Why is python slow

Maciej Fijałkowski

Hello everyone, my name is Maciej Fijalkowski

Мачей Фиялковский

... which can be also spelled like that, if you insist

you might know me as a pypy guy, which I've been doing for around a decade now. These days I'm also running a consulting company that tries to drive pypy funding

open source and funding

open source funding is a complicated topic that I can give a whole talk about - if you're interested look for my talk from pycon APAC last year, where I explained in detail why it is hard, despite the abundance of money in the industry

cffi

to give you just a quick glimpse, while pypy managed to make enough money to sustain few people working on it full time, cffi, which is by far the most successful piece of software we wrote so far, managed to only get $5k funding from Python software foundation. The incentives are strange, by promoting almost-ready software where you can sell consulting

strengths & weaknesses

Today I'm going to talk about strengths and weaknesses of the python as a language and python as a culture with special focus on performance, since this is the topic we deal with at pypy

limited technical details

I'll try to keep the talk to minimal technical details. There is a far more detailed PyPy-related talk after that one and there is plenty of material online. Feel free to ask me everything technical after the talk. Despite the outline here, I am a pretty technical person.

python is a good language

Python is a good language with a great ecosystem. The main strength lies with a big and diverse set of libraries for doing pretty much anything. Of course it does not come from nowhere - python ease of use and ease of integration with C has been the driving factor here.

web
scientific computing

Python seems to occupy two main niches - server side web, understood broadly, and scientific computing. Richness of the libraries is a primary driving factor for both

no client side :(

I hate javascript. I moderately dislike C#. And yet those are the languages I have to deal with on a day to day basis if I do *any* sort of client side development - python on the web (client side) is unusable, while things like unity totally take over client side for VR etc.

https://
blog.sourced.tech/post/
language_migrations/

I would normally try to insert graphs, but it's impossible on that projector - this is the translation matrix and a lot of maths that's way above my head that explains more or less transitions from and to python. Good read.

Migration to python: R, Fortran, C++, Perl, PHP

The key findings - that if you use some substandard web or numerics language, python is a natural transition point should not come as a surprise to anyone. Well done us, pat on the back

we migrated our key services to {java,c,c++,go,rust}

the key problem that I'm going to talk about today is the above sentence. We are loosing key customers that need to do something fast. Surprisingly, from the findings, but not really, is the fact that transitioning to java is more common than go

is python in danger?

I don't think python is in danger (just yet). But if we fail to acknowledge the problems presented, we'll be loosing the most prominent big python users.

bad runtime

The current status is the following - we have one of the worst runtimes in terms of performance within languages that people are actually considering when thinking of starting a new project. Noone sane would start a new project in PHP, if they can help it.

example: java

It's not just runtime. Let's get an example - java programming language

jvm is *really* good

JVM, more than one, is REALLY REALLY good. There is a tremendous amount of man years put into all of the prevalent JVMs.

who thinks java is fast?

So, who thinks java is fast? see? However, it's not the runtime that's the problem. Is the variety of libraries that are both poorly written and without any expectations of what does it actually mean for performance. You can totally write Java code that runs fast (but maybe not starts fast)

culture and how you write software

Culture is important. Vast majority of C world thinks fast is good, so the libraries are fast. C also makes it hard to do expensive things, like allocations, so you try to avoid it as much as possible.

efficient idioms

A good language is a language that's easy to write software that's not error prone. I would like to consider myself moderately competent and yet I can't write javascript or php software that's not virtually littered with bugs. A great language is one where it's also harder to write software that's slow. Python makes it too easy in places.

