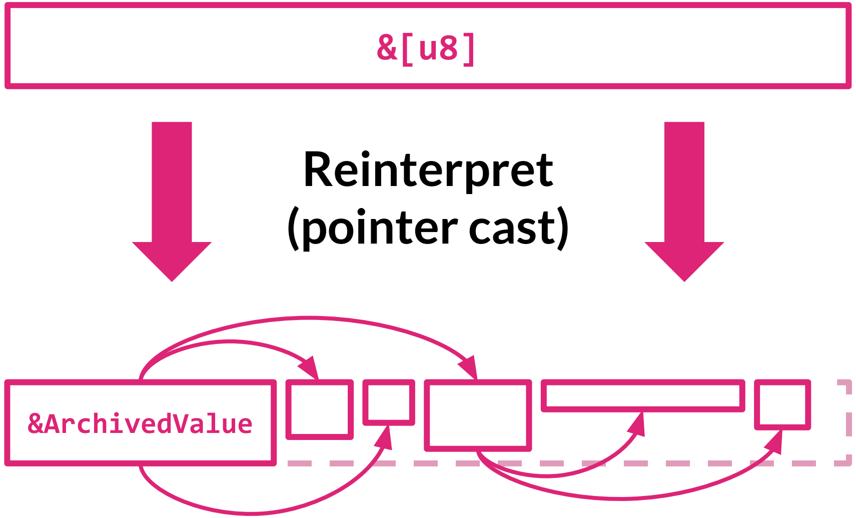
The (many) mistakes I made in rkyv

David Koloski
github.com/djkoloski

× RUSTCONF 2024

Background

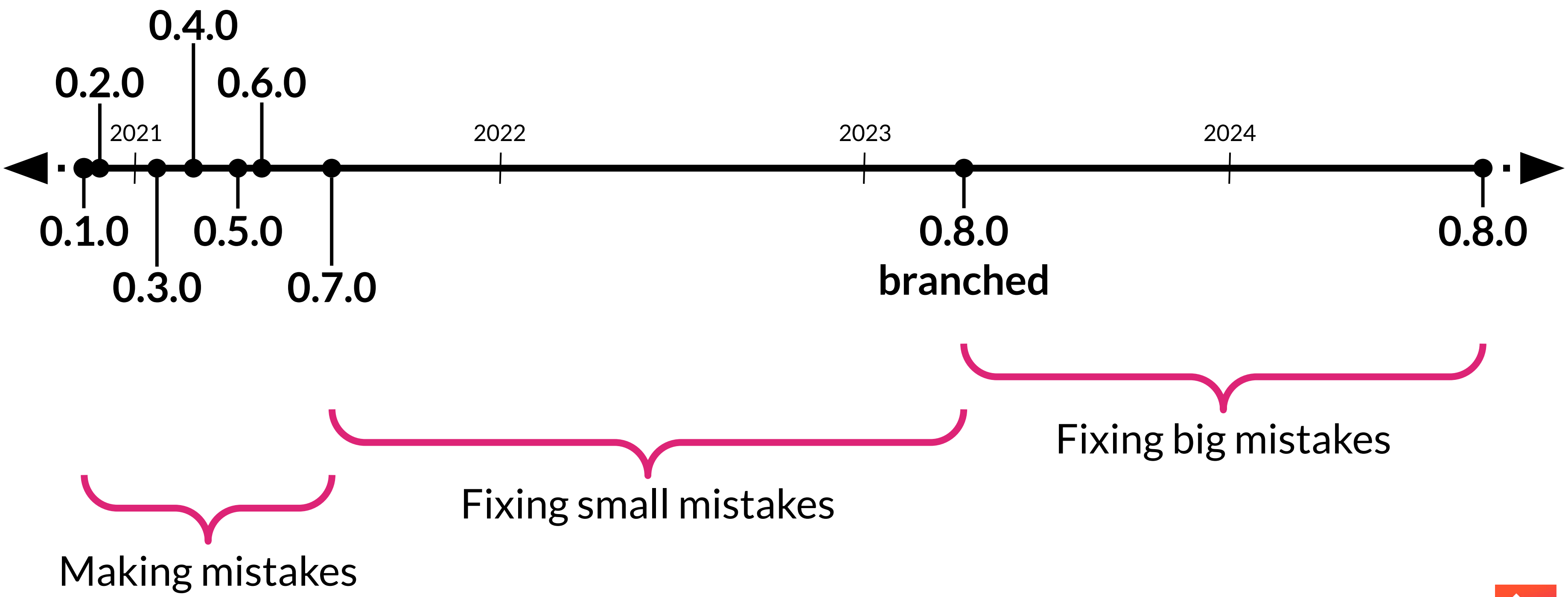
- rkyv is a zero-copy deserialization (ZCD) framework for Rust
- Use serialized data in-place without a deserialization step
  - `&[u8]` → `&Foo`
- Other serialization frameworks support partial ZCD
  - Some parsing required
  - Borrow *some* data from bytes, usually strings or byte slices
- rkyv supports full ZCD
  - No parsing required
  - Borrow *all* data from bytes



0.8 is out now!

