Rust for Java Programmers

A personal experience

- what did Java promise?
- what does Rust promise?
- how is Rust different?
- why is Java used in business?
- moving from Java to Rust

Future predictions (from the past)

1995

- The future is Object Oriented.

2000

- The future is XML.

Not so long ago

- You can't grow without making everything social.
- Gamify, gamify, gamify.
- Block chain will change the universe.
- You need an AMP website.
- Your stuff is not competitive without Machine Learning.

Prophet

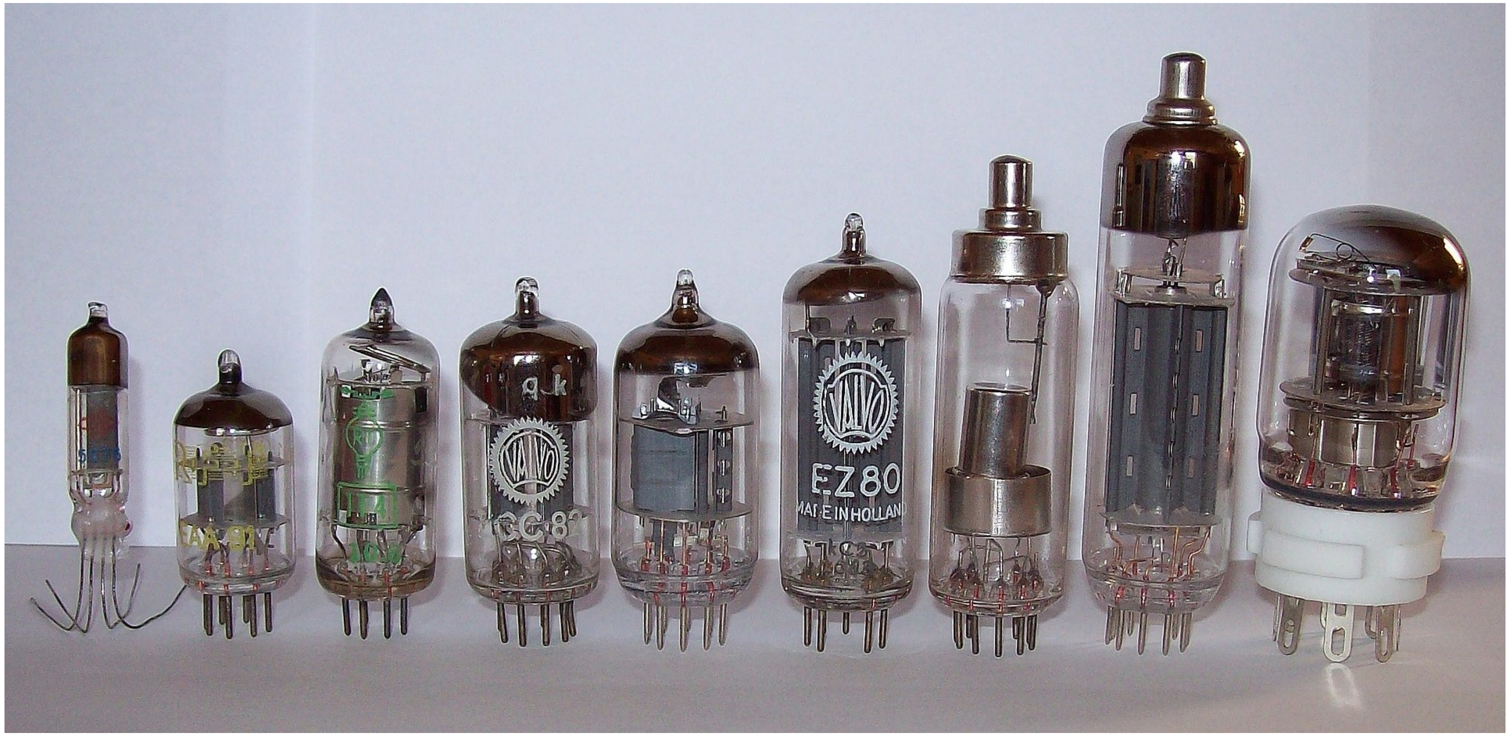
In *A short history of decay* Cioran wrote:

In every man sleeps a prophet, and when he wakes there is a little more evil in the world.

