Python, Go and Rust for Network Automation

A lovely versus battle

ConFLOSS
Conferência de Free/Libre e Open Source Software

Claus Töpke - linkedin.com/in/claus-topke - claus@telcomanager.com

Who am I ?

20+ years experience
Network Operator and Engineer
Network Architect
Software Developer
Product Developer & Language explorer
Technology Writer
Network Automation

ConFLOSS
Conferência de Free/Libre e Open Source Software

Disclaimer

Most of the slides are for reference, and we are going to skim.

No intention to give Software Engineering or Coding guidance

Intended for NEs, DevOps and SDEs

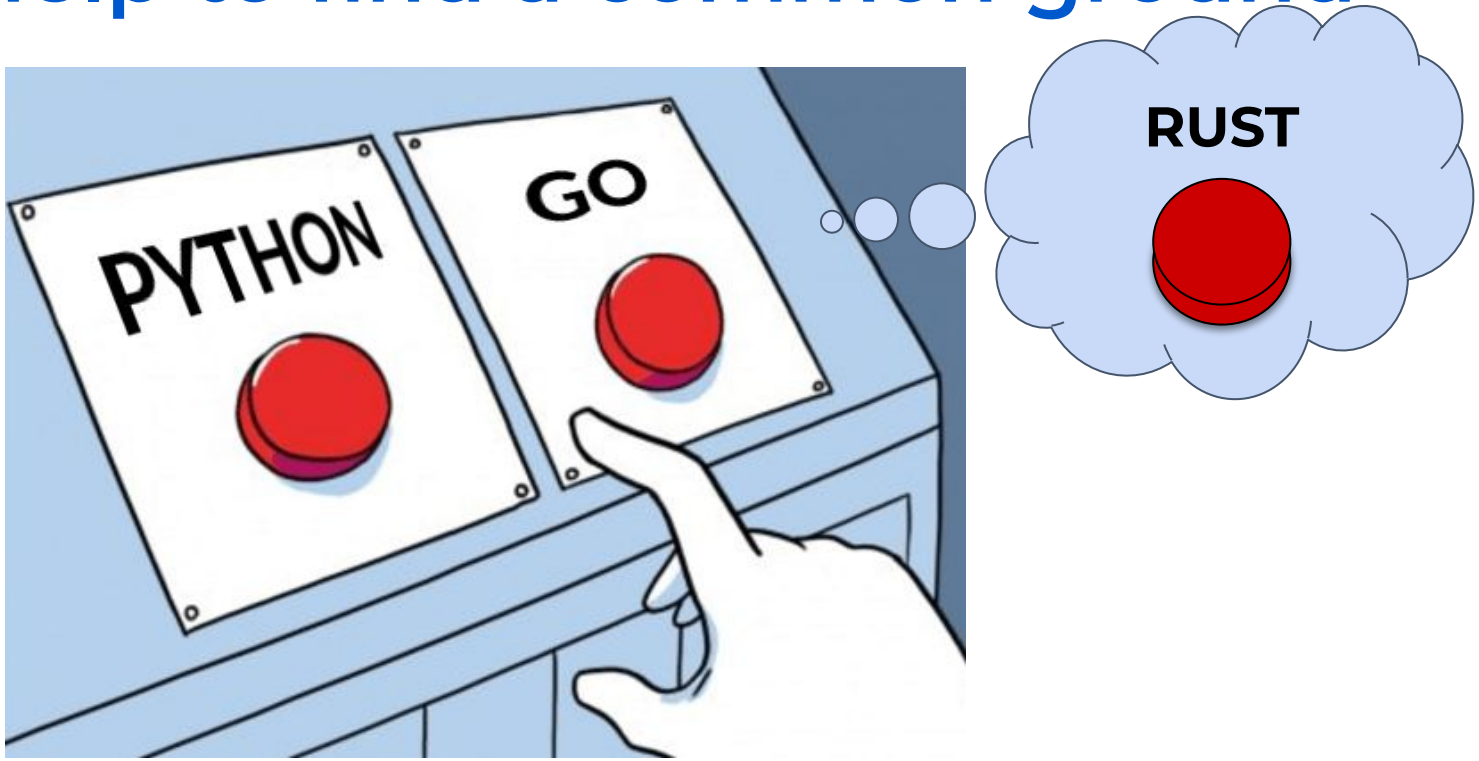
No affiliation to any of the language creators or maintainers

A bit shallow but deep enough to offer insights for comparing

Based on years of experience and recent research

Some images were taken from Internet, URLs of original included when possible

Let’s help to find a common ground



qit.software/go-vs-python-comparing-performance-features/

The versus table score

	Python	Go	Rust
Language release and core development			
Implementations			
Memory, speed and parallelism			
Deployment and dependencies			
Program Logging, Error handling and Exceptions			
IP and network native libraries			
Network access methods and tools			
Network Automation community tools			

Language release cycle and implementations

Python language Release Timeline

1989	1991	1994	2000	2008	2010	2018	2021	2023	2024
	0.9	1.0	2.0	2.6 3.0	2.7 3.1	2.7.15 3.7.0	3.10	3.12	3.13

by Guido van Rossum

en.wikipedia.org/wiki/History_of_Python

www.python.org/doc/sunset-python-2/ ← Jan/2020

www.python.org/downloads/release/python-3120/

www.python.org/psf/sponsors/

ConFLOSS Conferência de Free/Libre e Open Source Software

us.pycon.org/2024/

www.etsy.com/hk-en/listing/1241340884/python-plushie-python-plush-python-plush

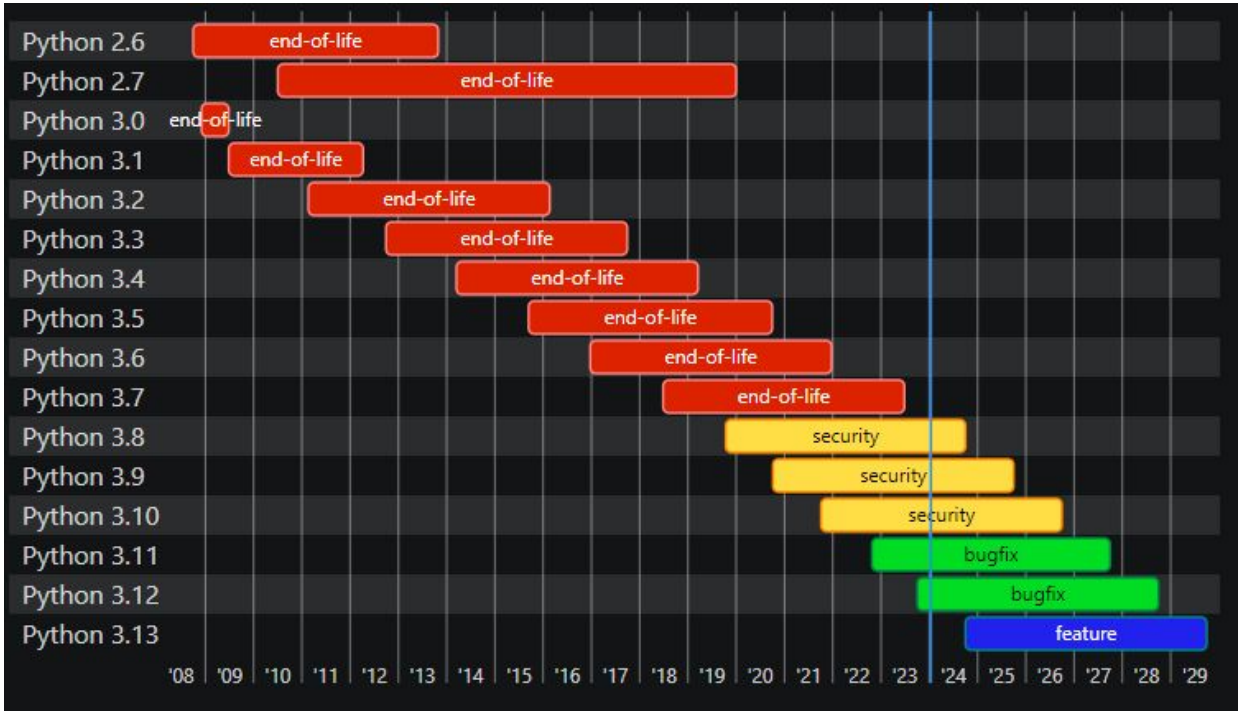
Python language Release Cycle

Python uses follows Semantic Versioning. So major.minor.micro for production-ready releases. So for Python 3.1.2 final, that is a major version of 3, a minor version of 1, and a micro version of 2.

- ➔ new major versions are exceptional; they only come when strongly incompatible changes are deemed necessary;
- ➔ new minor versions are feature releases; they get released annually, from the current in-development branch;
- ➔ new micro versions are bugfix releases; they get released roughly every 2 months

devguide.python.org/developer-workflow/development-cycle/

Python language Release Cycle



devguide.python.org/versions/

Some Python implementations

CPython (github.com/python/cpython) Pre Python release

PyPy (www.pypy.org/ and foss.heptapod.net/pypy/pypy)

Numba (numba.pydata.org/)

IronPython (.NET) (ironpython.net/)

Jython (JVM) (github.com/jython/jython)

MicroPython (github.com/micropython/micropython/)

IPython (ipython.org/)

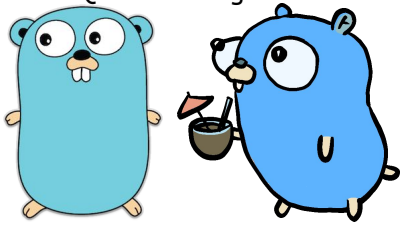
Example Jupyter notebook

Iteratively work with web interface with IPython (jupyter.org/)

Share easily notebooks

Great for Proof of Concepts for Network Automation

Go language Release Timeline



2009	2011	2012	2014	2016	2019	2021	2023	2024
Weekly Snapshots	r56	1.0	1.4	1.6	1.12	1.17	1.21	1.23

by Robert Griesemer, Rob Pike, and Ken Thompson

go.dev/doc/devel/weekly

groups.google.com/g/golang-dev/

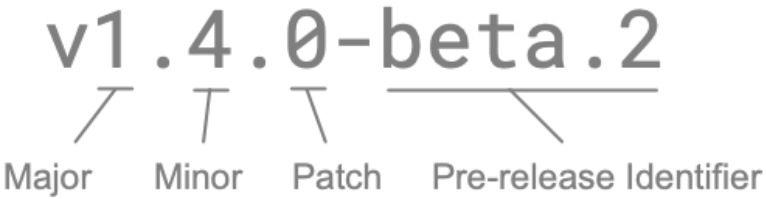
go.dev/blog

go.dev/blog/14years

go.dev/blog/go119runtime

www.amazon.com/Desktop-Decor-Golang-Mascot-Gopher/dp/B0C1CSSV2Z

Go language Version Numbers

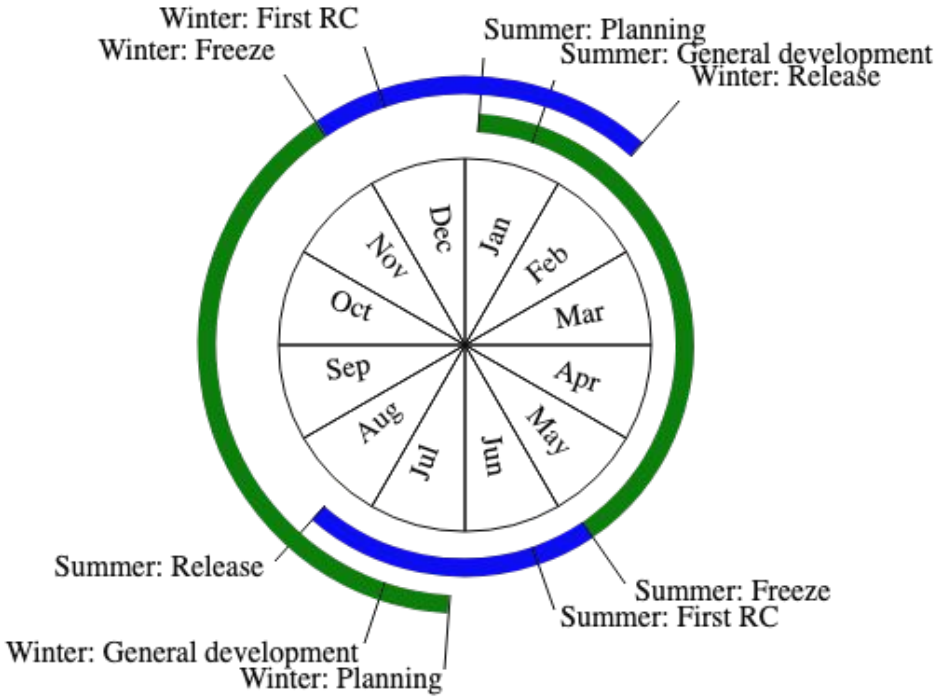


Major version	**v1.x.x**	**Signals backward-incompatible public API changes. This release carries no guarantee that it will be backward compatible with preceding major versions.**
Minor version	**vx.4.x**	**Signals backward-compatible public API changes. This release guarantees backward compatibility and stability.**
Patch version	**vx.x.1**	**Signals changes that don't affect the module's public API or its dependencies. This release guarantees backward compatibility and stability.**
Pre-release version	**vx.x.x-beta.2**	**Signals that this is a pre-release milestone, such as an alpha or beta. This release carries no stability guarantees.**