3.8 years of development

(to scale)



The mistakes

Soundness

Unstable layouts

Failing MIRI

Writing uninit bytes

Abusing `Pin`

Malicious validation

...

Ergonomics

The Most Unhelpful Error

Terrible API names

Bad error handling

Incomplete no-std APIs

Overly-unsafe `unsafe`

...

Project health

Unhealthy optimism

Public scrutiny

Keeping contributors

Burning out

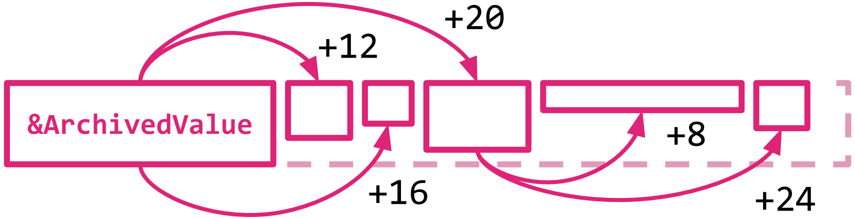
CI death spirals

...

Unstable layouts

Background

- rkyv reinterprets a byte slice as special “archived” types
- Regular Rust types can’t just be reinterpreted from bytes
  - Pointers like *const and *mut are “absolute” - they care about the address of memory
- No problem - we’ll replace absolute pointers with relative ones
  - Relative pointers specify how many bytes forwards or backwards instead



Regular	Archived
*const _	RelPtr<_>
Box<_>	ArchivedBox<_>
Vec<_>	ArchivedVec<_>
HashMap<_, _>	ArchivedHashMap<_, _>

Mistakes

- Yes problem
- No portable primitives
  - Serialized data always used the target machine’s endianness
- No `#[repr(C)]`
  - The compiler was allowed to reorder struct fields
- 0.7 tried to fix this
  - Opt-in `strict` feature for adding `#[repr(C)]` to your structs
  - Opt-in wrapped primitives with explicit endianness

```
// You write
#[derive(rkyv::Archive)]
struct Foo {
    a: Box<u32>,
}

// Derive generates
struct ArchivedFoo {
    a: ArchivedBox<u32>,
}
```

No explicit representation

Native primitives

The wrong fix

- That was not enough
  - The alignments of multi-byte primitives may differ across targets
  - Just handling endianness is not enough
- These incorrect generated types are part of rkyv's public API
  - We cannot fix these issues without breaking existing users
- Got lucky: all Tier 1 targets agree on primitive type layout
  - ARM disagrees on alignment for 128-bit integers (Tier 2)

	`x64`	`aarch64`	`i686`	`arm`
`i8,u8`	`1`	`1`	`1`	`1`
`i16,u16`	`2`	`2`	`2`	`2`
`i32,u32`	`4`	`4`	`4`	`4`
`i64,u64`	`8`	`8`	`8`	`8`
`i128,u128`	`16`	`16`	`16`	`8`
`bool`	`1`	`1`	`1`	`1`
`char`	`4`	`4`	`4`	`4`
`f32`	`4`	`4`	`4`	`4`
`f64`	`8`	`8`	`8`	`8`

Alignments of primitive types by target arch

The right fix

- 0.8 finally fixed these mistakes
  - `#[repr(C)]` is always required
  - Endianness is always well-defined
  - Primitive types are always wrapped
- Replacing all of the primitive types was a lot of work
  - Ergonomics were severely impacted
  - Added a lot of line noise
- Upside: made it really easy to add a new feature

Well-defined layout

```
#[repr(C, align(4))]
pub struct u32_le(u32);

impl u32_le {
    #[inline]
    pub const fn from_native(value: u32) -> Self {
        // On little-endian targets
        Self(value)
    }

    #[inline]
    pub const fn to_native(self) -> u32 {
        // On little-endian targets
        self.0
    }
}
```

