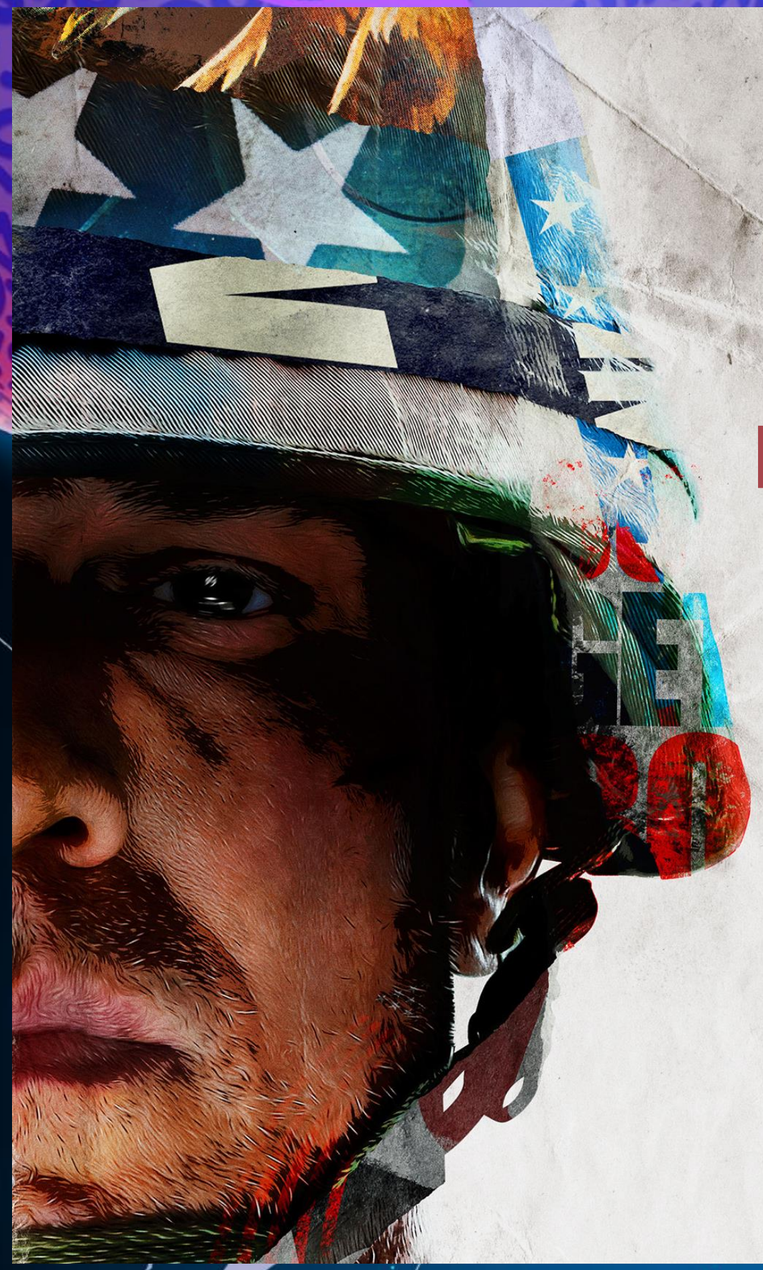
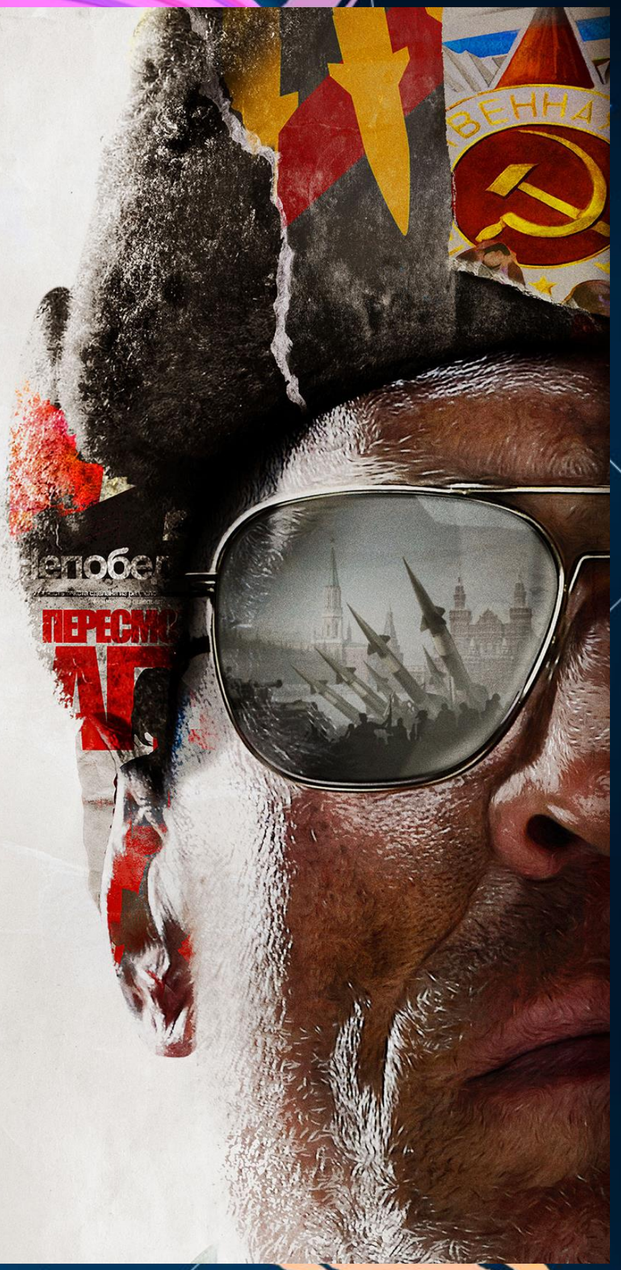


