perplexity

future cannot be sent between threads safely

the trait `Send` is not implemented for `dyn std::error::Error`
required for the cast from `Pin<Box<

why

The error message you are encountering:

```
future cannot be sent between threads safely
the trait `Send` is not implemented for `dyn std::error::Error`
required for the cast from `Pin&lt;Box&lt;...&gt;&gt;`
```

occurs because Rust's async executors (like Tokio) often require futures to be `Send` so they can be safely moved across threads. When you have a future that returns a `Box<dyn std::error::Error>`, the trait object `dyn std::error::Error` by itself does **not** implement `Send` or `Sync` traits. This means the future that contains it is not `Send`, and thus cannot be safely sent between threads.

Why does this happen?

- `dyn std::error::Error` is a trait object that represents any error type.
- By default, `dyn std::error::Error` does **not** imply `Send` or `Sync`.
- If your async function returns a `Result<T, Box<dyn std::error::Error>>`, the error inside the box might not be safe to send across threads.
- Tokio and other async runtimes require futures to be `Send` to allow them to be executed on thread pools.

How to fix it?

You need to add `+ Send + Sync + 'static` bounds to the error trait object to ensure the error is safe to send across threads:

```
async fn some_async_fn() -&gt; Result&lt;T, Box&lt;dyn std::error::Error + Send + Sync +
    // ...
}
```

This means:

- `Send`: The error can be transferred across thread boundaries.
- `Sync`: The error can be referenced from multiple threads safely.
- `'static`: The error does not contain non-static references, so it lives long enough.

Additional considerations

- When you add `Send + Sync` to the error type, all errors you return must also implement these traits.
- This might require changing the error types in your code or wrapping them appropriately.
- You might need to propagate these trait bounds throughout your async call chain to satisfy the compiler.
- Sometimes, you encounter errors because some local variables or guards (like `MutexGuard`) are not `Send` and are held across `.await` points, which also prevents the future from being `Send`.

Example from community experience

- Adding `+ Send + Sync` to the error trait object fixed the issue in Tokio async code, but required updating all error returns to match the new trait bounds[1].
- Holding non-`Send` values (like `MutexGuard`) across `.await` causes similar errors because the future is not `Send`[2].
- You cannot manually implement `Send` for trait objects; the compiler enforces this based on the underlying types[3].
- If you have `Box<dyn Error>`, convert it to `Box<dyn Error + Send + Sync>` to satisfy the thread safety requirements[4] [1].

Summary

The root cause is that `dyn std::error::Error` by itself is not guaranteed to be thread-safe (`Send` and `Sync`). Async runtimes require futures to be `Send` to run them on thread pools. To fix this, you must:

- Use `Box<dyn std::error::Error + Send + Sync + 'static>` for error types in async functions.
- Ensure all errors you return implement `Send` and `Sync`.
- Avoid holding non-`Send` types (like `MutexGuard`) across `.await` points.

This will resolve the "future cannot be sent between threads safely" error related to `dyn std::error::Error` trait objects in futures[4] [5] [3] [1] [2] [6].

⁂

1. https://nathanleclaire.com/blog/2021/11/06/tokio/rust-dyn-stderrorerror-cannot-be-sent-between-threads-safely/
2. https://stackoverflow.com/questions/68976937/rust-future-cannot-be-sent-between-threads-safely
3. https://www.reddit.com/r/learnrust/comments/ului03/how_to_implement_send_for_vecpinboxdyn_future/
4. https://users.rust-lang.org/t/convert-box-dyn-error-to-box-dyn-error-send/48856
5. https://github.com/spacejam/sled/issues/1148

6. https://users.rust-lang.org/t/get-error-source-of-sync-send-and-make-it-sync-send/67970