ConFLOSS
Conferência de Free/Libre e Open Source Software

go.dev/doc/modules/version-numbers

Go language Release Cycle



zchee.github.io/golang-wiki/Go-Release-Cycle/

Go Implementations

gc (original compiler written in C, but now in Go)

gccgo (github.com/golang/go)

llgo (go1.3 - 9 y old github.com/go-llvm/llgo moved to llvm)

tinygo (go1.21 tinygo.org/

- Combination of gccgo tools and llvm llvm.org/)

gopherjs (go1.18 github.com/gopherjs/gopherjs)

go.dev/doc/faq#What_compiler_technology_is_used_to_build_the_compilers

Go Kernel/OS Implementations

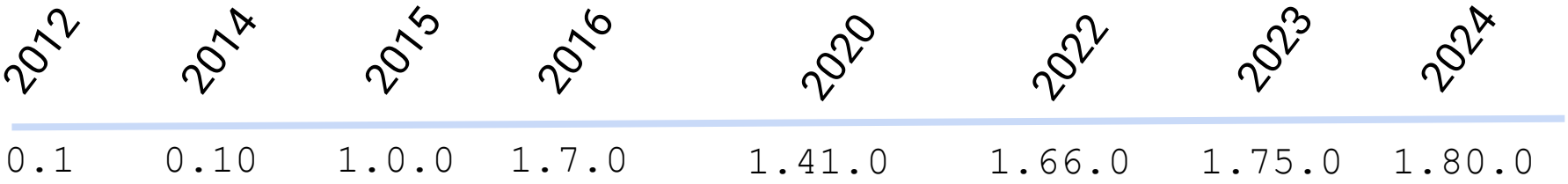
gVisor (github.com/google/gvisor)

gopher-os(github.com/gopher-os/gopher-os 6 years)

biscuit (github.com/mit-pdos/biscuit 6 years)

clive (github.com/fjballest/clive 8 years)

Rust language Release timeline



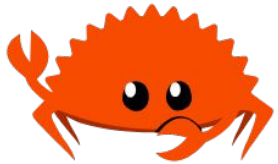
By Graydon Hoare at Mozilla (2006-2012)

www.rust-lang.org/

foundation.rust-lang.org/

doc.rust-lang.org/book/

engineering.fb.com/2021/04/29/developer-tools/rust/



rustacean.net/

ConFLOSS

forge.rust-lang.org/release/process.html

Rust language Version Numbers

Rust follows Semantic Versioning docs.rs/version-number/latest/version_number/

MAJOR version when you make incompatible API changes
MINOR version when you add functionality in a backward compatible manner
PATCH version when you make backward compatible bug fixes

Version 1.74.0 (2023-11-16)
Version 1.74.1 (2023-12-07)

Resolved spurious STATUS_ACCESS_VIOLATIONs in LLVM

Clarify guarantees for std::mem::discriminant

Fix some subtyping-related regressions

Rust language Release Cycle

Minor version every 6 weeks (42 days). releases.rs/

August 2024 Status

Stable: 1.80.0

Beta: 1.81.0 (5 September, 2024, 14 days left)

Nightly: 1.82.0 (17 October, 2024, 56 days left)

Rust Implementations

rustc (default compiler written in Rust)

OS and Kernels:

- **Remit** (github.com/hermit-os/kernel)
- **Cluu** (github.com/valibali/cluu experimental)
- **Redox** (OS/Kernel written in Rust www.redox-os.org/)

Side track:

- **Py03** (github.com/PyO3/pyo3)

The versus table score

Language Release and code development:

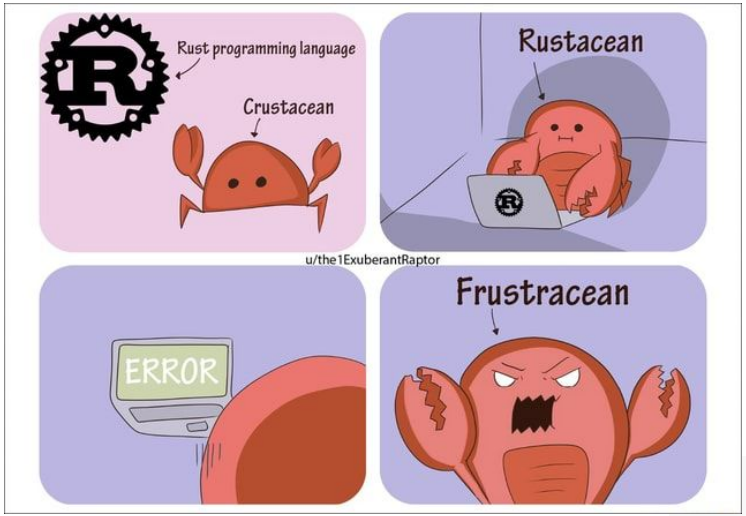
- Python, Go and Rust got **10**.

Implementation:

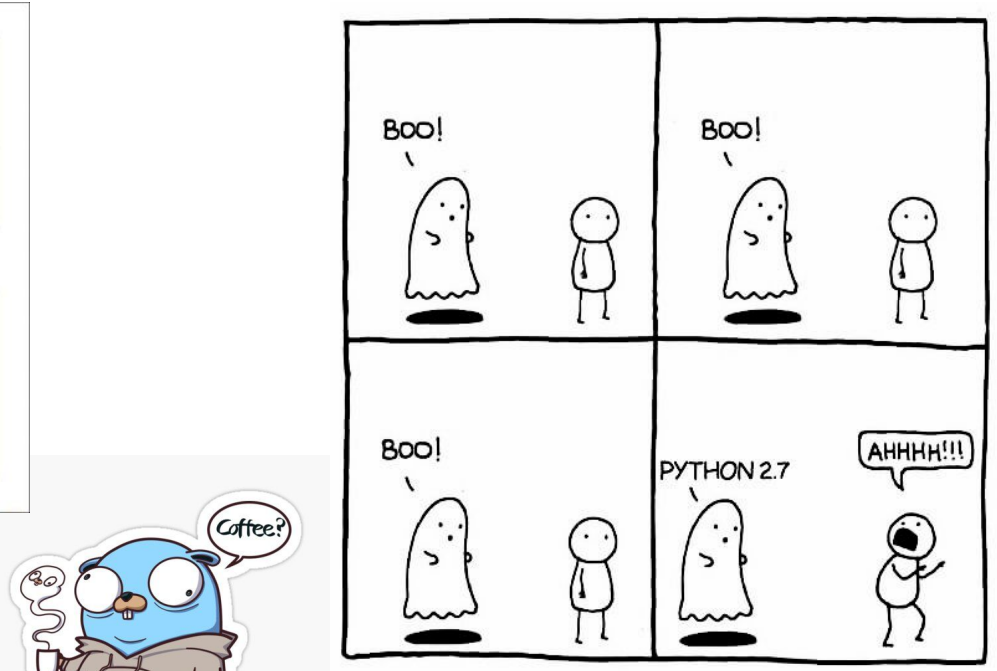
- Python got **9** (Can’t build a self contained system)
- GO got **8** (Not many as Python, but can have OS/Kernel)
- Rust got **7** (Not many as Go, but can have OS/Kernel)

The versus table score

	Python	Go	Rust
Language release and core development	10	10	10
Implementations	9	8	7
Memory, speed and parallelism			
Deployment and dependencies			
Program Logging, Error handling and Exceptions			
IP and network native libraries			
Network access methods and tools			
Network Automation community tools			



br.ifunny.co/picture/rustacean-crustacean-rust-programming-language-trapter-frustracean-exuberant



www.python.org/doc/sunset-python-2/

http://www.redbubble.com/people/clgtar

Memory, speed, parallelism and network I/O

Interpreted and Compiled

- Go and Rust are compiled
  - Run compiler and create a binary file static linked

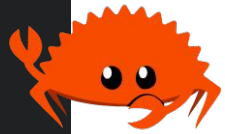
```
 # file countdown
countdown: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
statically linked
```

- Python is interpreted
  - CPython (most used) interprets the Python bytecode

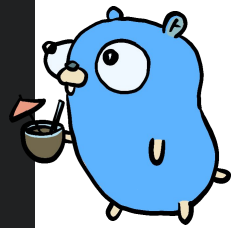
Dynamically typed and statically typed

```
x = 5
y = 5.5

print(x + y)
```

```
fn main() {
    let x: i32 = 5;
    let y: f32 = 5.5;

    println!(x as f32 + y);
}
```



```
func main() {
    var x int = 5
    var y float64 = 5.5

    fmt.Println(float64(x) + y)
}
```



Python bytecode

```
$ python3.12
Python 3.12.0 (main, Oct 21 2023, 17:42:12) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> def sayHello():
...     print("Hello people!");
...
>>> import dis
>>> dis.dis(sayHello)
  1           0 RESUME                   0

  2           2 LOAD_GLOBAL              1 (NULL + print)
             12 LOAD_CONST               1 ('Hello people!')
             14 CALL                     1
             22 POP_TOP
             24 RETURN_CONST             0 (None)
>>>
```

Python bytecode

```
% python3.12 -m py_compile countdown.py

% ls -l
% ls -l
total 7768
drwxr-xr-x  3 claustopke  staff       96 Nov 30 14:54 __pycache__
-rw-rw-r--  1 claustopke  staff      229 Nov 30 13:09 countdown.py

% ls -l __pycache__
total 8
-rw-r--r--  1 claustopke  staff  598 Nov 30 14:54 countdown.cpython-312.pyc
```


docs.python.org/3/library/dis.html

Just in Time Compiling

Standard Python distribution comes with CPython, which is written in C and Python and includes an interpreter and a Python bytecode compiler. So it does not convert directly to the machine instructions for specific CPU. CPython interprets line by line Python bytecode.

Other implementations like Pypy, IronPython and Numba does include Just in Time Compiler (JIT), which compiles during runtime the Python bytecode to the machine code (similar to JAVA). The consequence remove lots of features, and does support only a set of packages and capabilities of standard Python code.

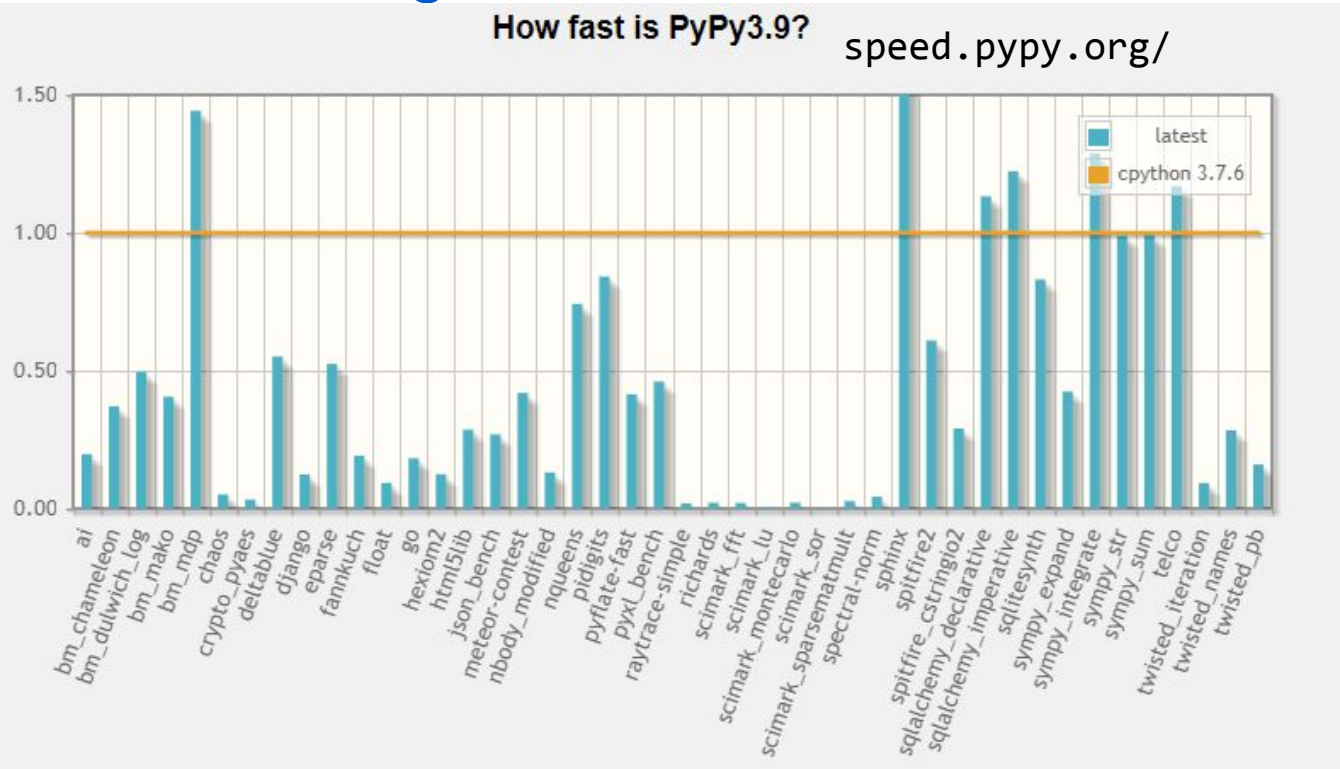
Why CPython does not have JIT ?

