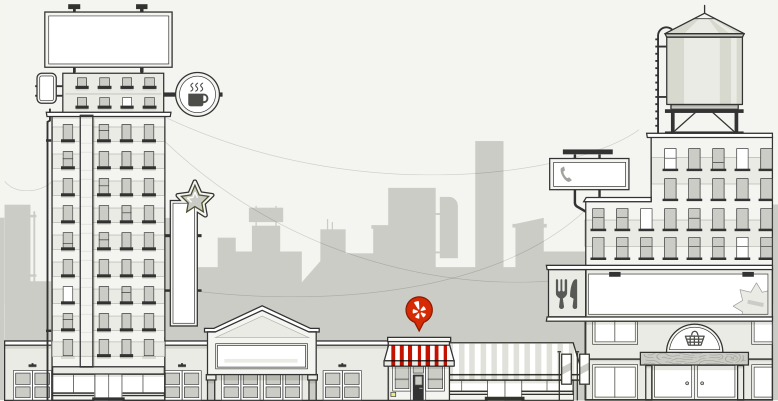
How to write Rust instead of C and get away with it

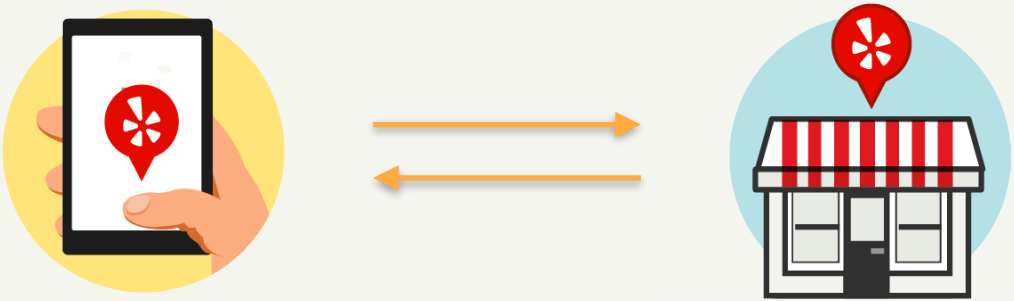
(yes, it's a Python talk)

Antonio Verardi – Flavien Raynaud
@porosVII – @flavray

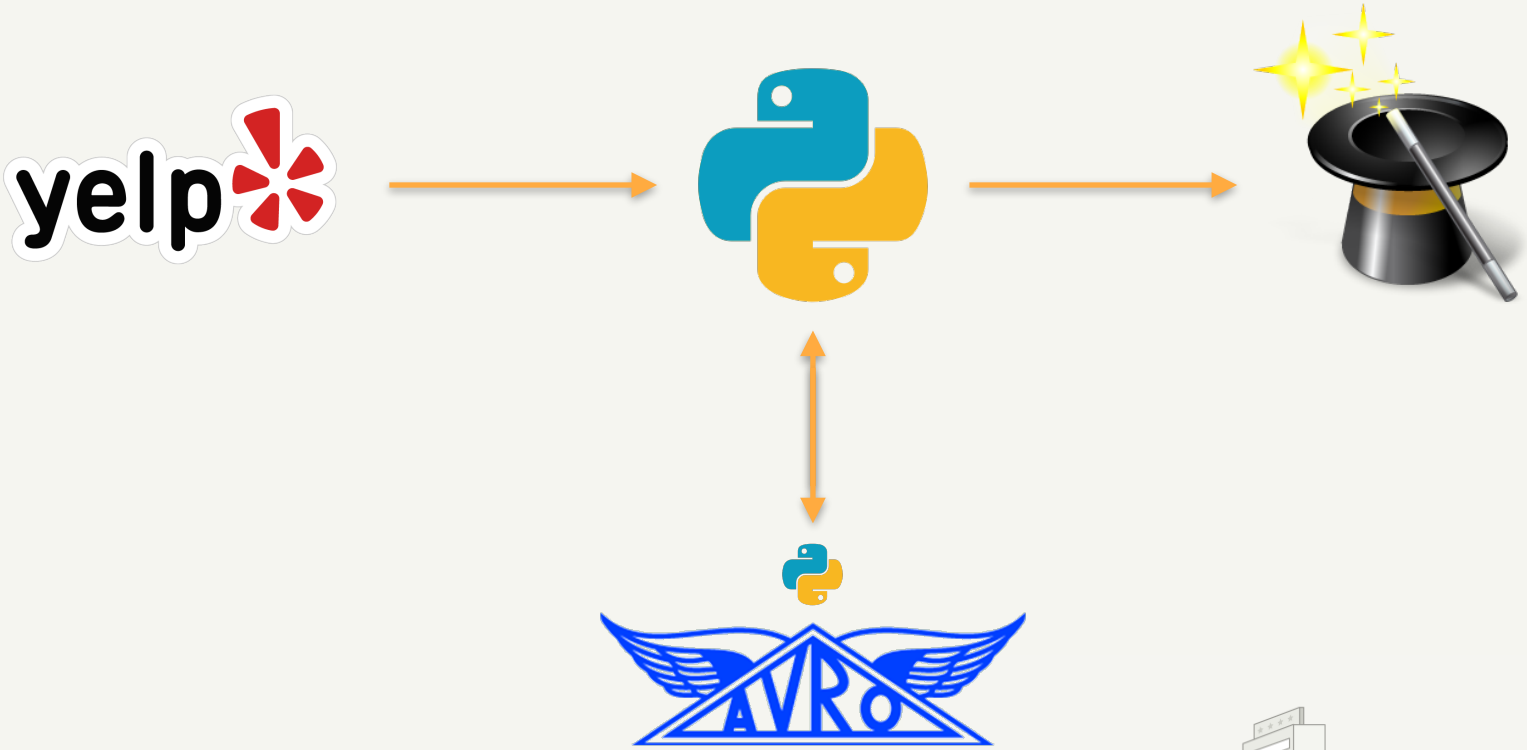


Yelp's Mission

Connecting people with great local businesses.



Why we are here


yelp
AVRO


```
schema = {
  "type": "record",
  "name": "my_record",
  "fields": [
    {
      "name": "a",
      "type": "long",
      "default": 42
    },
    {
      "name": "b",
      "type": "string"
    },
  ]
}
```


```
schema = {
  "type": "record",
  "name": "my_record",
  "fields": [
    {
      "name": "a",
      "type": "long",
      "default": 42
    },
    {
      "name": "b",
      "type": "string"
    },
  ]
}
```