Explicit endianness

The Most Unhelpful Error

What is The Most Unhelpful Error?

```
type annotations needed for `With<T, W>`
multiple `impl`s satisfying `_: DeserializeWith<ArchivedMessage, _, Strategy<(), Failure>>` found in the `rkyv` crate:
- impl<A, T, D> DeserializeWith<A, Arc<T>, D> for rkyv::with::Cloned
  where A: Deserialize<T, D>, D: Fallible, D: ?Sized;
- impl<A, T, D> DeserializeWith<A, Rc<T>, D> for rkyv::with::Cloned
  where A: Deserialize<T, D>, D: Fallible, D: ?Sized;
required for `ArchivedMessage` to implement `Deserialize<With<_, _>, Strategy<(), Failure>>`
```

What’s wrong with that? Looks pretty helpful to me!

The error should be:

```
the trait bound `(): Pooling<Failure>` is not satisfied
the following other types implement trait `Pooling<E>`:
  Strategy<T, E>
  rkyv::de::Unify
  rkyv::de::Duplicate
required for `Strategy<(), Failure>` to implement `Pooling<Failure>`
required for `ArchivedRc<ArchivedString, ArcFlavor>` to implement `Deserialize<Arc<String>, Strategy<(), Failure>>`
required for `ArchivedMessage` to implement `Deserialize<Message, Strategy<(), Failure>>`
```

How did we get here?

```
let state = archived.deserialize(..).unwrap();
```

```
type annotations needed for `With<T, W>`
```

- That’s not supposed to be a `With<T, W>`
- Try fully-qualified syntax?

```
let state = <
    ArchivedMessage as Deserialize<Message, Strategy<(), Failure>>
>::deserialize(archived, ..).unwrap();
```

```
the trait bound `(): Pooling<Failure>` is not satisfied
```

Trait selection

- Fully-qualified syntax removes all ambiguity in trait selection
- So this must be a trait selection problem
- What impls are in scope?
  - The one we want to call is definitely in scope
  - … but one of its clauses isn't satisfied
- We want the compiler to stop there and tell us the clause isn't satisfied

```rust
#[derive(Archive, Serialize, Deserialize)]
pub enum Message {
    NotReady,
    Ready(Arc<String>),
}
```

Perfectly good impl… right?

```rust
impl<D> Deserialize<Message, D> for ArchivedMessage
where
    ArchivedArc<ArchivedString>: Deserialize<Arc<String, D>>,
{ .. }
```

