zetta scale

Zenoh

High-Performance Networking with Rust

Angelo Corsaro

zetta scale

Why Rust?

zetta scale

Productivity

We find that Rust is an extremely productive system programming language

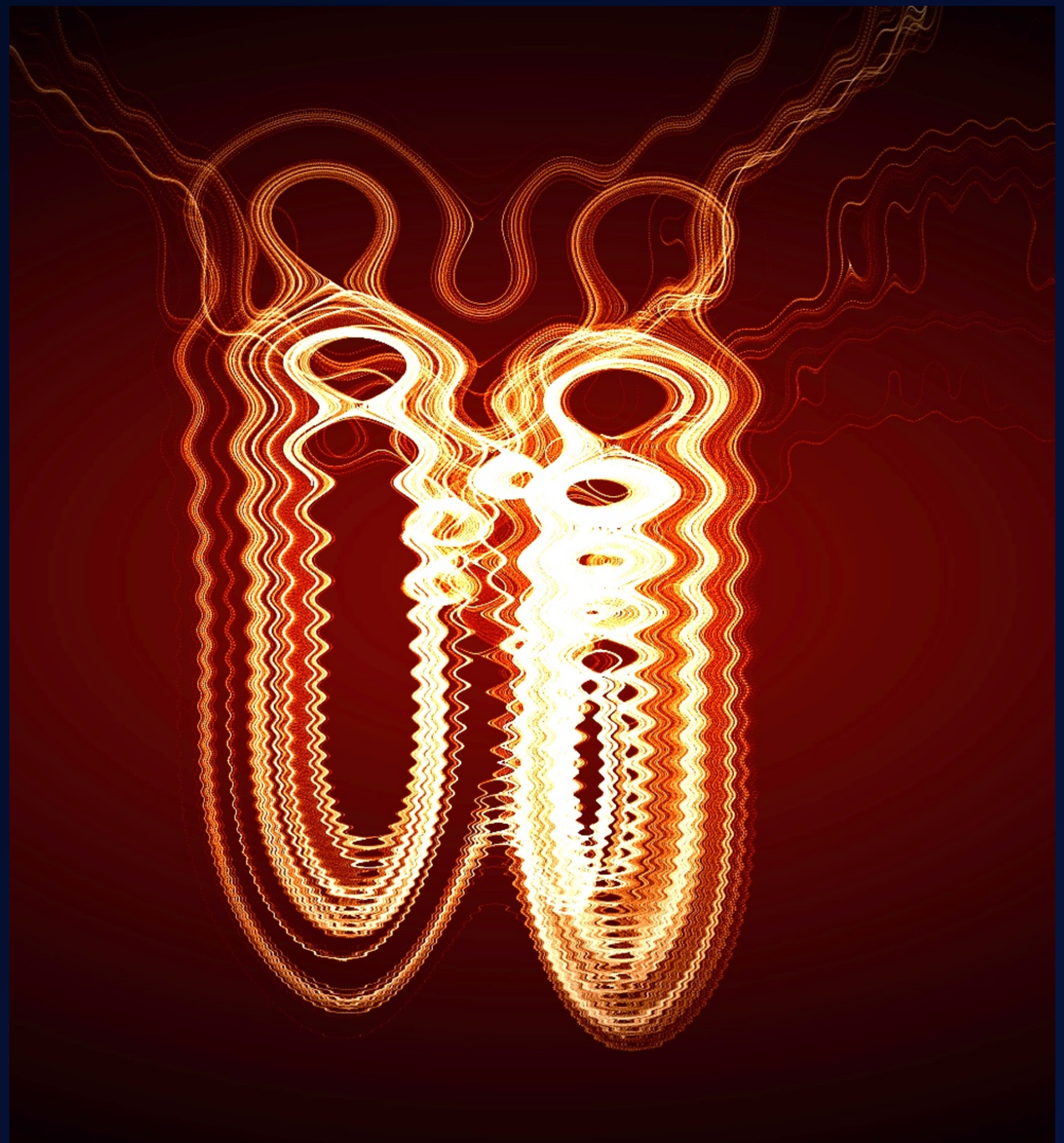
It has a learning, curve, but once that is passed, we see much better productivity when compared to C/C++

zetta scale

Asynchronous Programming

We leverage heavily Rust support for Asynchronous programming for both I/O (networking code) as well as API