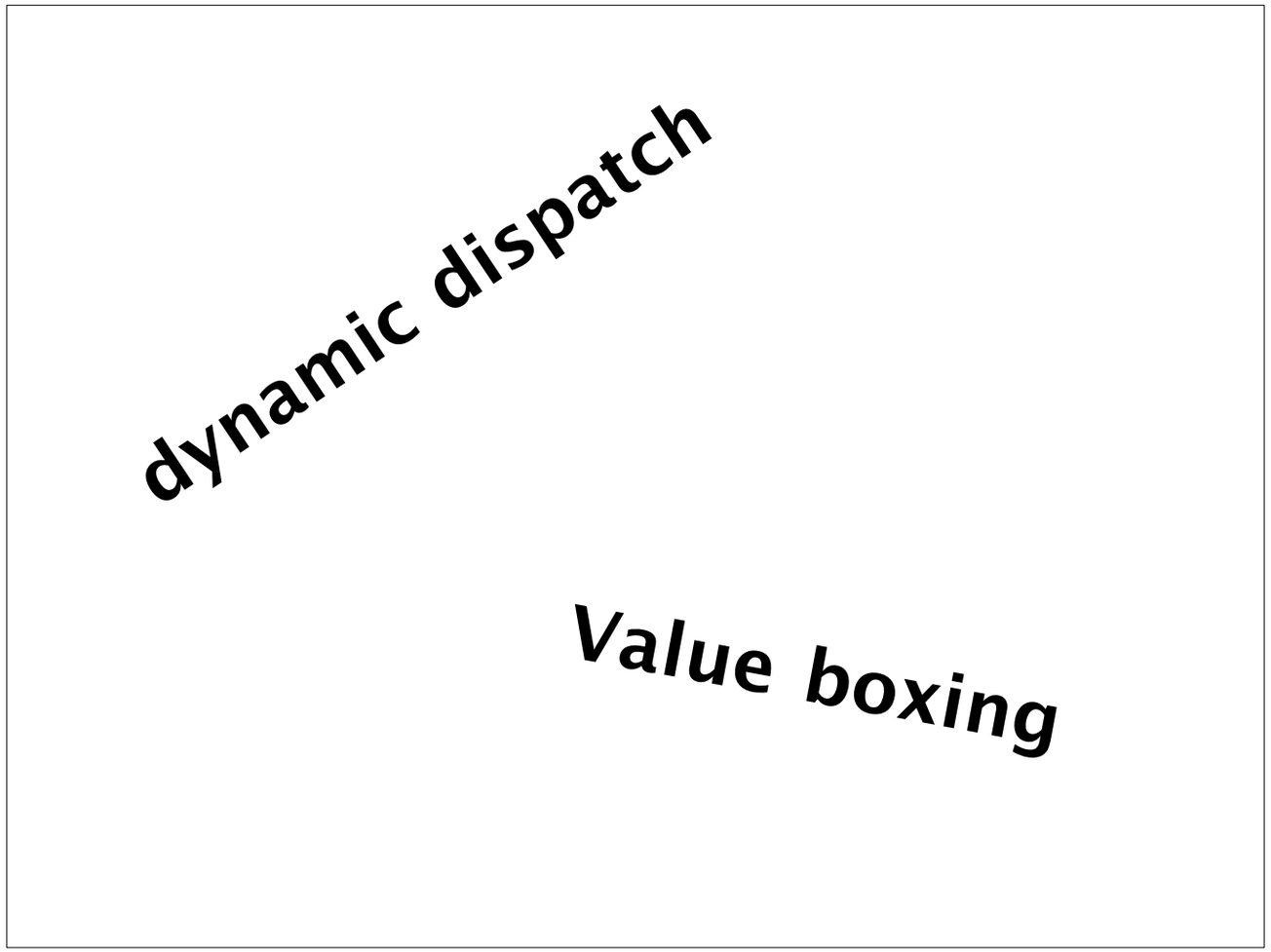
runtime matters

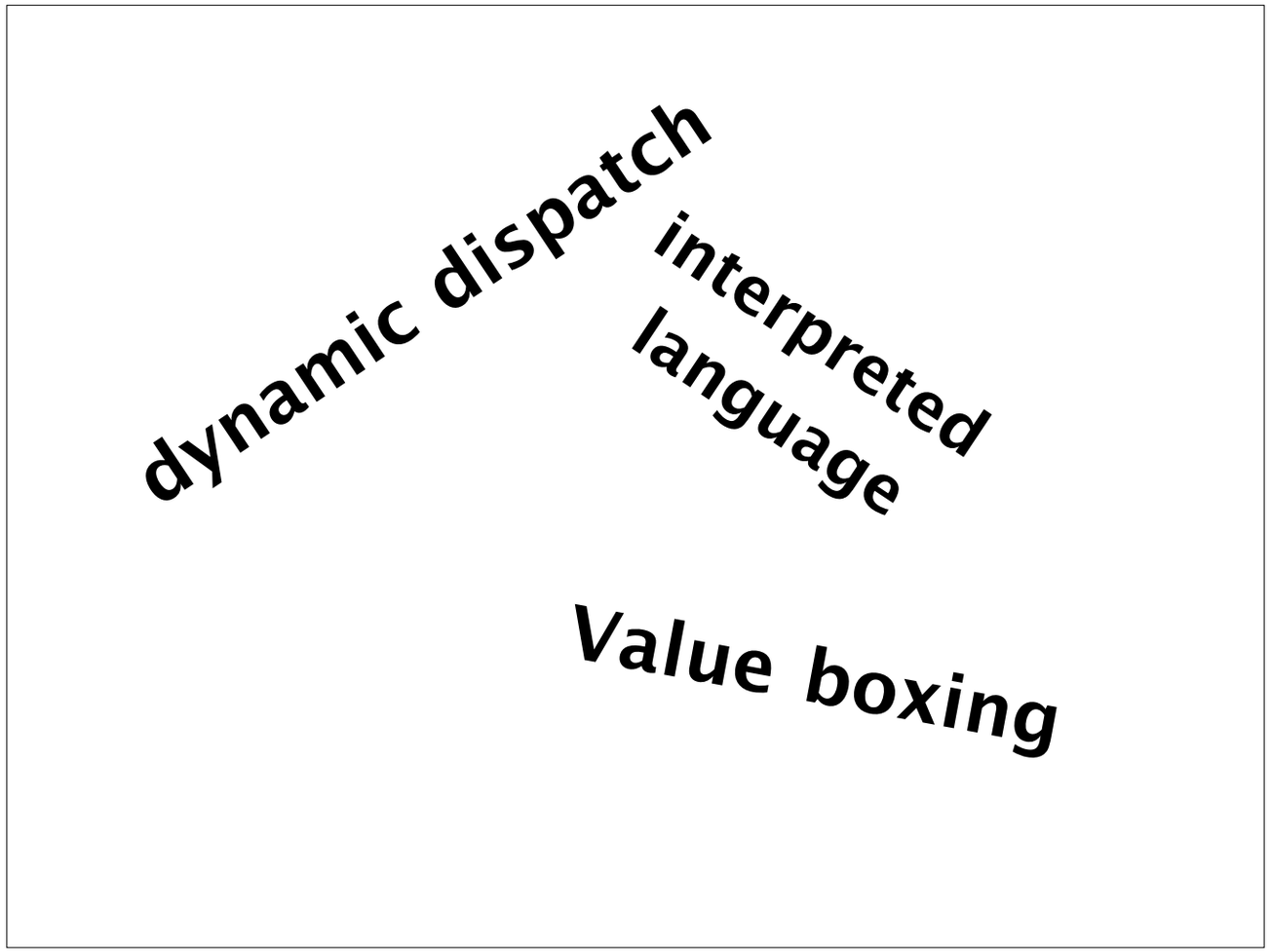
What is universally accepted as a runtime matters a whole lot. That depends on what is universally accepted as a language. In ruby it's "everything that runs rails", in python it's "supports all of Python and all of CPython C API with the same performance characteristics", so essentially CPython