This clause isn't satisfied

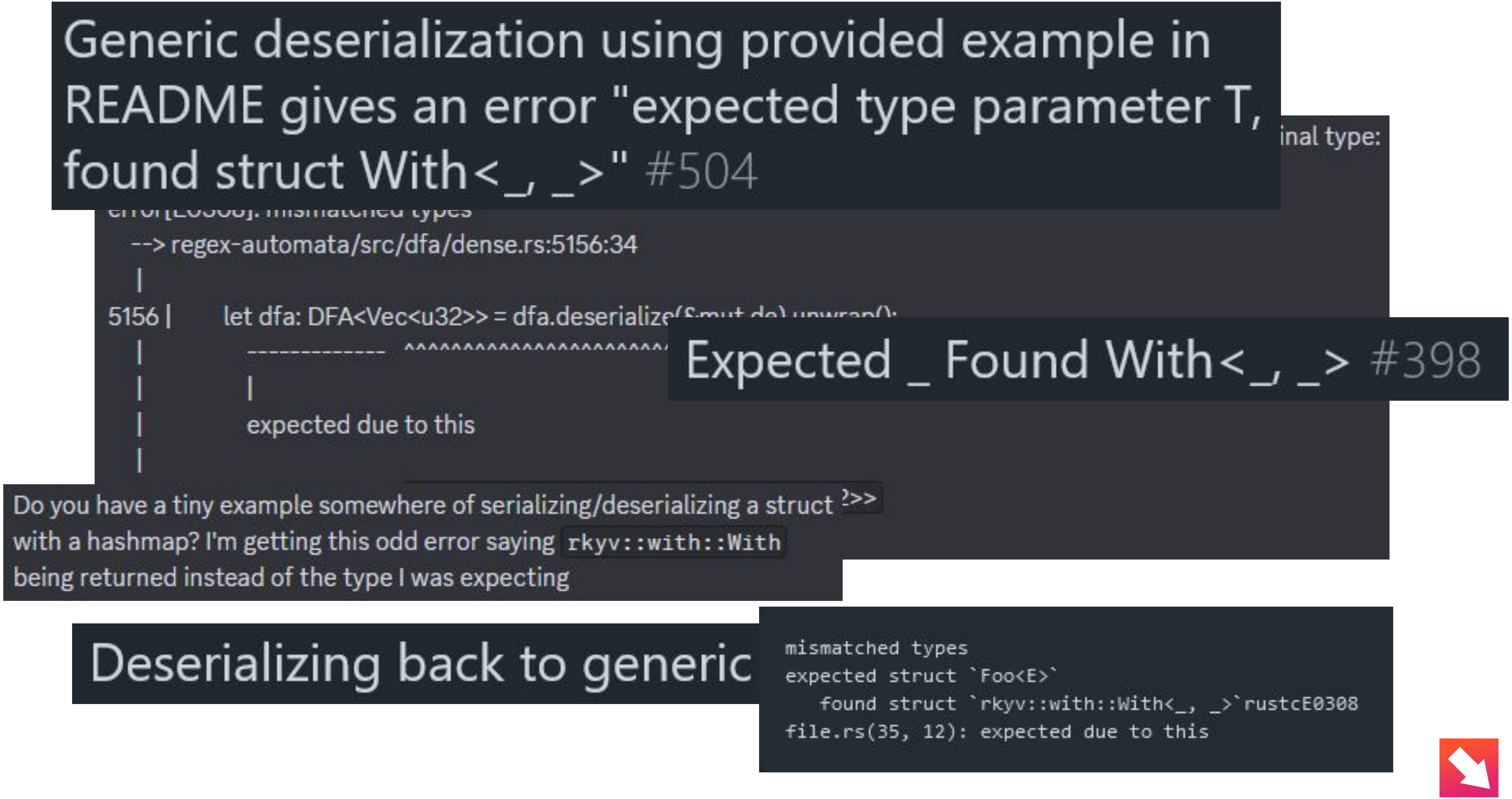
A blanket impl trap

- Exactly one other impl in scope
- The compiler is certain it's not the first impl, so it must be this one
  - It just needs to deduce the correct types of `T` and `W`... right?
  - There's not enough type information for it to figure them out
  - The compiler doesn't know it's impossible pick a correct `T` and `W`
- This leads the compiler astray
  - We want it to emit a diagnostic about the impl that we definitely didn't match

This impl is completely unrelated to what we're trying to do

```
impl<F, W, T, D> Deserialize<With<T, W>, D> for F
where
    W: DeserializeWith<F, T, D>,
    D: Fallible + ?Sized,
{ .. }
```

But this could match any type!



Generic deserialization using provided example in README gives an error "expected type parameter T, found struct With<_, _>" #504

```
inal type:
error[E0308]: mismatched types
  --> regex-automata/src/dfa/dense.rs:5156:34
   |
5156 |       let dfa: DFA<Vec<u32>> = dfa.deserialize(&mut de).unwrap();
   |           ------------   ^^^^^^^^^^^^^^^^^^^^^^^^
   |           |
   |           expected due to this
   |
                                                  >>
```

Expected _ Found With<_, _> #398

Do you have a tiny example somewhere of serializing/deserializing a struct with a hashmap? I'm getting this odd error saying `rkyv::with::With` being returned instead of the type I was expecting

Deserializing back to generic

```
mismatched types
expected struct `Foo<E>`
   found struct `rkyv::with::With<_, _>`rustcE0308
file.rs(35, 12): expected due to this
```

