Accessible Software Performance

Alexander “zamazan4ik” Zaitsev

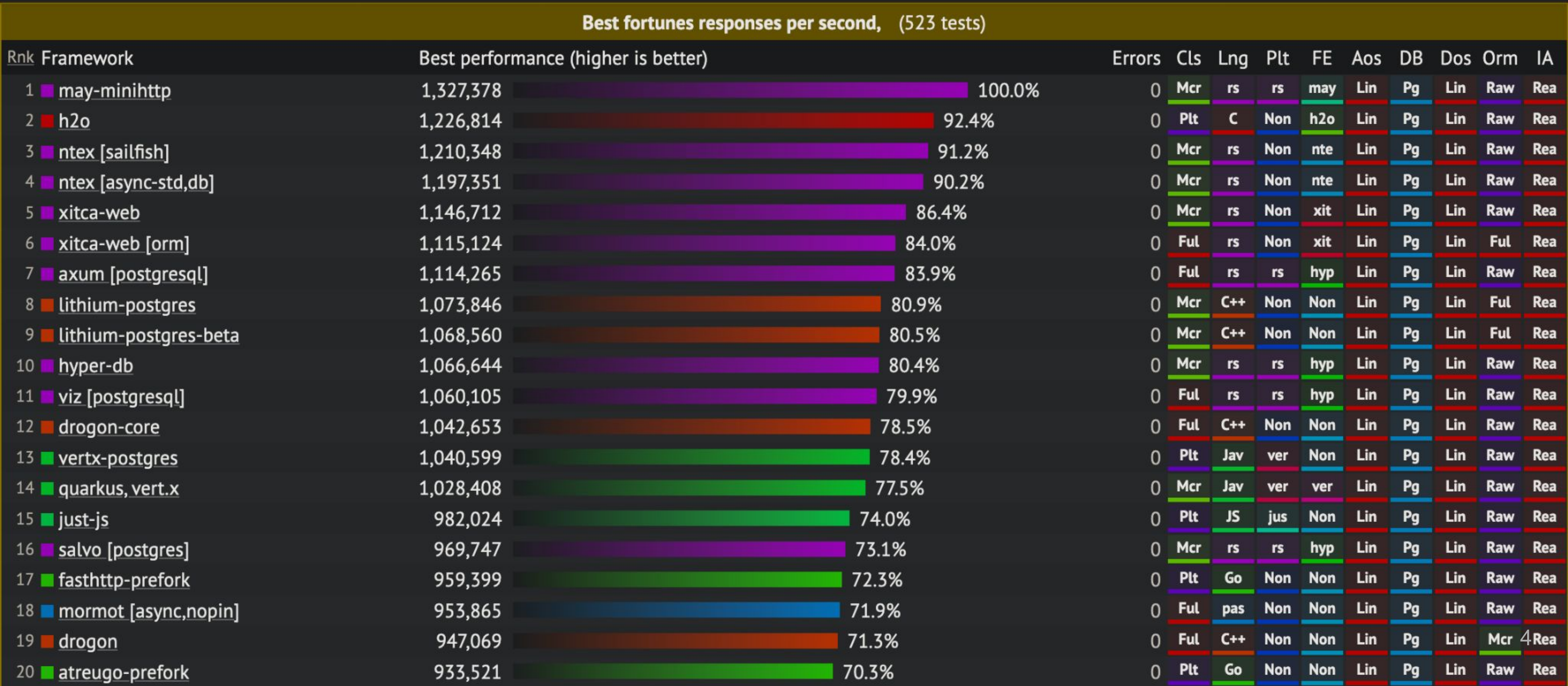
Few words about me

- Used to be a C++ engineer (now my C++ skills are Rust-y ;)
- Spent several years on “hacking” LLVM compiler/static analyzers/C++ standard library (libc++), etc.
- Awesome PGO author
- “Software Performance” devroom organizer (you are sitting here ;)
- Interested in data-driven software optimizations
- Like **rapid** software!

What do we usually hear about performance?

- Various benchmark reports

Example: TechEmpower “benches”



Best fortunes responses per second, (523 tests)

Rnk	Framework	Best performance (higher is better)		Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	🟪 may-minihttp	1,327,378	100.0%	0	Mcr	rs	rs	may	Lin	Pg	Lin	Raw	Rea
2	🟥 h2o	1,226,814	92.4%	0	Plt	C	Non	h2o	Lin	Pg	Lin	Raw	Rea
3	🟪 ntex [sailfish]	1,210,348	91.2%	0	Mcr	rs	Non	nte	Lin	Pg	Lin	Raw	Rea
4	🟪 ntex [async-std,db]	1,197,351	90.2%	0	Mcr	rs	Non	nte	Lin	Pg	Lin	Raw	Rea
5	🟪 xitca-web	1,146,712	86.4%	0	Mcr	rs	Non	xit	Lin	Pg	Lin	Raw	Rea
6	🟪 xitca-web [orm]	1,115,124	84.0%	0	Ful	rs	Non	xit	Lin	Pg	Lin	Ful	Rea
7	🟪 axum [postgresql]	1,114,265	83.9%	0	Ful	rs	rs	hyp	Lin	Pg	Lin	Raw	Rea
8	🟧 lithium-postgres	1,073,846	80.9%	0	Mcr	C++	Non	Non	Lin	Pg	Lin	Ful	Rea
9	🟧 lithium-postgres-beta	1,068,560	80.5%	0	Mcr	C++	Non	Non	Lin	Pg	Lin	Ful	Rea
10	🟪 hyper-db	1,066,644	80.4%	0	Mcr	rs	rs	hyp	Lin	Pg	Lin	Raw	Rea
11	🟪 viz [postgresql]	1,060,105	79.9%	0	Ful	rs	rs	hyp	Lin	Pg	Lin	Raw	Rea
12	🟧 drogon-core	1,042,653	78.5%	0	Ful	C++	Non	Non	Lin	Pg	Lin	Raw	Rea
13	🟩 vertx-postgres	1,040,599	78.4%	0	Plt	Jav	ver	Non	Lin	Pg	Lin	Raw	Rea
14	🟩 quarkus, vert.x	1,028,408	77.5%	0	Mcr	Jav	ver	ver	Lin	Pg	Lin	Raw	Rea
15	🟩 just-js	982,024	74.0%	0	Plt	JS	jus	Non	Lin	Pg	Lin	Raw	Rea
16	🟪 salvo [postgres]	969,747	73.1%	0	Mcr	rs	rs	hyp	Lin	Pg	Lin	Raw	Rea
17	🟩 fasthttp-prefork	959,399	72.3%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
18	🟦 mormot [async,nopin]	953,865	71.9%	0	Ful	pas	Non	Non	Lin	Pg	Lin	Raw	Rea
19	🟧 drogon	947,069	71.3%	0	Ful	C++	Non	Non	Lin	Pg	Lin	Mcr	Rea
20	🟩 atreugo-prefork	933,521	70.3%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea

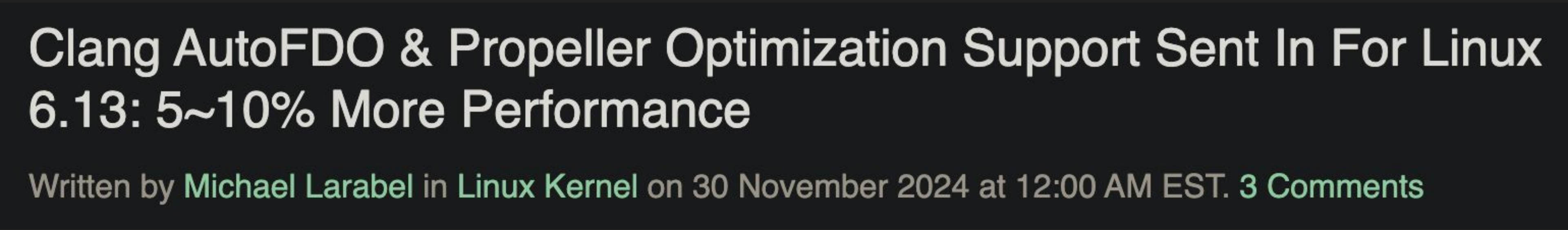
What do we usually hear about performance?

- Various benchmark reports
- Engineering blogs like "We’ve rewrote smth ~~in Rust~~ and now our performance is blazing fast!"
- Hardcore / low-level stuff about performance optimizations
- Academy papers about various performance-related stuff like a new algorithm for something (e.g. the simd-json paper)

But one point is frequently missed...

How **accessible to you** is yet another performance win?

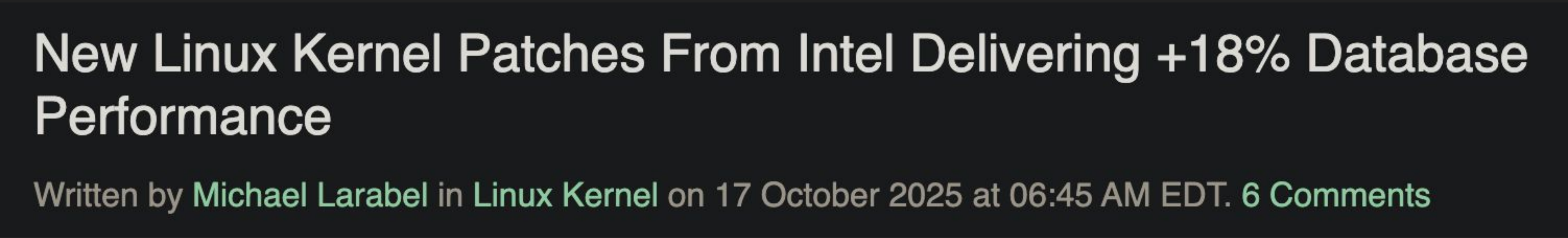
Which improvement is easier to use and why?



Clang AutoFDO & Propeller Optimization Support Sent In For Linux 6.13: 5~10% More Performance

Written by Michael Larabel in Linux Kernel on 30 November 2024 at 12:00 AM EST. 3 Comments

OR



New Linux Kernel Patches From Intel Delivering +18% Database Performance

Written by Michael Larabel in Linux Kernel on 17 October 2025 at 06:45 AM EDT. 6 Comments

Compilers, their compilation models, and data-driven optimizations

AOT vs JIT in workload-specific optimizations

- In JIT in most cases it "just works"
- In AOT world it's called PGO. And it needs to be integrated separately

Is PGO that important?

Application	**Improvement**	**Library**	**Improvement**
Rustc	up to +15% compilation speed	serde_json	~15% improvement
Vector	+15% EPS	xml-rs	~35% improvement
Rust Analyzer	+20% speedup	quick-xml	~25% improvement
PostgreSQL	up to +15% faster queries	tonic	~10% improvement
SQLite	up to 20% faster queries	rustls	~6% improvement
ClickHouse	up to +13% QPS	axum	~10% improvement
MySQL	up to +35% QPS	tantivy	~30% improvement
MongoDB	up to **2x** faster queries	wgpu	~25% improvement
Redis	up to +70% RPS	tracing libs	~35-40% improvement

To enable PGO you need…

1. Compile your application with instrumentation
2. Run on a target workload and collect a profile
3. Recompile your application once again with the profile
4. You got your PGO-optimized app!

