PyCon China 2021

2021 中国 PYTHON 开发者大会

2021.10.16-17

A survey of current Python implementations

Feng Li(李枫)

Agenda

PyCon China 2021

I. Background

- Overview
- Ranking
- Ecosystem
- Limitations
- Tech Stack

II. C-based implementation

- Overview
- Make CPython faster
- Cinder

III. Java-based implementation

- Overview
- Jython
- GraalPython

IV. LLVM-based implementation

- Overview
- Pyston

V. DotNet-based implementation

- Overview
- IronPython
- Pyjion

VI. Wasm-based implementation

- Overview
- Pyodide

VII. Rust-based implementation

- Overview
- RustPython

VIII. RPython-based implementation

- Overview
- PyPy

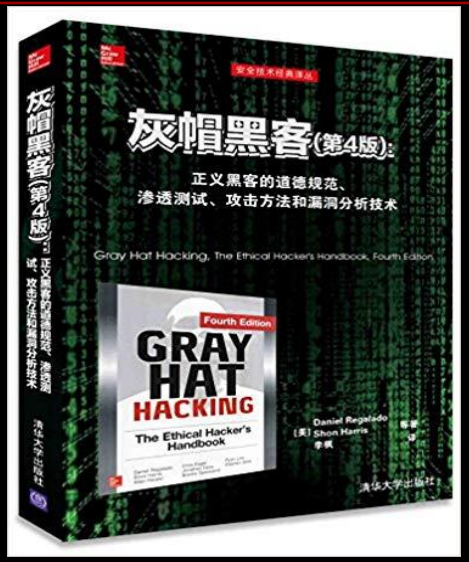
IX. Comparison

- Overview
- Benchmarking

X. Wrap-up

Who Am I

- The main translator of the book «Gray Hat Hacking The Ethical Hacker's Handbook, Fourth Edition» (ISBN：9787302428671) & «Linux Hardening in Hostile Networks, First Edition» (ISBN: 9787115544384)



- Pure software developmement for ~15 years
- Actively participate in various activities of the open source community
- https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/Conf
  https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/LTS
- Recently, focus on infrastructure of Cloud/Edge Computing, AI, Virtualization, Program Runtimes, Network, 5G, RISC-V, EDA…

I. Background

1) Overview

- https://en.wikipedia.org/wiki/Python_(programming_language)

Python was conceived in the late 1980s[39] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to ABC programming language, which was inspired by SETL,[40] capable of exception handling and interfacing with the Amoeba operating system.[11] Its implementation began in December 1989.[41] Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "Benevolent Dictator For Life", a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker.[42] In January 2019, active Python core developers elected a five-member "Steering Council" to lead the project.[43][44]

Python 2.0 was released on 16 October 2000, with many major new features, including a cycle-detecting garbage collector and support for Unicode.[45]

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible.[46] Many of its major features were backported to Python 2.6.x[47] and 2.7.x version series. Releases of Python 3 include the `2to3` utility, which automates the translation of Python 2 code to Python 3.[48]

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3.[49][50] No more security patches or other improvements will be released for it.[51][52] With Python 2's end-of-life, only Python 3.6.x[53] and later are supported.

Python 3.9.2 and 3.8.8 were expedited[54] as all versions of Python (including 2.7[55]) had security issues, leading to possible remote code execution[56] and web cache poisoning.[57]

Design philosophy and features

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming[58] and metaobjects (magic methods)).[59] Many other paradigms are supported via extensions, including design by contract[60][61] and logic programming.[62]

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management.[63] It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has `filter`, `map` and `reduce` functions; list comprehensions, dictionaries, sets, and generator expressions.[64] The standard library has two modules (`itertools` and `functools`) that implement functional tools borrowed from Haskell and Standard ML.[65]

The language's core philosophy is summarized in the document *The Zen of Python* (*PEP 20*), which includes aphorisms such as:[66]

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible (with modules). This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.[39] It is often described as a "batteries included" language due to its comprehensive standard library.[67]

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one— and preferably only one —obvious way to do it" design philosophy.[66] Alex Martelli, a Fellow at the Python Software Foundation and Python book author, writes that "To describe something as 'clever' is *not* considered a compliment in the Python culture."[68]

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity.[69] When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

Python's developers aim for the language to be fun to use. This is reflected in its name—a tribute to the British comedy group Monty Python[70]—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (a reference to a Monty Python sketch) instead of the standard foo and bar.[71][72]

A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*.[73][74]

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as *Pythonistas*.[75][76]

2) Ranking

TIOBE

- https://www.tiobe.com/tiobe-index/

Oct 2021	Oct 2020	Change	Programming Language	Ratings	Change
1	3	↑	Python	11.27%	-0.00%
2	1	↓	C	11.16%	-5.79%
3	2	↓	Java	10.46%	-2.11%
4	4		C++	7.50%	+0.57%
5	5		C#	5.26%	+1.10%
6	6		Visual Basic	5.24%	+1.27%
7	7		JavaScript	2.19%	+0.05%
8	10	↑	SQL	2.17%	+0.61%
9	8	↓	PHP	2.10%	+0.01%
10	17	⇈	Assembly language	2.06%	+0.99%

PYPL

- http://pypl.github.io/PYPL.html

Worldwide, Oct 2021 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	29.66 %	-2.1 %
2		Java	17.18 %	+0.8 %
3		JavaScript	8.81 %	+0.4 %
4		C#	7.3 %	+1.1 %
5	↑	C/C++	6.48 %	+0.7 %
6	↓	PHP	5.92 %	+0.1 %
7		R	4.09 %	+0.2 %
8		Objective-C	2.24 %	-1.2 %
9	↑	TypeScript	1.91 %	+0.1 %
10	↑↑	Kotlin	1.9 %	+0.3 %

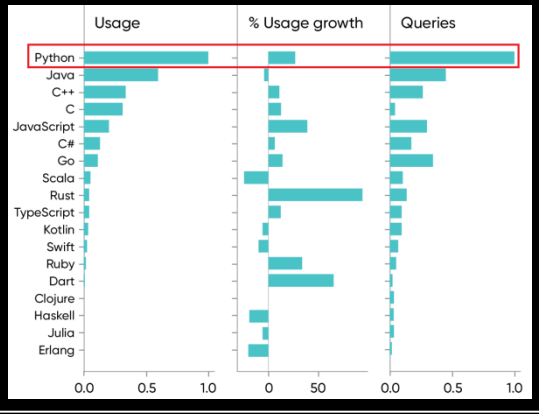
IEEE Spectrum

- https://spectrum.ieee.org/top-programming-languages/

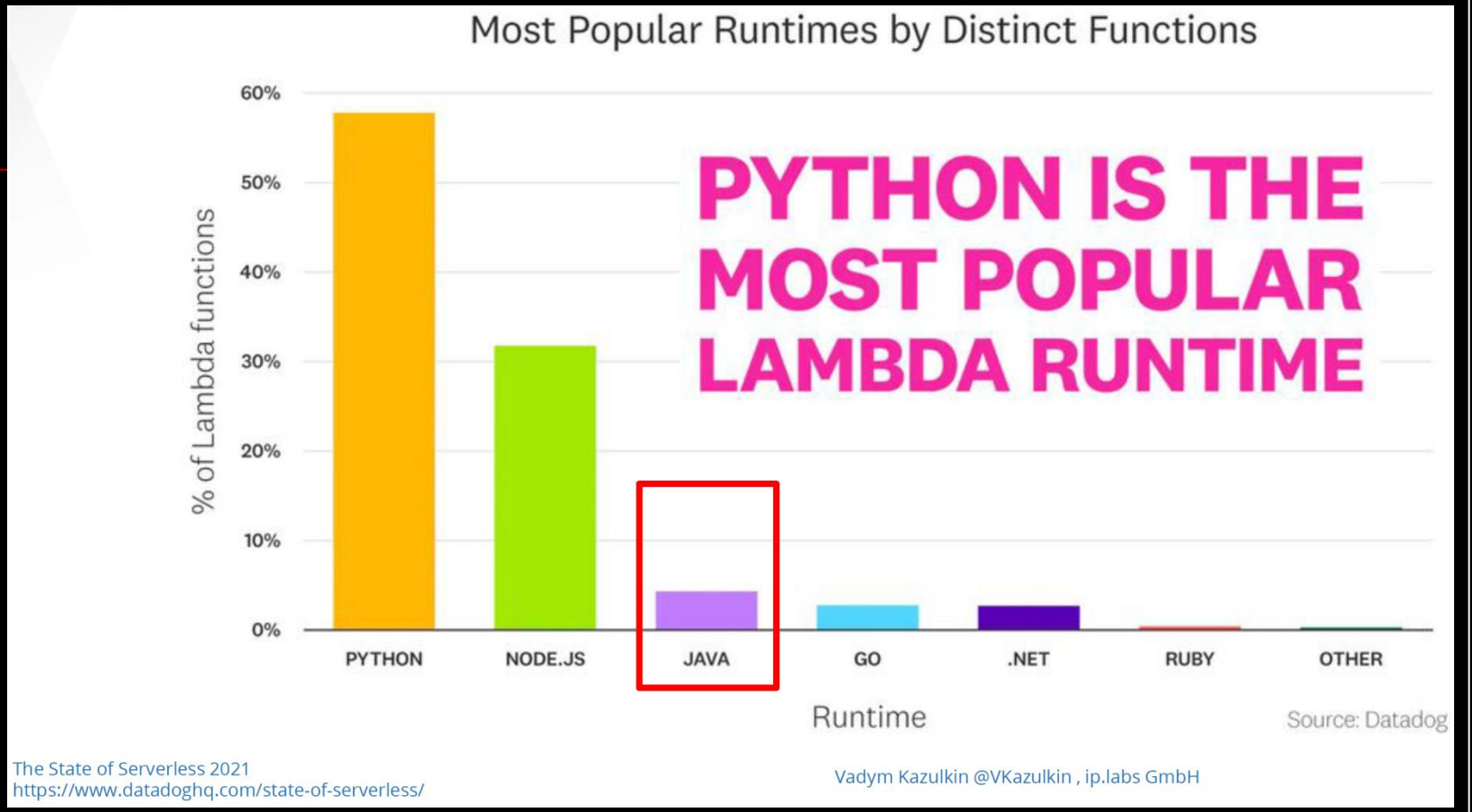
1	Python	100.0
2	Java	95.4
3	C	94.7
4	C++	92.4
5	JavaScript	88.1
6	C#	82.4
7	R	81.7
8	Go	77.7
9	HTML	75.4
10	Swift	70.4

O'Reilly

- https%3A//www.oreilly.com/radar/where-programming-ops-ai-and-the-cloud-are-headed-in-2021/



AWS Lambda


Most Popular Runtimes by Distinct Functions
% of Lambda functions
60%
50%
40%
30%
20%
10%
0%
PYTHON
NODE.JS
JAVA
GO
.NET
RUBY
OTHER
Runtime
Source: Datadog
PYTHON IS THE MOST POPULAR LAMBDA RUNTIME
The State of Serverless 2021
https://www.datadoghq.com/state-of-serverless/
Vadym Kazulkin @VKazulkin , ip.labs GmbH

3) Ecosystem

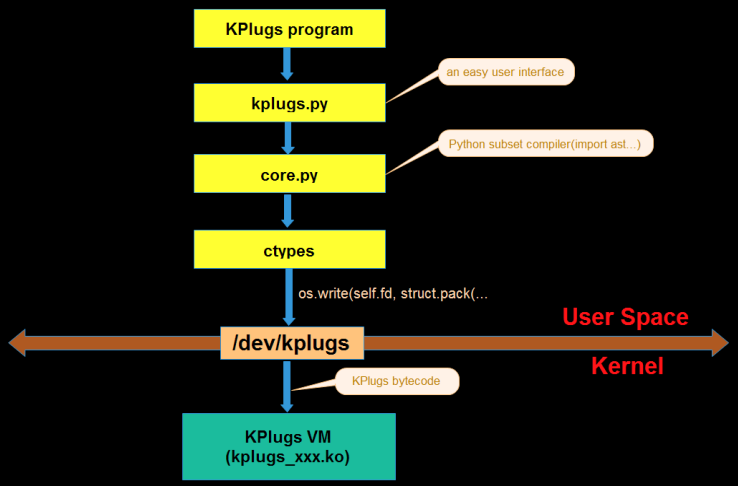
- https://www.developintelligence.com/the-python-ecosystem-of-2020/
- https://deprogrammaticaipsum.com/the-state-of-python-in-2021/
- …

4) Limitations

- **Not a system language**

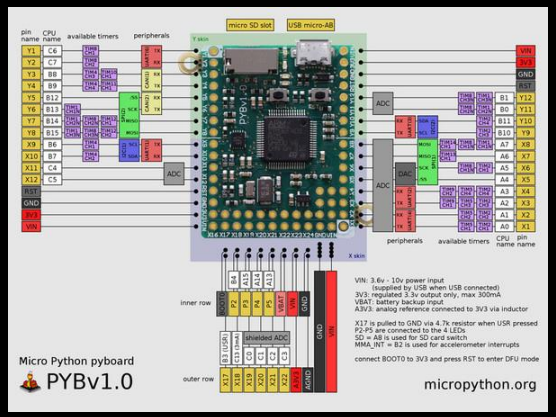
  So it is not a good fit for kernel space or infrastructure development.

  But this is not entirely true:



openstack

Original author(s)	Rackspace Hosting and NASA
Developer(s)	Open Infrastructure Foundation and community
Initial release	21 October 2010; 10 years ago
Stable release	Wallaby[1] / 6 October 2021; 9 days ago
Repository	opendev.org /openstack
Written in	Python
Platform	Cross-platform
Type	Cloud computing
License	Apache License 2.0
Website	www.openstack.org



and more.

Performance

https://github.com/kostya/benchmarks

Base64

Testing base64 encoding/decoding of the large blob into the newly allocated buffers.

Base64

Language	Time, s	Memory, MiB	Energy, J
C/gcc (aklomp)	$0.143_{\pm0.003}$	$1.89_{\pm00.03}$ + $0.00_{\pm00.00}$	$3.48_{\pm00.12}$
Rust	$1.117_{\pm0.018}$	$2.46_{\pm00.09}$ + $0.07_{\pm00.06}$	$24.21_{\pm01.51}$
C/gcc	$1.181_{\pm0.034}$	$1.87_{\pm00.05}$ + $0.00_{\pm00.00}$	$24.32_{\pm01.36}$
V/clang	$1.253_{\pm0.038}$	$1.69_{\pm00.06}$ + $0.17_{\pm00.12}$	$27.29_{\pm01.44}$
Nim/gcc	$1.344_{\pm0.036}$	$2.26_{\pm00.04}$ + $4.44_{\pm00.00}$	$24.70_{\pm01.26}$
V/gcc	$1.419_{\pm0.037}$	$2.12_{\pm00.06}$ + $0.00_{\pm00.00}$	$31.28_{\pm01.15}$
Nim/clang	$1.730_{\pm0.022}$	$2.76_{\pm00.06}$ + $4.38_{\pm00.00}$	$31.06_{\pm01.47}$
Crystal	$1.731_{\pm0.003}$	$3.75_{\pm00.02}$ + $1.84_{\pm00.05}$	$36.14_{\pm00.19}$
D/ldc2	$1.978_{\pm0.009}$	$3.57_{\pm00.04}$ + $3.67_{\pm00.00}$	$35.56_{\pm00.80}$
Vala/gcc	$2.088_{\pm0.028}$	$0.00_{\pm00.00}$ + $0.00_{\pm00.00}$	$46.61_{\pm01.25}$
Ruby (--jit)	$2.094_{\pm0.076}$	$14.47_{\pm00.04}$ + $54.36_{\pm01.53}$	$44.47_{\pm04.35}$
Ruby	$2.142_{\pm0.064}$	$14.48_{\pm00.03}$ + $54.75_{\pm01.87}$	$39.50_{\pm03.31}$
Java	$2.164_{\pm0.062}$	$38.51_{\pm00.32}$ + $318.97_{\pm39.40}$	$49.59_{\pm01.34}$
Vala/clang	$2.239_{\pm0.043}$	$0.00_{\pm00.00}$ + $0.00_{\pm00.00}$	$39.40_{\pm00.88}$
Kotlin	$2.332_{\pm0.063}$	$40.71_{\pm00.14}$ + $335.77_{\pm10.77}$	$49.36_{\pm03.99}$
Scala	$2.364_{\pm0.040}$	$53.91_{\pm00.07}$ + $337.67_{\pm13.53}$	$54.24_{\pm02.59}$
Go	$2.587_{\pm0.005}$	$4.43_{\pm00.01}$ + $5.25_{\pm00.12}$	$51.93_{\pm00.40}$
C++/g++ (libcrypto)	$2.733_{\pm0.108}$	$5.55_{\pm00.12}$ + $0.07_{\pm00.00}$	$55.73_{\pm05.46}$
Node.js	$2.814_{\pm0.064}$	$32.30_{\pm00.08}$ + $36.36_{\pm00.07}$	$60.55_{\pm02.04}$
Perl (MIME::Base64)	$2.896_{\pm0.030}$	$14.11_{\pm00.03}$ + $0.08_{\pm00.00}$	$63.67_{\pm01.34}$
PHP	$2.960_{\pm0.049}$	$15.72_{\pm00.14}$ + $0.00_{\pm00.00}$	$50.69_{\pm01.54}$
Go/gccgo	$3.233_{\pm0.021}$	$23.39_{\pm00.25}$ + $8.76_{\pm00.33}$	$66.18_{\pm00.67}$
D/gdc	$3.295_{\pm0.102}$	$7.04_{\pm00.02}$ + $3.52_{\pm00.00}$	$69.16_{\pm03.88}$
D/dmd	$3.718_{\pm0.055}$	$3.80_{\pm00.07}$ + $3.61_{\pm00.00}$	$73.01_{\pm02.23}$
Python	$3.971_{\pm0.063}$	$9.96_{\pm00.03}$ + $0.18_{\pm00.00}$	$90.62_{\pm01.60}$
Zig	$4.513_{\pm0.010}$	$1.88_{\pm00.02}$ + $0.34_{\pm00.03}$	$82.21_{\pm01.48}$
Python/pypy	$4.886_{\pm0.142}$	$65.43_{\pm00.06}$ + $45.75_{\pm00.09}$	$98.29_{\pm08.01}$
Tcl	$4.998_{\pm0.026}$	$4.88_{\pm00.06}$ + $0.19_{\pm00.03}$	$84.07_{\pm02.10}$
Julia	$5.740_{\pm0.202}$	$218.99_{\pm00.04}$ + $63.02_{\pm00.17}$	$128.13_{\pm09.02}$
F#/.NET Core	$5.765_{\pm0.045}$	$37.22_{\pm00.04}$ + $36.42_{\pm08.18}$	$106.41_{\pm01.19}$
C#/.NET Core	$5.847_{\pm0.015}$	$34.66_{\pm00.04}$ + $40.93_{\pm07.60}$	$108.42_{\pm01.65}$
Ruby/truffleruby (--jvm)	$6.302_{\pm0.210}$	$625.38_{\pm09.77}$ + $141.33_{\pm42.46}$	$137.95_{\pm11.33}$
C#/Mono	$7.259_{\pm0.059}$	$20.84_{\pm00.03}$ + $18.49_{\pm00.07}$	$174.00_{\pm01.80}$
Ruby/jruby	$11.868_{\pm0.308}$	$184.35_{\pm02.57}$ + $138.17_{\pm15.24}$	$266.12_{\pm09.10}$
Perl (MIME::Base64::Perl)	$15.947_{\pm0.181}$	$15.51_{\pm00.03}$ + $0.16_{\pm00.06}$	$362.72_{\pm04.38}$
Ruby/truffleruby	$20.507_{\pm0.491}$	$427.34_{\pm01.89}$ + $320.53_{\pm02.48}$	$367.59_{\pm20.16}$

...

https://www.techempower.com/benchmarks/

Best (bar chart) | Data table | Latency | Framework overhead

Best fortunes responses per second, Dell R440 Xeon Gold + 10 GbE (436 tests)

Rnk	Framework	Best performance (higher is better)		Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	drogon-core	666,737	100.0%	0	Ful	C++	Non	Non	Lin	Pg	Lin	Raw	Rea
2	lithium-postgres-batch	659,850	99.0%	0	Mcr	C++	Non	Non	Lin	Pg	Lin	Ful	Rea
3	ntex [sailfish]	655,964	98.4%	0	Mcr	rs	Non	nte	Lin	Pg	Lin	Raw	Rea
4	ntex [db]	654,073	98.1%	0	Mcr	rs	Non	nte	Lin	Pg	Lin	Raw	Rea
5	actix-core	653,529	98.0%	0	Plt	rs	Non	act	Lin	Pg	Lin	Raw	Rea
6	actix-pg	612,258	91.8%	0	Mcr	rs	Non	act	Lin	Pg	Lin	Raw	Rea
7	drogon	552,293	82.8%	0	Ful	C++	Non	Non	Lin	Pg	Lin	Mcr	Rea
8	may-minihttp	489,691	73.4%	0	Mcr	rs	rs	may	Lin	Pg	Lin	Raw	Rea
9	just-js	467,321	70.1%	0	Plt	JS	jus	Non	Lin	Pg	Lin	Raw	Rea
10	jooby-pgclient	423,234	63.5%	0	Ful	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea

...

How to break the ice

- Use a subset of Python, a variant, or a Python-based DSL for system programming.
- Customize a special Python runtime…
- https://en.wikipedia.org/wiki/Source-to-source_compiler.
- …

5) Tech Stack

- Managed programming language
- Dynamic programming language

Runtime

- https://en.wikipedia.org/wiki/Runtime_system
- https://en.wikipedia.org/wiki/Register_machine
- https://en.wikipedia.org/wiki/Stack_machine
- https://en.wikipedia.org/wiki/Intermediate_representation
- https://en.wikipedia.org/wiki/Bytecode
- https://en.wikipedia.org/wiki/Compiler
- https://en.wikipedia.org/wiki/Just-in-time_compilation
- https://en.wikipedia.org/wiki/Ahead-of-time_compilation
- https://en.wikipedia.org/wiki/Source-to-source_compiler
- https://en.wikipedia.org/wiki/Interpreter_(computing)
- …
- https://en.wikipedia.org/wiki/Polyglot_(computing)

  Polyglot — use the best tool for the right jobs: high performance, scripting, web, functional programming, etc

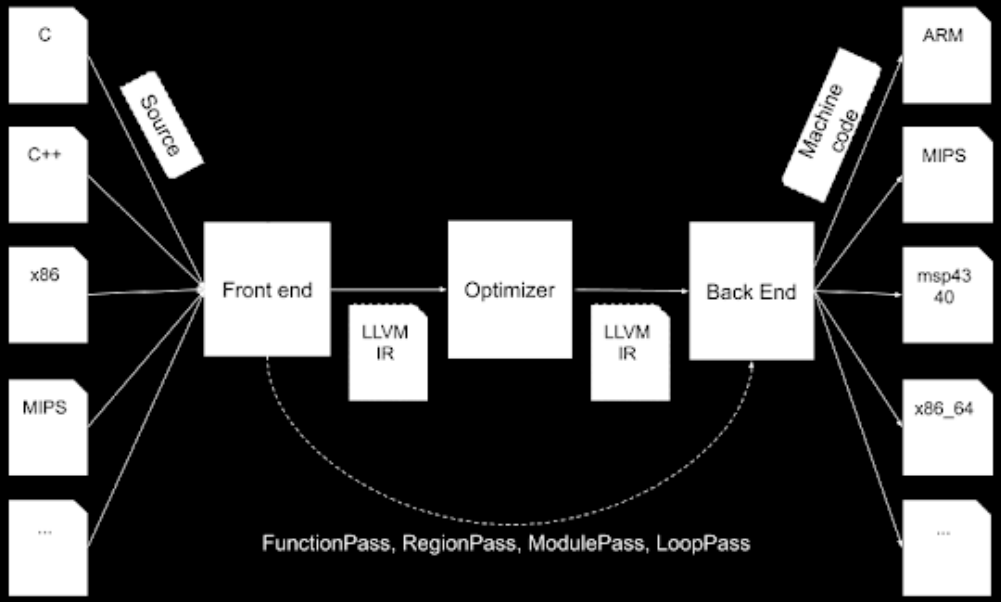
  The most popular Polyglot Runtimes: GraalVM, .Net, WASM…

5.1 LLVM

- https://en.wikipedia.org/wiki/LLVM

LLVM is a set of compiler and toolchain technologies,[5] which can be used to develop a front end for any programming language and a back end for any instruction set architecture. LLVM is designed around a language-independent intermediate representation (IR) that serves as a portable, high-level assembly language that can be optimized with a variety of transformations over multiple passes.[6]

- https://llvm.org
- http://clang.llvm.org/
-



Source: http://blog.k3170makan.com/2020/04/learning-llvm-i-introdution-to-llvm.html

- https://llvm.org/docs/
- https://www.llvm.org/ProjectsWithLLVM/

IR

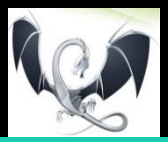
The core of LLVM is the intermediate representation (IR), a low-level programming language similar to assembly. IR is a strongly typed reduced instruction set computing (RISC) instruction set which abstracts away most details of the target. For example, the calling convention is abstracted through *call* and *ret* instructions with explicit arguments. Also, instead of a fixed set of registers, IR uses an infinite set of temporaries of the form %0, %1, etc. LLVM supports three equivalent forms of IR: a human-readable assembly format, an in-memory format suitable for frontends, and a dense bitcode format for serializing. A simple "Hello, world!" program in the IR format:[32]

```
@.str = internal constant [14 x i8] c"hello, world\0A\00"

declare i32 @printf(i8*, ...)

define i32 @main(i32 %argc, i8** %argv) nounwind {
entry:
    %tmp1 = getelementptr [14 x i8], [14 x i8]* @.str, i32 0, i32 0
    %tmp2 = call i32 (i8*, ...) @printf( i8* %tmp1 ) nounwind
    ret i32 0
}
```

- https://llvm.org/docs/LangRef.html
- MLIR:
  https://mlir.llvm.org/
  A hybrid IR which can support multiple different requirements in a unified infrastructure.

- Bitcode:
  https://llvm.org/docs/BitCodeFormat.html
  https://llvm.org/docs/CommandGuide/llvm-bcanalyzer.html
  https://zhuanlan.zhihu.com/p/308201373

LLVM vs GCC



GPL v3	UIUC, MIT
Front-end: CC1 / CPP	Front-end: Clang
ld.bfd / ld.gold	lld / mclinker
gdb	lldb
as / objdump	MC layer
glibc	llvm-libc?
libstdc++	libc++
libsupc++	libc++abi
libgcc	libcompiler-rt
libgccjit	libLLVMMCJIT
. . .	ORC JIT, Coroutines, Clangd, libclc, Falcon...

- http://lld.llvm.org/
- https://llvm.org/docs/Proposals/LLVMLibC.html

5.2 WASM

- https://en.wikipedia.org/wiki/WebAssembly

C source code and corresponding WebAssembly

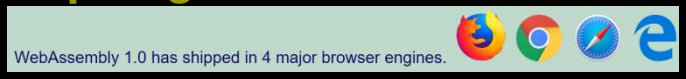
C source code	WebAssembly `.wat` text format	WebAssembly `.wasm` binary format

`int factorial(int n) {`
`  if (n == 0)`
`  return 1;`
`  else`
`  return n * factorial(n-1);`
`}`
	`(func (param i64) (result i64)`
	`  local.get 0`
	`  i64.eqz`
	`  if (result i64)`
	`  i64.const 1`
	`  else`
	`  local.get 0`
	`  local.get 0`
	`  i64.const 1`
	`  i64.sub`
	`  call 0`
	`  i64.mul`
	`  end)`
		`00 61 73 6D 01 00 00 00`
		`01 00 01 60 01 73 01 73 06`
		`03 00 01 00 02`
		`0A 00 01`
		`00 00`
		`20 00`
		`50`
		`04 7E`
		`42 01`
		`05`
		`20 00`
		`20 00`
		`42 01`
		`7D`
		`10 00`
		`7E`
		`0B`
		`0B 15 17`

- https://webassembly.org/

  WebAssembly(abbr. *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.

- https://github.com/WebAssembly/design

- 

- https://webassembly.org/specs/

Limitations

1. In general, WebAssembly does not allow direct interaction with the DOM. All interaction must flow through JavaScript interop.
2. Multithreading (although there are plans to address this.)
3. Garbage collection (although there are plans to address this.)
4. Security considerations (discussed below)

WebAssembly is supported on desktops, and mobile, but on the latter, in practice (for non-small memory allocations, such as with Unity game engine) there are "grave limitations that make many applications infeasible to be *reliably* deployed on mobile browsers [..] Currently allocating more than ~300MB of memory is not reliable on Chrome on Android without resorting to Chrome-specific workarounds, nor in Safari on iOS."[62]

All major web browsers allow WebAssembly if Content-Security-Policy is not specified, or if "unsafe-eval" is used, but otherwise the major web browsers behave differently.[63] In practice WebAssembly can't be used on Chrome without "unsafe-eval",[64][65] while a worker thread workaround is available.[66]

Runtime Implementations

- While WebAssembly was initially designed to enable near-native code execution speed in the web browser, it has been considered valuable outside of such, in more generalized contexts.[37][38] Since WebAssembly's runtime environments (RE) are low level virtual stack machines (akin to JVM or Flash VM) that can be embedded into host applications, some of them have found a way to standalone runtime environments like Wasmtime and Wasmer.[8][13]

  Web browsers [edit]

  In November 2017, Mozilla declared support "in all major browsers",[39] after WebAssembly was enabled by default in Edge 16.[40] The support includes mobile web browsers for iOS and Android. As of July 2021, 94% of installed browsers support WebAssembly.[41] But for older browsers, Wasm can be compiled into asm.js by a JavaScript polyfill.[42]

Compilers:

Because WebAssembly executables are precompiled, it is possible to use a variety of programming languages to make them.[43] This is achieved either through direct compilation to Wasm, or through implementation of the corresponding virtual machines in Wasm. There have been around 40 programming languages reported to support Wasm as a compilation target.[44]

Emscripten compiles C and C++ to Wasm[32] using the Binaryen and LLVM as backend.[45]

As of version 8, a standalone Clang can compile C and C++ to Wasm.[46]

Its initial aim is to support compilation from C and C++,[47] though support for other source languages such as Rust, .NET languages[48][49][44] and AssemblyScript[50] (TypeScript-like) is also emerging. After the MVP release, there are plans to support multithreading and garbage collection[51][52] which would make WebAssembly a compilation target for garbage-collected programming languages like C# (supported via Blazor), F# (supported via Bolero[53] with help of Blazor), Python, and even JavaScript where the browser's just-in-time compilation speed is considered too slow. A number of other languages have some support including Python,[54] Java,[55] Julia,[56][57][58] Zig,[59] and Ruby,[60] as well as Go.[61]

- https://github.com/appcypher/awesome-wasm-langs
- https://github.com/appcypher/awesome-wasm-runtimes

- **Wasm & Rust**

  https://www.rust-lang.org/what/wasm

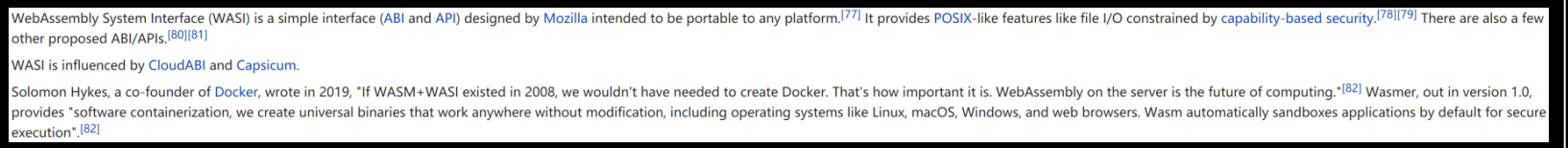
  https://rustwasm.github.io/book

  https://github.com/rustwasm

Beyond the browser

- https://webassembly.org/docs/non-web/
- https://www.zdnet.com/article/microsoft-google-back-bytecode-alliance-to-move-webassembly-beyond-the-browser/

- **WASI (WebAssembly System Interface)**



WebAssembly System Interface (WASI) is a simple interface (ABI and API) designed by Mozilla intended to be portable to any platform.[77] It provides POSIX-like features like file I/O constrained by capability-based security.[78][79] There are also a few other proposed ABI/APIs.[80][81]

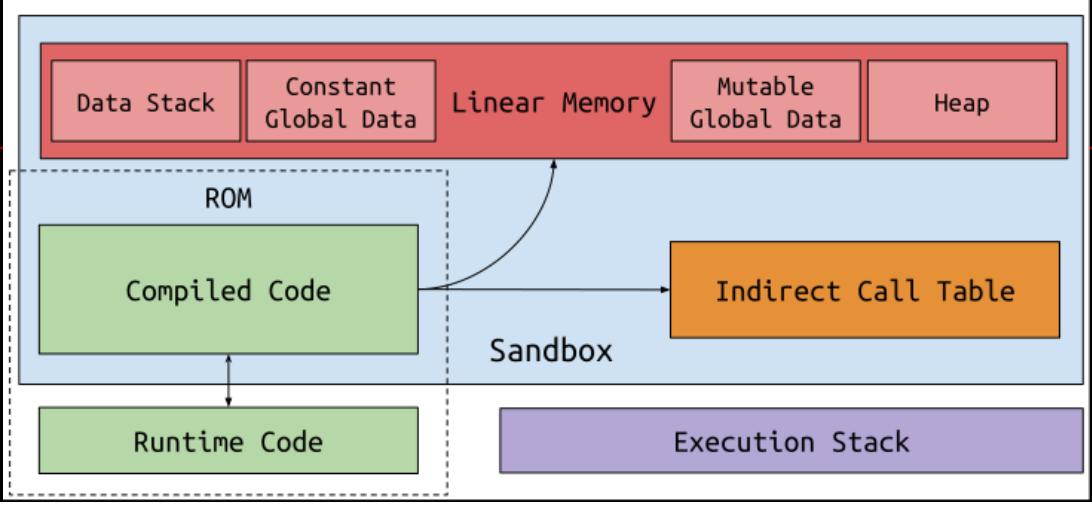
WASI is influenced by CloudABI and Capsicum.