The Rust Programming Language for Game Tooling



Dan Olson
Principal Software Architect
TREYARCH

Who am I?

- Working in games since 2004, Treyarch since 2008.
- Core Engine team.
- Focused on data pipeline and infrastructure tooling.

What is Rust?

- Started by Mozilla in 2006, stable release in 2015.
- Currently supported by Amazon, Facebook, Microsoft, and others.
- Focused on security and performance.

Outline

- The case for Rust.
- Survey of several interesting uses for Rust.
- Integrating Rust at Treyarch.

Code comparison: md5sum

- Easy to read.
- Easy to write.
- Errors are handled.
- Performs well.

```
hasher = hashlib.md5()
with open(filename, 'rb') as f:
    hasher.update(f.read())
print(hasher.hexdigest())
```

Code comparison: md5sum

- Easy to read. 😀
- Easy to write. 😀
- Errors are handled.
- Performs well.

```
hasher = hashlib.md5()
with open(filename, 'rb') as f:
    hasher.update(f.read())
print(hasher.hexdigest())
```

Code comparison: md5sum

- Easy to read. 😀
- Easy to write. 😀
- Errors are handled. 😐
- Performs well.

```
hasher = hashlib.md5()
with open(filename, 'rb') as f:
    hasher.update(f.read())
print(hasher.hexdigest())
```

Code comparison: md5sum

- Easy to read. 😀
- Easy to write. 😀
- Errors are handled. 😐
- Performs well. 🤔

```
hasher = hashlib.md5()
with open(filename, 'rb') as f:
    hasher.update(f.read())
print(hasher.hexdigest())
```

Code comparison: md5sum

- Easy to read.
- Easy to write.
- Errors are handled.
- Performs well.

```
int file_descript = open(filename, O_RDONLY);
if(file_descript < 0) exit(-1);

unsigned long file_size =
    get_size_by_fd(file_descript);

char* file_buffer = mmap(0, file_size,
    PROT_READ, MAP_SHARED, file_descript, 0);
MD5((unsigned char*) file_buffer, file_size,
    result);
munmap(file_buffer, file_size);

print_md5_sum(result);
```

Source: https://stackoverflow.com/a/1220177/69283

Code comparison: md5sum

- Easy to read. 😐
- Easy to write.
- Errors are handled.
- Performs well.

```
int file_descript = open(filename, O_RDONLY);
if(file_descript < 0) exit(-1);

unsigned long file_size =
    get_size_by_fd(file_descript);

char* file_buffer = mmap(0, file_size,
    PROT_READ, MAP_SHARED, file_descript, 0);
MD5((unsigned char*) file_buffer, file_size,
    result);
munmap(file_buffer, file_size);

print_md5_sum(result);
```