However, additionally you get …

PGO IMPLEMENTER

"This will definitely improve performance" -- breaks production

"Trust me, I know what I'm doing" -- hasn't read the documentation

"We'll see 20% speedup easy" -- 0.1% improvement after 3 months

"Just need to tweak a few more flags" -- 500 compiler errors

"Just one more build cycle" -- said 6 hours ago

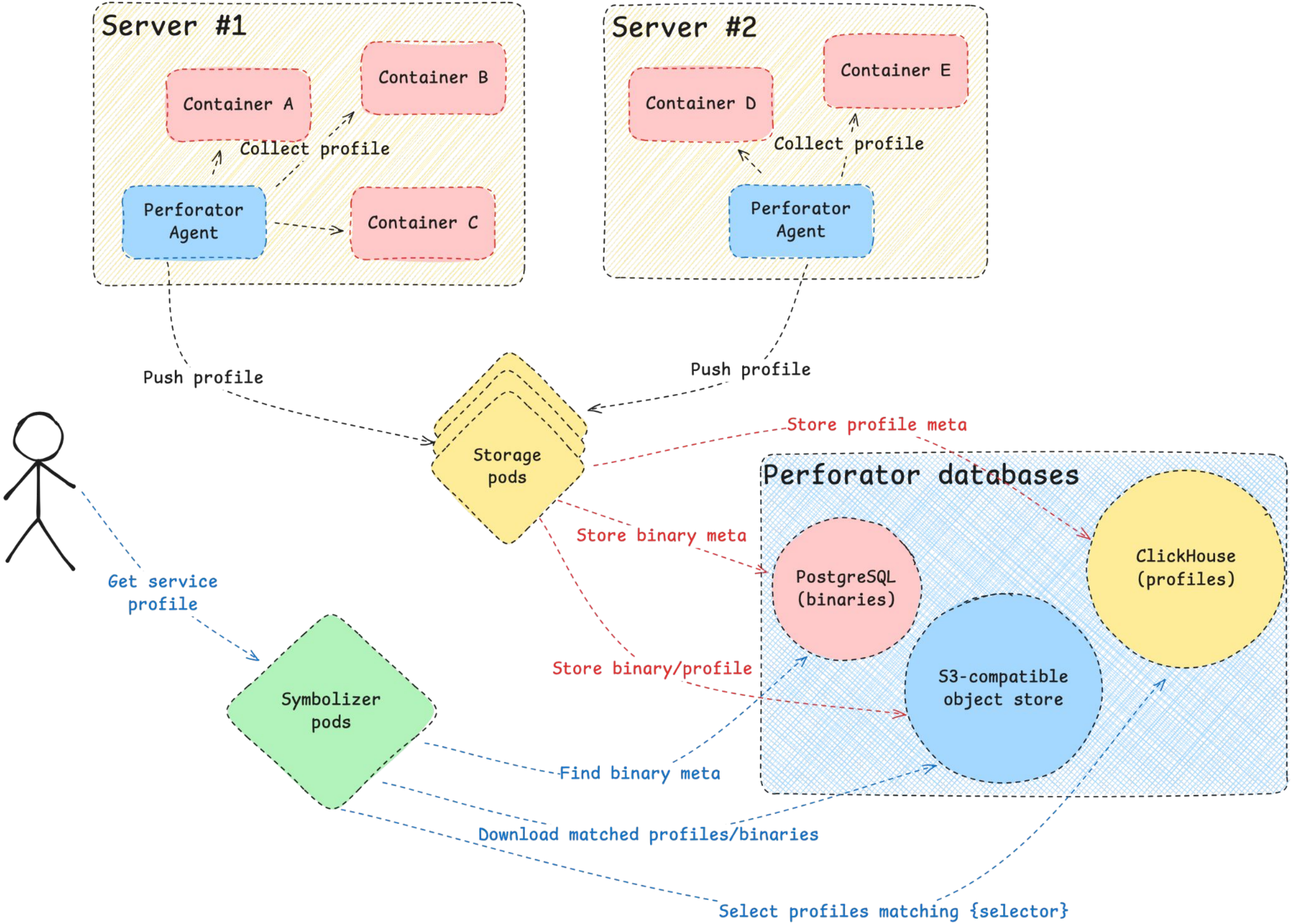
"It's not my code, it's the compiler" -- forgot to enable optimizations

"Profile-guided optimization is the future" -- can't profile real workloads

"PGO will solve all our problems" -- introduces 17 new ones

… a bunch of additional issues!

- Double / triple compilation
- Think about profile regeneration, profile storage, profile update strategy
- Integrate PGO into your application / build system / CI properly
- Mitigate instrumentation overhead via even more complicated things like Sampling PGO and Parca / Yandex.Perforator

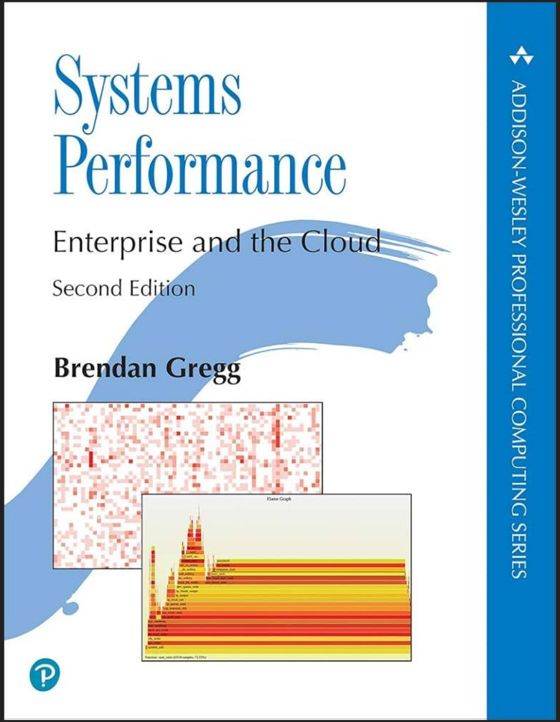

Server #1
Container A
Container B
Collect profile
Perforator Agent
Container C
Server #2
Container D
Container E
Collect profile
Perforator Agent
Push profile
Push profile
Storage pods
Store profile meta
Perforator databases
Store binary meta
PostgreSQL (binaries)
ClickHouse (profiles)
Get service profile
Store binary/profile
S3-compatible object store
Symbolizer pods
Find binary meta
Download matched profiles/binaries
Select profiles matching {selector}

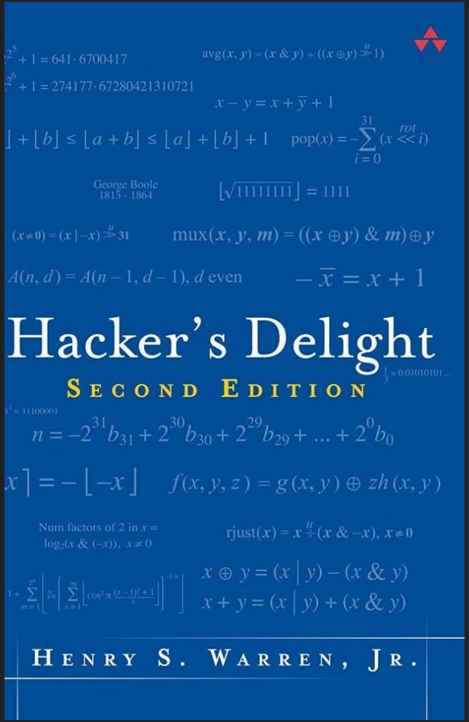
… a bunch of additional issues!

- Double / triple / even quadruple (!) compilation
- Think about profile regeneration, profile storage, profile update strategy, etc.
- Integrate PGO into your application / build system / CI **properly**
- Mitigate instrumentation PGO overhead via even more complicated things like Sampling PGO and Parca / Yandex.Perforator

Documentation

Books





Intel® 64 and IA-32 Architectures Optimization Reference Manual

Documentation Changes

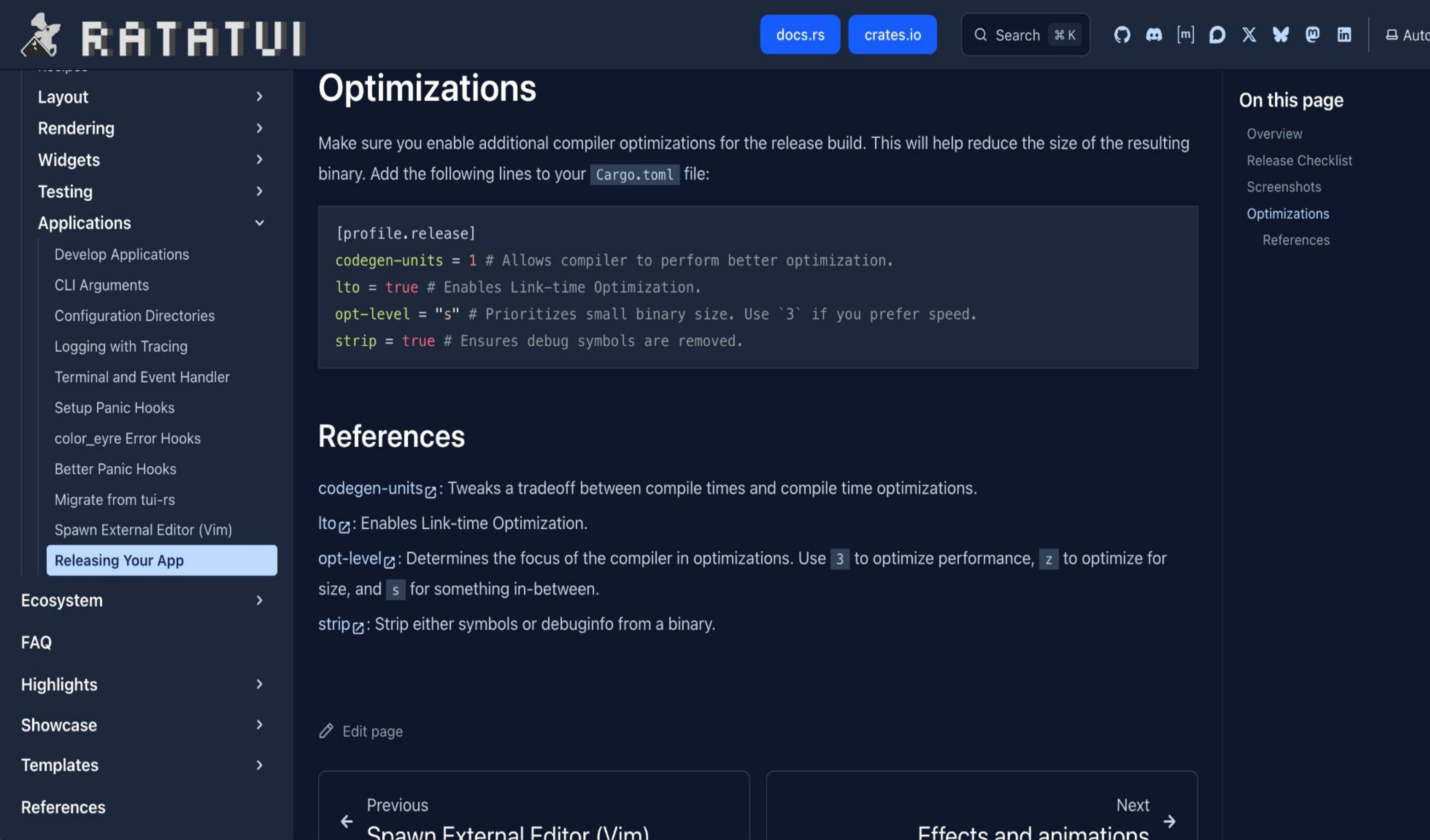
August 2022

Or in any other form

- (Unofficial) Rust Performance book
- min-sized-rust
- Project-specific optimization guidelines
- Various blogs with “Top 10 best practices” articles
- Reddit
- Youtube coding influencers
- “Awesome PGO” wall of text
- “Accessible Software Performance” talk

All of them have the same obvious issue…

Most people just don’t read/listen/watch them!