Bad errors are worth a code change

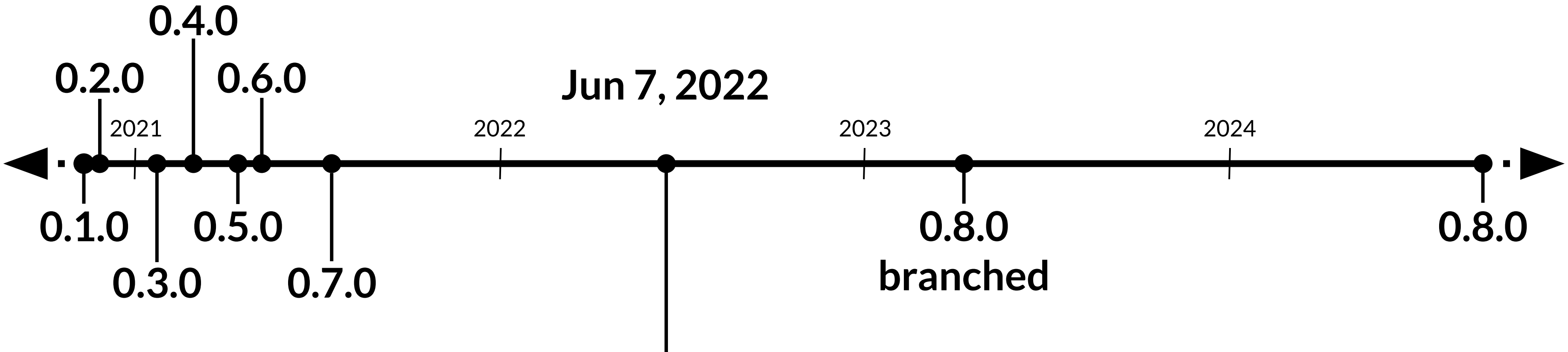
- I hoped the compiler would change their error reporting to fix it
- I've since come to accept that the compiler can't save me from myself
- As of rkyv 0.8, that blanket impl trap is gone
  - The offending type has been removed
  - Replaced by derive macro logic
- The compiler gives a much better error now

```
impl<D> Deserialize<Message, D> for ArchivedMessage
where
    ArchivedArc<ArchivedString>: Deserialize<Arc<String, D>>,
{ .. }
```

Now rustc tells you why this clause isn't satisfied

Unhealthy Optimism

Development timeline



0.4.0

0.2.0

0.6.0

Jun 7, 2022

2021

2022

2023

2024

0.1.0

0.5.0

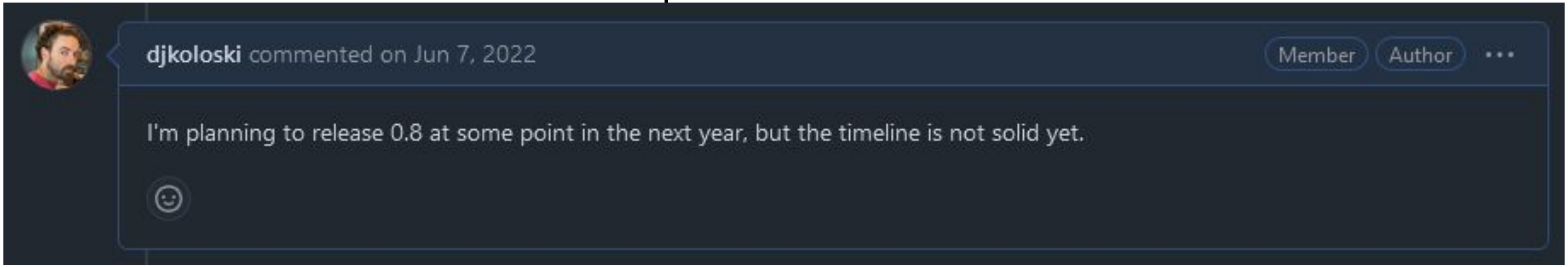
0.8.0

0.8.0

0.3.0

0.7.0

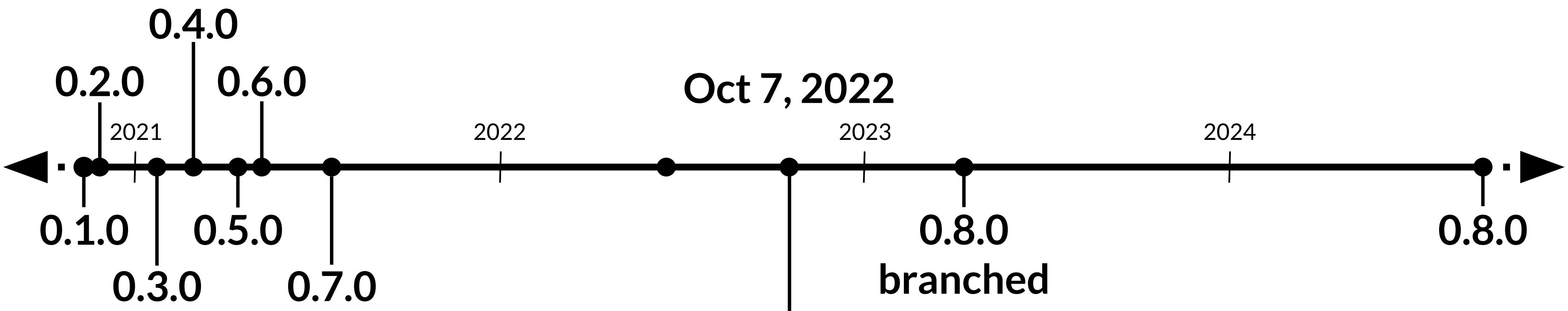
branched



djkoloski commented on Jun 7, 2022

I'm planning to release 0.8 at some point in the next year, but the timeline is not solid yet.

