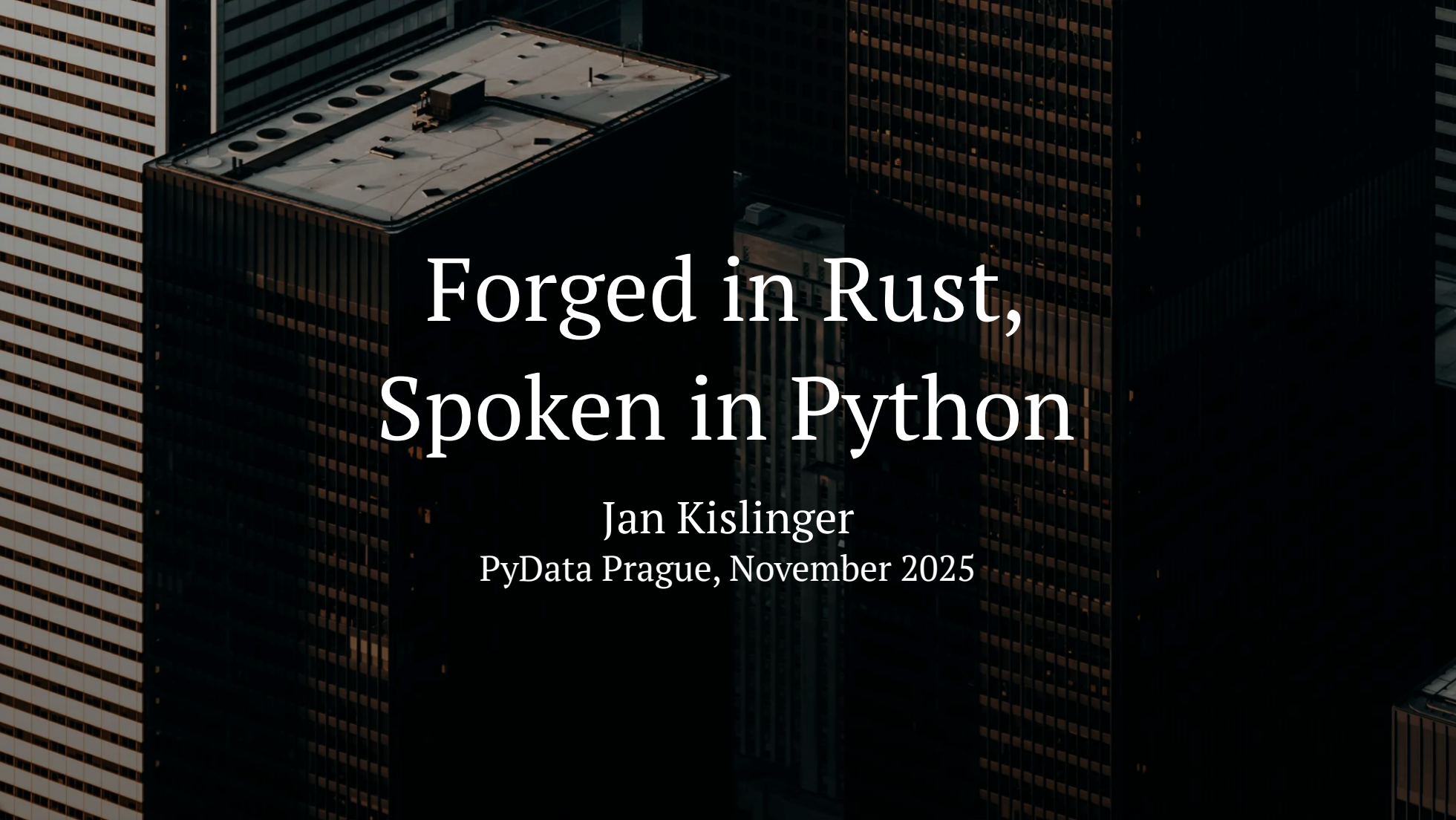


Forged in Rust, Spoken in Python

Jan Kislinger

PyData Prague, November 2025

2018: R ⇨ Python

- dplyr ⇨ Polars
- ggplot2 😔
- Rcpp ⇨ PyO3 + maturin

Why Rust

- Performance
- Most admired language
- Effortless integration

You'll Learn Today

- Basic Rust concepts
- Call Rust from Python
- Popular projects
- Write Polars extension
- (+1 bonus tip)

Concepts of Rust

- Compiled
- Strongly typed
- Struct (no inheritance)
- Trait, Generics
- Enum
- Option, Result
- Error as value
- Borrow checker
- Macros

```rust
#[derive(Debug)]
struct Point {
    x: i32,
    y: i32,
}

fn main() {
    let point = Point { x: 1, y: 2 };
    println!("{:?}", point);
    // Point { x: 1, y: 2 }
}
```