Solomon Hykes, a co-founder of Docker, wrote in 2019, "If WASM+WASI existed in 2008, we wouldn't have needed to create Docker. That's how important it is. WebAssembly on the server is the future of computing."[82] Wasmer, out in version 1.0, provides "software containerization, we create universal binaries that work anywhere without modification, including operating systems like Linux, macOS, Windows, and web browsers. Wasm automatically sandboxes applications by default for secure execution".[82]

https://wasi.dev/
https://github.com/WebAssembly/WASI
https://github.com/bytecodealliance/wasmtime/blob/main/docs/WASI-documents.md
https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/
https://training.linuxfoundation.org/announcements/wasi-bringing-webassembly-way-beyond-browsers

Runtime

- Sandboxing



Wasm uses a co-design between the compiler, and the dynamic checks of the runtime system to provide the sandbox that isolates the surrounding system from the logic of the contained code. The figure depicts the main aspects of the sandbox. These include:

- *Linear memory* that holds all memory accessed by the sandbox. The compiler emits code that checks that all loads and stores remain within the linear memory, thus preventing errant accesses outside the sandbox. Linear memory is expandable much like a traditional heap.
- The *indirect function call table* that facilitates function pointer calls. To ensure that function pointer invocations are safe (to code generated by the compiler), function pointers reference an *offset* into the table. Each entry includes the type of the function, and ensures that function invocations are well-typed.
- The separation of the *execution stack* -- used to track function calls -- and the *data stack* -- used to contain stack-allocated data that can be referenced, thus must be in linear memory.

The first of these ensures the proper memory isolation of the sandbox, while the latter two provide control-flow integrity of the sandbox.

Source: https://github.com/gwsystems/aWsm/blob/master/doc/design.md

Nanoprocesses

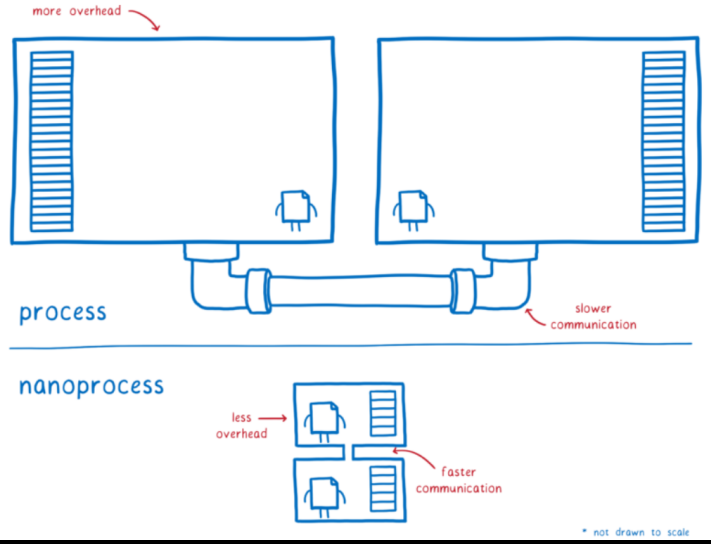
Tomorrow's solution: WebAssembly "nanoprocesses"

WebAssembly can provide the kind of isolation that makes it safe to run untrusted code. We can have an architecture that's like Unix's many small processes, or like containers and microservices.

But this isolation is much lighter weight, and the communication between them isn't much slower than a regular function call.

This means you can use them to wrap a single WebAssembly module instance, or a small collection of module instances that want to share things like memory among themselves.

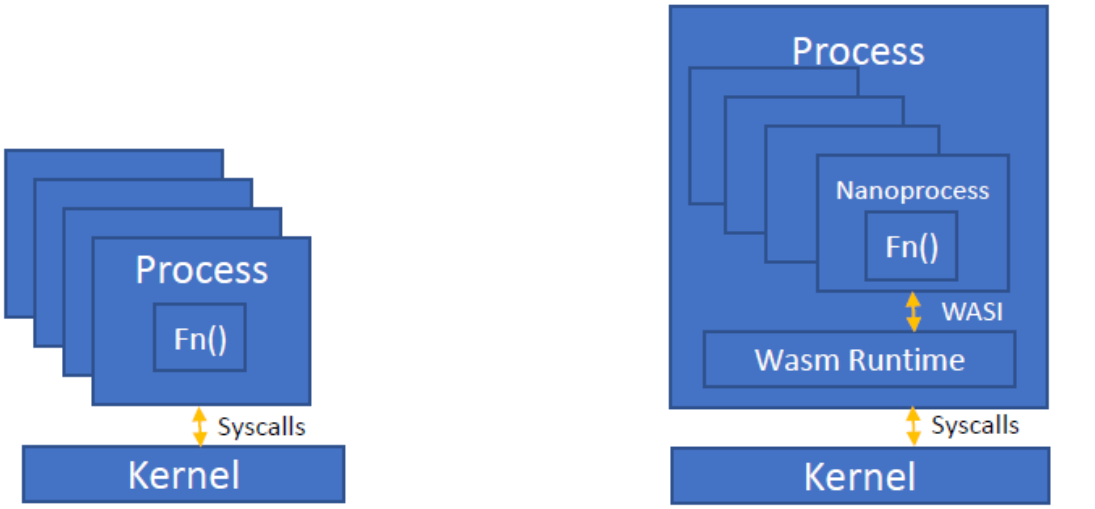
Plus, you don't have to give up the nice programming language affordances—like function signatures and static type checking.



Source: https://hacks.mozilla.org/2019/11/announcing-the-bytecode-alliance/

Architecture

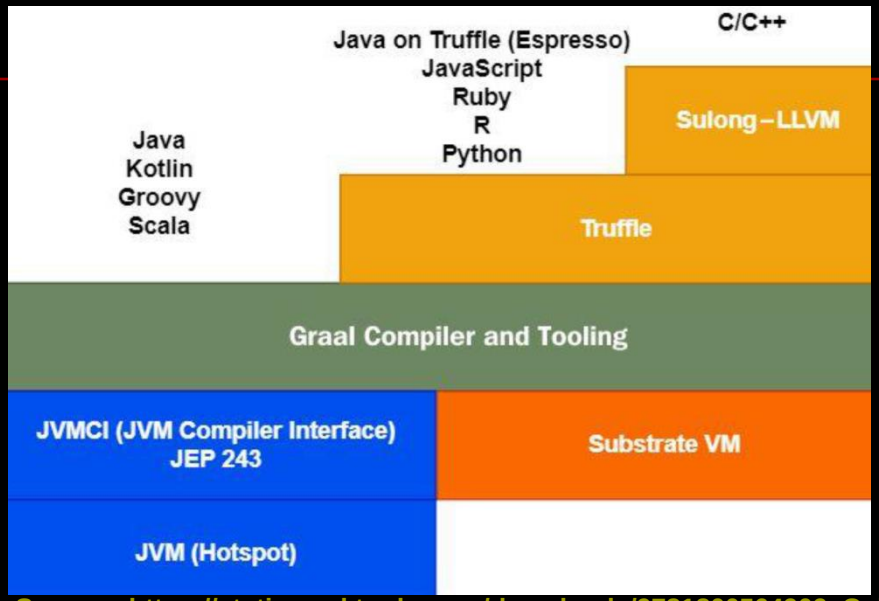
High-level reference architecture for running multiple WebAssembly sandboxes within a single native OS process. A userspace runtime schedules sandbox execution and provides system services.



Source: https://conix.io/wp-content/uploads/pubs/3878/CONIX-Sledge-Poster.pdf

5.3 GraalVM

- https://en.wikipedia.org/wiki/GraalVM
- https://www.graalvm.org/
-

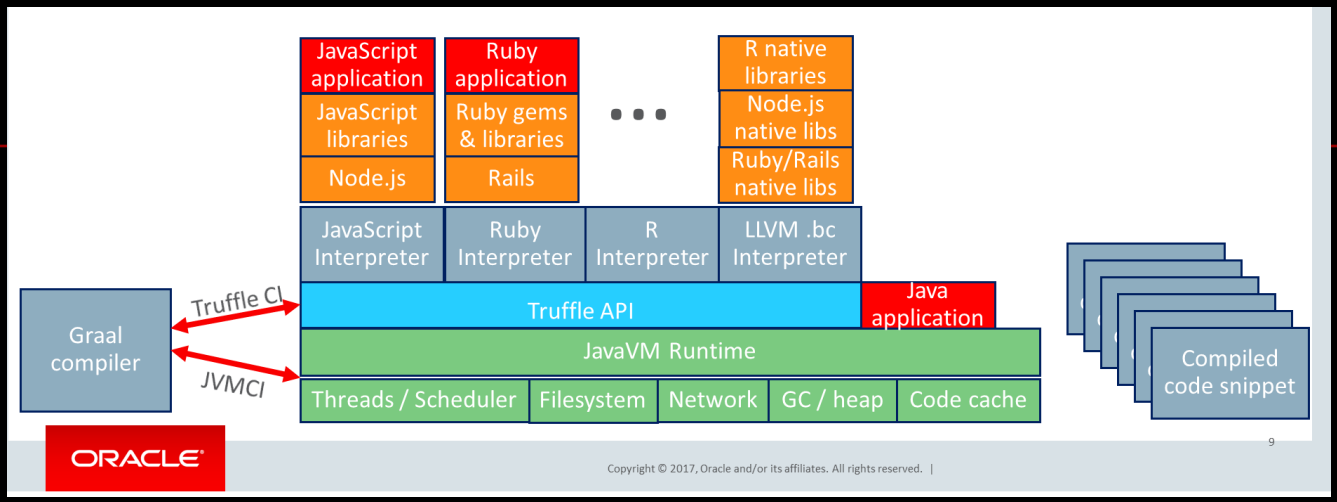


Source: https://static.packt-cdn.com/downloads/9781800564909_ColorImages.pdf

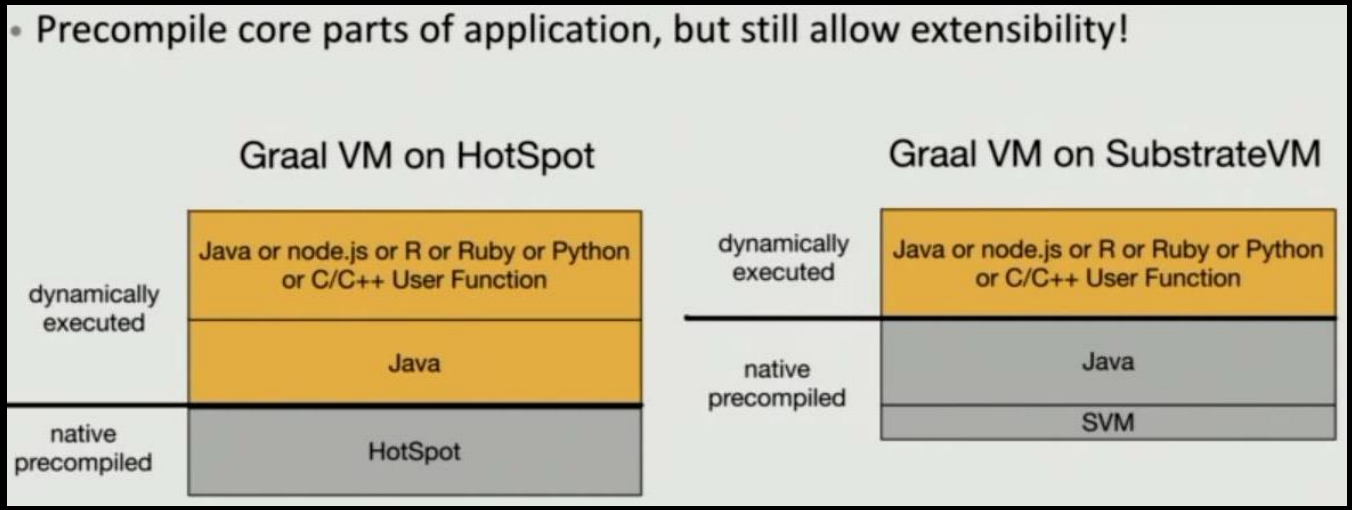
- A Universal High-Performance Polyglot VM
- A meta-runtime for Language-Level Virtualization
- Base on OpenJDK 8, 11, 16, and 17 with JVMCI support.
- https://www.graalvm.org/docs/introduction/
- https://github.com/graalvm
- https://github.com/oracle/graal

Architecture

- A hybrid of static & dynamic runtimes

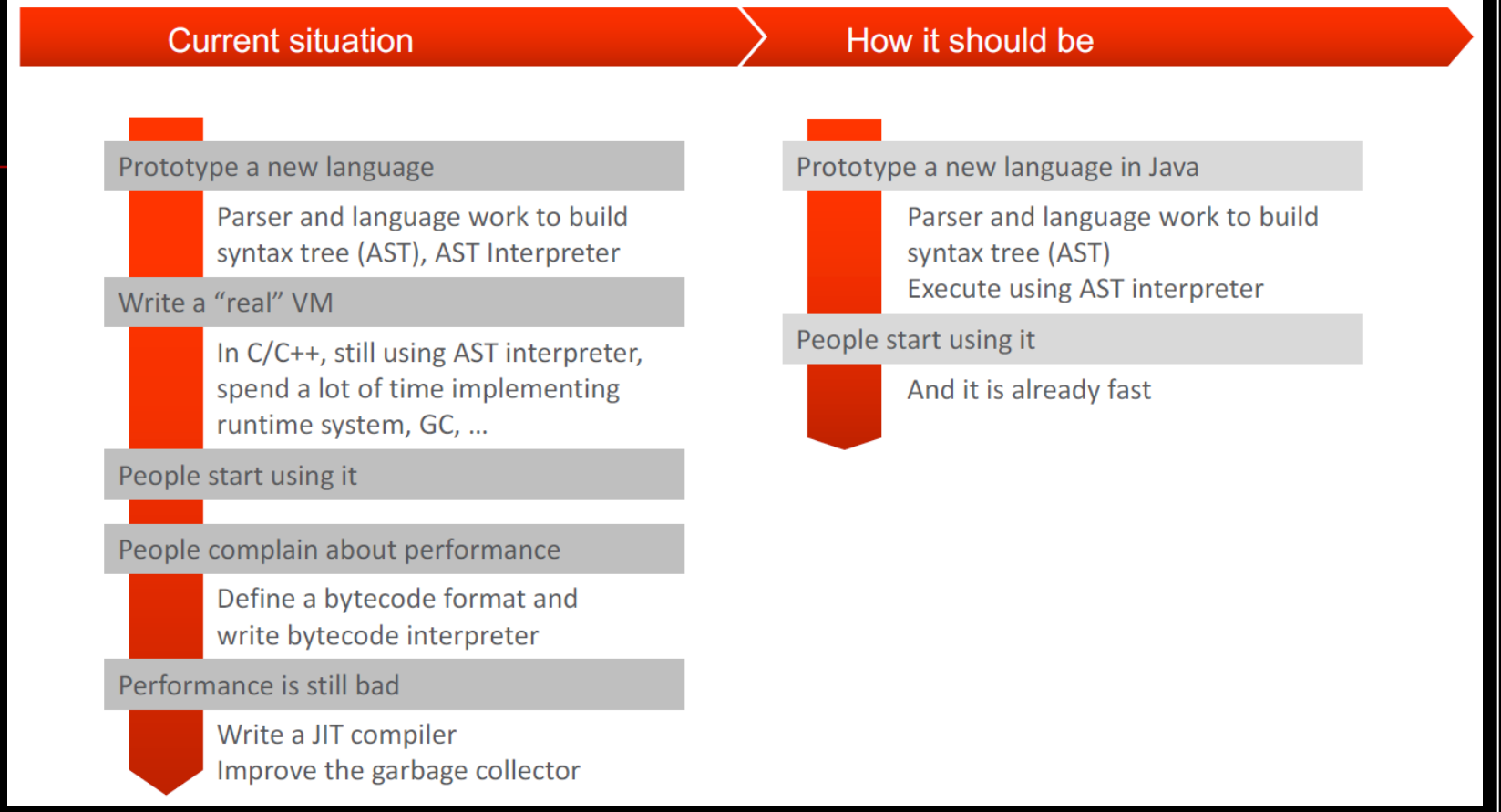


Source: https://ics.psu.edu/wp-content/uploads/2017/02/GraalVM-PSU.pptx



Source: "Adopting Java for the Serverless world", Vadym Kazulkin, JUG London 2020.

Implement a new language runtime



Source: "Turning the JVM into a Polyglot VM with Graal", Chris Seaton, Oracle Labs

Java 17

- https://www.infoq.com/news/2021/09/java17-released/
- https://openjdk.java.net/projects/jdk/17/
- Features

306: Restore Always-Strict Floating-Point Semantics
356: Enhanced Pseudo-Random Number Generators
382: New macOS Rendering Pipeline
391: macOS/AArch64 Port
398: Deprecate the Applet API for Removal
403: Strongly Encapsulate JDK Internals
406: Pattern Matching for switch (Preview)
407: Remove RMI Activation
409: Sealed Classes
410: Remove the Experimental AOT and JIT Compiler
411: Deprecate the Security Manager for Removal
412: Foreign Function & Memory API (Incubator)
414: Vector API (Second Incubator)
415: Context-Specific Deserialization Filters

- Faster LTS and free JDK with Java 17

Languages

- https://github.com/oracle/graal/blob/master/truffle/docs/Languages.md

Language Implementations

This page is intended to keep track of the growing number of language implementations and experiments on top of Truffle. The following language implementations exist already:

- Espresso, a meta-circular Java bytecode interpreter. *
- FastR, an implementation of GNU R. *
- Graal.js, an ECMAScript 2020 compliant JavaScript implementation. *
- Graal.Python, an early-stage implementation of Python. *
- grCUDA, a polyglot CUDA integration.
- SimpleLanguage, a toy language implementation to demonstrate Truffle features.
- SOMns, a Newspeak implementation for Concurrency Research.
- Sulong, an LLVM bitcode interpreter. *
- TRegex, a generic regular expression engine (internal, for use by other languages only). *
- TruffleRuby, an implementation of Ruby. *
- TruffleSOM, a SOM Smalltalk implementation.
- TruffleSqueak, a Squeak/Smalltalk VM implementation and polyglot programming environment.
- Yona, the reference implementation of a minimalistic, strongly and dynamically-typed, parallel and non-blocking, polyglot, strict, functional programming language.
- Enso, an open source, visual language for data science that lets you design, prototype and develop any application by connecting visual elements together.

* Shipped as part of GraalVM.

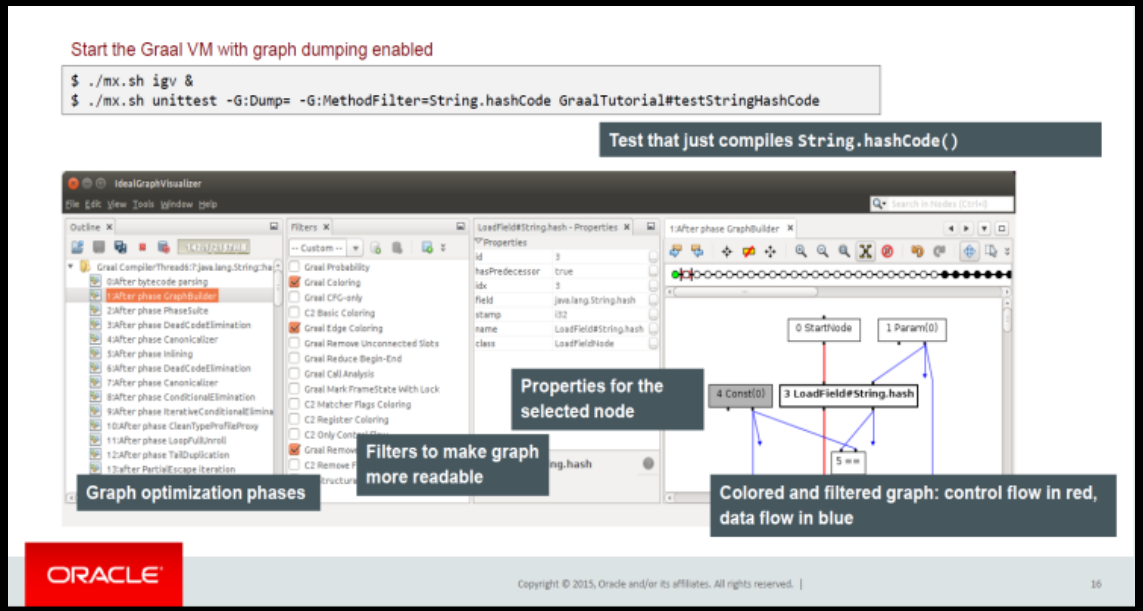
Experiments

- bf, an experimental Brainfuck programming language implementation.
- brainfuck-jvm, another Brainfuck language implementation.
- Cover, a Safe Subset of C++.
- DynSem, a DSL for declarative specification of dynamic semantics of languages.
- Heap Language, a tutorial showing the embedding of Truffle languages via interoperability.
- hextruffle, an implementation of Hex.
- LuaTruffle, an implementation of the Lua language.
- Mozart-Graal, an implementation of the Oz programming language.
- Mumbler, an experimental Lisp programming language.
- PorcE, an Orc language implementation.
- ProloGraal a Prolog language implementation supporting interoperability.
- PureScript, a small, strongly-typed programming language.
- Reactive Ruby, TruffleRuby meets Reactive Programming.
- shen-truffle, a port of the Shen programming language.
- TruffleMATE, a Smalltalk with a completely reified runtime system.
- TrufflePascal, a Pascal interpreter.
- ZipPy, a Python implementation.

- https://llvm.org/docs/Proposals/LLVMLibC.html

MX

- https://github.com/graalvm/mx
- mx is a command line based tool for managing the development of (primarily) Java code. It includes a mechanism for specifying the dependencies as well as making it simple to build, test, run, update, etc the code and built artifacts
- mx is written in Python and is extensible.
- mx –-help
- IR Example: Ideal Graph Visualizer



5.4 .Net

- https://en.wikipedia.org/wiki/.NET
- https://dotnet.microsoft.com/
- https://docs.microsoft.com/en-us/dotnet/standard/glossary
- https://en.wikipedia.org/wiki/C_Sharp_(programming_language)
- …
- https://en.wikipedia.org/wiki/Windows_Runtime
- https://en.wikipedia.org/wiki/.NET_Framework
- …
- https://github.com/quozd/awesome-dotnet
- https://github.com/thangchung/awesome-dotnet-core

Open Source

- https://dotnet.microsoft.com/platform/open-source
- https://github.com/dotnet
- https://github.com/aspnet
- https://github.com/microsoft
- https://en.wikipedia.org/wiki/Mono_(software)
- ...

CLI

- https://en.wikipedia.org/wiki/Common_Language_Infrastructure

The **Common Language Infrastructure** (**CLI**) is an open specification and technical standard originally developed by Microsoft and standardized by ISO (**ISO/IEC 23271**) and Ecma International (**ECMA 335**)[1][2] that describes executable code and a runtime environment that allows multiple high-level languages to be used on different computer platforms without being rewritten for specific architectures. This implies it is platform agnostic. The .NET Framework, .NET and Mono are implementations of the CLI. The metadata format is also used to specify the API definitions exposed by the Windows Runtime.[3][4]

Among other things, the CLI specification describes the following four aspects:

The Common Type System (CTS)

A set of data types and operations that are shared by all CTS-compliant programming languages.

The Metadata

Information about program structure is language-agnostic, so that it can be referenced between languages and tools, making it easy to work with code written in a language the developer is not using.

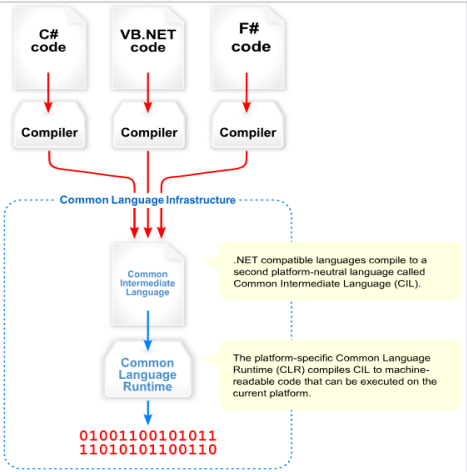
The Common Language Specification (CLS)

The CLI should conform with the set of base rules to which any language targeting, since that language should interoperate with other CLS-compliant languages. The CLS rules define a subset of the Common Type System.

The Virtual Execution System (VES)

The VES loads and executes CLI-compatible programs, using the metadata to combine separately generated pieces of code at runtime.

All compatible languages compile to Common Intermediate Language (CIL), which is an intermediate language that is abstracted from the platform hardware. When the code is executed, the platform-specific VES will compile the CIL to the machine language according to the specific hardware and operating system.



Visual overview of the Common Language Infrastructure (CLI)

- https://en.wikipedia.org/wiki/Common_Intermediate_Language
- https://en.wikipedia.org/wiki/Common_Language_Runtime

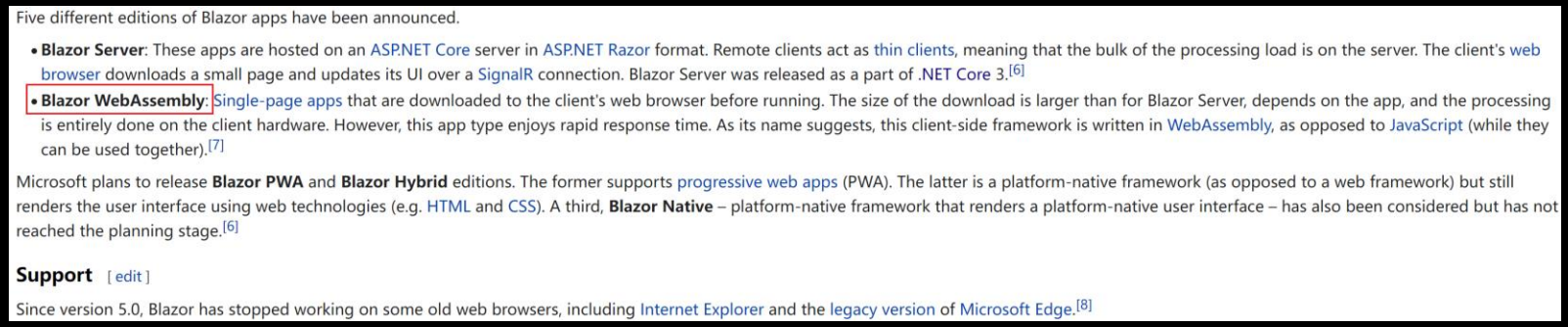
.Net on Web

- **Blazor**

  https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor
  https://github.com/dotnet/aspnetcore
  https://github.com/AdrienTorris/awesome-blazor

- https://en.wikipedia.org/wiki/Blazor

  a free and open-source web framework that enables developers to create web apps using **C# and HTML.**



Five different editions of Blazor apps have been announced.

- **Blazor Server**: These apps are hosted on an ASP.NET Core server in ASP.NET Razor format. Remote clients act as thin clients, meaning that the bulk of the processing load is on the server. The client's web browser downloads a small page and updates its UI over a SignalR connection. Blazor Server was released as a part of .NET Core 3.[6]
- **Blazor WebAssembly**: Single-page apps that are downloaded to the client's web browser before running. The size of the download is larger than for Blazor Server, depends on the app, and the processing is entirely done on the client hardware. However, this app type enjoys rapid response time. As its name suggests, this client-side framework is written in WebAssembly, as opposed to JavaScript (while they can be used together).[7]

Microsoft plans to release **Blazor PWA** and **Blazor Hybrid** editions. The former supports progressive web apps (PWA). The latter is a platform-native framework (as opposed to a web framework) but still renders the user interface using web technologies (e.g. HTML and CSS). A third, **Blazor Native** – platform-native framework that renders a platform-native user interface – has also been considered but has not reached the planning stage.[6]

Support [edit]

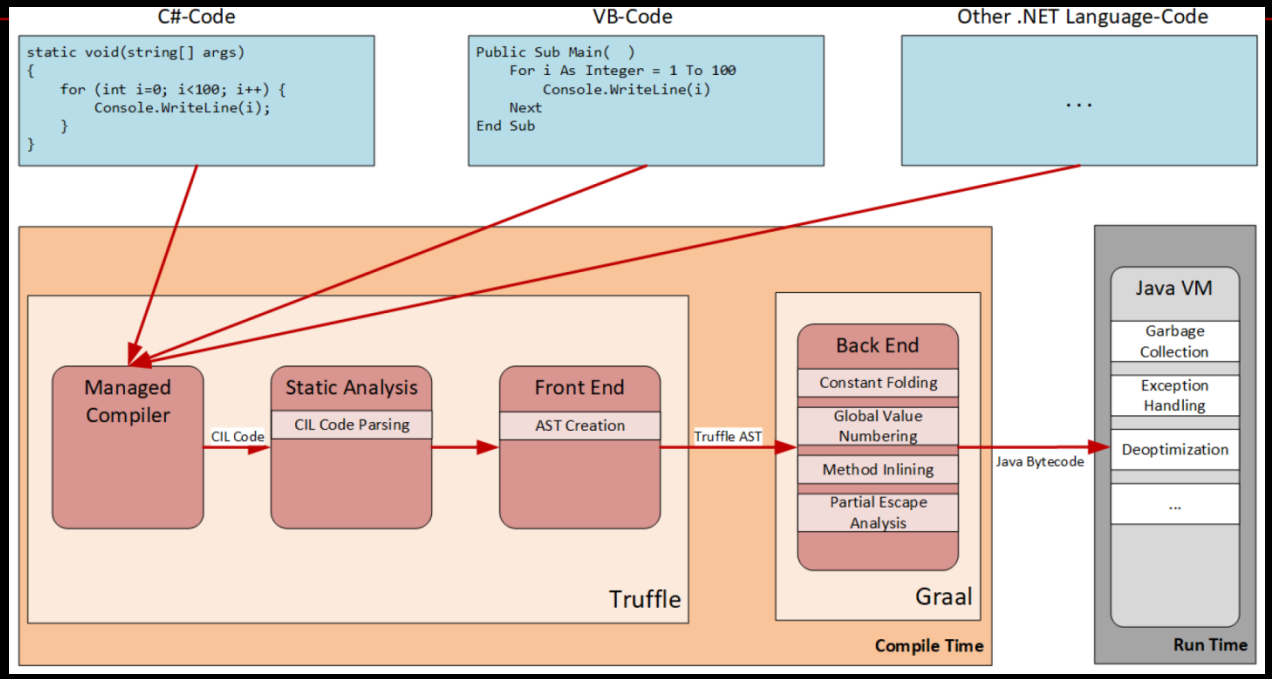
Since version 5.0, Blazor has stopped working on some old web browsers, including Internet Explorer and the legacy version of Microsoft Edge.[8]

.Net on GraalVM

- Truffle CIL Interpreter

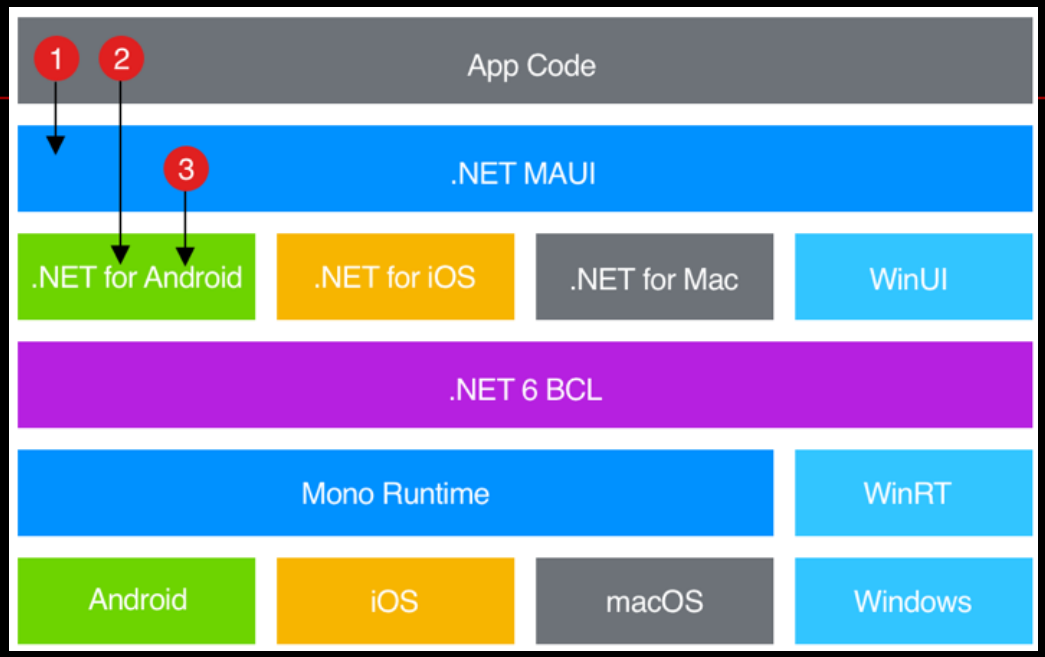
  https://githubmemory.com/repo/jagotu/BACIL

  https://epub.jku.at/obvulihs/download/pdf/5473678



.Net 6

- https://rubikscode.net/2021/08/30/net-6-top-6-new-features-in-the-upcoming-net-6-version/



-

.NET 5	2020-11-10[25]	Visual Studio 2019 Version 16.8	5.0.11	2021-10-12	6 months after .NET 6 release (around May 2022)
.NET 6[25][26]	2021-11-09[27]		6.0.0 RC 2 (release version will be LTS[28])	2021-10-12	November 2024 (projected)
.NET 7[26]	2022-11 (projected)				May 2024 (projected)
.NET 8[26]	2023-11 (projected)		(will be LTS)		November 2026 (projected)

- https://github.com/dotnet/maui/wiki/Installing-.NET-6
- https://www.telerik.com/blogs/whats-new-dotnet-6-blazor
- …