Optimizations

Make sure you enable additional compiler optimizations for the release build. This will help reduce the size of the resulting binary. Add the following lines to your `Cargo.toml` file:

```
[profile.release]
codegen-units = 1 # Allows compiler to perform better optimization.
lto = true # Enables Link-time Optimization.
opt-level = "s" # Prioritizes small binary size. Use `3` if you prefer speed.
strip = true # Ensures debug symbols are removed.
```

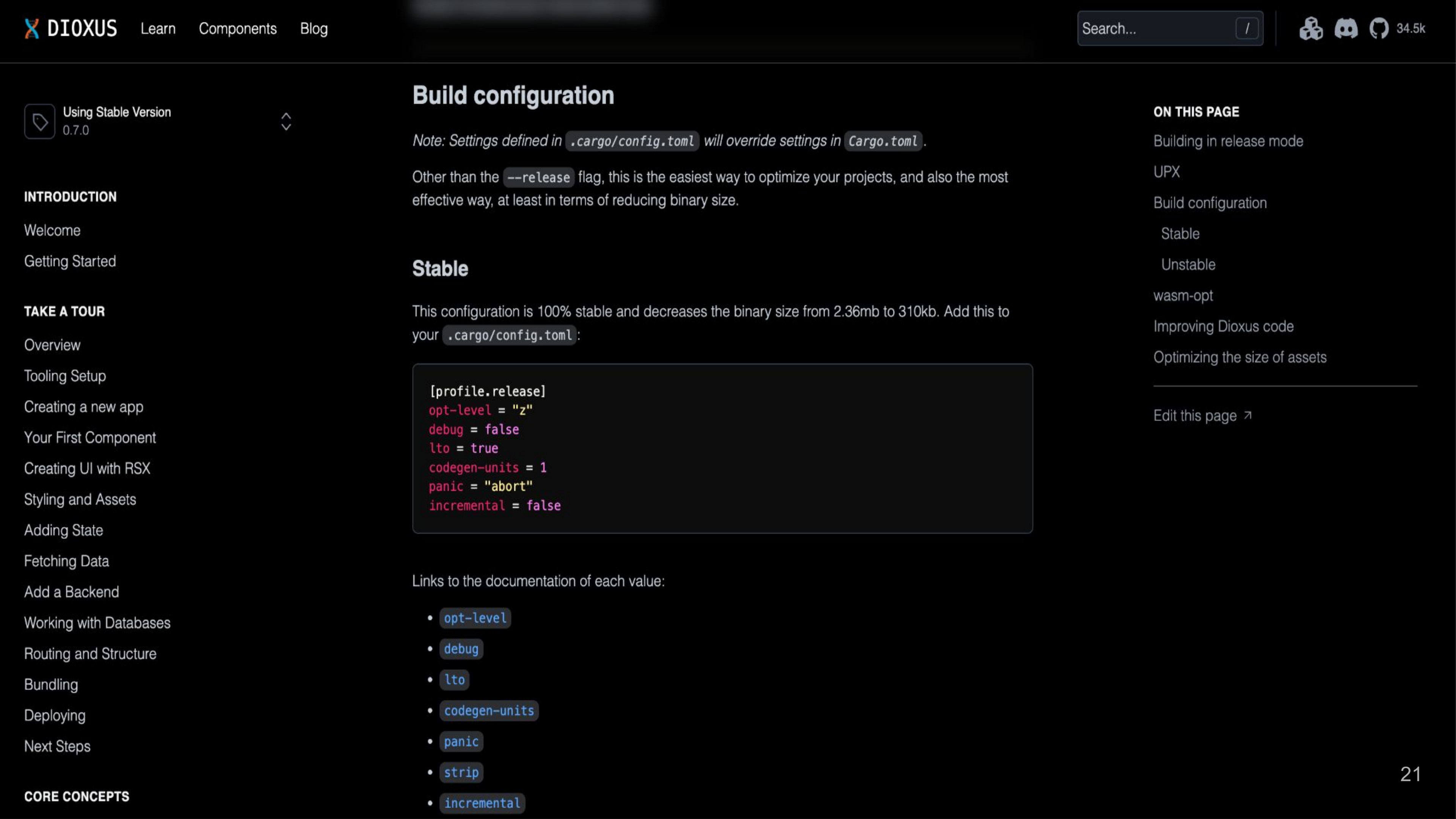
References

codegen-units: Tweaks a tradeoff between compile times and compile time optimizations.

lto: Enables Link-time Optimization.

opt-level: Determines the focus of the compiler in optimizations. Use `3` to optimize performance, `z` to optimize for size, and `s` for something in-between.

strip: Strip either symbols or debuginfo from a binary.



Build configuration

Note: Settings defined in `.cargo/config.toml` will override settings in `Cargo.toml`.

Other than the `--release` flag, this is the easiest way to optimize your projects, and also the most effective way, at least in terms of reducing binary size.

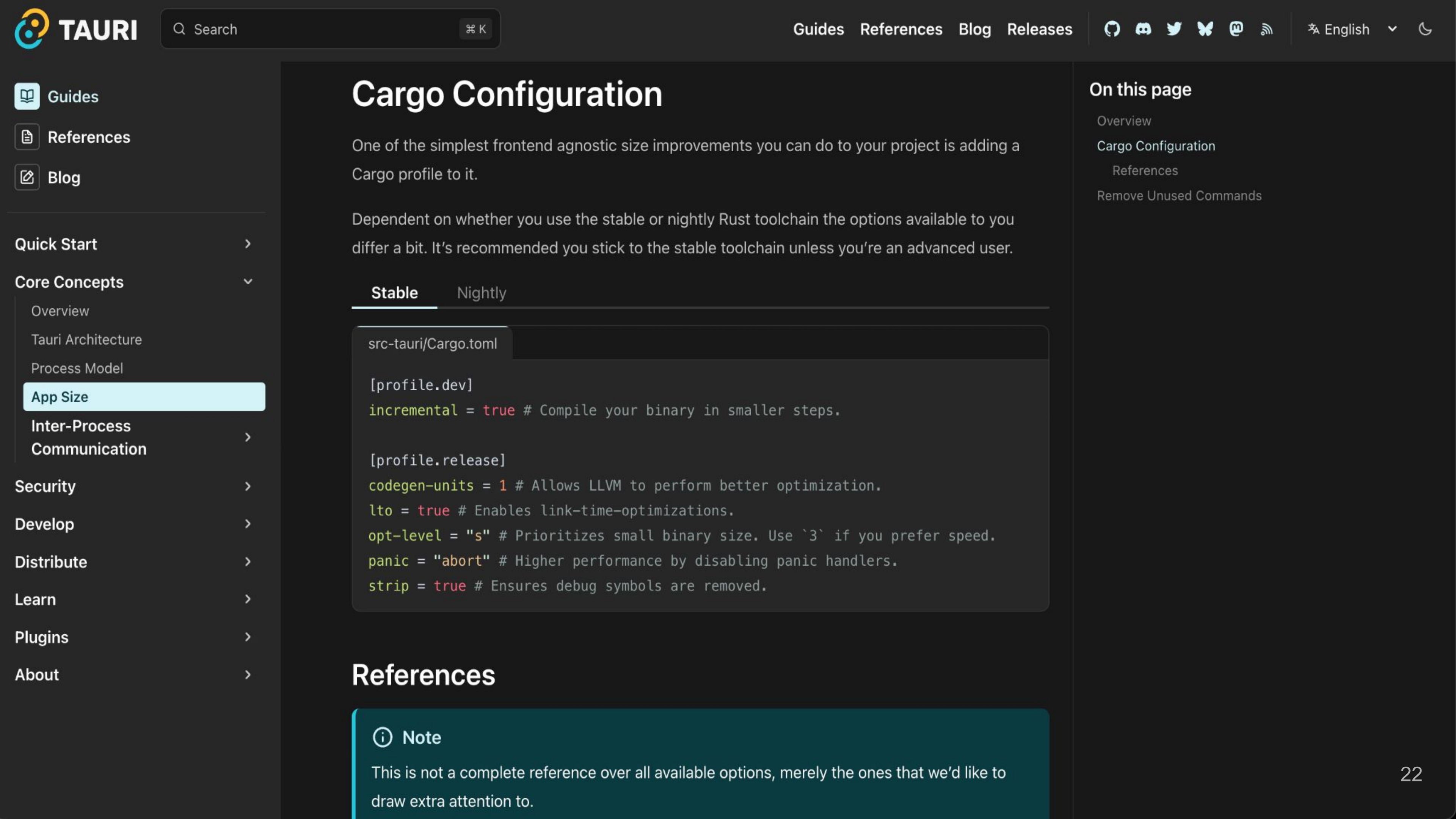
Stable

This configuration is 100% stable and decreases the binary size from 2.36mb to 310kb. Add this to your `.cargo/config.toml`:

```toml
[profile.release]
opt-level = "z"
debug = false
lto = true
codegen-units = 1
panic = "abort"
incremental = false
```

Links to the documentation of each value:

- `opt-level`
- `debug`
- `lto`
- `codegen-units`
- `panic`
- `strip`
- `incremental`



Cargo Configuration

One of the simplest frontend agnostic size improvements you can do to your project is adding a Cargo profile to it.

Dependent on whether you use the stable or nightly Rust toolchain the options available to you differ a bit. It's recommended you stick to the stable toolchain unless you're an advanced user.

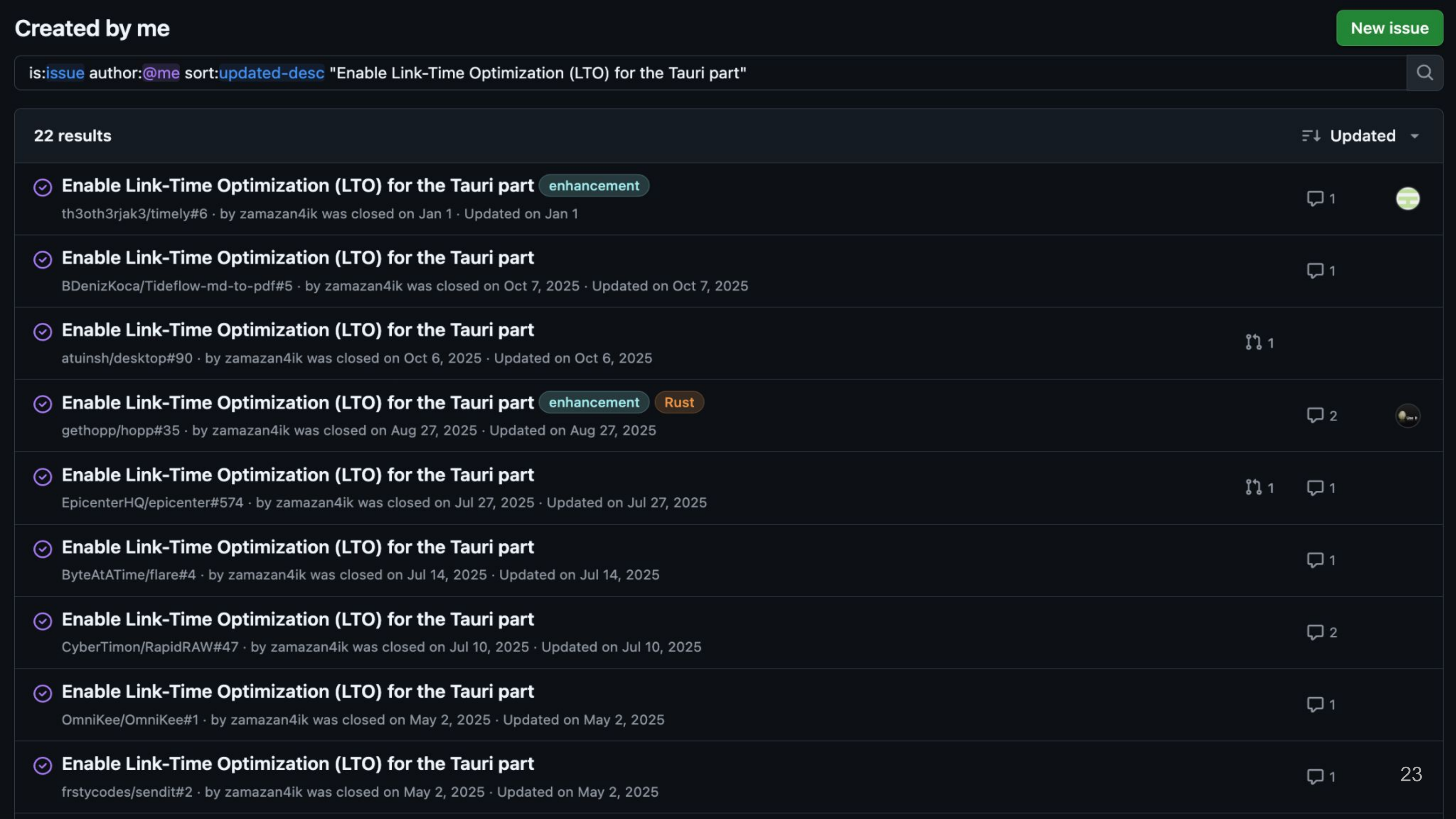
src-tauri/Cargo.toml

```toml
[profile.dev]
incremental = true # Compile your binary in smaller steps.

[profile.release]
codegen-units = 1 # Allows LLVM to perform better optimization.
lto = true # Enables link-time-optimizations.
opt-level = "s" # Prioritizes small binary size. Use `3` if you prefer speed.
panic = "abort" # Higher performance by disabling panic handlers.
strip = true # Ensures debug symbols are removed.
```

References

Note

This is not a complete reference over all available options, merely the ones that we'd like to draw extra attention to.