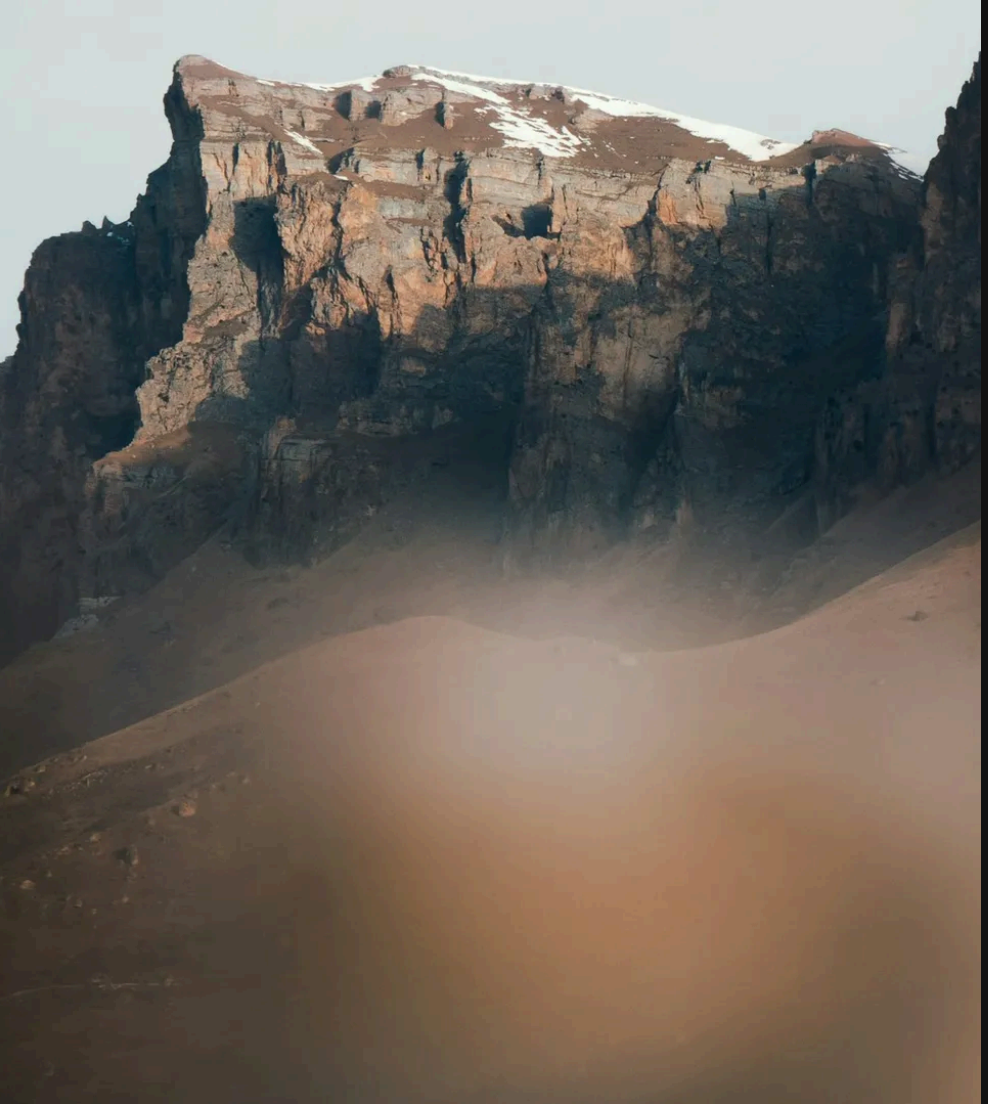
Development Tools

Python

- Environment: `pip`, `virtualenv`, `pipx`, `poetry`, `uv`
- Formatting: `black`, `isort`, `flake8`, `ruff`
- Static Analysis: `mypy`, `ty`
- Documentation: `pydoc`, `sphinx`

Rust

- `cargo`



Just in Python

Hidden features

- `int` (unbounded)
- `str` (cached)
- `dict` (sorted)

Popular Projects

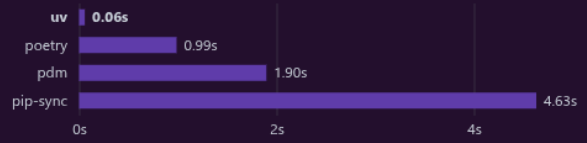
Written in Rust; Used by Python developers

Python Tooling

astral.sh

uv

An extremely fast Python package and project manager, written in Rust.



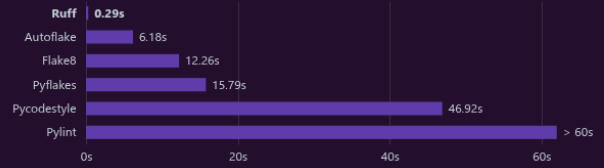
Installing Trio's dependencies with a warm cache.

Ruff

Ruff | pypi v0.14.6 | license MIT | python 3.7 | 3.8 | 3.9 | 3.10 | 3.11 | 3.12 | 3.13 | CI failing | Discord

Docs | Playground

An extremely fast Python linter and code formatter, written in Rust.



Linting the CPython codebase from scratch.