5.5 Rust

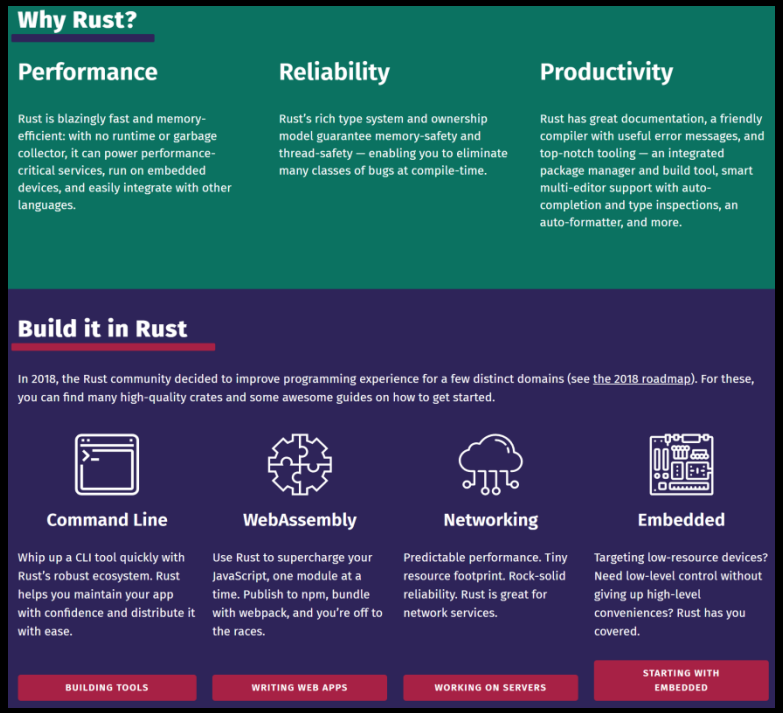
- https://en.wikipedia.org/wiki/Rust_(programming_language)

Rust is a multi-paradigm, high-level, general-purpose programming language designed for performance and safety, especially safe concurrency.[12][13] Rust is syntactically similar to C++,[14] but can guarantee memory safety by using a *borrow checker* to validate references.[15] Rust achieves memory safety without garbage collection, and reference counting is optional.[16][17]

Rust was originally designed by Graydon Hoare at Mozilla Research, with contributions from Dave Herman, Brendan Eich, and others.[18][19] The designers refined the language while writing the Servo layout or browser engine,[20] and the Rust compiler. It has gained increasing use in industry, and Microsoft has been experimenting with the language for secure and safety-critical software components.[21][22]

Rust has been voted the "most loved programming language" in the Stack Overflow Developer Survey every year since 2016.[23]

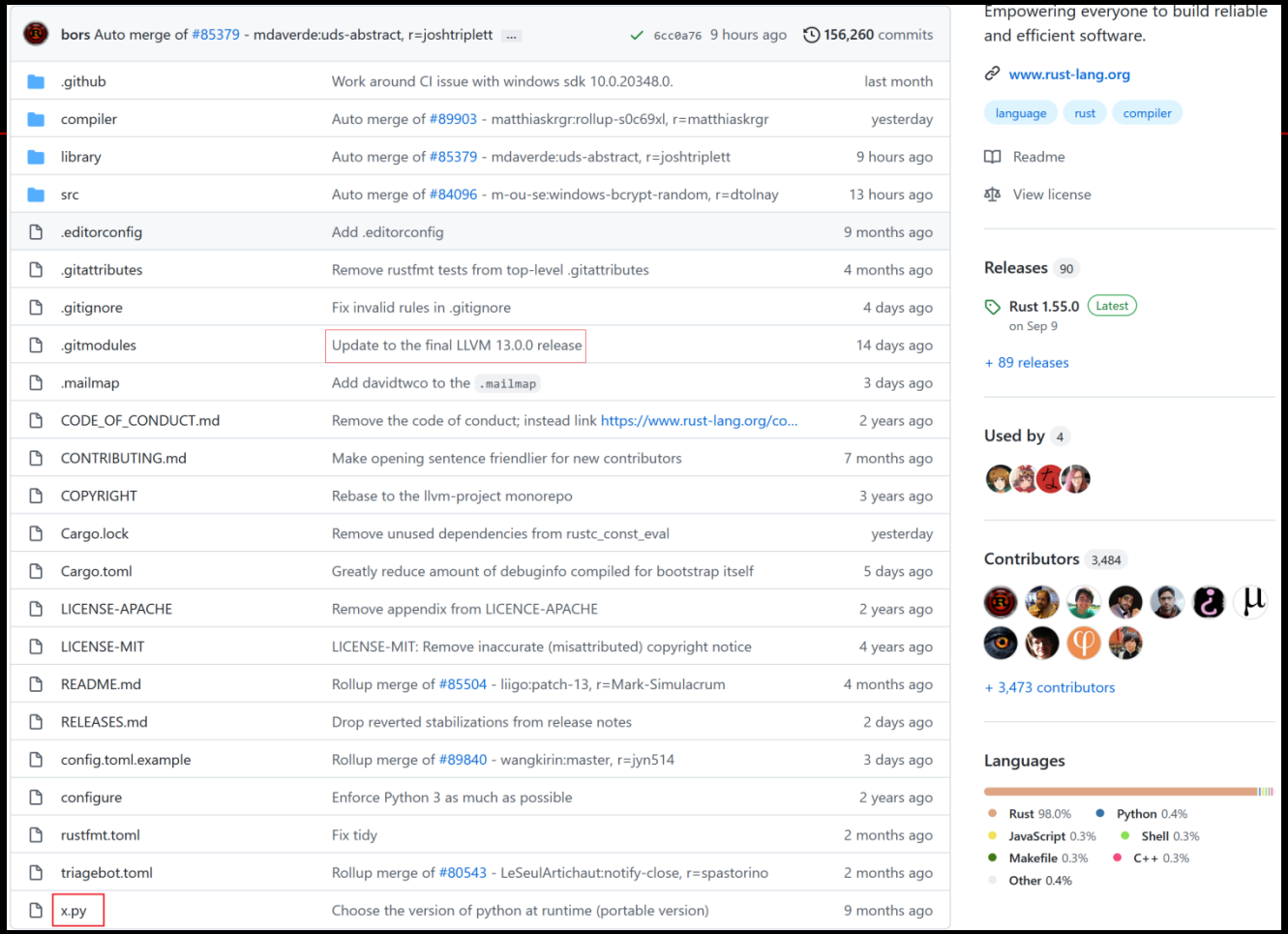
- https://www.rust-lang.org/



- https://www.memorysafety.org/
- ...

rustc & cargo

- **https://github.com/rust-lang/rust**



bors Auto merge of #85379 - mdaverde:uds-abstract, r=joshtriplett ... 6cc0a76 9 hours ago 156,260 commits

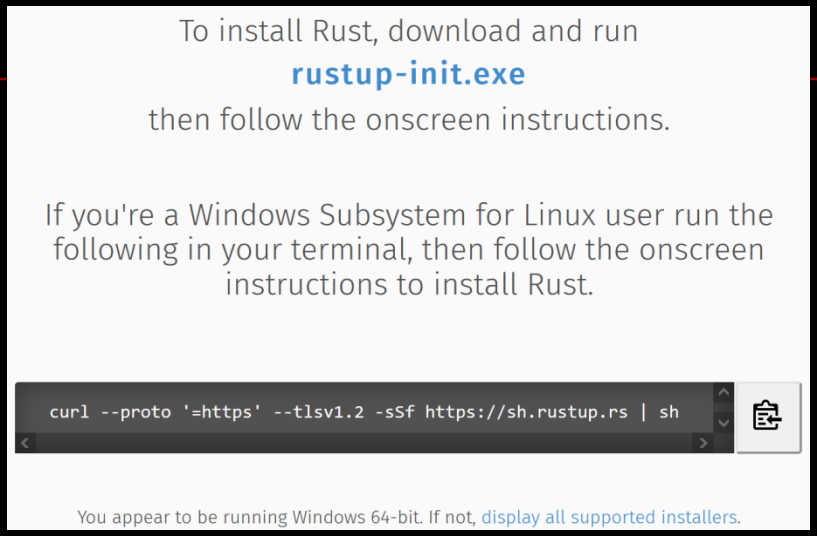
.github	Work around CI issue with windows sdk 10.0.20348.0.	last month
compiler	Auto merge of #89903 - matthiaskrgr:rollup-s0c69xl, r=matthiaskrgr	yesterday
library	Auto merge of #85379 - mdaverde:uds-abstract, r=joshtriplett	9 hours ago
src	Auto merge of #84096 - m-ou-se:windows-bcrypt-random, r=dtolnay	13 hours ago
.editorconfig	Add .editorconfig	9 months ago
.gitattributes	Remove rustfmt tests from top-level .gitattributes	4 months ago
.gitignore	Fix invalid rules in .gitignore	4 days ago
.gitmodules	Update to the final LLVM 13.0.0 release	14 days ago
.mailmap	Add davidtwco to the `.mailmap`	3 days ago
CODE_OF_CONDUCT.md	Remove the code of conduct; instead link https://www.rust-lang.org/co...	2 years ago
CONTRIBUTING.md	Make opening sentence friendlier for new contributors	7 months ago
COPYRIGHT	Rebase to the llvm-project monorepo	3 years ago
Cargo.lock	Remove unused dependencies from rustc_const_eval	yesterday
Cargo.toml	Greatly reduce amount of debuginfo compiled for bootstrap itself	5 days ago
LICENSE-APACHE	Remove appendix from LICENCE-APACHE	2 years ago
LICENSE-MIT	LICENSE-MIT: Remove inaccurate (misattributed) copyright notice	4 years ago
README.md	Rollup merge of #85504 - liigo:patch-13, r=Mark-Simulacrum	4 months ago
RELEASES.md	Drop reverted stabilizations from release notes	2 days ago
config.toml.example	Rollup merge of #89840 - wangkirin:master, r=jyn514	3 days ago
configure	Enforce Python 3 as much as possible	2 years ago
rustfmt.toml	Fix tidy	2 months ago
triagebot.toml	Rollup merge of #80543 - LeSeulArtichaut:notify-close, r=spastorino	2 months ago
x.py	Choose the version of python at runtime (portable version)	9 months ago

- **https://github.com/rust-lang/cargo**
- **https://crates.io/**

rustup

- https://rustup.rs/

  The Rust toolchain installer.



- https://github.com/rust-lang/rustup
- https://rust-lang.github.io/rustup/
- …

Rust heads into Linux Kernel

- **What's happening!**

https://lwn.net/Articles/853423/ //Rust heads into the kernel?
https://lwn.net/Articles/852704/ //Rust in the Linux kernel

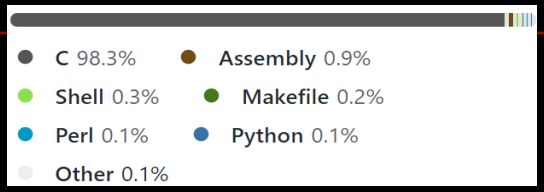
---

https://lwn.net/Articles/849849/ //Rust support hits linux-next
https://lwn.net/Articles/829858/ //Supporting Linux kernel development in Rust
…
https://www.zdnet.com/article/rust-in-the-linux-kernel-why-it-matters-and-whats-happening-next/
https://blogs.gartner.com/manjunath-bhat/2021/01/03/why-2021-will-be-a-rusty-year-for-system-programmers/
https://developers.slashdot.org/story/21/04/17/009241/linus-torvalds-says-rust-closer-for-linux-kernel-development-calls-c-a-crap-language
https://www.infoq.com/news/2021/04/rust-linux-kernel-development/
https://itwire.com/open-source/rust-support-in-linux-may-be-possible-by-5-14-release-torvalds.html
…

https://github.com/Rust-for-Linux

The goal of this project is to add support for the Rust language to the Linux kernel. This repository contains the work that will be eventually submitted for review to the LKML.

Feel free to contribute! To start, take a look at `Documentation/rust`.



Rust for Cloud Native

- https://rust-cloud-native.github.io/

Applications and Services

- apache/incubator-teaclave: open source universal secure computing platform, making computation on privacy-sensitive data safe and simple
- bottlerocket-os/bottlerocket: an operating system designed for hosting containers
- containers/krunvm: manage lightweight VMs created from OCI images
- containers/youki: a container runtime written in Rust
- datafuselabs/datafuse: A Modern Real-Time Data Processing & Analytics DBMS with Cloud-Native Architecture, built to make the Data Cloud easy
- firecracker-microvm/firecracker: secure and fast microVMs for serverless computing
- infinyon/fluvio: Cloud-native real-time data streaming platform with in-line computation capabilities
- krustlet/krustlet: Kubernetes Rust Kubelet
- kube-rs/controller-rs: a Kubernetes example controller
- kube-rs/version-rs: example Kubernetes reflector and web server
- kubewarden/policy-server: webhook server that evaluates WebAssembly policies to validate Kubernetes requests
- linkerd/linkerd2-proxy: a purpose-built proxy for the Linkerd service mesh
- openebs/mayastor: A cloud native declarative data plane in containers for containers
- rancher-sandbox/lockc: eBPF-based MAC security audit for container workloads
- tikv/tikv: distributed transactional key-value database
- tremor-rs/tremor-runtime: an event processing system that supporting complex workflows such as aggregation, rollups, an ETL language, and a query language
- valeriansaliou/sonic: fast, lightweight & schema-less search backend
- WasmEdge/WasmEdge: WasmEdge is a high-performance WebAssembly (Wasm) Virtual Machine (VM) runtime, which enables serverless functions to be embedded into any software platform; from cloud's edge to SaaS to automobiles

Libraries

- CNI Plugins: crate/framework to write CNI (container networking) plugins in Rust (includes a few custom plugins as well)
- containers/libkrun: a dynamic library providing Virtualization-based process isolation capabilities
- kube-rs/kube-rs: Kubernetes Rust client and async controller runtime
- qovery/engine: Qovery Engine is an open-source abstraction layer library that turns easy apps deployment on AWS, GCP, Azure, and other Cloud providers in just a few minutes
- open-telemetry/opentelemetry-rust: OpenTelemetry is a set of APIs, SDKs, tooling and integrations that are designed for the creation and management of telemetry data such as traces, metrics, and logs.

- ...

Rust for Android

- https://source.android.com/setup/build/rust/building-rust-modules/overview/

The Android platform provides support for developing native OS components in Rust, a modern systems-programming language that provides memory safety guarantees with performance equivalent to C/C++. Rust uses a combination of compile-time checks that enforce object lifetime and ownership, and runtime checks that ensure valid memory accesses, thereby eliminating the need for a garbage collector.

Rust provides a range of modern language features which allow developers to be more productive and confident in their code:

- **Safe concurrent programming** - The ease with which this allows users to write efficient, thread-safe code has given rise to Rust's Fearless Concurrency slogan.
- **Expressive type system** - Rust helps prevent logical programming bugs by allowing for highly expressive types (such as Newtype wrappers, and enum variants with contents).
- **Stronger Compile-time Checks** - More bugs caught at compile-time increases developer confidence that when code compiles successfully, it works as intended.
- **Built-in Testing Framework** - Rust provides a built-in testing framework where unit tests can be placed alongside the implementation they test, making unit testing easier to include.
- **Error handling enforcement** - Functions with recoverable failures can return a Result type, which will be either a success variant or an error variant. The compiler requires callers to check for and handle the error variant of a `Result` enum returned from a function call. This reduces the potential for bugs resulting from unhandled failures.
- **Initialization** - Rust requires every variable to be initialized to a legal member of its type before use, preventing an unintentional initialization to an unsafe value.
- **Safer integer handling** - All integer-type conversions are explicit casts. Developers can't accidentally cast during a function call when assigning to a variable, or when attempting to do arithmetic with other types. Overflow checking is on by default in Android for Rust, which requires overflow operations to be explicit.

For more information, see the series of blog posts on Android Rust support:

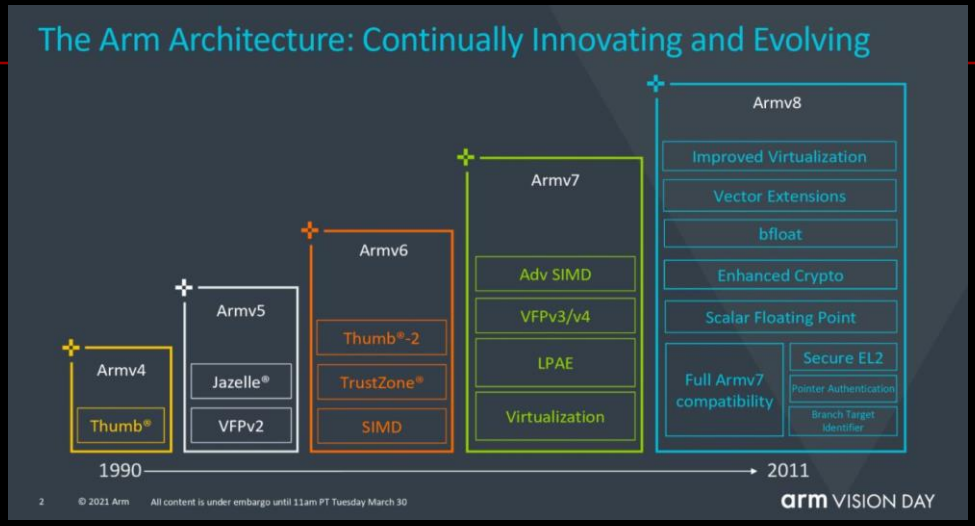
- Rust in the Android Platform
  Provides an overview on why the Android team introduced Rust as a new platform language.
- Integrating Rust into the Android Open Source Project
  Discusses how Rust support has been introduced to the build system, and why certain design decisions were made.
- Rust/C++ interop in the Android Platform
  Discusses the approach to Rust/C++ interoperability within Android.

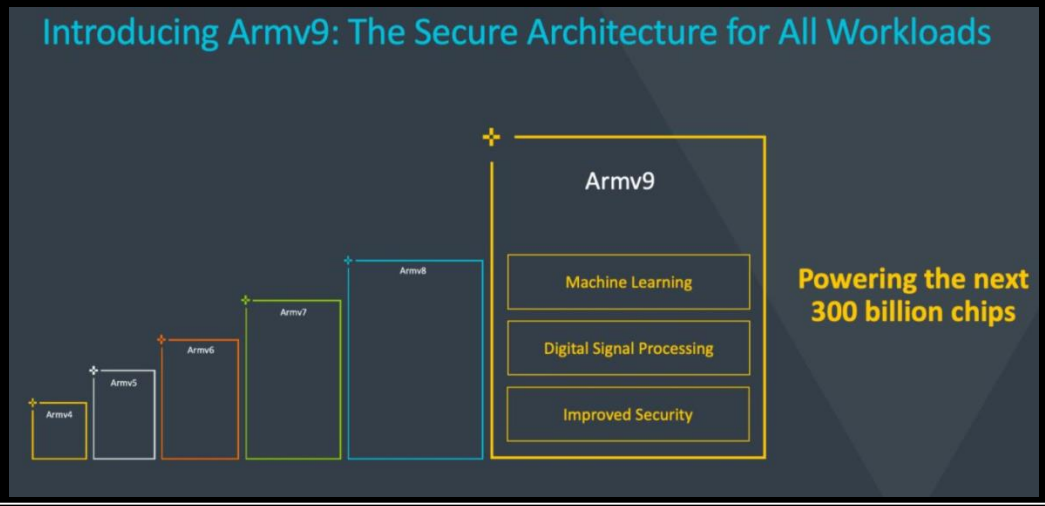
- ...

5.6 ARM Ecosystem in 2021

ARM v9

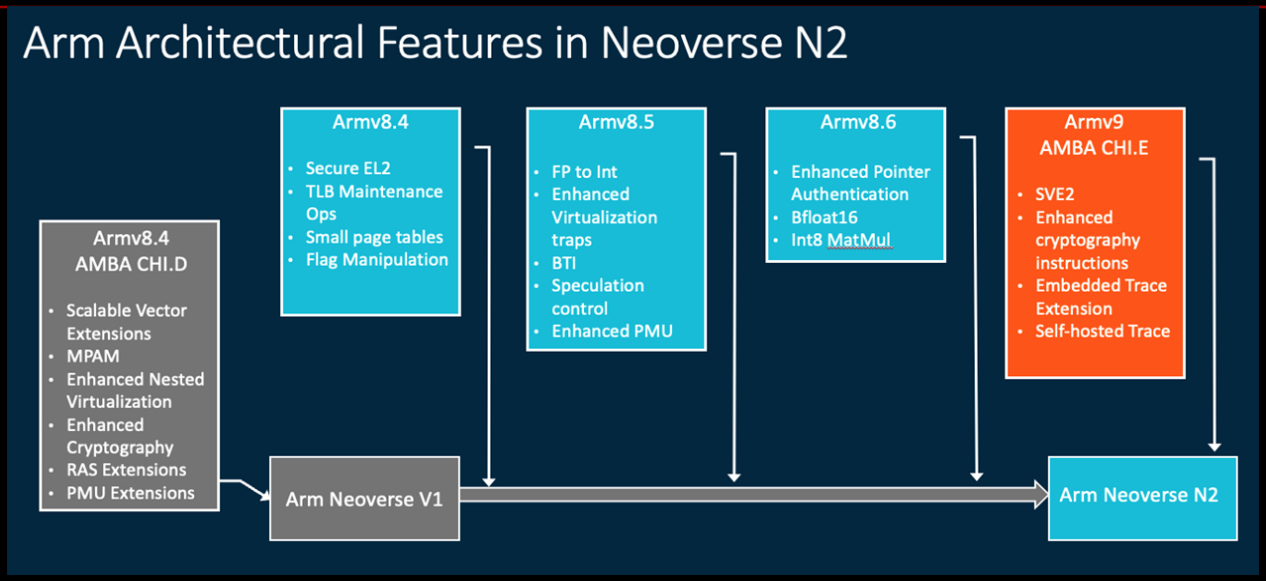
- https://www.arm.com/campaigns/arm-vision



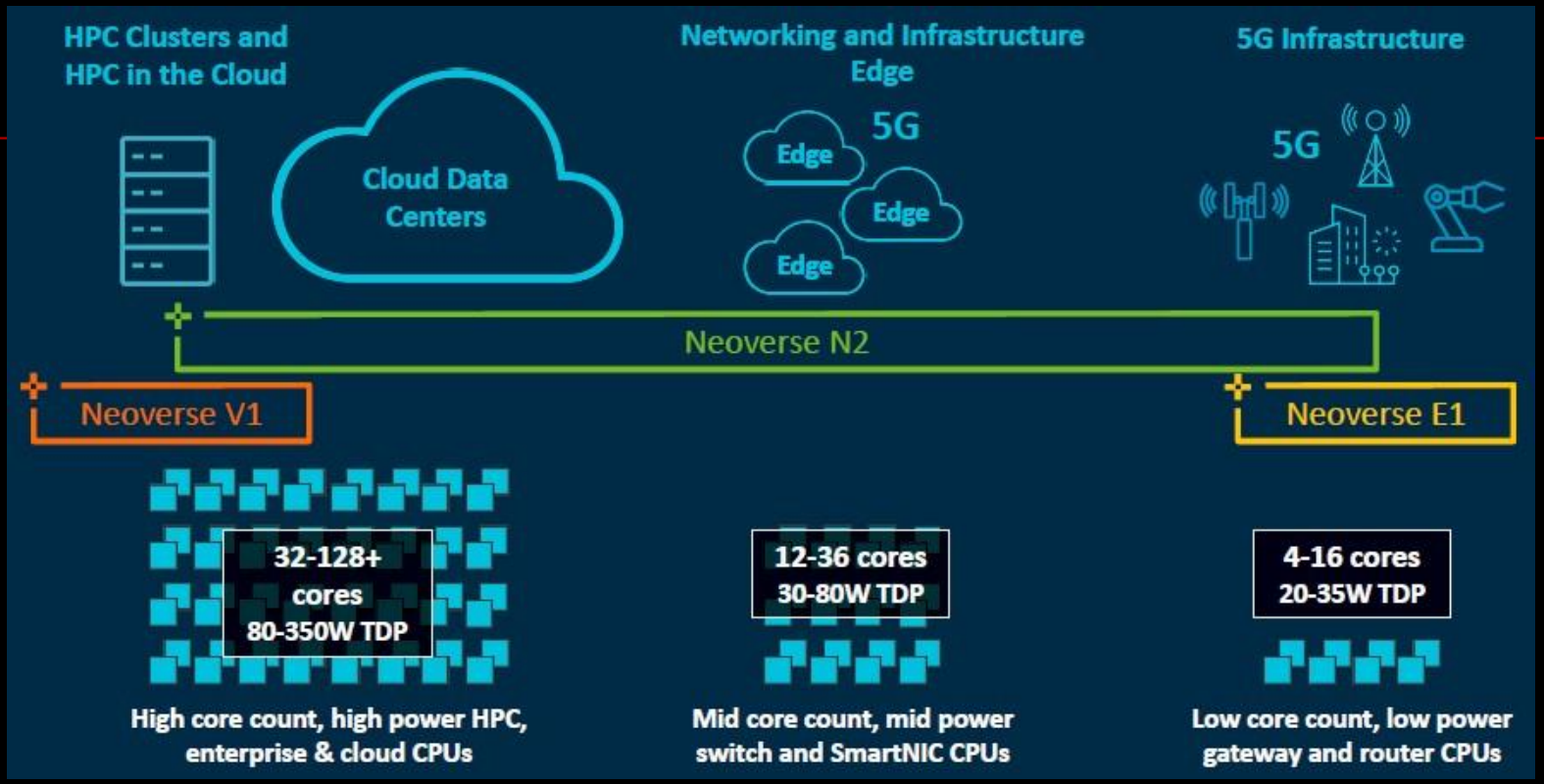


ARM for Edge Computing

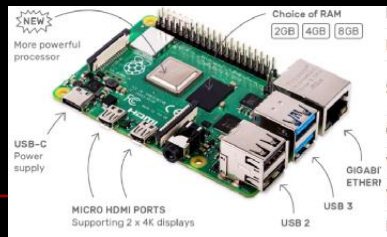
- **https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/arm-neoverse-n2-industry-leading-performance-efficiency**

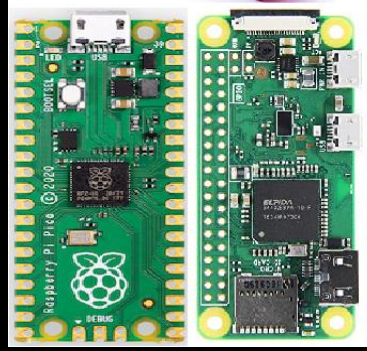


- https://www.nextplatform.com/2021/04/27/arm-puts-some-muscle-into-future-neoverse-server-cpu-designs/

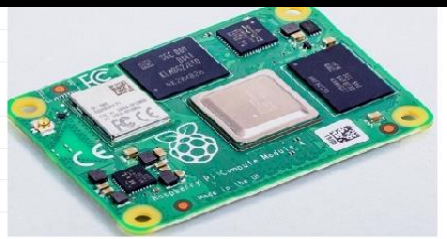


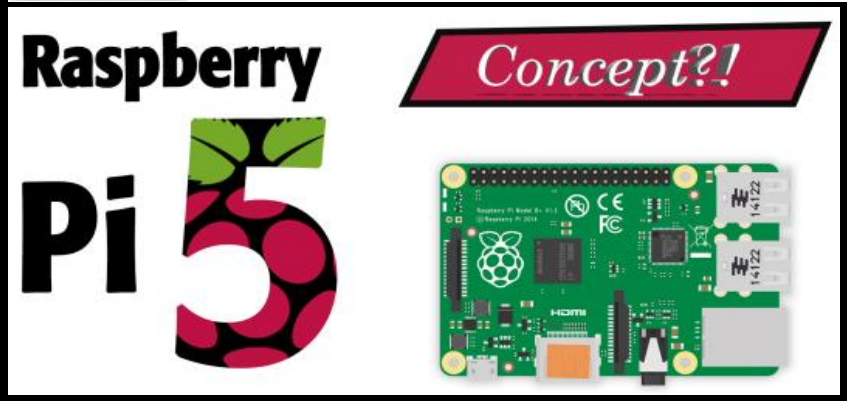
Raspberry Pi





Features/Specs	Raspberry Pi 4B	Raspberry Pi 3 B+
Release date	24th June 2019	14th March 2018
SoC	Broadcom BCM2711 quad-core Cortex-A72 @ 1.5 GHz	Broadcom BCM2837B0 quad-core Cortex-A53 @ 1.4 GHz
GPU	VideoCore VI with OpenGL ES 1.1, 2.0, 3.0	VideoCore IV with OpenGL ES 1.1, 2.0
Video Decode	H.265 4Kp60, H.264 1080p60	H.264 & MPEG-4 1080p30
Video Encode	H.264 1080p30	
Memory	1GB, 2GB, or 4GB LPDDR4	1GB LPDDR2
Storage	microSD card	
Video & Audio Output	2x micro HDMI ports up to 4Kp60 3.5mm AV port (composite + audio) MIPI DSI connector	1x HDMI 1.4 port up to 1080p60 3.5mm AV port (composite + audio) MIPI DSI connector
Camera	MIPI CSI connector	
Ethernet	Native Gigabit Ethernet	Gigabit Ethernet over USB (300 Mbps max.)
WiFi	Dual band 802.11 b/g/n/ac	
Bluetooth	Bluetooth 5.0 + BLE	Bluetooth 4.2 + BLE
USB	2x USB 3.0 + 2x USB 2.0	4x USB 2.0
Expansion	40-pin GPIO header	
Power Supply	5V via USB type-C up to 3A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT	5V via micro USB up to 2.5A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT
Dimensions	85×56 mm	
Default OS	Raspbian (after June 24, 2019)	Raspbian (after March 2018)
Price	$35 (1GB RAM), $45 (2GB RAM), $55 (4GB RAM)	$35 (1GB RAM)





Cortex-M

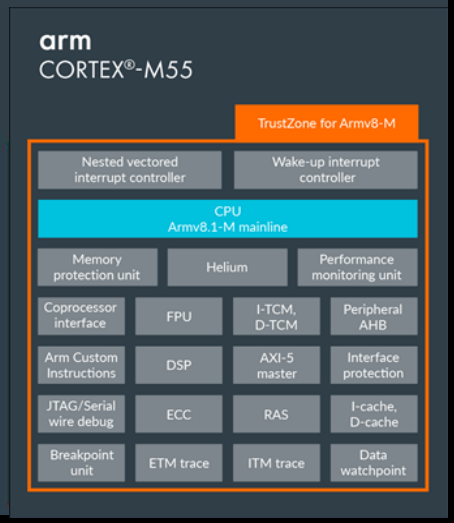
- https://www.arm.com/blogs/blueprint/tinyml
- Cortex-M55

  https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m55

First Helium and Custom Instruction capable CPU core

Cortex-M55: The Most AI-capable Cortex-M Processor

- ✓ First CPU based on Arm Helium technology
  - Energy-efficient and configurable with vector processing capabilities
  - Delivers up to 5x DSP performance and up to 15x ML performance*
  - Versatile capability for both classical ML and NN inference
- ✓ Advanced memory interfaces for fast access to ML data and weights
- ✓ Arm TrustZone security, accelerating the route to PSA Certified



- https://www.arm.com/why-arm/technologies/helium
- https://www.arm.com/why-arm/technologies/custom-instructions
- https://www.zhihu.com/question/371097288/answer/1010694311
- https://www.arm.com/blogs/blueprint/cambridge-consultants

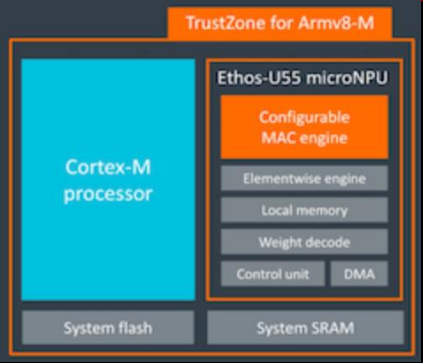
Ethos: MicroNPU for ARM

https://www.arm.com/product-filter?families=ethos%20npus&showall=true

Ethos-U55:

Ethos-U55: The First microNPU for Cortex-M

- ✓ Highest efficiency and small memory footprint
- ✓ 32, 64, 128, or 256 unit multiply-accumulate (MAC) engine
- ✓ Weight decoder and DMA for on-the-fly weight decompression
- ✓ Tooling available for offline optimization
- ✓ Works with a range of Cortex-M processors:
  - Cortex-M55
  - Cortex-M7
  - Cortex-M33
  - Cortex-M4



Key Features	Performance (At 1GHz)	64 to 512 GOP/s
	MACs (8x8)	32, 64, 128, 256
	Utilization on popular networks	Up to 85%
	Data Types	Int-8 and Int-16
	Network Support	CNN and RNN/LSTM
	Winograd Support	No
	Sparsity	Yes
Memory System	Internal SRAM	18 to 50 KB
	External On Chip SRAM	KB to Multi-MB
	Compression	Weights only
	Memory Optimizations	Extended compression, layer/operator fusion
Development Platform	Neural Frameworks	TensorFlow Lite Micro
	Operating Systems	RTOS or bare-metal
	Software Components	TensorFlow Lite Micro Runtime, CMSIS-NN, Optimizer, Driver
	Debug and Profile	Layer-by-layer visibility with PMUs
	Evaluation and Early Prototyping	Performance Model, Cycle Accurate Model, or FPGA Evaluations

5.7 Good resources for Python implementation

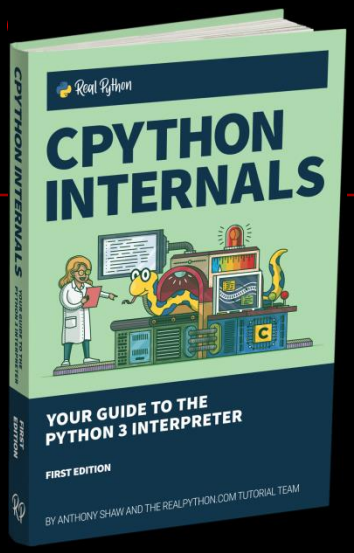
- https://wiki.python.org/moin/PythonImplementations
- https://pyfound.blogspot.com/2021/05/the-2021-python-language-summit.html
- ...