Source: https://stackoverflow.com/a/1220177/69283

Code comparison: md5sum

- Easy to read. 😐
- Easy to write. 😢
- Errors are handled.
- Performs well.

```
int file_descript = open(filename, O_RDONLY);
if(file_descript < 0) exit(-1);

unsigned long file_size =
    get_size_by_fd(file_descript);

char* file_buffer = mmap(0, file_size,
    PROT_READ, MAP_SHARED, file_descript, 0);
MD5((unsigned char*) file_buffer, file_size,
    result);
munmap(file_buffer, file_size);

print_md5_sum(result);
```

Source: https://stackoverflow.com/a/1220177/69283

Code comparison: md5sum

- Easy to read. 😐
- Easy to write. 😢
- Errors are handled. 😬
- Performs well.

```
int file_descript = open(filename, O_RDONLY);
if(file_descript < 0) exit(-1);

unsigned long file_size =
    get_size_by_fd(file_descript);

char* file_buffer = mmap(0, file_size,
    PROT_READ, MAP_SHARED, file_descript, 0);
MD5((unsigned char*) file_buffer, file_size,
    result);
munmap(file_buffer, file_size);

print_md5_sum(result);
```

Source: https://stackoverflow.com/a/1220177/69283

Code comparison: md5sum

- Easy to read. 😐
- Easy to write. 😢
- Errors are handled. 😬
- Performs well. 😀

```
int file_descript = open(filename, O_RDONLY);
if(file_descript < 0) exit(-1);

unsigned long file_size =
    get_size_by_fd(file_descript);

char* file_buffer = mmap(0, file_size,
    PROT_READ, MAP_SHARED, file_descript, 0);
MD5((unsigned char*) file_buffer, file_size,
    result);
munmap(file_buffer, file_size);

print_md5_sum(result);
```

Source: https://stackoverflow.com/a/1220177/69283

Code comparison: md5sum

- Easy to read.
- Easy to write.
- Errors are handled.
- Performs well.

```
let data = std::fs::read(filename)?;
let hash = md5::compute(&data);
println!("{:x}", hash);
```

Code comparison: md5sum

- Easy to read. 😀
- Easy to write. 😀
- Errors are handled.
- Performs well.

```
let data = std::fs::read(filename)?;
let hash = md5::compute(&data);
println!("{:x}", hash);
```

Code comparison: md5sum

- Easy to read. 😀
- Easy to write. 😀
- Errors are handled. 😐
- Performs well.

```
let data = std::fs::read(filename)?;
let hash = md5::compute(&data);
println!("{:x}", hash);
```

Code comparison: md5sum

- Easy to read. 😀
- Easy to write. 😀
- Errors are handled. 😐
- Performs well. 😀

```
let data = std::fs::read(filename)?;
let hash = md5::compute(&data);
println!("{:x}", hash);
```

Dan’s Rust Sales Pitch

- Efficiency of writing code: closer to Python.
- Efficiency of running code: closer to C++.
- Large, centralized ecosystem of “crates”, or community libraries (https://crates.io/).
- Integrated build + package + test tool (“cargo”).

Code comparison: md5sum

```
q:\>cargo new md5sum
     Created binary (application) `md5sum` package
q:\>cd md5sum
q:\md5sum>cargo add md5
      Adding md5 v0.7.0 to dependencies
q:\md5sum>cargo run -- src/main.rs
   Compiling md5 v0.7.0
   Compiling md5sum v0.1.0 (Q:\md5sum)
    Finished dev [unoptimized + debuginfo] target(s) in 1.66s
     Running `target\debug\md5sum.exe src/main.rs`
4911739566caf58bf40be5b6d6a19262
```

Code comparison: md5sum

```
q:\>cargo new md5sum
     Created binary (application) `md5sum` package
q:\>cd md5sum
q:\md5sum>cargo add md5
      Adding md5 v0.7.0 to dependencies
q:\md5sum>cargo run -- src/main.rs
   Compiling md5 v0.7.0
   Compiling md5sum v0.1.0 (Q:\md5sum)
    Finished dev [unoptimized + debuginfo] target(s) in 1.66s
     Running `target\debug\md5sum.exe src/main.rs`
4911739566caf58bf40be5b6d6a19262
```