Created by me

```
is:issue author:@me sort:updated-desc "Enable Link-Time Optimization (LTO) for the Tauri part"
```

22 results

Enable Link-Time Optimization (LTO) for the Tauri part enhancement
th3oth3rjak3/timely#6 · by zamazan4ik was closed on Jan 1 · Updated on Jan 1
💬 1

Enable Link-Time Optimization (LTO) for the Tauri part
BDenizKoca/Tideflow-md-to-pdf#5 · by zamazan4ik was closed on Oct 7, 2025 · Updated on Oct 7, 2025
💬 1

Enable Link-Time Optimization (LTO) for the Tauri part
atuinsh/desktop#90 · by zamazan4ik was closed on Oct 6, 2025 · Updated on Oct 6, 2025
⇅ 1

Enable Link-Time Optimization (LTO) for the Tauri part enhancement Rust
gethopp/hopp#35 · by zamazan4ik was closed on Aug 27, 2025 · Updated on Aug 27, 2025
💬 2

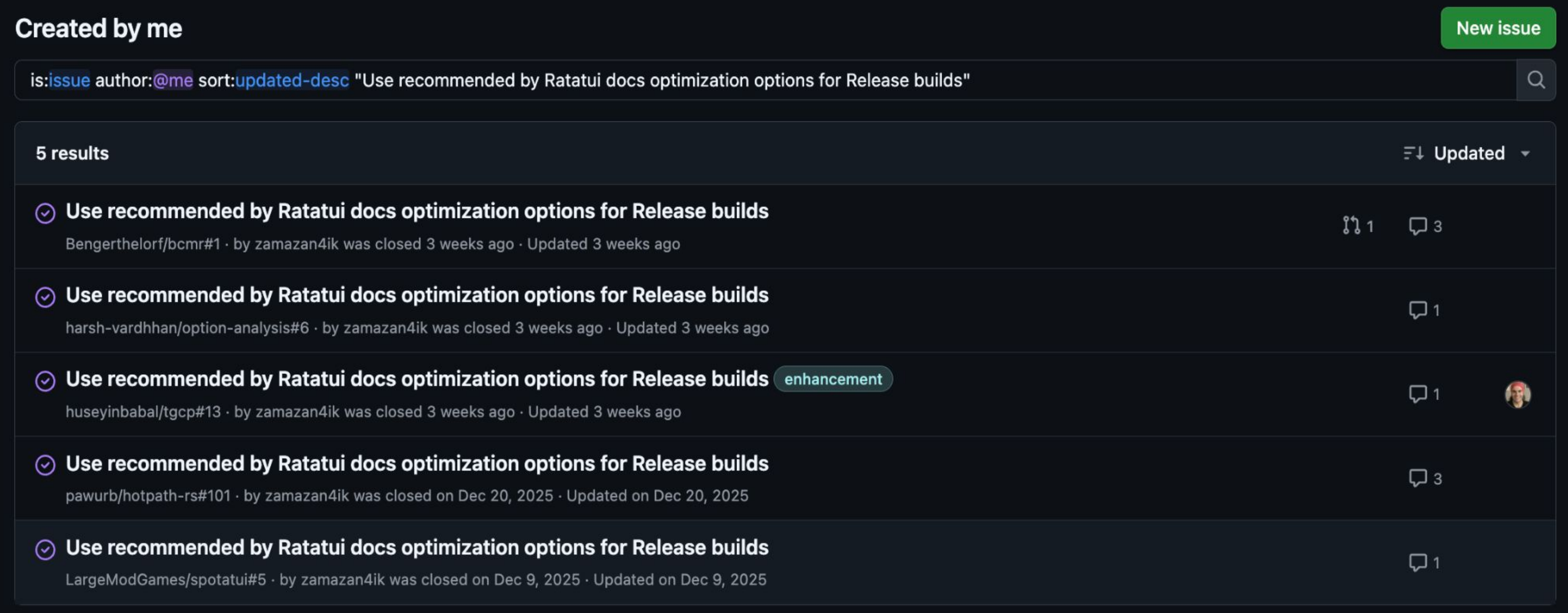
Enable Link-Time Optimization (LTO) for the Tauri part
EpicenterHQ/epicenter#574 · by zamazan4ik was closed on Jul 27, 2025 · Updated on Jul 27, 2025
⇅ 1 💬 1

Enable Link-Time Optimization (LTO) for the Tauri part
ByteAtATime/flare#4 · by zamazan4ik was closed on Jul 14, 2025 · Updated on Jul 14, 2025
💬 1

Enable Link-Time Optimization (LTO) for the Tauri part
CyberTimon/RapidRAW#47 · by zamazan4ik was closed on Jul 10, 2025 · Updated on Jul 10, 2025
💬 2

Enable Link-Time Optimization (LTO) for the Tauri part
OmniKee/OmniKee#1 · by zamazan4ik was closed on May 2, 2025 · Updated on May 2, 2025
💬 1

Enable Link-Time Optimization (LTO) for the Tauri part
frstycodes/sendit#2 · by zamazan4ik was closed on May 2, 2025 · Updated on May 2, 2025
💬 1



Created by me

is:issue author:@me sort:updated-desc "Use recommended by Ratatui docs optimization options for Release builds"

5 results

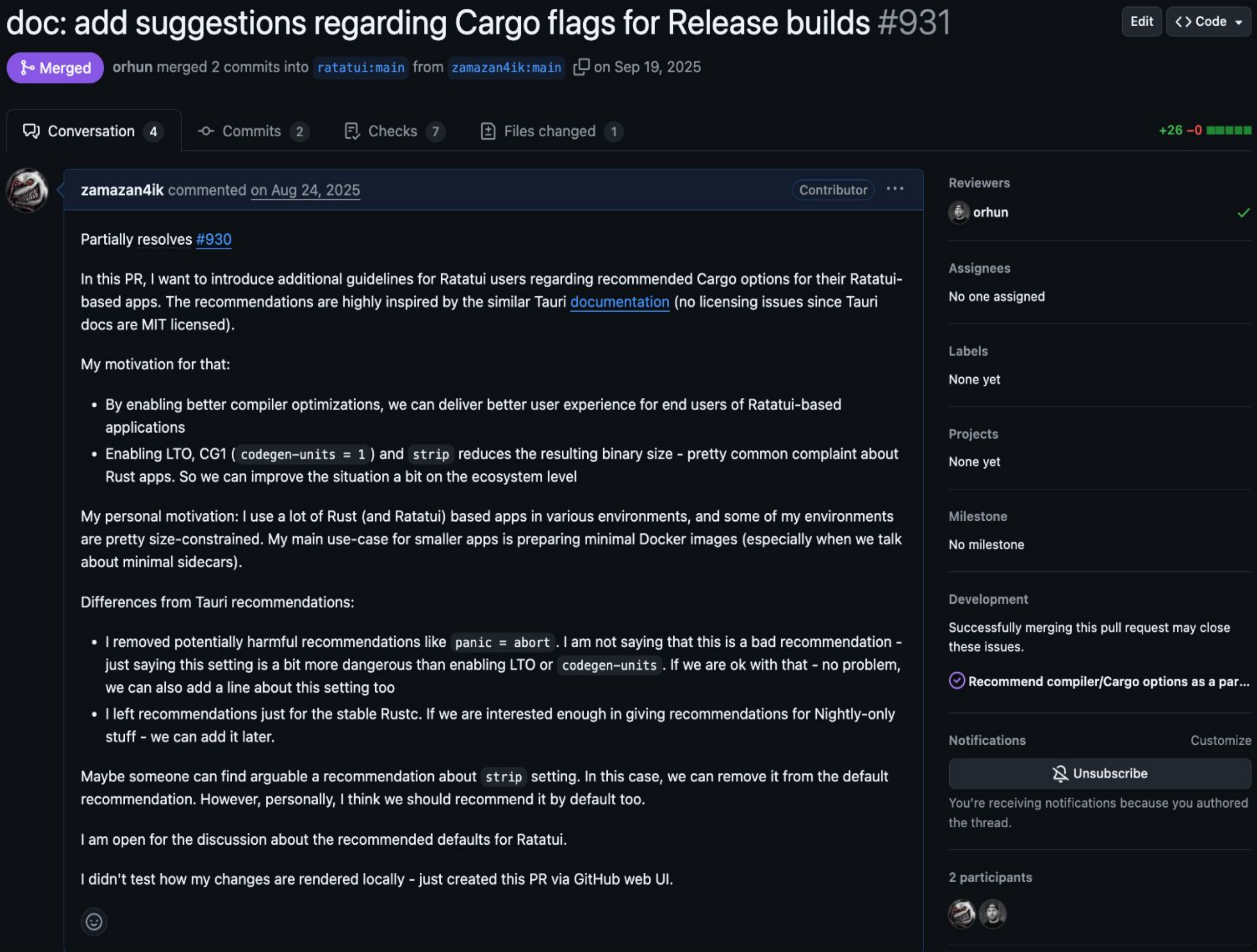
Use recommended by Ratatui docs optimization options for Release builds
Bengerthelorf/bcmr#1 · by zamazan4ik was closed 3 weeks ago · Updated 3 weeks ago
1 3

Use recommended by Ratatui docs optimization options for Release builds
harsh-vardhhan/option-analysis#6 · by zamazan4ik was closed 3 weeks ago · Updated 3 weeks ago
1

Use recommended by Ratatui docs optimization options for Release builds enhancement
huseyinbabal/tgcp#13 · by zamazan4ik was closed 3 weeks ago · Updated 3 weeks ago
1

Use recommended by Ratatui docs optimization options for Release builds
pawurb/hotpath-rs#101 · by zamazan4ik was closed on Dec 20, 2025 · Updated on Dec 20, 2025
3

Use recommended by Ratatui docs optimization options for Release builds
LargeModGames/spotatui#5 · by zamazan4ik was closed on Dec 9, 2025 · Updated on Dec 9, 2025
1



doc: add suggestions regarding Cargo flags for Release builds #931

Merged orhun merged 2 commits into `ratatui:main` from `zamazan4ik:main` on Sep 19, 2025

zamazan4ik commented on Aug 24, 2025

Partially resolves #930

In this PR, I want to introduce additional guidelines for Ratatui users regarding recommended Cargo options for their Ratatui-based apps. The recommendations are highly inspired by the similar Tauri documentation (no licensing issues since Tauri docs are MIT licensed).

My motivation for that:

- By enabling better compiler optimizations, we can deliver better user experience for end users of Ratatui-based applications
- Enabling LTO, CG1 (`codegen-units = 1`) and `strip` reduces the resulting binary size - pretty common complaint about Rust apps. So we can improve the situation a bit on the ecosystem level

My personal motivation: I use a lot of Rust (and Ratatui) based apps in various environments, and some of my environments are pretty size-constrained. My main use-case for smaller apps is preparing minimal Docker images (especially when we talk about minimal sidecars).

Differences from Tauri recommendations:

- I removed potentially harmful recommendations like `panic = abort`. I am not saying that this is a bad recommendation - just saying this setting is a bit more dangerous than enabling LTO or `codegen-units`. If we are ok with that - no problem, we can also add a line about this setting too
- I left recommendations just for the stable Rustc. If we are interested enough in giving recommendations for Nightly-only stuff - we can add it later.

Maybe someone can find arguable a recommendation about `strip` setting. In this case, we can remove it from the default recommendation. However, personally, I think we should recommend it by default too.

I am open for the discussion about the recommended defaults for Ratatui.

I didn't test how my changes are rendered locally - just created this PR via GitHub web UI.

Reviewers

orhun ✓

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

Successfully merging this pull request may close these issues.

Recommend compiler/Cargo options as a par...

Notifications

You're receiving notifications because you authored the thread.

