My Python is Rusting

— A Python and Rust Love Story —

Armin @mitsuhiko Ronacher

Hi, I'm Armin

… and I do Open Source, lots of Python and SaaS

Flask
Sentry
…

... and here
is where you
can find me

twitter.com/@mitsuhiko
github.com/mitsuhiko
lucumr.pocoo.org/

Flask

web development,
one drop at a time


SENTRY

“so I heard you are doing Rust now …”

that's true but ...

we love python

STRONG ECOSYSTEM

Fast Iteration

Stable Environment

Powerful Metaprogramming

Fast Interpreter Introspection

and rust?


SPEED

Functionality

Reliability

what we use it for

Mach-O / Dwarf Parsing

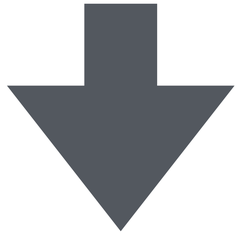
Javascript Source Maps

Proguard Mappings

Command Line Tools

one to the other

virtualenv & pip & distutils & setuptools



rustup & cargo

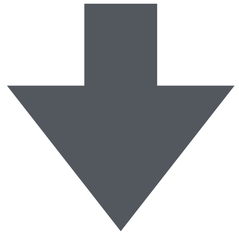
rustup

the rust toolchain manager

cargo

the rust package manager

pydoc & sphinx



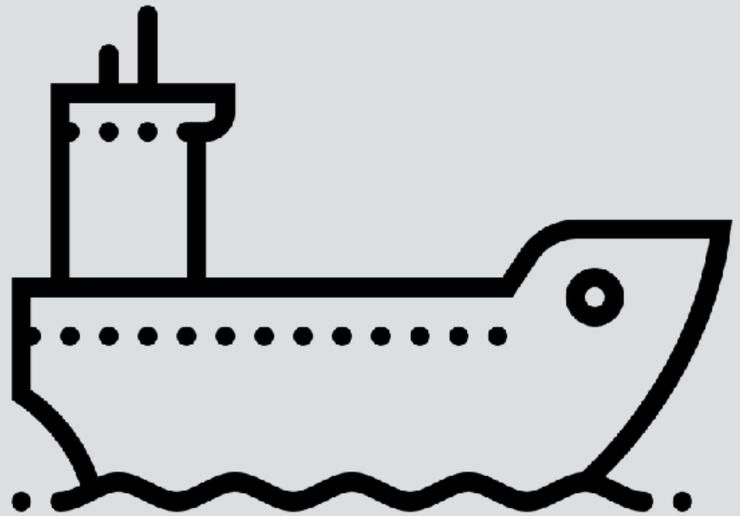
rustdoc

rustdoc

the rust documentation builder

a rust primer

code ahead



```rust
fn main() {
    println!("Hello World!");
}
```

```rust
use std::io::{stdin, BufRead, BufReader};
use std::collections::HashMap;

fn main() {
    let mut counts = HashMap::new();

    for line_rv in BufReader::new(stdin()).lines() {
        let line = line_rv.unwrap();
        *counts.entry(line).or_insert(0) += 1;
    }

    let mut items: Vec<_> = counts.into_iter().collect();
    items.sort_by_key(|&(_, count)| -count);

    for (item, count) in items.into_iter().take(10) {
        println!("{}: {}", item, count);
    }
}
```

```rust
use std::io::{stdin, BufRead, BufReader};
use std::collections::HashMap;

fn main() {
    let mut counts = HashMap::new();

    for line_rv in BufReader::new(stdin()).lines() {
        let line = line_rv.unwrap();
        *counts.entry(line).or_insert(0) += 1;
    }

    let mut items: Vec<_> = counts.into_iter().collect();
    items.sort_by_key(|&(_, count)| -count);

    for (item, count) in counts {
        println!("{}: {}", item, count);
    }
}
```

```
error[E0382]: use of moved value: `counts`
  --> test.rs:16:26
   |
13 |     let mut items: Vec<_> = counts.into_iter().collect();
   |                             ------ value moved here
...
16 |     for (item, count) in counts {
   |                          ^^^^^^ value used here after move
   |
   = note: move occurs because `counts` has type
`std::collections::HashMap<std::string::String, i32>`,
  which does not implement the `Copy` trait
```

```rust
use std::fmt;

struct User {
    id: i64,
    name: String,
}

impl fmt::Display for User {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        write!(f, "<User {}: {:?}>", self.id, self.name)
    }
}

fn main() {
    println!("{}", User { id: 42, name: "Peter".to_string() });
}
```