- Data frame library
- Backend written in Rust
- Multithreaded by default
- 10x - 30x faster than Pandas

- Lazy expressions 🤗
- Consistent syntax
- No (multi-) indices

```python
# Polars
result = (
    pl.scan_csv("data.csv")
    .filter(pl.col("value") > 10)
    .with_columns(double=pl.col("value") * 2)
    .group_by("category")
    .agg(double_mean=pl.col("double").mean())
    .collect()
)
```

```python
# Pandas
result = (
    pd.read_csv("data.csv")
    .iloc[lambda d: d["value"] > 10]
    .assign(double=lambda d: d["value"] * 2)
    .groupby("category")
    .agg(double_mean=("double", "mean"))
    .reset_index()
)
```

```r
# R::dplyr
result <- read_csv("data.csv") |>
  filter(value > 10) |>
  mutate(double = value * 2) |>
  group_by(category) |>
  summarise(double_mean = mean(double, na.rm = TRUE))
```

Other Examples

orjson

Fast, correct JSON library for Python.

Pydantic

Data validation using Python type hints.

Robyn

High-Performance [...] Web Framework with a Rust runtime.

PyO3 + maturin

Call Rust from Python

- Create new project
- Write Rust code
- Decorate using `pyo3` macros
- Export as Python module
- Compile & install
- Run Python

```python
# main.py
from my_lib import fibo

print(fibo(12))
```

Binding Structs to Classes

```rust
#[pyclass]
#[derive(Debug)]
struct Accumulator {
    value: i64,
}

#[pymethods]
impl Accumulator {
    #[new]
    fn new() -> Self {
        Self { value: 0 }
    }

    fn add(&mut self, x: i64) { ... }
    fn get(&self) -> i64 { ... }

    fn __str__(&self) -> String {
        format!("{:?}", &self)
    }
}
```

```python
from my_lib import Accumulator

x = Accumulator()
x.add(1)
x.add(2)

print(x.get())
# 3

print(x)
# Accumulator { value: 3 }
```

Python Exceptions

```rust
use pyo3::prelude::*;

#[pyfunction]
fn parse_int(text: String) -> PyResult<i64> {
    text.trim().parse()
        .map_err(PyValueError::new_err)
}

#[pymodule]
mod my_lib {
    #[pymodule_export]
    use super::parse_int;
}
```

```python
from my_lib import parse_int

assert parse_int("  123 ") == 123

try:
    parse_int("foo")
except ValueError as e:
    print(e)
    # invalid digit found in string
```

Good Case for Rust

- Performance-critical code
- Tight loops over large arrays
- Parsing / validation
- Specific algorithm
- Wrapping an existing Rust crate

Stay in Python

- I/O-bound tasks
- Leverage existing library
- Rapid prototyping
- Interactive data exploration

(Not) Calling Python from Rust

```python
import polars as pl

df = (
    pl.DataFrame({"x": [1, 2, 3]})
    .with_columns(y=pl.col("x").map_elements(lambda x: x+1))
)

# Expr.map_elements is significantly slower than the native expressions API.
# Only use if you absolutely CANNOT implement your logic otherwise.
# Replace this expression...
#   - pl.col("x").map_elements(lambda x: ...)
# with this one instead:
#   + pl.col("x") + 1
#
#   .with_columns(y=pl.col("x").map_elements(lambda x: x+1))
```

Polars Extension

```python
import polars as pl
from polars.plugins import register_plugin_function

LIB = Path(__file__).parent / "my_lib.abi3.so"

def farm64(col_name: str) -> pl.Expr:
    return register_plugin_function(
        plugin_path=LIB,
        args=[pl.col(col_name)],
        function_name="farm64",
        is_elementwise=True,
    )

(
    pl.DataFrame({"text": ["hello", "world"]})
    .with_columns(hash=farm64("text"))
)
```

```rust
use polars::prelude::*;
use pyo3_polars::derive::polars_expr;

#[polars_expr(output_type=UInt64)]
fn farm64(inputs: &[Series]) -> PolarsResult<Series> {
    let strings: &StringChunked = inputs[0].str()?;
    let hashes: UInt64Chunked = strings.iter()
        .map(|x| x.map(farm::fingerprint64))
        .collect();
    Ok(hashes.into_series())
}

#[pymodule]
mod my_lib {}
```

Docker Image Optimization

```
// pipeline.rs
let lf = LazyCsvReader::new("data.csv")
    .has_header(true)
    .finish()?
    .filter(col("value").gt(lit(10)))
    .with_columns([
        col("value").mul(lit(2)).alias("double"),
    ])
    .groupby([col("category")])
    .agg([
        col("double").mean().alias("double_mean"),
    ]);

lf.sink(
    SinkDestination::File("result.csv".into()),
    FileType::Csv.into(),
    Default::default(),
)?;
```

```
FROM rust as builder

COPY . /build
RUN cargo install

FROM debian
COPY --from builder cargo/bin/myapp /bin/myapp
```

Conclusion

- Rust performance with the interactivity of Python
- The ecosystem is ready: PyO3, maturin
- Several projects have already shown success
- Start tiny: one function, one Polars plugin

Q & A

Jan Kislinger

- ML Engineer
- Personalization

- PhD student
- Full-page recommendations