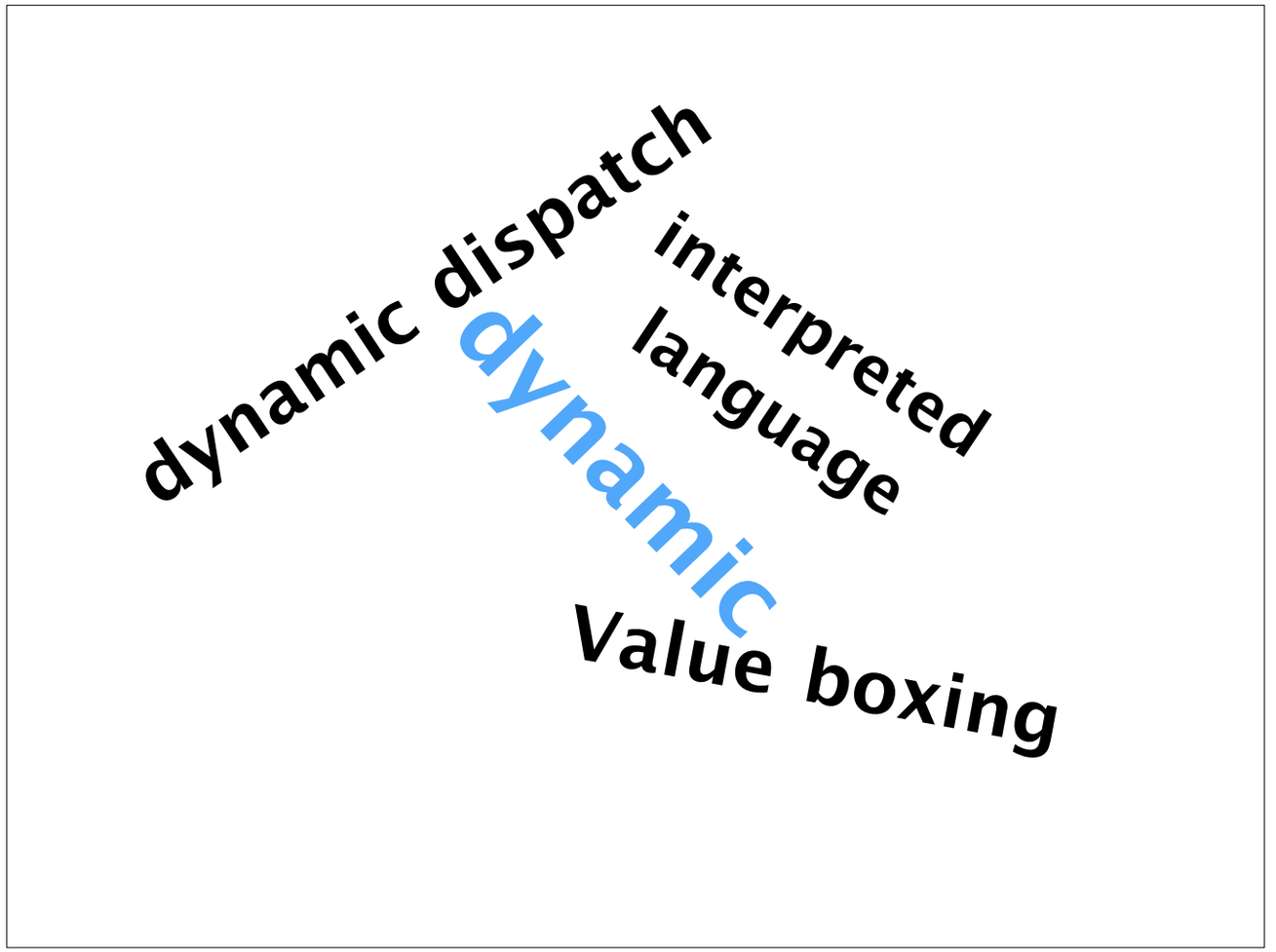
details

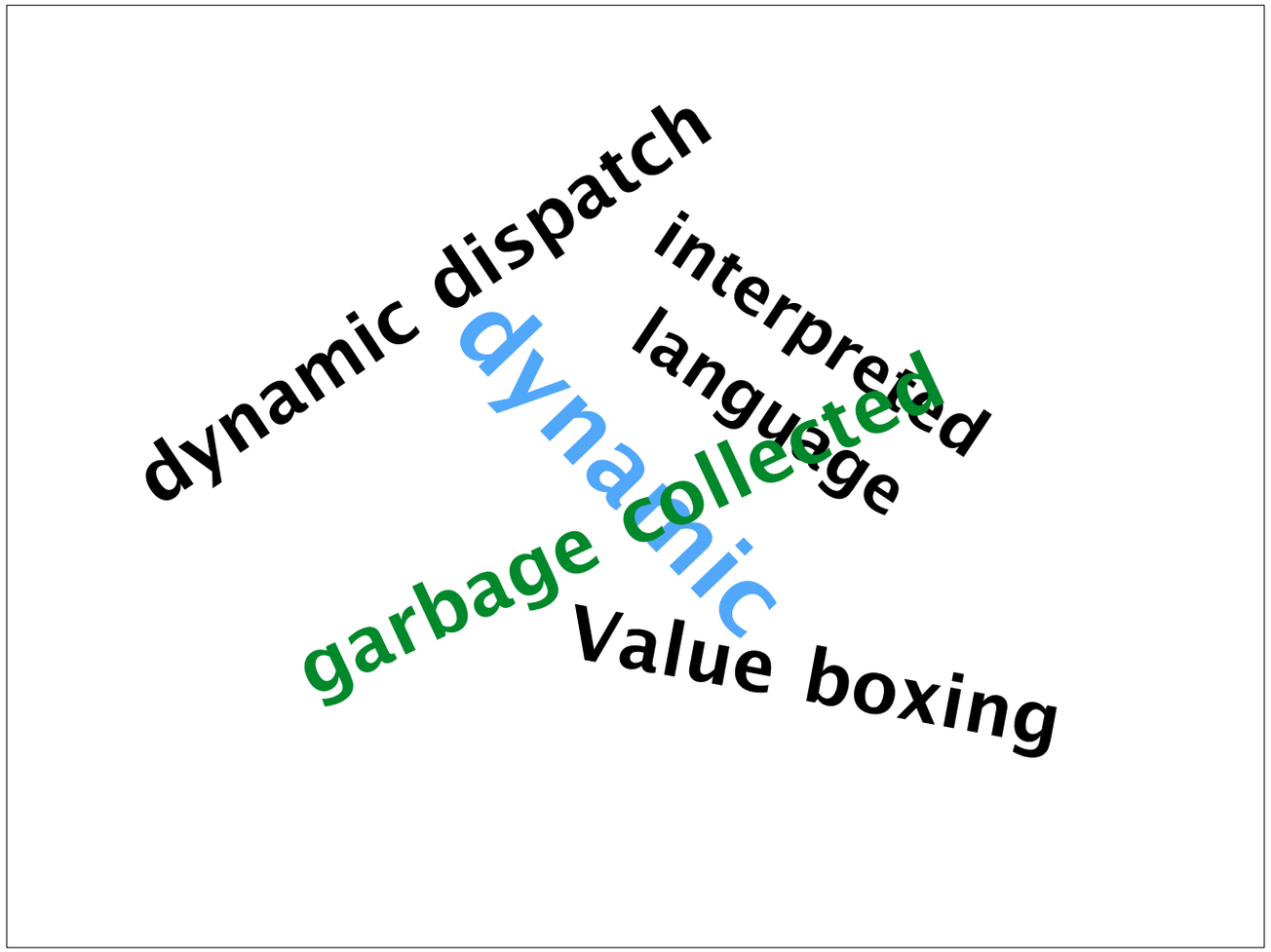
Let's look into details - what exactly am I talking about? Let's move through often discussed reasons why python is slow