This makes it easier to write concurrent code that has no (or limited) inversion of control



zetta scale

Memory and concurrency safe programming language => Safe and Secure

High-level abstractions => Productive

Zero-Cost Abstractions => High Performance

Built-in Support for Asynchronous Programming => Great for Network Programming

zetta scale

High Performance with Rust

Usual Recommendations

Concurrency

One OS-level thread per core

Limit / Avoid locking

Limit / Avoid Context Switches (especially of OS Threads)

Memory

Avoid/limit dynamic memory allocation on the critical path

Use data structures that are cache affine, especially on your critical path (e.g. favour contiguous data structures implementations)

zetta scale

Measure don’t Guess

Systematically measure performance

Great tools such as Criterion are available

Build performance tests that are relevant for your application

Look at the raw data and do proper statistics on it

Profile your code (perf, vtune, etc)

Avoid Surprises

Key Libraries

When choosing a library, make sure you evaluate it

Don’t just pick a library because everyone else is using it, perhaps they have different needs than you

zetta scale

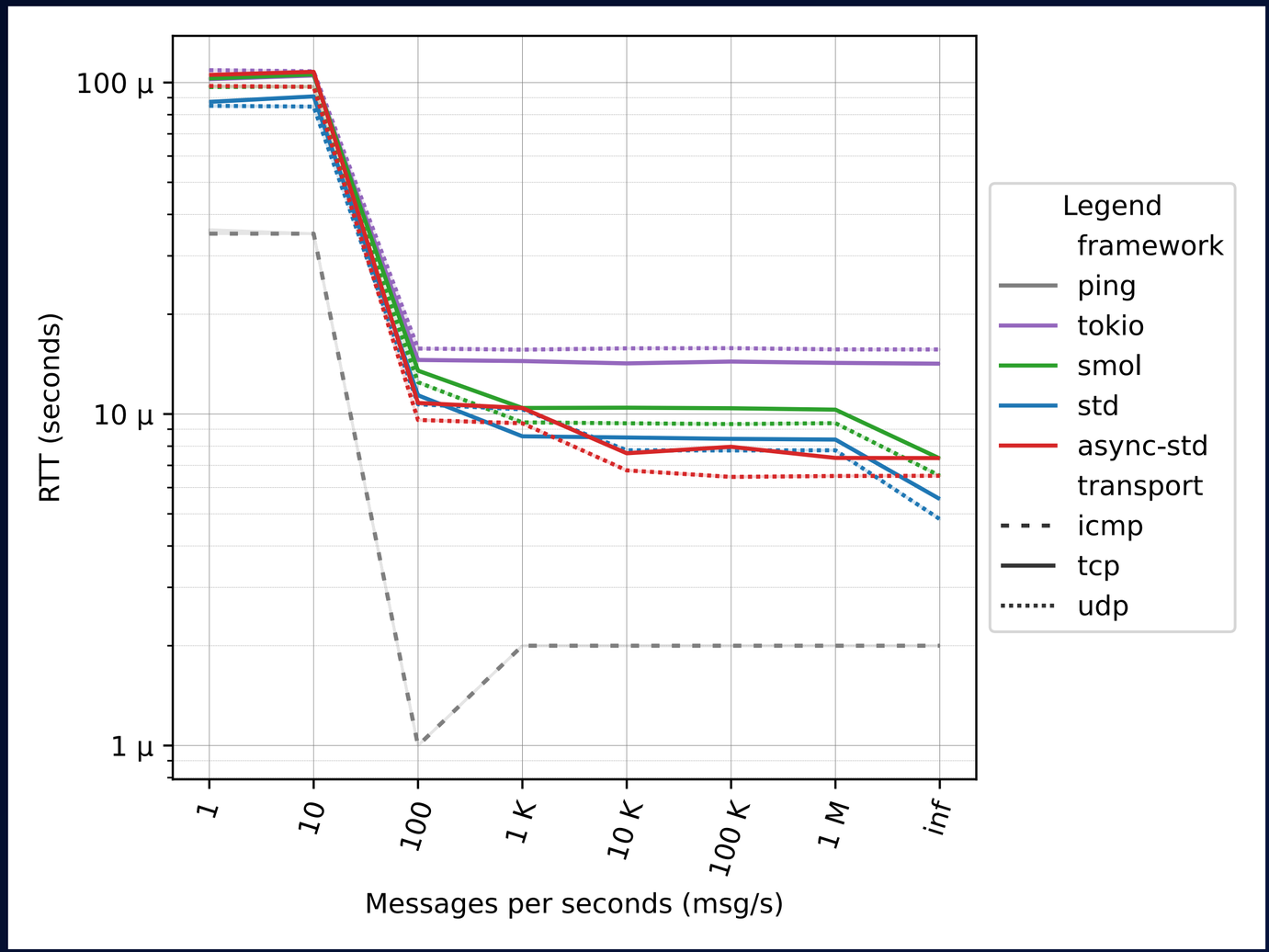
Rust Async Libraries

zetta scale

Popularity vs Performance

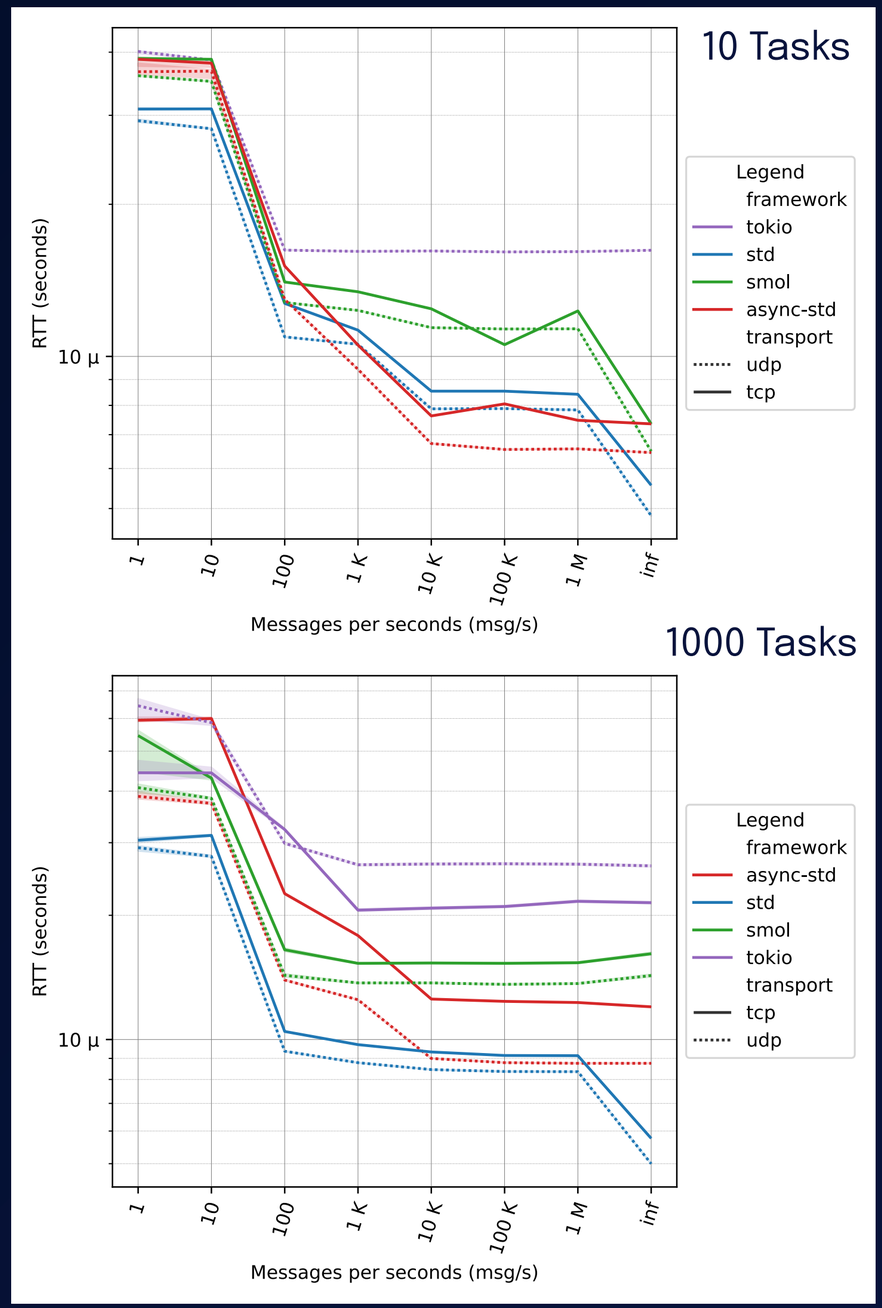
Today Tokio seems to be the dominant asynchronous framework

Yet, is severely underperforms when compared to async-std



Impact of Async Tasks

Tokio performance get even worse as the number of tasks in a runtime increases



zetta scale

Tokio vs async-std

The situation is getting rather complicated as more and more library adopt Tokio, perhaps because everyone else uses it...