Code comparison: md5sum

```
q:\>cargo new md5sum
     Created binary (application) `md5sum` package
q:\>cd md5sum
q:\md5sum>cargo add md5
      Adding md5 v0.7.0 to dependencies
q:\md5sum>cargo run -- src/main.rs
   Compiling md5 v0.7.0
   Compiling md5sum v0.1.0 (Q:\md5sum)
    Finished dev [unoptimized + debuginfo] target(s) in 1.66s
     Running `target\debug\md5sum.exe src/main.rs`
4911739566caf58bf40be5b6d6a19262
```

Code comparison: md5sum

```
q:\>cargo new md5sum
     Created binary (application) `md5sum` package
q:\>cd md5sum
q:\md5sum>cargo add md5
      Adding md5 v0.7.0 to dependencies
q:\md5sum>cargo run -- src/main.rs
   Compiling md5 v0.7.0
   Compiling md5sum v0.1.0 (Q:\md5sum)
    Finished dev [unoptimized + debuginfo] target(s) in 1.66s
     Running `target\debug\md5sum.exe src/main.rs`
4911739566caf58bf40be5b6d6a19262
```

Dan’s Rust Sales Pitch

- Efficiency of writing code: closer to Python.
- Efficiency of running code: closer to C++.
- Large, centralized ecosystem of “crates”, or community libraries (https://crates.io/).
- Integrated build + package + test tool (“cargo”).
- Static, compile-time validation of common memory problems.
- Static, compile-time validation of common multithreading problems.

Case study: Treyarch Image Packer

- Rust version deployed in 2018.
- Heavily multithreaded.
- Active development throughout its lifetime.
- Total “crash” issues encountered: 2.

Dan’s Rust Sales Pitch

- Efficiency of writing code: closer to Python.
- Efficiency of running code: closer to C++.
- Large, centralized ecosystem of “crates”, or community libraries (https://crates.io/).
- Integrated build + package + test tool (“cargo”).
- Static, compile-time validation of common memory problems.
- Static, compile-time validation of common multithreading problems.

Rust for Game Tools

- Error Handling
- Multithreading
- Parsing Text
- Command Line Interfaces
- Parsing Debug Info
- C ABI compatibility
- Web Applications
- GUIs

1/8 - Error Handling

- Result – holds the success or failure state of an operation.
- Panic – instant program failure for unrecoverable errors.
- ? Operator – pass a failed Result up the callstack.

```
let file = std::fs::read(path)?;
```

- Use the **anyhow** crate to add context to errors.

1/8 - Error Handling

```
let file = std::fs::read(path)?;
```

Error: The system cannot find the file specified. (os error 2)

```
let file = std::fs::read(path)
    .with_context(|| format!("Reading contents of {:?}", path))?;
```

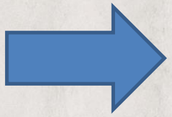
Error: Reading contents of "test.txt"

Caused by:
    The system cannot find the file specified. (os error 2)

2/8 - Multithreading

- Using the `rayon` crate, multithreading is quick, easy, and safe.

```
file_names
    .iter()
    .map(|x| hash_file(x))
    .collect();
```



```
file_names
    .par_iter()
    .map(|x| hash_file(x))
    .collect();
```

2/8 - Multithreading

- Using the **rayon** crate, multithreading is quick, easy, and safe.

```
file_names
    .par_iter()
    .map(|x| hash_file(x, &mut count))
    .collect();
```

```
  --> src\main.rs:16:31
   |
16 |         .map(|x| hash_file(x, &mut count))
   |                               ^^^^^^^^^^ cannot borrow as mutable

error: aborting due to previous error
```

3/8 - Parsing Text

- Use the `serde` crate for generic serialization/deserialization.

```rust
#[derive(Deserialize)]
struct Config {
    string: String,
    number: i32,
    list: Vec<String>,
}
```

```rust
let config: Config = serde_json::from_str(&text)?;
```

```rust
let config: Config = serde_yaml::from_str(&text)?;
```