A large reason CPython doesn't have a JIT is the extra complexity it introduces to the core implementation and issues regarding C extensions.

Due mainly because of some Python capabilities like dynamic typing, polymorphism, and various introspective features. Therefore, some code will run as fast as with or without JIT.

www.theregister.com/2021/05/19/faster_python_mark_shannon_author/

PyPy versus CPython



Which cases PyPy excel ?

PyPy tends to be fast in numerically-intensive codes with hot loops dealing with (small) integers/float number as it can directly use native types instead of variable dynamic types.

PyPy uses a Garbage Collector (GC) as opposed to CPython which uses Automatic Reference Counting (ARC). GCs can be faster to allocate/free many objects, but they needs to track the object alive to know which one are dead and then free them. This means codes dealing with a huge amount of references and regular object allocations can actually be slower.

doc.pypy.org/en/latest/gc_info.html

www.pypy.org/performance.html

Which cases PyPy excel ?

PyPy does not well with large dynamic code. It uses a tracing just-in-time (JIT) compiler that can track which part of the code are more likely to be executed and compile this path dynamically at runtime if it is executed often. When there are many path executed changing dynamically, the overhead of the JIT can be significant, and in the worst case, PyPy can choose not to compile any path.

Pypy would not perform better on code that used libraries which depends heavily on C extensions, for example using numpy which is not fully compatible with JIT.

doc.pypy.org/en/latest/gc_info.html

www.pypy.org/performance.html

Parallelism and Concurrency

Oracle Multithreaded Programming Guide (1994):

Defining Concurrency and Parallelism

"*Concurrency exists when at least two threads are in progress at the same time.*

Parallelism arises when at least two threads are executing simultaneously.

In a multithreaded process on a single processor, the processor can switch execution resources between threads, resulting in concurrent execution.

In the same multithreaded process on a shared-memory multiprocessor, each thread in the process can run on a separate processor at the same time, resulting in parallel execution. "

docs.oracle.com/cd/E19457-01/801-6659/801-6659.pdf

Threads, Routines and Processes

- **Processes** are independent, isolated units of execution with their own memory space.
- **Threads** are lightweight execution units within a process, sharing memory and resources.
- **Routines** are similar to threads but run within a single execution context, often enabling cooperative multitasking.

CPU-bound and I/O-bound tasks

CPU-bound tasks benefit from parallel execution across multiple cores.

I/O-bound tasks benefit from concurrency to minimize idle time during I/O operations, including Network access.

CPU-bound tasks benefit from parallel execution to fully utilize CPU resources, while I/O-bound tasks benefit from asynchronous or concurrent execution to minimize waiting time for I/O operations.

Network automation tasks might be only I/O-bound, depending on the computation required. Complementary systems, like APIs and interfaces might be classified on the same I/O-bound.

Python Threading, Multiprocessing and Coroutines

Python 2.4.1 (2005) introduced **`threading`** standard library

docs.python.org/3/whatsnew/2.4.html

Python 2.6 (2008) introduced **`multiprocessing`** standard library

docs.python.org/3/whatsnew/2.6.html

Python 3.14 (2014) introduced **`asyncio`** standard library

docs.python.org/3/whatsnew/3.4.html

Python Global Interpreter Lock (GIL)

The Global Interpreter Lock (GIL) in Python ensures thread safety by restricting Python bytecode execution to one thread at a time, preventing memory access conflicts, chosen for interpreter simplicity. Example of conflict is reference counting for memory management, which involves assigning a reference count variable to objects created in Python.

The GIL in Python allows only one thread to run Python bytecode at a time, ensuring thread safety and simplifying memory management to prevent conflicts when multiple threads accessing objects.

Python Global Interpreter Lock (GIL)

```
NUMBER = 1_000_000_000  # 1 billion

def flipsum(args):
    start, end = args
    flipsum = 0
    for n in range(end, start - 1, -1):
        if n % 2 == 0:
            flipsum += n
        else:
            flipsum -= n
    return flipsum

if __name__ == "__main__":

    print("Flipsum {}".format(NUMBER))
    s_time = time.time()
    result = flipsum((1, NUMBER))
    e_time = time.time()
    print("Result =", result)
    print("Took {:f}s".format(e_time - s_time))
```

```
time -f "%MKB %P %e" python3.12 ./flipsum.py
Flipsum 1000000000
Result = 500000000
Took 79.184543s
10144KB 99% 79.19
```

Python Global Interpreter Lock (GIL)

```
from threading import Thread

def parallel_flip_sum(num_threads):
    chunk_size = NUMBER // num_threads
    ranges = [(i * chunk_size + 1, (i + 1) * chunk_size) for i in range(num_threads)]
    results = []
    threads = []

    for range_tuple in ranges:
        thread = Thread(target=flip_sum_chunk, args=(range_tuple, results))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

    return sum(results)

if __name__ == "__main__":
    num_threads = 4

    print("Flipsum {} with {} threads ".format(NUMBER, num_threads))
    start_time = time.time()
    result = parallel_flip_sum(num_threads)
    end_time = time.time()
```

```
time -f "%MKB %P %e" python3.12 ./flipsum-thre
Flipsum 1000000000 with 4 threads
Result = 500000000
Took 82.527606s
11308KB 100% 82.54
```

Python Global Interpreter Lock (GIL)

```
from multiprocessing import Pool

def parallel_flip_sum(num_processes):
    chunk_size = NUMBER // num_processes
    ranges = [(i * chunk_size + 1, (i + 1) * chunk_size) for i in
range(num_processes)]

    with Pool(num_processes) as pool:
        results = pool.map(flip_sum_chunk, ranges)

    return sum(results)

if __name__ == "__main__":
    num_procs = 4

    print("Flipsum {} with {} processes ".format(NUMBER, num_procs))
    start_time = time.time()
    result = parallel_flip_sum(num_procs)
    end_time = time.time()


    print("Result =", result)
    print("Took {:f}s".format(end_time - start_time))
```

```
time -f "%MKB %P %e" python3.12 ./flipsum-mult
Flipsum 1000000000 with 4 processes
Result = 500000000
Took 20.332754s
17020KB 392% 20.36
```

JIT and GIL influence on PyPy and CPython

```
python3.12 ./flipsum.py
Flipsum 1000000000
Took 79.184543s (10144KB 99% 79.19)
```

```
pypy3 flipsum.py
Flipsum 1000000000
Took 2.790204s (61520KB 99% 2.88)
```

```
python3.12 flipsum-threads.py
Flipsum 1000000000 with 4 threads
Took 82.527606s (11308KB 101% 82.54)
```

```
pypy3 flipsum-threads.py
Flipsum 1000000000 with 4 threads
Took 3.029221s (63248KB 104% 3.14)
```

```
python3.12 flipsum-multiprc.py
Flipsum 1000000000 with 4 processes
Took 20.332754s (17020KB 392% 20.36)
```

```
$ pypy3 flipsum-multiproc.py
Flipsum 1000000000 with 4 processes
Took 0.865098s (73000KB 310% 1.01s)
```


```
# pypy
Python 3.10.13 (fc59e61cfbff, Jan 14 2024, 13:01:46)
[PyPy 7.3.15 with GCC 10.2.1 20210130 (Red Hat 10.2.1-11)] on linux
```

The Flipsum in Go

```
const NUMBER = 1_000_000_000

func flipSum(start, end int) int {
	var flipSum int
	for n := end; n >= start; n-- {
		if n%2 == 0 {
			flipSum += n
		} else {
			flipSum -= n
		}
	}
	return flipSum
}

func main() {
	fmt.Printf("Flipsum %d\n", NUMBER)
	startTime := time.Now()
	result := flipSum(1, NUMBER)
	endTime := time.Now()

	fmt.Printf("Result = %d\n", result)
	fmt.Printf("Took %fs\n", endTime.Sub(startTime).Seconds())
}
```

```
$ time -f "%MKB %P %e" ./flipsum
Flipsum 1000000000
Result = 500000000
Took 1.232550s
1820KB 100% 1.23
```

The Flipsum in Go with goroutines

```
func parallelFlipSum(numGoroutines int) int {
        result := make(chan int, numGoroutines)
        wg := sync.WaitGroup{}

        chunkSize := NUMBER / numGoroutines
        for i := 0; i < numGoroutines; i++ {
                start := 1 + i*chunkSize
                end := start + chunkSize - 1
                wg.Add(1)
                go flipSumRange(start, end, &wg, result)
        }

        go func() {
                wg.Wait()
                close(result)
        }()

        totalSum := 0
        for chunkSum := range result {
                totalSum += chunkSum
        }

        return totalSum
}
```

```
$ time -f "%MKB %P %e" ./flipsum-goroutines
Flipsum 1000000000 with 4 goroutines
Result = 500000000
Took 0.455321s
1884KB 395% 0.45
```

The Flipsum in Rust

```
const NUMBER = 1_000_000_000

func flipSum(start, end int) int {
	var flipSum int
	for n := end; n >= start; n-- {
		if n%2 == 0 {
			flipSum += n
		} else {
			flipSum -= n
		}
	}
	return flipSum
}

func main() {
	fmt.Printf("Flipsum %d\n", NUMBER)
	startTime := time.Now()
	result := flipSum(1, NUMBER)
	endTime := time.Now()

	fmt.Printf("Result = %d\n", result)
	fmt.Printf("Took %fs\n", endTime.Sub(startTime).Seconds())
}
```

```
$ rustc flipsum.rs
$ time -f "%MKB %P %e" ./flipsum
Flipsum 1000000000
Result = 500000000
Took 11.966334848s
1952KB 99% 11.96
```

```
$ rustc -C opt-level=3 flipsum.rs
$ time -f "%MKB %P %e" ./flipsum
Flipsum 1000000000
Result = 500000000
Took 2.478427698s
2000KB 100% 2.47
```