Yet its performance are not at the point and additionally it creates issues of resource usage when mixed with async -std

zetta scale

Async & Your Stack

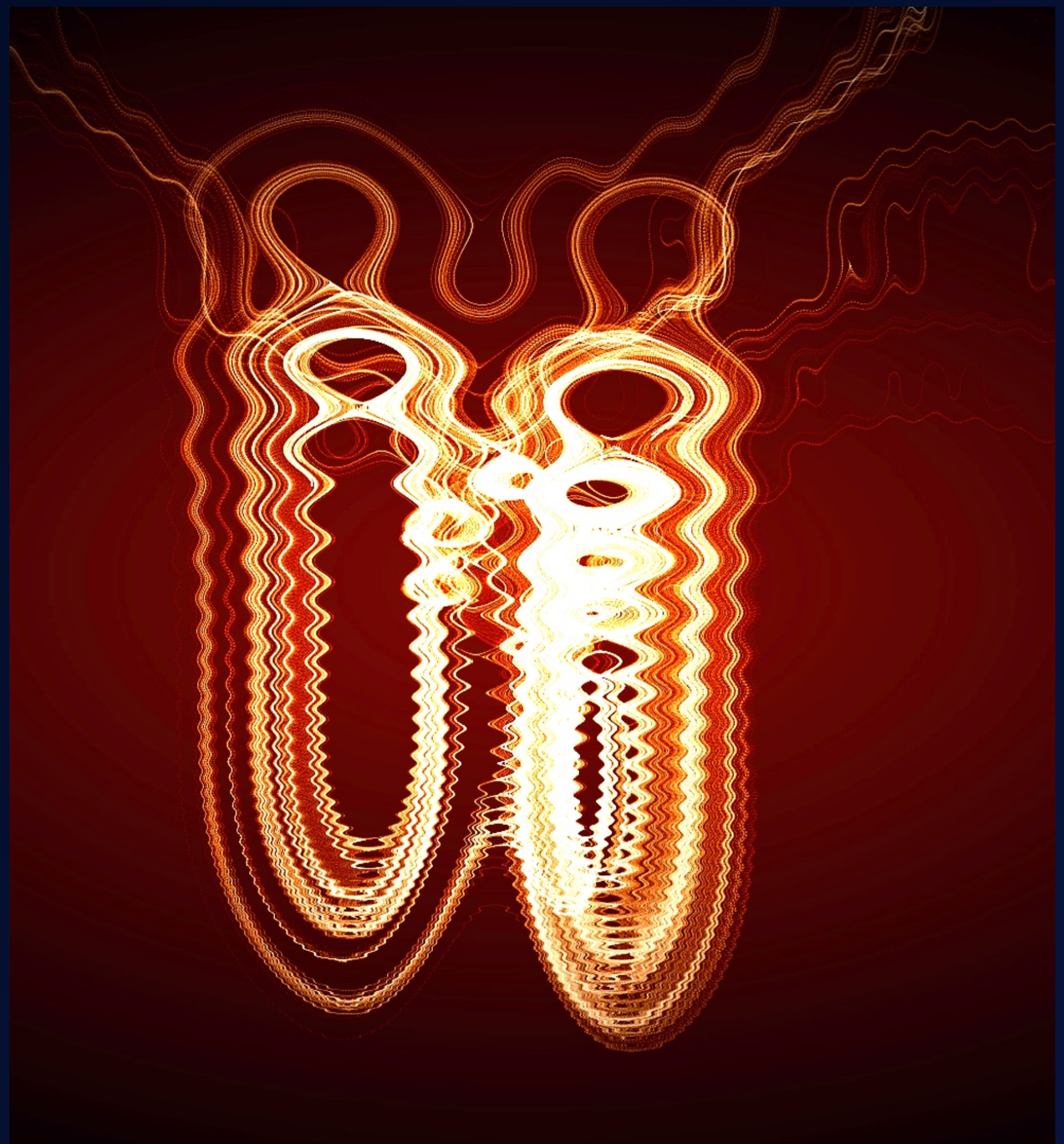
zetta scale

Asynchronous Programming

Hard to profile performance

The impact on the stack size can seriously hit performance

Measure and profile the size of your futures as well as the impact of nested call... It can kill your performances!



Concluding Remarks

zetta scale

Step after Step

The path to performance is all about a disciplined step after step journey

Many of the tricks are the same that apply to C/C++

Rust has some specials, and great attentions should be used with aynch/futures