14           .10         .14         .14         .15           .7         .7         .8         .7           1         .2         .3         .4	Part of the section of	14242226142421381110113415771027142553321422517423793332361613103220883213263421542018421234	40 30 20 10 - 10 - 20 - 10 - 10 - 20 - 10 - 20 - 10 - 20 - 10 - 20 - 20 - 30 - 20 - 40 - 20 - 40 - 20 - 40 - 20 - 20 - 20 - 20 - 20 - 20 - 20 - 2
2       29       1       5       2       1       0         4       -3       -1       -1       1       3       0         1       4       -3       -1       -1       1       3       0         1       -10       -0       -4       -3       -2       -9         1       1       -0       -4       -3       -2       -9         3       -0       -0       -3       -2       -0       1       -0         3       -0       -0       -0       -3       -2       -0       1       -0         1       2       3       0       -2       -0       1       -0	Sevential         Sevential <t< td=""><td>Image: Normal Structure       Image: Normal Structure</td></t<> <td>-2       23       -29       5         5       -27       23       -29       5         5       -37       -28       -20       -12       -12         0       -22       -35       -44       -19       -3         2       -23       -12       -11       -52       -13         2       -27       -20       -9       18         -16       -16       -28       -46       -62       -4         3       -35       -27       -20       -9       18         -16       -16       -28       -46       -62       -4         3       -35       -27       -20       -9       18         -16       -16       -28       -46       -62       -4         -3       -16       -28       -24       -27       -4         -3       -16       -3       -22       -5       -4         -1       -14       -21       -26       -11       24         -24       -24       -17       -41       -14       -6         5       -32       -39       -29       36       -8         -3</td> <td>- 60 - 0 - 40 - 0 - 0 - 0 - 0 - 0  10  0  10  10  10  10  0  0 </td>	Image: Normal Structure	-2       23       -29       5         5       -27       23       -29       5         5       -37       -28       -20       -12       -12         0       -22       -35       -44       -19       -3         2       -23       -12       -11       -52       -13         2       -27       -20       -9       18         -16       -16       -28       -46       -62       -4         3       -35       -27       -20       -9       18         -16       -16       -28       -46       -62       -4         3       -35       -27       -20       -9       18         -16       -16       -28       -46       -62       -4         -3       -16       -28       -24       -27       -4         -3       -16       -3       -22       -5       -4         -1       -14       -21       -26       -11       24         -24       -24       -17       -41       -14       -6         5       -32       -39       -29       36       -8         -3	- 60 - 0 - 40 - 0 - 0 - 0 - 0 - 0 0 0 0 0 0 0 0 0 0 0 0 0 10 0 10 10 10 10 0 0 

- 1. Improved code quality and readability
  - RFL improves the Linux documentation coverage by *rustdoc*
  - RFL has the built-in CI system which improves the code quality

Subsystems	Docs%	CI errors/10K LoC
RFL	100%	3.8
ebpf	15%	7.5
io_uring	31%	11.9

- 1. Improved code quality and readability
- 2. More young blood to the Linux community
  - RFL has the most novice developers
  - We observe 5 out 6 RFL drivers are developed by the non-novice





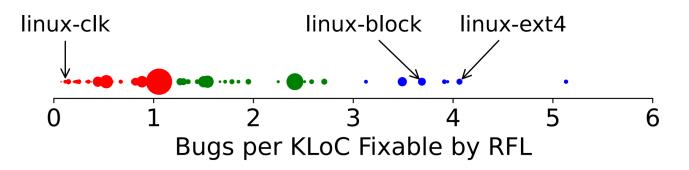
#### RQ3: what are the lessons learned from RFL?

For the developers using and building RFL

- 1. use Rustbelt/miri to evaluate the correctness of safety abstraction
- 2. write your program with ownership in mind
- 3. accept unsafe if you have to

For the developers expanding RFL scope

1. choose the subsystem/drivers that are more fragile



#### Takeaways

- 1. Memory safety *with the price*
- 2. No sliver bullet and no guarantee
- 3. Nearly zero-cost

# Shout out to the RROS team and my advisors!



≡ 🜍 BUPT-OS / RROS	Q Type [] to searc	h	>_   <b>  + →</b>	0 II 6 💌
Code O Issues 7 1 Pull requests 4 Q Discussions	🕑 Actions 🖽 Projects 🖽 Wiki 🛈	Security 🗠 Insights	鈞 Settings	
C RROS (Public)	🖈 Edit Pins 👻	⊙ Watch 8 →	<sup>ల్ల</sup> Fork 34 ా	☆ Star 510 -
🐉 main 👻 🐉 2 Branches 🛇 0 Tags	Q Go to file	- <> Code -	About	
StevenFryto and ruiqurm rust: dovetail: improve the subscriber	o 🕚 68 Commits	RROS is a dual-kerne other scenarios that r	need both real-time	
.github/workflows CI: Add pre-	-check make LLVM=1 rustfmtcheck		and general-purpose RTOS (Rust) + Linux (	



#### Hongyu Li



#### Liwei Guo



#### Yangye Xuan



#### Shangguang wang



#### Mengwei Xu







# Thanks Q&A

Source code: <a href="https://github.com/Richardhongyu/rfl\_empirical\_tools">https://github.com/Richardhongyu/rfl\_empirical\_tools</a>

RROS: <a href="https://github.com/BUPT-OS/RROS/">https://github.com/BUPT-OS/RROS/</a>

Email: lihongyu1999@bupt.edu.cn, lwg@uestc.edu.cn