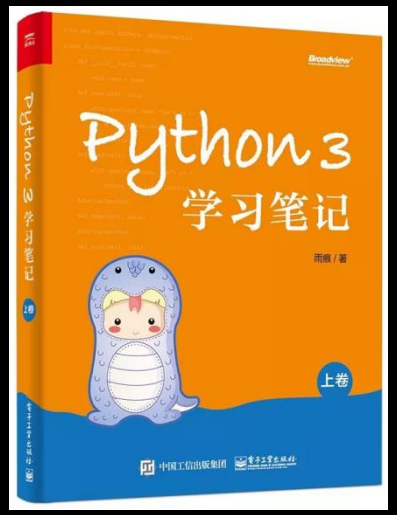
II. C-based implementation

1) Overview

- …

Good Resource





- ...

2) Make CPython Faster

Bottlenecks of current CPython implementation

- No JIT
- GIL

  https://en.wikipedia.org/wiki/Global_interpreter_lock
- …

Plan fron the Father of Python

- https://thenewstack.io/guido-van-rossums-ambitious-plans-for-improving-python-performance/

- https://zhuanlan.zhihu.com/p/399398829

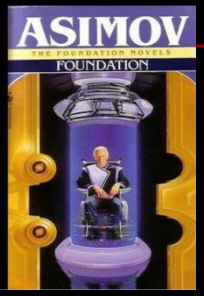
Python 之父立 Flag：明年要把 Python 速度提高 2 倍！

- https://www.zdnet.com/article/python-programming-we-want-to-make-the-language-twice-as-fast-says-its-creator/

> Microsoft currently has five core developers who contribute to the development of CPython, including Brett Cannon, Steve Dower, Guido van Rossum, Eric Snow, and Barry Warsaw — all veterans in the Python core developer community.

The “Shannon Plan”

- Posted to GitHub and python-dev last October
  - github.com/markshannon/faster-cpython
- Based on experience with “HotPy”, “HoyPy 2”
- Promises 5x in 4 years (1.5x per year)
- Looking for funding



https://github.com/markshannon/faster-cpython/blob/master/plan.md

The stages to high performance

Stage 1 -- Python 3.10

The key improvement for 3.10 will be an adaptive, specializing interpreter. The interpreter will adapt to types and values during execution, exploiting type stability in the program, without needing runtime code generation.

Stage 2 -- Python 3.11

This stage will make many improvements to the runtime and key objects. Stage two will be characterized by lots of "tweaks", rather than any "headline" improvement. The planned improvements include:

- Improved performance for integers of less than one machine word.
- Improved peformance for binary operators.
- Faster calls and returns, through better handling of frames.
- Better object memory layout and reduced memory management overhead.
- Zero overhead exception handling.
- Further enhancements to the interpreter
- Other small enhancements.

Stage 3 -- Python 3.12 (requires runtime code generation)

Simple "JIT" compiler for small regions. Compile small regions of specialized code, using a relatively simple, fast compiler.

Stage 4 -- Python 3.13 (requires runtime code generation)

Extend regions for compilation. Enhance compiler to generate superior machine code.

https://github.com/markshannon/faster-cpython/blob/master/tiers.md

Making CPython Faster

- https://pyfound.blogspot.com/2021/05/the-2021-python-language-summit-making.html

Seven months ago, Guido van Rossum left a brief retirement to work at Microsoft. He was given the freedom to pick a project and decided to work on making CPython faster. Microsoft will be funding a small team consisting of Guido van Rossum, Mark Shannon, Eric Snow, and possibly others.

The team will:

- Collaborate fully and openly with CPython's core developers
- Make incremental changes to CPython
- Take care of maintenance and support
- Keep all project-specific repos open
- Have all discussions in trackers on open GitHub repos

The team will need to work within some constraints. They'll need to keep code maintainable, not break stable ABI compatibility, not break limited API compatibility, and not break or slow down extreme cases, such as pushing a million cases on the eval stack. Although they won't be able to change the data model, they will be able to change:

- The byte code
- The compiler
- The internals of a lot of objects

Who Will Benefit?

You'll benefit from the speed increase if you:

- Run CPU-intensive pure Python code
- Use tools or websites built in CPython

You might not benefit from the speed increase if you:

- Rewrote your code in C, Cython, C++, or similar to increase speed already (e.g. NumPy, TensorFlow)
- Have code that is mostly waiting for I/O
- Use multithreading
- Need to make your code more algorithmically efficient first

Specializing Adaptive Interpreter

https://www.python.org/dev/peps/pep-0659/

Abstract

In order to perform well, virtual machines for dynamic languages must specialize the code that they execute to the types and values in the program being run. This specialization is often associated with "JIT" compilers, but is beneficial even without machine code generation.

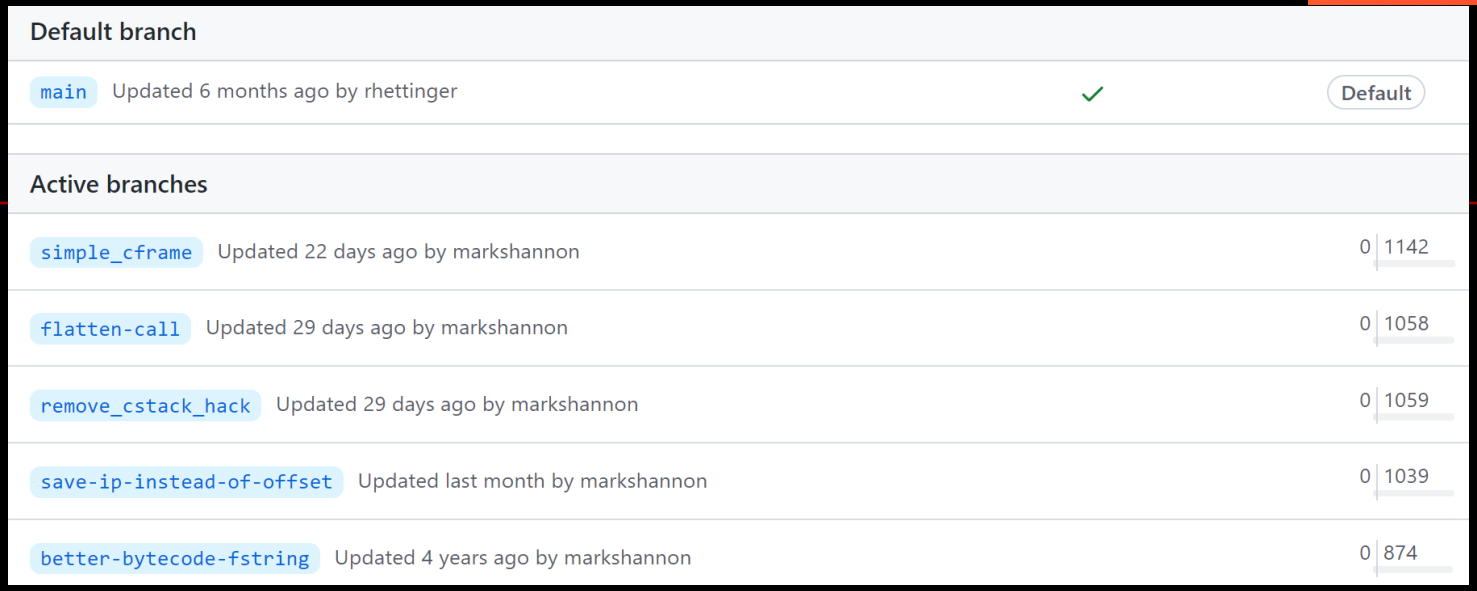
A specializing, adaptive interpreter is one that speculatively specializes on the types or values it is currently operating on, and adapts to changes in those types and values.

Specialization gives us improved performance, and adaptation allows the interpreter to rapidly change when the pattern of usage in a program alters, limiting the amount of additional work caused by mis-specialization.

This PEP proposes using a specializing, adaptive interpreter that specializes code aggressively, but over a very small region, and is able to adjust to mis-specialization rapidly and at low cost.

Adding a specializing, adaptive interpreter to CPython will bring significant performance improvements. It is hard to come up with meaningful numbers, as it depends very much on the benchmarks and on work that has not yet happened. Extensive experimentation suggests speedups of up to 50%. Even if the speedup were only 25%, this would still be a worthwhile enhancement.

- https://github.com/markshannon/cpython

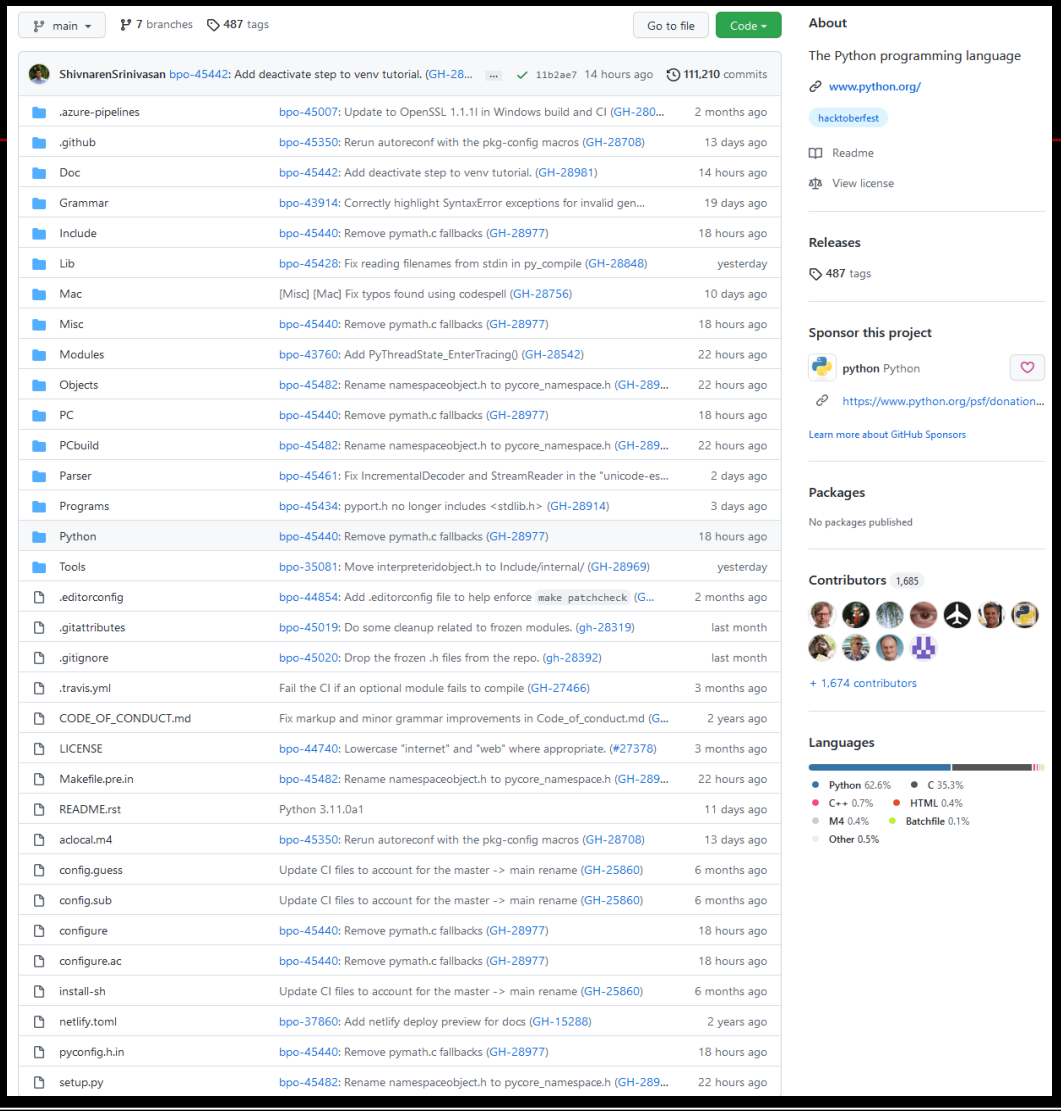


...

https://github.com/faster-cpython/ideas

Project

- https://github.com/python/cpython



main ▾ | 7 branches | 487 tags | Go to file | Code ▾

ShivnarenSrinivasan bpo-45442: Add deactivate step to venv tutorial. (GH-28... ... 11b2ae7 14 hours ago 111,210 commits

.azure-pipelines	bpo-45007: Update to OpenSSL 1.1.1l in Windows build and CI (GH-280...	2 months ago
.github	bpo-45350: Rerun autoreconf with the pkg-config macros (GH-28708)	13 days ago
Doc	bpo-45442: Add deactivate step to venv tutorial. (GH-28981)	14 hours ago
Grammar	bpo-43914: Correctly highlight SyntaxError exceptions for invalid gen...	19 days ago
Include	bpo-45440: Remove pymath.c fallbacks (GH-28977)	18 hours ago
Lib	bpo-45428: Fix reading filenames from stdin in py_compile (GH-28848)	yesterday
Mac	[Misc] [Mac] Fix typos found using codespell (GH-28756)	10 days ago
Misc	bpo-45440: Remove pymath.c fallbacks (GH-28977)	18 hours ago
Modules	bpo-43760: Add PyThreadState_EnterTracing() (GH-28542)	22 hours ago
Objects	bpo-45482: Rename namespaceobject.h to pycore_namespace.h (GH-289...	22 hours ago
PC	bpo-45440: Remove pymath.c fallbacks (GH-28977)	18 hours ago
PCbuild	bpo-45482: Rename namespaceobject.h to pycore_namespace.h (GH-289...	22 hours ago
Parser	bpo-45461: Fix IncrementalDecoder and StreamReader in the "unicode-es...	2 days ago
Programs	bpo-45434: pyport.h no longer includes <stdlib.h> (GH-28914)	3 days ago
Python	bpo-45440: Remove pymath.c fallbacks (GH-28977)	18 hours ago
Tools	bpo-35081: Move interpreteridobject.h to Include/internal/ (GH-28969)	yesterday
.editorconfig	bpo-44854: Add .editorconfig file to help enforce `make patchcheck` (G...	2 months ago
.gitattributes	bpo-45019: Do some cleanup related to frozen modules. (gh-28319)	last month
.gitignore	bpo-45020: Drop the frozen .h files from the repo. (gh-28392)	last month
.travis.yml	Fail the CI if an optional module fails to compile (GH-27466)	3 months ago
CODE_OF_CONDUCT.md	Fix markup and minor grammar improvements in Code_of_conduct.md (G...	2 years ago
LICENSE	bpo-44740: Lowercase "internet" and "web" where appropriate. (#27378)	3 months ago
Makefile.pre.in	bpo-45482: Rename namespaceobject.h to pycore_namespace.h (GH-289...	22 hours ago
README.rst	Python 3.11.0a1	11 days ago
aclocal.m4	bpo-45350: Rerun autoreconf with the pkg-config macros (GH-28708)	13 days ago
config.guess	Update CI files to account for the master -> main rename (GH-25860)	6 months ago
config.sub	Update CI files to account for the master -> main rename (GH-25860)	6 months ago
configure	bpo-45440: Remove pymath.c fallbacks (GH-28977)	18 hours ago
configure.ac	bpo-45440: Remove pymath.c fallbacks (GH-28977)	18 hours ago
install-sh	Update CI files to account for the master -> main rename (GH-25860)	6 months ago
netlify.toml	bpo-37860: Add netlify deploy preview for docs (GH-15288)	2 years ago
pyconfig.h.in	bpo-45440: Remove pymath.c fallbacks (GH-28977)	18 hours ago
setup.py	bpo-45482: Rename namespaceobject.h to pycore_namespace.h (GH-289...	22 hours ago

About

The Python programming language

www.python.org/

hacktoberfest

Readme

View license

Releases

487 tags

Sponsor this project

python Python

https://www.python.org/psf/donation...

Learn more about GitHub Sponsors

Packages

No packages published

Contributors 1,685

+ 1,674 contributors

Languages

Python 62.6% C 35.3% C++ 0.7% HTML 0.4% M4 0.4% Batchfile 0.1% Other 0.5%

Stats

```
[mydev@fedora cpython-main]$ git branch
* main
[mydev@fedora cpython-main]$ tokei
```

Language	Files	Lines	Code	Comments	Blanks
Assembly	1	46	18	26	2
GNU Style Assembly	5	1930	1421	305	204
Autoconf	13	4797	2942	959	896
BASH	2	56	44	4	8
Batch	32	2207	1845	7	355
C	330	478375	371752	57125	49498
C Header	423	207992	180209	11445	16338
C Shell	1	26	12	5	9
C++	5	4191	3479	216	496
CSS	1	112	81	6	25
D	5	83	74	1	8
Fish	1	66	40	13	13
INI	2	197	121	28	48
JavaScript	1	53	46	6	1
JSON	8	16269	16266	0	3
Lisp	1	692	502	81	109
Makefile	5	454	326	43	85
Markdown	3	178	0	130	48
Module-Definition	6	592	557	12	23
MSBuild	11	1164	967	97	100
Objective-C	7	794	635	61	98
PowerShell	7	618	358	195	65
Prolog	1	24	24	0	0
Python	2031	874843	703166	58906	112771
ReStructuredText	635	363900	257789	0	106111
Shell	6	889	550	218	121
SVG	9	9	9	0	0
Plain Text	156	110771	0	107930	2841
TOML	2	13	13	0	0
VBScript	1	1	0	1	0
Visual Studio Pro|	50	6612	6560	11	41
Visual Studio Sol|	2	1535	1534	0	1
XSL	1	5	5	0	0
XML	59	31334	31258	18	58
HTML	11	2075	1961	6	108
|- CSS	3	17	17	0	0
|- JavaScript	3	53	44	4	5
(Total)		2145	2022	10	113
Total	3834	2112973	1584625	237859	290489

```
[mydev@fedora cpython-main]$
```

...

3) Cinder

- https://github.com/facebookincubator/cinder

  Instagram's performance oriented fork of CPython.

> Cinder is Instagram's internal performance-oriented production version of CPython 3.8. It contains a number of performance optimizations, including bytecode inline caching, eager evaluation of coroutines, a method-at-a-time JIT, and an experimental bytecode compiler that uses type annotations to emit type-specialized bytecode that performs better in the JIT.
>
> For more information on CPython, see `README.cpython.rst`.

- Docker-based Dev Env
- 

  Source: https://www.viehland.com/PyCon_2021.pdf

- Currently only support X64.

What's Special about it?

Immortal Instances

Instagram uses a multi-process webserver architecture; the parent process starts, performs initialization work (e.g. loading code), and forks tens of worker processes to handle client requests. Worker processes are restarted periodically for a number of reasons (e.g. memory leaks, code deployments) and have a relatively short lifetime. In this model, the OS must copy the entire page containing an object that was allocated in the parent process when the object's reference count is modified. In practice, the objects allocated in the parent process outlive workers; all the work related to reference counting them is unnecessary.

Instagram has a very large Python codebase and the overhead due to copy-on-write from reference counting long-lived objects turned out to be significant. We developed a solution called "immortal instances" to provide a way to opt-out objects from reference counting. See Include/object.h for details. This feature is controlled by defining Py_IMMORTAL_INSTANCES and is enabled by default in Cinder. This was a large win for us in production (~5%), but it makes straight-line code slower. Reference counting operations occur frequently and must check whether or not an object participates in reference counting when this feature is enabled.

Shadowcode

"Shadowcode" or "shadow bytecode" is our inline caching implementation. It observes particular optimizable cases in the execution of generic Python opcodes and (for hot functions) dynamically replaces those opcodes with specialized versions. The core of shadowcode lives in `Python/shadowcode.c`, though the implementations for the specialized bytecodes are in `Python/ceval.c` with the rest of the eval loop. Shadowcode-specific tests are in `Lib/test/test_shadowcode.py`.

Eager coroutine evaluation

If a call to an async function is immediately awaited, we immediately execute the called function up to its first `await`. If the called function reaches a `return` without needing to await, we will be able to return that value directly without ever even creating a coroutine object or deferring to the event loop. This is a significant (~5%) CPU optimization in our async-heavy workload.

This is mostly implemented in `Python/ceval.c`, via a new vectorcall flag `_Py_AWAITED_CALL_MARKER`, indicating the caller is immediately awaiting this call. Look for uses of the `IS_AWAITED()` macro and this vectorcall flag, as well as the `_PyEval_EvalEagerCoro` function.

Strict Modules

Strict modules is a few things rolled into one:

1. A static analyzer capable of validating that executing a module's top-level code will not have side effects visible outside that module.

2. An immutable `StrictModule` type usable in place of Python's default module type.

3. A Python module loader capable of recognizing modules opted in to strict mode (via an `import __strict__` at the top of the module), analyzing them to validate no import side effects, and populating them in `sys.modules` as a `StrictModule` object.

The version of strict modules that we currently use in production is written in Python and is not part of Cinder. The `StrictModules/` directory in Cinder is an in-progress C++ rewrite of the import side effects analyzer.

Static Python

Static Python is an experimental bytecode compiler that makes use of type annotations to emit type-specialized and type-checked Python bytecode. Used along with the Cinder JIT, it can deliver performance similar to MyPyC or Cython in many cases, while offering a pure-Python developer experience (normal Python syntax, no extra compilation step). Static Python plus Cinder JIT achieves 7x the performance of stock CPython on a typed version of the Richards benchmark. At Instagram we have successfully used Static Python in production to replace the majority of the Cython modules in our primary webserver codebase, with no performance regression.

The Static Python compiler is built on top of the Python `compiler` module that was removed from the standard library in Python 3 and has since been maintained and updated externally; this compiler is incorporated into Cinder in `Lib/compiler`. The Static Python compiler is implemented in `Lib/compiler/static/`, and its tests are in `Lib/test/test_compiler/test_static.py`.

Classes defined in Static Python modules are automatically given typed slots (based on inspection of their typed class attributes and annotated assignments in `__init__`), and attribute loads and stores against instances of these types use new `STORE_FIELD` and `LOAD_FIELD` opcodes, which in the JIT become direct loads/stores from/to a fixed memory offset in the object, with none of the indirection of a `LOAD_ATTR` or `STORE_ATTR`. Classes also gain vtables of their methods, for use by the `INVOKE_*` opcodes mentioned below. The runtime support for these features is located in `Include/classloader.h` and `Python/classloader.c`.

A static Python function begins with a new `CHECK_ARGS` opcode which checks that the supplied arguments' types match the type annotations, and raises `TypeError` if not. Calls from a static Python function to another static Python function will skip this opcode (since the types are already validated by the compiler). Static to static calls can also avoid much of the overhead of a typical Python function call. We emit an `INVOKE_FUNCTION` or `INVOKE_METHOD` opcode which carries with it metadata about the called function or method; this plus optionally immutable modules (via `StrictModule`) and types (via `cinder.freeze_type()`, which we currently apply to all types in strict and static modules in our import loader, but in future may become an inherent part of Static Python) and compile-time knowledge of the callee signature allow us to (in the JIT) turn many Python function calls into direct calls to a fixed memory address using the x64 calling convention, with little more overhead than a C function call.

Static Python is still gradually typed, and supports code that is only partially annotated or uses unknown types by falling back to normal Python dynamic behavior. In some cases (e.g. when a value of statically-unknown type is returned from a function with a return annotation), a runtime `CAST` opcode is inserted which will raise `TypeError` if the runtime type does not match the expected type.

Static Python also supports new types for machine integers, bools, doubles, and vectors/arrays. In the JIT these are handled as unboxed values, and e.g. primitive integer arithmetic avoids all Python overhead. Some operations on builtin types (e.g. list or dictionary subscript or `len()`) are also optimized.

Cinder doesn't currently come bundled with a module loader that is able to automatically detect static modules and load them as static with cross-module compilation; we currently do this via our strict/static import loader which is not part of Cinder. Currently the best way to experiment with static python in Cinder is to use `./python -m compiler --static some_module.py`, which will compile the module as static Python and execute it.

See `CinderDoc/static_python.rst` for more detailed documentation.

JIT

The Cinder JIT is a method-at-a-time custom JIT implemented in C++. It is enabled via the `-X jit` flag or the `PYTHONJIT=1` environment variable. It supports almost all Python opcodes, and can achieve 1.5-4x speed improvements on many Python performance benchmarks.

By default when enabled it will JIT-compile every function that is ever called, which may well make your program slower, not faster, due to overhead of JIT-compiling rarely-called functions. The option `-X jit-list-file=/path/to/jitlist.txt` or `PYTHONJITLISTFILE=/path/to/jitlist.txt` can point it to a text file containing fully qualified function names (in the form `path.to.module:funcname` or `path.to.module:ClassName.method_name`), one per line, which should be JIT-compiled. We use this option to compile only a set of hot functions derived from production profiling data. (A more typical approach for a JIT would be to dynamically compile functions as they are observed to be called frequently. It hasn't yet been worth it for us to implement this, since our production architecture is a pre-fork webserver, and for memory sharing reasons we wish to do all of our JIT compiling up front in the initial process before workers are forked, which means we can't observe the workload in-process before deciding which functions to JIT-compile.)

The JIT lives in the `Jit/` directory, and its C++ tests live in `RuntimeTests/` (run these with `make testruntime`). There are also some Python tests for it in `Lib/test/test_cinderjit.py`; these aren't meant to be exhaustive, since we run the entire CPython test suite under the JIT via `make testcinder_jit`; they cover JIT edge cases not otherwise found in the CPython test suite.

See `Jit/pyjit.cpp` for some other `-X` options and environment variables that influence the behavior of the JIT. There is also a `cinderjit` module defined in that file which exposes some JIT utilities to Python code (e.g. forcing a specific function to compile, checking if a function is compiled, disabling the JIT). Note that `cinderjit.disable()` only disables future compilation; it immediately compiles all known functions and keeps existing JIT-compiled functions.

The JIT first lowers Python bytecode to a high-level intermediate representation (HIR); this is implemented in `Jit/hir/`. HIR maps reasonably closely to Python bytecode, though it is a register machine instead of a stack machine, it is a bit lower level, it is typed, and some details that are obscured by Python bytecode but important for performance (notably reference counting) are exposed explicitly in HIR. HIR is transformed into SSA form, some optimization passes are performed on it, and then reference counting operations are automatically inserted into it according to metadata about the refcount and memory effects of HIR opcodes.

HIR is then lowered to a low-level intermediate representation (LIR), which is an abstraction over assembly, implemented in `Jit/lir/`. In LIR we do register allocation, some additional optimization passes, and then finally LIR is lowered to assembly (in `Jit/codegen/`) using the excellent asmjit library.

The JIT is in its early stages. While it can already eliminate interpreter loop overhead and offers significant performance improvements for many functions, we've only begun to scratch the surface of possible optimizations. Many common compiler optimizations are not yet implemented. Our prioritization of optimizations is largely driven by the characteristics of the Instagram production workload.

Asmjit

https://asmjit.com/

GitHub Projects

AsmJit project, as the name implies, started as a library to allow JIT code generation and execution. However, AsmJit evolved and now contains features that are far beyond the initial scope. AsmJit now consists of multiple projects:

- AsmJit - AsmJit library, the main project.
- AsmTK - AsmTK library, toolkit that implements some functionality on top of AsmJit, for example assembler parser.
- AsmDB - Assembler database in JSON format, used to generate AsmJit tables.

Online Tools

- AsmGrid - A grid view of assembler instructions (AsmDB) and their latencies (generated by CULT tool).

Highlights

- Lightweight - ~300kB compiled binary with all built-in features depending on C++ compiler and optimization level.
- Modular - Unneeded features can be disabled at compile-time to save space.
- Zero dependencies - No external libraries nor STL containers are used, easy to embed and/or link statically.
- No exceptions & RTTI - AsmJit doesn't use exceptions, but allows to attach a throwable ErrorHandler if required.

Key Features

- Complete X86/X64 instruction set - MMX, SSE+, BMI+, AVX+, FMA+, AVX-512+, AMX, privileged instructions, and other recently added ISA extensions are supported.
- Dynamic architecture that allows users to construct instructions and operands at runtime - to inspect them, to validate them, and to emit them.
- Different emitters providing various abstraction levels - Assembler, Builder, and Compiler.
- Support for sections that can be used to separate code and data or to use separate buffers during code generation.
- Built-in logging (includes formatting) and user-friendly error handling.
- JIT memory allocator with malloc-like API for JIT code generation and execution.
- Compiler can be used to emit large chunks of code with the help of a built-in register allocator.

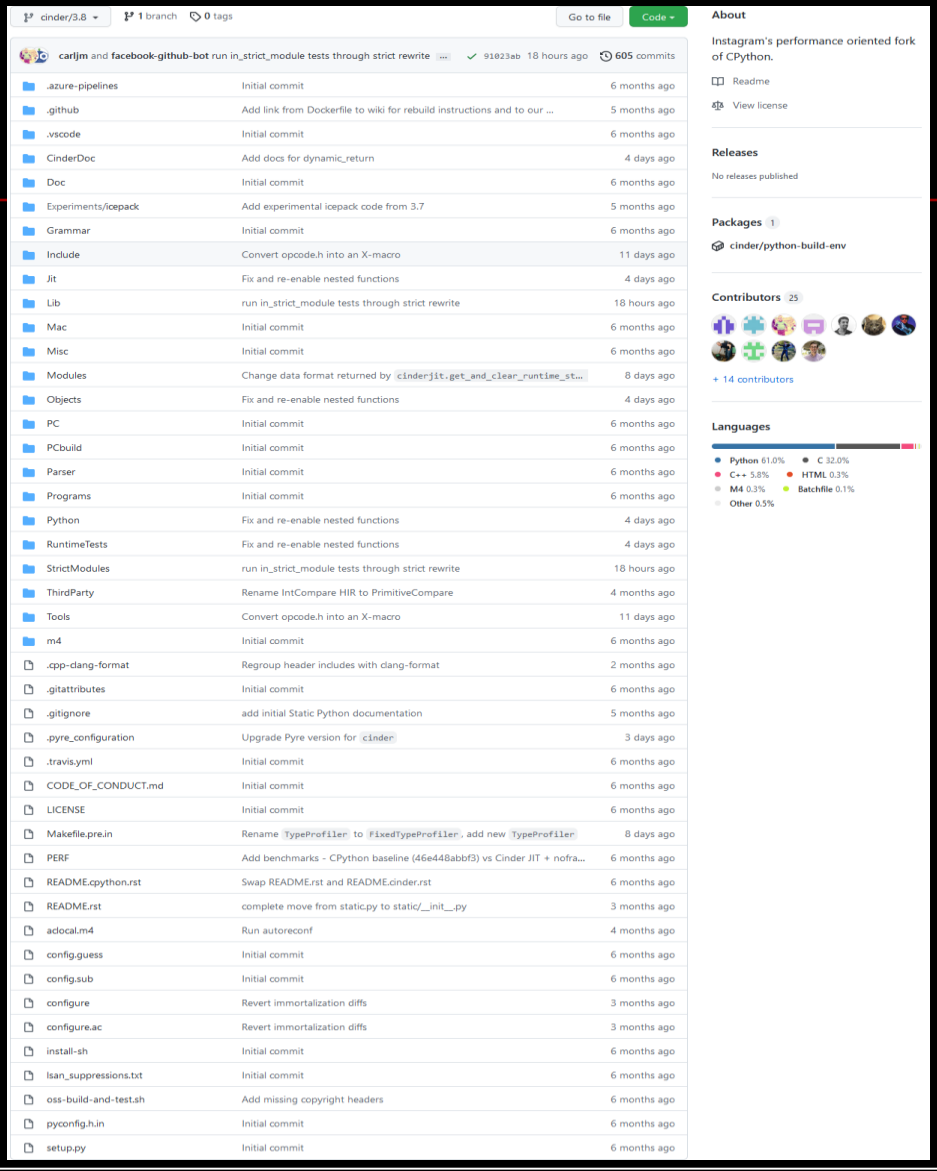
Open Source Using AsmJit

- Blend2D - A high performance 2D vector graphics engine written in C++ (git).
- CULT - A tool that can be used to measure instruction latencies in user-space.
- Erlang OTP - Erlang OTP uses AsmJit in its BEAM JIT compiler (git, blog).
- FBGEMM - Facebook GEneral Matrix Multiplication library.
- GZDoom - Doom engine, which uses AsmJit for JIT compilation of ZScript code (git).
- MathPresso - Mathematical expression parser and JIT compiler (example of using AsmJit).
- X64dbg - An open-source x64/x32 debugger for Windows (git).
- (this is not a complete list, there are many other projects using AsmJit)

https://github.com/asmjit

AsmJit was initially developed for X86 and X86_64 architectures, however, an experimental AArch64 backend is already in aarch64 branch.