Binary Data Serialization format

JSON

```
{"a": 27, "b": "foo"}
```

Apache Avro™

```
0x 36 06 66 6f 6f
```

https://avro.apache.org

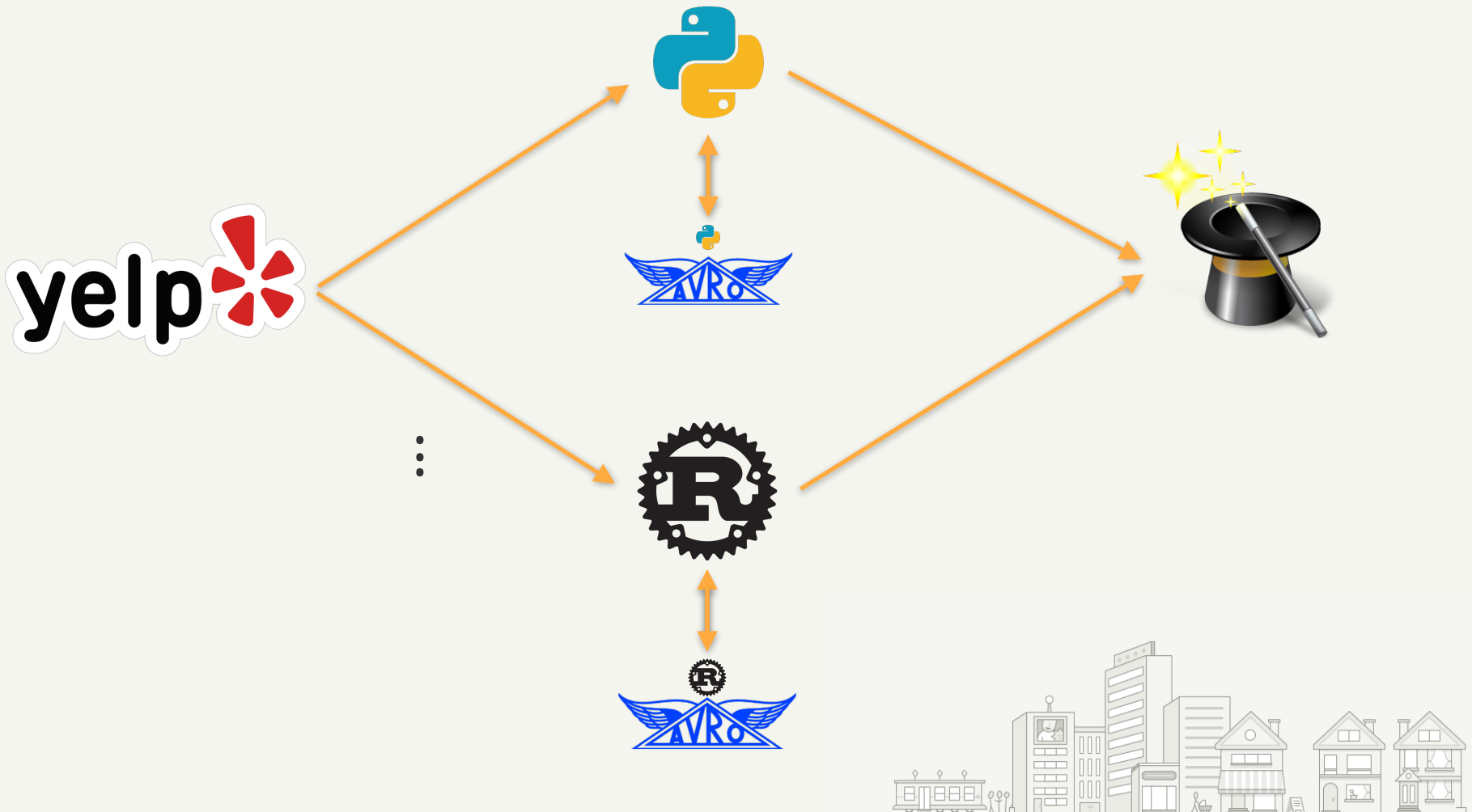
2004

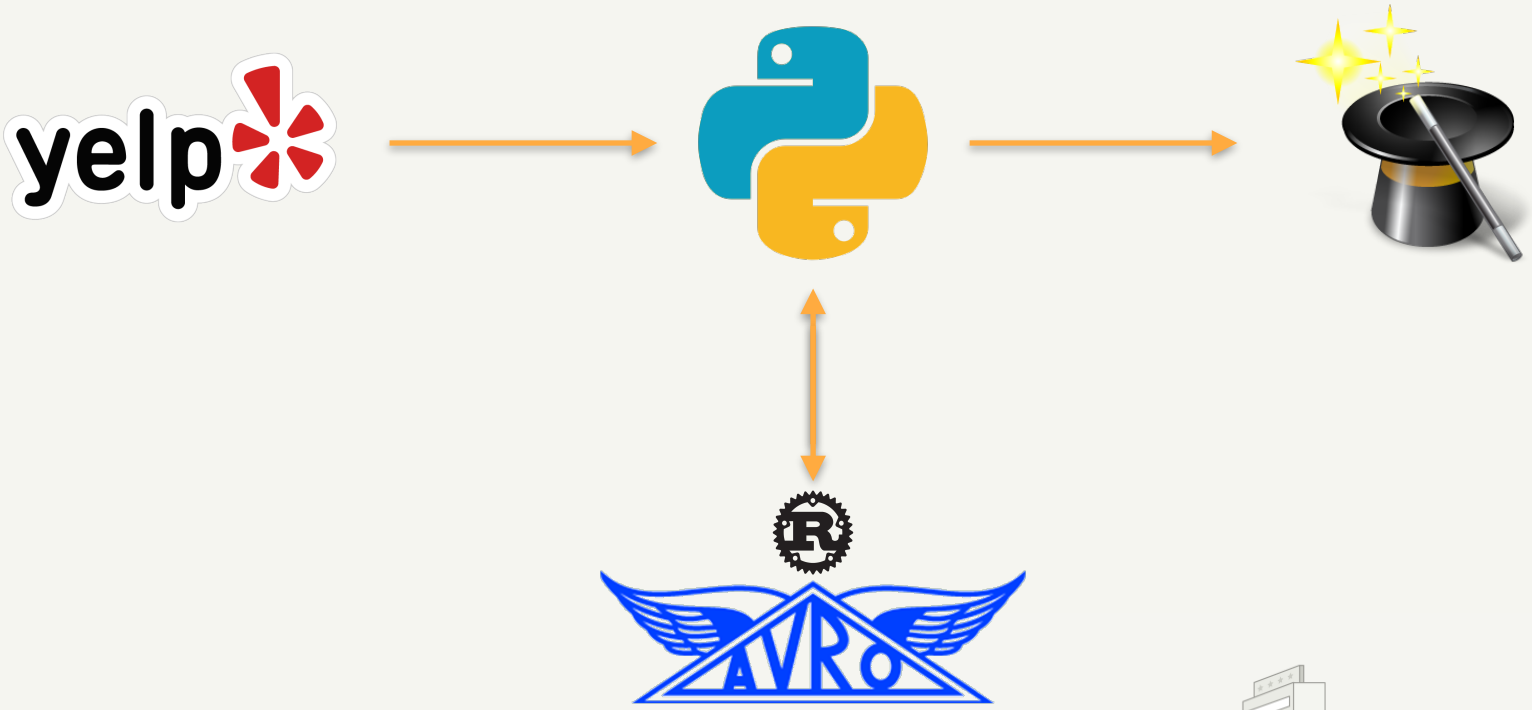
2018

...


yelp


AVRO

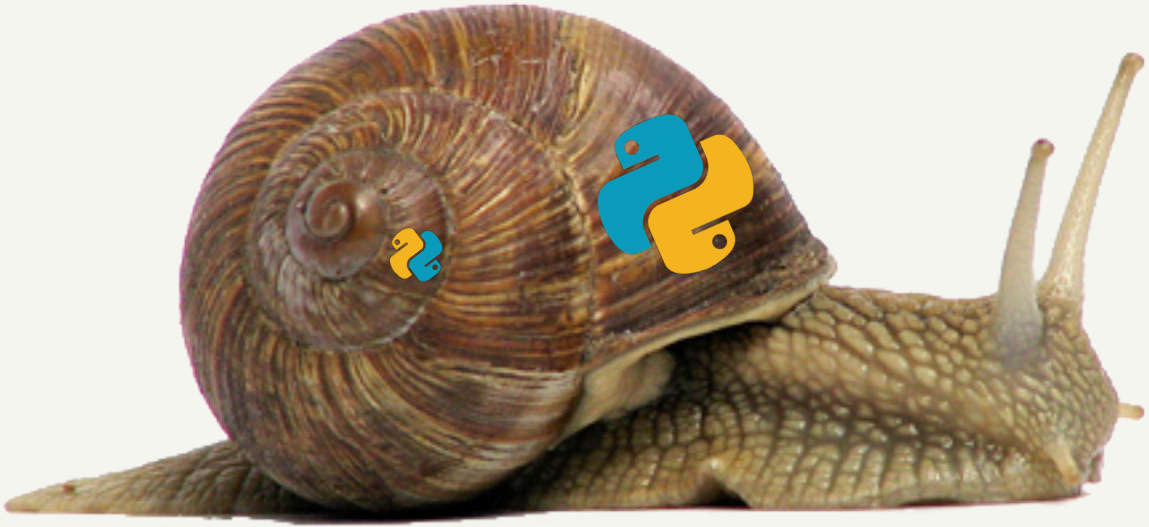

yelp
AVRO
AVRO


yelp
AVRO

What you will bring home

- How to use Rust code in Python packages
- Why and When
- Python, C and Rust facts
- Jokes?

The Problem



Scale up/out

Scale up/out ¯_(ツ)_/¯

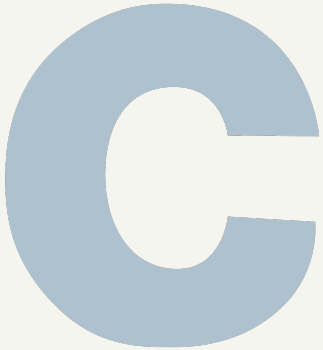
Change Interpreter


pypy

Use

THE C PROGRAMMING LANGUAGE

THE



PROGRAMMING
LANGUAGE

C Extensions

THE

C

PROGRAMMING LANGUAGE

ctypes / cffi

THE

C

PROGRAMMING

LANGUAGE

pyavro-rs

ctypes / cffi

THE

C

PROGRAMMING
LANGUAGE


THE
C
PROGRAMMING
LANGUAGE
fastavro
Cython

How to CFFI

```
from cffi import FFI

ffi = FFI()

with open("my_header.h") as header:
    ffi.cdef(header.read())

lib = ffi.dlopen("my_binary.so")

lib.my_function()
```

```
from cffi import FFI

ffi = FFI()

with open("my_header.h") as header:
    ffi.cdef(header.read())

lib = ffi.dlopen("my_binary.so")

lib.my_function()
```

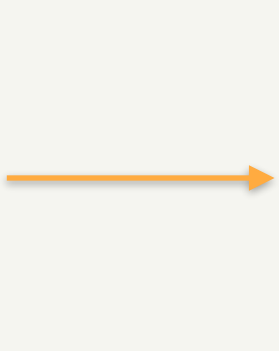
```
from cffi import FFI

ffi = FFI()

with open("my_header.h") as header:
    ffi.cdef(header.read())

lib = ffi.dlopen("my_binary.so")

lib.my_function()
```

```
from cffi import FFI

ffi = FFI()

with open("my_header.h") as header:
    ffi.cdef(header.read())

lib = ffi.dlopen("my_binary.so")

lib.my_function()
```

ABI

Application Binary Interface

ABI



C

ABI

ABI

ABI

C

C

Rust

- guaranteed memory safety

Rust

- guaranteed memory safety
- concurrency without data races

Rust

- guaranteed memory safety
- concurrency without data races
- zero-cost abstractions

Rust

- guaranteed memory safety
- concurrency without data races
- zero-cost abstractions
- modern syntax

Rust

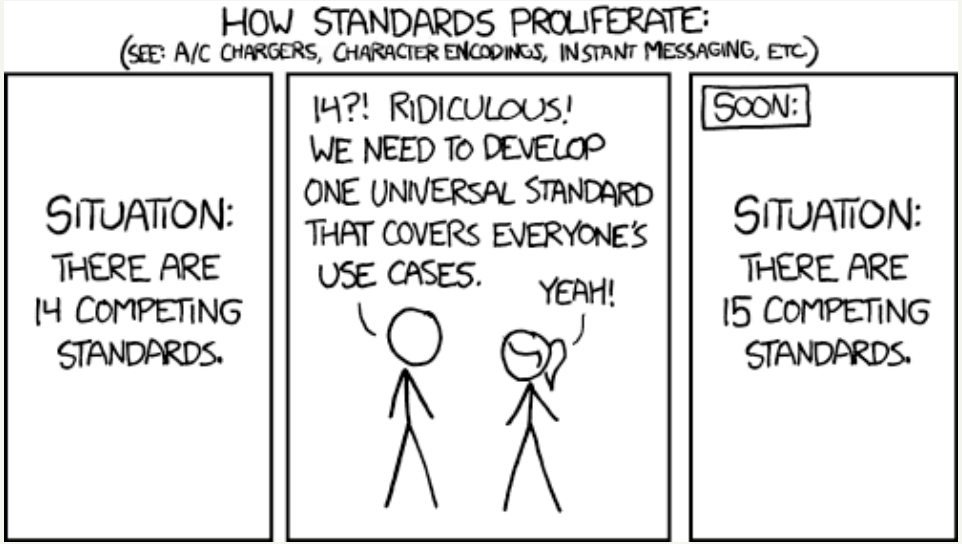
- guaranteed memory safety
- concurrency without data races
- zero-cost abstractions
- modern syntax
- awesome tooling

