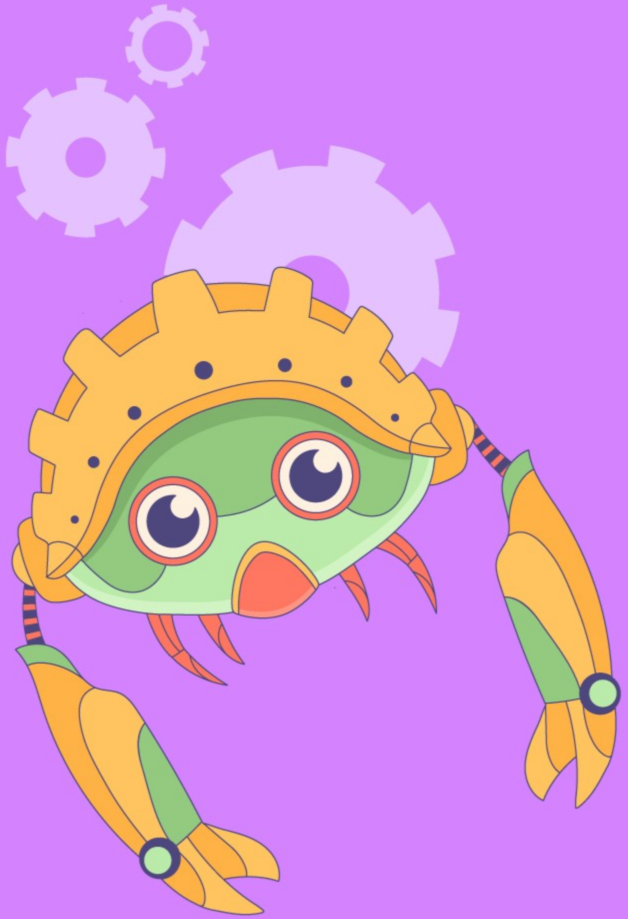
MICHAL ROSTECKI

Rust Engineer at Exein

Enhancing Rust with BTF Debug Format Support



INTRODUCTION

- What is eBPF?
  - How it differs from kernel modules?
  - What is BTF? How is it produced?

- **You can write eBPF in Rust!** But there is still some work to be done to meet 100% feature parity.

INTRODUCTION

- There are two ways of extending the Linux kernel
  - Kernel modules
  - eBPF programs

The difference

Kernel modules vs eBPF

- **Kernel modules**
  - Have full access to kernel internals.
  - Need kernel sources for building.
  - It’s encouraged to submit them upstream, to the main kernel repo.
  - **Can crash the kernel.**
  - **Great for device drivers, filesystems and extending kernel’s functionality directly.**

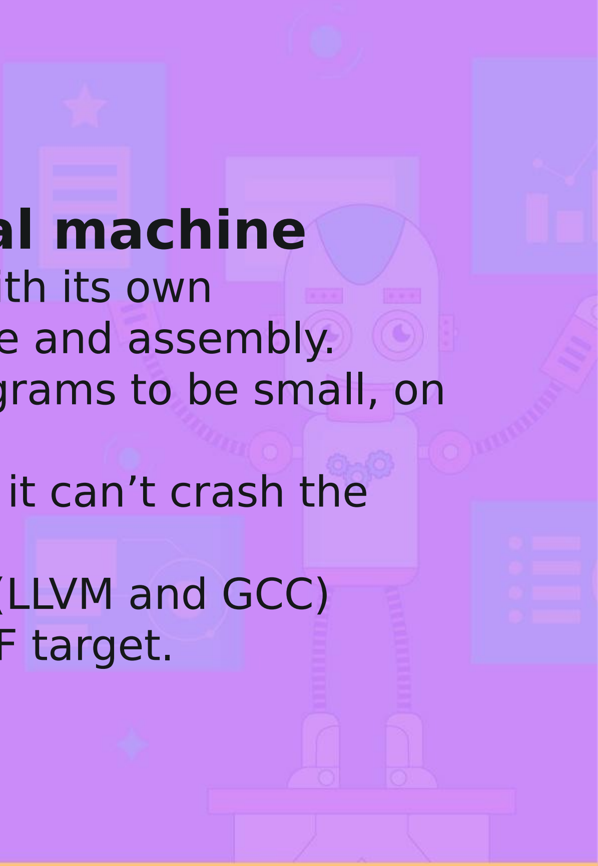
The difference

Kernel modules vs eBPF

- **eBPF**
  - Built independently from the kernel. You just need a compiler supporting BPF target.
  - **Cannot crash the kernel.**
  - **Great for third-party tools which hook into Linux internals – firewalls, tracers, security monitors.**

Tell me more

How eBPF works?



- **It’s a virtual machine**
  - It comes with its own architecture and assembly.
  - Forces programs to be small, on purpose.
  - That’s why it can’t crash the kernel.
  - Compilers (LLVM and GCC) support BPF target.

- **Kernel modules vs BPF** C vs Rust

Aya

- **Rust library for writing eBPF**
  - https://aya-rs.dev
  - Rustup and cargo is all you need!

Security observability framework
for IoT devices (but not only!)