The Flipsum in Rust with threads

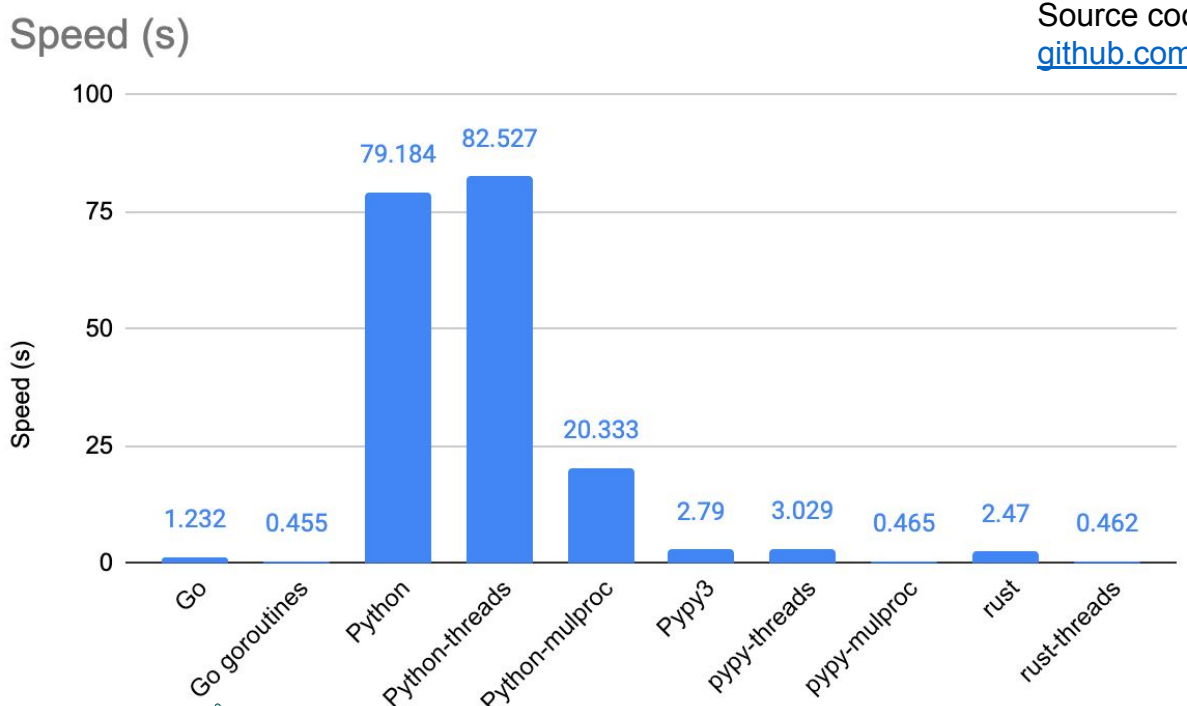
```
fn parallel_flip_sum(num_threads: usize) -> i32 {
    let chunk_size = NUMBER / num_threads as i32;
    let (result_sender, result_receiver) = std::sync::mpsc::channel()

    let mut handles = vec![];

    for i in 0..num_threads {
        let start = 1 + (i as i32) * chunk_size;
        let end = start + chunk_size - 1;
        let sender_clone = result_sender.clone();
        let handle = thread::spawn(
              move || flip_sum_range(start, end, sender_clone));
        handles.push(handle);
    }

    for handle in handles {
        handle.join().unwrap();
    }

    drop(result_sender);
    let mut total_sum = 0;
    for chunk_sum in result_receiver {
        total_sum += chunk_sum;
    }

    total_sum
}
```

```
$ rustc flipsum-threads.rs
$ time -f "%MKB %P %e" ./flipsum-threads
Flipsum 1000000000 with 4 threads
Result = 500000000
Took 3.406618203s
2420KB 385% 3.40
```

```
$ rustc -C opt-level=3 flipsum-threads.rs
$ time -f "%MKB %P %e" ./flipsum-threads
Flipsum 1000000000 with 4 threads
Result = 500000000
Took 0.462803148s
2264KB 398% 0.46
```

Flipsum speed usage side-by-side

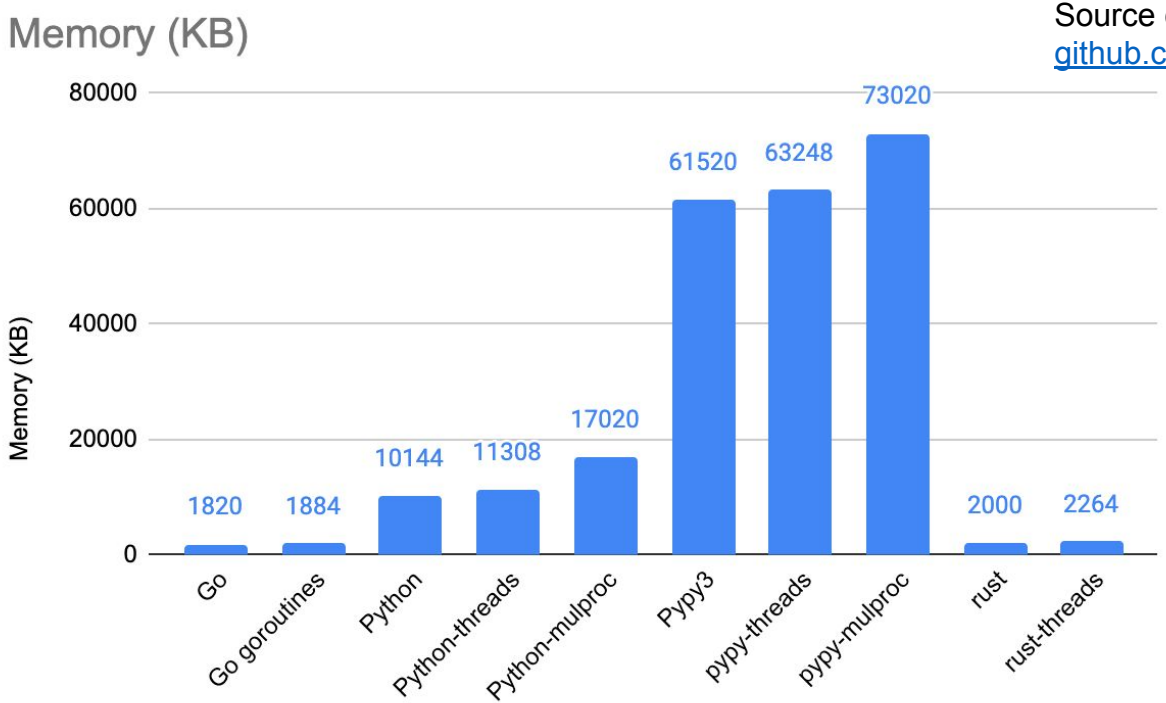
Source codes at:
github.com/brnuts/nanog90



4 threads
4 goroutines
4 processes

Flipsum memory usage side-by-side

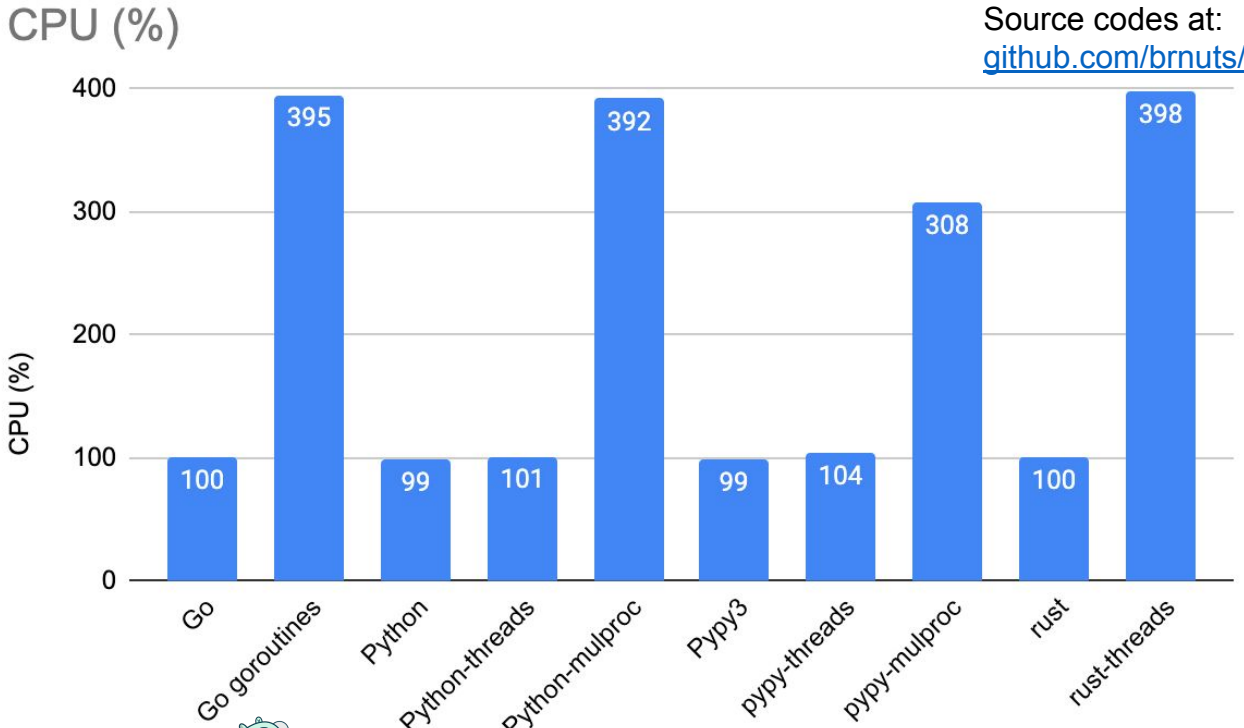
Source codes at:
github.com/brnuts/nanog90



4 threads
4 goroutines
4 processes

Flipsum CPU usage side-by-side

Source codes at:
github.com/brnuts/nanog90



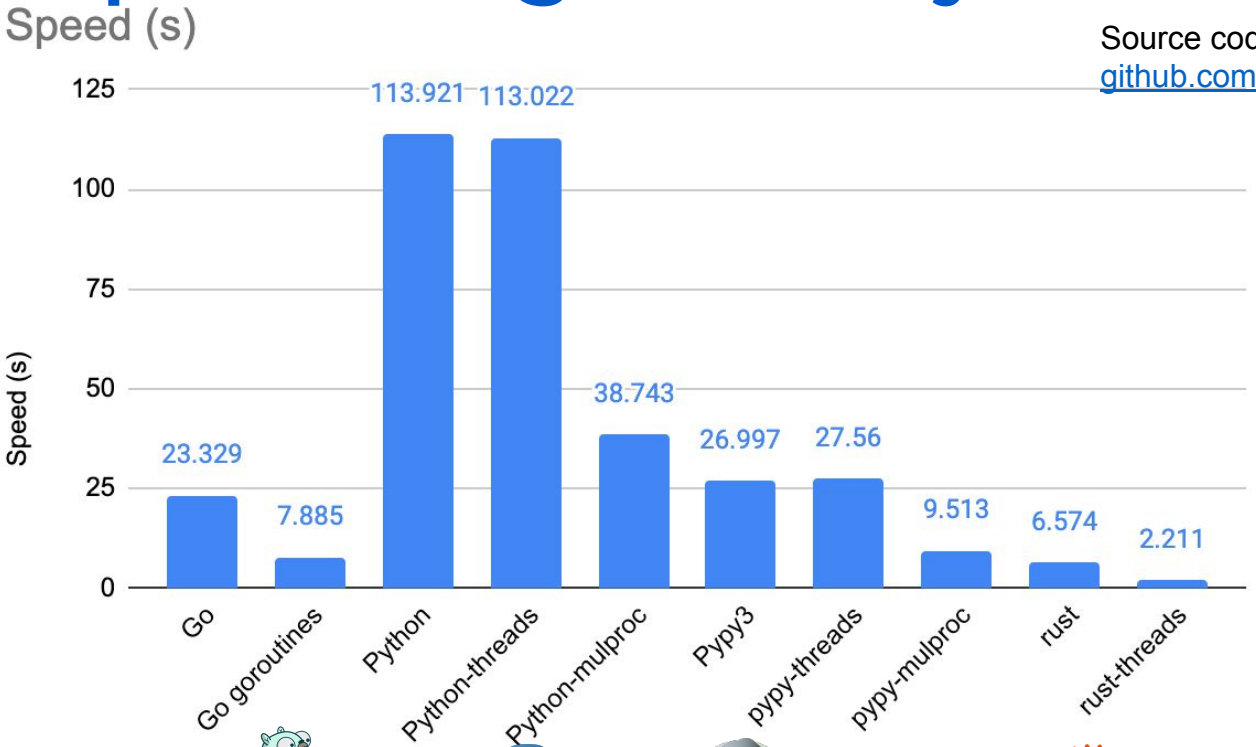
4 threads
4 goroutines
4 processes

Finding prime numbers

- Loop from 1 to 10 Million
  - Check if is prime
  - If is, add the prime number to a list
- Return the list
- Use 4 threads, goroutines or processes