Development timeline



0.4.0

0.2.0

0.6.0

Oct 7, 2022

2021

2022

2023

2024

0.1.0

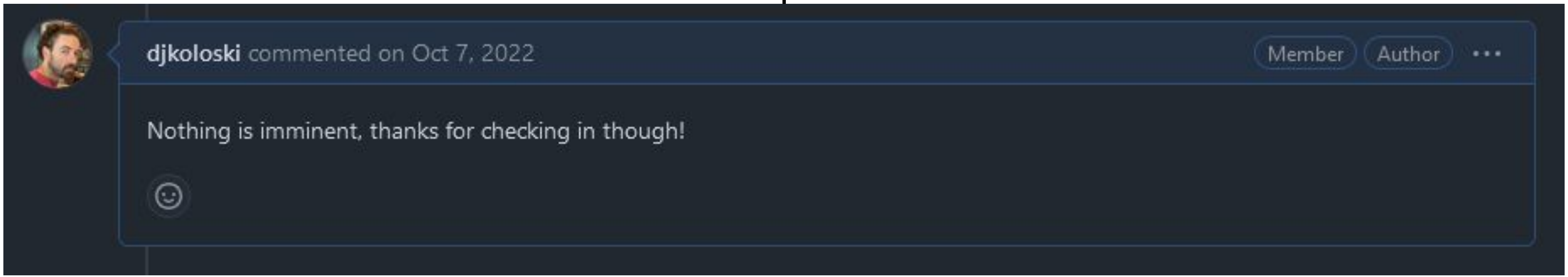
0.5.0

0.8.0
branched

0.8.0

0.3.0

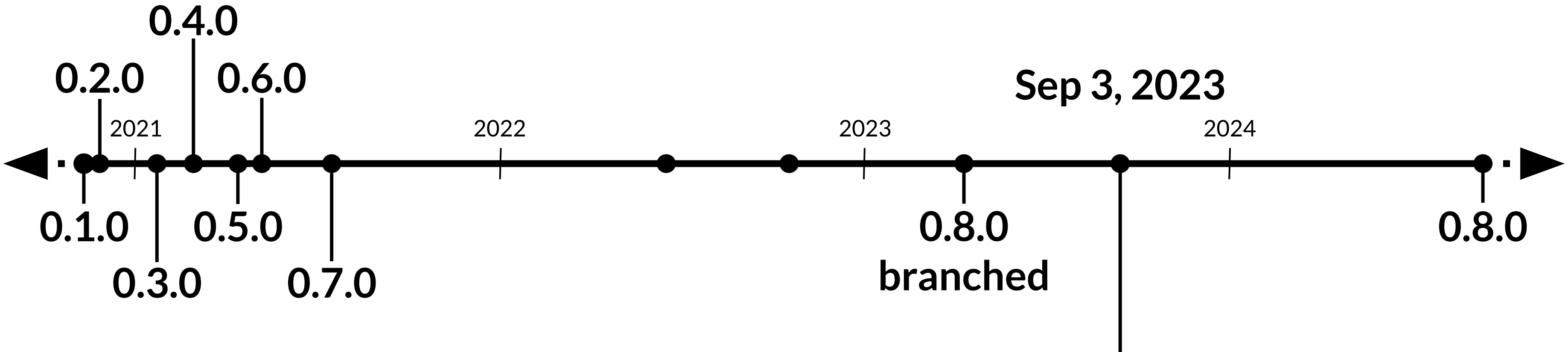
0.7.0

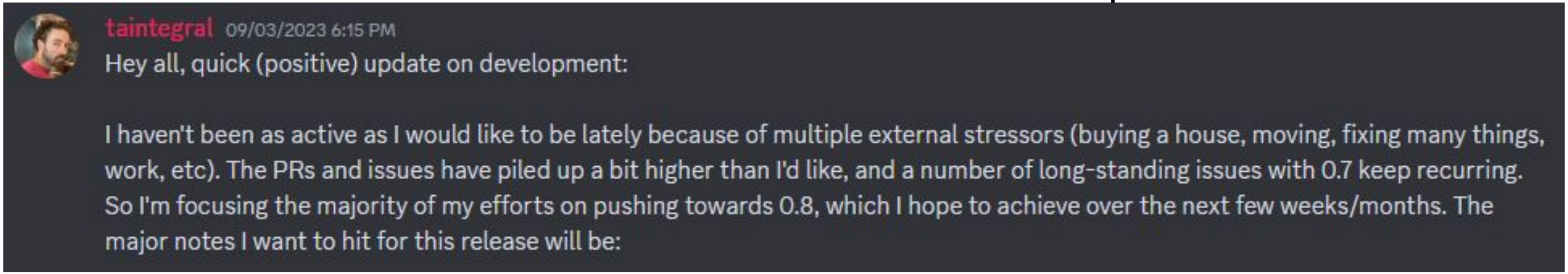


djkoloski commented on Oct 7, 2022

Nothing is imminent, thanks for checking in though!