Pulsar

PULSAR

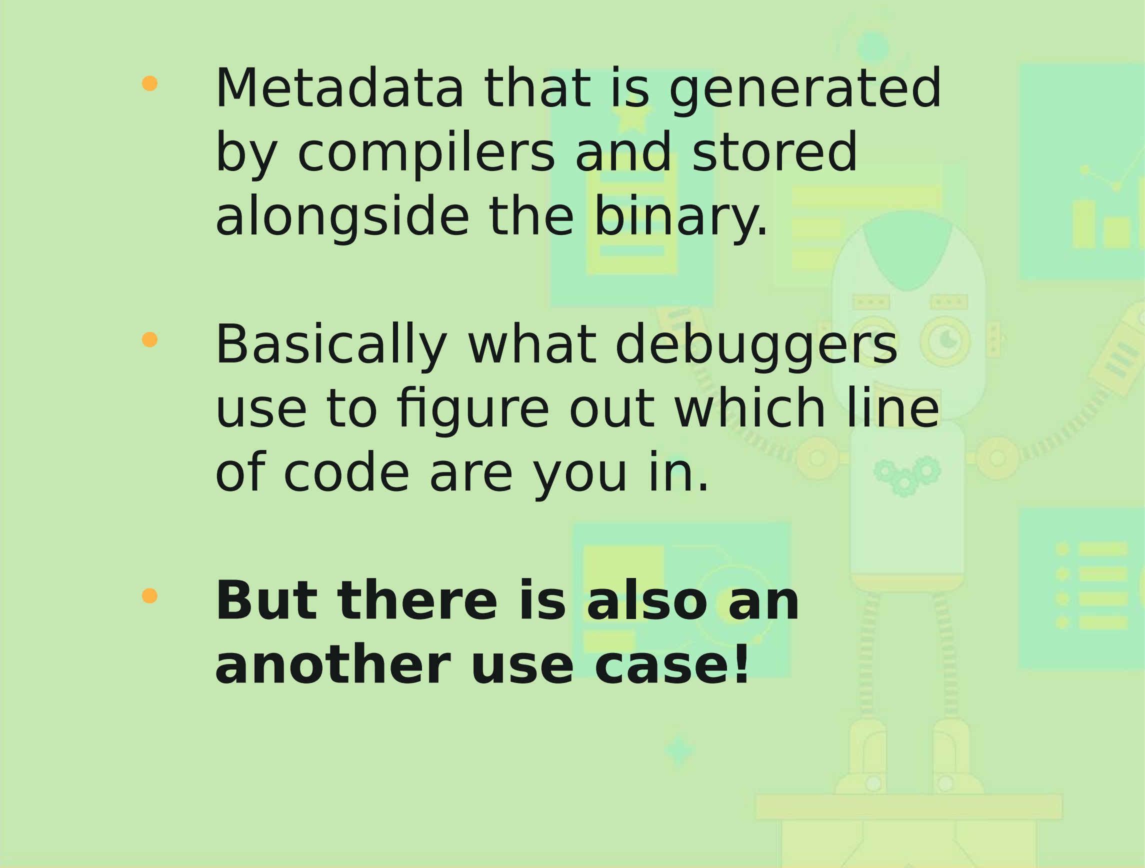
- **Security monitor, using Rust and Aya**
  - https://pulsar.sh/
  - Lightweight.
  - Designed mostly for IoT devices. Has support for ARM and RISC-V.
  - But useful on x86 as well.

- **Using eBPF for network tracing** XDP

- **Using eBPF for network filtering** XDP

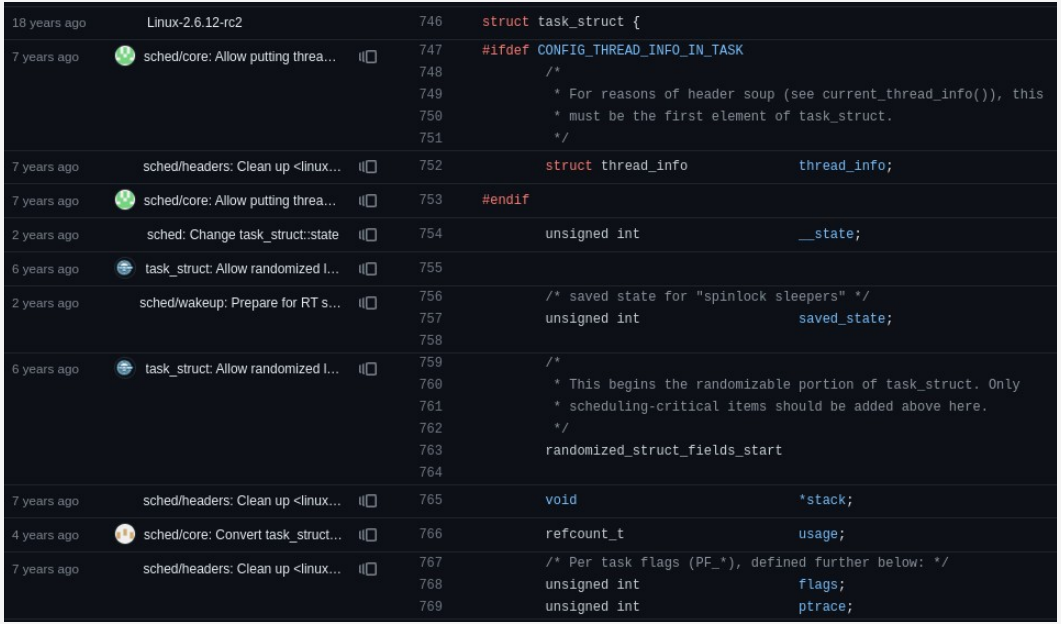
Here be dragons

What is Debug Info?



- Metadata that is generated by compilers and stored alongside the binary.
- Basically what debuggers use to figure out which line of code are you in.
- **But there is also an another use case!**

