What Rust checks for you

In PHP

PHP reads the file when the request arrives (or when you run `php script.php`) and interprets it. A function that asks for `int` will, by default, coerce a numeric string. That check happens as the line runs. It is not a compile step. Only `declare(strict_types=1);` at the top of that file turns the call into a `TypeError`, and even then the error is still raised at runtime, for that file only.

Memory for ordinary values is freed when the last reference goes away. `__destruct` runs then, for data that is not stuck in a cycle. It is not "the garbage collector will get to it later."

add.php

```
<?php

function add(int $a, int $b): int
{
    return $a + $b;
}

echo add('5', 3), "\n";
```

Run on PHP 8.3, that prints `8`. The string `'5'` was coerced. Add `declare(strict_types=1);` as the first statement and the same call throws `TypeError`. The file still has to run for that to happen.

In Rust

You compile, then you run the binary. A type mismatch never becomes a coercion. This program is the whole file. It compiles, and it prints `8`.

src/main.rs

```
fn add(a: i32, b: i32) -> i32 {
    a + b
```

```
}

fn main() {
    let sum = add(5, 3);
    println!("{sum}");
}
```

`add("5", 3)` does not compile (`E0308` , mismatched types). There is no binary until the argument is an `i32` .

The same "fail at compile time" rule covers borrowing, with one detail that is easy to get wrong. An immutable borrow ends at its last use, not at the closing brace. This program compiles. `r1` is printed, then the mutable borrow starts, then `r2` is used. Non-lexical lifetimes are why the first borrow does not last until the end of `main` .

src/main.rs · compiles

```
fn main() {
    let mut v = vec![1, 2, 3];
    let r1 = &v;
    println!("{r1:?}");
    let r2 = &mut v;
    r2.push(4);
    println!("{r2:?}");
}
```

Use `r1` again after the mutable borrow and the compiler stops you. The error is `E0502` .

src/main.rs · must fail

```
fn main() {
    let mut v = vec![1, 2, 3];
    let r1 = &v;
    let r2 = &mut v;
    println!("{r1:?}");
    let _ = r2;
}
```

`println!` of `r1` is a later use of the immutable borrow, so `r2` is rejected. Ownership and the full set of borrow errors are a later chapter. For now, the rule is: the compiler is looking at uses, not at the line where you wrote `let` .

> **Watch for:** Reading a borrow error, look for the line where the first borrow is used again. That is the line the compiler is protecting.

Idiomatic Rust: `{sum}` inside the format string, not `"{}", sum`.

Side by side

	PHP	Rust
When types are checked	When that line runs	When you compile
`add("5", 3)` with an `int` parameter	`8`, unless that file has `strict_types`	Does not compile
When unused memory goes	When the last reference drops	When the owner ends, decided at compile time

Try it

Open the working `add` program in the Playground (the button uses that file). Change `add(5, 3)` to `add(5, "3")` and read the error (`E0308`, mismatched types). Put it back, then paste the must-fail sample and confirm the code is `E0502`.

Key takeaways

- PHP coerces `"5"` to `8` unless that file opts into `strict_types`. Rust refuses the call at compile time.
- PHP frees acyclic values when the last reference drops. It does not wait for a vague GC pause.
- A conflicting borrow is `E0502` only while the first borrow is still used.

What's next: Installing Rust