Project



cinder/3.8 · 1 branch · 0 tags · Go to file · Code

carljm and facebook-github-bot run in_strict_module tests through strict rewrite ... 91023ab 18 hours ago 605 commits

.azure-pipelines	Initial commit	6 months ago
.github	Add link from Dockerfile to wiki for rebuild instructions and to our ...	5 months ago
.vscode	Initial commit	6 months ago
CinderDoc	Add docs for dynamic_return	4 days ago
Doc	Initial commit	6 months ago
Experiments/icepack	Add experimental icepack code from 3.7	5 months ago
Grammar	Initial commit	6 months ago
Include	Convert opcode.h into an X-macro	11 days ago
Jit	Fix and re-enable nested functions	4 days ago
Lib	run in_strict_module tests through strict rewrite	18 hours ago
Mac	Initial commit	6 months ago
Misc	Initial commit	6 months ago
Modules	Change data format returned by `cinderjit.get_and_clear_runtime_st...`	8 days ago
Objects	Fix and re-enable nested functions	4 days ago
PC	Initial commit	6 months ago
PCbuild	Initial commit	6 months ago
Parser	Initial commit	6 months ago
Programs	Initial commit	6 months ago
Python	Fix and re-enable nested functions	4 days ago
RuntimeTests	Fix and re-enable nested functions	4 days ago
StrictModules	run in_strict_module tests through strict rewrite	18 hours ago
ThirdParty	Rename IntCompare HIR to PrimitiveCompare	4 months ago
Tools	Convert opcode.h into an X-macro	11 days ago
m4	Initial commit	6 months ago
.cpp-clang-format	Regroup header includes with clang-format	2 months ago
.gitattributes	Initial commit	6 months ago
.gitignore	add initial Static Python documentation	5 months ago
.pyre_configuration	Upgrade Pyre version for `cinder`	3 days ago
.travis.yml	Initial commit	6 months ago
CODE_OF_CONDUCT.md	Initial commit	6 months ago
LICENSE	Initial commit	6 months ago
Makefile.pre.in	Rename `TypeProfiler` to `FixedTypeProfiler`, add new `TypeProfiler`	8 days ago
PERF	Add benchmarks - CPython baseline (46e448abbf3) vs Cinder JIT + nofra...	6 months ago
README.cpython.rst	Swap README.rst and README.cinder.rst	6 months ago
README.rst	complete move from static.py to static/__init__.py	3 months ago
aclocal.m4	Run autoreconf	4 months ago
config.guess	Initial commit	6 months ago
config.sub	Initial commit	6 months ago
configure	Revert immortalization diffs	3 months ago
configure.ac	Revert immortalization diffs	3 months ago
install-sh	Initial commit	6 months ago
lsan_suppressions.txt	Initial commit	6 months ago
oss-build-and-test.sh	Add missing copyright headers	6 months ago
pyconfig.h.in	Initial commit	6 months ago
setup.py	Initial commit	6 months ago

About

Instagram's performance oriented fork of CPython.

Readme

View license

Releases

No releases published

Packages 1

cinder/python-build-env

Contributors 25

+ 14 contributors

Languages

Python 61.0% · C 32.0% · C++ 5.8% · HTML 0.3% · M4 0.3% · Batchfile 0.1% · Other 0.5%

Stats

```
[mydev@fedora cinder-3.8]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Assembly                1           48           18           26            4
 GNU Style Assembly      5         1930         1421          305          204
 Autoconf               22         5716         3653         1072          991
 Automake                3          572          419           80           73
 BASH                    3           76           59            5           12
 Batch                  34         2146         1789            4          353
 C                     332       465288       359060        55995        50233
 C Header              669       315810       248438        36014        31358
 CMake                  10         1610         1052          331          227
 C Shell                 1           37           21            7            9
 C++                   330       156455       118128        16630        21697
 CSS                    11         1411          961          118          332
 D                       7          107           90            5           12
 Fish                    1           75           47           15           13
 Haskell                 1           65           55            1            9
 INI                     2          202          118           38           46
 JavaScript             42        24779        18486         3000         3293
 JSON                    9         9349         9346            0            3
 Lisp                    1          692          502           81          109
 Makefile                6          450          313           50           87
 Module-Definition       8         1401         1364           14           23
 MSBuild                12         1076          896           95           85
 Objective-C             7          794          635           61           98
 PowerShell              6          610          352          194           64
 Prolog                  1           24           24            0            0
 Python               2324       893219       718184        59821       115214
 ReStructuredText      595       324417       229586            0        94831
 Shell                  27         1458          725          554          179
 SVG                    11          338          334            0            4
 Plain Text            170       111921            0       108411         3510
 VBScript                1            1            0            1            0
 Visual Studio Pro|     65         9162         9108            9           45
 Visual Studio Sol|      7         1706         1702            0            4
 XSL                     1            5            5            0            0
 Xcode Config            6          146           33           81           32
 XML                    58          417          357            7           53
 YAML                    1          154          125            9           20
-------------------------------------------------------------------------------
 HTML                   18         2745         2076          530          139
 |- CSS                  7          172          171            0            1
 |- JavaScript           5          206          183            3           20
 (Total)                            3123         2430          533          160
-------------------------------------------------------------------------------
 Markdown               34        11962            0         8978         2984
 |- C++                  6         2138         1412          318          408
 |- JSON                 1          121          121            0            0
 |- Shell                1            1            1            0            0
 |- XML                  1           27           27            0            0
 (Total)                           14249         1561         9296         3392
===============================================================================
 Total                4842      2351039      1731397       292863       326779
===============================================================================
[mydev@fedora cinder-3.8]$
```

...

CPython Improvements at Instagram

- https://pyfound.blogspot.com/2021/05/the-2021-python-language-summit-cpython.html

...

Experimental Work

Instagram has tried some experimental changes as well. One big one was the JIT. They have a custom method at a time JIT. There is nearly full coverage for all of the opcodes. They do have some unsupported opcodes, but they are rare and not used in methods, such as `IMPORT_STAR`. There are a couple of intermediate representations. The front end lowers to an HIR where they do an SSA and have a ref count insertion pass as well as other optimization passes. After they go though the HIR level, they lower it to an LIR, which is closer to x64.

Another experimental idea is something they call static Python. It provides similar performance gains as MyPyC or Cython, but it works at runtime and has no extra compile steps. It starts with a new source loader that loads files marked with `import __static__`, and it supports cross module compilation across different source files. There are also new byte codes such as `INVOKE_FUNCTION` and `LOAD_FIELD` that can be bound tightly at runtime. It uses normal PEP 484 annotations.

InterOp needs to enforce types at the boundaries between untyped Python and static Python. If you call a typed function, then you might get a `TypeError`. Static Python has a whole new static compiler that uses the regular Python `ast` module and is based on the Python 2.x `compiler` package.

In addition, Pyro is an unannounced, experimental, from-scratch implementation that reuses the standard library. The main differences between Pyro and CPython are:

- Compacting garbage collection
- Tagged pointers
- Hidden classes

The C API is emulated for the PEP 384 subset for supporting C extensions.

...

Performance Results

Production improvements are difficult to measure because changes have been incremental over time, but they are estimated at between 20% and 30% overall. When Instagram was benchmarking, they used CPython 3.8 as the baseline and compared Cinder, Cinder with the JIT, and Cinder JIT noframe, which Instagram is not yet using in production but wants to move towards so they won't have to create Python frame objects for jitted code.

Cinder had good results on a large set of benchmarks and a 4x return on `richards` but did worse with others, particularly `2to3`, `python_startup`, and `python_startup_no_site`. This was probably because they JIT every single function when it's invoked the first time. They haven't yet made the changes to JIT a function when it becomes hot. They also haven't yet tested comparisons with Pypy.

III. Java-based implementation

1) Overview

- …

2) Jython

- http://www.jython.org //No new release since 2015…
-

Jython is a Java implementation of Python that combines expressive power with clarity. Jython is freely available for both commercial and non-commercial use and is distributed with source code under the PSF License v2. Jython is complementary to Java and is especially suited for the following tasks:

- Embedded scripting - Java programmers can add the Jython libraries to their system to allow end users to write simple or complicated scripts that add functionality to the application.
- Interactive experimentation - Jython provides an interactive interpreter that can be used to interact with Java packages or with running Java applications. This allows programmers to experiment and debug any Java system using Jython.
- Rapid application development - Python programs are typically 2-10x shorter than the equivalent Java program. This translates directly to increased programmer productivity. The seamless interaction between Python and Java allows developers to freely mix the two languages both during development and in shipping products.

Here is an example of running Python code inside a simple Java application

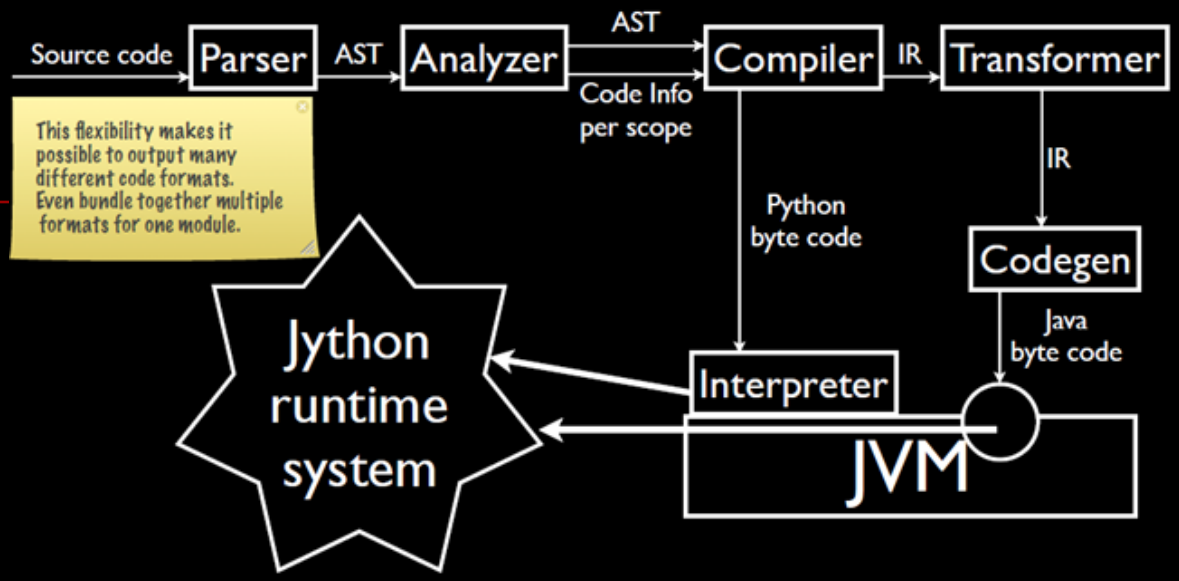
```
import org.python.util.PythonInterpreter;

public class JythonHelloWorld {
  public static void main(String[] args) {
    try(PythonInterpreter pyInterp = new PythonInterpreter()) {
      pyInterp.exec("print('Hello Python World!')");
    }
  }
}
```

Here is an example of using Java from Python code

```
from java.lang import System # Java import

print('Running on Java version: ' + System.getProperty('java.version'))
print('Unix time from Java: ' + str(System.currentTimeMillis()))
```

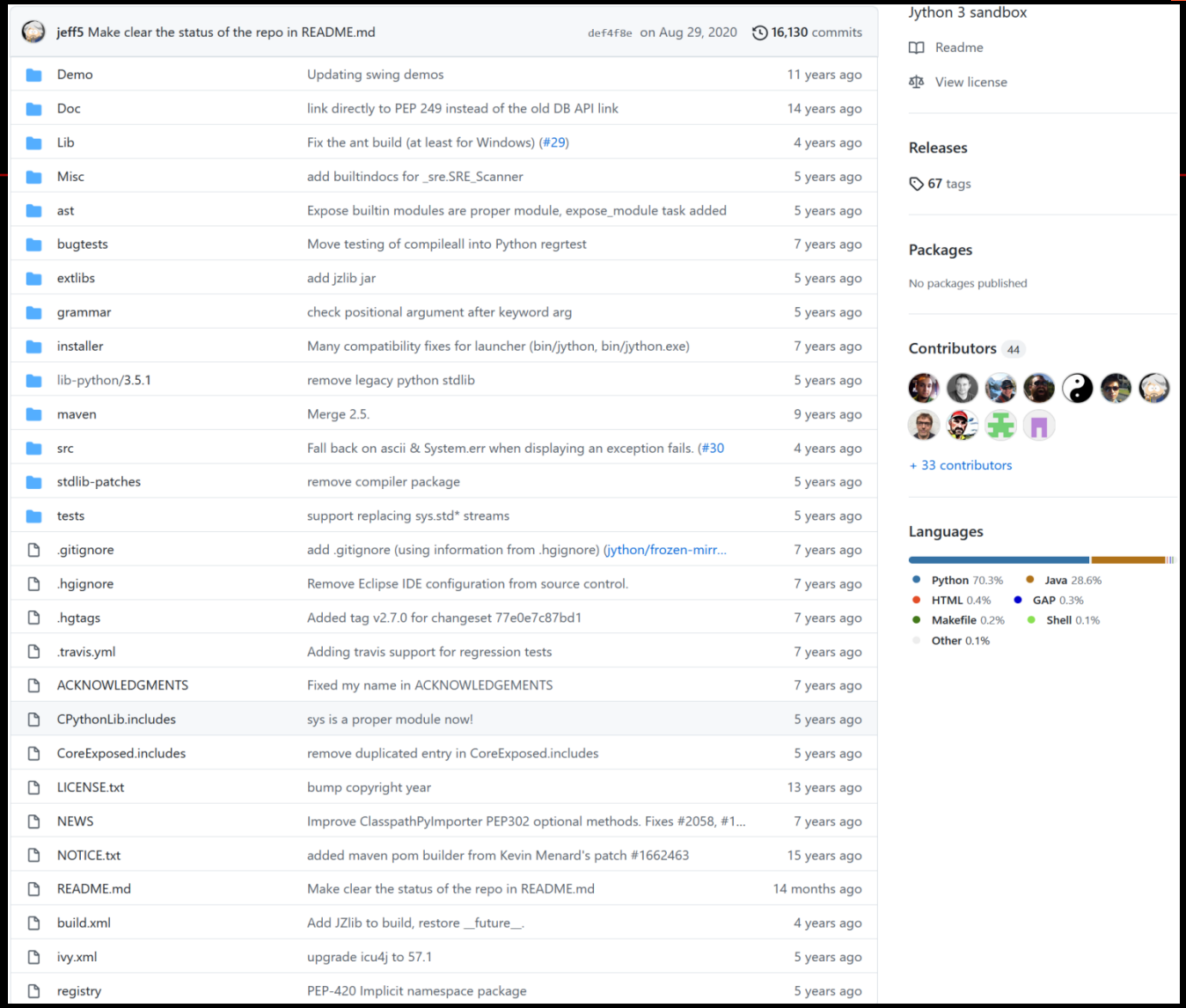
-

How it works



- **VOC**
  https://github.com/pybee/voc/r Computer Science
  A transpiler that converts Python code into Java bytecode.

Project



jeff5 Make clear the status of the repo in README.md — def4f8e on Aug 29, 2020 — 16,130 commits

Demo	Updating swing demos	11 years ago
Doc	link directly to PEP 249 instead of the old DB API link	14 years ago
Lib	Fix the ant build (at least for Windows) (#29)	4 years ago
Misc	add builtindocs for _sre.SRE_Scanner	5 years ago
ast	Expose builtin modules are proper module, expose_module task added	5 years ago
bugtests	Move testing of compileall into Python regrtest	7 years ago
extlibs	add jzlib jar	5 years ago
grammar	check positional argument after keyword arg	5 years ago
installer	Many compatibility fixes for launcher (bin/jython, bin/jython.exe)	7 years ago
lib-python/3.5.1	remove legacy python stdlib	5 years ago
maven	Merge 2.5.	9 years ago
src	Fall back on ascii & System.err when displaying an exception fails. (#30	4 years ago
stdlib-patches	remove compiler package	5 years ago
tests	support replacing sys.std* streams	5 years ago
.gitignore	add .gitignore (using information from .hgignore) (jython/frozen-mirr...	7 years ago
.hgignore	Remove Eclipse IDE configuration from source control.	7 years ago
.hgtags	Added tag v2.7.0 for changeset 77e0e7c87bd1	7 years ago
.travis.yml	Adding travis support for regression tests	7 years ago
ACKNOWLEDGMENTS	Fixed my name in ACKNOWLEDGEMENTS	7 years ago
CPythonLib.includes	sys is a proper module now!	5 years ago
CoreExposed.includes	remove duplicated entry in CoreExposed.includes	5 years ago
LICENSE.txt	bump copyright year	13 years ago
NEWS	Improve ClasspathPyImporter PEP302 optional methods. Fixes #2058, #1...	7 years ago
NOTICE.txt	added maven pom builder from Kevin Menard's patch #1662463	15 years ago
README.md	Make clear the status of the repo in README.md	14 months ago
build.xml	Add JZlib to build, restore __future__.	4 years ago
ivy.xml	upgrade icu4j to 57.1	5 years ago
registry	PEP-420 Implicit namespace package	5 years ago

Jython 3 sandbox

Readme

View license

Releases

67 tags

Packages

No packages published

Contributors 44

+ 33 contributors

Languages

Python 70.3% Java 28.6%
HTML 0.4% GAP 0.3%
Makefile 0.2% Shell 0.1%
Other 0.1%

Stats

```
[mydev@fedora jython3]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Autoconf                1           67           41            5           21
 BASH                    1          287          215           42           30
 Batch                   2            5            5            0            0
 C                       3          640          476           67           97
 C Header                1           24           24            0            0
 C Shell                 1           37           21            7            9
 CSS                     1            6            0            6            0
 Fish                    1           74           50           13           11
 Java                 1091       356230       276916        38176        41138
 Makefile                1         1746         1315          212          219
 Markdown                1           23            0           20            3
 Module-Definition       4          484          460            0           24
 Python               2220       817566       652056        59920       105590
 ReStructuredText        1          216          165            0           51
 Shell                   4          364          206           85           73
 Plain Text            136       169438            0       167640         1798
 VBScript                2           14           12            1            1
 XML                    13        33148        32584          324          240
-------------------------------------------------------------------------------
 HTML                    4         1437         1384            6           47
 |- CSS                  1            7            7            0            0
 |- JavaScript           2           17           16            1            0
 (Total)                            1461         1407            7           47
===============================================================================
 Total                3488      1381830       965953       266525       149352
===============================================================================
[mydev@fedora jython3]$
```

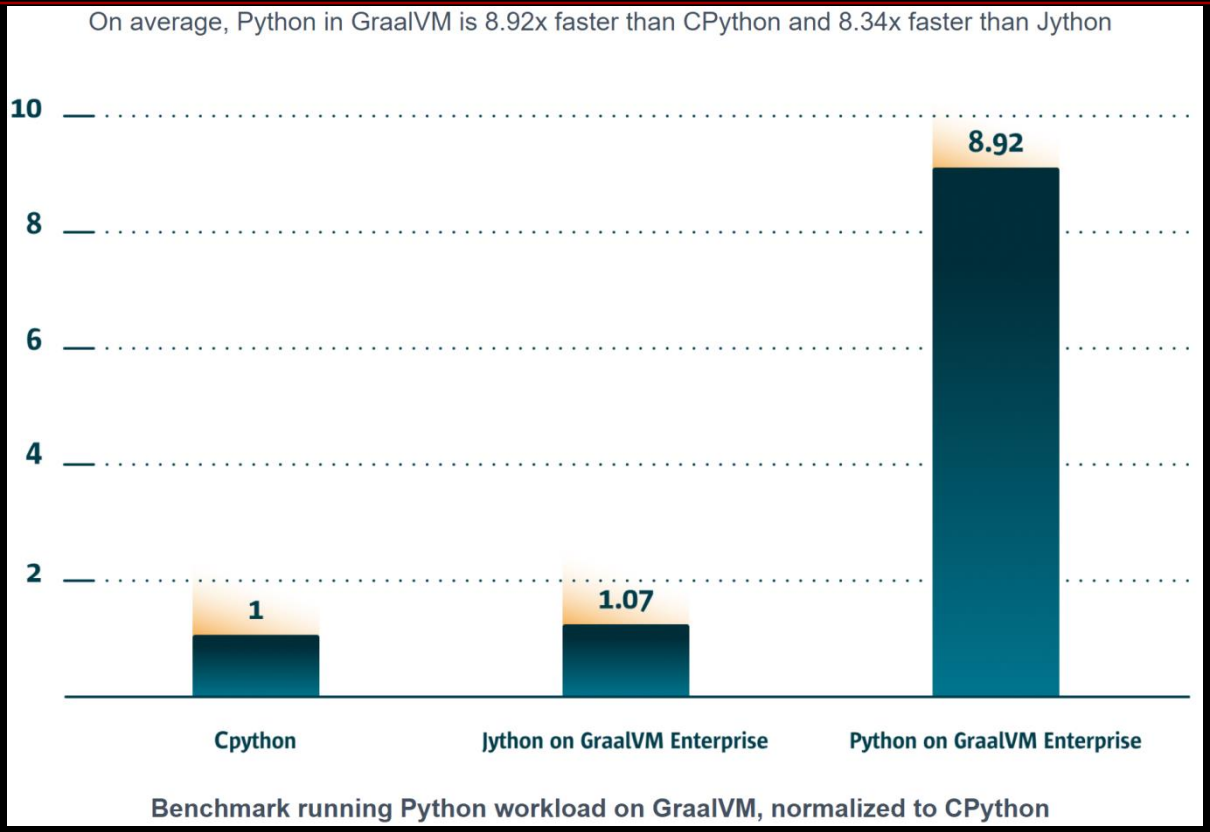
...

3) GraalPython

- https://github.com/oracle/graalpython

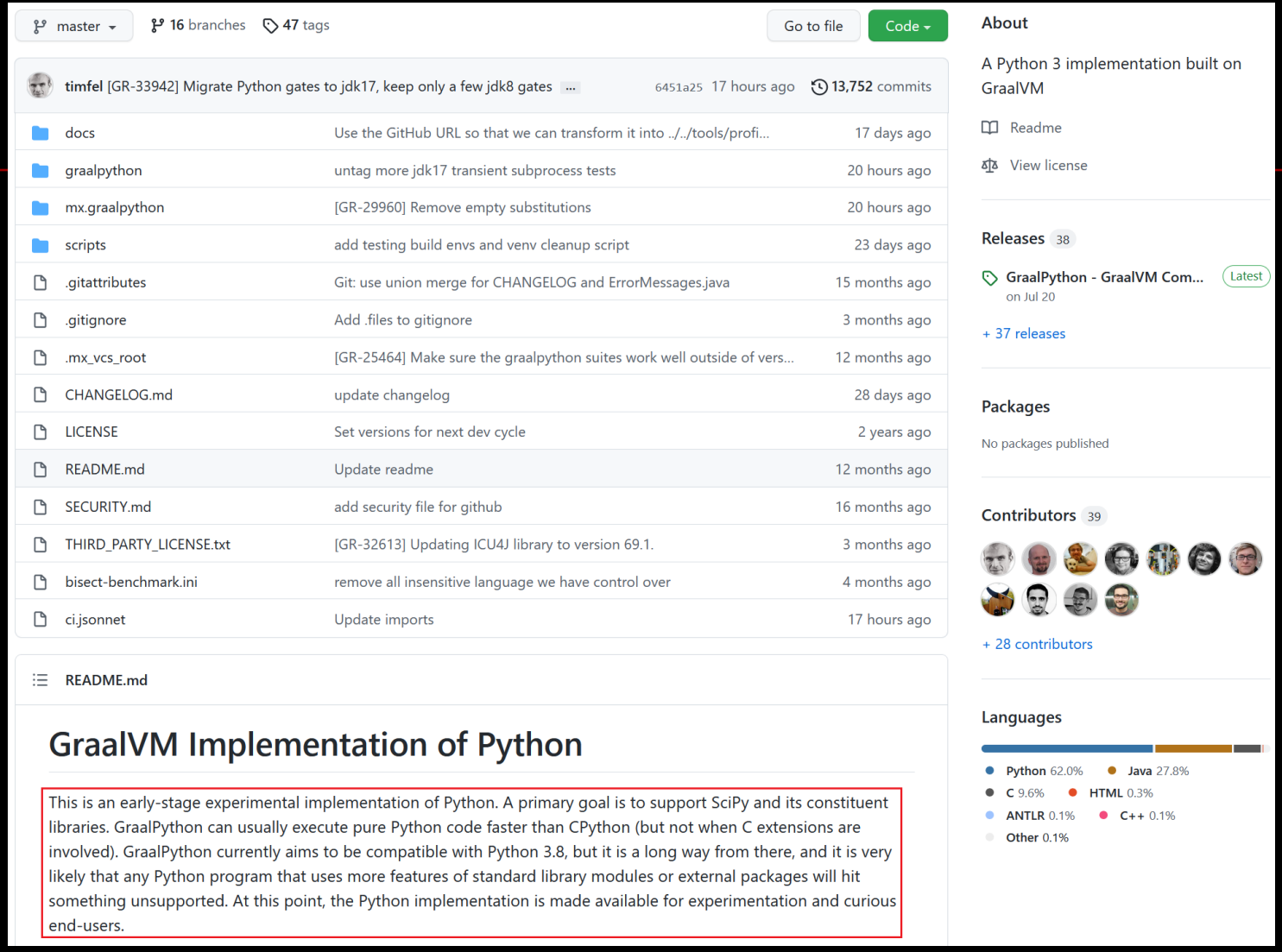
  A Python 3 implementation built on GraalVM.

- https://www.graalvm.org/python/



- Currently only support X64.

Project



master · 16 branches · 47 tags · Go to file · Code

timfel [GR-33942] Migrate Python gates to jdk17, keep only a few jdk8 gates ... 6451a25 17 hours ago 13,752 commits

docs	Use the GitHub URL so that we can transform it into ../../tools/profi...	17 days ago
graalpython	untag more jdk17 transient subprocess tests	20 hours ago
mx.graalpython	[GR-29960] Remove empty substitutions	20 hours ago
scripts	add testing build envs and venv cleanup script	23 days ago
.gitattributes	Git: use union merge for CHANGELOG and ErrorMessages.java	15 months ago
.gitignore	Add .files to gitignore	3 months ago
.mx_vcs_root	[GR-25464] Make sure the graalpython suites work well outside of vers...	12 months ago
CHANGELOG.md	update changelog	28 days ago
LICENSE	Set versions for next dev cycle	2 years ago
README.md	Update readme	12 months ago
SECURITY.md	add security file for github	16 months ago
THIRD_PARTY_LICENSE.txt	[GR-32613] Updating ICU4J library to version 69.1.	3 months ago
bisect-benchmark.ini	remove all insensitive language we have control over	4 months ago
ci.jsonnet	Update imports	17 hours ago

README.md

GraalVM Implementation of Python

This is an early-stage experimental implementation of Python. A primary goal is to support SciPy and its constituent libraries. GraalPython can usually execute pure Python code faster than CPython (but not when C extensions are involved). GraalPython currently aims to be compatible with Python 3.8, but it is a long way from there, and it is very likely that any Python program that uses more features of standard library modules or external packages will hit something unsupported. At this point, the Python implementation is made available for experimentation and curious end-users.

About

A Python 3 implementation built on GraalVM

Readme

View license

Releases 38

GraalPython - GraalVM Com... Latest
on Jul 20

+ 37 releases

Packages

No packages published

Contributors 39

+ 28 contributors

Languages

Python 62.0% Java 27.8% C 9.6% HTML 0.3% ANTLR 0.1% C++ 0.1% Other 0.1%

Stats

```
[mydev@fedora graalpython-master]$ git branch
* master
[mydev@fedora graalpython-master]$ tokei
```

Language	Files	Lines	Code	Comments	Blanks
BASH	2	56	44	4	8
Batch	4	59	44	0	15
C	88	37300	26688	6124	4488
C Header	120	66387	56500	6542	3345
C Shell	1	37	21	7	9
CSS	1	6	0	6	0
D	4	61	55	0	6
Fish	1	75	47	15	13
INI	1	42	7	28	7
Java	1130	321323	231947	56664	32712
JSON	3	86	86	0	0
Makefile	3	303	160	113	30
Module-Definition	4	569	547	0	22
PowerShell	1	241	104	105	32
Python	2177	866666	683871	72081	110714
R	1	78	20	49	9
ReStructuredText	2	217	166	0	51
Shell	17	761	121	596	44
Plain Text	469	119314	0	117160	2154
VBScript	1	1	0	1	0
XSL	1	5	5	0	0
XML	56	412	352	7	53
HTML	3	1707	1631	6	70
|- CSS	2	12	12	0	0
|- JavaScript	2	11	10	1	0
(Total)		1730	1653	7	70
Markdown	17	2023	0	1574	449
|- Java	2	27	22	0	5
|- Python	3	139	104	14	21
|- Shell	5	28	28	0	0
(Total)		2217	154	1588	475
Total	4107	1417946	1002592	261097	154257

```
[mydev@fedora graalpython-master]$
```

...

IV. LLVM-based implementation

1) Overview

- …

VMKit

- https://vmkit.llvm.org/

The VMKit project is retired

You can still play with the last VMKit release, but the project is not more maintained. Moreover, the information on these pages may be out of date.

If you are interested in restarting the project, please contact Gaël Thomas

VMKit: a substrate for virtual machines

Current MREs are monolithic. Extending them to propose new features or reusing them to execute new languages is difficult. VMKit is a library that eases the development of new MREs and the process of experimenting with new mechanisms inside MREs. VMKit provides the basic components of MREs: a JIT compiler, a GC, and a thread manager.

VMKit relies on LLVM for compilation and MMTk to manage memory. Currently, a full Java virtual machine called J3 is distributed with VMKit.

Features

For the end user, VMKit provides:

- Precise garbage collection.
- Just-in-Time and Ahead-of-Time compilation.
- Portable on many architectures (x86, x64, ppc32, ppc64, arm).

For the MRE developer, VMKit provides:

- Relatively small code base (~ 20k loc per VM)
- Infrastructure for virtual machine research and development

Current Status

VMKit currently has a decent implementation of a JVM called J3. It executes large projects (e.g. OSGi Felix, Tomcat, Eclipse) and the DaCapo benchmarks. A R virtual machine is currently under heavy development.

J3 has been tested on Linux/x64, Linux/x86, Linux/ppc32, MacOSX/x64, MacOSX/x86, MacOSX/ppc32. The JVM may work on ppc64. Support for Windows has not been investigated.

While this work aims to provide a fully functional JVM, it is still early work and is under heavy development. Some of the common missing pieces in vmkit/llvm are:

- Mixed interpretation/compilation.
- Adaptive optimization.

- ...

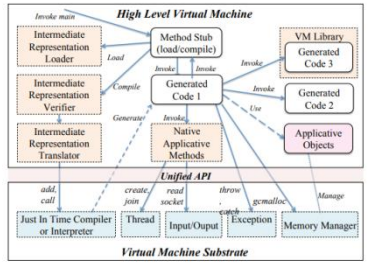
VMKit : a Substrate for Virtual Machines

Nicolas Geoffray, Gaël Thomas, Charles Clément, Bertil Folliot and Gilles Muller

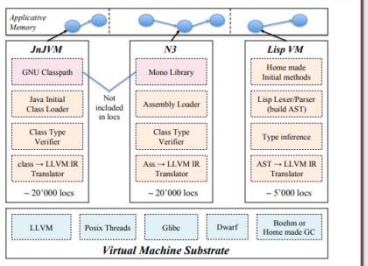
Virtual Machines optimizations require a huge amount of time. Although they share some common principles, such as a Just In Time Compiler or a Garbage Collector, this opportunity for sharing has not been exploited in current VMs.

VMKit is a first attempt to build a common substrate that eases the development of high-level VMs by reusing existing projects: LLVM, GNU Classpath, Mono Library, Boehm GC, GCC, Posix Threads.

Design of VMKit



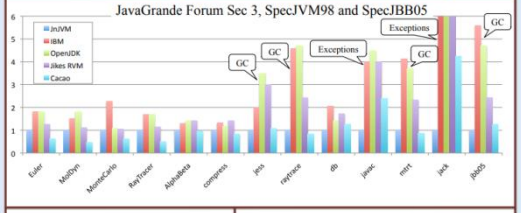
Implementation of three VMs



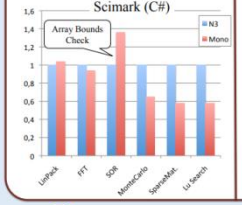
VMKit has been successfully used to build three VMs: a Java Virtual Machine (JnJVM), a Common Language Runtime (N3) and a Lisp-like runtime with type inference. Our high level VMs are only 20,000 lines of code, it took one of the author a month to develop a Common Language Runtime and implementing new ideas in the VMs was remarkably easy [IJVM - DSN09].

Performance (Linux 2.6/1.5GHz/512Mo)

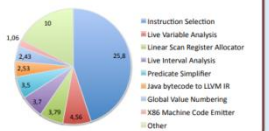
JavaGrande Forum Sec 3, SpecJVM98 and SpecJBB05



Scimark (C#)



Compilation overhead (Tomcat 6.0.16) : OpenJDK 8s, JnJVM 75s



Results

+ LLVM excellent for CPU intensive Java and .Net applications
+ Fast VM development
+ Ahead of Time Compiler

− Ongoing work for a baseline JIT
− Ongoing work to use MMTk
− Ongoing work to optimize exceptions

Homepage

http://vmkit.llvm.org

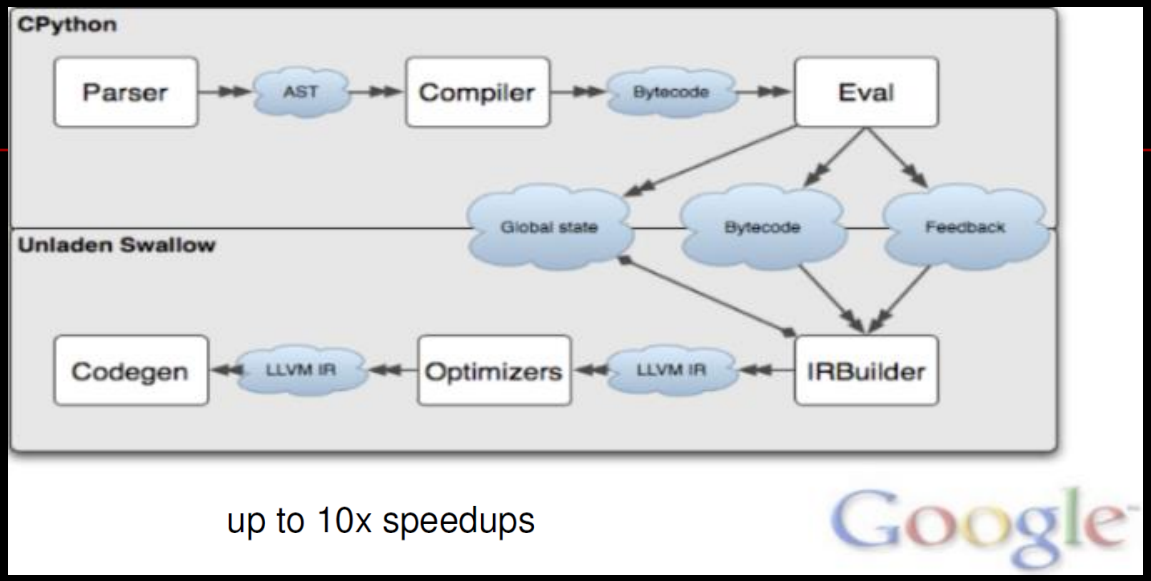
Contact

nicolas.geoffray@lip6.fr
gael.thomas@lip6.fr

LIP6 — UPMC PARIS UNIVERSITAS — INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE — INRIA centre de recherche PARIS - ROCQUENCOURT

Source: http://www-public.it-sudparis.eu/~thomas_g/research/biblio/2009/geoffray09poster-asplos-vmkit.pdf

Unladen Swallow

- https://code.google.com/p/unladen-swallow/
- 

  up to 10x speedups
- There was a branch for CPython…

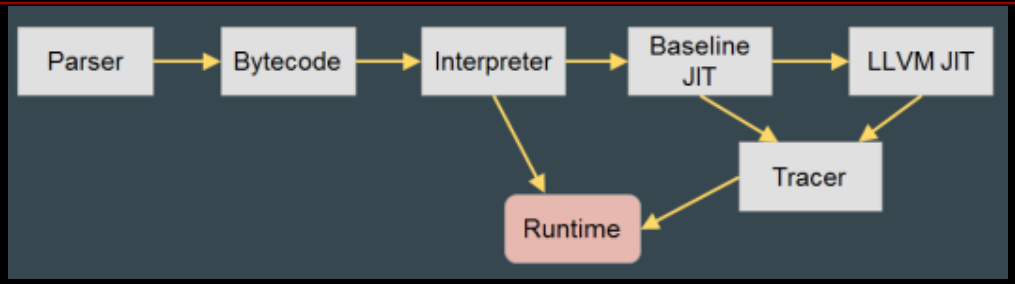
2) Pyston

Dropbox

PyCon China 2021

- https://www.pyston.org/

Pyston is a fork of CPython 3.8.8 with additional optimizations for performance. It is targeted at large real-world applications such as web serving, delivering up to a 30% speedup with no development work required.