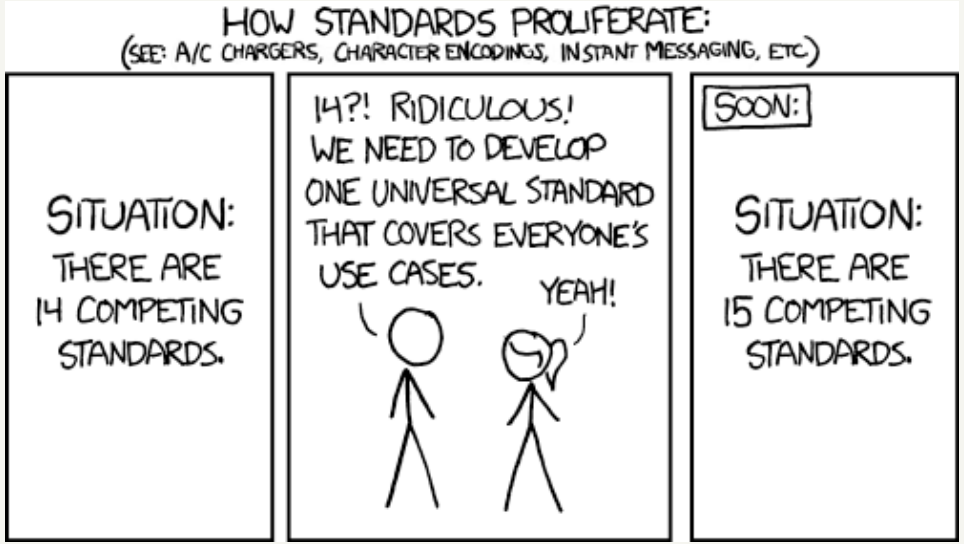
Rust

- guaranteed memory safety
- concurrency without data races
- zero-cost abstractions
- modern syntax
- awesome tooling 

How to Avro

- serialization **and** deserialization
- serde.rs
  - implement `Serializer` & `Deserializer`
  - code generation
- avro.apache.org/docs/current/spec.html


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION:
THERE ARE
14 COMPETING
STANDARDS.
14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.
YEAH!
SOON:
SITUATION:
THERE ARE
15 COMPETING
STANDARDS.



flavray / **avro-rs**

Unwatch 3 | Star 10 | Fork 7

Code | Issues 2 | Pull requests 1 | Projects 0 | Wiki | Insights | Settings

Avro client library implementation in Rust

Edit

Add topics

96 commits | **2** branches | **7** releases | **4** contributors | MIT

```rust
#[derive(Debug, Deserialize, Serialize)]
pub struct Test {
    pub a: i64,
    pub b: String,
}
```

```rust
#[derive(Debug, Deserialize, Serialize)]
pub struct Test {
    pub a: i64,
    pub b: String,
}
```

```
let schema = Schema::parse_str(r#"{
  "type": "record",
  "name": "test",
  "fields": [
    {
      "name": "a",
      "type": "long",
      "default": 42
    },
    {
      "name": "b",
      "type": "string"
    },
  ]
}"#)?;
```

```rust
let mut writer = Writer::new(
    &schema,
    Vec::new()  // io::Write
);
```

```
let mut writer = Writer::new(
    &schema,
    Vec::new()  // io::Write
);

let record1 = Test { a: 27, b: "foo".to_owned() };
```

```
let mut writer = Writer::new(
    &schema,
    Vec::new()  // io::Write
);

let record1 = Test { a: 27, b: "foo".to_owned() };

let record2 = Record::new();
record2.put("a", 27);
record2.put("b", “foo”.to_owned());
```

```
let mut writer = Writer::new(
    &schema,
    Vec::new()  // io::Write
);

let record1 = Test { a: 27, b: "foo".to_owned() };

let record2 = Record::new();
record2.put("a", 27);
record2.put("b", “foo”.to_owned());

writer.append_ser(record1);
writer.append(record2);
writer.flush();
```

```rust
let mut bytes = writer.into_inner();
```

```rust
let mut bytes = writer.into_inner();

let mut reader = Reader::new(&bytes[..]);
```

```rust
let mut bytes = writer.into_inner();

let mut reader = Reader::new(&bytes[..]);

for record in reader {
  let test = from_value::<Test>(&record?);
  // …
}
```

How to FFI

Structs

```rust
#[repr(C)]
pub struct AvroStr {
    pub data: *mut c_char,
    pub len: usize,
    pub owned: bool,
}


pub struct AvroReader;
```

```rust
#[repr(C)]
pub struct AvroStr {
    pub data: *mut c_char,
    pub len: usize,
    pub owned: bool,
}

pub struct AvroReader;
```

```rust
#[repr(C)]
pub struct AvroStr {
    pub data: *mut c_char,
    pub len: usize,
    pub owned: bool,
}


pub struct AvroReader;
```

```rust
#[repr(C)]
pub struct AvroStr {
    pub data: *mut c_char,
    pub len: usize,
    pub owned: bool,
}


pub struct AvroReader;
```

Functions

```rust
ffi_fn! {
  unsafe fn avro_schema_from_json(
    json: *const AvroStr
  ) -> Result<*mut AvroSchema> {
    let schema = Schema::parse_str((&*json).as_str())?;
    Ok(
      Box::into_raw(Box::new(schema))
      as *mut AvroSchema
    )
  }
}
```

```
ffi_fn! {
  unsafe fn avro_schema_from_json(
    json: *const AvroStr
  ) -> Result<*mut AvroSchema> {
    let schema = Schema::parse_str((&*json).as_str())?;
    Ok(
      Box::into_raw(Box::new(schema))
      as *mut AvroSchema
    )
  }
}
```

```rust
ffi_fn! {
  unsafe fn avro_schema_from_json(
    json: *const AvroStr
  ) -> Result<*mut AvroSchema> {
    let schema = Schema::parse_str((&*json).as_str())?;
    Ok(
      Box::into_raw(Box::new(schema))
      as *mut AvroSchema
    )
  }
}
```

```
ffi_fn! {
  unsafe fn avro_schema_from_json(
    json: *const AvroStr
  ) -> Result<*mut AvroSchema> {
    let schema = Schema::parse_str((&*json).as_str())?;
    Ok(
      Box::into_raw(Box::new(schema))
      as *mut AvroSchema
    )
  }
}
```

```
ffi_fn! {
  unsafe fn avro_schema_from_json(
    json: *const AvroStr
  ) -> Result<*mut AvroSchema> {
    let schema = Schema::parse_str((&*json).as_str())?;
    Ok(
      Box::into_raw(Box::new(schema))
      as *mut AvroSchema
    )
  }
}
```

Inside the macro

```rust
#[no_mangle]
pub unsafe extern "C" fn avro_schema_from_json(
  json: *const AvroStr
) -> Result<*mut AvroSchema, Error> + panic::UnwindSafe
{
  utils::safe_unwind(|| {
    let schema = Schema::parse_str((&*json).as_str())?;
    Ok(
      Box::into_raw(Box::new(schema))
      as *mut AvroSchema
    )
  })
}
```

```rust
#[no_mangle]
pub unsafe extern "C" fn avro_schema_from_json(
  json: *const AvroStr
) -> Result<*mut AvroSchema, Error> + panic::UnwindSafe
{
  utils::safe_unwind(|| {
    let schema = Schema::parse_str((&*json).as_str())?;
    Ok(
      Box::into_raw(Box::new(schema))
      as *mut AvroSchema
    )
  })
}
```

```rust
#[no_mangle]
pub unsafe extern "C" fn avro_schema_from_json(
  json: *const AvroStr
) -> Result<*mut AvroSchema, Error> + panic::UnwindSafe
{
  utils::safe_unwind(|| {
    let schema = Schema::parse_str((&*json).as_str())?;
    Ok(
      Box::into_raw(Box::new(schema))
      as *mut AvroSchema
    )
  })
}
```

```rust
#[no_mangle]
pub unsafe extern "C" fn avro_schema_from_json(
  json: *const AvroStr
) -> Result<*mut AvroSchema, Error> + panic::UnwindSafe
{
  utils::safe_unwind(|| {
    let schema = Schema::parse_str((&*json).as_str())?;
    Ok(
      Box::into_raw(Box::new(schema))
      as *mut AvroSchema
    )
  })
}
```