I am also human but in here, I will try to be an anti-prophet: the future very often is hidden in the past.

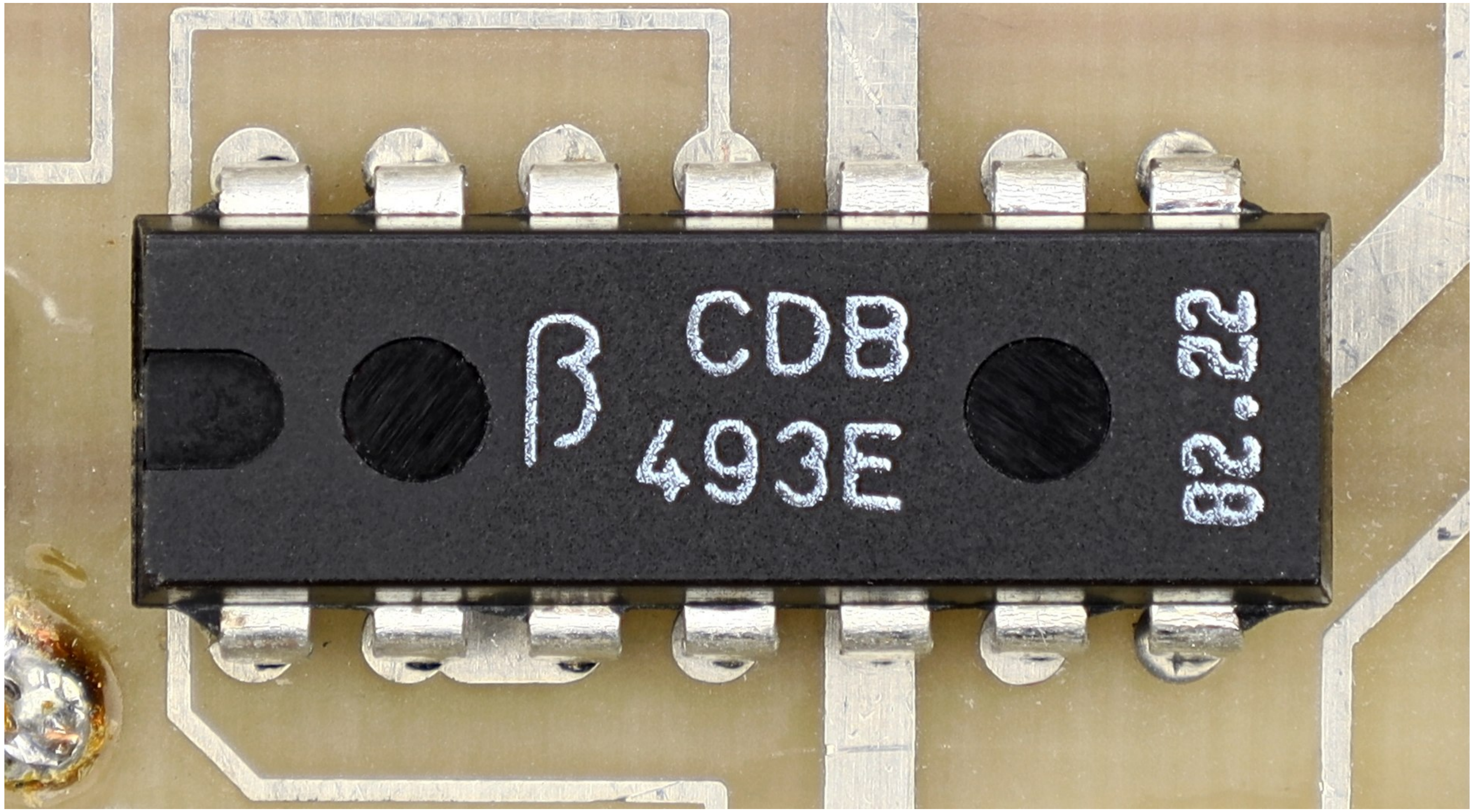
Let's have a quick look at the evolution of computer hardware.