- 

- Submodules

```
[mydev@fedora pyston-main]$ find . -name "*.git"
./.git
./pyston/LuaJIT/.git
./pyston/bolt/.git
./pyston/llvm/.git
./pyston/macrobenchmarks/.git
./pyston/test/external/django/.git
./pyston/test/external/numpy/.git
./pyston/test/external/numpy/doc/source/_static/scipy-mathjax/.git
./pyston/test/external/pandas/.git
./pyston/test/external/requests/.git
./pyston/test/external/setuptools/.git
./pyston/test/external/six/.git
./pyston/test/external/sqlalchemy/.git
./pyston/test/external/urllib3/.git
./pyston/tools/FlameGraph/.git
[mydev@fedora pyston-main]$
```

- https://blog.pyston.org/

  https://blog.pyston.org/2021/08/30/pyston-team-joins-anaconda/

- ...

Techniques

We plan on explaining our techniques in more detail in future blog posts, but the main ones we use are:

- A very-low-overhead JIT using DynASM
- Quickening
- Aggressive attribute caching
- General CPython optimizations
- Build process improvements

- **DynASM**
  https://luajit.org/dynasm.html
  https://github.com/LuaJIT/LuaJIT
  https://luajit.org/dynasm_features.html

DynASM Toolchain Features

- DynASM is a pre-processing assembler.
- DynASM converts mixed C/Assembler source to plain C code.
- The primary knowledge about instruction names, operand modes, registers, opcodes and how to encode them is *only* needed in the pre-processor.
- The generated C code is extremely small and fast.
- A tiny embeddable C library helps with the process of dynamically assembling, relocating and linking machine code.
- There are no outside dependencies on other tools (such as stand-alone assemblers or linkers).
- Internal consistency checks catch runtime errors (e.g. undefined labels).
- The toolchain is split into a portable subset and CPU-specific modules.
- DynASM itself (the pre-processor) is written in Lua.
- There is no machine-dependency for the pre-processor itself. It should work everywhere you can get Lua 5.1 and Lua BitOp up and running (i.e. Linux, *BSD, Windows, ... you name it).

DynASM Assembler Features

- C code and assembler code can be freely mixed. *Readable*, too.
- All the usual syntax for instructions and operand modes you come to expect from a standard assembler.
- Access to C variables and CPP defines in assembler statements.
- Access to C structures and unions via type mapping.
- Convenient shortcuts for accessing C structures.
- Local and global labels.
- Numbered labels (e.g. for mapping bytecode instruction numbers).
- Multiple code sections (e.g. for tailcode).
- Defines/substitutions (inline and from command line).
- Conditionals (translation time) with proper nesting.
- Macros with parameters.
- Macros can mix assembler statements and C code.
- Captures (output diversion for code reordering).
- Simple and extensible template system for instruction definitions.

Restrictions

Currently the x86, x64, ARM, ARM64, PowerPC and MIPS instruction sets are supported. This includes most user-mode instructions available on modern CPUs. For x86/x64 this includes SSE, SSE2, SSE3, SSSE3, SSE4a, SSE4.1, SSE4.2, AVX, AVX2, BMI, ADX, AES-NI and FMA3. For PPC this also includes the e500 instruction set extension.

The whole toolchain has been designed to support multiple CPU architectures. As LuaJIT gets support for more architectures, DynASM will be extended with new CPU-specific modules.

Note that runtime conditionals are not really needed, since you can just use plain C code for that (and LuaJIT does this *a lot*). It's not going to be more (time-) efficient if conditionals are done by the embedded C library (maybe a bit more space-efficient).

DynASM Dependencies

Please don't be shied away because DynASM itself is written in Lua. This *only* applies to the pre-processor. This is pure text-processing and writing such stuff in C would be a waste of time. Pre-processing is done only *once* while your code generator itself is compiled. There are no dependencies on Lua during runtime, i.e. when your code generator is in action.

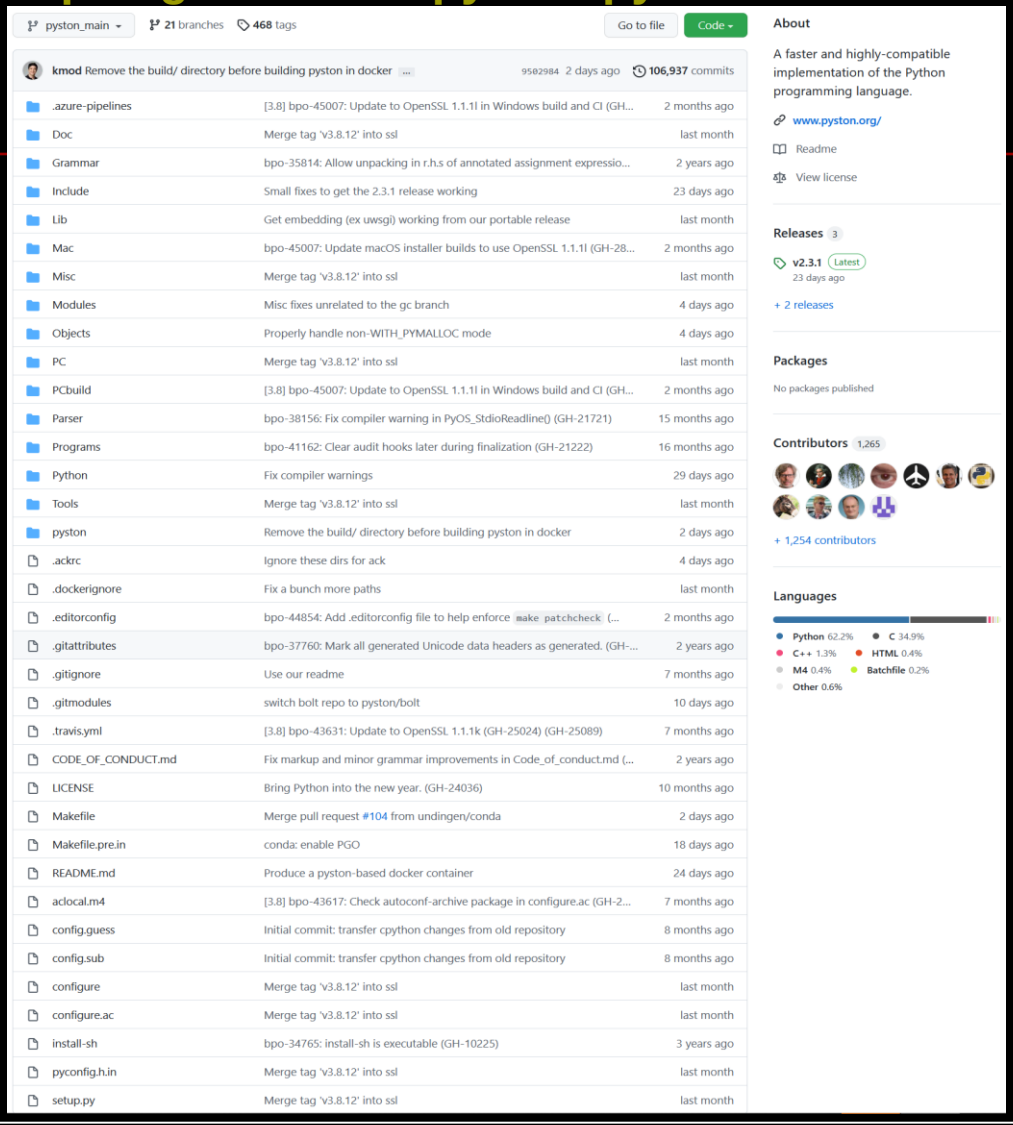
Apart from that, a full Lua distribution is around 200K and can be compiled in a few seconds. Consider it a part of the toolchain, if you want.

Or bundle `src/host/minilua.c` from the LuaJIT source tree, which is a minified Lua 5.1 + BitOp in a single file (45K compressed).

...

Project

- https://github.com/pyston/pyston



pyston_main · 21 branches · 468 tags · Go to file · Code

kmod Remove the build/ directory before building pyston in docker ... 9502984 2 days ago · 106,937 commits

.azure-pipelines	[3.8] bpo-45007: Update to OpenSSL 1.1.1l in Windows build and CI (GH...	2 months ago
Doc	Merge tag 'v3.8.12' into ssl	last month
Grammar	bpo-35814: Allow unpacking in r.h.s of annotated assignment expressio...	2 years ago
Include	Small fixes to get the 2.3.1 release working	23 days ago
Lib	Get embedding (ex uwsgi) working from our portable release	last month
Mac	bpo-45007: Update macOS installer builds to use OpenSSL 1.1.1l (GH-28...	2 months ago
Misc	Merge tag 'v3.8.12' into ssl	last month
Modules	Misc fixes unrelated to the gc branch	4 days ago
Objects	Properly handle non-WITH_PYMALLOC mode	4 days ago
PC	Merge tag 'v3.8.12' into ssl	last month
PCbuild	[3.8] bpo-45007: Update to OpenSSL 1.1.1l in Windows build and CI (GH...	2 months ago
Parser	bpo-38156: Fix compiler warning in PyOS_StdioReadline() (GH-21721)	15 months ago
Programs	bpo-41162: Clear audit hooks later during finalization (GH-21222)	16 months ago
Python	Fix compiler warnings	29 days ago
Tools	Merge tag 'v3.8.12' into ssl	last month
pyston	Remove the build/ directory before building pyston in docker	2 days ago
.ackrc	Ignore these dirs for ack	4 days ago
.dockerignore	Fix a bunch more paths	last month
.editorconfig	bpo-44854: Add .editorconfig file to help enforce `make patchcheck` (...	2 months ago
.gitattributes	bpo-37760: Mark all generated Unicode data headers as generated. (GH-...	2 years ago
.gitignore	Use our readme	7 months ago
.gitmodules	switch bolt repo to pyston/bolt	10 days ago
.travis.yml	[3.8] bpo-43631: Update to OpenSSL 1.1.1k (GH-25024) (GH-25089)	7 months ago
CODE_OF_CONDUCT.md	Fix markup and minor grammar improvements in Code_of_conduct.md (...	2 years ago
LICENSE	Bring Python into the new year. (GH-24036)	10 months ago
Makefile	Merge pull request #104 from undingen/conda	2 days ago
Makefile.pre.in	conda: enable PGO	18 days ago
README.md	Produce a pyston-based docker container	24 days ago
aclocal.m4	[3.8] bpo-43617: Check autoconf-archive package in configure.ac (GH-2...	7 months ago
config.guess	Initial commit: transfer cpython changes from old repository	8 months ago
config.sub	Initial commit: transfer cpython changes from old repository	8 months ago
configure	Merge tag 'v3.8.12' into ssl	last month
configure.ac	Merge tag 'v3.8.12' into ssl	last month
install-sh	bpo-34765: install-sh is executable (GH-10225)	3 years ago
pyconfig.h.in	Merge tag 'v3.8.12' into ssl	last month
setup.py	Merge tag 'v3.8.12' into ssl	last month

About

A faster and highly-compatible implementation of the Python programming language.

www.pyston.org/

Readme

View license

Releases 3

v2.3.1 Latest
23 days ago

+ 2 releases

Packages

No packages published

Contributors 1,265

+ 1,254 contributors

Languages

Python 62.2% · C 34.9% · C++ 1.3% · HTML 0.4% · M4 0.4% · Batchfile 0.2% · Other 0.6%

Stats (include submodules)

```
[mydev@fedora pyston-main]$ git branch
* pyston_main
[mydev@fedora pyston-main]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Alex                    5           68           51            0           17
 Assembly               40         4817         2368         1618          831
 GNU Style Assembly  14593      3225607      1559415       980276       685916
 Autoconf              741        74180        36765        24934        12481
 Automake               15         6299         5514          121          664
 BASH                   23         3393         2567          485          341
 Batch                  95         6395         5375          132          888
 C                   14345      3007898      1705697       988717       313484
 C Header            17307      3402444      2171978       745541       484925
 CMake                2794       138275       111172        11413        15690
 C#                     16         1532         1140          214          178
 C Shell                 1           37           21            7            9
 C++                 44380     11562046      8222335      1928671      1411040
 C++ Header             64        23869        19970         1740         2159
 CSS                   109        14444        11752          617         2075
 D                       9          183          160            1           22
 Dockerfile             15          755          376          275          104
 .NET Resource           4          538          264          274            0
 Emacs Lisp             19         2638         1841          502          295
 Elixir                  4          149           46           68           35
 Fish                    1           75           47           15           13
 FORTRAN Legacy         68         1045          745          285           15
 FORTRAN Modern        690        43716        31770         7337         4609
 GDB Script              3          289           59          173           57
 Go                   1934       486594       368911        71788        45895
 HEX                     2           30           30            0            0
 INI                    27          951          636          147          168
 JavaScript            325        81958        56810        16344         8804
 JSON                  235       423225       423165            0           60
 Julia                   1         1335          766          451          118
 LLVM                51511      9723272      3796754      5100621       825897
 LD Script               2            8            8            0            0
 Lisp                    1          692          502           81          109
 Lua                    31        15865        13022         1588         1255
 Makefile             1278        10120         7047          609         2464
 Module-Definition     268        56148        50454          172         5522
 MSBuild                26         2232         1828          275          129
 OCaml                 136        20782        11432         5880         3470
 Objective-C          3529       230316       129230        62709        38377
 Objective-C++        1014        77458        50784        13133        13541
 Pan                    10          352          333            1           18
 Perl                   65        21290        13978         5278         2034
 PHP                     1          197          159           25           13
 PowerShell              7          612          354          194           64
 Prolog                  1           24           24            0            0
 Protocol Buffers        8          693          481          103          109
 Python              11164      3046911      2422883       192875       431153
 R                       1           15            7            5            3
 ReStructuredText     3941      1072201       791573            0       280628
 Rust                    6           60           26           22           12
 Scala                  20          202          202            0            0
 Shell                 260        90640        67301        14068         9271
 SVG                   154        97142        97005           94           43
 Swift                   4           46           34            0           12
 SWIG                  151        25803        19124         1597         5082
 TeX                     9         4980         4523          101          356
 Plain Text           3235      1364420            0      1028004       336416
 Thrift                  1           23           17            1            5
 TOML                    8          304          253           17           34
 TypeScript              8          915          657          193           65
 VBScript                1            1            0            1            0
 Vim script             28         1325         1170           92           63
 Visual Studio Pro|     48         6265         6211            9           45
 Visual Studio Sol|      6         1566         1561            0            5
 XSL                     3           42           37            0            5
 XML                   174         2688         2551           24          113
 YAML                 1656       175044       133422        34344         7278
-------------------------------------------------------------------------------
 HTML                  652       132314       105576         1052         5686
 |- CSS                 60         1384         1081           14          289
 |- JavaScript          42        15008        13357          550         1101
 (Total)                      148706       120014         1616         7076
-------------------------------------------------------------------------------
 Jupyter Notebooks       1            0            0            0            0
 |- Markdown             1          400            1          281          118
 |- Python               1          376          344           22           10
 (Total)                             776          345          303          128
-------------------------------------------------------------------------------
 Markdown              269        63398            0        49953        13445
 |- BASH                12          288          258           28            2
 |- C                    5           40           36            4            0
 |- CMake                3           35           31            0            4
 |- C++                 59         5575         3726         1326          523
 |- Go                   1           52           43            0            9
 |- JavaScript           2           70           70            0            0
 |- JSON                 4          140          140            0            0
 |- LLVM                 3           88           81            0            7
 |- Python               5          128          111            4           13
 |- Shell               16          366          295           30           41
 (Total)                            70180         4791        51345        14044
===============================================================================
 Total               177553     38765101     22491843     11297526      4975732
===============================================================================
[mydev@fedora pyston-main]$
[mydev@fedora pyston-main]$ du -sh
6.5G    .
[mydev@fedora pyston-main]$
```

V. DotNet-based implementation

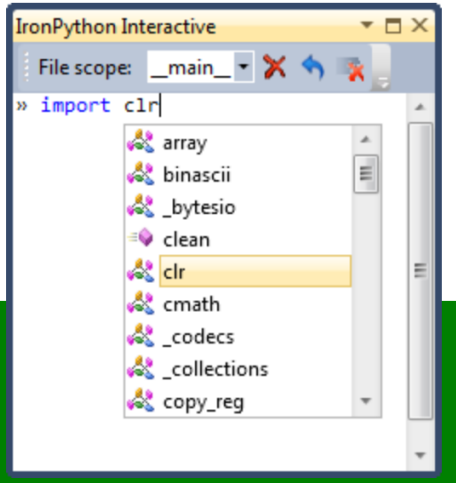
1) Overview

- …

2) IronPython

- https://ironpython.net/
-

IronPython is an open-source implementation of the Python programming language which is tightly integrated with .NET. IronPython can use .NET and Python libraries, and other .NET languages can use Python code just as easily.



Download IronPython 2.7

2.7.11 released on 2020-11-17
release notes | source

Download IronPython 3.4

3.4.0-alpha1 released on 2021-04-19
release notes | source

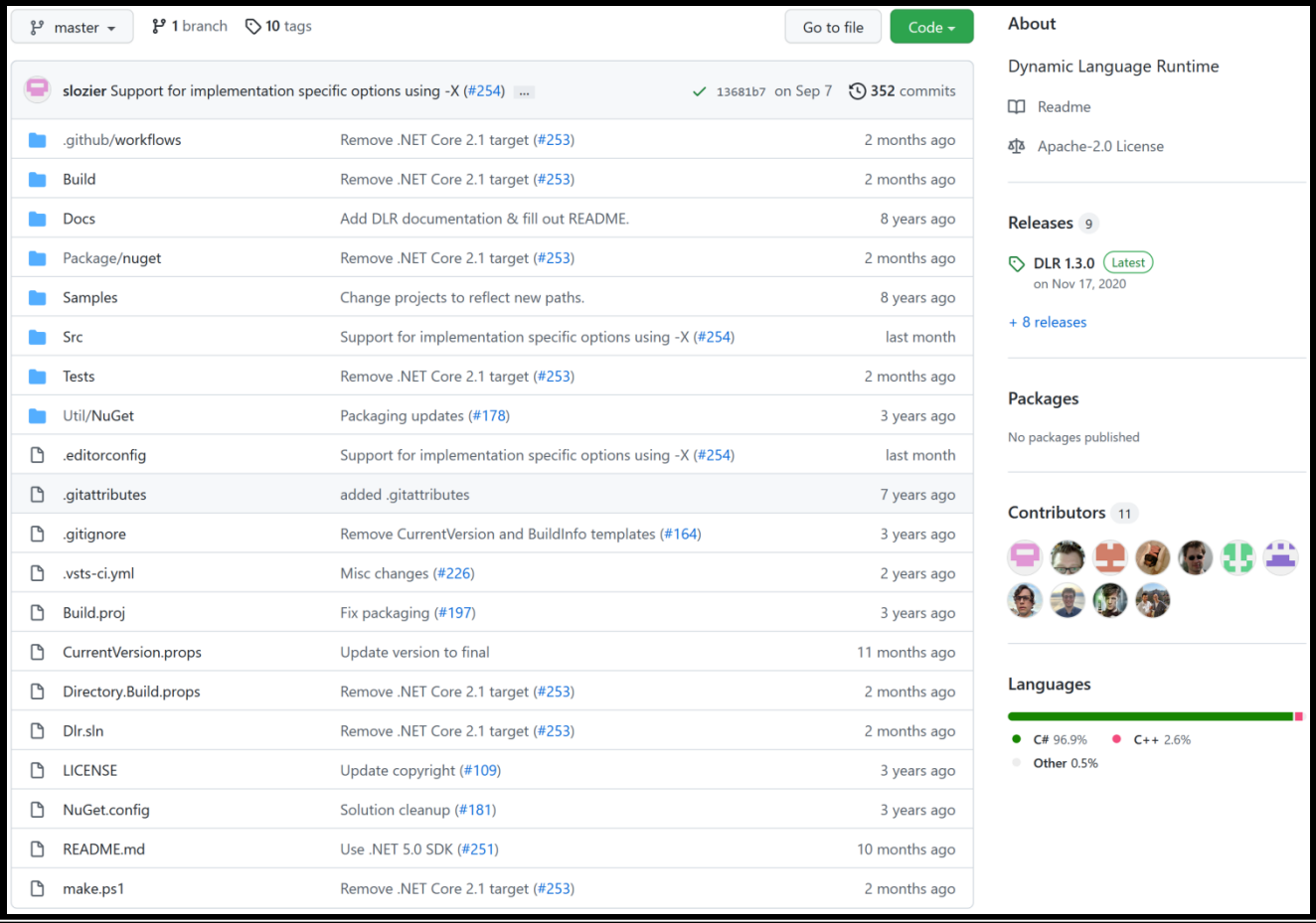
Experience a more interactive .NET and Python development experience with Python Tools for Visual Studio.

- ...

DLR (Dynamic Language Runtime)

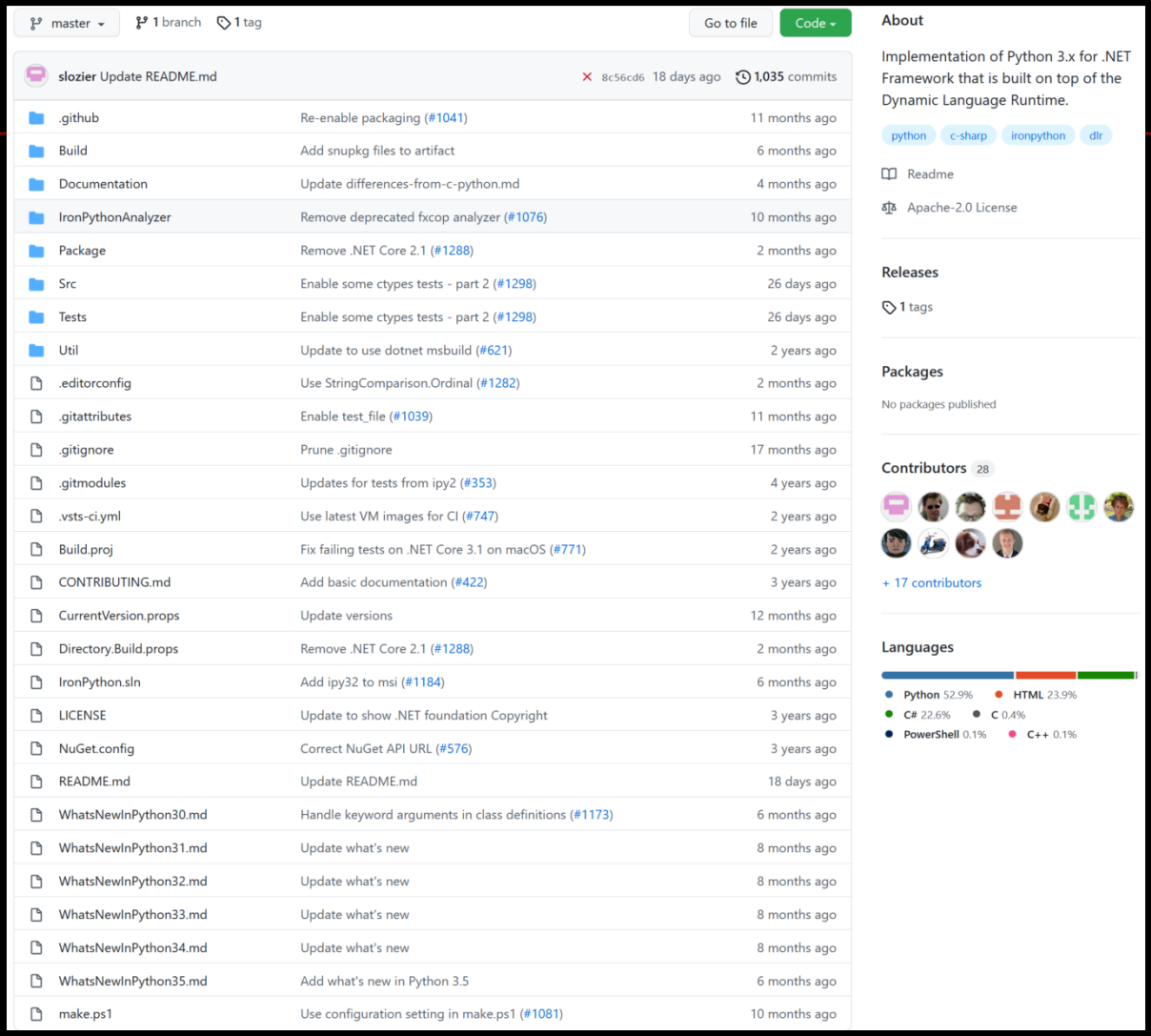
- https://github.com/IronLanguages/dlr

The Dynamic Language Runtime enables language developers to more easily create dynamic languages for the .NET platform. In addition to being a pluggable back-end for dynamic language compilers, the DLR provides language interop for dynamic operations on objects. The DLR has common hosting APIs for using dynamic languages as libraries or for scripting in your .NET applications.



Project

- https://github.com/IronLanguages/ironpython3



master · 1 branch · 1 tag · Go to file · Code

slozier Update README.md — 8c56cd6 18 days ago — 1,035 commits

.github	Re-enable packaging (#1041)	11 months ago
Build	Add snupkg files to artifact	6 months ago
Documentation	Update differences-from-c-python.md	4 months ago
IronPythonAnalyzer	Remove deprecated fxcop analyzer (#1076)	10 months ago
Package	Remove .NET Core 2.1 (#1288)	2 months ago
Src	Enable some ctypes tests - part 2 (#1298)	26 days ago
Tests	Enable some ctypes tests - part 2 (#1298)	26 days ago
Util	Update to use dotnet msbuild (#621)	2 years ago
.editorconfig	Use StringComparison.Ordinal (#1282)	2 months ago
.gitattributes	Enable test_file (#1039)	11 months ago
.gitignore	Prune .gitignore	17 months ago
.gitmodules	Updates for tests from ipy2 (#353)	4 years ago
.vsts-ci.yml	Use latest VM images for CI (#747)	2 years ago
Build.proj	Fix failing tests on .NET Core 3.1 on macOS (#771)	2 years ago
CONTRIBUTING.md	Add basic documentation (#422)	3 years ago
CurrentVersion.props	Update versions	12 months ago
Directory.Build.props	Remove .NET Core 2.1 (#1288)	2 months ago
IronPython.sln	Add ipy32 to msi (#1184)	6 months ago
LICENSE	Update to show .NET foundation Copyright	3 years ago
NuGet.config	Correct NuGet API URL (#576)	3 years ago
README.md	Update README.md	18 days ago
WhatsNewInPython30.md	Handle keyword arguments in class definitions (#1173)	6 months ago
WhatsNewInPython31.md	Update what's new	8 months ago
WhatsNewInPython32.md	Update what's new	8 months ago
WhatsNewInPython33.md	Update what's new	8 months ago
WhatsNewInPython34.md	Update what's new	8 months ago
WhatsNewInPython35.md	Add what's new in Python 3.5	6 months ago
make.ps1	Use configuration setting in make.ps1 (#1081)	10 months ago

About

Implementation of Python 3.x for .NET Framework that is built on top of the Dynamic Language Runtime.

python · c-sharp · ironpython · dlr

Readme

Apache-2.0 License

Releases

1 tags

Packages

No packages published

Contributors 28

+ 17 contributors

Languages

Python 52.9% · HTML 23.9% · C# 22.6% · C 0.4% · PowerShell 0.1% · C++ 0.1%

Stats (include submodules)

```
[mydev@fedora ironpython3]$ git branch
* master
[mydev@fedora ironpython3]$
[mydev@fedora ironpython3]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Batch                  12          303          214           32           57
 C                       1          662          535           27          100
 C Header               87        10243         7649         1051         1543
 C#                   1277       421186       292508        74640        54038
 C Shell                 1           37           21            7            9
 C++                    20         3279         2412           89          778
 CSS                     1            6            0            6            0
 .NET Resource           1          244          185           59            0
 Fish                    1           74           50           13           11
 INI                     2         1288          920            2          366
 JSON                    1            5            5            0            0
 Makefile                1           19           15            0            4
 Module-Definition       6          503          474            1           28
 MSBuild                51         8498         5208         2321          969
 PowerShell              8         2407         1875          269          263
 Python               1916       732933       587515        51236        94182
 ReStructuredText        1          216          165            0           51
 Ruby                    1            7            6            0            1
 Shell                   5           47           34            5            8
 SVG                     1           93           86            1            6
 Plain Text            119        54083            0        52449         1634
 VBScript                2           14           12            1            1
 Visual Basic            3          294          223           28           43
 Visual Studio Pro|      3          850          850            0            0
 Visual Studio Sol|      4          381          377            0            4
 XML                    11        15057        15017            2           38
 YAML                    2          278          221           15           42
-------------------------------------------------------------------------------
 HTML                   42       255539       227974            6        27559
 |- CSS                 1            7            7            0            0
 |- JavaScript          2           17           16            1            0
 (Total)                           255563       227997            7        27559
-------------------------------------------------------------------------------
 Markdown               17         1616            0         1307          309
 |- Python              1            2            2            0            0
 (Total)                            1618            2         1307          309
===============================================================================
 Total                3597      1510188      1144576       183568       182044
===============================================================================
[mydev@fedora ironpython3]$
```

3) Pyjion

- https://pyjion.readthedocs.io/en/latest/

  A JIT extension(not a standalone Python implementation) for CPython that compiles your Python code into native CIL and executes it using the .NET CLR.

- Build from source

  Prerequisites:

  - CPython 3.10
  - CMake 3.2 +
  - .NET 6 RC2
  - scikit-build

  …

- https://www.trypyjion.com/
- Originally from https://github.com/microsoft/Pyjion
- https://thautwarm.github.io/Site-32/Design/Research-Restrain-JIT.html
- Support both X64 and AArch64!
- Maybe a good fit for Serverless functions in Python…

Usage

To get started, you need to have .NET installed, with Python 3.10 and the Pyjion package (I also recommend using a virtual environment).

