BUILDING YOUR OWN BINARY FORMAT

Arpad Borsos / Swatinem

https://swatinem.de/blog/binary-formats/

y tho?

just use existing formats?

optimized for raw speed

`O(1)` parsing / loading

random access, aka binary search in `O(log N)`

BUT: not compact trades space efficient for read speed / simplicity

random access?

zero-allocation?

zero-copy?

JSON is parsed all at once, and allocates

```
pub enum serde_json::Value {
    Null,
    Bool(bool),
    Number(Number),
    String(String),
    Array(Vec<Value>),
    Object(Map<String, Value>),
}
```

There is a lifetime though?

```
pub fn serde_json::from_str<'a, T>(s: &'a str) -> Result<T>
    where T: Deserialize<'a>
```

Best you can do:

```
Cow<'a, str>
```

Copies and allocates:

```
struct Format {
    some: u64,
    other: u32,
    value: u32,
    children: Vec<u32>,
}

fn parse(bytes: &[u8]) -> Result<Format> {}
```

Returns owned data, still copies:

```
impl Iterator for ChildIter {
    type Item = Result<FormatChild>
}
```

We want this:

```
struct Format<'data> {
    header: &'data FormatHeader,
    children: &'data [FormatChild],
}

impl Format {
    pub fn parse(data: &[u8]) -> Result<Format<'_>> {}
}
```

loading just checks headers and does bounds checks

only data that is needed is being copied / validated

random access: fixed size structs

- `serde` / `bincode`, etc
- `scroll`, custom derive, parses every item
- `rkyv`, supports complex data formats
- `zerocopy`, custom derive, verifies layout
- `watto`, our very own


The fastest POD racer ever built

```rust
#[repr(C)]
struct Header {
    version: u32,
    num_as: u32,
}
unsafe impl watto::Pod for Header {}

#[repr(C)]
struct A(u16);
unsafe impl watto::Pod for A {}
```

```rust
// buffer: &'buffer [u8]
let (header, buffer): (&'buffer Header, &[u8]) =
    Header::ref_from_prefix(buffer).unwrap();
let (_, buffer) =
    watto::align_to(buffer, mem::align_of::<A>()).unwrap();

let num_as = header.num_as as usize;
let (r#as, buffer): (&'buffer [A], &[u8]) =
    A::slice_from_prefix(buffer, num_as).unwrap();
```

alignment?

endianness?

SIGSEGV: unaligned memory access

```rust
u32::from_le_bytes([1, 0, 0, 0]) // 1
u32::from_be_bytes([1, 0, 0, 0]) // 16_777_216

// Pretty much everything is LE

let ptr = &[1u8, 0, 0, 0] as *const u8;
unsafe { *(ptr as *const u16) } // 1
unsafe { *(ptr as *const u32) } // 1
```

What next?

Your format should have:

- a format identifier (magic bytes)
- a version?
- how about compatibility

Compatibility?

backwards: old writer, new reader

forwards: new writer, old reader

backwards compatibility:

your format needs a `version`

forwards compatibility:

- give collections / structs a `size_of` marker
- add new fields, never remove or reuse
- can't use `&'data [Struct]` directly anymore

Go and build!

https://swatinem.de/blog/binary-formats/

Bonus slide:

Be mindful of `slice[index]` syntax, use `slice.get(index)` instead.