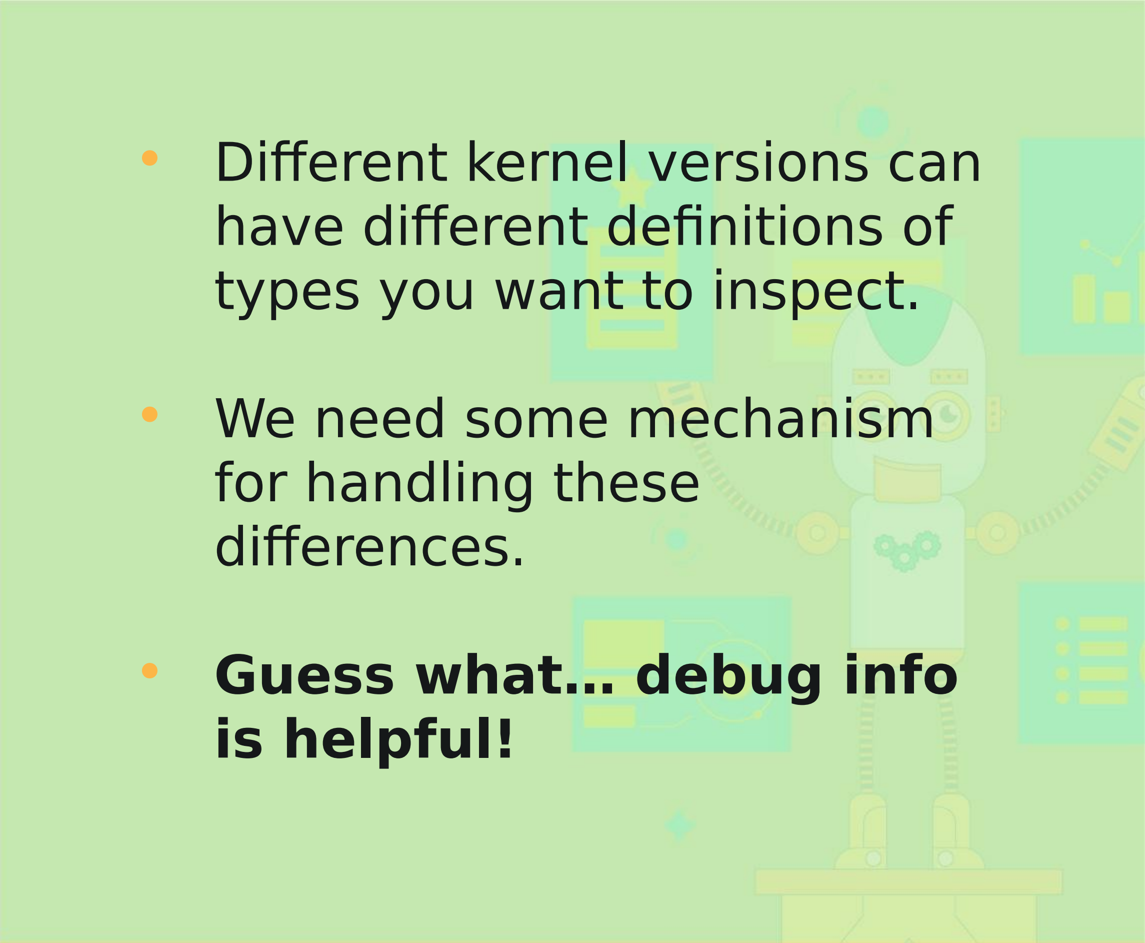
- **Let’s look at some kernel type to trace** Whoops, we have a problem!



```
18 years ago   Linux-2.6.12-rc2                      struct task_struct {
7 years ago    sched/core: Allow putting threa...    #ifdef CONFIG_THREAD_INFO_IN_TASK
                                                             /*
                                                              * For reasons of header soup (see current_thread_info()), this
                                                              * must be the first element of task_struct.
                                                              */
7 years ago    sched/headers: Clean up <linux...             struct thread_info              thread_info;
7 years ago    sched/core: Allow putting threa...    #endif
2 years ago    sched: Change task_struct::state              unsigned int                    __state;
6 years ago    task_struct: Allow randomized l...
2 years ago    sched/wakeup: Prepare for RT s...             /* saved state for "spinlock sleepers" */
                                                             unsigned int                    saved_state;

6 years ago    task_struct: Allow randomized l...            /*
                                                              * This begins the randomizable portion of task_struct. Only
                                                              * scheduling-critical items should be added above here.
                                                              */
                                                             randomized_struct_fields_start

7 years ago    sched/headers: Clean up <linux...             void                            *stack;
4 years ago    sched/core: Convert task_struct...            refcount_t                      usage;
7 years ago    sched/headers: Clean up <linux...             /* Per task flags (PF_*), defined further below: */
                                                             unsigned int                    flags;
                                                             unsigned int                    ptrace;
```

Compatibility

How to support many kernel versions?



- Different kernel versions can have different definitions of types you want to inspect.
- We need some mechanism for handling these differences.
- **Guess what… debug info is helpful!**

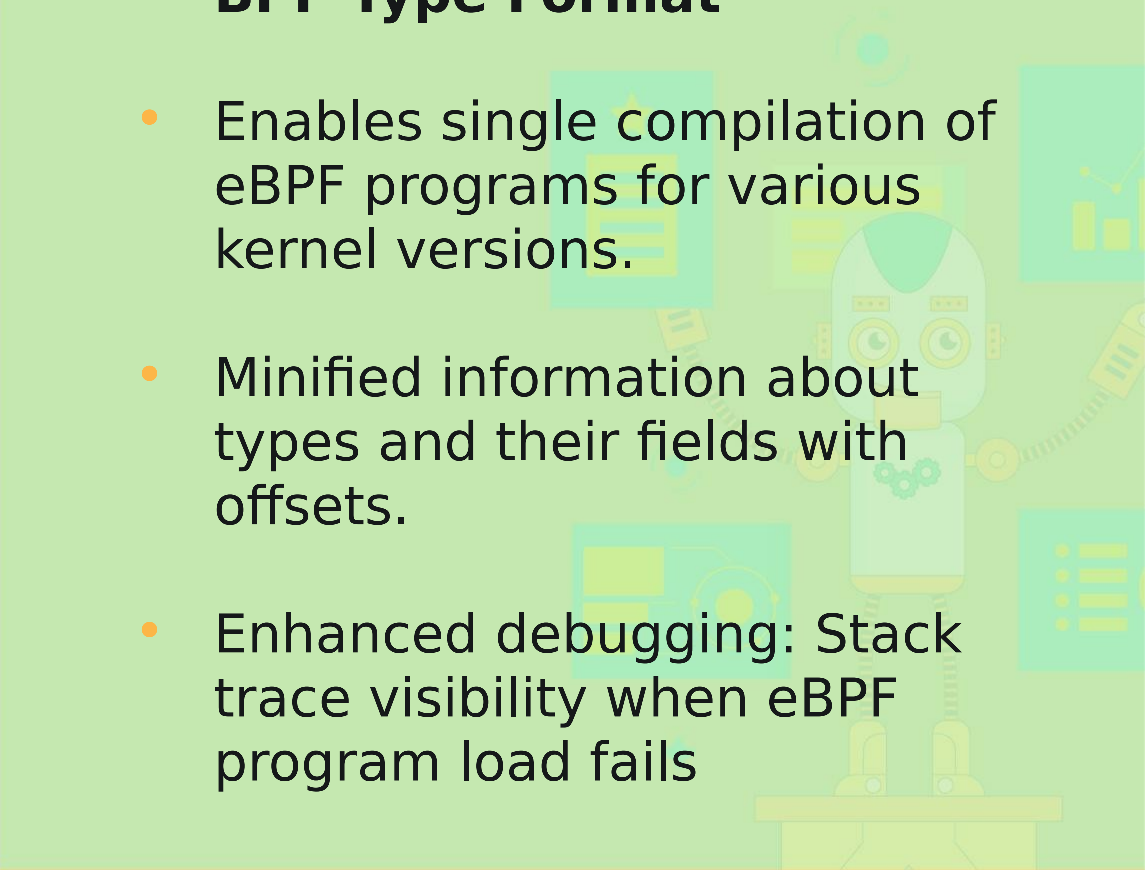
The most popular format

DWARF

- The most popular debug information standard.
- Again, that’s basically what your debugger uses!
- **It’s quite big! It can take megabytes.**