4/8 - Command Line Interfaces

- Use the `structopt` crate to create command line interfaces.

```
sourcehash 0.1.0
Generate a hash of all source and include files specified by one or
more .vcxproj files

USAGE:
    sourcehash.exe [FLAGS] [files]...

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information
        --verbose

ARGS:
    <files>...    One or more vcxproj files contributing to the hash
```

5/8 - Parsing Debug Info

- Use `pdb` (win) and `gimli` (elf) crates to inspect debug info.
- This tool is now open-sourced!
- https://github.com/Activision/structpack

```
gz_header_s - 80 bytes, 12 padding
(optimal size should be 72 bytes, 4 padding)
struct gz_header_s
{
    int text; // 4 bytes
    <padding> ; // 4 bytes
    uLong time; // 8 bytes
    int xflags; // 4 bytes
    int os; // 4 bytes
    Bytef* extra; // 8 bytes
    uInt extra_len; // 4 bytes
    uInt extra_max; // 4 bytes
    Bytef* name; // 8 bytes
    uInt name_max; // 4 bytes
    <padding> ; // 4 bytes
    Bytef* comment; // 8 bytes
    uInt comm_max; // 4 bytes
    int hcrc; // 4 bytes
    int done; // 4 bytes
    <padding> ; // 4 bytes
};
```

6/8 - C ABI compatibility

- Bind Rust code to other languages (e.g. python, nodejs, C, wasm).

```rust
/// Formats the sum of two numbers as string.
#[pyfunction]
fn sum_as_string(a: usize, b: usize) -> PyResult<String> {
    Ok((a + b).to_string())
}

/// A Python module implemented in Rust.
#[pymodule]
fn string_sum(py: Python, m: &PyModule) -> PyResult<()> {
    m.add_function(wrap_pyfunction!(sum_as_string, m)?)?;

    Ok(())
}
```

Source: https://crates.io/crates/pyo3

7/8 - Web Applications

- There are lots and lots of crates for web apps. I like **`rouille`** for quick, simple ones.

```
rouille::start_server("0.0.0.0:80", move |request| {
    Response::text("hello world")
});
```

Source: https://docs.rs/rouille/3.1.1/rouille/

- But **`tide`** or **`actix-web`** might be better for more substantial apps.

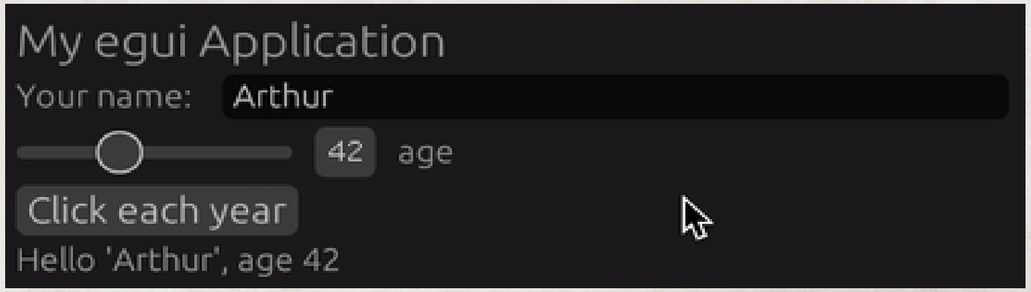
```
#[async_std::main]
async fn main() -> tide::Result<()> {
    let mut app = tide::new();
    app.at("/orders/shoes").post(order_shoes);
    app.listen("127.0.0.1:8080").await?;
    Ok(())
}
```

Source: https://crates.io/crates/tide

8/8 - GUIs

- Native Rust: `iced`, `druid`, `egui`
- C++ Bindings: `ritual` (Qt), `relm` (Gtk), `imgui-rs`, `web-view`

```
ui.heading("My egui Application");
ui.horizontal(|ui| {
    ui.label("Your name: ");
    ui.text_edit_singleline(&mut name);
});
ui.add(egui::Slider::new(&mut age, 0..=120).text("age"));
if ui.button("Click each year").clicked() {
    age += 1;
}
ui.label(format!("Hello '{}', age {}", name, age));
```



Source: https://crates.io/crates/egui

Integrating Rust at Treyarch

- Started work in late 2017.
- 3 major tools, around 20 smaller one-off tools.
- Around 120K LOC
- 27 individual contributors.
- Extremely high stability.