Electronic Computers - Vaccuum Tubes

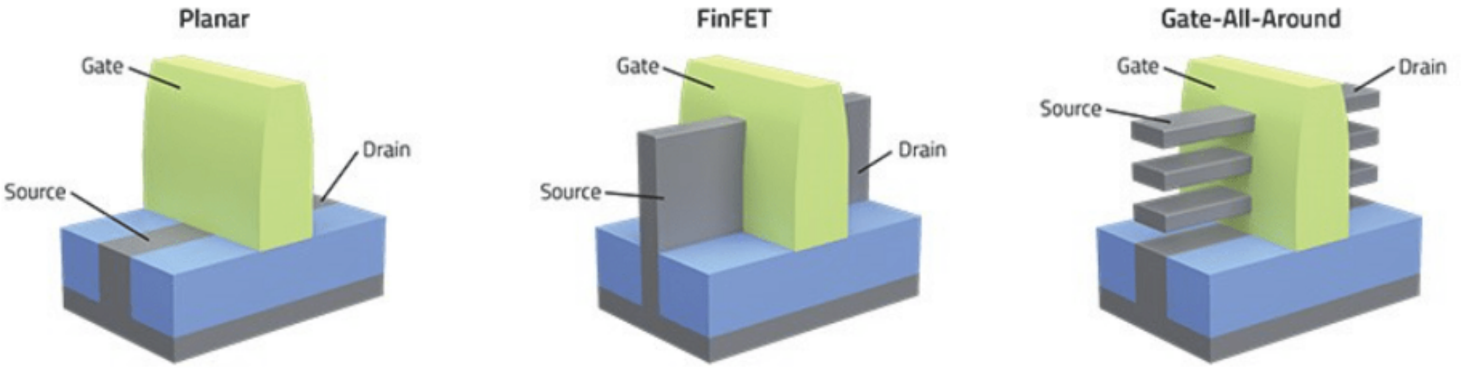


The keen observer will notice the precursor of the Rust logo on the fourth and sixth tube.

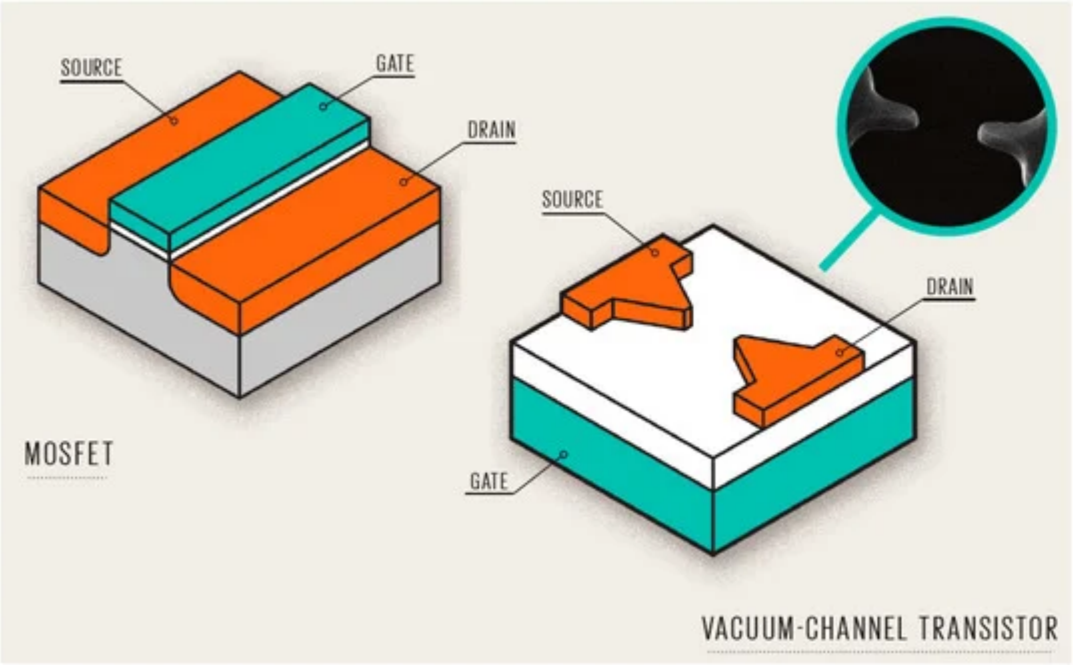
Bipolar Transistors in Microchips



Very Large Scale Integrated Circuits



Back to the Future: Nonotubes



Programming types

- Structured Programming
- Object Oriented Programming
- Functional Programming
- Logical Programming