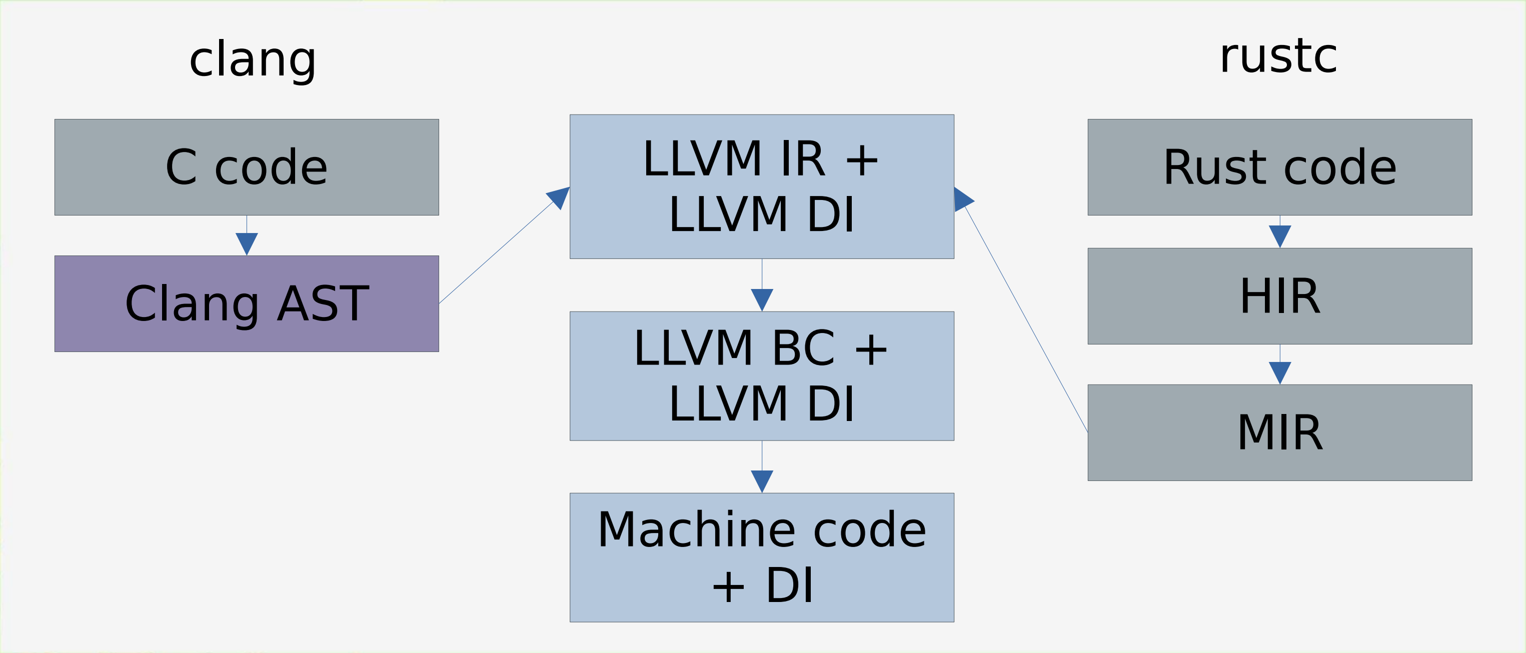
Here it comes

BTF



- **BPF Type Format**
- Enables single compilation of eBPF programs for various kernel versions.
- Minified information about types and their fields with offsets.
- Enhanced debugging: Stack trace visibility when eBPF program load fails

- **LLVM** How does Debug Information get produced?



- **LLVM Debug Info** Example

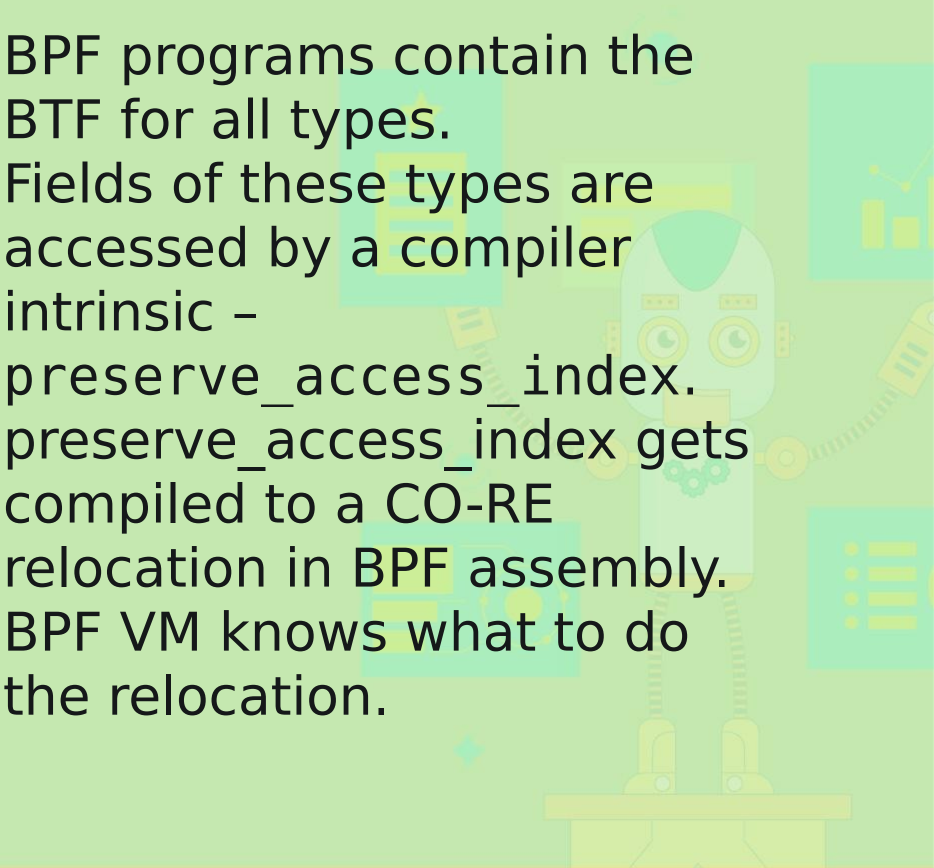
```
!28223 = distinct !DICompositeType(tag: DW_TAG_structure_type, name:
"sockaddr_in6", file: !6, line: 22816, size: 224, elements: !28224)

!28224 = !{!28225, !28226, !28227, !28228, !28229}

!28225 = !DIDerivedType(tag: DW_TAG_member, name: "sin6_family", scope: !28223,
file: !6, line: 22817, baseType: !682, size: 16)

!28226 = !DIDerivedType(tag: DW_TAG_member, name: "sin6_port", scope: !28223, file:
!6, line: 22818, baseType: !990, size: 16, offset: 16)

!28227 = !DIDerivedType(tag: DW_TAG_member, name: "sin6_flowinfo", scope: !28223,
file: !6, line: 22819, baseType: !971, size: 32, offset: 32)

!28228 = !DIDerivedType(tag: DW_TAG_member, name: "sin6_addr", scope: !28223, file:
!6, line: 22820, baseType: !9016, size: 128, offset: 64)

!28229 = !DIDerivedType(tag: DW_TAG_member, name: "sin6_scope_id", scope: !28223,
file: !6, line: 22821, baseType: !899, size: 32, offset: 192)
```