the zen of python

Beautiful is better than ugly.

```rust
#[derive(Serialize, Deserialize, Debug)]
pub struct Deploy {
    #[serde(rename="environment")]
    pub env: String,
    pub name: Option<String>,
    pub url: Option<String>,
}

impl Deploy {

    pub fn list(&self, api: &Api, id: u64) -> ApiResult<Vec<Deploy>> {
        api.get(&format!("/deploys/{}/", id))?.convert()
    }
}
```

Explicit is better than implicit.

```
fn parse_rev_range(rng: &str) -> (Option<&str>, &str) {
    if rng == "" {
        return (None, "HEAD".into());
    }
    let mut iter = rng.rsplitn(2, "..");
    let rev = iter.next().unwrap_or("HEAD");
    (iter.next(), rev)
}
```

Simple is better than complex.

```rust
use std::{fs, env, io};

let here = env::current_dir()?;
for dent_rv in fs::read_dir(here)? {
    let dent = dent_rv?;
    let md = dent.metadata()?;
    println!("{: <60}{: <12}{}",
             dent.path().display(),
             md.len(),
             if md.is_file() { "file" } else { "dir" });
}
```

Complex is better than complicated.

```rust
use redis::{Client, PipelineCommands, pipe};

let client = Client::open("redis://127.0.0.1/")?;
let con = client.get_connection()?;
let (k1, k2) : (i32, i32) = pipe()
    .atomic()
    .set("key_1", 42).ignore()
    .set("key_2", 43).ignore()
    .get("key_1")
    .get("key_2").query(&con)?;
```

Errors should never pass silently.

```
use std::fs;

fn main() {
    fs::File::open("/tmp/test.txt");
}
```

```
$ rustc test.rs
warning: unused result which must be used,
#[warn(unused_must_use)] on by default
 --> test.rs:4:5
  |
4 |     fs::File::open("/tmp/test.txt");
  |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

side by side

protocols vs traits

`__del__`	`Drop::drop`
`__add__`	`Add::add`
`__str__`	`Display::fmt`
`__repr__`	`Debug::fmt`
`__getitem__`	`Index::index`

error causing