Gotchas

- unwind un-safety

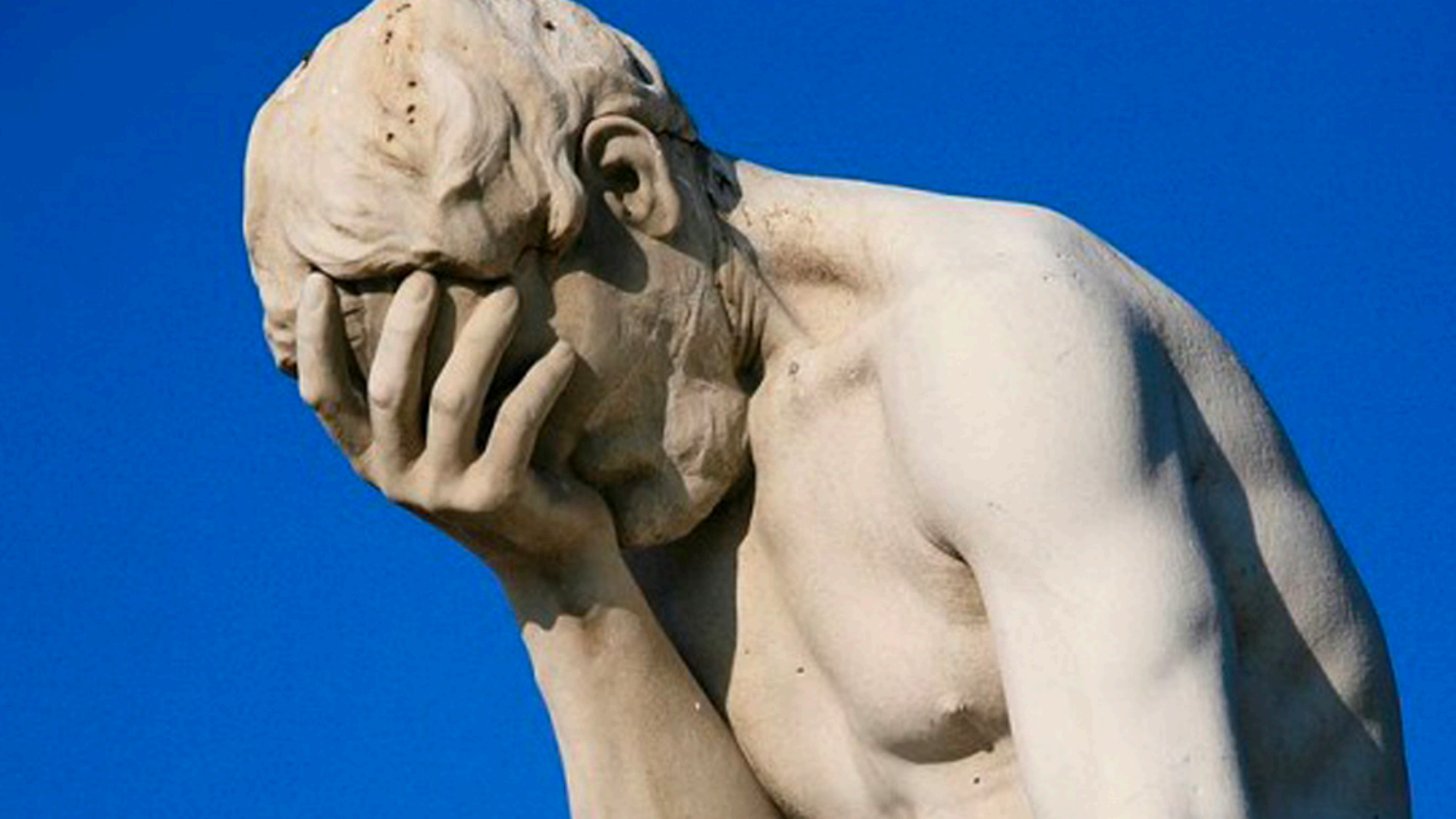
- unwind un-safety
- error codes

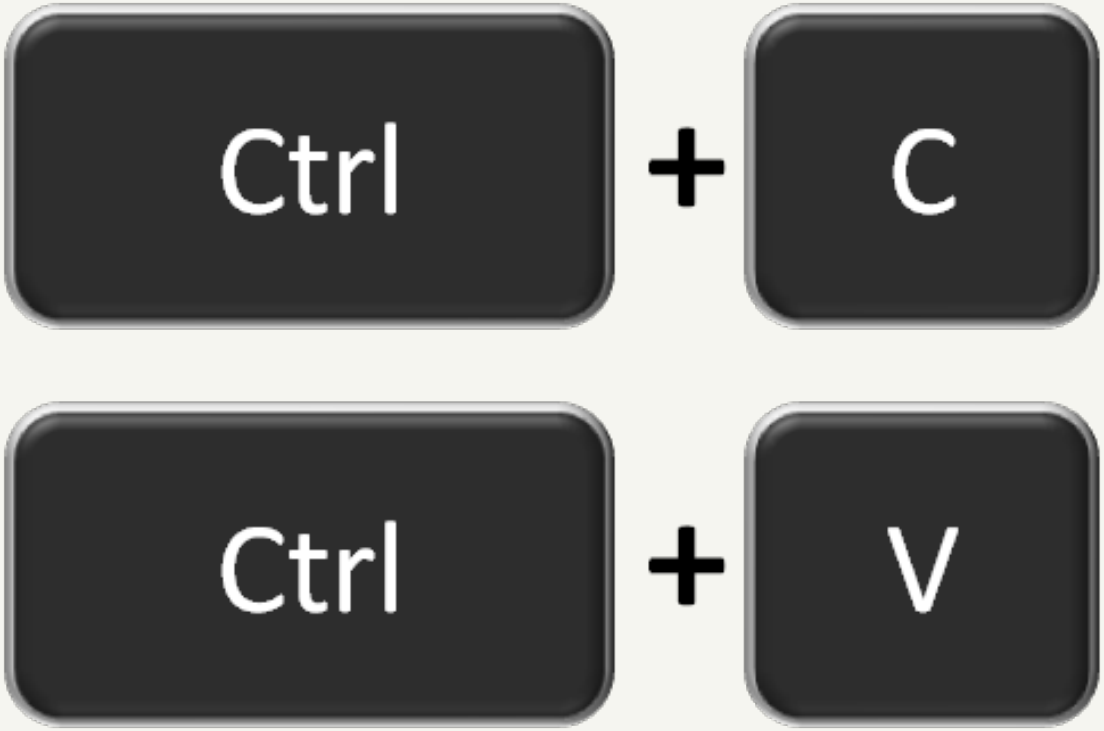
- unwind un-safety
- error codes
- explicit memory management

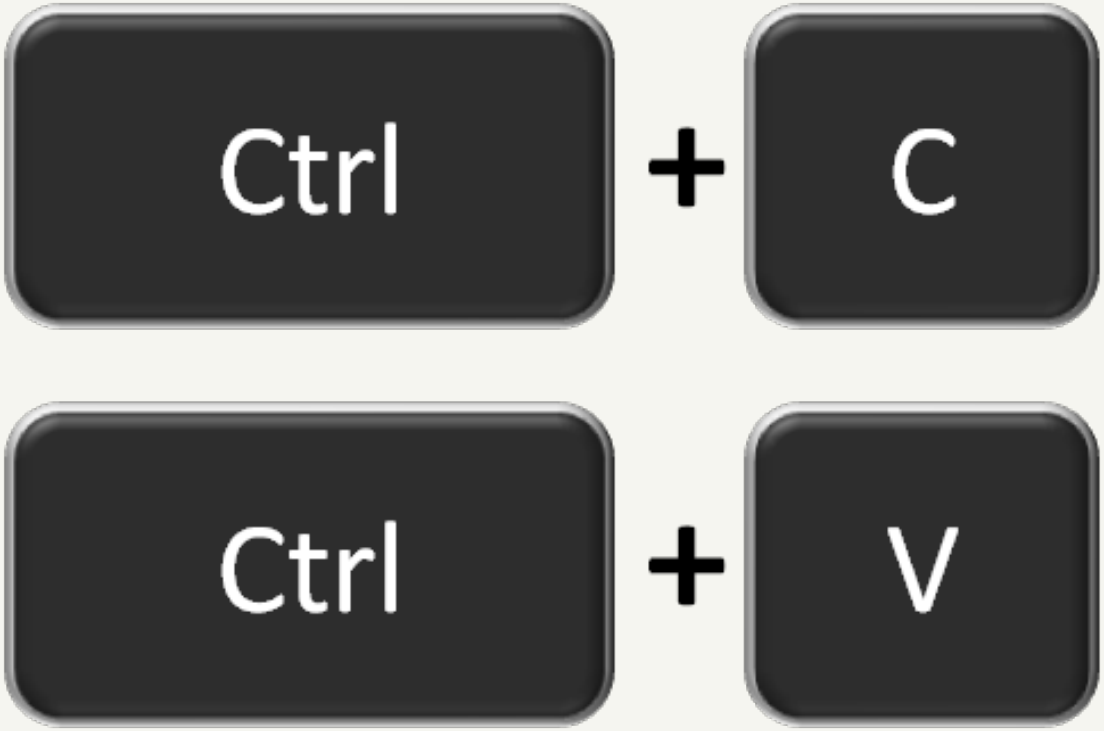
- unwind un-safety
- error codes
- explicit memory management
- enum duplication

- unwind un-safety
- error codes
- explicit memory management
- enum duplication
- complex arguments

- unwind un-safety
- error codes
- explicit memory management
- enum duplication
- complex arguments
- ...