- **BTF** Example

```
[200] STRUCT 'sockaddr_in6' size=28 vlen=5

        'sin6_family' type_id=19 bits_offset=0

        'sin6_port' type_id=24 bits_offset=16

        'sin6_flowinfo' type_id=14 bits_offset=32

        'sin6_addr' type_id=35 bits_offset=64

        'sin6_scope_id' type_id=15 bits_offset=192
```

BTF relocations

How do they work?

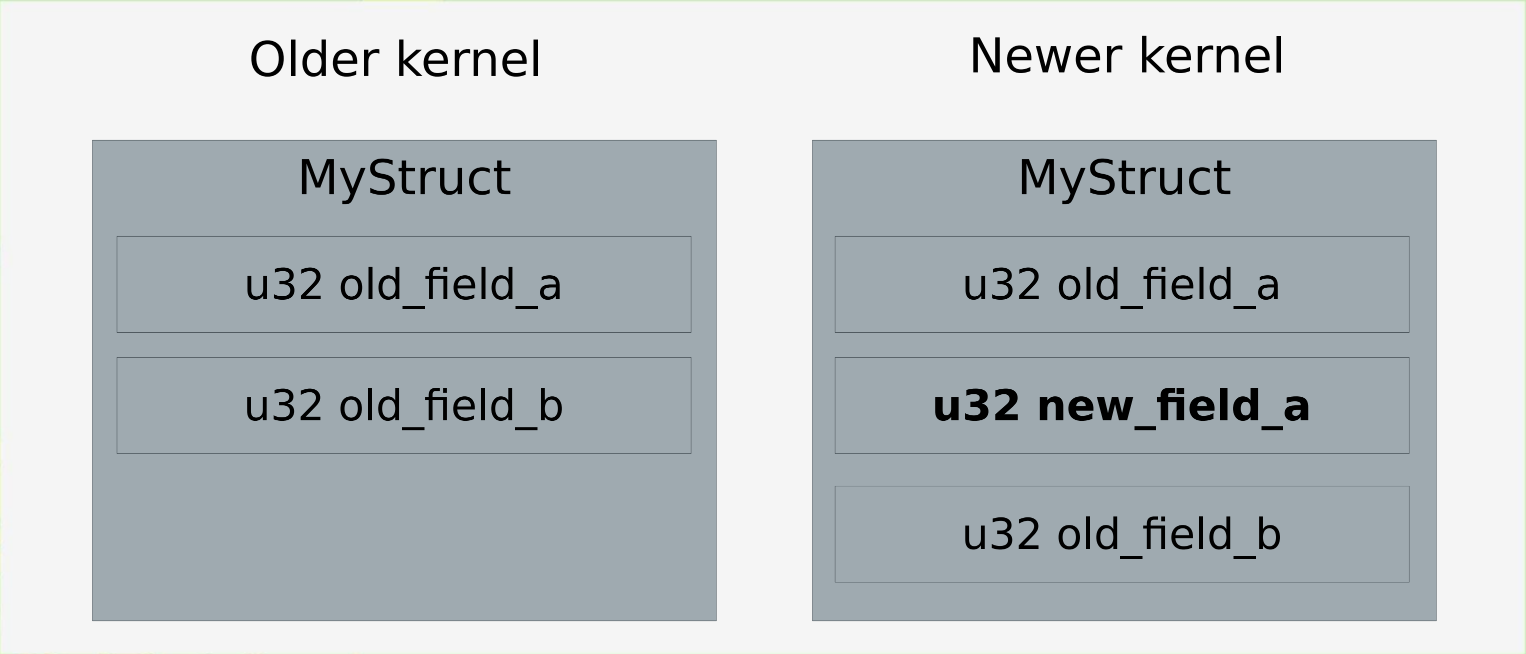


- BPF programs contain the BTF for all types.
- Fields of these types are accessed by a compiler intrinsic – `preserve_access_index`.
- preserve_access_index gets compiled to a CO-RE relocation in BPF assembly.
- BPF VM knows what to do the relocation.

- **preserve_access_index** Intrinsic

```
declare <ret_type>

@llvm.preserve.union.access.index.p0s_union.anons.p0s_union.anons(

    <type> base, i32 di_index)



declare <ret_type>

@llvm.preserve.struct.access.index.p0i8.p0s_struct.anon.0s(

    <type> base, i32 gep_index, i32 di_index)
```

- **BTF relocations** How do they work?