After importing pyjion, enable it by calling `pyjion.enable()` which sets a compilation threshold to 0 (the code only needs to be run once to be compiled by the JIT):

```
>>> import pyjion
>>> pyjion.enable()
```

Any Python code you define or import after enabling pyjion will be JIT compiled. You don't need to execute functions in any special API, its completely transparent:

```
>>> def half(x):
...     return x/2
>>> half(2)
1.0
```

Pyjion will have compiled the `half` function into machine code on-the-fly and stored a cached version of that compiled function inside the function object. You can see some basic stats by running `pyjion.info(f)`, where `f` is the function object:

```
>>> pyjion.info(half)
JitInfo(failed=False, compile_result=<CompilationResult.Success: 1>, compiled=True, optimizations=<Optimizatio
```

You can also execute Pyjion against any script or module:

```
pyjion my_script.py
```

Or, for an existing Python module:

```
pyjion -m calendar
```

You can see the machine code for the compiled function by disassembling it in the Python REPL. Pyjion has essentially compiled your small Python function into a small, standalone application. Install `distorm3` and `rich` first to disassemble x86-64 assembly and run `pyjion.dis.dis_native(f)`:

```
>>> import pyjion.dis
>>> pyjion.dis.dis_native(half)
00000000: PUSH RBP
00000001: MOV RBP, RSP
00000004: PUSH R14
00000006: PUSH RBX
00000007: MOV RBX, RSI
0000000a: MOV R14, [RDI+0x40]
0000000e: CALL 0x1b34
00000013: CMP DWORD [RAX+0x30], 0x0
00000017: JZ 0x31
00000019: CMP QWORD [RAX+0x40], 0x0
0000001e: JZ 0x31
00000020: MOV RDI, RAX
00000023: MOV RSI, RBX
00000026: XOR EDX, EDX
00000028: POP RBX
00000029: POP R14
...
```

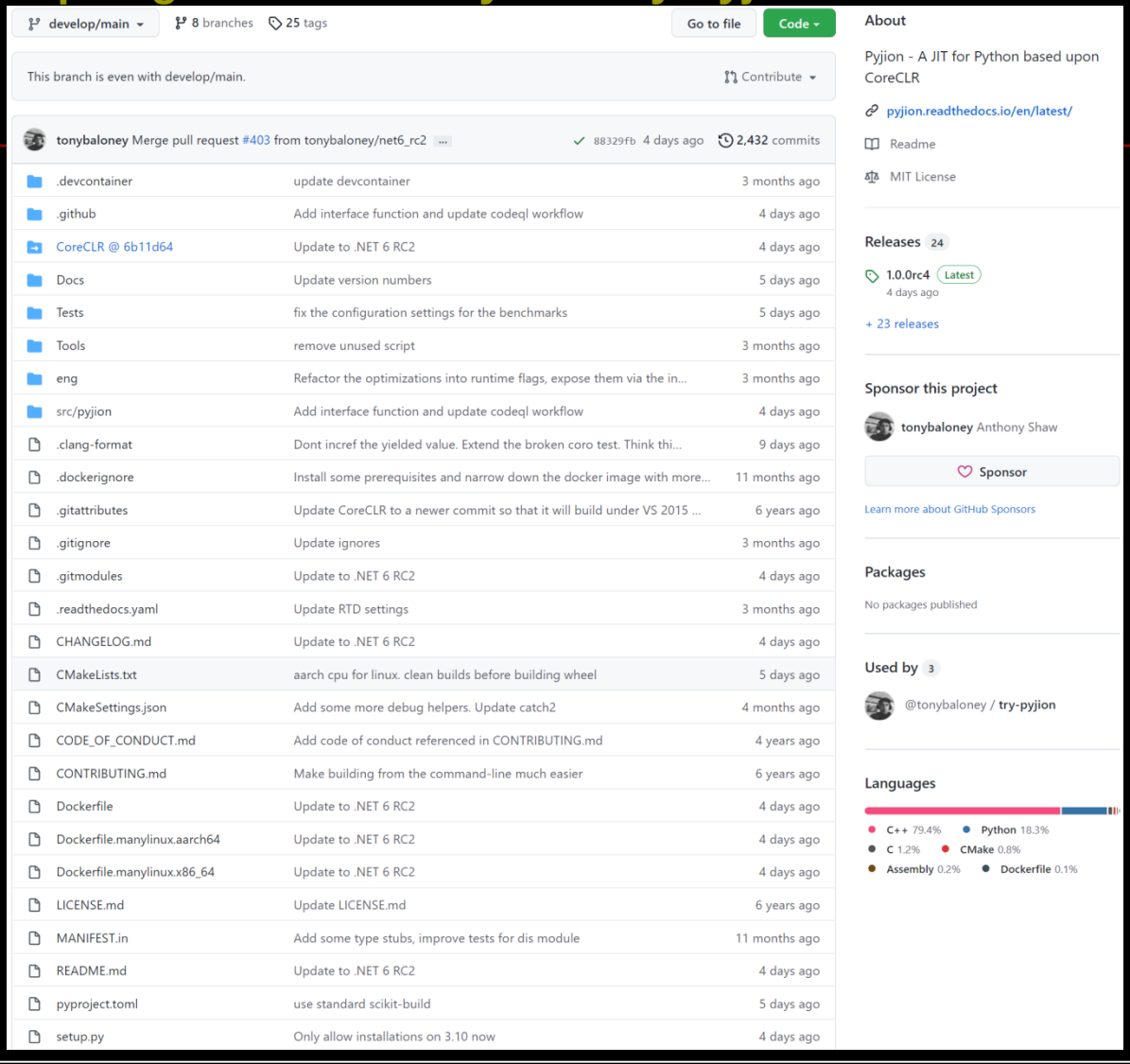
The complex logic of converting a portable instruction set into low-level machine instructions is done by .NET's CLR JIT compiler.

All Python code executed after the JIT is enabled will be compiled into native machine code at runtime and cached on disk. For example, to enable the JIT on a simple `app.py` for a Flask web app:

```
from src import pyjion
pyjion.enable()

from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

app.run()
```

Project

- https://github.com/tonybaloney/Pyjion



develop/main · 8 branches · 25 tags · Go to file · Code

This branch is even with develop/main. · Contribute

tonybaloney Merge pull request #403 from tonybaloney/net6_rc2 ... 88329fb 4 days ago 2,432 commits

.devcontainer	update devcontainer	3 months ago
.github	Add interface function and update codeql workflow	4 days ago
CoreCLR @ 6b11d64	Update to .NET 6 RC2	4 days ago
Docs	Update version numbers	5 days ago
Tests	fix the configuration settings for the benchmarks	5 days ago
Tools	remove unused script	3 months ago
eng	Refactor the optimizations into runtime flags, expose them via the in...	3 months ago
src/pyjion	Add interface function and update codeql workflow	4 days ago
.clang-format	Dont incref the yielded value. Extend the broken coro test. Think thi...	9 days ago
.dockerignore	Install some prerequisites and narrow down the docker image with more...	11 months ago
.gitattributes	Update CoreCLR to a newer commit so that it will build under VS 2015 ...	6 years ago
.gitignore	Update ignores	3 months ago
.gitmodules	Update to .NET 6 RC2	4 days ago
.readthedocs.yaml	Update RTD settings	3 months ago
CHANGELOG.md	Update to .NET 6 RC2	4 days ago
CMakeLists.txt	aarch cpu for linux. clean builds before building wheel	5 days ago
CMakeSettings.json	Add some more debug helpers. Update catch2	4 months ago
CODE_OF_CONDUCT.md	Add code of conduct referenced in CONTRIBUTING.md	4 years ago
CONTRIBUTING.md	Make building from the command-line much easier	6 years ago
Dockerfile	Update to .NET 6 RC2	4 days ago
Dockerfile.manylinux.aarch64	Update to .NET 6 RC2	4 days ago
Dockerfile.manylinux.x86_64	Update to .NET 6 RC2	4 days ago
LICENSE.md	Update LICENSE.md	6 years ago
MANIFEST.in	Add some type stubs, improve tests for dis module	11 months ago
README.md	Update to .NET 6 RC2	4 days ago
pyproject.toml	use standard scikit-build	5 days ago
setup.py	Only allow installations on 3.10 now	4 days ago

About

Pyjion - A JIT for Python based upon CoreCLR

pyjion.readthedocs.io/en/latest/

Readme

MIT License

Releases 24

1.0.0rc4 Latest
4 days ago

+ 23 releases

Sponsor this project

tonybaloney Anthony Shaw

Sponsor

Learn more about GitHub Sponsors

Packages

No packages published

Used by 3

@tonybaloney / try-pyjion

Languages

C++ 79.4% Python 18.3%
C 1.2% CMake 0.8%
Assembly 0.2% Dockerfile 0.1%

Stats (include submodules)

```
[mydev@fedora pyjion-develop-main]$ git branch
* develop/main
[mydev@fedora pyjion-develop-main]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 Assembly               46        12992         6364         3728         2900
 GNU Style Assembly    109        13659         7632         3589         2438
 Autoconf               39         4010         1940         1396          674
 Automake                4         1237         1000           87          150
 BASH                    2          111           75           15           21
 Batch                  51         4251         3106          443          702
 C                    1217       591014       438867        74262        77885
 C Header             2124       679796       453195       111164       115437
 CMake                 296        23206        18349         1793         3064
 C#                  28370      8257055      6474247       666487      1116321
 C++                  2326      1480262      1028757       235111       216394
 C++ Header            230        97988        69915        12840        15233
 CSS                     2          132          108            1           23
 Dockerfile             12          313          222           25           66
 .NET Resource         213        70153        63468         6680            5
 F#                     10         1521         1196           94          231
 Happy                   1         2063         1847            0          216
 INI                     1            5            5            0            0
 Java                    2          204          168            7           29
 JavaScript             14         4788         3572          524          692
 JSON                   64        12821        12811            0           10
 Makefile               16          812          563           80          169
 Module-Definition      31         3682         3343           16          323
 MSBuild              8243       163250       156891         3396         2963
 Objective-C            11          972          746           63          163
 Objective-C++           5          219          156           26           37
 Perl                   10         1888         1475          204          209
 PowerShell             58         6728         5270          600          858
 Prolog                  6         2547         2296            0          251
 Python                113        31563        23326         2703         5534
 ReStructuredText       23         1837         1255            0          582
 Shell                 182        26509        20385         1679         4445
 SVG                     3          800          789            3            8
 TeX                    32         3253         2511            0          742
 Plain Text            445       177936            0       173817         4119
 TOML                    1           14           13            0            1
 Visual Basic           74        50623        36496         5027         9100
 Visual Studio Pro|     11         2398         2391            6            1
 Visual Studio Sol|    252        30703        30680            0           23
 XSL                   640        14345        11693          387         2265
 XML                   852        67357        63271         2319         1767
 YAML                  140        16170        13341         1356         1473
-------------------------------------------------------------------------------
 HTML                   29          628          584           12           32
 |- JavaScript           6          237          207            4           26
 (Total)                             865          791           16           58
-------------------------------------------------------------------------------
 Markdown              377        61848            0        45333        16515
 |- Assembly             1          109           81           28            0
 |- BASH                13          195          148           29           18
 |- C                    3           40           36            0            4
 |- CMake                2           56           46            0           10
 |- C#                  20         1076          851           90          135
 |- C++                 17         1513         1070          262          181
 |- Java                 1            7            7            0            0
 |- JavaScript           1           33           33            0            0
 |- JSON                 5          150          150            0            0
 |- Markdown             1            9            0            5            4
 |- PowerShell           4           17           17            0            0
 |- Python               2           21           18            0            3
 |- Shell                2           16            8            5            3
 |- XML                 21          232          211           12            9
 (Total)                            65322         2676        45764        16882
===============================================================================
 Total                46687     11927374      8967202      1355708      1604464
===============================================================================
[mydev@fedora pyjion-develop-main]$
[mydev@fedora pyjion-develop-main]$ du -sh
1.6G    .
[mydev@fedora pyjion-develop-main]$
```

VI. Wasm-based implementation

1) Overview

- …

2) Pyodide

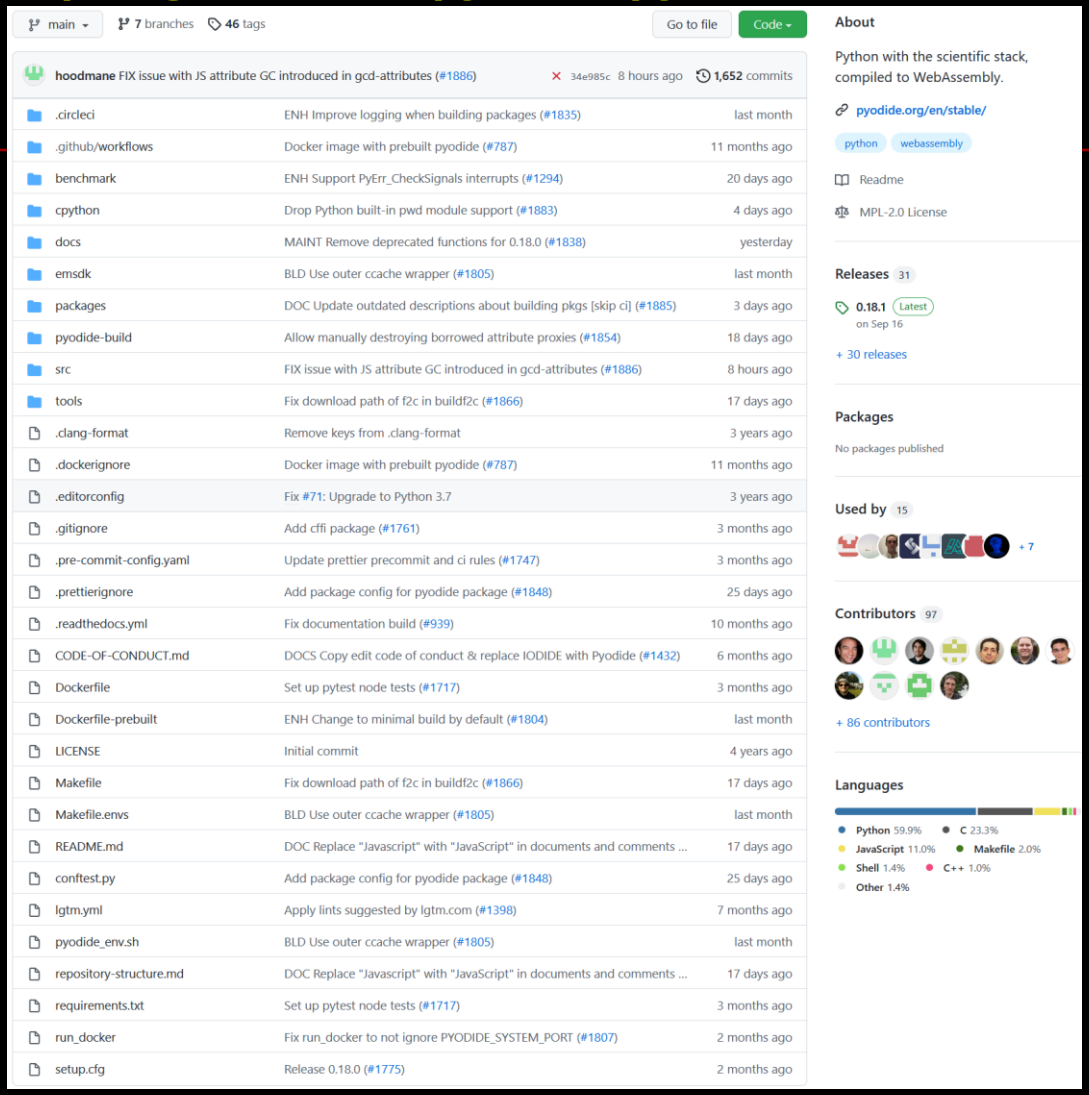
- https://pyodide.org
- Pyodide may be used in any context where you want to run Python inside a web browser.

  Pyodide brings the Python 3.9 runtime to the browser via WebAssembly, along with the Python scientific stack including NumPy, Pandas, Matplotlib, SciPy, and scikit-learn. The packages directory lists over 75 packages which are currently available. In addition it's possible to install pure Python wheels from PyPi.

  Pyodide provides transparent conversion of objects between JavaScript and Python. When used inside a browser, Python has full access to the Web APIs.
- **Should not support WASI yet.**
- **Officially only support X64, but it seems that we could add support for AArch as Emscripten SDK should be available on ARM soon.**
- …

Project

- https://github.com/pyodide/pyodide



main · 7 branches · 46 tags · Go to file · Code

hoodmane FIX issue with JS attribute GC introduced in gcd-attributes (#1886) — 34e985c 8 hours ago — 1,652 commits

.circleci	ENH Improve logging when building packages (#1835)	last month
.github/workflows	Docker image with prebuilt pyodide (#787)	11 months ago
benchmark	ENH Support PyErr_CheckSignals interrupts (#1294)	20 days ago
cpython	Drop Python built-in pwd module support (#1883)	4 days ago
docs	MAINT Remove deprecated functions for 0.18.0 (#1838)	yesterday
emsdk	BLD Use outer ccache wrapper (#1805)	last month
packages	DOC Update outdated descriptions about building pkgs [skip ci] (#1885)	3 days ago
pyodide-build	Allow manually destroying borrowed attribute proxies (#1854)	18 days ago
src	FIX issue with JS attribute GC introduced in gcd-attributes (#1886)	8 hours ago
tools	Fix download path of f2c in buildf2c (#1866)	17 days ago
.clang-format	Remove keys from .clang-format	3 years ago
.dockerignore	Docker image with prebuilt pyodide (#787)	11 months ago
.editorconfig	Fix #71: Upgrade to Python 3.7	3 years ago
.gitignore	Add cffi package (#1761)	3 months ago
.pre-commit-config.yaml	Update prettier precommit and ci rules (#1747)	3 months ago
.prettierignore	Add package config for pyodide package (#1848)	25 days ago
.readthedocs.yml	Fix documentation build (#939)	10 months ago
CODE-OF-CONDUCT.md	DOCS Copy edit code of conduct & replace IODIDE with Pyodide (#1432)	6 months ago
Dockerfile	Set up pytest node tests (#1717)	3 months ago
Dockerfile-prebuilt	ENH Change to minimal build by default (#1804)	last month
LICENSE	Initial commit	4 years ago
Makefile	Fix download path of f2c in buildf2c (#1866)	17 days ago
Makefile.envs	BLD Use outer ccache wrapper (#1805)	last month
README.md	DOC Replace "Javascript" with "JavaScript" in documents and comments ...	17 days ago
conftest.py	Add package config for pyodide package (#1848)	25 days ago
lgtm.yml	Apply lints suggested by lgtm.com (#1398)	7 months ago
pyodide_env.sh	BLD Use outer ccache wrapper (#1805)	last month
repository-structure.md	DOC Replace "Javascript" with "JavaScript" in documents and comments ...	17 days ago
requirements.txt	Set up pytest node tests (#1717)	3 months ago
run_docker	Fix run_docker to not ignore PYODIDE_SYSTEM_PORT (#1807)	2 months ago
setup.cfg	Release 0.18.0 (#1775)	2 months ago

About

Python with the scientific stack, compiled to WebAssembly.

pyodide.org/en/stable/

python · webassembly

Readme

MPL-2.0 License

Releases 31

0.18.1 Latest
on Sep 16

+ 30 releases

Packages

No packages published

Used by 15

+ 7

Contributors 97

+ 86 contributors

Languages

Python 59.9% · C 23.3% · JavaScript 11.0% · Makefile 2.0% · Shell 1.4% · C++ 1.0% · Other 1.4%

Stats

```
[mydev@fedora pyodide]$ git branch
* main
[mydev@fedora pyodide]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 BASH                    1          145          123            5           17
 Batch                   1           35           26            1            8
 C                      11         6092         4339         1198          555
 C Header               15         1416          719          518          179
 CSS                     1           23           16            3            4
 Dockerfile              1           44           32            4            8
 JavaScript             15         3114         1833         1121          160
 JSON                    6         3495         3495            0            0
 Makefile                5          471          333           35          103
 Python                149        19034        15755          733         2546
 ReStructuredText        1           73           55            0           18
 Shell                   5          407          160           66          181
 Plain Text              4          607            0          606            1
 TOML                    2            4            4            0            0
 TypeScript              2          159          138            1           20
 YAML                   86         1435         1299           42           94
-------------------------------------------------------------------------------
 HTML                    2           29           28            1            0
 |- CSS                  1            4            4            0            0
 |- JavaScript           2          145          134            6            5
 (Total)                            178          166            7            5
-------------------------------------------------------------------------------
 Markdown               40         4418            0         3360         1058
 |- BASH                 3           34           34            0            0
 |- C                    1           71           58           12            1
 |- HTML                 2           76           70            1            5
 |- JavaScript           5           63           58            5            0
 |- Python               4           35           29            2            4
 |- YAML                 1           17           14            0            3
 (Total)                           4714          263         3380         1071
===============================================================================
 Total                 347        41446        28756         7720         4970
===============================================================================
[mydev@fedora pyodide]$
```

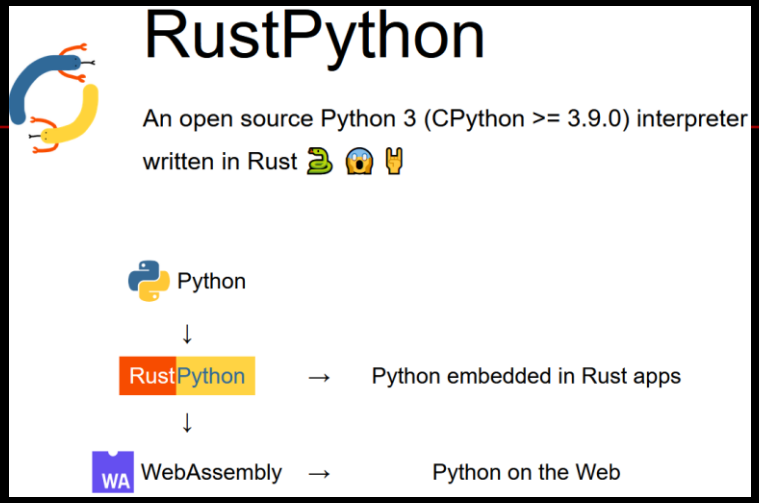
VII. Rust-based implementation

1) Overview

- …

2) RustPython

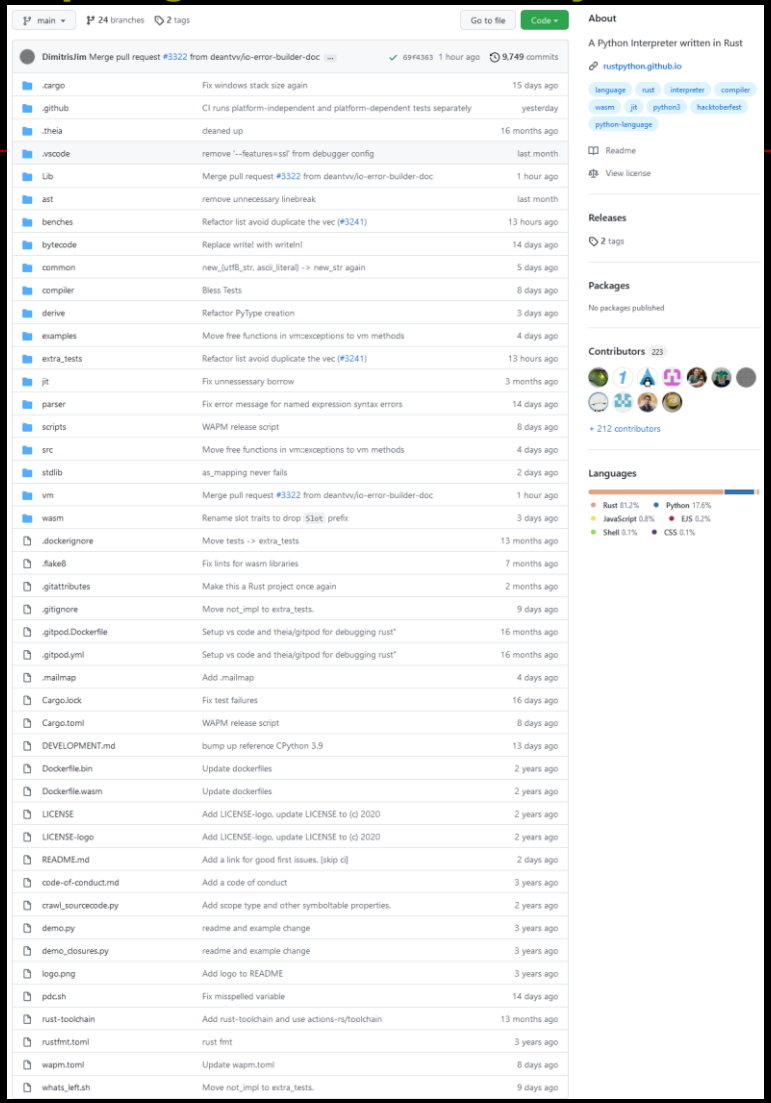
- https://rustpython.github.io/



- Should support WASI yet!
- Built-in support for any platform that support by Rust toolchain.
- …

Project

- https://github.com/RustPython/RustPython



main · 24 branches · 2 tags · Go to file · Code

DimitrisJim Merge pull request #3322 from deantvv/io-error-builder-doc … 69f4363 1 hour ago 9,749 commits

.cargo	Fix windows stack size again	15 days ago
.github	CI runs platform-independent and platform-dependent tests separately	yesterday
.theia	cleaned up	16 months ago
.vscode	remove '--features=ssl' from debugger config	last month
Lib	Merge pull request #3322 from deantvv/io-error-builder-doc	1 hour ago
ast	remove unnecessary linebreak	last month
benches	Refactor list avoid duplicate the vec (#3241)	13 hours ago
bytecode	Replace write! with writeln!	14 days ago
common	new_[utf8_str, ascii_literal] -> new_str again	5 days ago
compiler	Bless Tests	8 days ago
derive	Refactor PyType creation	3 days ago
examples	Move free functions in vm::exceptions to vm methods	4 days ago
extra_tests	Refactor list avoid duplicate the vec (#3241)	13 hours ago
jit	Fix unnessessary borrow	3 months ago
parser	Fix error message for named expression syntax errors	14 days ago
scripts	WAPM release script	8 days ago
src	Move free functions in vm::exceptions to vm methods	4 days ago
stdlib	as_mapping never fails	2 days ago
vm	Merge pull request #3322 from deantvv/io-error-builder-doc	1 hour ago
wasm	Rename slot traits to drop `Slot` prefix	3 days ago
.dockerignore	Move tests -> extra_tests	13 months ago
.flake8	Fix lints for wasm libraries	7 months ago
.gitattributes	Make this a Rust project once again	2 months ago
.gitignore	Move not_impl to extra_tests.	9 days ago
.gitpod.Dockerfile	Setup vs code and theia/gitpod for debugging rust"	16 months ago
.gitpod.yml	Setup vs code and theia/gitpod for debugging rust"	16 months ago
.mailmap	Add .mailmap	4 days ago
Cargo.lock	Fix test failures	16 days ago
Cargo.toml	WAPM release script	8 days ago
DEVELOPMENT.md	bump up reference CPython 3.9	13 days ago
Dockerfile.bin	Update dockerfiles	2 years ago
Dockerfile.wasm	Update dockerfiles	2 years ago
LICENSE	Add LICENSE-logo, update LICENSE to (c) 2020	2 years ago
LICENSE-logo	Add LICENSE-logo, update LICENSE to (c) 2020	2 years ago
README.md	Add a link for good first issues. [skip ci]	2 days ago
code-of-conduct.md	Add a code of conduct	3 years ago
crawl_sourcecode.py	Add scope type and other symboltable properties.	2 years ago
demo.py	readme and example change	3 years ago
demo_closures.py	readme and example change	3 years ago
logo.png	Add logo to README	3 years ago
pdc.sh	Fix misspelled variable	14 days ago
rust-toolchain	Add rust-toolchain and use actions-rs/toolchain	13 months ago
rustfmt.toml	rust fmt	3 years ago
wapm.toml	Update wapm.toml	8 days ago
whats_left.sh	Move not_impl to extra_tests.	9 days ago

About

A Python Interpreter written in Rust

rustpython.github.io

language · rust · interpreter · compiler · wasm · jit · python3 · hacktoberfest · python-language

Readme

View license

Releases

2 tags

Packages

No packages published

Contributors 223

+ 212 contributors

Languages

Rust 81.2% · Python 17.6% · JavaScript 0.8% · EJS 0.2% · Shell 0.1% · CSS 0.1%

Stats

```
[mydev@fedora RustPython-main]$ git branch
* main
[mydev@fedora RustPython-main]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 BASH                    2           56           44            4            8
 Batch                   2           54           39            0           15
 C Shell                 1           25           11            5            9
 CSS                     4          373          301            8           64
 Fish                    1           64           38           13           13
 JavaScript             12          982          774          120           88
 JSON                    4         2311         2311            0            0
 PowerShell              2          249          110          107           32
 Python               1120       507832       406766        38583        62483
 ReStructuredText        1          216          165            0           51
 Shell                  11          261          159           36           66
 Plain Text             17         5047            0         4718          329
 TOML                   15          564          474           21           69
 XSL                     1            5            5            0            0
 XML                    55          297          274            7           16
-------------------------------------------------------------------------------
 HTML                    2          531          506            0           25
 |- CSS                 1            7            7            0            0
 (Total)                             538          513            0           25
-------------------------------------------------------------------------------
 Markdown               12          847            0          589          258
 |- Python              2           12            8            1            3
 |- Shell               3           16           16            0            0
 |- TOML                1            5            5            0            0
 |- TypeScript          1           36           17           17            2
 (Total)                             916           46          607          263
-------------------------------------------------------------------------------
 Rust                 221        86575        76557         2242         7776
 |- Markdown          108         1564            6         1368          190
 (Total)                           88139        76563         3610         7966
===============================================================================
 Total                1483       607929       488593        47839        71497
===============================================================================
[mydev@fedora RustPython-main]$
```

Build and test on RPi4

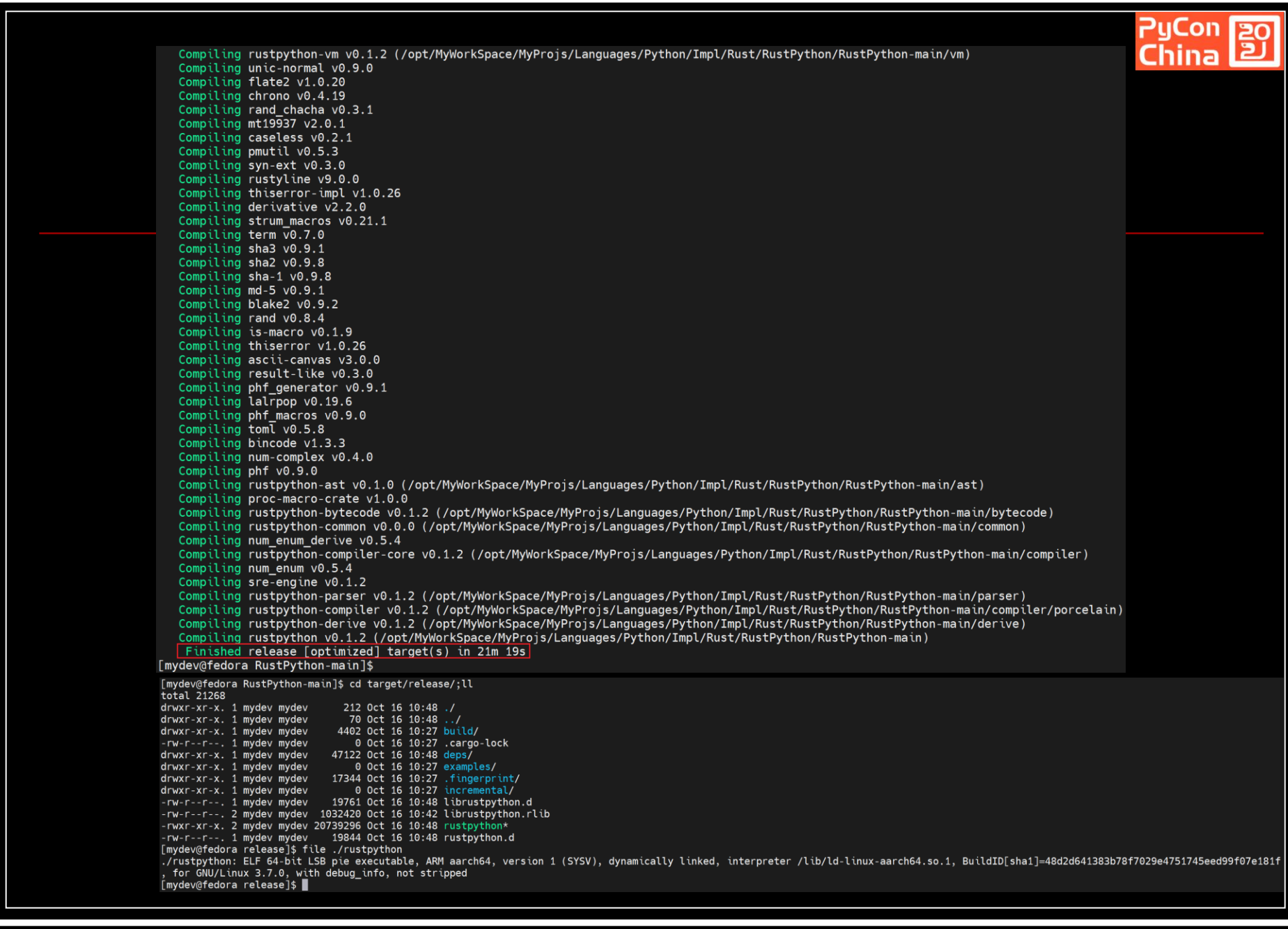
- Dev Env

```
[mydev@fedora RustPython-main]$ uname -a
Linux fedora 5.14.10-200.fc34.aarch64 #1 SMP Thu Oct 7 20:33:59 UTC 2021 aarch64 aarch64 aarch64 GNU/Linux
[mydev@fedora RustPython-main]$
[mydev@fedora RustPython-main]$ cat ~/.cargo/config
[source.crates-io]
registry = "https://github.com/rust-lang/crates.io-index"
replace-with = 'ustc'

[source.ustc]
registry = "https://mirrors.ustc.edu.cn/crates.io-index"

[source.sjtu]
registry = "https://mirrors.sjtug.sjtu.edu.cn/git/crates.io-index/"

[source.tuna]
registry = "https://mirrors.tuna.tsinghua.edu.cn/git/crates.io-index.git"

[source.rustcc]
registry = "https://code.aliyun.com/rustcc/crates.io-index.git"
[mydev@fedora RustPython-main]$
```

```
[mydev@fedora RustPython-main]$ rustup -V
rustup 1.24.3 (ce5817a94 2021-05-31)
info: This is the version for the rustup toolchain manager, not the rustc compiler.
info: The currently active `rustc` version is `rustc 1.55.0 (c8dfcfe04 2021-09-06)`
[mydev@fedora RustPython-main]$
[mydev@fedora RustPython-main]$ rustup toolchain list
stable-aarch64-unknown-linux-gnu (default) (override)
[mydev@fedora RustPython-main]$
```