2 participants

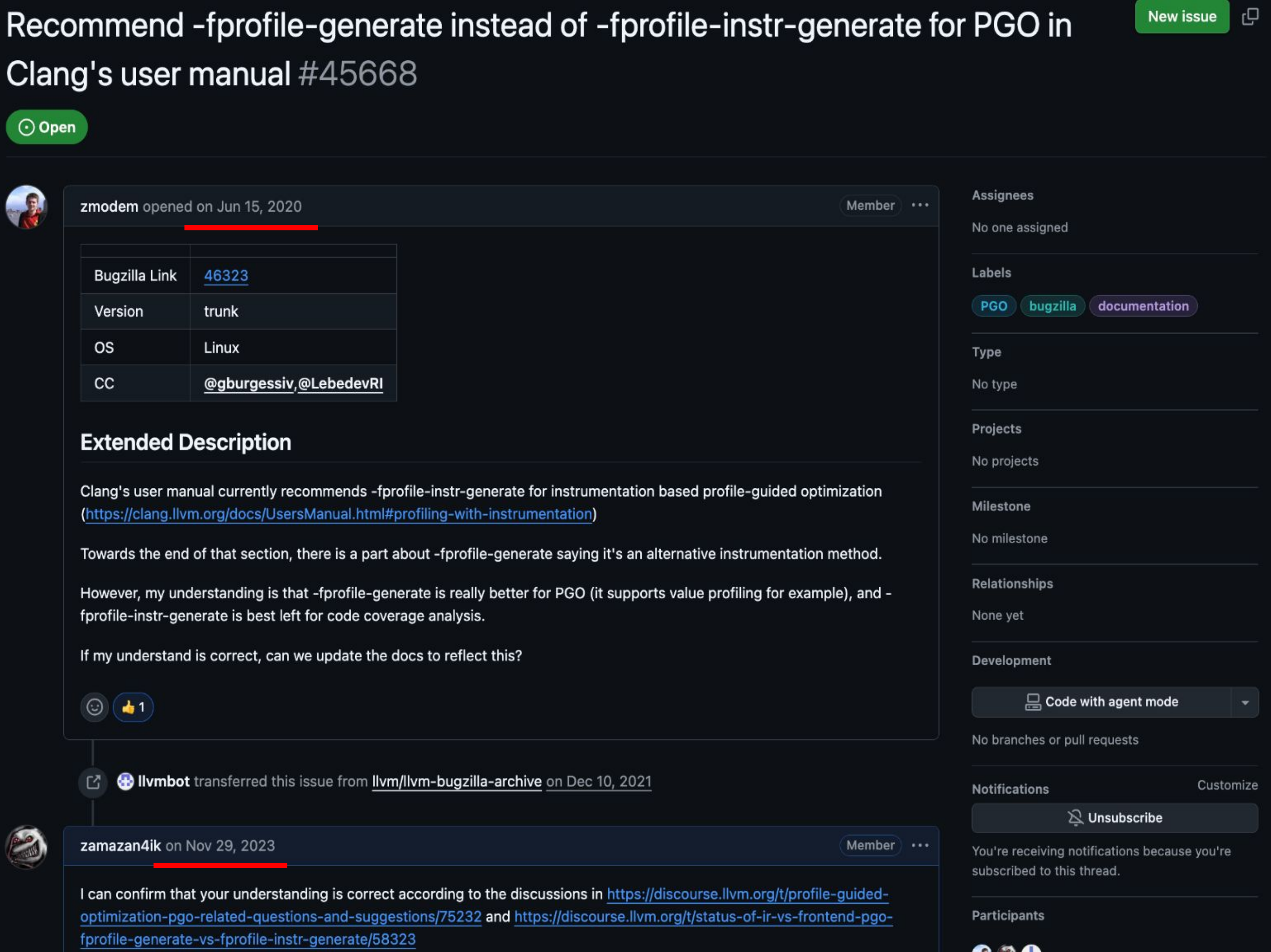
And even if they read documentation…

- Documentation doesn’t have **really important** details
    - That’s how “Awesome PGO” was created

Just read Clang manual
and use PGO
not finished after 3 years

And even if they read documentation…

- Documentation is outdated
- Documentation is outdated and you cannot figure it out
  - E.g. previous Clang FE PGO vs IR PGO recommendations



Recommend -fprofile-generate instead of -fprofile-instr-generate for PGO in Clang's user manual #45668

zmodem opened on Jun 15, 2020 · Member

Bugzilla Link	46323
Version	trunk
OS	Linux
CC	@gburgessiv,@LebedevRI

Extended Description

Clang's user manual currently recommends -fprofile-instr-generate for instrumentation based profile-guided optimization (https://clang.llvm.org/docs/UsersManual.html#profiling-with-instrumentation)

Towards the end of that section, there is a part about -fprofile-generate saying it's an alternative instrumentation method.

However, my understanding is that -fprofile-generate is really better for PGO (it supports value profiling for example), and -fprofile-instr-generate is best left for code coverage analysis.

If my understand is correct, can we update the docs to reflect this?

👍 1

llvmbot transferred this issue from llvm/llvm-bugzilla-archive on Dec 10, 2021

zamazan4ik on Nov 29, 2023 · Member

I can confirm that your understanding is correct according to the discussions in https://discourse.llvm.org/t/profile-guided-optimization-pgo-related-questions-and-suggestions/75232 and https://discourse.llvm.org/t/status-of-ir-vs-frontend-pgo-fprofile-generate-vs-fprofile-instr-generate/58323

Assignees

No one assigned

Labels

PGO bugzilla documentation

Type

No type

Projects

No projects

Milestone

No milestone

Relationships

None yet

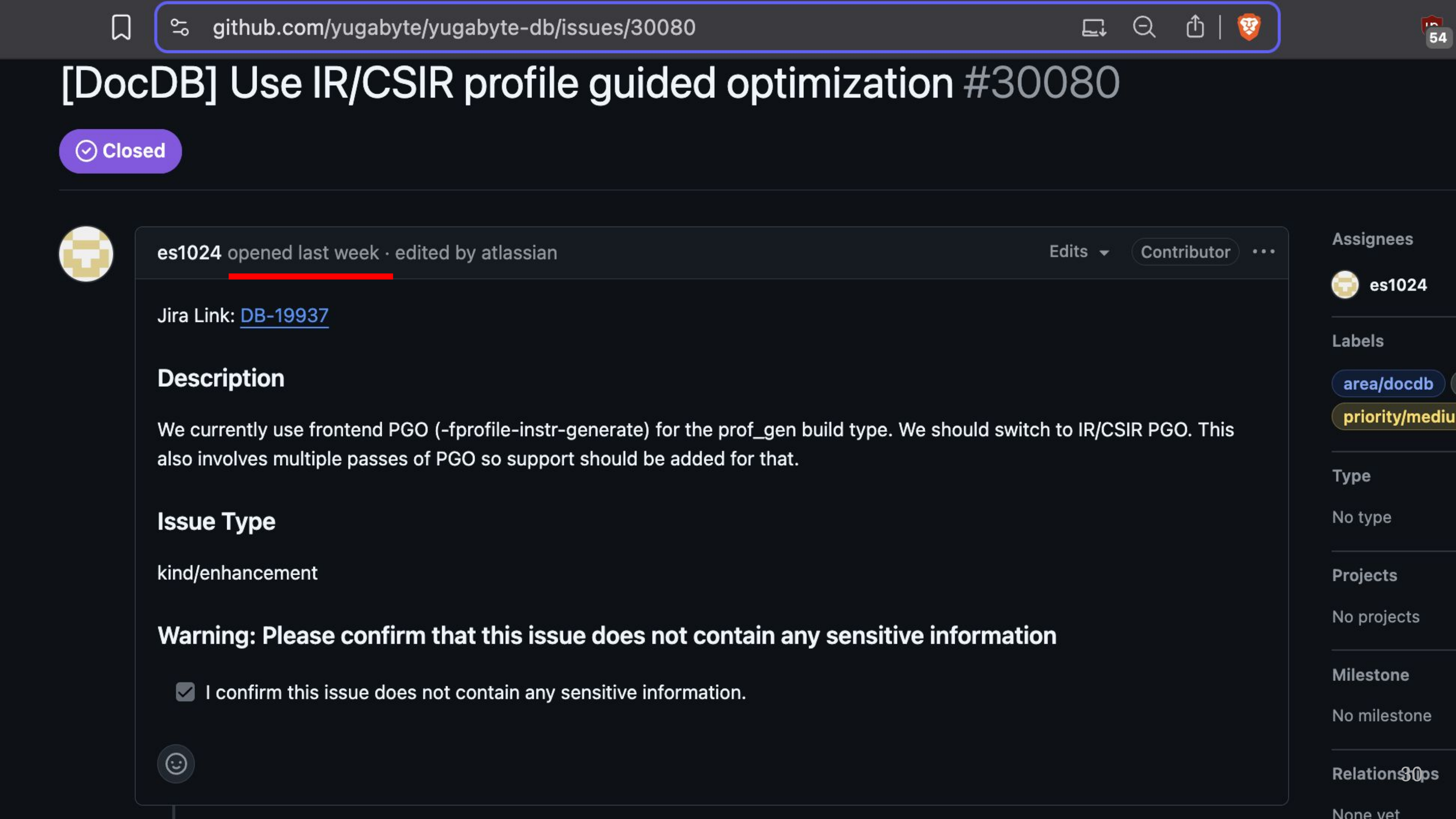
Development

No branches or pull requests

Notifications

You're receiving notifications because you're subscribed to this thread.

Participants



[DocDB] Use IR/CSIR profile guided optimization #30080

es1024 opened last week · edited by atlassian

Jira Link: DB-19937

Description

We currently use frontend PGO (-fprofile-instr-generate) for the prof_gen build type. We should switch to IR/CSIR PGO. This also involves multiple passes of PGO so support should be added for that.

Issue Type

kind/enhancement

Warning: Please confirm that this issue does not contain any sensitive information

☑ I confirm this issue does not contain any sensitive information.