Development timeline

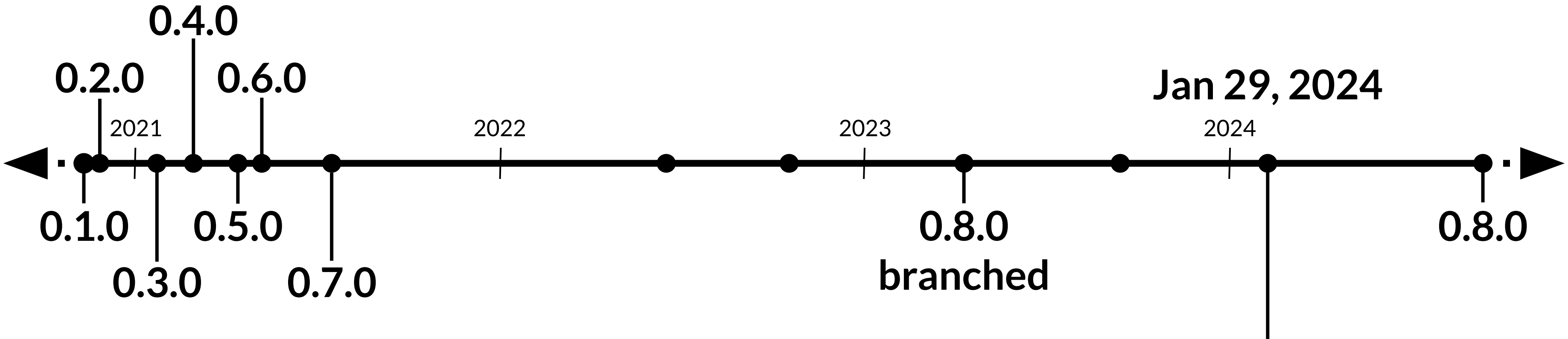



taintegral 09/03/2023 6:15 PM

Hey all, quick (positive) update on development:

I haven't been as active as I would like to be lately because of multiple external stressors (buying a house, moving, fixing many things, work, etc). The PRs and issues have piled up a bit higher than I'd like, and a number of long-standing issues with 0.7 keep recurring. So I'm focusing the majority of my efforts on pushing towards 0.8, which I hope to achieve over the next few weeks/months. The major notes I want to hit for this release will be:

