perplexity

deep explain

trait MyTrait where Self: Sized

The error "the size for values of type `Self` cannot be known at compilation time" arises because Rust needs to know the size of every type at compile time to properly manage memory. However, traits introduce a level of indirection where the concrete type implementing the trait isn't known in advance. This becomes a problem when the trait method returns `Self` by value, because the compiler needs to know how much space to allocate for the return value[1].

Here's a breakdown of the concepts:

1. `Sized` **Trait**: The `Sized` trait in Rust indicates that the size of a type is known at compile time[2]. Most types in Rust are `Sized`, including primitive types like `i32`, structs, and enums with fixed sizes[2].
2. **Implicit** `Sized` **Bound**: In generic functions, structs, and traits, type parameters have an implicit `Sized` bound[3]. This means that if you write `fn foo<T>(...)`, Rust automatically assumes that `T: Sized`[3].
3. `Self` **in Traits**: Within a trait definition, `Self` refers to the type that is implementing the trait[3]. Unlike other type parameters, `Self` does **not** have an implicit `Sized` bound[2] [3]. This is because traits are designed to support trait objects, which are dynamically sized types[2].
4. **Trait Objects**: Trait objects (e.g., `&dyn MyTrait` or `Box<dyn MyTrait>`) allow you to work with values of different types that all implement the same trait[2]. The size of a trait object isn't known at compile time because it could be any type that implements the trait[2].
5. **The Problem**: When a trait method returns `Self` by value, the compiler needs to know the size of `Self` to allocate space for the return value[1]. If `Self` could be a trait object (i.e., unsized), the compiler wouldn't know how much space to allocate, hence the error[1].
6. **The Solution**: Adding a `where Self: Sized` bound to the trait definition tells the compiler that this trait can only be implemented for types that *are* `Sized`[3] [4]. This allows the compiler to know the size of `Self` at compile time, resolving the error[3] [4]. However, this also means that you can no longer use this trait with trait objects[2].

In summary, the `where Self: Sized` bound is necessary when a trait method returns `Self` by value to ensure that the compiler knows the size of the implementing type at compile time. If you want to allow trait objects, you'll need to use techniques like returning a boxed trait object (`Box<dyn MyTrait>`) or using references (`&Self`) instead[1].

⁂

1. https://users.rust-lang.org/t/explanation-on-fn-self-box-self-for-trait-objects/34024

2. https://doc.rust-lang.org/std/marker/trait.Sized.html
3. https://stackoverflow.com/questions/30938499/why-is-the-sized-bound-necessary-in-this-trait
4. https://github.com/pretzelhammer/rust-blog/blob/master/posts/sizedness-in-rust.md