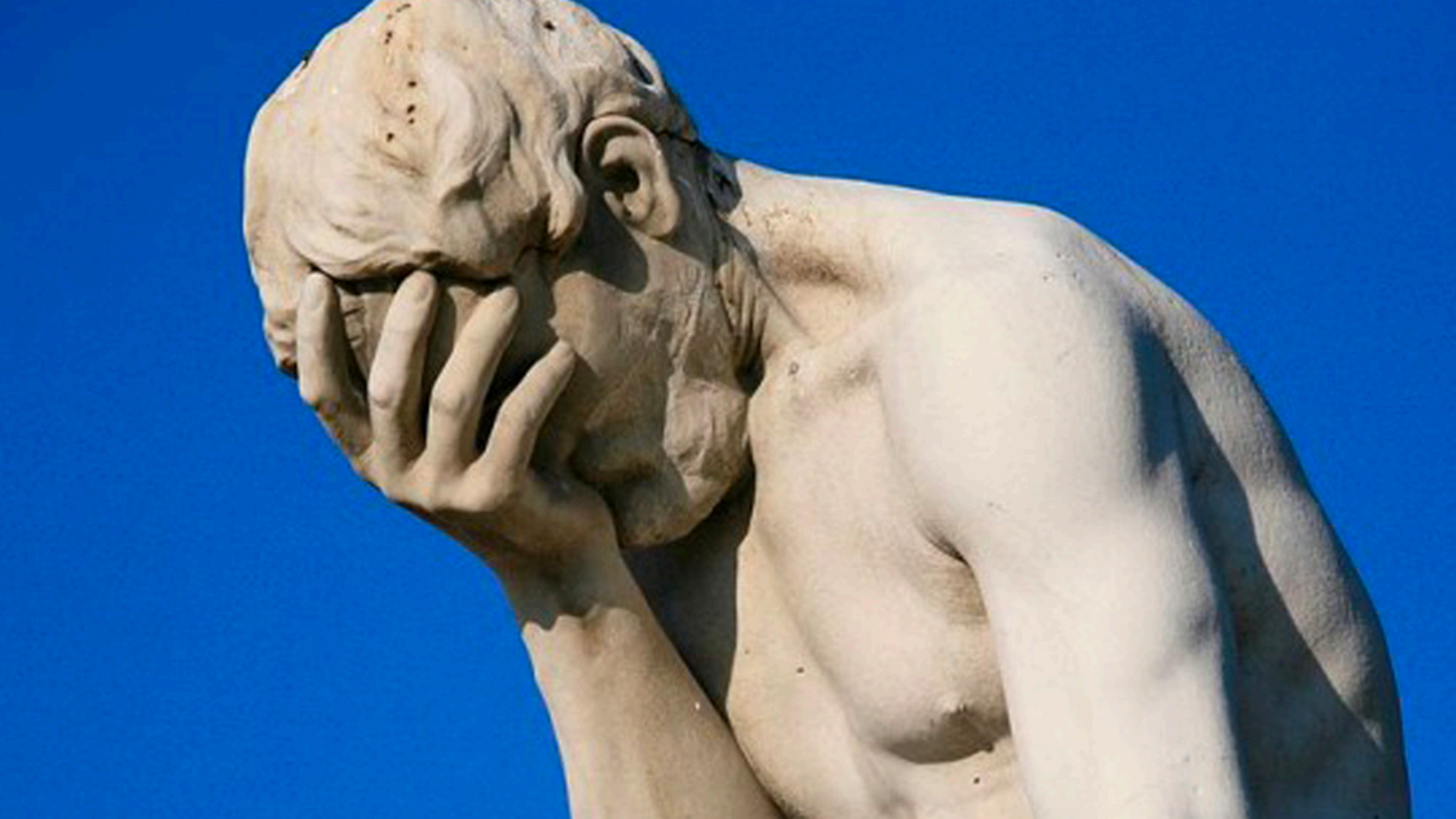

Ctrl
+
C
Ctrl
+
V



avro-rs-ffi

Header File

```
typedef struct {
  char *data;
  uintptr_t len;
  bool owned;
} AvroStr;

typedef struct AvroReader AvroReader;

AvroSchema *avro_schema_from_json(const AvroStr *json);
```



cbindgen is awesome!

```
include/avro.h: $(shell find src -type f -name "*.rs")
	RUSTUP_TOOLCHAIN=nightly \
	cbindgen -v -c cbindgen.toml . -o $@
```

Makefile

```
include/avro.h: $(shell find src -type f -name "*.rs")
	RUSTUP_TOOLCHAIN=nightly \
	cbindgen -v -c cbindgen.toml . -o $@
```

Makefile

cbindgen is awesome!

How to Python wrapper

```
from pyavro_rs._lowlevel import ffi, lib


def avro_int(n):
  return lib.avro_value_int_new(n)


def avro_list(items):
  array = lib.avro_value_array_new(len(items))
  for item in items:
    value = Value(item)
    lib.avro_array_append(array, value.value)
  return array
```

```
class Value(RustObject):
    __dealloc_func__ = lib.avro_value_free

    __TYPE_TO_AVRO = {
        NoneType: avro_null,
        bool: avro_bool,
        int: avro_int,
        float: avro_float,
        str: avro_str,
        bytes: avro_bytes,
        list: avro_list,
        tuple: avro_list,
        dict: avro_dict,
    }

    def __new__(cls, datum):
        fn = cls.__TYPE_TO_AVRO.get(type(datum))
        if fn is None:
            raise Exception('Unable to encode type {}'.format(type(datum)))
        return cls._from_objptr(fn(datum))

    @property
    def value(self):
        return self._objptr
```

```
schema = Schema('''{.....}''')

writer = Writer(schema)

writer.append({'a': 27, 'b': 'foo'})
writer.append({'a': 42, 'b': 'bar'})
writer.flush()

output = writer.into()

reader = Reader(output)

for record in reader:
    print(record)
```

```
init:
  git submodule add https://github.com/flavray/avro-rs-ffi

build-rust:
  cd avro-rs-ffi && cargo build --release

build:
  python setup.py build

wheel:
  python setup.py sdist bdist_wheel
```

Makefile

setup.py

```
setup(
    name='pyavro_rs',
    packages=find_packages(),
    include_package_data=True,
    package_data={
      'avro-rs-ffi': {
        'include/avro_rs.h',
        ‘target/release/libavro_rs_ffi.so'
      },
    },
    zip_safe=False,
    setup_requires=['cffi'],
    install_requires=['cffi'],
    cmdclass={
        'bdist_wheel': bdist_wheel,
    }
)
```

setup.py

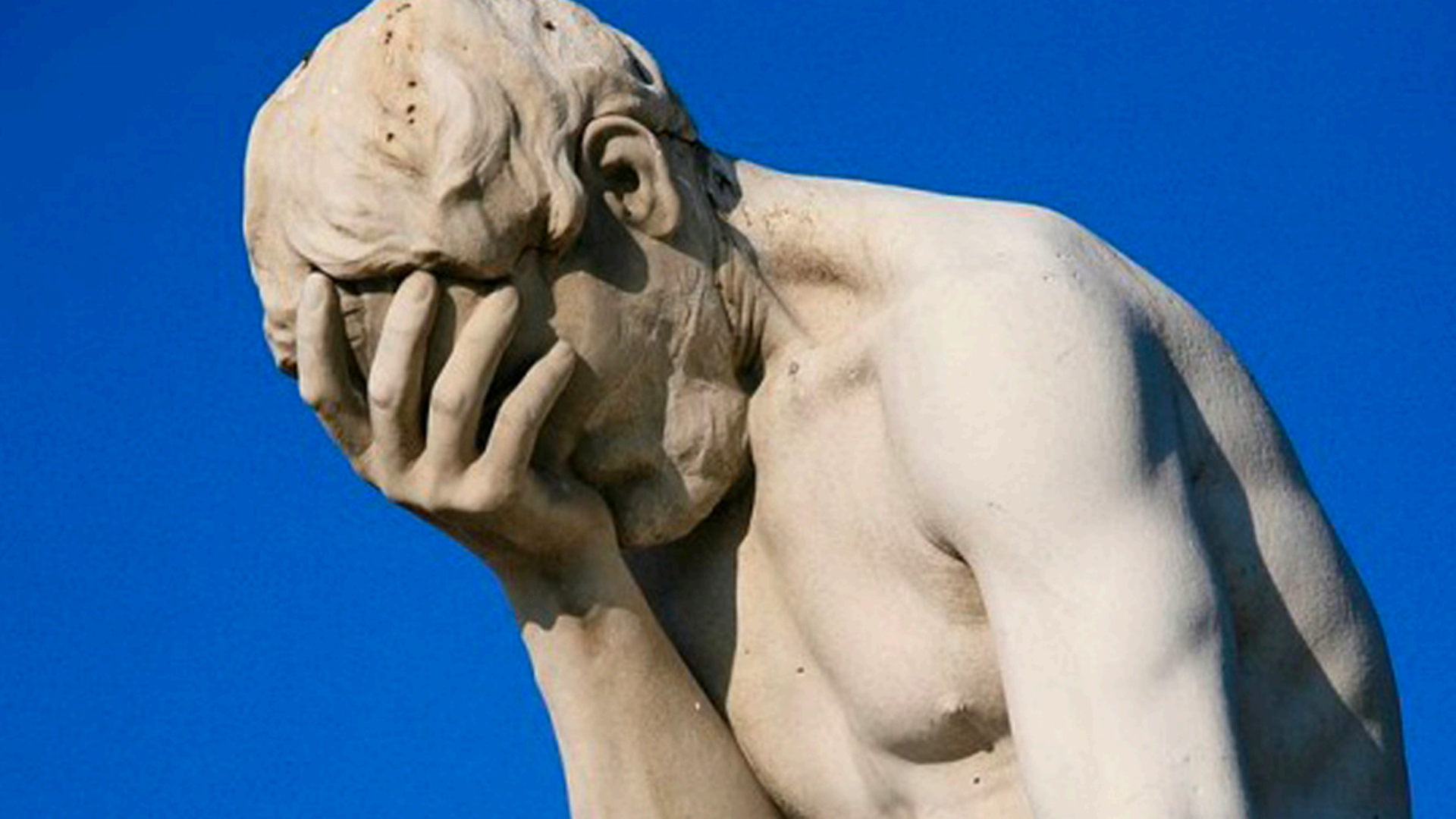
```
setup(
    name='pyavro_rs',
    packages=find_packages(),
    include_package_data=True,
    package_data={
      'avro-rs-ffi': {
        'include/avro_rs.h',
        'target/release/libavro_rs_ffi.so'
      },
    },
    zip_safe=False,
    setup_requires=['cffi'],
    install_requires=['cffi'],
    cmdclass={
        'bdist_wheel': bdist_wheel,
    }
)
```

setup.py

```
setup(
    name='pyavro_rs',
    packages=find_packages(),
    include_package_data=True,
    package_data={
      'avro-rs-ffi': {
        'include/avro_rs.h',
        ‘target/release/libavro_rs_ffi.so'
      },
    },
    zip_safe=False,
    setup_requires=['cffi'],
    install_requires=['cffi'],
    cmdclass={
        'bdist_wheel': bdist_wheel,
    }
)
```

setup.py

```
setup(
    name='pyavro_rs',
    packages=find_packages(),
    include_package_data=True,
    package_data={
      'avro-rs-ffi': {
        'include/avro_rs.h',
        ‘target/release/libavro_rs_ffi.so'
      },
    },
    zip_safe=False,
    setup_requires=['cffi'],
    install_requires=['cffi'],
    cmdclass={
        'bdist_wheel': bdist_wheel,
    }
)
```



milksnake is awesome!