Source codes at: github.com/brnuts/nanog90

Prime speed usage side-by-side

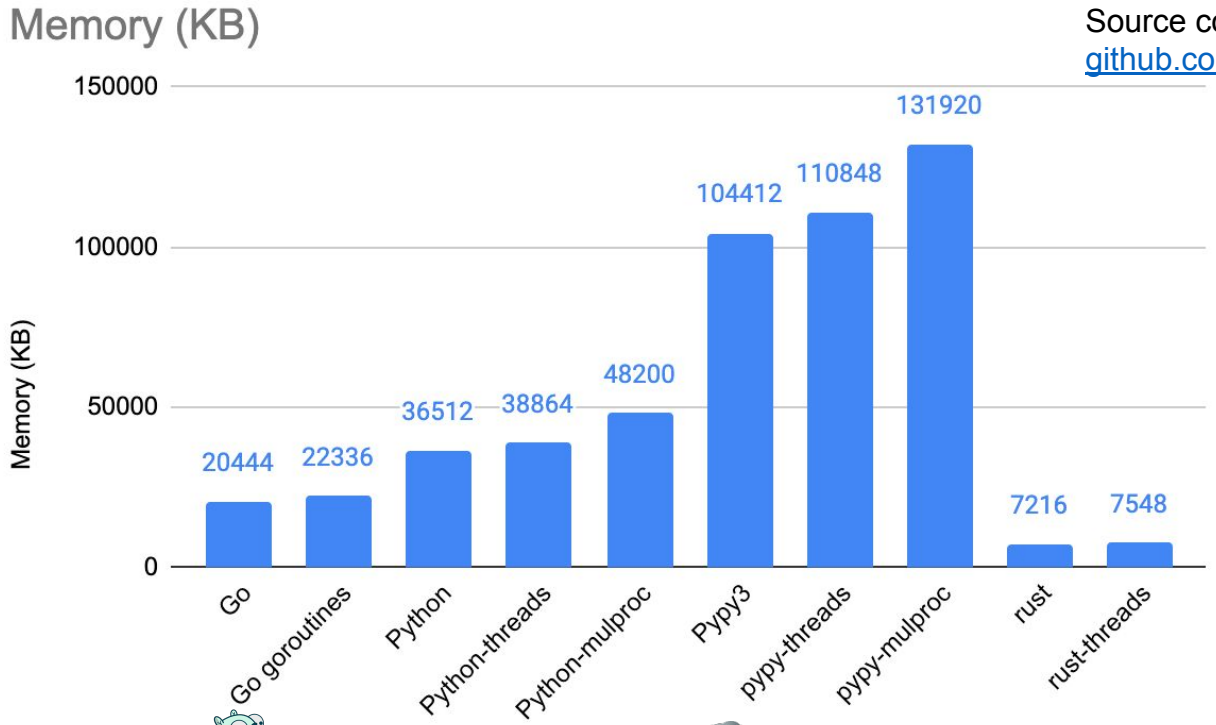


Source codes at:
github.com/brnuts/nanog90

4 threads
4 goroutines
4 processes

Prime memory usage side-by-side

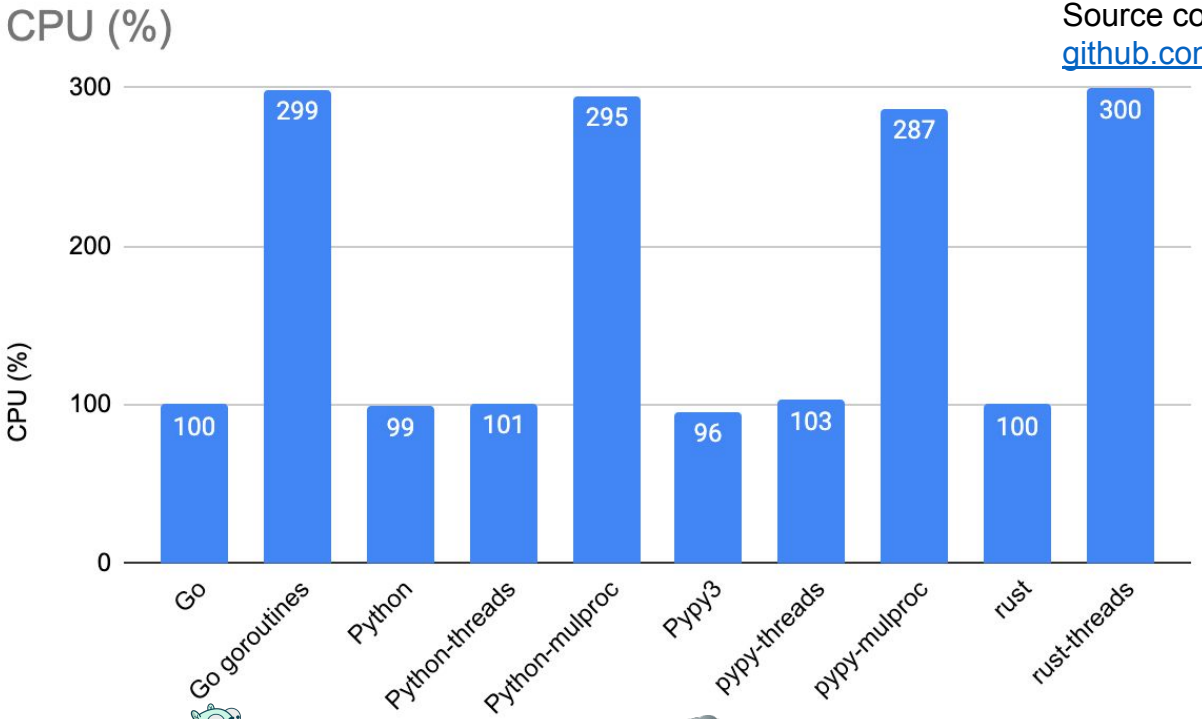
Source codes at:
github.com/brnuts/nanog90



4 threads
4 goroutines
4 processes

Prime CPU usage side-by-side

Source codes at:
github.com/brnuts/nanog90



4 threads
4 goroutines
4 processes

Speed, CPU and Memory usage

- **Countdown**
  - Empty loops and Compilers optimization
- **Array** (memory grows)
- **Sum of Squares**
  - Integer size limited in Go and Rust (int 64)
  - Python is unbounded limited to system 😄

Source codes at: github.com/brnuts/nanog90

Community benchmarking

"Which programming language is fastest?"

Top 5+ program performance comparisons –

C# vs Java Go versus Java

Ruby vs Python Rust versus C++

Rust vs Go

benchmarksgame-team.pages.debian.net/benchmarksgame/index.html

Community benchmarking

×	source	secs	mem	cpu secs	cpu load
1.0	**Rust** #4	**1.01**	32,380	3.99	100% 99% 99% 100%
1.1	Rust #6	1.06	34,120	4.18	98% 98% 98% 100%
1.3	Julia #7	1.33	232,380	4.83	88% 88% 98% 88%
1.4	Classic **Fortran** #8	**1.41**	75,272	5.58	100% 99% 98% 99%
1.6	**C** gcc #8	**1.63**	33,256	6.29	100% 95% 95% 95%
2.3	**C++** g++ #0	**2.34**	34,204	9.28	100% 99% 99% 99%
3.7	**Go** #4	**3.73**	36,116	14.87	99% 99% 99% 99%
3.7	Go #3	3.73	36,140	14.86	99% 99% 99% 99%

pidigits

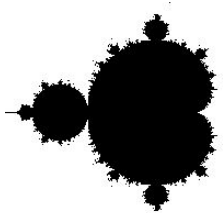
The work is to use arbitrary precision arithmetic and the same step-by-step single-threaded algorithm to generate digits of Pi.

benchmarksgame-team.pages.debian.net/benchmarksgame/performance/mandelbrot.html

Community benchmarking

×	source	secs	mem	cpu secs	cpu load
1.0	**Rust** #4	**1.01**	32,380	3.99	100% 99% 99% 100%
1.1	Rust #6	1.06	34,120	4.18	98% 98% 98% 100%
1.3	Julia #7	1.33	232,380	4.83	88% 88% 98% 88%
1.4	Classic **Fortran** #8	**1.41**	75,272	5.58	100% 99% 98% 99%
1.6	**C** gcc #8	**1.63**	33,256	6.29	100% 95% 95% 95%
2.3	**C++** g++ #0	**2.34**	34,204	9.28	100% 99% 99% 99%
3.7	**Go** #4	**3.73**	36,116	14.87	99% 99% 99% 99%
3.7	Go #3	3.73	36,140	14.86	99% 99% 99% 99%

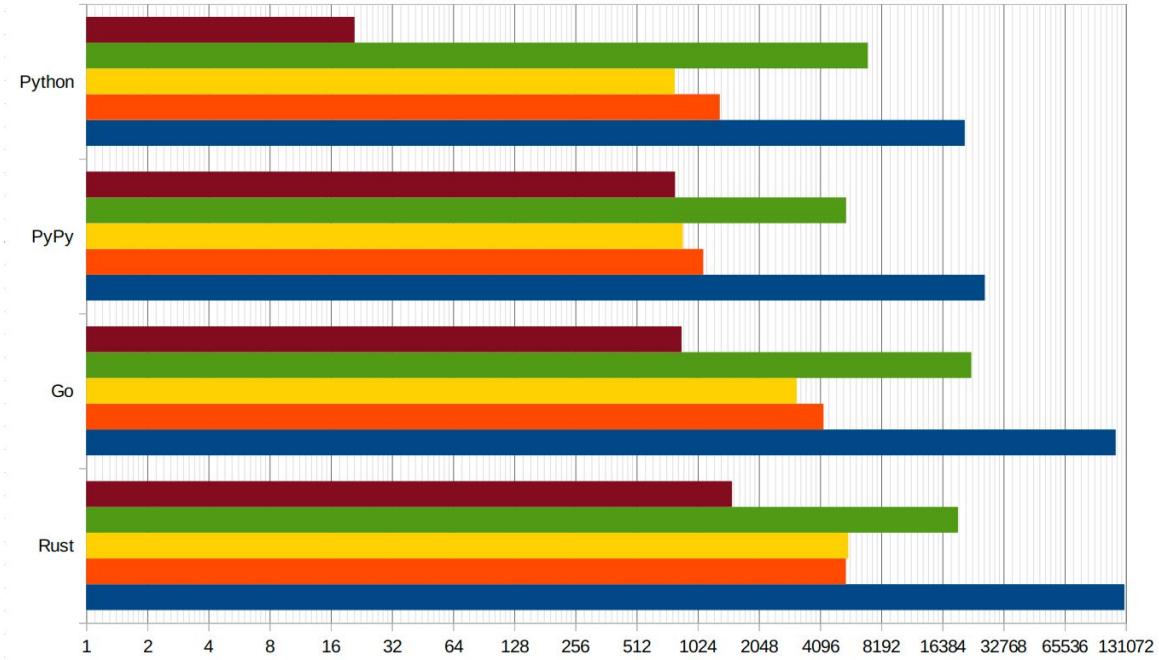
`mandelbrot`



plot the Mandelbrot set [-1.5-i,0.5+i] on an N-by-N bitmap.

benchmarksgame-team.pages.debian.net/benchmarksgame/performance/mandelbrot.html

Community benchmarking



	transfer speed
Rust	2,491.53 MiB/s
Go	2,897.66 MiB/s
PyPy	701.15 MiB/s
Python	897.27 MiB/s

David Martínez

deavid.wordpress.com/2019/10/12/benchmarking-python-vs-pypy-vs-go-vs-rust/

The versus table score

Memory footprint:

- Python and CPython are really bad.

Speed and parallelism:

- Python is bad. PyPy is good (but limited version)
- Rust is great, 10 (focus on security).
- Go is great, 10 (focus on performance and simplicity).

The versus table score

	Python	Go	Rust
Language release and core development	10	10	10
Implementations	9	8	7
Memory, speed and parallelism	5	10	10
Deployment and dependencies			
Program Logging, Error handling and Exceptions			
IP and network native libraries			
Network access methods and tools			
Network Automation community tools			

www.incredibuild.com/blog/lets-joke-our-top-10-compiling-memes

www.pinterest.com/pin/why-python-is-popular-despite-being-super-slow--615867317773696019/

Deployment and dependencies

Python importing modules

When importing a module without specifying its specific version, Python typically imports the first encountered module that matches the given name following a file path criteria (built-in modules, sys.path, site-packages)

Therefore, effectively managing dependencies and understanding the import search sequence becomes crucial to ensure the correct module version is imported.

Python importing modules

In complex projects with many dependencies, managing versions and resolving conflicting dependencies can become challenging.

Different versions of dependencies might introduce breaking changes or compatibility issues with other libraries, leading to problems.

Python dependency files

command: `strace -e openat python3.8 <program.py>` # with paramiko & pysnmp)

```
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/x509/__pycache__/extensions.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/ipaddress.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/x509/__pycache__/general_name.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/__init__.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email", O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/utils.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/random.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/bisect.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/_parseaddr.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/calendar.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/locale.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/charset.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/base64mime.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/quoprimime.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/errors.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/email/__pycache__/encoders.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/quopri.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/x509/__pycache__/name.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/x509/__pycache__/oid.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/aead.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/ciphers.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/cmac.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/ec.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/utils.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/backends/openssl/__pycache__/rsa.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/bindings/openssl/__pycache__/__init__.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/bindings/openssl", O_RDONLY|O_NONBLOCK|O_CLOEXEC|O_DIRECTORY) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/bindings/openssl/__pycache__/binding.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/bindings/openssl/__pycache__/_conditional.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/opt/pyca/cryptography/openssl/openssl.cnf", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/cryptography/hazmat/primitives/serialization/__pycache__/pkcs12.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/paramiko/__pycache__/client.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/local/lib/python3.8/dist-packages/paramiko/__pycache__/agent.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/tempfile.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/shutil.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/bz2.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/__pycache__/_compression.cpython-38.pyc", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/python3.8/lib-dynload/_bz2.cpython-38-x86_64-linux-gnu.so", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libbz2.so.1.0", O_RDONLY|O_CLOEXEC) = 3
.
.
.
```

269 files !!