Development timeline



0.4.0

0.2.0

0.6.0

Jan 29, 2024

2021

2022

2023

2024

0.1.0

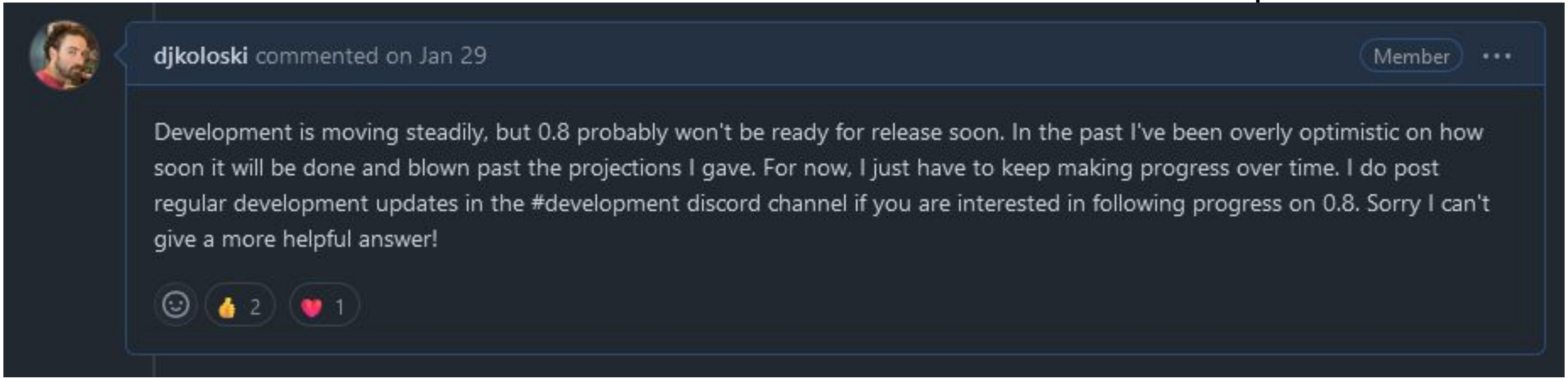
0.5.0

0.8.0
branched

0.8.0

0.3.0

0.7.0



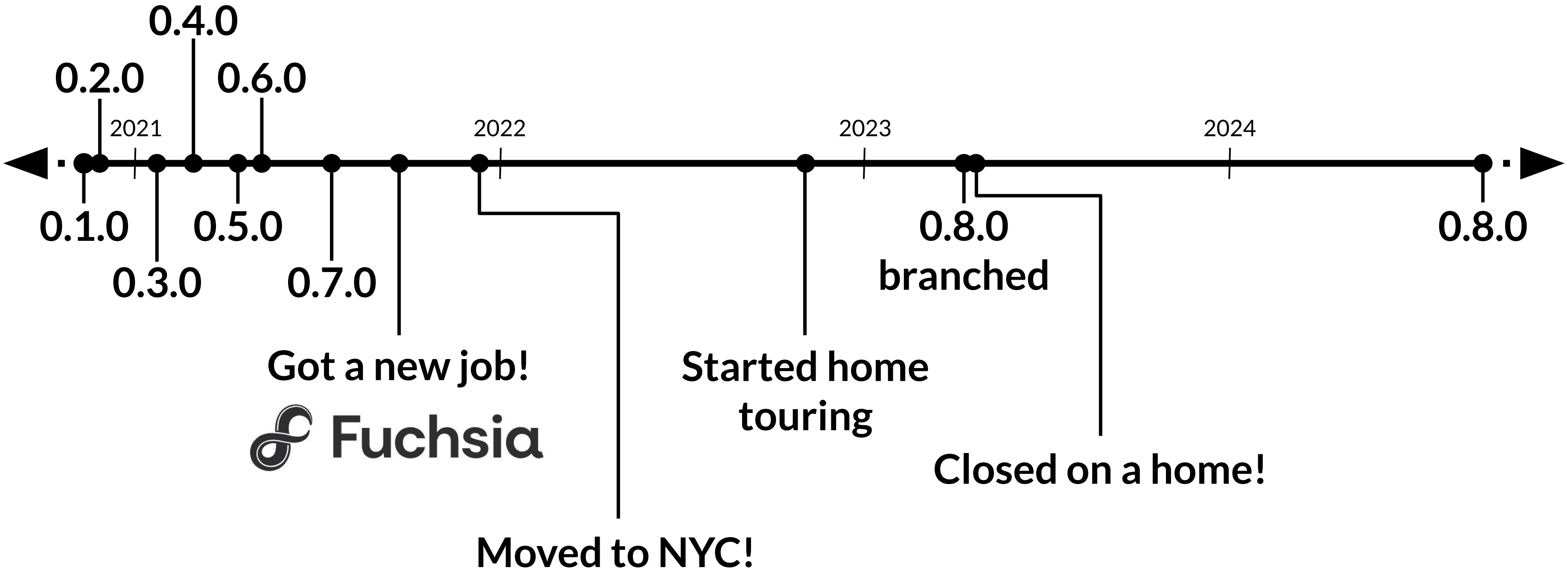
djkoloski commented on Jan 29

Development is moving steadily, but 0.8 probably won't be ready for release soon. In the past I've been overly optimistic on how soon it will be done and blown past the projections I gave. For now, I just have to keep making progress over time. I do post regular development updates in the #development discord channel if you are interested in following progress on 0.8. Sorry I can't give a more helpful answer!

Giving estimates was a mistake

- I had no idea how heavily those estimates and projections would weigh on me
- Prior to 0.7, I’d been shipping new versions every few weeks to months
- Now it seemed like there was an impossible amount of work to get done
- I often felt like I didn’t have room to experiment as much as I wanted to
  - And when I did, I felt like I was wasting time
- I love my project and want it to succeed
- People were depending on me!
- I didn’t know how much time it would take to fix my mistakes
- So I stopped giving estimates
  - I’m working hard, it will be ready when it’s ready

Development timeline



rkyv was still there

Thank you!