

Introduction

Topic: Rust vs Python for AWS Lambda Functions

Rust focuses on safety and performance. It doesn't pretend to make things easy.

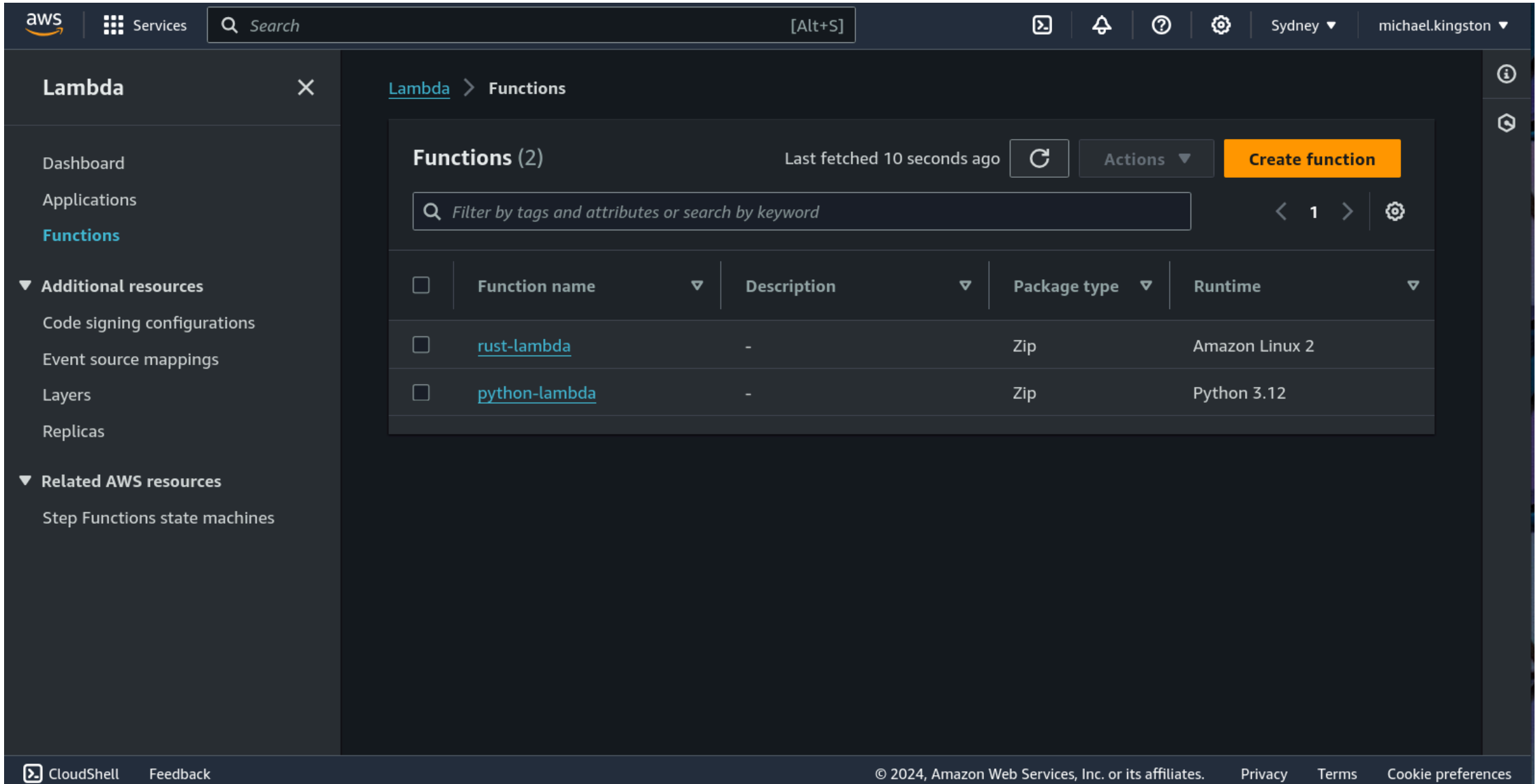
Rust History

- **2006:** created by Graydon Hoare
- **2009:** Mozilla became involved
- **2010:** official public announcement
- **2015:** v1.0 released
- **2020:** AWS joined to Rust foundation
- **2023:** AWS announced Rust SDK for AWS



Rust Features (in comparison to Python)