setup.py

```
def build_native(spec):
    build = spec.add_external_build(
        cmd=['cargo', 'build', '--release'],
        path='./avro-rs-ffi'
    )
    spec.add_cffi_module(
        module_path='pyavro_rs._lowlevel',
        dylib=lambda: build.find_dylib(
            'avro_rs_ffi',
            in_path='target/release'
        ),
        header_filename=lambda: build.find_header(
            'avro_rs.h',
            in_path='include'
        ),
    )
```

setup.py

```
def build_native(spec):
    build = spec.add_external_build(
        cmd=['cargo', 'build', '--release'],
        path='./avro-rs-ffi'
    )
    spec.add_cffi_module(
        module_path='pyavro_rs._lowlevel',
        dylib=lambda: build.find_dylib(
            'avro_rs_ffi',
            in_path='target/release'
        ),
        header_filename=lambda: build.find_header(
            'avro_rs.h',
            in_path='include'
        ),
    )
```

setup.py

```
def build_native(spec):
    build = spec.add_external_build(
        cmd=['cargo', 'build', '--release'],
        path='./avro-rs-ffi'
    )
    spec.add_cffi_module(
        module_path='pyavro_rs._lowlevel',
        dylib=lambda: build.find_dylib(
            'avro_rs_ffi',
            in_path='target/release'
        ),
        header_filename=lambda: build.find_header(
            'avro_rs.h',
            in_path='include'
        ),
    )
```

setup.py

```
def build_native(spec):
    build = spec.add_external_build(
        cmd=['cargo', 'build', '--release'],
        path='./avro-rs-ffi'
    )
    spec.add_cffi_module(
        module_path='pyavro_rs._lowlevel',
        dylib=lambda: build.find_dylib(
            'avro_rs_ffi',
            in_path='target/release'
        ),
        header_filename=lambda: build.find_header(
            'avro_rs.h',
            in_path='include'
        ),
    )
```

setup.py

```
setup(
    name='pyavro_rs',
    packages=find_packages(),
    include_package_data=True,
    setup_requires=['milksnake'],
    install_requires=['milksnake'],
    milksnake_tasks=[build_native],
)
```

setup.py

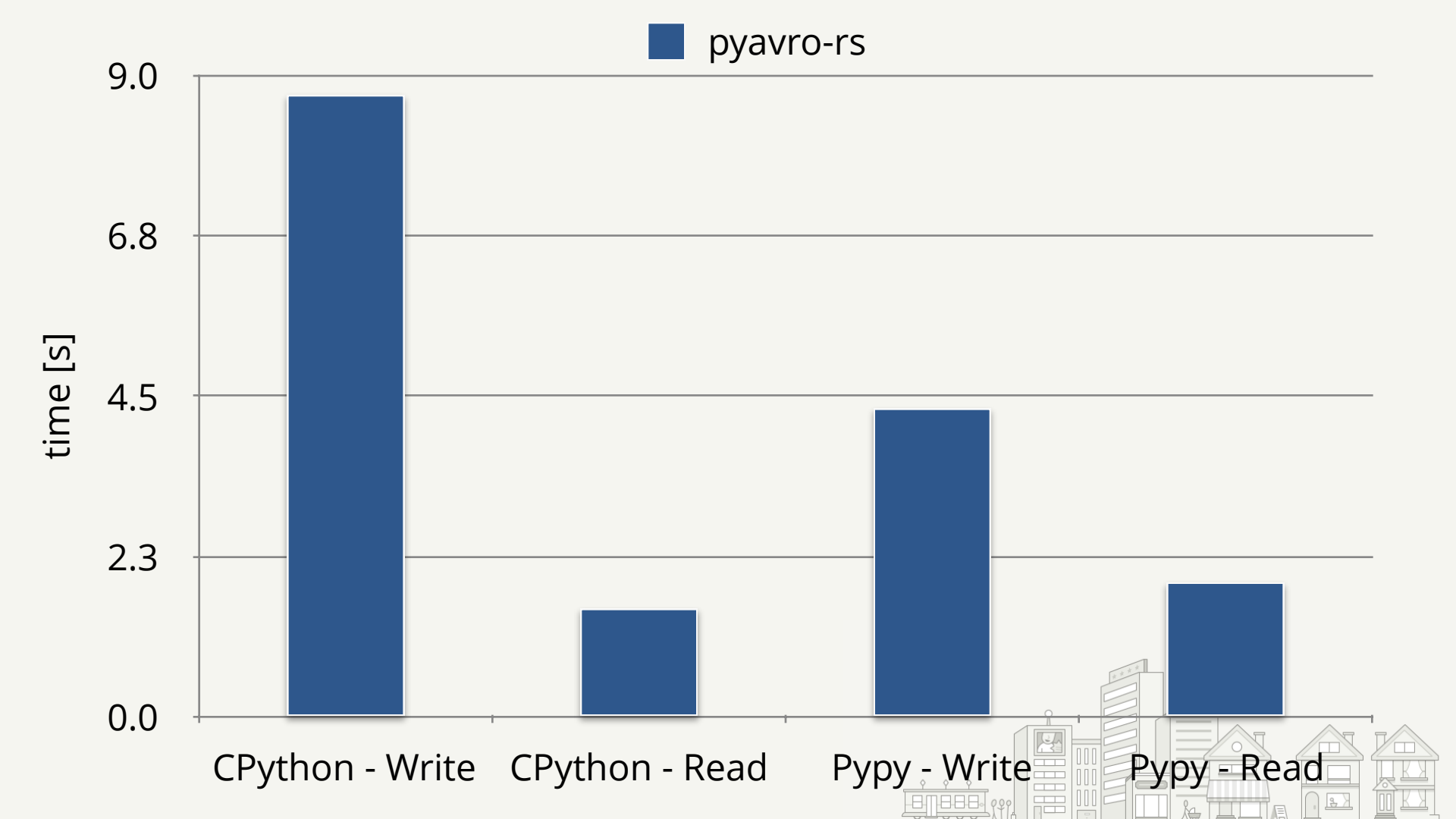
```
setup(
    name='pyavro_rs',
    packages=find_packages(),
    include_package_data=True,
    setup_requires=['milksnake'],
    install_requires=['milksnake'],
    milksnake_tasks=[build_native],
)
```

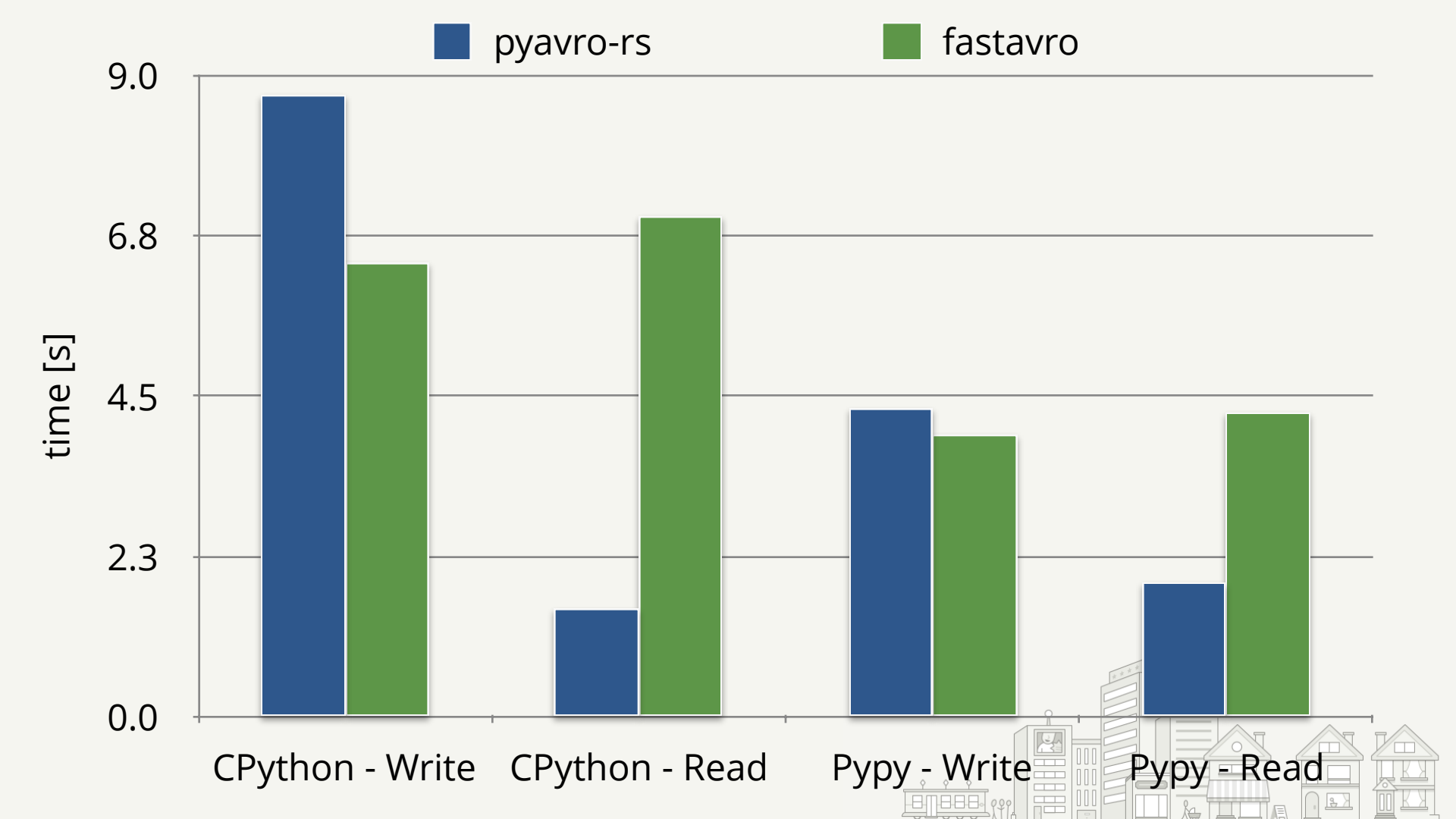
```
>> python setup.py build
running build
running build_py
…
    Compiling avro-rs v0.4.1
    Compiling avro-rs-ffi v0.0.1
      Finished release [optimized] target(s) in 115.66 secs
…
```