Rust Downsides

- Steep learning curve… very different from C++.
- Complicated language… lots of corners.
- Relies heavily on ecosystem.
- Compile times as bad as or worse than C++.

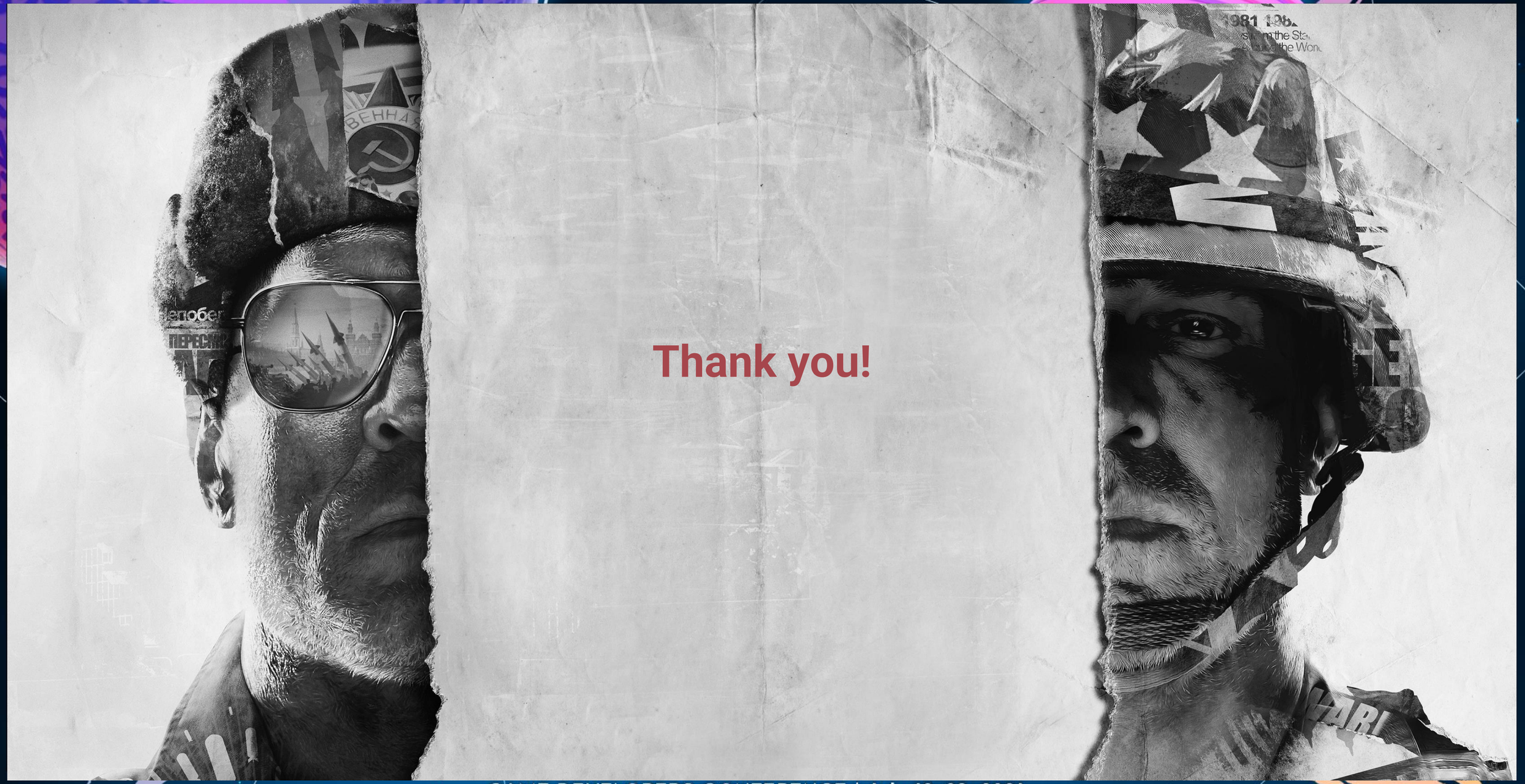
Managing these downsides is key to successful integration!