Feature	Rust	Python
Compilation	Compiled	Interpreted
Concurrency	Native concurrency support	GIL
Memory Management	No Garbage Collector	Garbage Collector
Performance	Petaflop-level speed	Slower in comparison
Error Handling	Strict compile-time checks	Dynamic typing and runtime checks

Visualising the difference between compiled and interpreted languages



The screenshot displays the AWS Lambda console interface. The top navigation bar includes the AWS logo, a search bar, and user information for 'Sydney' and 'michael.kingston'. The left sidebar shows the 'Lambda' service selected, with a sub-menu for 'Functions'. The main content area, titled 'Functions (2)', shows a list of two functions: 'rust-lambda' and 'python-lambda'. The 'rust-lambda' function is configured with 'Amazon Linux 2' as the runtime, while 'python-lambda' uses 'Python 3.12'. Both functions have a 'Zip' package type and no description. The interface includes a search bar for filtering functions and a 'Create function' button.

Functions (2) Last fetched 10 seconds ago  **Actions**  **Create function**

<input type="checkbox"/>	Function name	Description	Package type	Runtime
<input type="checkbox"/>	rust-lambda	-	Zip	Amazon Linux 2
<input type="checkbox"/>	python-lambda	-	Zip	Python 3.12

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Why is AWS investing in Rust?

- Memory Safety
- Performance
- Resource efficiency & cost
- Environment

How is AWS using Rust?

AWS has already deployed a number of projects that use Rust, including

- Firecracker
- Bottlerocket
- AWS IoT Greengrass
- Amazon Prime
- Lambda Runtime
- AWS SDK for Rust
- the last 2 are what interest us today

Apples & Oranges?

Given the differences between the two languages, is this a fair comparison?

The Rust Sales Pitch

Using Rust instead of Python (or TypeScript) to write your lambda functions will make them

- faster
- more resource efficient
- cheaper
- "Green", ie much more energy efficient

Sounds too good to be true? Thats kinda true. There are a number of caveats that are important and seldom mentioned we'll talk about these at the end of the presentation

The Test

I've written two lambda functions, one in Rust, one in Python. They do the same thing

- generate a UUID
- save it as a .txt file in an S3 bucket
- enter it as a record into a DynamoDB table
- we use K6 to test the performance of the two functions under a range of conditions

Its intended to be a simple example that we can replicate easily

[K6](#) is a free performance testing tool

The Code

- look at the script for the Rust lambda function
- compare with the Python script
- demonstrate how to compile the Rust code
- demonstrate errors in Rust