```
throw …
```

```
panic!(…)
return Err(…);
```

error conversion

```
try:
    x = foo()
except SomeError as e:
    raise NewError(e)
```

```
let x = foo()?
```

so you want to try it

[Intermission]

monolithic / SOA

this is absurd

modular code

+

same process

marriage

you declared your compatibility and now I

cffi you Python and Rust

Rust Library

> Rust CABI + C header

> CFFI

> Python

cargo | cffi | wheel | setuptools

repeat after me:

GO AWAY LIBPYTHON

do start threads
and thread pools

don't pass complex data structures around

do wrap some rust objects in python

don't move complex logic from Python to Rust

do use docker for builds

setuptools

please halp

building

Pillow-4.0.0-cp36-cp36m-manylinux1_x86_64.whl

Python 2 builds:

Versions: 2.7
ABI: cpm + cpmu
Platforms: OS X + 2 Linux

Total: 1 × 2 × 3 = **6**

Python 3 builds:

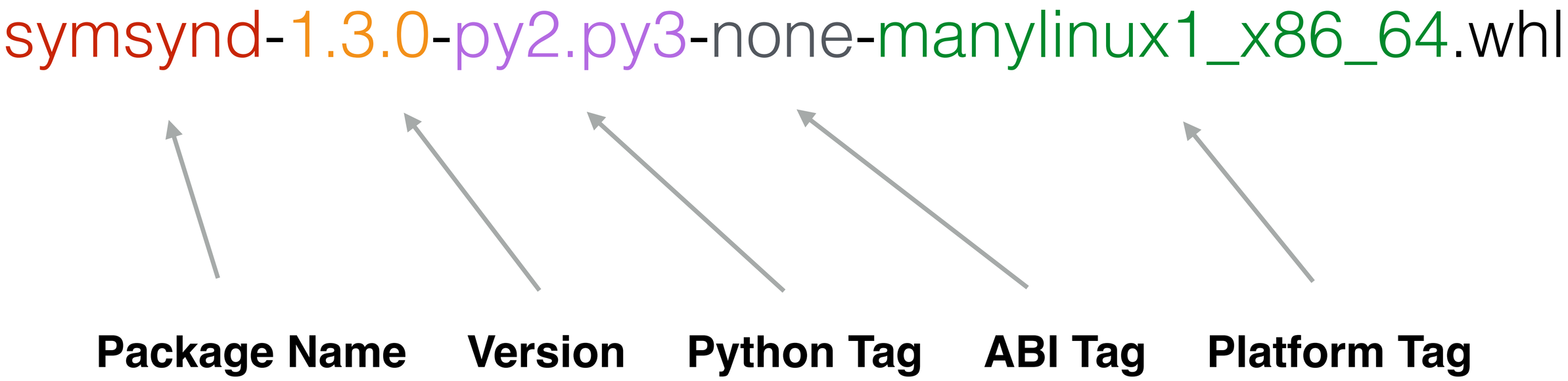
Versions: 3.3 + 3.4 + 3.5 + 3.6 + 3.7
ABI: cpm
Platforms: OS X + 2 Linux

Total: 5 × 1 × 3 = **15**

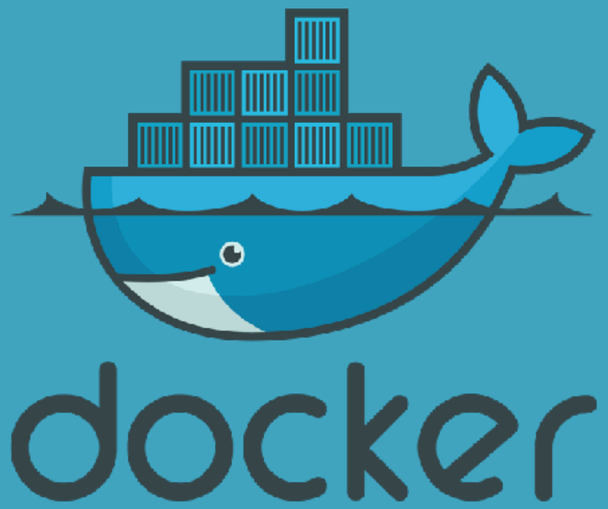
21 BUILDS!!!

path to success:

- do not link to libpython
- use cffi
- 2.x/3.x compatible sources
- fuck around with setuptools


symsynd-1.3.0-py2.py3-none-manylinux1_x86_64.whl
Package Name
Version
Python Tag
ABI Tag
Platform Tag

3 builds!


docker

useful images

quay.io/pypa/manylinux1_i686
quay.io/pypa/manylinux1_x86_64

- ★ It's an ancient CentOS (*for instance it has no SNI Support*)
- ★ 32bit builds on on 64bit Docker typically. Use the `linux32` command
- ★ Dockerfile allows you to "cache" steps

the bridge

```rust
use std::mem;
use std::panic;

fn silent_panic_handler(_pi: &panic::PanicInfo) {
    /* don't do anything here */
}

#[no_mangle]
pub unsafe extern "C" fn mylib_init() {
    panic::set_hook(Box::new(silent_panic_handler));
}
```