Integration Tips

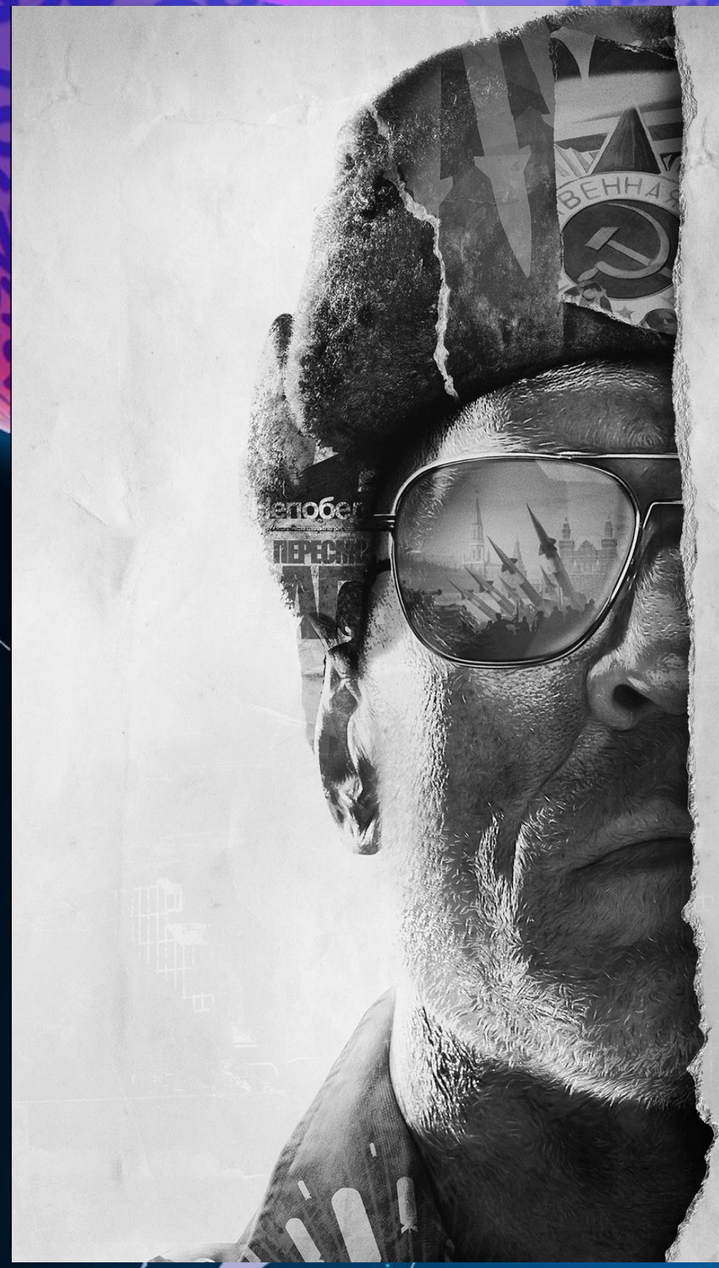
- Communicate!
- Keep Rust siloed off until there is a critical mass of experience.
  - Find an unmaintained tool and make it better!
- Start with the best workflow.
  - vscode + rust_analyzer + cargo-edit + rustfmt + clippy
- Bad Rust code has the same safety guarantees as good Rust code!
- Initial hurdles are high, but large productivity gains after they are cleared.
- Provide time and space to both learn and teach.
  - Presentations, courses, examples, code reviews.

Learning Resources

- Dive into code!
  - https://leetcode.com/
  - https://adventofcode.com/
- Dive into a book!
  - “The Rust Programming Language” – Klabnik & Nichols
  - “Programming Rust” – Blandy & Orendorff
- Dive into online resources!
  - https://www.rust-lang.org/learn
  - https://play.rust-lang.org/



Thank you!



TREYARCH

We're Hiring!

Gameplay Engineers (mid/senior)
UI Engineers (mid/senior)
Online Engineers (mid/senior)
Senior Test Automation Engineers

And many more!

https://careers.treyarch.com/