Older kernel

MyStruct

u32 old_field_a

u32 old_field_b

Newer kernel

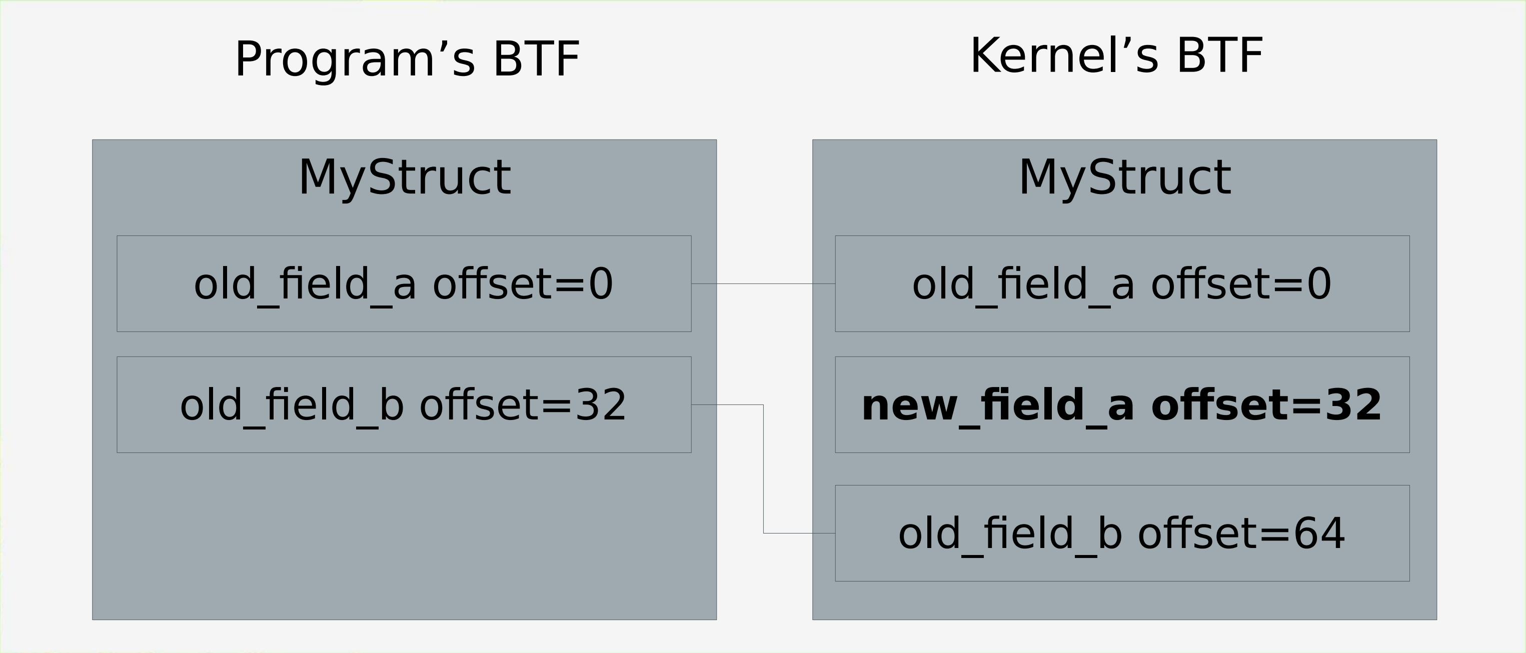
MyStruct

u32 old_field_a

u32 new_field_a

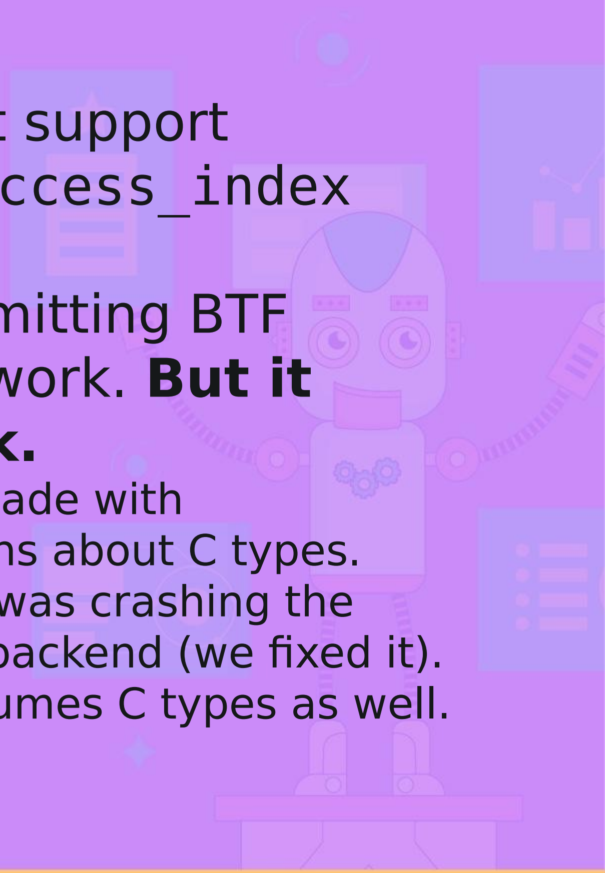
u32 old_field_b

- **BTF relocations** How do they work?



The Challenge

BTF - Aya's Feature Gap



- Rust doesn’t support `preserve_access_index` intrinsic.
- In theory, emitting BTF should just work. **But it didn’t work.**
    - BTF was made with assumptions about C types.
    - Initially, it was crashing the LLVM BPF backend (we fixed it).
    - Kernel assumes C types as well.