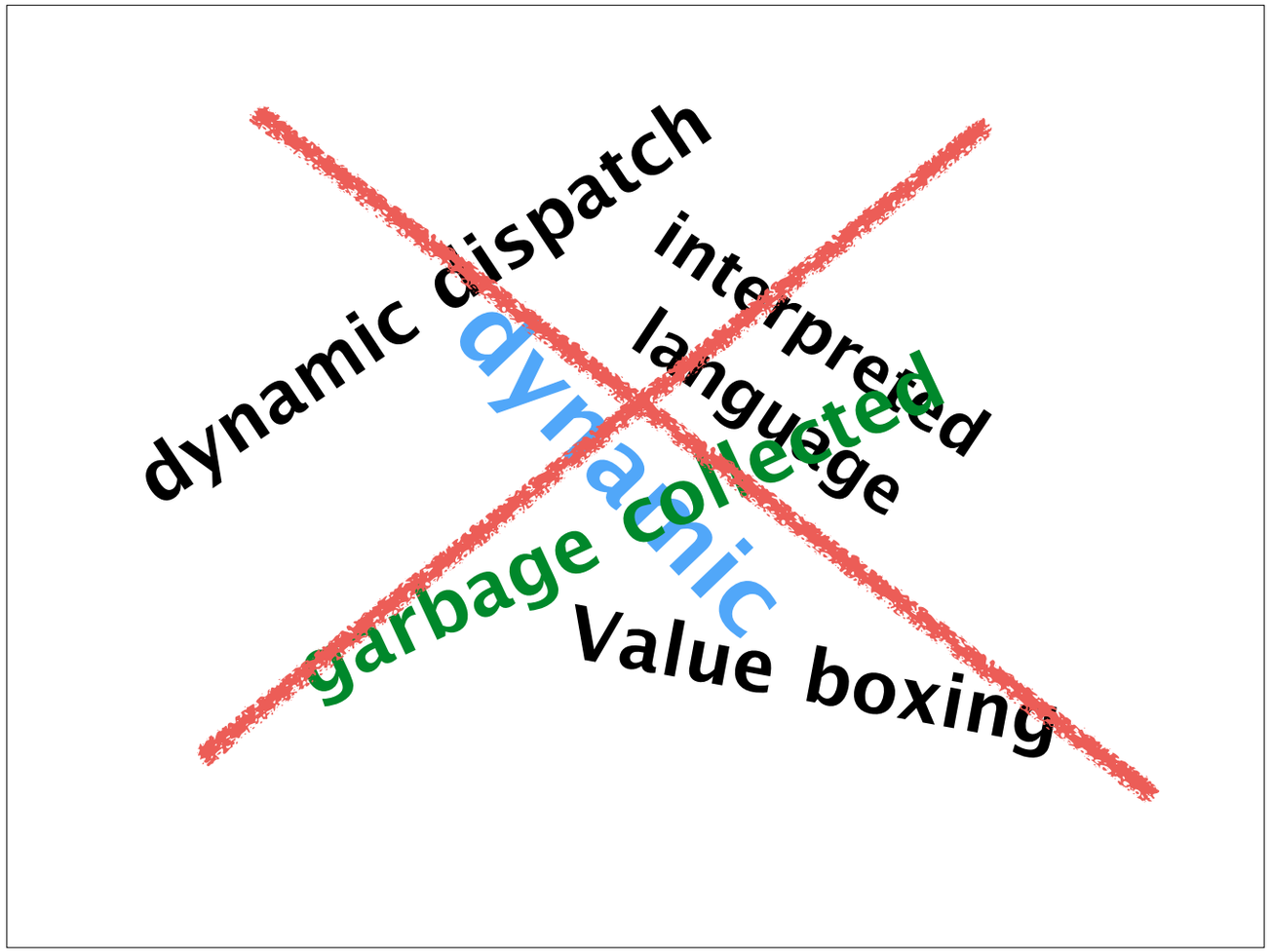
dynamic dispatch


dynamic dispatch
Value boxing


dynamic dispatch
interpreted language
Value boxing


dynamic dispatch
interpreted
language
dynamic
Value boxing


dynamic dispatch
interpreted
language
dynamic
collected
garbage
Value boxing


dynamic dispatch
interpreted language
dynamic
compiled
garbage collected
Value boxing