```rust
unsafe fn set_err(err: Error, err_out: *mut CError) {
    if err_out.is_null() {
        return;
    }
    let s = format!("{}\x00", err);
    (*err_out).message = Box::into_raw(s.into_boxed_str()) as *mut u8;
    (*err_out).code = err.get_error_code();
    (*err_out).failed = 1;
}
```

```rust
unsafe fn landingpad<F: FnOnce() -> Result<T> + panic::UnwindSafe, T>(
    f: F, err_out: *mut CError) -> T
{
    if let Ok(rv) = panic::catch_unwind(f) {
        rv.map_err(|err| set_err(err, err_out)).unwrap_or(mem::zeroed())
    } else {
        set_err(ErrorKind::InternalError.into(), err_out);
        mem::zeroed()
    }
}
```

```
macro_rules! export (
    ($n:ident($($an:ident: $aty:ty),*) -> Result<$rv:ty> $body:block) => (
        #[no_mangle]
        pub unsafe extern "C" fn $n($($an: $aty,)* err: *mut CError) -> $rv
        {
            landingpad(|| $body, err)
        }
    );
);
```

```
export!(lsm_view_dump_memdb(
    view: *mut View, len_out: *mut c_uint, with_source_contents: c_int,
    with_names: c_int) -> Result<*mut u8>
{
    let memdb = (*view).dump_memdb(DumpOptions {
        with_source_contents: with_source_contents != 0,
        with_names: with_names != 0,
    })?;
    *len_out = memdb.len() as c_uint;
    Ok(Box::into_raw(memdb.into_boxed_slice()) as *mut u8)
});
```

```c
typedef void lsm_view_t;
typedef struct lsm_error_s {
    char *message;
    int failed;
    int code;
} lsm_error_t;

char *lsm_view_dump_memdb(const lsm_view_t *view,
                          unsigned int *len_out,
                          int with_source_contents,
                          int with_names,
                          lsm_error_t *err);
```