Assignees

es1024

Labels

area/docdb
priority/mediu

Type

No type

Projects

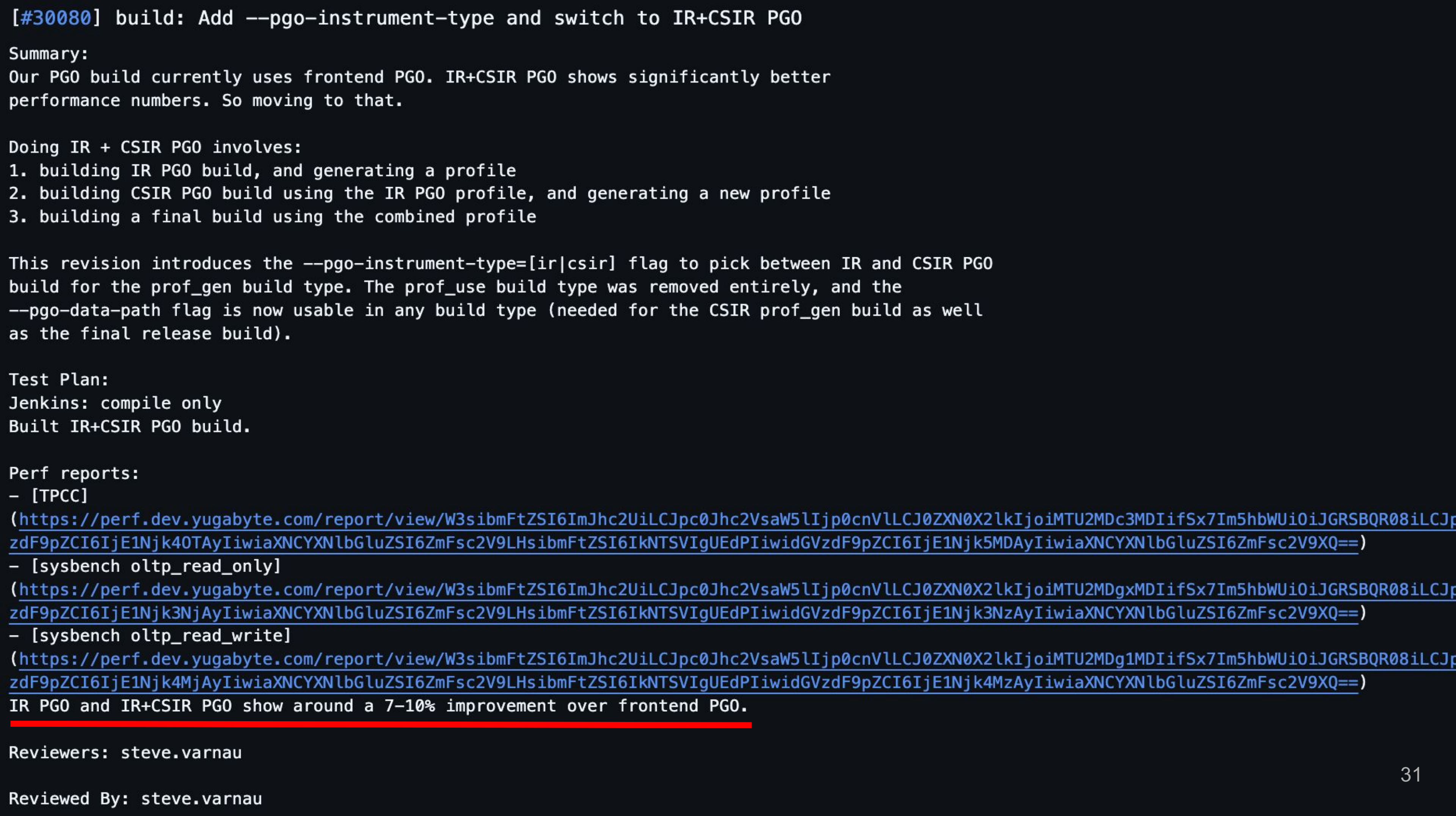
No projects

Milestone

No milestone

Relationships

None yet



```
[#30080] build: Add --pgo-instrument-type and switch to IR+CSIR PGO

Summary:
Our PGO build currently uses frontend PGO. IR+CSIR PGO shows significantly better
performance numbers. So moving to that.

Doing IR + CSIR PGO involves:
1. building IR PGO build, and generating a profile
2. building CSIR PGO build using the IR PGO profile, and generating a new profile
3. building a final build using the combined profile

This revision introduces the --pgo-instrument-type=[ir|csir] flag to pick between IR and CSIR PGO
build for the prof_gen build type. The prof_use build type was removed entirely, and the
--pgo-data-path flag is now usable in any build type (needed for the CSIR prof_gen build as well
as the final release build).

Test Plan:
Jenkins: compile only
Built IR+CSIR PGO build.

Perf reports:
- [TPCC]
(https://perf.dev.yugabyte.com/report/view/W3sibmFtZSI6ImJhc2UiLCJpc0Jhc2VsaW5lIjp0cnVlLCJ0ZXN0X2lkIjoiMTU2MDc3MDIifSx7Im5hbWUiOiJGRSBQR08iLCJp
zdF9pZCI6IjE1Njk4OTAyIiwiaXNCYXNlbGluZSI6ZmFsc2V9LHsibmFtZSI6IkNTSVIgUEdPIiwidGVzdF9pZCI6IjE1Njk5MDAyIiwiaXNCYXNlbGluZSI6ZmFsc2V9XQ==)
- [sysbench oltp_read_only]
(https://perf.dev.yugabyte.com/report/view/W3sibmFtZSI6ImJhc2UiLCJpc0Jhc2VsaW5lIjp0cnVlLCJ0ZXN0X2lkIjoiMTU2MDgxMDIifSx7Im5hbWUiOiJGRSBQR08iLCJp
zdF9pZCI6IjE1Njk3NjAyIiwiaXNCYXNlbGluZSI6ZmFsc2V9LHsibmFtZSI6IkNTSVIgUEdPIiwidGVzdF9pZCI6IjE1Njk3NzAyIiwiaXNCYXNlbGluZSI6ZmFsc2V9XQ==)
- [sysbench oltp_read_write]
(https://perf.dev.yugabyte.com/report/view/W3sibmFtZSI6ImJhc2UiLCJpc0Jhc2VsaW5lIjp0cnVlLCJ0ZXN0X2lkIjoiMTU2MDg1MDIifSx7Im5hbWUiOiJGRSBQR08iLCJp
zdF9pZCI6IjE1Njk4MjAyIiwiaXNCYXNlbGluZSI6ZmFsc2V9LHsibmFtZSI6IkNTSVIgUEdPIiwidGVzdF9pZCI6IjE1Njk4MzAyIiwiaXNCYXNlbGluZSI6ZmFsc2V9XQ==)
IR PGO and IR+CSIR PGO show around a 7-10% improvement over frontend PGO.

Reviewers: steve.varnau

Reviewed By: steve.varnau
```

And even if they read documentation…

- Documentation is simply lying
  - “PGO doesn’t help SQLite” in official docs vs 10% performance improvement from PGO in the official SQLite benches in practice

Tooling

How tooling can help us to unlock performance?

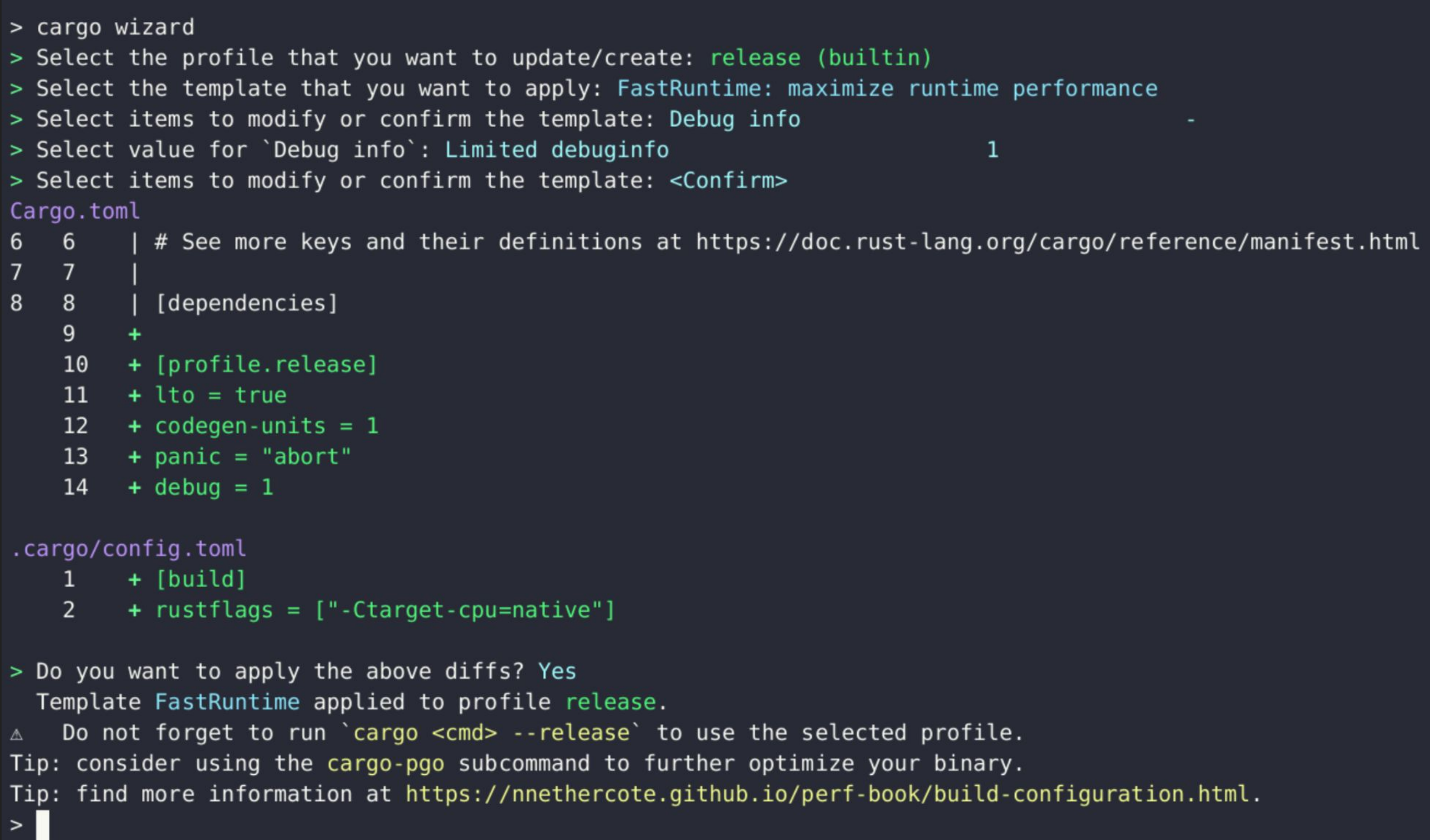
- Automate “best practice” integration into applications
- Convenient benchmarking
- Automate optimization routines
- Profilers with good vizualizations

Example: creating a new application from a template

- We can try to integrate recommended optimizations into templates, and create new apps from these templates
- Some examples:
  - Ratatui: templates via cargo-generate (merged!)
  - Tauri: create-tauri-app (open issue)
  - Dioxus: dioxus-template via the dx tool (open issue)
- However, this way still has issues:
  - People don’t use project generators for various reasons (copy&paste, lack of knowledge, LLM)
  - Already created applications **are not covered** anyway (e.g. Ratatui case)

Example: Benchmarking in Rust vs C++

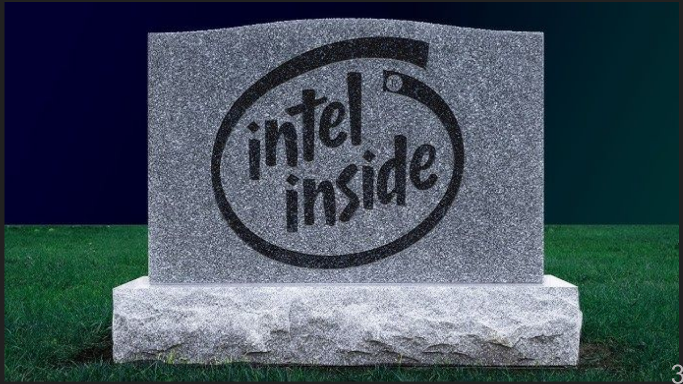
- Benchmarking is **needed** to measure performance improvements
- Good example: cargo bench in Rust
- Bad example: <blank> in C++
  - What to use by default? Probably Google Benchmark…
  - How to install Google Benchmark? Conan, Vcpkg, system deps…
  - How to run the benchmark? Check build scripts / README / whatever



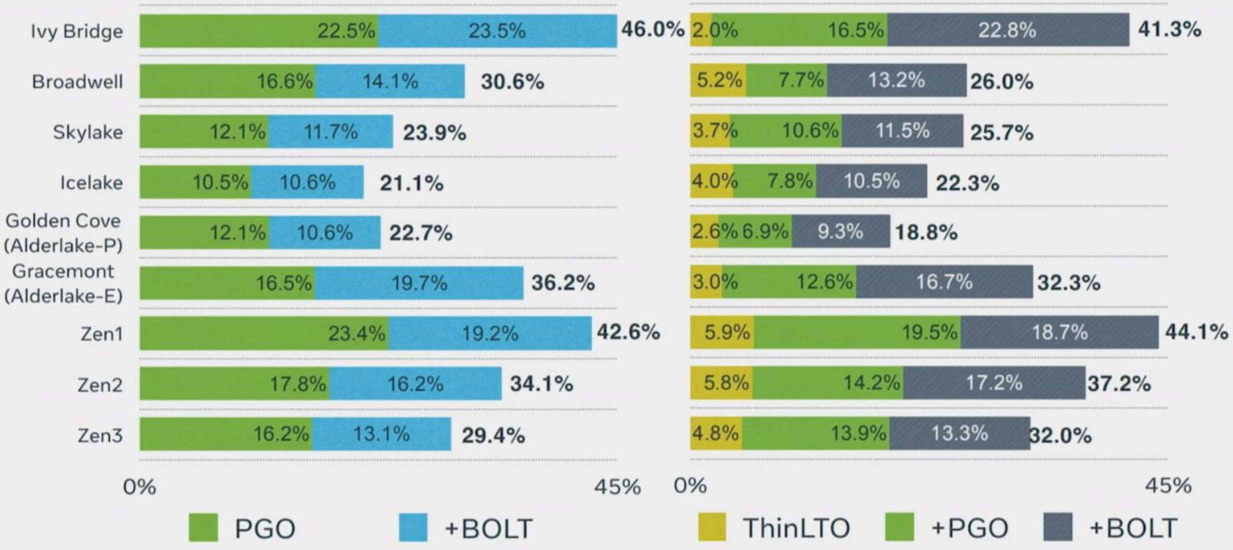
```
> cargo wizard
> Select the profile that you want to update/create: release (builtin)
> Select the template that you want to apply: FastRuntime: maximize runtime performance
> Select items to modify or confirm the template: Debug info                          -
> Select value for `Debug info`: Limited debuginfo                       1
> Select items to modify or confirm the template: <Confirm>
Cargo.toml
6   6    | # See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html
7   7    |
8   8    | [dependencies]
    9    +
    10   + [profile.release]
    11   + lto = true
    12   + codegen-units = 1
    13   + panic = "abort"
    14   + debug = 1

.cargo/config.toml
    1    + [build]
    2    + rustflags = ["-Ctarget-cpu=native"]

> Do you want to apply the above diffs? Yes
  Template FastRuntime applied to profile release.
⚠  Do not forget to run `cargo <cmd> --release` to use the selected profile.
Tip: consider using the cargo-pgo subcommand to further optimize your binary.
Tip: find more information at https://nnethercote.github.io/perf-book/build-configuration.html.
> 
```

Example: Post-Link Optimization (PLO)

- TL;DR: PLO = PGO on steroids
- Enables even more optimizations like **function reorder**, **hot/cold split**, etc.
- Available open-source tools at the moment:
    - LLVM BOLT - the most popular PLO tool nowadays
    - Google Propeller
    - Intel Thin Layout Optimizer (TLO) - RIP