solved problems

Those are all solved problems for a long time since the release of optimizing smalltalk compilers. When I put figure 20 years here, my colleagues mentioned it was 20 years when we started pypy over 10 years ago. You *can* have a language which has those properties and which is considered a "fast" language.

introspection

Surprisingly - the mere existence of introspection is not a problem at all with a small asterisk. It definitely complicates the runtime and raises the bar for how easy it is to write a new one, but at the same time makes it possible to write debuggers etc. The real issue is how *easy* it is to introspect stuff. Python made it both very easy and inconspicuous by reading a field, which makes people use it all the time everywhere, ORMs I'm looking at you.

unnecessary introspection

Who needs to modify the frame of a running generator? Keep the frame on the exception object?

```
x = (i for i in range(10))
```

```
x.gi_frame.f_locals['.0'].next()
```

lack of
compile time

Maybe a more surprising part would be lack of compile time. Combined with introspection capabilities it forces you to do templating, ORM etc. which should be done via metaprogramming into doing it at runtime.

no notion of
binary

What do you pass to socket.send? Bytes right? Something that implements buffer protocol? maybe buffer? bytearray? memoryview? array.array? cffi.buffer? numpy array? So we have too many notions of binary, with conflicting interfaces, conflicting way to interact with operating system and inevitably making copies and copies of the same stuff.

sheer vastness

That leads to another point - how the hell are VM implementors supposed to deal with all of it? interactions between different pieces, lots and lots of builtin modules (itertools anyone?). Great language should be small.

C API

My personal favorite - C API. The existence of C API essentially prevented any competition in the terms of runtimes. PyPy is the only one that gives a serious go at supporting it, but it creates a major headache and an entry barrier for everyone who tries to improve the runtime, whether in CPython or not.

numeric libraries
I'm looking at you

Since I should probably point some fingers here, pretty much everyone else moved to cffi, but numeric libraries, especially numpy linger along with no plan in sight how to have a fast python

concurrency models

Python lacks a decent concurrency model, which is also an increasing problem with performance. Even if we remove the GIL, which we're trying to do, it would be very difficult to come to agree to a set of semantics how the result behaves.

years of promoting hacks

Since we only ever had one popular python implementation, there are years and years how to write fast software in python that boil down to "don't write python", promoting spaghetti code, strange APIs, that are faster because the loop is in C or using C API.

culture

The culture created tons of popular software that cannot be made fast, no matter how much effort we put into the implementation. The net effect is that people who really need it rewrite software to other languages, despite the fact that same could have been achieved in python, if something like PyPy would be considered. But not without asking hard questions about libraries that we use

performance oriented programming

A good example of that is capnpy - cap'n'proto implementation written with performance in mind. It's fast on CPython and super fast on PyPy - good luck trying to make it faster on some other language.

future

I think it's warranted to end this talk with a somewhat positive summary. There is no language to take python spot. Javascript, go and java are not a real competition, despite massive improvements. Lack of competition should not leave us complacent though.

we improve Python

One option is to improve python. We need to get rid of the C API, we need to get rid of the "python is slow" mindset, but we can't do it without providing a path forward for libraries to transform.

we provide an alternative

There are other options - come up with a language that's better, has no GIL and unicorns and rainbows, but can call into python, hence leveraging the ecosystem.

we do nothing

There is also an option to do nothing. I would not recommend, since doing nothing is against human nature.

Q&A

https://speakerdeck.com/fijal/why-is-python-slow

Thank you. Any questions?