Build

```
[mydev@fedora RustPython-main]$ cargo build --release
  Downloaded Inflector v0.11.4 (registry `ustc`)
  Downloaded siphasher v0.3.6 (registry `ustc`)
  Downloaded smawk v0.3.1 (registry `ustc`)
  Downloaded socket2 v0.4.1 (registry `ustc`)
  Downloaded sre-engine v0.1.2 (registry `ustc`)
  Downloaded string_cache v0.8.1 (registry `ustc`)
  Downloaded strum v0.21.0 (registry `ustc`)
  Downloaded strum_macros v0.21.1 (registry `ustc`)
  Downloaded subtle v2.4.1 (registry `ustc`)
  Downloaded syn-ext v0.3.0 (registry `ustc`)
  Downloaded textwrap v0.13.4 (registry `ustc`)
  Downloaded timsort v0.1.2 (registry `ustc`)
...
  Downloaded volatile v0.3.0 (registry `ustc`)
  Downloaded xml-rs v0.8.4 (registry `ustc`)
  Downloaded 82 crates (3.6 MB) in 27.05s
   Compiling autocfg v1.0.1
   Compiling unicode-xid v0.2.2
   Compiling proc-macro2 v1.0.28
   Compiling syn v1.0.75
   Compiling libc v0.2.102
   Compiling serde v1.0.127
...
   Compiling rustpython-pylib v0.1.0 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/vm/pylib-crate)
   Compiling static_assertions v1.1.0
   Compiling arrayvec v0.5.2
   Compiling itoa v0.4.7
   Compiling lalrpop-util v0.19.6
   Compiling crossbeam-utils v0.8.5
   Compiling endian-type v0.1.2
   Compiling unicode-segmentation v1.8.0
   Compiling smawk v0.3.1
   Compiling build_const v0.2.2
   Compiling hexf-parse v0.1.0
   Compiling maplit v1.0.2
   Compiling utf8parse v0.2.0
   Compiling matches v0.1.9
   Compiling subtle v2.4.1
   Compiling ascii v1.0.0
   Compiling volatile v0.3.0
   Compiling exitcode v1.1.2
   Compiling strum v0.21.0
   Compiling half v1.7.1
   Compiling timsort v0.1.2
   Compiling keccak v0.1.0
   Compiling adler32 v1.2.0
   Compiling hex v0.4.3
   Compiling paste v1.0.5
   Compiling rustpython-stdlib v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/stdlib)
   Compiling unicode-casing v0.1.0
...
```


PyCon China 2021
Compiling rustpython-vm v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/vm)
Compiling unic-normal v0.9.0
Compiling flate2 v1.0.20
Compiling chrono v0.4.19
Compiling rand_chacha v0.3.1
Compiling mt19937 v2.0.1
Compiling caseless v0.2.1
Compiling pmutil v0.5.3
Compiling syn-ext v0.3.0
Compiling rustyline v9.0.0
Compiling thiserror-impl v1.0.26
Compiling derivative v2.2.0
Compiling strum_macros v0.21.1
Compiling term v0.7.0
Compiling sha3 v0.9.1
Compiling sha2 v0.9.8
Compiling sha-1 v0.9.8
Compiling md-5 v0.9.1
Compiling blake2 v0.9.2
Compiling rand v0.8.4
Compiling is-macro v0.1.9
Compiling thiserror v1.0.26
Compiling ascii-canvas v3.0.0
Compiling result-like v0.3.0
Compiling phf_generator v0.9.1
Compiling lalrpop v0.19.6
Compiling phf_macros v0.9.0
Compiling toml v0.5.8
Compiling bincode v1.3.3
Compiling num-complex v0.4.0
Compiling phf v0.9.0
Compiling rustpython-ast v0.1.0 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/ast)
Compiling proc-macro-crate v1.0.0
Compiling rustpython-bytecode v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/bytecode)
Compiling rustpython-common v0.0.0 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/common)
Compiling num_enum_derive v0.5.4
Compiling rustpython-compiler-core v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/compiler)
Compiling num_enum v0.5.4
Compiling sre-engine v0.1.2
Compiling rustpython-parser v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/parser)
Compiling rustpython-compiler v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/compiler/porcelain)
Compiling rustpython-derive v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main/derive)
Compiling rustpython v0.1.2 (/opt/MyWorkSpace/MyProjs/Languages/Python/Impl/Rust/RustPython/RustPython-main)
Finished release [optimized] target(s) in 21m 19s
[mydev@fedora RustPython-main]$
[mydev@fedora RustPython-main]$ cd target/release/;ll
total 21268
drwxr-xr-x. 1 mydev mydev   212 Oct 16 10:48 ./
drwxr-xr-x. 1 mydev mydev   70 Oct 16 10:48 ../
drwxr-xr-x. 1 mydev mydev   4402 Oct 16 10:27 build/
-rw-r--r--. 1 mydev mydev   0 Oct 16 10:27 .cargo-lock
drwxr-xr-x. 1 mydev mydev   47122 Oct 16 10:48 deps/
drwxr-xr-x. 1 mydev mydev   0 Oct 16 10:27 examples/
drwxr-xr-x. 1 mydev mydev   17344 Oct 16 10:27 .fingerprint/
drwxr-xr-x. 1 mydev mydev   0 Oct 16 10:27 incremental/
-rw-r--r--. 1 mydev mydev   19761 Oct 16 10:48 librustpython.d
-rw-r--r--. 2 mydev mydev 1032420 Oct 16 10:42 librustpython.rlib
-rwxr-xr-x. 2 mydev mydev 20739296 Oct 16 10:48 rustpython*
-rw-r--r--. 1 mydev mydev   19844 Oct 16 10:48 rustpython.d
[mydev@fedora release]$ file ./rustpython
./rustpython: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-aarch64.so.1, BuildID[sha1]=48d2d641383b78f7029e4751745eed99f07e181f
, for GNU/Linux 3.7.0, with debug_info, not stripped
[mydev@fedora release]$

Test

```
[mydev@fedora release]$ ./rustpython --help
RustPython 0.1.2
RustPython Team
Rust implementation of the Python language

USAGE:
    rustpython [OPTIONS] [-c CMD | -m MODULE | FILE] [PYARGS]...

FLAGS:
    -b              issue warnings about using bytes where strings are usually expected (-bb: issue errors)
    -d              Debug the parser.
    -B              don't write .pyc files on import
    -h, --help      Prints help information
    -E              Ignore environment variables PYTHON* such as PYTHONPATH
    -i              Inspect interactively after running the script.
    -I              isolate Python from the user's environment (implies -E and -s)
    -S              don't imply 'import site' on initialization
    -s              don't add user site directory to sys.path.
    -O              Optimize. Set __debug__ to false. Remove debug statements.
    -q              Be quiet at startup.
    -u              force the stdout and stderr streams to be unbuffered; this option has no effect on stdin; also
                    PYTHONUNBUFFERED=x
    -V, --version   Prints version information
    -v              Give the verbosity (can be applied multiple times)

OPTIONS:
    -c <cmd, args>...                    run the given string as a program
    -X <implementation-option>...        set implementation-specific option
        --install-pip <get-pip args>...  install the pip package manager for rustpython
    -m <module, args>...                 run library module as script
    -W <warning-control>...              warning control; arg is action:message:category:module:lineno

ARGS:
    <script, args>...
[mydev@fedora release]$ ./rustpython
Welcome to the magnificent Rust Python 0.1.2 interpreter 😱 🖖
No previous history.
>>>>> 3+5
8
>>>>>
```

VIII. RPython-based implementation

1) Overview

- …

1.1 RPython

- https://rpython.readthedocs.io/en/latest/getting-started.html

RPython is a subset of Python that can be statically compiled. The PyPy interpreter is written mostly in RPython (with pieces in Python), while the RPython compiler is written in Python. The hard to understand part is that Python is a meta-programming language for RPython, that is, "being valid RPython" is a question that only makes sense on the live objects **after** the imports are done. This might require more explanation. You start writing RPython from `entry_point`, a good starting point is rpython/translator/goal/targetnopstandalone.py. This does not do all that much, but is a start. Now if code analyzed (in this case `entry_point`) calls some functions, those calls will be followed. Those followed calls have to be RPython themselves (and everything they call etc.), however not entire module files. To show how you can use metaprogramming, we can do a silly example (note that closures are not RPython):

```
def generator(operation):
    if operation == 'add':
       def f(a, b):
           return a + b
    else:
       def f(a, b):
           return a - b
    return f

add = generator('add')
sub = generator('sub')

def entry_point(argv):
    print add(sub(int(argv[1]), 3) 4)
    return 0
```

In this example `entry_point` is RPython, `add` and `sub` are RPython, however, `generator` is not.

...

- ...

2) PyPy

- https://www.pypy.org/

PyPy is a replacement for CPython. It is built using the RPython language that was co-developed with it. The main reason to use it instead of CPython is speed: it runs generally faster (see next section).

PyPy implements **Python 2.7.18, and 3.7.10**. It supports all of the core language, passing the Python 2.7 test suite and almost all of the 3.7 test suite (with minor modifications) It supports most of the commonly used Python standard library modules. For known differences with CPython, see our compatibility page.

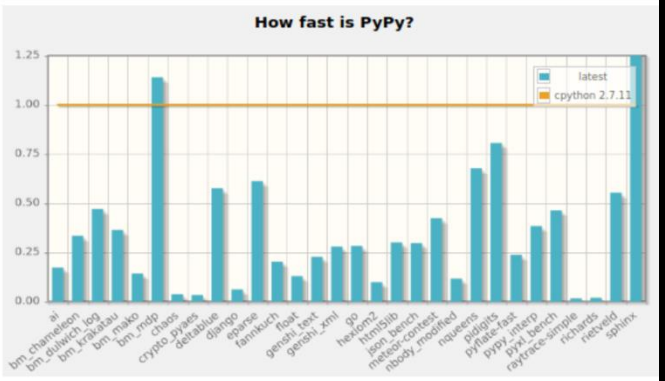
The following CPU architectures are supported and maintained:

- x86 (IA-32) and x86_64
- ARM platforms (ARMv6 or ARMv7, with VFPv3)
- AArch64
- PowerPC 64bit both little and big endian
- System Z (s390x)

PyPy's x86 version runs on several operating systems, such as Linux (32/64 bits), Mac OS X (64 bits), Windows (32 bits), OpenBSD, FreeBSD. All non-x86 versions are only supported on Linux.

- Performance

On average, PyPy is **4.2 times faster** than CPython



PyPy (with JIT) benchmark times normalized to CPython. Smaller is better. Based on the geometric average of all benchmarks

- ...

Features

Speed

Our main executable comes with a Just-in-Time compiler. It is really fast in running most benchmarks —including very large and complicated Python applications, not just 10-liners.

There are two cases that you should be aware where PyPy will *not* be able to speed up your code:

- Short-running processes: if it doesn't run for at least a few seconds, then the JIT compiler won't have enough time to warm up.
- If all the time is spent in run-time libraries (i.e. in C functions), and not actually running Python code, the JIT compiler will not help.

So the case where PyPy works best is when executing long-running programs where a significant fraction of the time is spent executing Python code. This is the case covered by the majority of our benchmarks, but not all of them --- the goal of PyPy is to get speed but still support (ideally) any Python program.

Memory usage

Memory-hungry Python programs (several hundreds of MBs or more) might end up taking less space than they do in CPython. It is not always the case, though, as it depends on a lot of details. Also note that the baseline is higher than CPython's.

Stackless

Support for Stackless and greenlets are now integrated in the normal PyPy. More detailed information is available here.

Other features

PyPy has many secondary features and semi-independent projects. We will mention here:

- **Other languages:** we also implemented other languages that makes use of our RPython toolchain: Prolog (almost complete), as well as Smalltalk, JavaScript, Io, Scheme and Gameboy.

  There is also a Ruby implementation called Topaz and a PHP implementation called HippyVM.

Sandboxing

PyPy's *sandboxing* is a working prototype for the idea of running untrusted user programs. Unlike other sandboxing approaches for Python, PyPy's does not try to limit language features considered "unsafe". Instead we replace all calls to external libraries (C or platform) with a stub that communicates with an external process handling the policy.

> Note
>
> **Please be aware that it is a prototype only.** It needs work to become more complete, and you are welcome to help. In particular, almost none of the extension modules work (not even `time`), and `pypy_interact` is merely a demo. Also, a more complete system would include a way to do the same as `pypy_interact` from other languages than Python, to embed a sandboxed interpreter inside programs written in other languages.

To run the sandboxed process, you need to get the full sources and build `pypy-sandbox` from it (see Building from source). These instructions give you a `pypy-c` that you should rename to `pypy-sandbox` to avoid future confusion. Then run:

```
cd pypy/sandbox
pypy_interact.py path/to/pypy-sandbox
# don't confuse it with pypy/goal/pyinteractive.py!
```

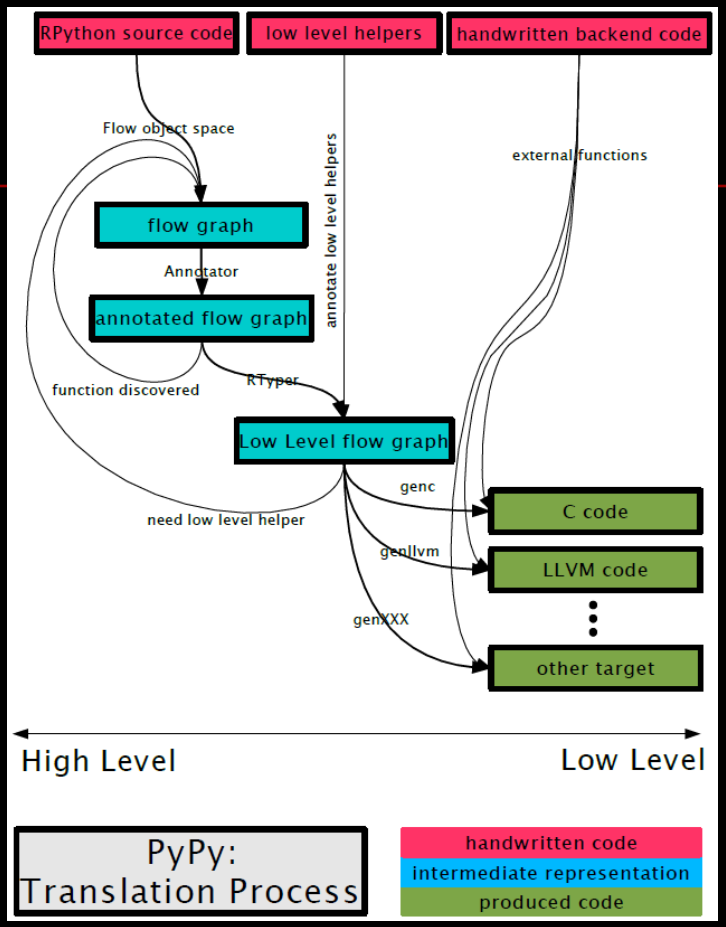
Technologies

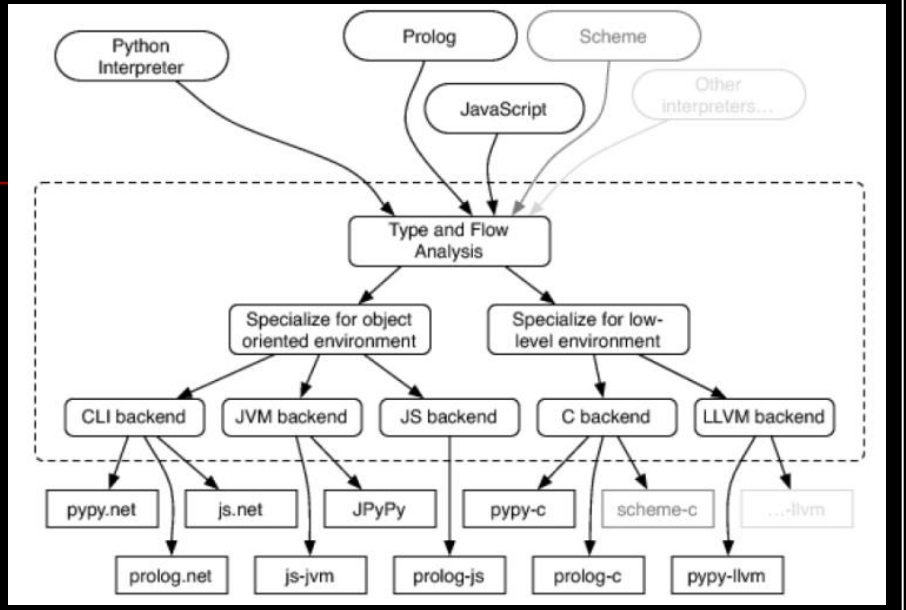
- **Meta-tracing**

  http://stefan-marr.de/papers/oopsla-marr-ducasse-meta-tracing-vs-partial-evaluation-artifacts/submitted-draft.pdf

- …

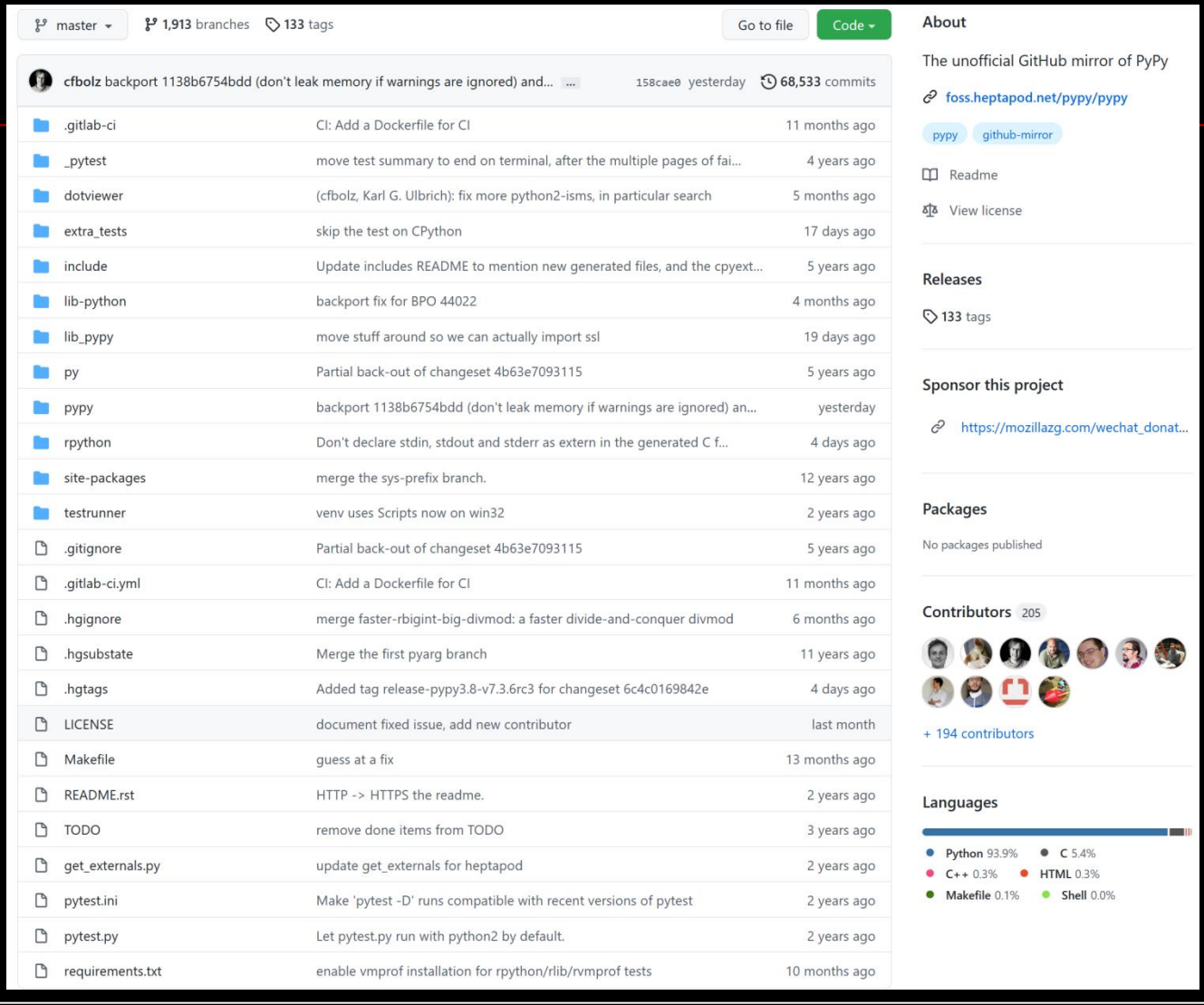
How it works


RPython source code
low level helpers
handwritten backend code
Flow object space
annotate low level helpers
external functions
flow graph
Annotator
annotated flow graph
function discovered
RTyper
Low Level flow graph
genc
C code
need low level helper
genllvm
LLVM code
genXXX
other target
High Level
Low Level
PyPy:
Translation Process
handwritten code
intermediate representation
produced code


Python Interpreter
Prolog
Scheme
JavaScript
Other interpreters...
Type and Flow Analysis
Specialize for object oriented environment
Specialize for low-level environment
CLI backend
JVM backend
JS backend
C backend
LLVM backend
pypy.net
js.net
JPyPy
pypy-c
scheme-c
...-llvm
prolog.net
js-jvm
prolog-js
prolog-c
pypy-llvm

Project (The unofficial GitHub mirror of PyPy)

- https://github.com/mozillazg/pypy



master · 1,913 branches · 133 tags · Go to file · Code

cfbolz backport 1138b6754bdd (don't leak memory if warnings are ignored) and... 158cae0 yesterday 68,533 commits

.gitlab-ci	CI: Add a Dockerfile for CI	11 months ago
_pytest	move test summary to end on terminal, after the multiple pages of fai...	4 years ago
dotviewer	(cfbolz, Karl G. Ulbrich): fix more python2-isms, in particular search	5 months ago
extra_tests	skip the test on CPython	17 days ago
include	Update includes README to mention new generated files, and the cpyext...	5 years ago
lib-python	backport fix for BPO 44022	4 months ago
lib_pypy	move stuff around so we can actually import ssl	19 days ago
py	Partial back-out of changeset 4b63e7093115	5 years ago
pypy	backport 1138b6754bdd (don't leak memory if warnings are ignored) an...	yesterday
rpython	Don't declare stdin, stdout and stderr as extern in the generated C f...	4 days ago
site-packages	merge the sys-prefix branch.	12 years ago
testrunner	venv uses Scripts now on win32	2 years ago
.gitignore	Partial back-out of changeset 4b63e7093115	5 years ago
.gitlab-ci.yml	CI: Add a Dockerfile for CI	11 months ago
.hgignore	merge faster-rbigint-big-divmod: a faster divide-and-conquer divmod	6 months ago
.hgsubstate	Merge the first pyarg branch	11 years ago
.hgtags	Added tag release-pypy3.8-v7.3.6rc3 for changeset 6c4c0169842e	4 days ago
LICENSE	document fixed issue, add new contributor	last month
Makefile	guess at a fix	13 months ago
README.rst	HTTP -> HTTPS the readme.	2 years ago
TODO	remove done items from TODO	3 years ago
get_externals.py	update get_externals for heptapod	2 years ago
pytest.ini	Make 'pytest -D' runs compatible with recent versions of pytest	2 years ago
pytest.py	Let pytest.py run with python2 by default.	2 years ago
requirements.txt	enable vmprof installation for rpython/rlib/rvmprof tests	10 months ago

About

The unofficial GitHub mirror of PyPy

foss.heptapod.net/pypy/pypy

pypy · github-mirror

Readme

View license

Releases

133 tags

Sponsor this project

https://mozillazg.com/wechat_donat...

Packages

No packages published

Contributors 205

+ 194 contributors

Languages

Python 93.9% · C 5.4% · C++ 0.3% · HTML 0.3% · Makefile 0.1% · Shell 0.0%

Stats

```
[mydev@fedora pypy-master]$ git branch
* master
[mydev@fedora pypy-master]$ tokei
```

Language	Files	Lines	Code	Comments	Blanks
Assembly	2	145	97	18	30
Autoconf	3	970	726	107	137
Automake	1	136	71	36	29
Batch	5	451	392	3	56
C	141	48649	37557	4988	6104
C Header	185	35486	28756	3697	3033
C++	14	2778	2235	121	422
CSS	1	63	49	0	14
Emacs Lisp	1	62	45	7	10
HCL	2	13	13	0	0
INI	3	4	4	0	0
JSON	3	1663	1663	0	0
Makefile	6	499	394	14	91
Markdown	2	46	0	32	14
Module-Definition	5	1257	1191	0	66
Python	4609	1859516	1611497	83933	164086
ReStructuredText	202	29112	21156	0	7956
Shell	6	677	460	134	83
Plain Text	399	609697	0	605822	3875
VBScript	1	1	0	1	0
Vim script	1	33	31	2	0
XML	17	433	313	4	116
HTML	4	1477	1424	6	47
|- CSS	2	34	34	0	0
|- JavaScript	2	17	16	1	0
(Total)		1528	1474	7	47
Total	5613	2593219	1708124	698926	186169

```
[mydev@fedora pypy-master]$
```

IX. Comparison

1) Overview

- https://en.wikipedia.org/wiki/Python_(programming_language)

2) Benchmarking

2.1 Pyperformance

- https://github.com/python/pyperformance
- Python Performance Benchmark Suite (mainly targets CPython)
- https://pyperformance.readthedocs.io/
- https://pyperformance.readthedocs.io/usage.html#run-benchmarks

  ...
- Be fit for standalone Python runtime, but not friendly for cases like IronPython, Pyjion, RustPython...

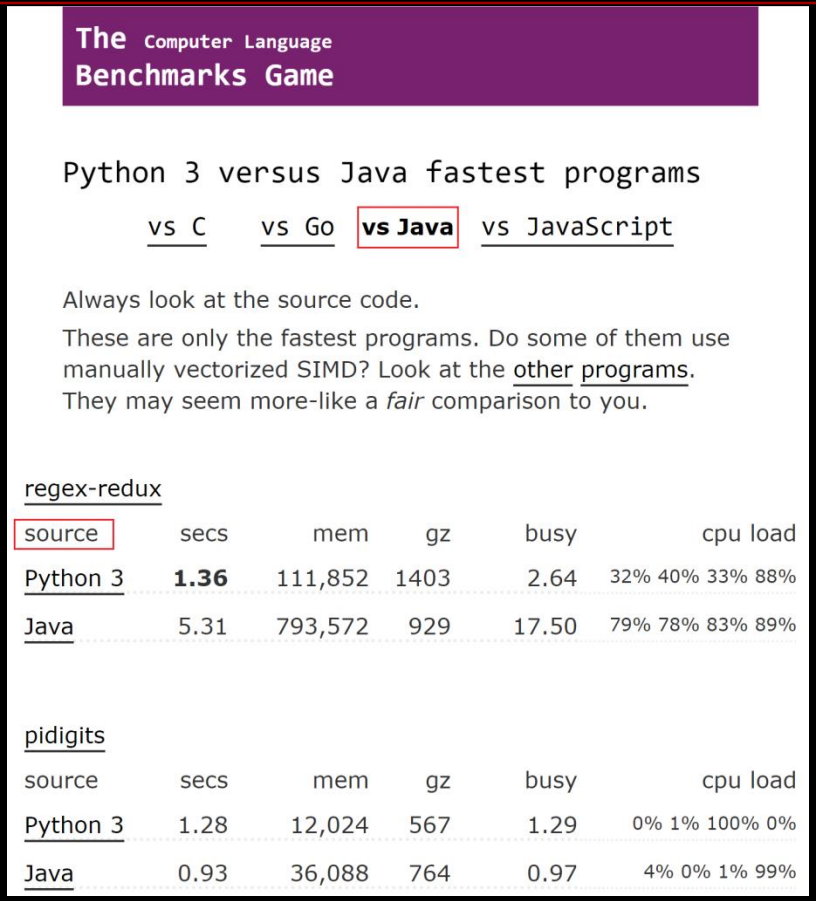
Summary

- Prebuilt PyPy for Linux AArch64 is not stable during the test, crash or timeout often occurred.
- RustPython is awesome, but it is still in development and lacks of some features to be used as a standalone Python runtime for testing.
- Made IronPython successfully running on RPi4B, but still got failed to run Pyperformance since the latter does not well handle the case of “Python executable” which need a “wrapper” (just like Mono for IronPython).

2.2 Computer Language Benchmarks Game

Python

- **httpshttps://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python.html**



The Computer Language Benchmarks Game

Python 3 versus Java fastest programs

vs C vs Go **vs Java** vs JavaScript

Always look at the source code.

These are only the fastest programs. Do some of them use manually vectorized SIMD? Look at the other programs. They may seem more-like a *fair* comparison to you.

regex-redux

source	secs	mem	gz	busy	cpu load
Python 3	**1.36**	111,852	1403	2.64	32% 40% 33% 88%
Java	5.31	793,572	929	17.50	79% 78% 83% 89%

pidigits

source	secs	mem	gz	busy	cpu load
Python 3	1.28	12,024	567	1.29	0% 1% 100% 0%
Java	0.93	36,088	764	0.97	4% 0% 1% 99%

...

- **https://benchmarksgame-team.pages.debian.net/benchmarksgame/how-programs-are-measured.html**

Measured on a quad-core 3.0GHz Intel® i5-3330® with 15.8 GiB of RAM and 2TB SATA disk drive; using Ubuntu™ 21.04 x86_64 GNU/Linux 5.11.0-18-generic.

No doubt on current hardware all the programs would be faster. Would that make the measurements more informative or just different?

Who wrote the programs

The programs have been crowd sourced, contributed to the project by an ever-changing self-selected group.

How programs are measured

1. Each program is run and measured at the smallest input value, program output redirected to a file and compared to expected output. As long as the output matches expected output, the program is then run and measured at the next larger input value until measurements have been made at every input value.
2. If the program gives the expected output within an arbitrary cutoff time (120 seconds) the program is measured again (5 more times) with output redirected to `/dev/null`.
3. If the program doesn't give the expected output within an arbitrary timeout (usually one hour) the program is forced to quit. If measurements at a smaller input value have been successful within an arbitrary cutoff time (120 seconds), the program is measured again (5 more times) at that smaller input value, with output redirected to `/dev/null`.
4. The measurements shown on the website are either:
   - within the arbitrary cutoff - the lowest time and highest memory use from 6 measurements
   - outside the arbitrary cutoff - the sole time and memory use measurement
5. For sure, programs taking 4 and 5 hours were only measured once!

How programs are timed

Each program is run as a child-process of a Python script using `Popen`:

- secs - The time is taken before forking the child-process and after the child-process exits, using `time.time()`
- cpu - The script child-process `usr+sys rusage` time is taken using `os.wait3`. Rarely (for example OCaml), that may not measure all processes forked from the script child-process.
- busy - The GTop cpu idle and GTop cpu total are taken before forking the child-process and after the child-process exits. The sum of GTop cpu not-idle for each core, scaled by secs.

On win32:

- secs - The time is taken before forking the child-process and after the child-process exits, using `QueryPerformanceCounter`
- cpu - `QueryInformationJobObject(hJob,JobObjectBasicAccountingInformation)` `TotalKernelTime + TotalUserTime`

(Note: Those measurements include startup time).

How program memory use is measured

By sampling `GLIBTOP_PROC_MEM_RESIDENT` for the program and it's child processes every 0.04 seconds. Obviously those measurements are unlikely to be reliable for programs that run for less than 0.04 seconds.

On win32: `QueryInformationJobObject(hJob,JobObjectExtendedLimitInformation)` `PeakJobMemoryUsed`

How source code size is measured

We start with the source-code markup you can see, remove comments, remove duplicate whitespace characters, and then apply minimum GZip compression. The measurement is the size in bytes of that GZip compressed source-code file.

Thanks to Brian Hurt for the idea of using **size of compressed source code** instead of lines of code.

median source code gzip (July 2018)	
Perl	513
TypeScript	532
Lua	553
Ruby	568
Dart	610
Chapel	632
Racket	638
Python	672
PHP	736
Hack	745
JavaScript	779
Erlang	792
Go	829
Haskell	842
Pascal	846
F#	876
OCaml	914
Java	945
Smalltalk	950
Lisp	1004
Fortran	1019
C++	1044
C#	1059
C	1115
Swift	1164
Rust	1319
Ada	1819

(Note: There is some evidence that complexity metrics don't provide any more information than SLoC or LoC.)

How CPU load is measured

The GTop cpu idle and GTop cpu total are taken before forking the child-process and after the child-process exits. The percentages represent the proportion of cpu not-idle to cpu total for each core.

On win32: `GetSystemTimes` `UserTime` `IdleTime` are taken before forking the child-process and after the child-process exits. The percentage represents the proportion of `TotalUserTime` to `UserTime + IdleTime` (because that's like the percentage you'll see in Task Manager).

- https://benchmarksgame-team.pages.debian.net/benchmarksgame/measurements/python3.html

all Python 3 programs & measurements

File system caches and swap are cleared before measurements are made for each program — so each program has a similiar initial context. That makes the first measurements (the smallest N workload) different from later measurements.

Python 3.9.2

source	secs	N	mem	gz	cpu	cpu load
binary-trees #2	0.54	7	7,204	338	0.03	39% 0% 6% 24%
binary-trees #2	0.69	14	9,828	338	0.69	3% 0% 0% 100%
binary-trees #2	148.16	21	274,244	338	148.11	0% 0% 0% 100%

source	secs	N	mem	gz	cpu	cpu load
binary-trees #4	0.96	7	11,572	472	0.13	8% 7% 13% 84%
binary-trees #4	0.34	14	13,100	472	0.94	74% 65% 75% 82%
binary-trees #4	48.03	21	462,732	472	173.57	89% 97% 88% 89%

...

2.3 Multilingual

- https://github.com/kostya/benchmarks

X. Wrap-up

- As the gap among different implementations against Python Specs is not small and CPython is the de facto standard, so continuously improve CPython is the most practical solution, especially for bringing JIT to CPython.
- GraalPython is the most attractive candidate in the near future.
- From performance point of view, GraalPython and RustPython probably have the highest growth potential in the future.
- Pyodide have great potential especially when add support for WASI.
- Though the performance improvement of PyPy seems have reached its limit, it comes with some advanced features.
- Pyjion is awesome, it seems provide a best way to keep pace with CPython, and is especially suited to the cloud.

- From the perspective of hardware platform support, RustPython and Pyjion have a ”built-in” advantage.
- If you are interested in some good Python-based open source projects, or the technologies like GraalVM, Wasm, .Net, Rust, IoT Edge, Serverless and so on, you may refer to our previous talks (all the slides are put at https://github.com/XianBeiTuoBaFeng2015/MySlides), and some new ones(includes this slides) will be uploaded soon.
- Please look forward to our follow-ups like “Revisiting current Python implementations” and more…

PyCon China 2021

Q & A

2021 中国 PYTHON 开发者大会

Thanks!

感谢您的观看

2021.10.16-17

Reference

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/
- https://en.wikipedia.org/wiki/Runtime_(program_lifecycle_phase)
- https://en.wikipedia.org/wiki/Dhrystone
- …