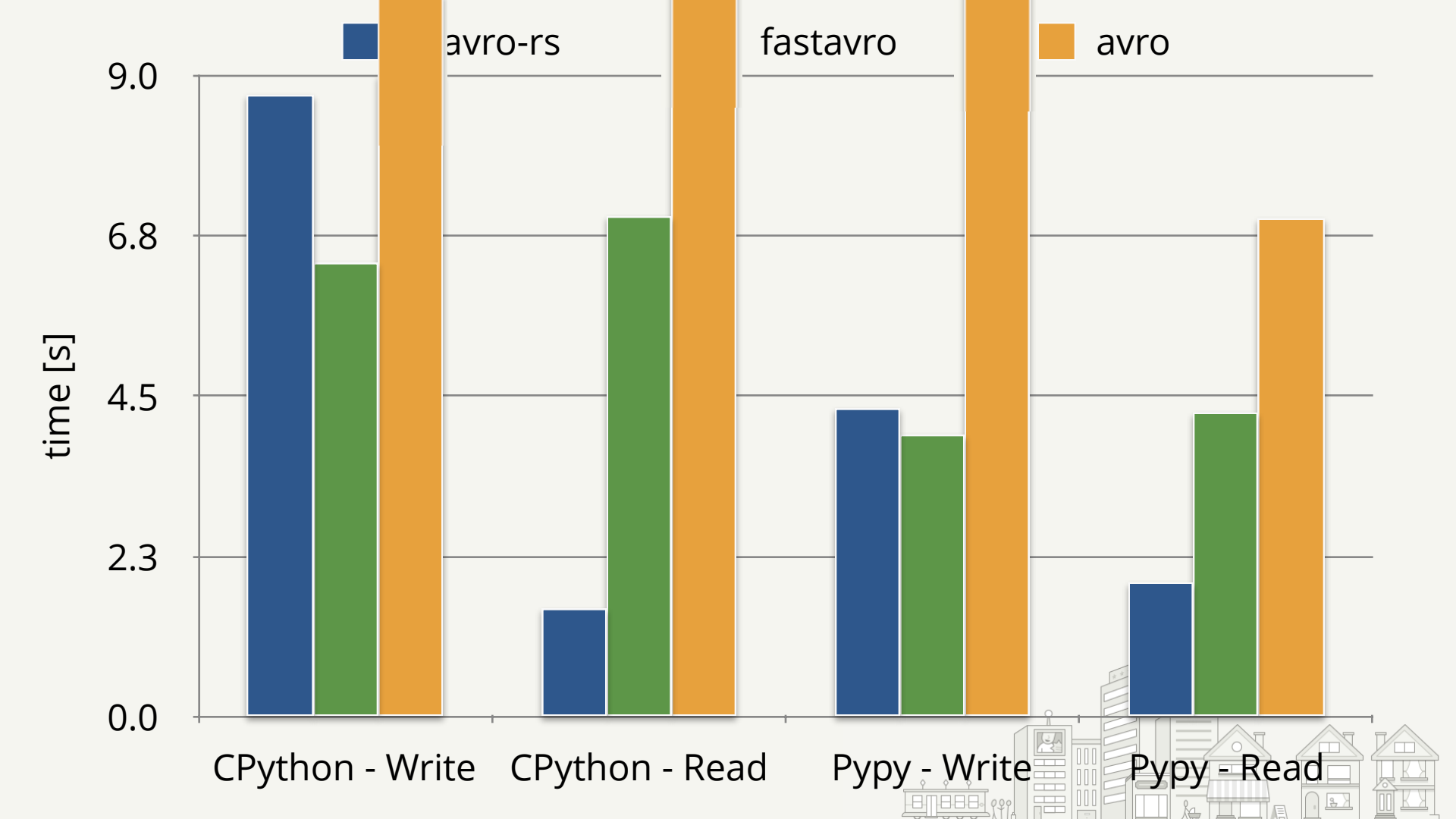
```
>> python setup.py build
running build
running build_py
…
    Compiling avro-rs v0.4.1
    Compiling avro-rs-ffi v0.0.1
      Finished release [optimized] target(s) in 115.66 secs
…

>> tree build/
build
└── lib
    └── pyavro_rs
        ├── __init__.py
        ├── _lowlevel.py
        ├── _lowlevel__ffi.py
        └── _lowlevel__lib.so
```

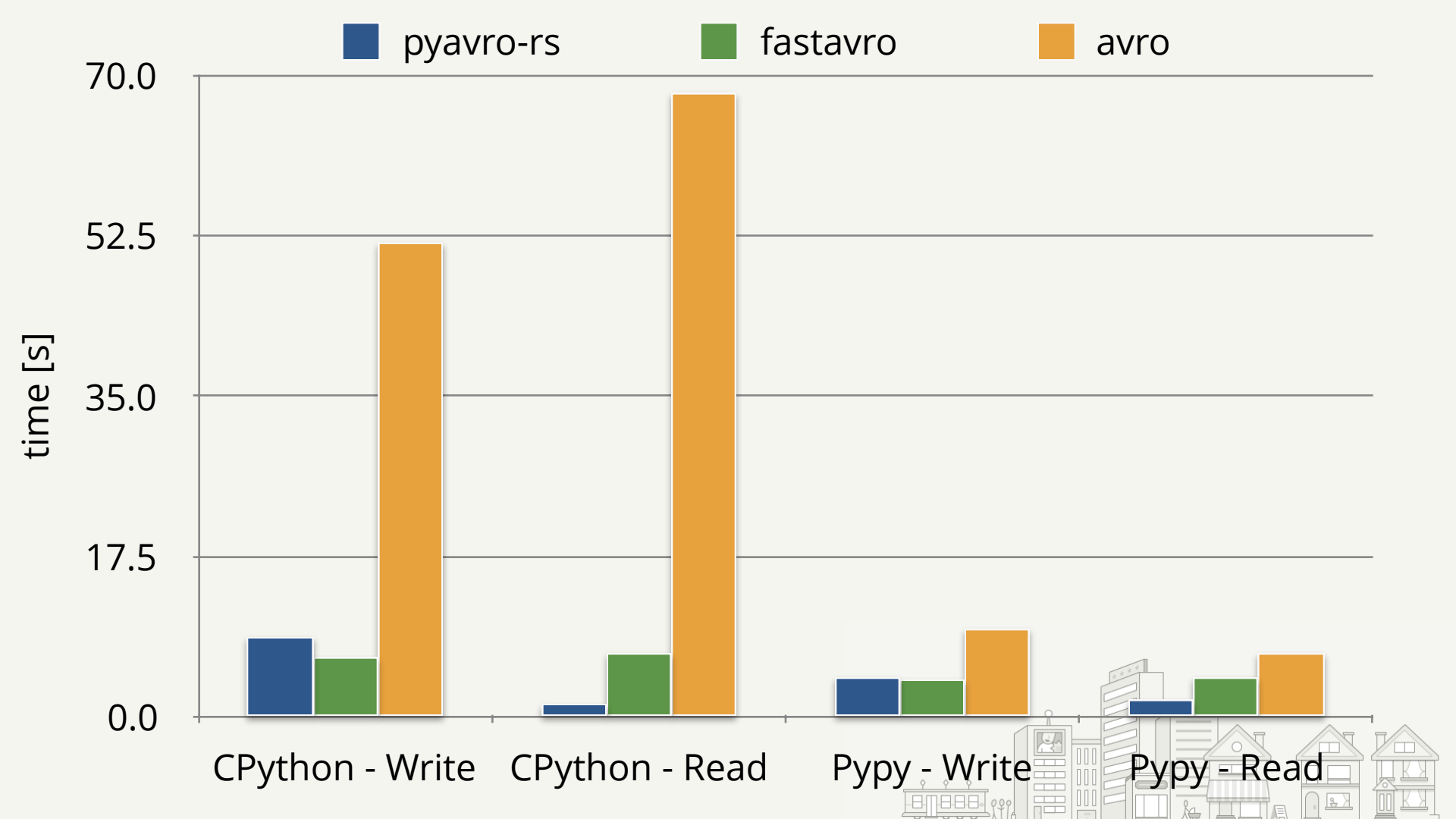
milksnake is awesome!