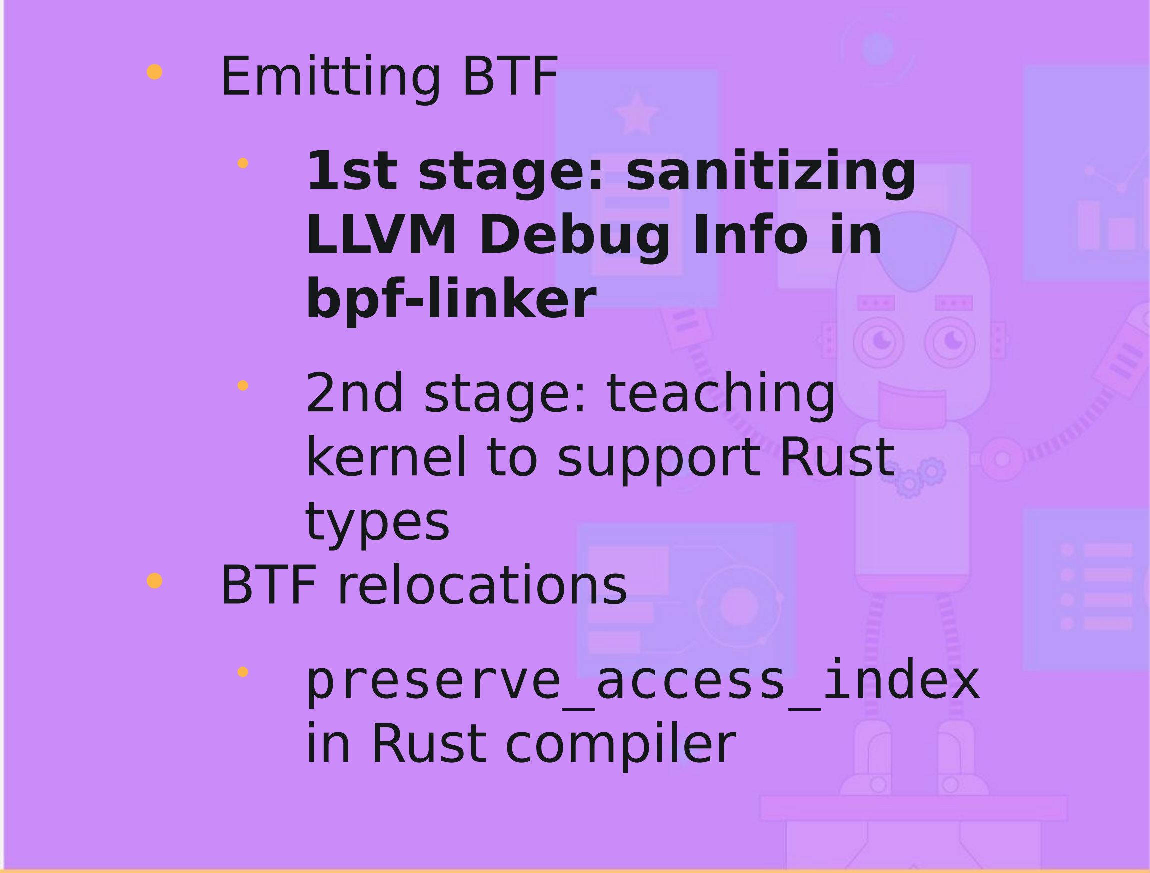
Be more precise, please

What does the kernel assume?

- **It gets annoyed by Rust-specific types**
  - Data-carrying enums.
  - Non-alphanumeric type names (e.g. types with generics).
- **It expects types Rust doesn’t support**
  - BPF map types have to be anonymous structs.
  - Universal eBPF program compilation for all kernel versions
  - Simplified debugging with stack traces

How can we fix it

Stages of BTF support

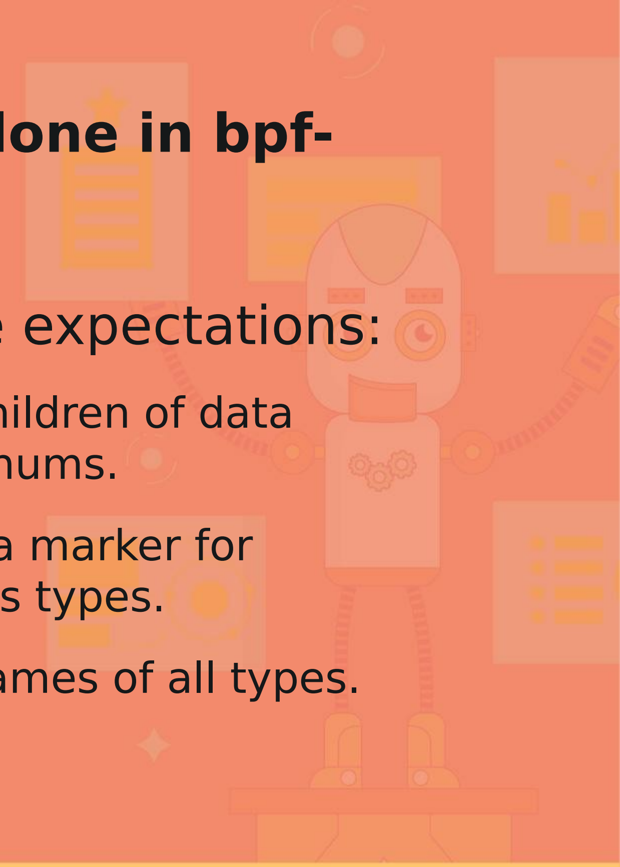


- Emitting BTF
  - **1st stage: sanitizing LLVM Debug Info in bpf-linker**
  - 2nd stage: teaching kernel to support Rust types
- BTF relocations
  - `preserve_access_index` in Rust compiler

The 1st stage

Sanitizing LLVM DI

- **It can be done in bpf-linker.**
- To meet the expectations:
  - Remove children of data carrying enums.
  - Introduce a marker for anonymous types.
  - Sanitize names of all types.