Python upgrade and deployment nightmare

What to pay attention:

- Third-Party libraries compatibility.
- System and OS version.
- RollBack Plan
- Follow OS standard installation when possible.

medium.com/@damngoodtech/the-great-python-package-management-war-49f25df33d26

packaging.python.org/en/latest/guides/tool-recommendations/

opensource.com/article/19/4/managing-python-packages

Python versions and OS

Python 3.3 (2012) created `venv` (docs.python.org/3/whatsnew/3.3.html)

Virtual environments allow developers to isolate project dependencies, ensuring that each project can have its own set of libraries and packages without interfering with system-wide or other project dependencies.

Using pip freeze a good practice.

`pipenv` was created in 2017

Good practice is to add `requirements.txt`. (not mandatory)

Python has Pypi

pypi.org/

- Pypi stands for Python Package Index
- Pypi is the package repository for Python community
- Owned by Python Packaging Authority (PyPA)
  - Which maintains **`pip`**, setuptools, virtualenv, and wheel.
  - packaging.python.org/en/latest/key_projects/
- Pypi package only depends on Pypi packages.
- Has security audit (blog.pypi.org/posts/2023-11-14-1-pypi-completes-first-security-audit/)

Go dependencies are simple

```
$ cat go.mod
go 1.20

require (
	github.com/urfave/cli/v3 v3.0.0-alpha2
	golang.org/x/crypto v0.7.0
	gopkg.in/yaml.v3 v3.0.1
	github.com/cpuguy83/go-md2man/v2 v2.0.2 // indirect
	github.com/mattn/go-colorable v0.1.12 // indirect
	github.com/mattn/go-isatty v0.0.14 // indirect
	github.com/rs/zerolog v1.29.0 // indirect
	github.com/russross/blackfriday/v2 v2.1.0 // indirect
	github.com/xrash/smetrics v0.0.0-20201216005158-039620a65673 // indirect
	golang.org/x/sys v0.6.0 // indirect
)

$ # strace -e openat ./vlab -c -f readconfig.go
openat(AT_FDCWD, "/sys/kernel/mm/transparent_hugepage/hpage_pmd_size", O_RDONLY) = 3
openat(AT_FDCWD, "/etc/localtime", O_RDONLY) = 3
```

Go dependencies are simple

```
$ cat go.sum
github.com/urfave/cli/v3 v3.0.0-alpha2 h1:JKkuTewMlS2leTQeAcsPGL7WmBVa2uoBLy89As4Jauc=
github.com/urfave/cli/v3 v3.0.0-alpha2/go.mod h1:gHI/xEYplFhOa3Y90xJleh3kqqsSanBj/19hVFxiVZ4=
github.com/xrash/smetrics v0.0.0-20201216005158-039620a65673 h1:bAn7/zixMGCfxrRTfdpNzjtPYqr8smhKouy9mxVdGPU=
github.com/xrash/smetrics v0.0.0-20201216005158-039620a65673/go.mod h1:N3UwUGtsrSj3ccvlPHLoLsHnpR27oXr4ZE984MbSER8=
golang.org/x/crypto v0.7.0 h1:AvwMYaRytfdeVt3u6mLaxYtErKYjxA2OXjJ1HHq6t3A=
golang.org/x/crypto v0.7.0/go.mod h1:pYwdfH91IfpZVANVyUOhSIPZaFoJGxTFbZhFTx+dXZU=
golang.org/x/sys v0.6.0 h1:MVltZSvRTcU2ljQOhs94SXPftV6DCNnZViHeQps87pQ=
golang.org/x/sys v0.6.0/go.mod h1:oPkhp1MJrh7nUepCBck5+mAzfO9JrbApNNgaTdGDITg=
golang.org/x/term v0.6.0 h1:clScbb1cHjoCkyRbWwBEUZ5H/tIFu5TAXIqaZD0Gcjw=
.
.
.
gopkg.in/check.v1 v0.0.0-20161208181325-20d25e280405 h1:yhCVgyC4o1eVCa2tZl7eS0r+SDo693bJlVdllGtEeKM=
gopkg.in/check.v1 v0.0.0-20161208181325-20d25e280405/go.mod h1:Co6ibVJAznAaIkqp8huTwlJQCZ016jof/cbN4VW5Yz0=
gopkg.in/yaml.v3 v3.0.1 h1:fxVm/GzAzEWqLHuvctI91KS9hhNmmWOoWu0XTYJS7CA=
gopkg.in/yaml.v3 v3.0.1/go.mod h1:K4uyk7z7BCEPqu6E+C64Yfv1cQ7kz7rIZviUmN+EgEM=
.
.
.
```

Rust dependencies are simple

```
$ cat Cargo.toml
[package]
name = "capture"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
pcap = "1.2.0"
```

Rust dependencies are simple

```
$ cat Cargo.lock
# This file is automatically @generated by Cargo.
# It is not intended for manual editing.
version = 3

[[package]]
name = "capture"
version = "0.1.0"
dependencies = [
 "pcap",
]

[[package]]
name = "pcap"
version = "1.2.0"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "77452fdf9d211d9ca35d092aeefe4d4b3f0c4eb529ffb87a8a3b8fe2bb7c37c3"
dependencies = [
 "bitflags",
 "errno",
 "libc",
.
.
.
```

Rust has Cargo and creates.io

- Cargo is a more feature-rich build system with extensive dependency management, testing, and documentation support.
- Dependencies are defined like in Go, but with the advantages of having Crates.io (Rust's package registry).
- Using Creates.io ensure your project will only depends on Creates.io.
- rustsec.org/

Github as well

blog.logrocket.com/comparing-rust-supply-chain-safety-tools/

The versus table score

	Python	Go	Rust
Language release and core development	10	10	10
Implementations	9	8	7
Memory, speed and parallelism	5	10	10
Deployment and dependencies	7	9	10
Program Logging, Error handling and Exceptions			
IP and network native libraries			
Network access methods and tools			
Network Automation community tools			

Program Logging, Error handling and Exceptions

Python logging

Python 2.3 (2008) introduced **`logging`** standard library

docs.python.org/3/whatsnew/2.3.html

The logging module offers functionalities to create loggers, set logging levels, route logs to different destinations (handlers) like files or the console, format log messages, and filter log records based on specific criteria.

Hierarchical logging in Python is achieved through the use of logger names separated by periods (dot notation).

Go logging

The initial standard logging package in Go is called `log`, but is very limited in compare to Python, for instance does not have log levels.

Version 1.21 (2023) introduced new package called `log/slog`, which includes some of the missing capabilities, like levels and handlers. (tip.golang.org/doc/go1.21)

More on Go slog:

go.dev/blog/slog

betterstack.com/community/guides/logging/logging-in-go/

Rust logging

Rust does not have built in logging capabilities.

Logging is provided by crates.

Still part of the Rust language tree, crates.io/crates/log (github.com/rust-lang/log), but with different group of developers.

Python error handling

Python uses Exception based and tracebacks.

Python's exception-based approach can make code more readable and avoid repetition, whereas Go's explicit error returns can lead to more explicit error handling and more tedious repetitions.

Built-in exceptions in Python are hierarchical, and easy to work with:

docs.python.org/3/library/exceptions.html

Go error handling

Go uses explicit return values checks, like:

```
func divide(a, b int) (int, error) {
    if b == 0 {
        return 0, errors.New("division by zero")
    }
    return a / b, nil
}
```

Which is up to the developer to handle correctly.

Go error handling is up to developer

Most programmers would not implement correctly error handling, due to cascade error check and message syntax.

So, the syntax still allows developers to make errors, with wrong or imprecise signatures of the error.

Checking every single method for errors and returning a proper message is extremely frustrating and most of the time not well maintained.

Therefore, is likely to introduce bugs. Specially if you have 1.000 places where you have to add error checks, the likelihood you forgot or write wrong message is high.

Go example of lack context error handling

```
        if err = c.realm.GetFrom(res); err != nil {
+               log.Println("GetFrom2 problem")
                return relayed, lifetime, nonce, err
        }
        c.realm = append([]byte(nil), c.realm...)
@@ -281,11 +288,13 @@ func (c *Client) sendAllocRequest(protocol proto.Protocol) (proto.RelayAddr
                stun.Fingerprint,)
        if err != nil {
+               log.Println("Build Authorize problem")
                return relayed, lifetime, nonce, err
        }

        trRes, err = c.PerformTransaction(msg, c.turnServerAddr, false)
        if err != nil {
+               log.Println("PerformTransation problem")
                return relayed, lifetime, nonce, err
        }
        res = trRes.Msg
```

Rust error handling

Rust error handling is similar to Go, however it has different methods to propagate the error.

Go uses a simple error interface and return values, while Rust employs a more comprehensive Result type with pattern matching and concise error propagation using the ? operator.

In addition, Rust's std::error::Error trait allows for richer custom error types.

Rust error handling

Rust's approach encourages more robust error handling by requiring explicit handling of error cases at compile-time, reducing the likelihood of error omission compared to Go's more manual error checks.

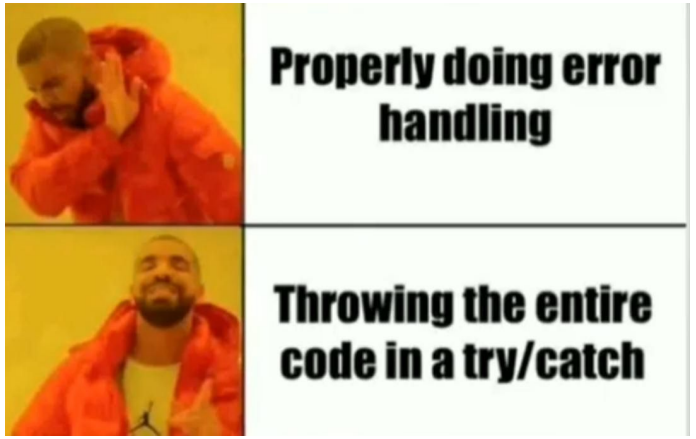
Go's simplicity may be more straightforward for smaller programs, while Rust's approach provides stronger safety guarantees for larger, more complex systems.

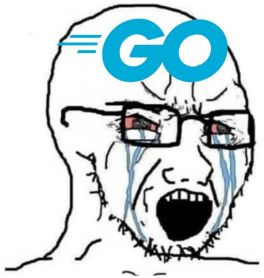
doc.rust-lang.org/std/ops/struct.Yeet.html

The versus table score

	Python	Go	Rust
Language release and core development	10	10	10
Implementations	9	8	7
Memory, speed and parallelism	5	10	10
Deployment and dependencies	7	9	10
Program Logging, Error handling and Exceptions	10	8	8
IP and network native libraries			
Network access methods and tools			
Network Automation community tools			

preslav.me/2023/04/14/golang-error-handling-is-a-form-of-storytelling/





Noo you can't just pretend you know what your `error` type is at compile time!

`Result<T, StaticallyKnownErrorType>`

www.reddit.com/r/ProgrammerHumor/comments/tfyan9/ive_been_studying_go_and_rust_lately/

IP and network native libraries

Python standard network libraries

Python 0.9.1 (1991) introduced **socket** standard library module.

github.com/smontanaro/python-0.9.1

www.python.org/download/releases/early/

Python 3.0 (2008) introduced **ssl**, **http**, **html** **and** **urllib** standard libraries.

docs.python.org/3/whatsnew/3.0.html

Python standard network libraries

Python 3.3 (2012) introduced **`ipaddress`** standard library

docs.python.org/3/whatsnew/3.3.html

It does include wide operations with IPv4 and IPv6, including network range, validation, multicast, loopback, CIDR, subnet, supernet, compare, collapse, among others.

docs.python.org/3/library/ipaddress.html

Example:

```
network_v4 = ipaddress.IPv4Network('192.168.0.0/24')
print("\nIPv4 Network Hosts:")
for host in network_v4.hosts():
    print(host)
```

Go standard network libraries

The majority of the network libraries were introduced in early Go, like version 1.0 and 1.2.

For IP manipulation is `net` package. Other important ones are `crypto/tls` , `http`, `net/http`, `net/rpc`, grpc, ssh, among others.

The IP manipulation capabilities are inferior to Python, for instance it is not possible to loop a subnet without a helper function.

Rust standard network libraries

Rust has better ip manipulation than Go.

Rust does not have HTTP standard libraries.

Rust does not have SSL/TLS standard libraries.

Rust does not have gRPC standard libraries.

Rust does not have SSH standard library.