Java Promise

A better C++

- garbage collector means automatic memory management
- a virtual machine that could run code without recompiling on different platforms

Reality

- use object pools - GC was too slow
- JVM would often crash on different architectures

Let's pretend we don't remember the Java applets

Rust Promise

A better C++

- affine types means automatic memory management
- LLVM means you can easily recompile for different platforms

Reality

- borrow checker will force you to think about memory management
- Microsoft's memory allocator can crash your code at any time

An example of the borrow checker will come in a moment

Hype

Java

Sun's propaganda machine

Rust

Various evangelists

Affine Types

Contraction is forbidden, Weakening and Exchange are allowed

	Left	Right	
Weakening	$\dfrac{\Gamma \vdash \Delta}{\varphi, \Gamma \vdash \Delta}$	$\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \varphi}$	✓
Contraction	$\dfrac{\varphi, \varphi, \Gamma \vdash \Delta}{\varphi, \Gamma \vdash \Delta}$	$\dfrac{\Gamma \vdash \Delta, \varphi, \varphi}{\Gamma \vdash \Delta, \varphi}$	✗
Exchange	$\dfrac{\Gamma, \varphi, \psi, \Pi \vdash \Delta}{\Gamma, \varphi, \psi, \Pi, \vdash \Delta}$	$\dfrac{\Gamma \vdash \Delta, \varphi, \psi, \Lambda}{\Gamma \vdash \Delta, \psi, \varphi, \Lambda}$	✓

Or in plain English: you can use a variable **at most once**.

Ownership

```rust
fn main() {
  let a = String::from("aha");
  let b = a;

  // this will fail as a is already destroyed
  println!("{}", a);
}
```

The checks are actually done at compile time

Functions

```
fn main() {
    let a = String::from("aha");
    // the ownership of a gets transferred to the function
    print(a);

    // this will fail to compile because a no longer exists
    let b = a;
    print(b);
}

fn print(a: String) {
  println!("{}", a);
}
```

Files

```rust
let mut file = File::create("file.txt")?;
file.write_all(b"aha")?;

// file gets used and ceases to exist
std::mem::drop(file);

// we can no longer write to the closed file
file.write_all(b"Hello world");
```

Borrowing

A variable can be borrowed for use, even if it's not owned.

- it can be borrowed multiple times for read only access
- it can be borrowed only once for write access.

Read Only Reference

```rust
fn main() {
    let a = String::from("aha");
    // the ownership of a gets transferred to the function
    print(&a);

    // this will compile because we only sent a reference
    let b = a;
    print(&b);
}

fn print(a: &String) {
  println!("{}", a);
}
```

Write Reference

```rust
fn main() {
    let mut s = String::from("hello");

    let r1 = &mut s;
    let r2 = &mut s;

    println!("{}, {}", r1, r2);
}
```

Enterprise Java

- Services calling other services and exchanging Json
- Lots of security checks
- Some software written only because of regulations
- Lots and lots of caches
- Configuration can bring a system down but is seldom tested

Spring

- The de facto standard in Enterprise Java
- Annotations do most of the magic

```
@DirtiesContext
@Tag("ContractTests")
@Suppress("MagicNumber")
@ExtendWith(WireMockExtension::class, RedisTestContainersExtension::class)
@TestPropertySource(properties = ["TESTCONTAINERS_RYUK_DISABLED=true"])
@SpringBootTest(classes = [StuffProcessorApiApplication::class], webEnvironment =
RANDOM_PORT)
@ActiveProfiles("tst", "local", "integration", "mockdatasources")
class ContractVerifierBase : WireMockServerPortProvider {
    @Autowired
    private lateinit var applicationContext: WebApplicationContext
```

Rust Backend Frameworks

- actix (used to be in the top benchmarks)
- axum
- warp
- rocket

My choice went to hyper, a low-level framework Added layers using tower

- logging
- security
- metrics
- database connection pool
- swagger (utoipa)