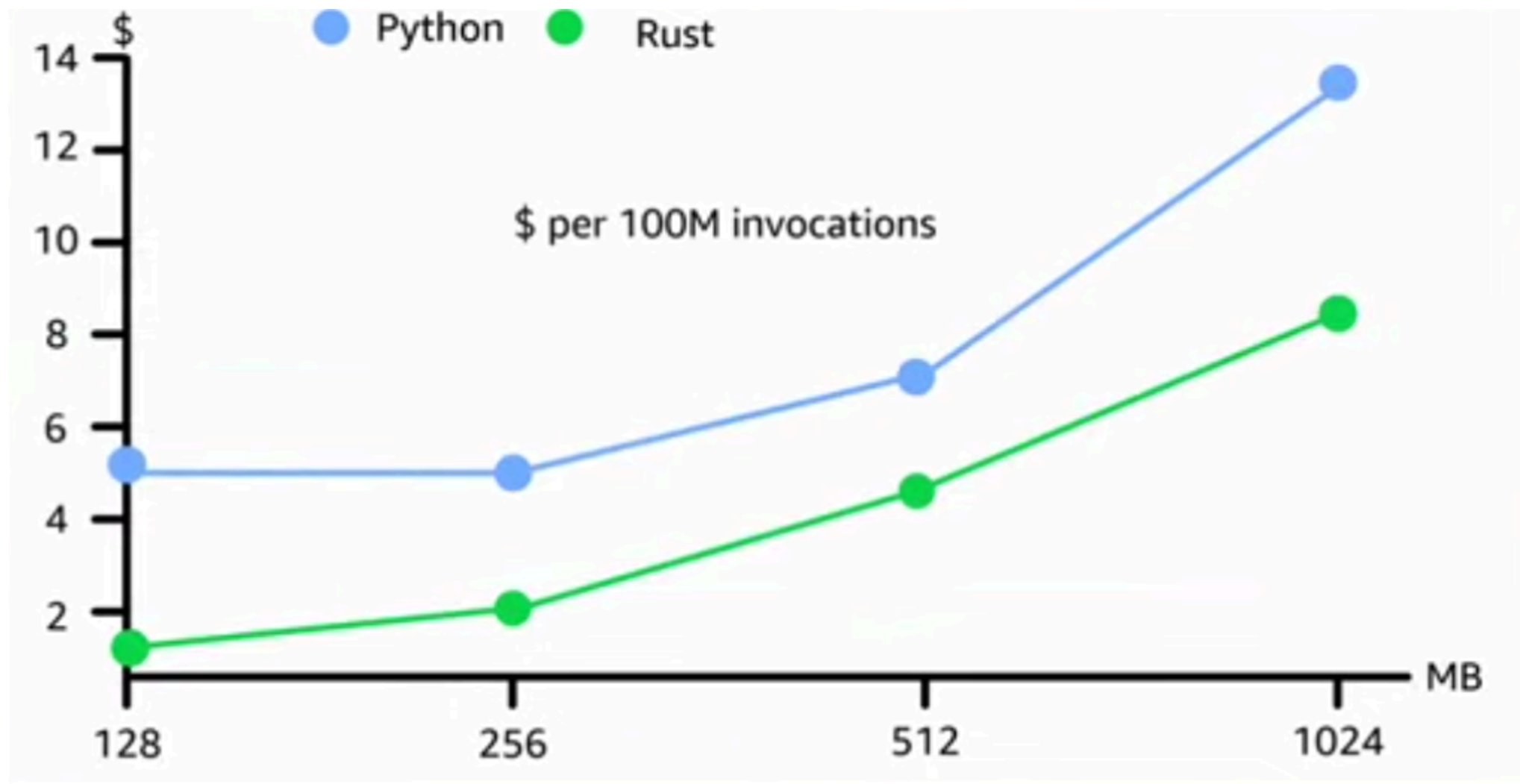
The Results: Cost

Confession: I could not achieve the 8 - 10x improvement that others have written about. I believe it's because my Rust code is not as optimised as it could be. Lets call it an opportunity for improvement

However, I did achieve 3-4x improvement in cost. This is still significant, but a little disappointing

let's look at costs first, because for most of you i expect that's the most interesting