The versus table score

	Python	Go	Rust
Language release and core development	10	10	10
Implementations	9	8	7
Memory, speed and parallelism	5	10	10
Deployment and dependencies	7	9	10
Program Logging, Error handling and Exceptions	10	8	8
IP and network native libraries	10	9	8
Network access methods and tools			
Network Automation community tools			

Network access methods and tools

Some methods to access network

Command Line Interface (Serial, TELNET, RSH, SSH)

SNMP (v2, v2c and v3) (bad implementation traps)

Private APIs

NETCONF, RESTCONF and YANG (HTTP/HTTPS)

gRPC (HTTPS)

gNMI (gRPC)

gNOI (gRPC)

* note

Pull

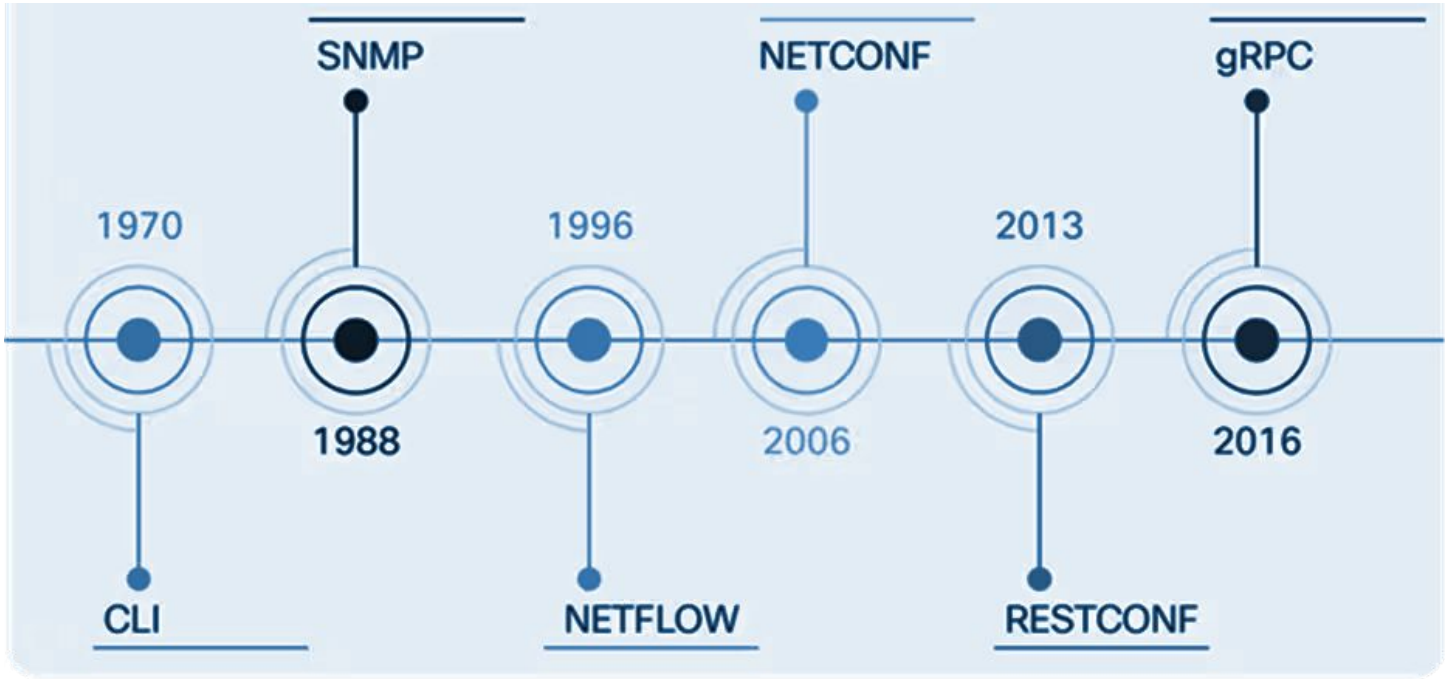
(like SNMP, private APIs, CLI scraping)

versus

Streaming telemetry

(like SNMP traps, netflow, sflow, syslog)

Telemetry timeline



CLI - Python

- Using local shell via `subprocess` (invoke shell `ssh` command).
- Using standard libraries with `socket`.
- Python bindings for libssh2 (C code), called `ssh2-python3`
- Using `pexpect` (not being updated).
- Using `paramiko`.
- Using `netmiko`.
- Using `scrapli`.

`asyncio` aware:

- Using `asyncssh`.
- Using `scrapli-asyncssh` plugin for `scrapli`. (Dec 2020)

CLI Python examples

subprocess

```
try:
    ssh_process = subprocess.Popen(...)
        ['/usr/bin/ssh', 'user@hostname'],
        .
        .
    )
    command_to_send = 'ls -l'
    ssh_process.stdin.write(command_to_send + '\n')
    output, error = ssh_process.communicate(timeout=5)
    print("Output:", output)
    print("Error:", error)

except subprocess.TimeoutExpired:
    print("Timeout: Process took too long.")

except Exception as e:
    print("Error:", e)
```

paramiko

```
try:
    client.connect(
        hostname=hostname,
        port=port,
        username=username,
        password=password,
    )
    stdin, stdout, stderr = client.exec_command('ls -l')
    output = stdout.read().decode('utf-8')
    print("Command output:", output)

except paramiko.AuthenticationException:
    print("Authentication failed")
except paramiko.SSHException as exc:
    print("Unable to connect:", exc)
```

CLI Python examples

netmiko

```
device = {
    'device_type': 'cisco_ios',
    'host': host,
    'username': username,
    'password': password,
}
try:
    net_connect = ConnectHandler(**device)
    net_connect.enable()
    output = net_connect.send_command('show version')
    print("Command output:", output)

except Exception as e:
    print("failed:", e)
```

scrapli

```
device = {
    'host': hostname,
    'Auth_username': username,
    'Auth_password': password,
}
try:
    conn = IOSXEDriver(**device)
    conn.open()
    response = conn.send_command('show version')
    print("Command output:", output)

except ScrapliException as err:
    print("An error occurred: {}", err)
```

CLI Python asyncio aware

asyncssh

```python
async def run_ssh_command():
    async with asyncssh.connect(
        'hostname',
        username='user',
        password='password') as conn:

            result = await conn.run('ls -l')
            print(result.stdout)

asyncio.run(run_ssh_command())
```

scrapli plugin

```python
async def run_cisco_command():
    iosxe = {
        "host": host,
        "auth_username": username,
        "auth_password": password,
        "platform": "cisco_iosxe",
    }
    async with AsyncScrapli(**iosxe, plugin="asyncssh") as conn:
        resp = await conn.send_command("show version")
        print(resp.result)

asyncio.run(run_cisco_command())
```

CLI - Go

- Using local shell via `os/exec` (invoke shell `ssh` command).
- Using standard libraries with `socket`.
- Using `golang.org/x/crypto/ssh`.
- Using `github.com/google/goexpect`. (with ssh wrapper)
- Using `github.com/gliderlabs/ssh`.
- Using `github.com/yahoo/vssh`.
- Using `github.com/scrapli/scrapligo`.

CLI Go examples

os/exec

```
func execCmd(host, user, cmd string) {

	sshCmd := fmt.Sprintf("ssh %s@%s '%s'",user, host,
command)

	cmd := exec.Command("bash", "-c", sshCmd)
	output, err := cmd.CombinedOutput()
	if err != nil {
		log.Fatalf("Error on %s: %s\n", host, err)
	}

	fmt.Printf("Output for %s:\n%s\n", host, output)
}
```

golang.org/x/crypto/ssh

```
func execCmd(host, user, passwd, cmd string) {
	config := &ssh.ClientConfig{
		User: user,
		Auth: []ssh.AuthMethod{
			ssh.Password(passwd),
		},
	}
	client, err := ssh.Dial("tcp", host+":22", config)
	if err != nil {
		log.Fatalf("Error on dial: %s", err)
	}
	defer client.Close()
	session, err := client.NewSession()
	if err != nil {
		log.Fatalf("Error creating session: %s", err)
	}
	defer session.Close()
	output, err := session.CombinedOutput(command)
	if err != nil {
		log.Fatalf("Error running command: %s", err)
	}
	fmt.Printf("Output:", string(output))
}
```

CLI Go examples

yahoo/vss

```
func main() {
	hosts := []string{"hostA", "hostB", "hostC", }
	username := "mynameisbob"
	passwd := "mypasswordissecret"

	var wg sync.WaitGroup

	for _, host := range hosts {
		wg.Add(1)
		go execComd(host, username, passwd, &wg)
	}

	wg.Wait()
}
```

```
func execCmd(host string, user string, pass string, wg *sync.WaitGroup) {
	defer wg.Done()

	config := &vssh.Config{
		User:            username,
		Password:        pass,
		Host:            host,
		ConnectionRetry: 3,
		Timeout:         10 * time.Second,
	}
	client, err := vssh.Dial(config)
	if err != nil {
		log.Printf("Dial error %s: %v\n", host, err)
		return
	}
	defer client.Close()
	output, err := client.Run("ls -l")
	if err != nil {
		log.Printf("Error on %s: %v\n", host, err)
		return
	}
	fmt.Printf("%s:\n", host, output)
}
```

CLI - Rust

- Using local shell via `std::process::Command` (invoke `ssh` cmd).
- Using standard libraries with `std::net::TcpStream`.
- Using crates `ssh2` (crates.io/crates/ssh2).
- With support of `rexpect` (crates.io/crates/rexpect)
- Using `scraplirs` (github.com/scrapli/scraplirs not crates*)
- Using crates `clap` (crates.io/crates/clap).
- Could not find similar to paramiko or netmiko.

SNMP

Python:

- Using `pysnmp`. (does have asyncio `pysnmp.hlapi.asyncio`)
- Using `pysmi`. (MIB parser)
- Net-snmp Python bindings (pypi.org/project/python3-netsnmp/).
- Using `puresnmp`. (no dependencies)
- Asyncio aware use `aiosnmp`.
- Also several MIB browsers.

SNMP

Go:

- Using `gosnmp` (github.com/gosnmp/gosnmp).
- Using `snmpgo` (github.com/k-sone/snmpgo).

Rust:

- Using crates `rasn-snmp` (crates.io/crates/rasn-snmp).

NETCONF, RESTCONF and YANG

Python:

- Using `ncclient` (supports subscription of YANG events, telemetry?).
- Using `aio-ncclient` (Juniper initiative).
- Using `libyang` (YANG parser, including yanglint).

NETCONF, RESTCONF and YANG

Go:

- Using `go-netconf` (Juniper initiative github.com/Juniper/go-netconf)
- Using `goyang` (openconfig github.com/openconfig/goyang)
- Using go mgmt library (github.com/damianoneill/net)

Rust:

- Crates netconf is 5 years old (crates.io/crates/netconf)
- Crates netconf-rs seems not used (crates.io/crates/netconf-rs)
- Do crates.io/search?q=netconf

gRPC

- Use HTTP2/TLS

- Required protobuf (serialize structured data), `.proto` file. (service definition)

- Generation of client and server code using protobuf compiler.

gRPC

Python:

- grpc.io/docs/languages/python/basics/
- grpc.github.io/grpc/python/

In Python is implemented using C++. github.com/grpc/grpc

- Asyncio grpc.github.io/grpc/python/grpc_asyncio.html
- To install: `pip install grpcio && pip install grpcio-tools`

gRPC

Go:

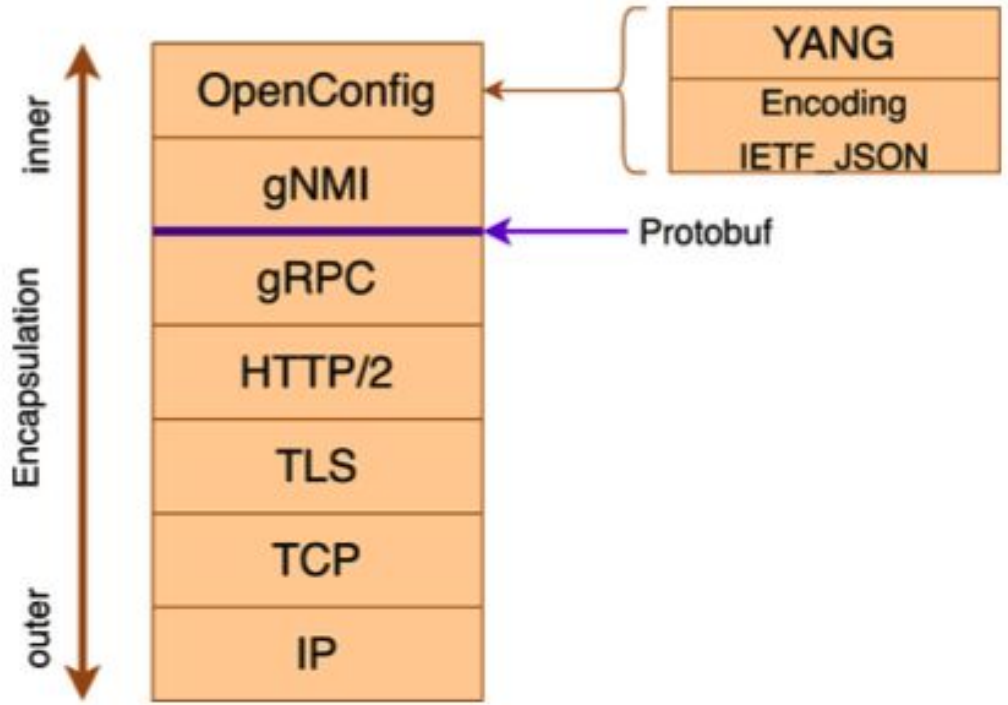
- grpc.io/docs/languages/go/basics/
- Native grpc implementation: github.com/grpc/grpc-go
- google.golang.org/protobuf/
- google.golang.org/grpc/

gRPC

Rust:

- Missing Rust at grpc.io/docs/languages/
- crates grpc crates.io/crates/grpc 3 years old
- crates tonic crates.io/crates/tonic
  - **tonic** relies only on other Rust libraries (hyper, tower, and prost) for HTTP/2 support, service middleware and Protocol Buffers.

gNMI - gRPC Network Management Interface



gNMI - gRPC Network Management Interface

NETCONF/RESTCONF uses HTTP/HTTPS, JSON and XML (slow)

Future of Telemetry subscription?