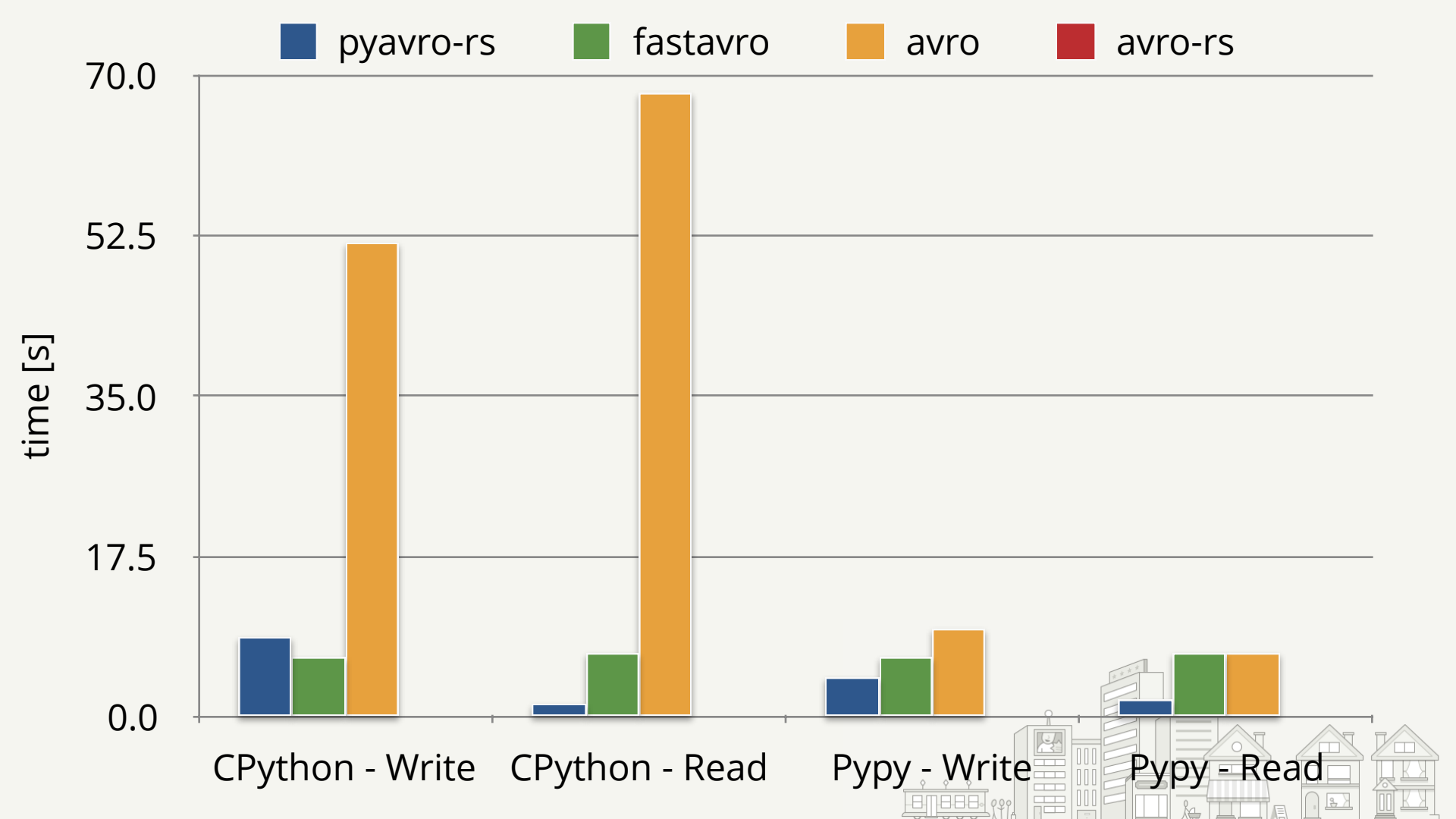
Was it faster?

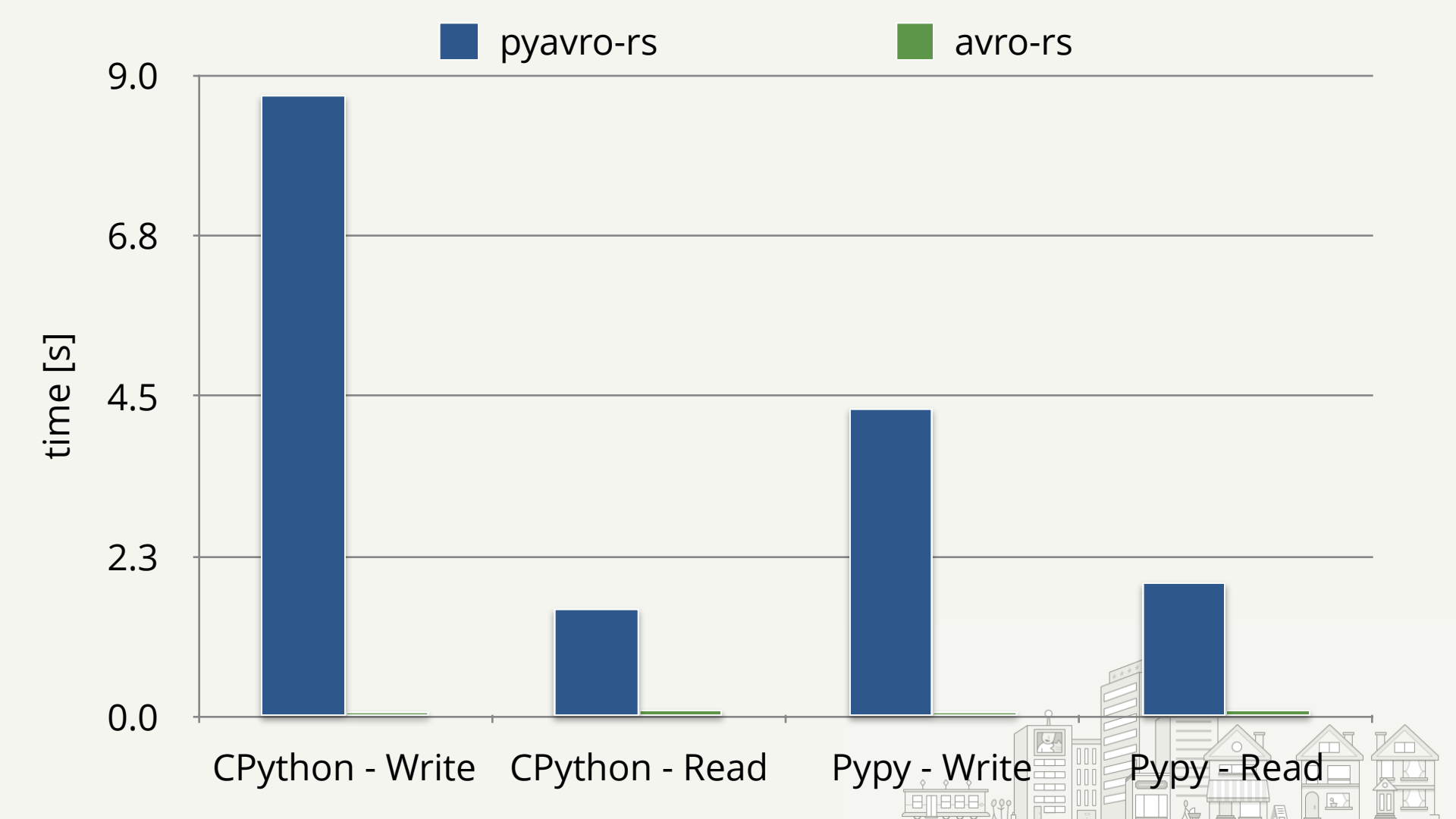

pyavro-rs
9.0
6.8
4.5
2.3
0.0
time [s]
CPython - Write
CPython - Read
Pypy - Write
Pypy - Read


pyavro-rs
fastavro
9.0
6.8
4.5
2.3
0.0
time [s]
CPython - Write
CPython - Read
Pypy - Write
Pypy - Read


avro-rs
fastavro
avro
time [s]
9.0
6.8
4.5
2.3
0.0
CPython - Write
CPython - Read
Pypy - Write
Pypy - Read


pyavro-rs
fastavro
avro
70.0
52.5
35.0
17.5
0.0
time [s]
CPython - Write
CPython - Read
Pypy - Write
Pypy - Read


pyavro-rs
fastavro
avro
avro-rs
70.0
52.5
35.0
17.5
0.0
time [s]
CPython - Write
CPython - Read
Pypy - Write
Pypy - Read


pyavro-rs
avro-rs
9.0
6.8
4.5
2.3
0.0
time [s]
CPython - Write
CPython - Read
Pypy - Write
Pypy - Read

How to get away with it

How to convince my colleagues?

How to convince my colleagues?

- Most Loved Language (78.9%)

  StackOverflow Survey 2017

How to convince my colleagues?

- Most Loved Language (78.9%)

  StackOverflow Survey 2017

- compilation just works (cargo)

How to convince my colleagues?

- Most Loved Language (78.9%)

  StackOverflow Survey 2017

- compilation just works (cargo)
- fast release cycle

How to convince my colleagues?

- Most Loved Language (78.9%)

  StackOverflow Survey 2017

- compilation just works (cargo)
- fast release cycle
- it's a ton of fun!

How to convince my company?

How to convince my company?

- wheels = compile once

How to convince my company?

- wheels = compile once
- FFI interoperability (Java, Objective-C, C++, etc)

How to convince my company?

- wheels = compile once
- FFI interoperability (Java, Objective-C, C++, etc)
- as fast as C = cheap to run

How to convince my company?

- wheels = compile once
- FFI interoperability (Java, Objective-C, C++, etc)
- as fast as C = cheap to run
- safer than C = cheap to maintain

How to convince my company?

- wheels = compile once
- FFI interoperability (Java, Objective-C, C++, etc)
- as fast as C = cheap to run
- safer than C = cheap to maintain
- used in production (Yelp, Mozilla, Dropbox, etc)

What to bring home

Write  instead of C

cbindgen is awesome!

milksnake is awesome!

Links

- avro-rs
- avro-rs-ffi
- pyavro-rs
- cbindgen
- milksnake


We're Hiring!
www.yelp.com/careers/

fb.com/YelpEngineers

@YelpEngineering

engineeringblog.yelp.com

github.com/yelp