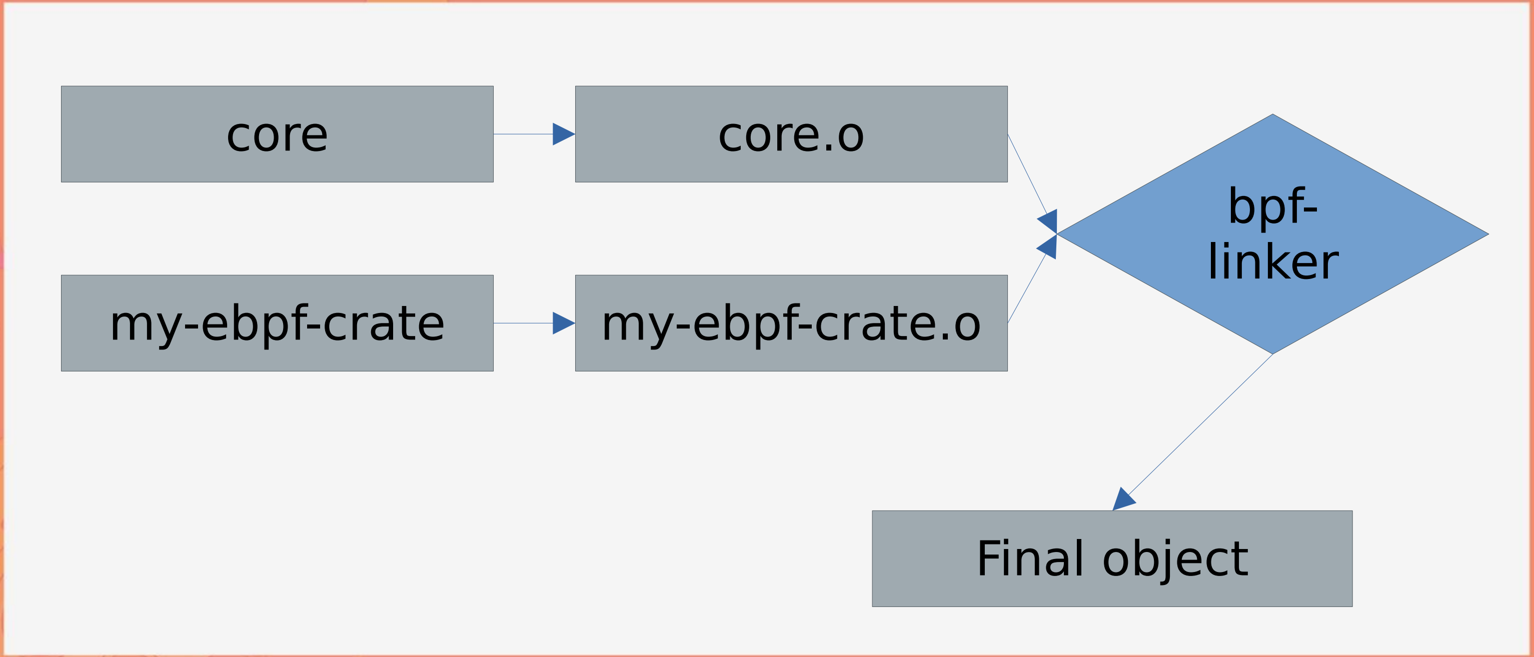


But...

What is bpf-linker?

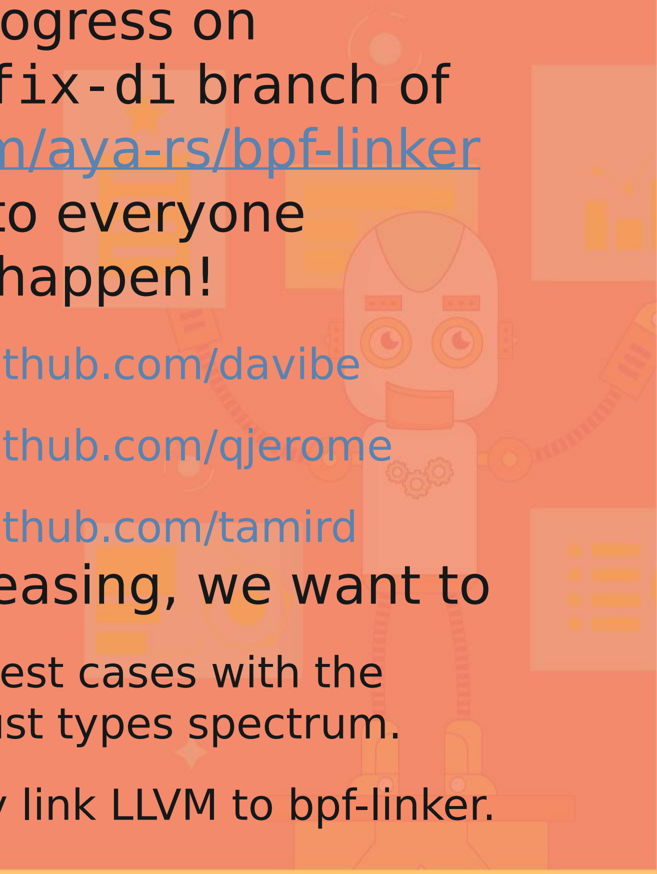
- It’s a bitcode linker.
- Linking BPF in C is optional and mostly not done.
- **Linking in Rust in mandatory, everything is a crate.**
- Traditional linkers (e.g. lld) don’t work for BPF.

- **How bpf-linker works?** Linking crates as bitcodes



A sneaky solution

Sanitizing LLVM DI



- Work in progress on `feature/fix-di` branch of github.com/aya-rs/bpf-linker
- Shoutout to everyone making it happen!
  - https://github.com/davibe
  - https://github.com/qjerome
  - https://github.com/tamird
- Before releasing, we want to
  - Provide test cases with the whole Rust types spectrum.
  - Statically link LLVM to bpf-linker.

Adding the BTF relocation support in

Rust compiler

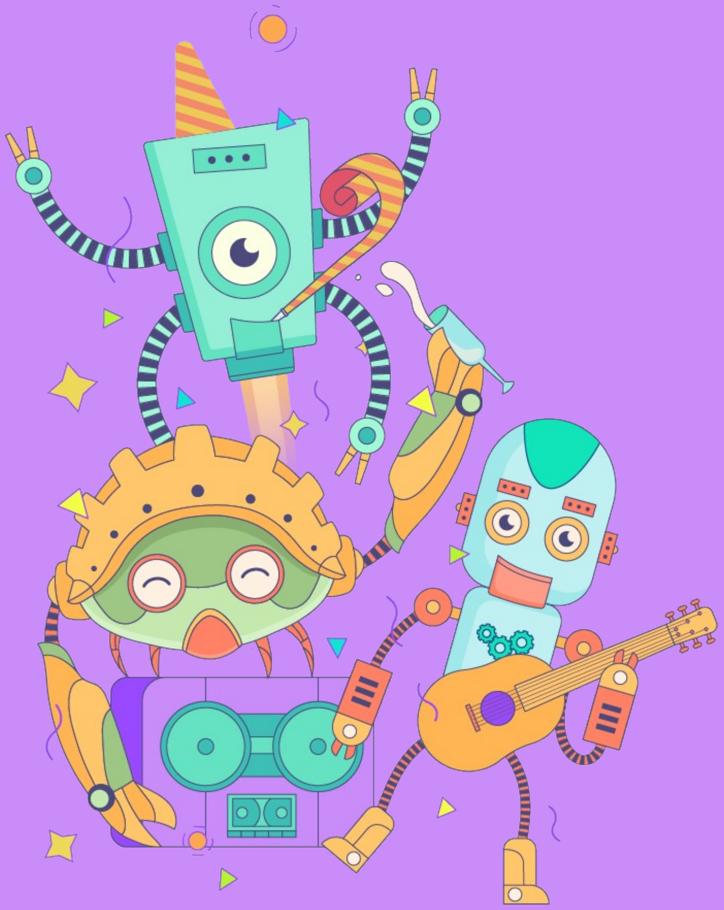
- preserve_access_index is an LLVM IR intrinsic.
- Works similar to GEP (`getElementPtr`) instruction.
- We need to add it to **core::intrinsics** and **rustc_codegen_llvm**.

Long-term, correct solution

Teaching the kernel to accept non-C types

- Probably would make sense to do when introducing BTF to Rust-for-Linux (**kernel moules need BTF for their types**).
- Would be nice to teach Rust to emit only BTF (other way than debug=2).
- Downside: will take years to be adopted.

Thank you for listening!



MICHAL ROSTECKI

michal@exein.io