Uses Openconfig data models
www.openconfig.net/projects/models

BGP neighbour data model:
github.com/openconfig/public/blob/master/release/models/bgp/openconfig-bgp-neighbor.yang

Interface IP data model (RFC7277 -> RFC8343):
github.com/openconfig/public/blob/master/release/models/interfaces/openconfig-if-ip.yang

gNMI - RFC8343

YANG data node in /interfaces/interface	IF-MIB object
name	ifName
type	ifType
description	ifAlias
admin-status	ifAdminStatus
oper-status	ifOperStatus
last-change	ifLastChange
if-index	ifIndex
link-up-down-trap-enable	ifLinkUpDownTrapEnable
phys-address	ifPhysAddress
higher-layer-if and lower-layer-if	ifStackTable
speed	ifSpeed and ifHighSpeed
discontinuity-time	ifCounterDiscontinuityTime
in-octets	ifHCInOctets
in-unicast-pkts	ifHCInUcastPkts
in-broadcast-pkts	ifHCInBroadcastPkts
in-multicast-pkts	ifHCInMulticastPkts
in-discards	ifInDiscards
in-errors	ifInErrors
in-unknown-protos	ifInUnknownProtos
out-octets	ifHCOutOctets
out-unicast-pkts	ifHCOutUcastPkts
out-broadcast-pkts	ifHCOutBroadcastPkts
out-multicast-pkts	ifHCOutMulticastPkts
out-discards	ifOutDiscards
out-errors	ifOutErrors

gNMI - Python

General github.com/akarneliuk/pygnmi

Cisco: github.com/cisco-ie/cisco-gnmi-python (uses Go)

```
github.com/cisco-ie/cisco-gnmi-python/tree/master/github.com/openconfig
```

General github.com/google/gnxi (basic operations in Go)

```
github.com/google/gnxi/blob/master/gnmi_subscribe/gnmi_subscribe.go
```

gNMI - Go

CLI version: github.com/openconfig/gnmic/

Server: pkg.go.dev/github.com/google/gnxi/gnmi

Client: github.com/openconfig/gnmi

Arista github.com/aristanetworks/goarista

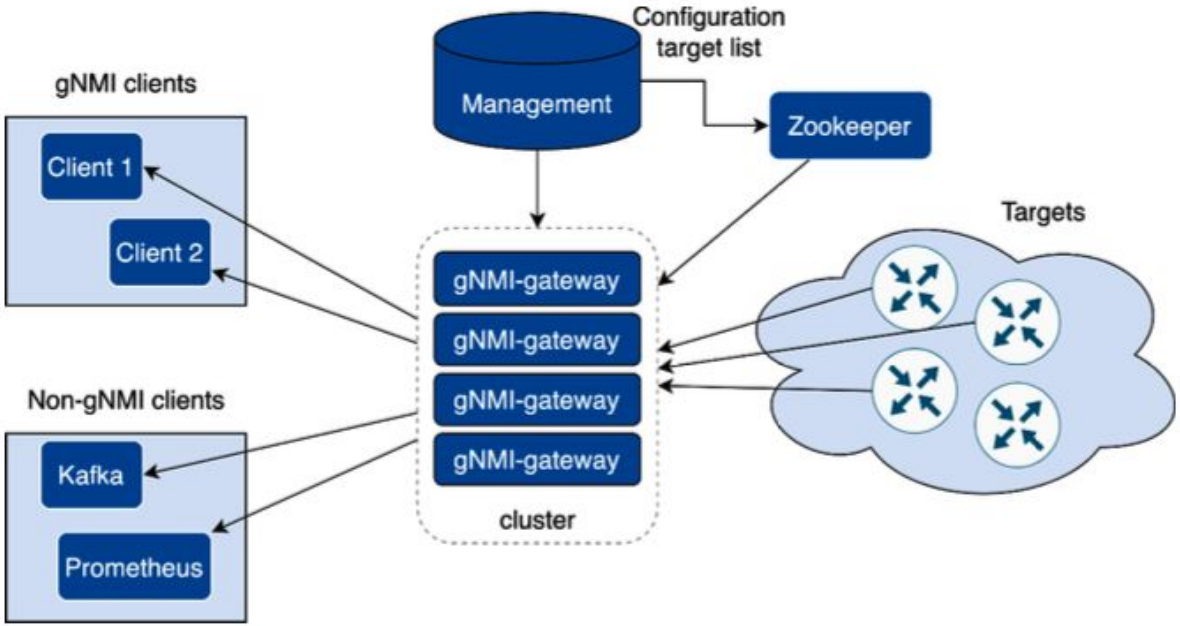
Juniper github.com/Juniper/openconfig-gnmi/

gNMI - Gateway github.com/openconfig/gnmi-gateway
(Netflix initiative)

gNMI - Go

```
subReq := &gnmi.SubscribeRequest{
	Request: &gnmi.SubscribeRequest_Subscribe{
		Subscribe: &gnmi.SubscriptionList{
			Subscription: []*gnmi.Subscription{
				{
					Path: &gnmi.Path{
						Origin: "openconfig",
						Elem: []*gnmi.PathElem{
							{Name: "interfaces"},
							{Name: "interface"},
							{Name: "state"},
							{Name: "counters"},
							{Name: "in-octets"},
						},
					},
					Mode: gnmi.SubscriptionMode_ON_CHANGE, // Or SAMPLE 30s default

				},
			},
		},
	},
}
```

gNMI Gateway - Go



netflixtechblog.com/simple-streaming-telemetry-27447416e68f

gNMI - Rust

- No yet library found, solution is to use tonic.

gNOI

While gNMI deals with the management of configuration and mainly **telemetry streaming**, gNOI idea is to encompass a broader range of network operations tasks, including device lifecycle management, software updates, and system diagnostics. Examples: `traceroute` , `ping` ,`pcap` or `clear bgp`.

gNOI

Python: Not natively, uses Go.
github.com/python-gnxi/python-gnoi-proto
github.com/google/gnxi/tree/master/gnoi

Go: Native implementation github.com/openconfig/gnoi

Pcap -> packet_capture/packet_capture_grpc.pb.go

Ping and traceroute -> system/system_grpc.pb.go

Rust: Not yet, has to use tonic.

Private APIs

Amazon AWS :

- Python (boto3): pypi.org/project/boto3/
- Go: github.com/aws/aws-sdk-go
- Rust: github.com/awslabs/aws-sdk-rust

Microsoft Azure:

- Python: pypi.org/project/azure-mgmt-compute/
- Go: github.com/Azure/azure-sdk-for-go/
- Rust: github.com/Azure/azure-sdk-for-rust

Lower level

Rust integrates with nftables: crates.io/crates/nftnl and nftables github.com/mullvad/nftnl-rs

Rust integrates with pcap crates.io/crates/pcap

Go integrates with nftables github.com/google/nftables

Go integrates with pcap github.com/google/gopacket

Python has, but limited performance, no asyncio.

Only Rust has

The Linux kernel 6.1, which has been in development for more than two months, now includes experimental support for the Rust programming language.

www.analyticsinsight.net/can-googles-obsession-with-rust-make-it-the-next-python/

www.analyticsinsight.net/updated-linux-kernel-6-1-makes-rust-the-greatest-programming-language/

Rust has been in Linux since Linus Torvalds gave the memory-safe language his blessing for the Linux 6.1 release. Now, though, Rust is taking the steps it needs to become -- along with C -- a member of the Linux language toolchain.

www.zdnet.com/article/rust-in-linux-where-we-are-and-where-were-going-next/

The versus table score

	Python	Go	Rust
Language release and core development	10	10	10
Implementations	9	8	7
Memory, speed and parallelism	5	10	10
Deployment and dependencies	7	9	10
Program Logging, Error handling and Exceptions	10	8	8
IP and network native libraries	10	9	8
Network access methods and tools	9	9	8
Network Automation community tools			

Network Automation community tools

Some Python tools

Paramiko (github.com/paramiko/paramiko)

Netmiko (github.com/ktbyers/netmiko)

PyGNMI (github.com/akarneliuk/pygnmi)

Scrapli (github.com/carlmontanari/scrapli)

NetTowel (github.com/InfrastructureAsCode-ch/nettowel)

Netconan (github.com/intentionet/netconan)

Some Python frameworks

Vrnetlab (github.com/vrnetlab/vrnetlab)

Netlab (github.com/ipspace/netlab)

Mininet (github.com/mininet/mininet)

StackStorm (github.com/StackStorm/st2)

Noir (github.com/nornir-automation/nornir)

Ansible (github.com/ansible/ansible)

Salt/Saltstack (github.com/saltstack/salt)

Some Go initiatives

Scrapligo (github.com/scrapli/scrapligo)

Terraform (github.com/hashicorp/terraform)

Prometheus (github.com/prometheus/prometheus)

Telegraf (github.com/influxdata/telegraf)

goBGP (github.com/osrg/gobgp)

Some Rust initiatives

RustyBGP (github.com/osrg/rustybgp) (better than FRR)

NetBricks (github.com/NetSys/NetBricks)

Not many higher levels tools.

The versus table score

	Python	Go	Rust
Language release and core development	10	10	10
Implementations	9	8	7
Memory, speed and parallelism	5	10	10
Deployment and dependencies	7	9	10
Program Logging, Error handling and Exceptions	10	8	8
IP and network native libraries	10	9	8
Network access methods and tools	9	9	8
Network Automation community tools	10	8	6

Not included

Not included

-Integrated Development Environments (IDE) (Intellij, MSVS...)

-Documentation generation.

-Code profiling (cProfile, `go pprof`, `go tool trace`, `cargo flamegraph`)

-Code testing like unit-testing and Integration-testing (pytests/unitest, `go test`, `cargo test`)

-Code formatting (python Black, `go fmt`, `cargo fmt`)

-Code linters (Pylint, golint, Rust clippy)

Conclusions

Final score

	Python	Go	Rust
Language release and core development	10	10	10
Implementations	9	8	7
Memory, speed and parallelism	5	10	10
Deployment and dependencies	7	9	10
Program Logging, Error handling and Exceptions	10	8	8
IP and network native libraries	10	9	8
Network access methods and tools	9	9	8
Network Automation community tools	10	8	6
	70	**71**	**67**

Conclusions

Which has potential for the future for Network Automation (NA)?

Rust and then Go

Which would be easier to maintain and collaborate for NA?

Rust, Go and then Python

Which would be easier to learn today for NA?

Python, Go and then Rust

Conclusions

I want to have an efficient, fast and easy deployment solution...

Rust and then Go (Python excluded)

I want easy integration with access legacy systems...

Python, Go and then Rust

I want to write less code and reuse community code...

Python, Go and then Rust

Conclusions

I want to have better telemetry and network communication ...

Go, Rust and then Python

I want to perform lower level solutions, packet manipulation and hardware/software control...

Rust and then Go (Python excluded)

I want to contribute to the community...

Rust, Go and then Python

Thank you !

Drop a message and let's talk about network automation, discuss solutions for network performance, network simulation, traffic analysis, network management, and more.

www.telcomanager.com

Network Automation Go/Python book: a.co/d/iO7iXMe

Github: github.com/brnuts

Linkedin : www.linkedin.com/in/claus-topke/

Work email: claus@telcomanager.com

Personal email: claus.topke@gmail.com