```
def rustcall(func, *args):
    err = _ffi.new('lsm_error_t *')
    rv = func(*(args + (err,)))
    if not err[0].failed:
        return rv
    try:
        cls = special_errors.get(err[0].code, SourceMapError)
        exc = cls(_ffi.string(err[0].message).decode('utf-8', 'replace'))
    finally:
        _lib.lsm_buffer_free(err[0].message)
    raise exc
```

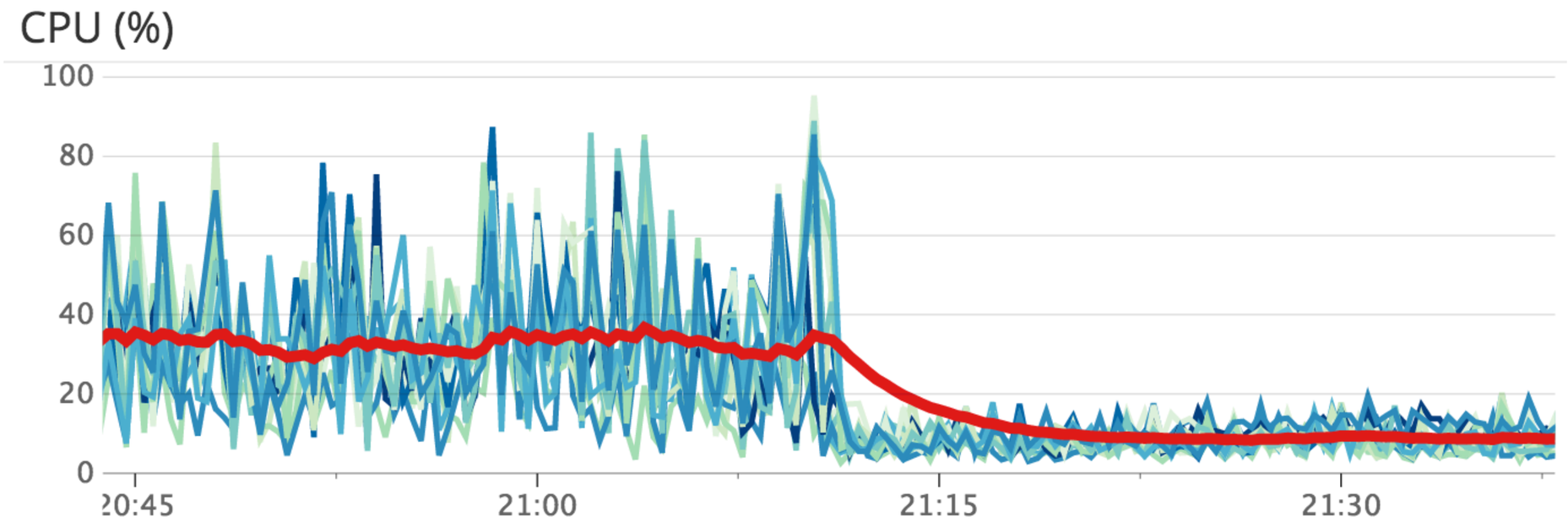
what is missing

better bdist_wheel

Reusable Docker Setups

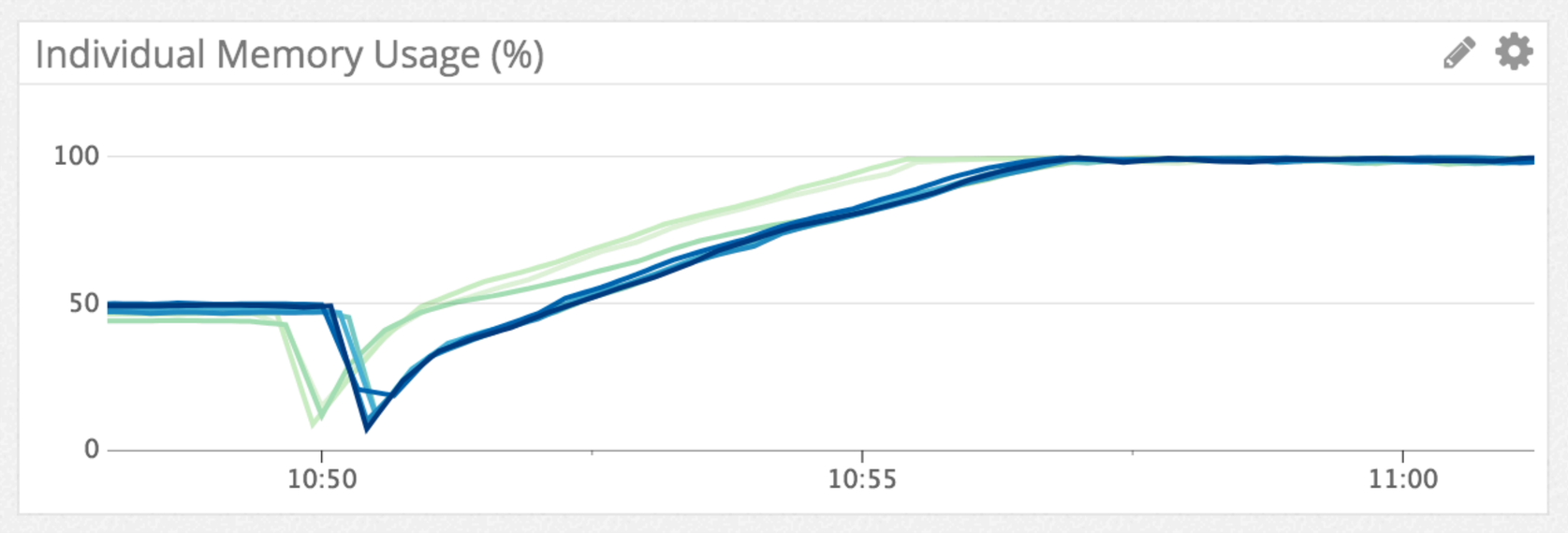
RustFFI + Shims

results



(python vs rust source map handling)

and when shit goes wrong



(bug in the binding caused memory leak)

it's great, but we need better tooling

Q&A