Human Factor

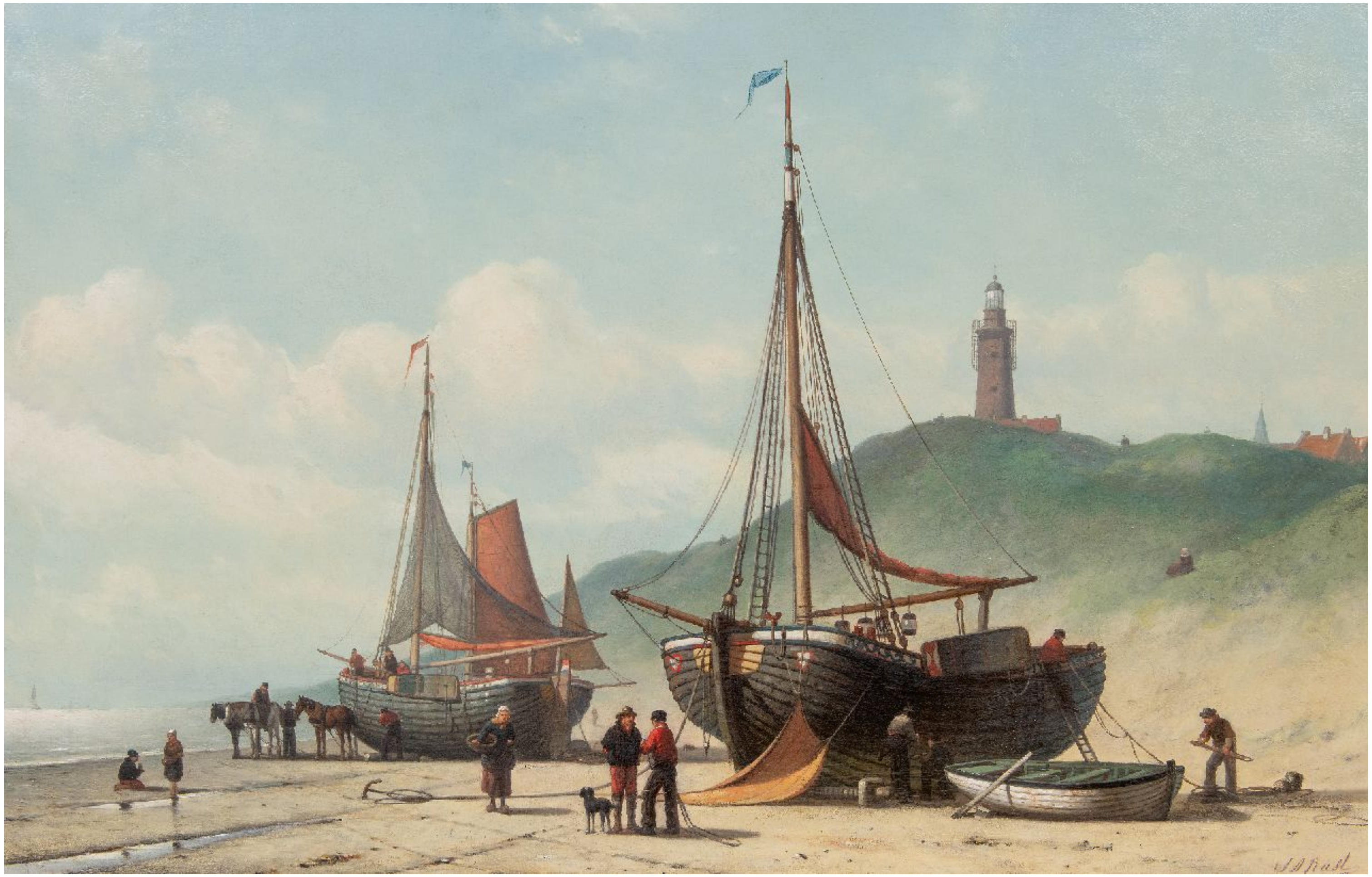
The challenge is not as much technical as it is social

People are reticent to learn a new set of skills where they have to start fresh

The age bracket is curious: only older programmers and very young seems to be open to Rust

Everybody in between are convinced they need a virtual machine. For many going to Rust means compiling to WebAssembly.

Johan Rust



Visserspinken op het strand