Cost comparison

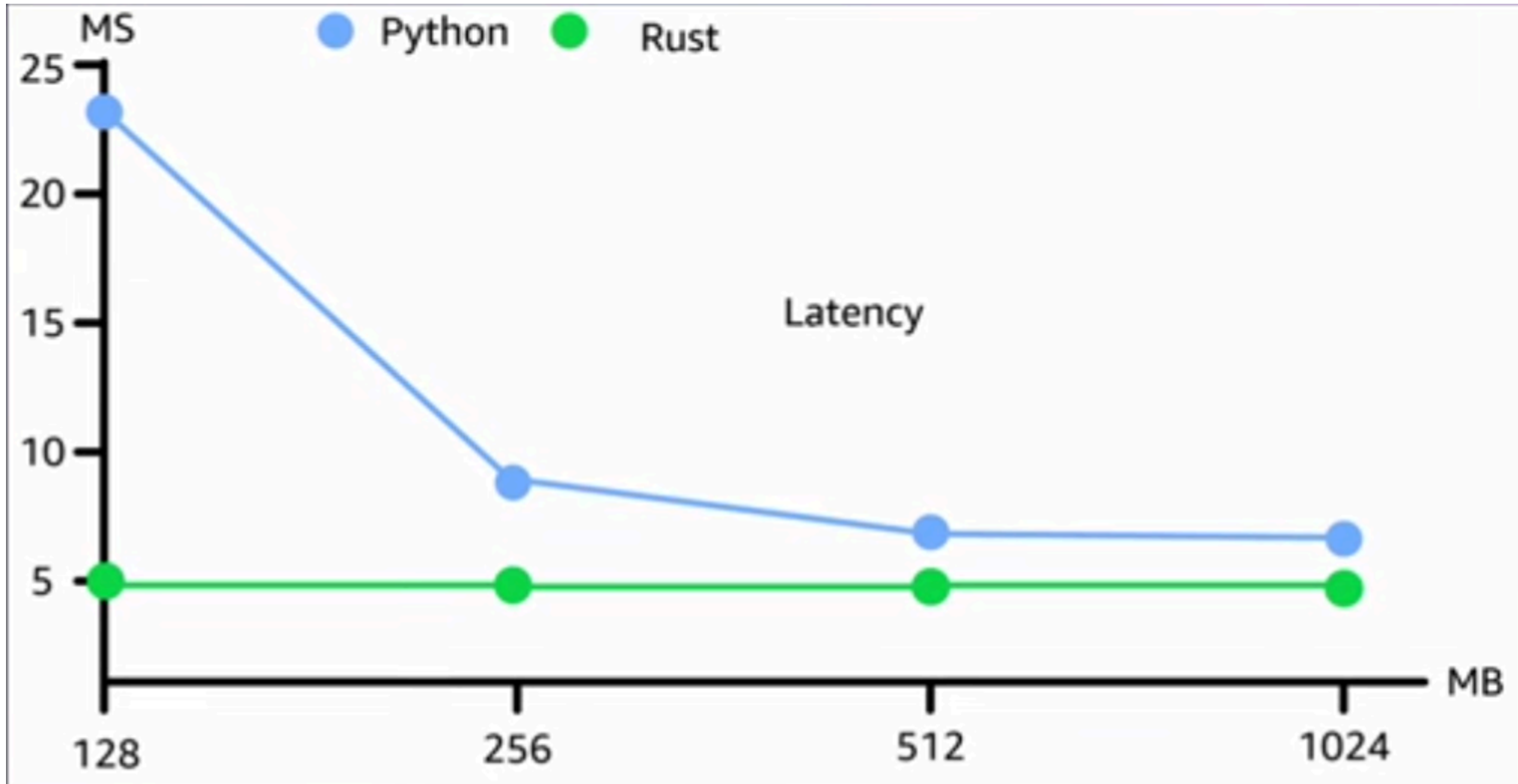


Language	Time	Cost	Comparison
Rust	5ms	\$0.00000000105	
Python	23ms	\$0.00000000210	360% slower. 357% more expensive

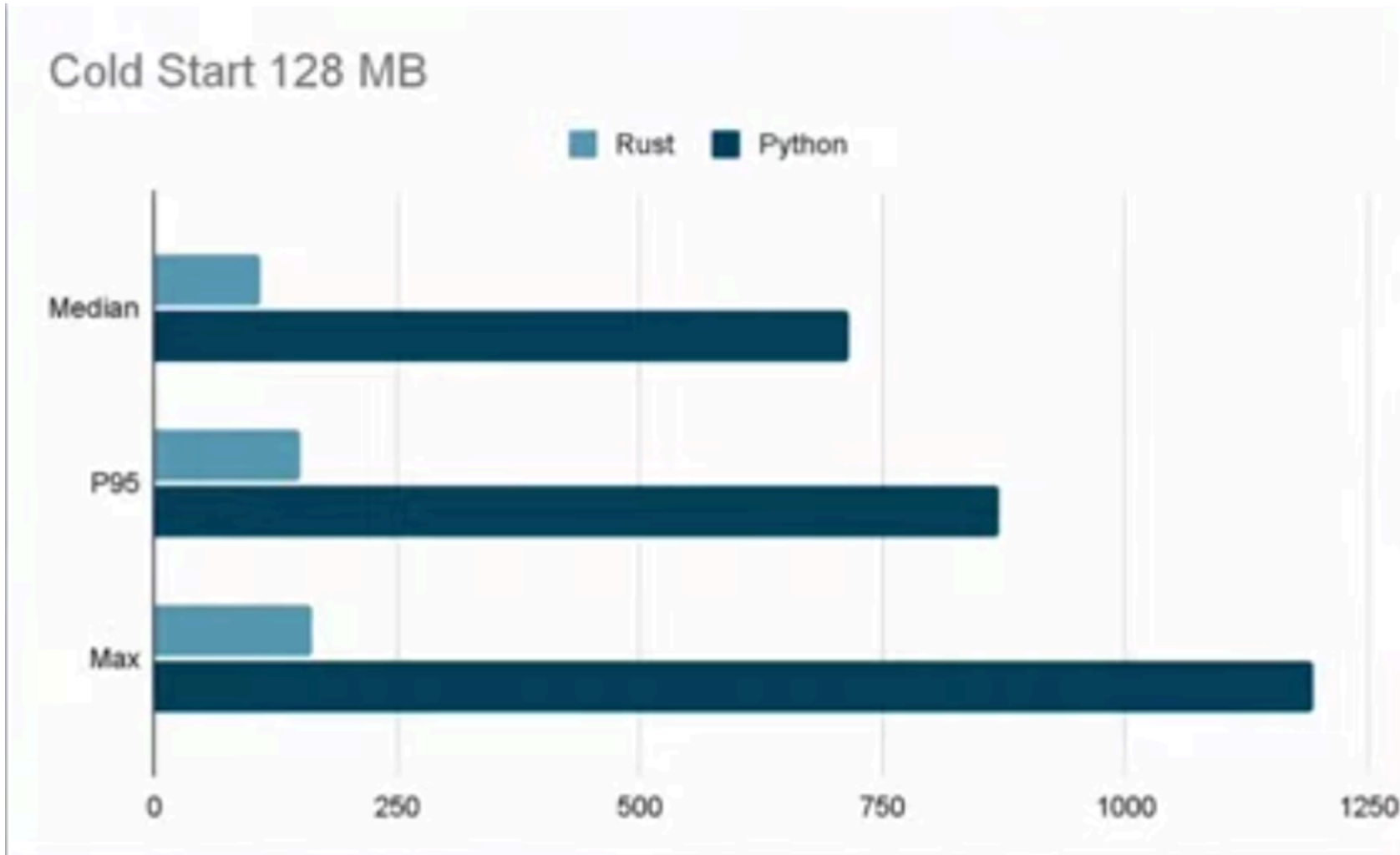
For 100m lambda invocations:

- Rust: \$1.05
- Python: \$4.80

Results: latency



Cold Start Comparison



The Experience

I think that the developer experience is important to mention. I've found that

- Rust is harder to learn and write than Python.
- Rust is significantly more verbose than Python.
- Rust throws a LOT more errors during development,
- the frustration is eased somewhat by the excellent messages the compiler provides.
- the size of the Rust zip file is substantially smaller than the Python equivalent
- Rust is apparently very popular, but there are not many Rust developers. This seems odd.
- Rust is not as mature as Python

Conclusions

- Rust delivers a massive improvement in performance, along with the other oft touted benefits
- Rust moves the cost to the development phase.
- Rust fixes old problems but introduces new ones too, specifically, at the development stage
- Rust trades costs on AWS for increased development costs
- There is a compromise solution which we'll come to

Why and when to use Python

- You can build things quickly in Python, so its the obvious choice When you're building prototypes
- Python is ubiquitous in data engineering, data science and machine learning. You'll have no problems finding developers who can work with it.
- For things that need to be read, understood and maintained by many developers, Python is the obvious choice.
- Everything has Python SDKs and APIs

Why and when to use Rust

- You can build the best tooling with Rust. It just takes more time.
- When you need maximum performance, stability and memory safety, you should consider Rust for your application. Its a clear alternative to C++ and C.
- Suitable for applications requiring high concurrency and memory safety. Faster than languages like Go.
- Ideal for infrastructure and systems programming.

A Compromise solution

- Its not necessary to re-write your entire code base
- You can write performance-critical components in Rust, while keeping the rest of your codebase in Python.
- 50-70% of the benefits

Why choose Rust over GO?

Overall, both languages are meant for many of the same purposes, and they both attempted to solve the same problem: In a C++ code base, you end up with 5+ year old code snippets everywhere that everyone's too afraid to touch in fear of breaking the entire program.

Go strives to distance itself from complexity -- code in Go is best seen as "composition from simple, easy to understand pieces," so that anyone approaching the code base has a chance to work on it.

The Rust philosophy is that you're willing to

- sacrifice ease of use now, and argue with the compiler
- spend a more time researching your solution, if it means
- in exchange you can write a code base that's still going strong five+ years from now.

Conclusions

- I like Rust and plan to continue learning and building projects with it.
- However I don't see it as a replacement for Python, rather perhaps as a compliment to it. I have this idea in my head: "Develop in Python, Deploy in Rust"
- My next task is to build a full ML pipeline in Rust, including serverless inference. I'm going to use [Candle](#) and the Rust SDK for AWS. If people are interested, I'd love to sure to share my project with you.

Acknowledgements

- a big thank you to Alan Blockley for coaching me in preparation for this presentation
- Also a big thank you to Illya Kavaliou from mantel Group. He runs the serverless group and his recent presentation was an inspiration for this experiment.

References

- **Rust Home** [Rust Programming Language](#)
- **Lambda vs EC2 Comparison** by [Illya Kavaliou](#)
- **Rustifying Serverless** by [Efi Merdler-Kravitz](#)
- **Sustainability with Rust** [AWS Blog](#)
- **Why AWS is the Best Place to Run Rust** [AWS Blog](#)