Cumulative speedup over bootstrapped build, Building Clang



20 22

LLVM

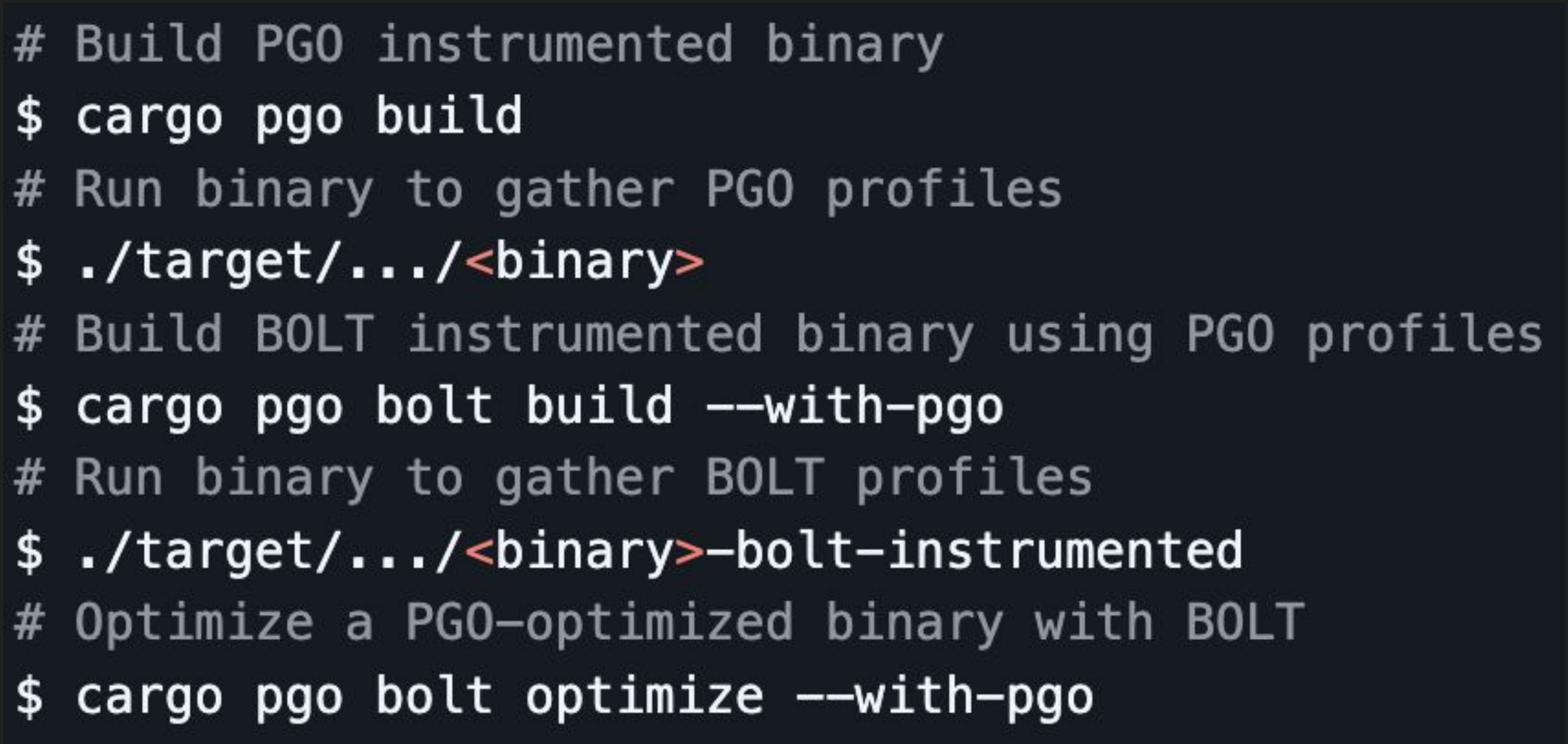
DEVELOPERS' MEETING

SAN JOSE, CALIFORNIA • HAYES MANSION

Optimizing Clang with BOLT using CMake

Amir Ayupov

Example: PGO + PLO with cargo-pgo in Rust



```
# Build PGO instrumented binary
$ cargo pgo build
# Run binary to gather PGO profiles
$ ./target/.../<binary>
# Build BOLT instrumented binary using PGO profiles
$ cargo pgo bolt build --with-pgo
# Run binary to gather BOLT profiles
$ ./target/.../<binary>-bolt-instrumented
# Optimize a PGO-optimized binary with BOLT
$ cargo pgo bolt optimize --with-pgo
```

Example: PGO + PLO manually in C++

1. Pass proper PGO compiler flags into your build scripts
2. Build
3. Run
4. Collect PGO profiles
5. Convert them in a proper format
6. Recompile your app once again with a different PGO flag
7. Figure out how to run LLVM BOLT
8. Instrument your binary from the the step 6 with BOLT
9. Run the new binary once again
10. Optimize binary with a different BOLT command

Issues with extra tooling

- People don’t know about these tools
- Tools are not easy to install:
    - No prebuilt package in a repository (LLVM BOLT case)
    - No easy way to build tool on your own (Google AutoFDO, Google Propeller, BOLT before LLVM)
- All the same issues with documentation :)

Defaults

Why defaults are so important?

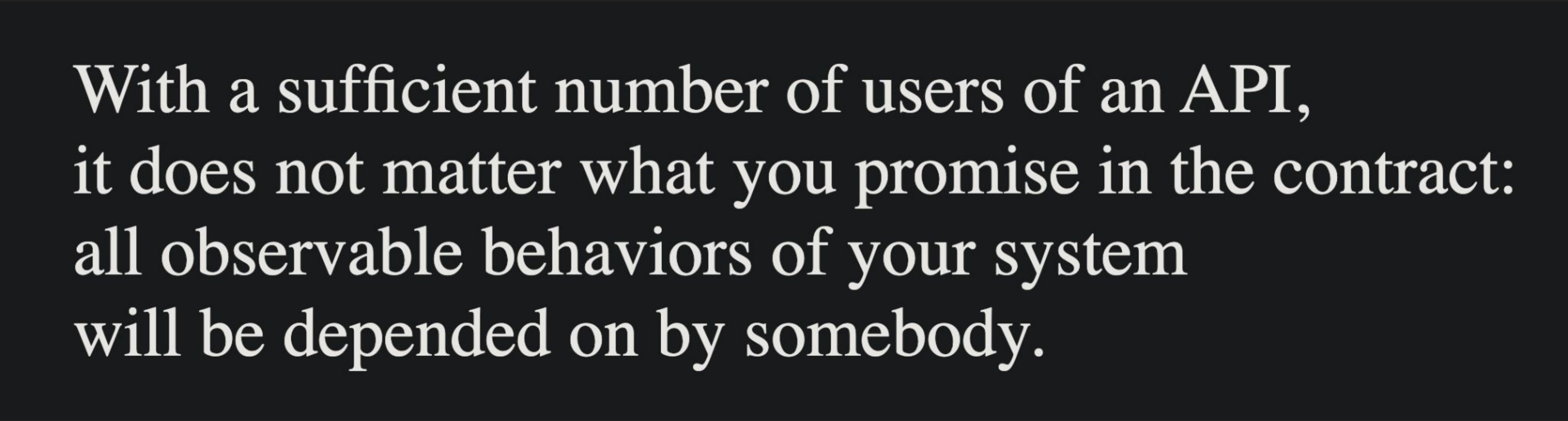
Defaults will be used by most users.

↓

Defaults will be the most impactful on the ecosystem.

Changing defaults is **hard**!

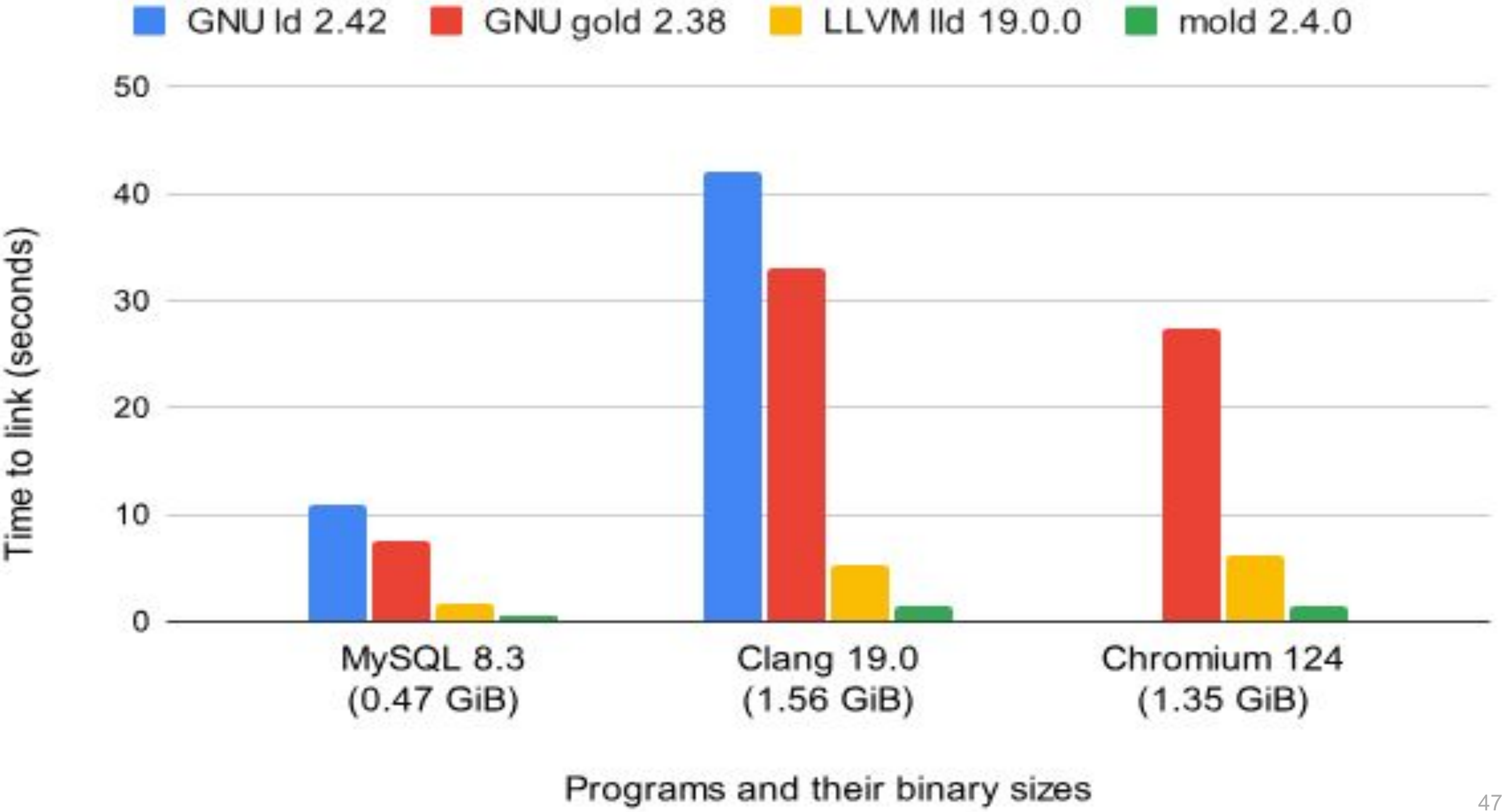
According to the Hyrum's law:



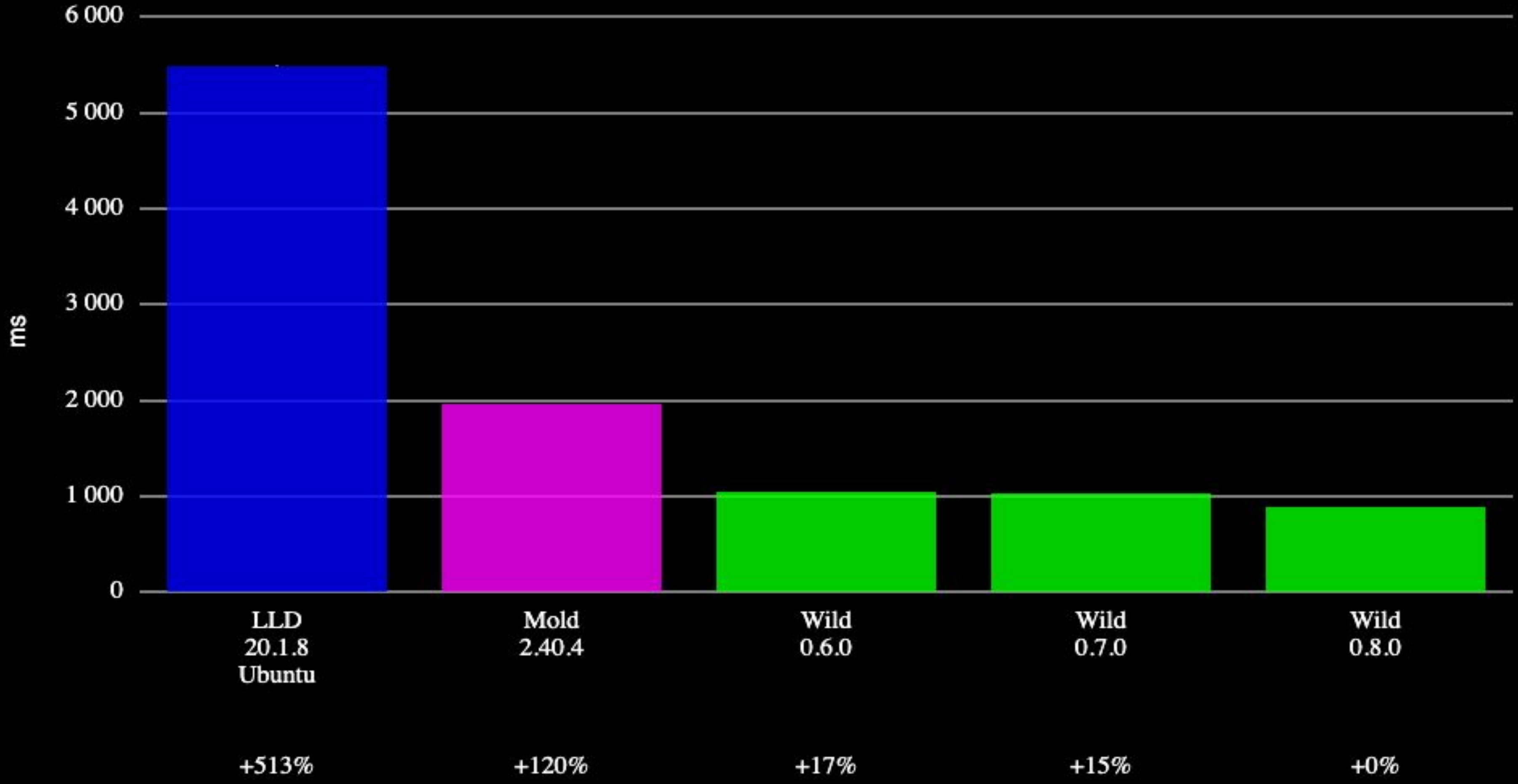
> With a sufficient number of users of an API,
> it does not matter what you promise in the contract:
> all observable behaviors of your system
> will be depended on by somebody.

Example: the Rust ecosystem - good examples

- Using LLD linker on Linux by default


GNU ld 2.42
GNU gold 2.38
LLVM lld 19.0.0
mold 2.4.0
50
40
30
20
10
0
Time to link (seconds)
MySQL 8.3
(0.47 GiB)
Clang 19.0
(1.56 GiB)
Chromium 124
(1.35 GiB)
Programs and their binary sizes

ryzen-9955hx - chrome - time



Example: the Rust ecosystem - good examples

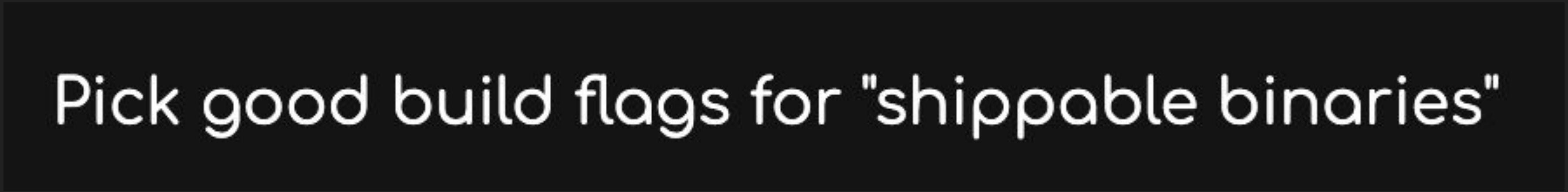
- Using LLD linker on Linux by default
- Building Rustc and Rust Analyzer with LTO/PGO/PLO by default
- Using Jemalloc for Clippy
- strip = “debuginfo” in Release by default

Example: **cargo install** in Rust

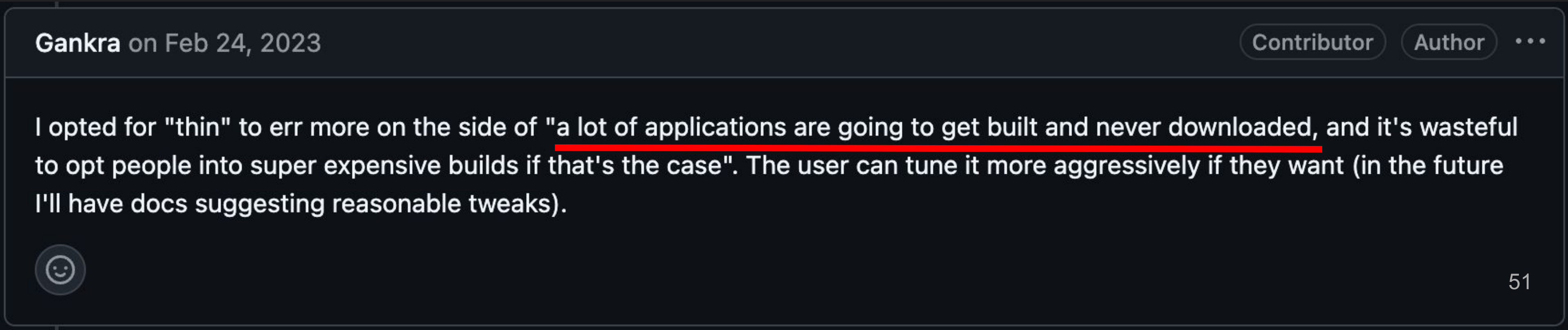
- De facto a default way to install Rust-based binaries
- Gentoo-style: compiling a binary on a target machine
- With the same tradeoffs:
  - Pros: you can optimize for your hardware (like CPU-specific instruction set)
  - Cons: you have more limitations for “expensive” optimizations like LTO
- Prebuilt binaries exist - cargo binstall - but are not that popular

Example: dist (aka cargo-dist)

- Probably the most popular tool for preparing Release binaries in Rust

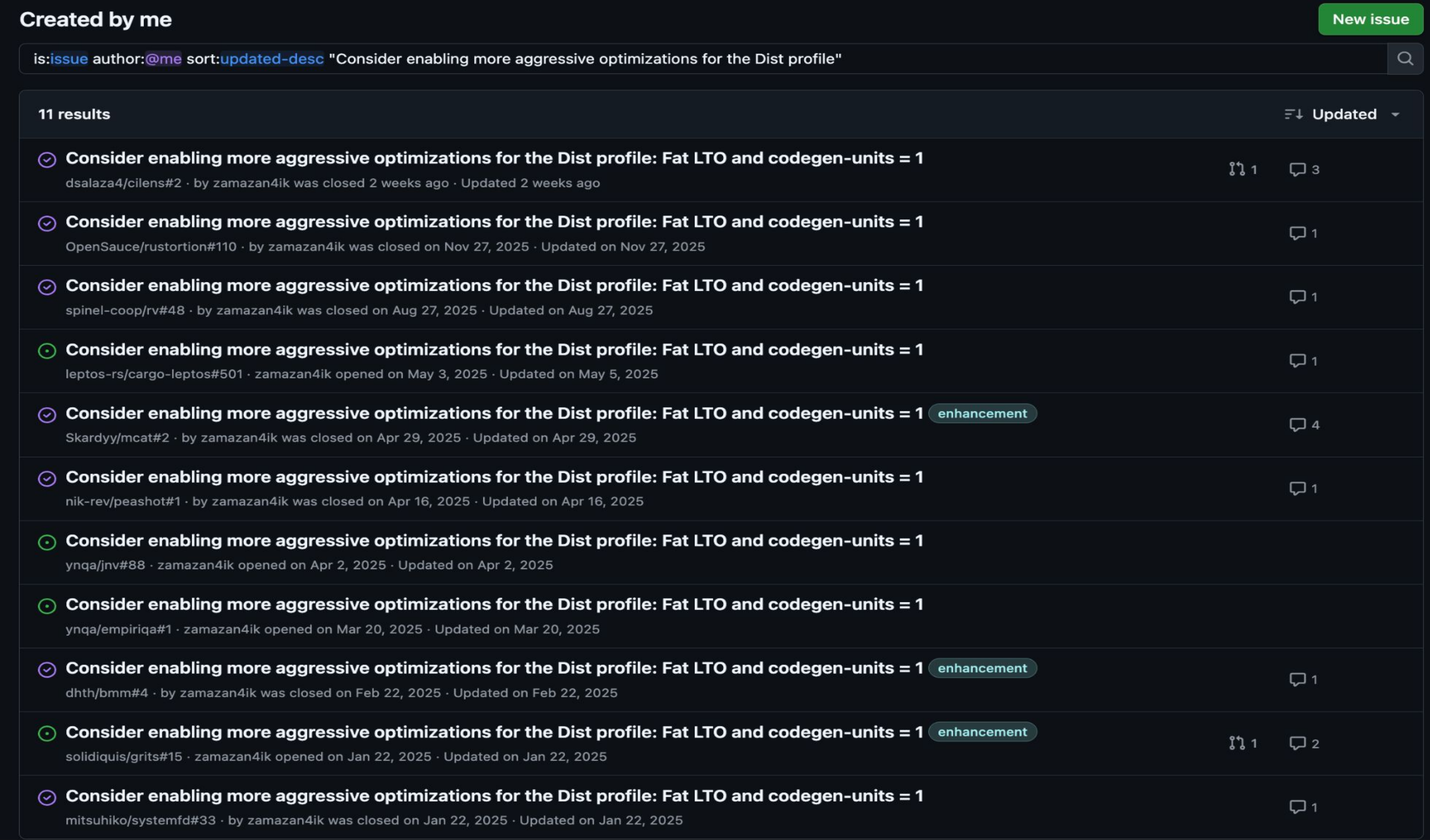


Pick good build flags for "shippable binaries"



Gankra on Feb 24, 2023

I opted for "thin" to err more on the side of "a lot of applications are going to get built and never downloaded, and it's wasteful to opt people into super expensive builds if that's the case". The user can tune it more aggressively if they want (in the future I'll have docs suggesting reasonable tweaks).



```
is:issue author:@me sort:updated-desc "Consider enabling more aggressive optimizations for the Dist profile"
```

11 results

✓ **Consider enabling more aggressive optimizations for the Dist profile: Fat LTO and codegen-units = 1** 1 3
dsalaza4/cilens#2 · by zamazan4ik was closed 2 weeks ago · Updated 2 weeks ago

✓ **Consider enabling more aggressive optimizations for the Dist profile: Fat LTO and codegen-units = 1** 1
OpenSauce/rustortion#110 · by zamazan4ik was closed on Nov 27, 2025 · Updated on Nov 27, 2025

✓ **Consider enabling more aggressive optimizations for the Dist profile: Fat LTO and codegen-units = 1** 1
spinel-coop/rv#48 · by zamazan4ik was closed on Aug 27, 2025 · Updated on Aug 27, 2025

○ **Consider enabling more aggressive optimizations for the Dist profile: Fat LTO and codegen-units = 1** 1
leptos-rs/cargo-leptos#501 · zamazan4ik opened on May 3, 2025 · Updated on May 5, 2025

✓ **Consider enabling more aggressive optimizations for the Dist profile: Fat LTO and codegen-units = 1** enhancement 4
Skardyy/mcat#2 · by zamazan4ik was closed on Apr 29, 2025 · Updated on Apr 29, 2025

✓ **Consider enabling more aggressive optimizations for the Dist profile: Fat LTO and codegen-units = 1** 1
nik-rev/peashot#1 · by zamazan4ik was closed on Apr 16, 2025 · Updated on Apr 16, 2025

○ **Consider enabling more aggressive optimizations for the Dist profile: Fat LTO and codegen-units = 1**
ynqa/jnv#88 · zamazan4ik opened on Apr 2, 2025 · Updated on Apr 2, 2025

○ **Consider enabling more aggressive optimizations for the Dist profile: Fat LTO and codegen-units = 1**
ynqa/empiriqa#1 · zamazan4ik opened on Mar 20, 2025 · Updated on Mar 20, 2025

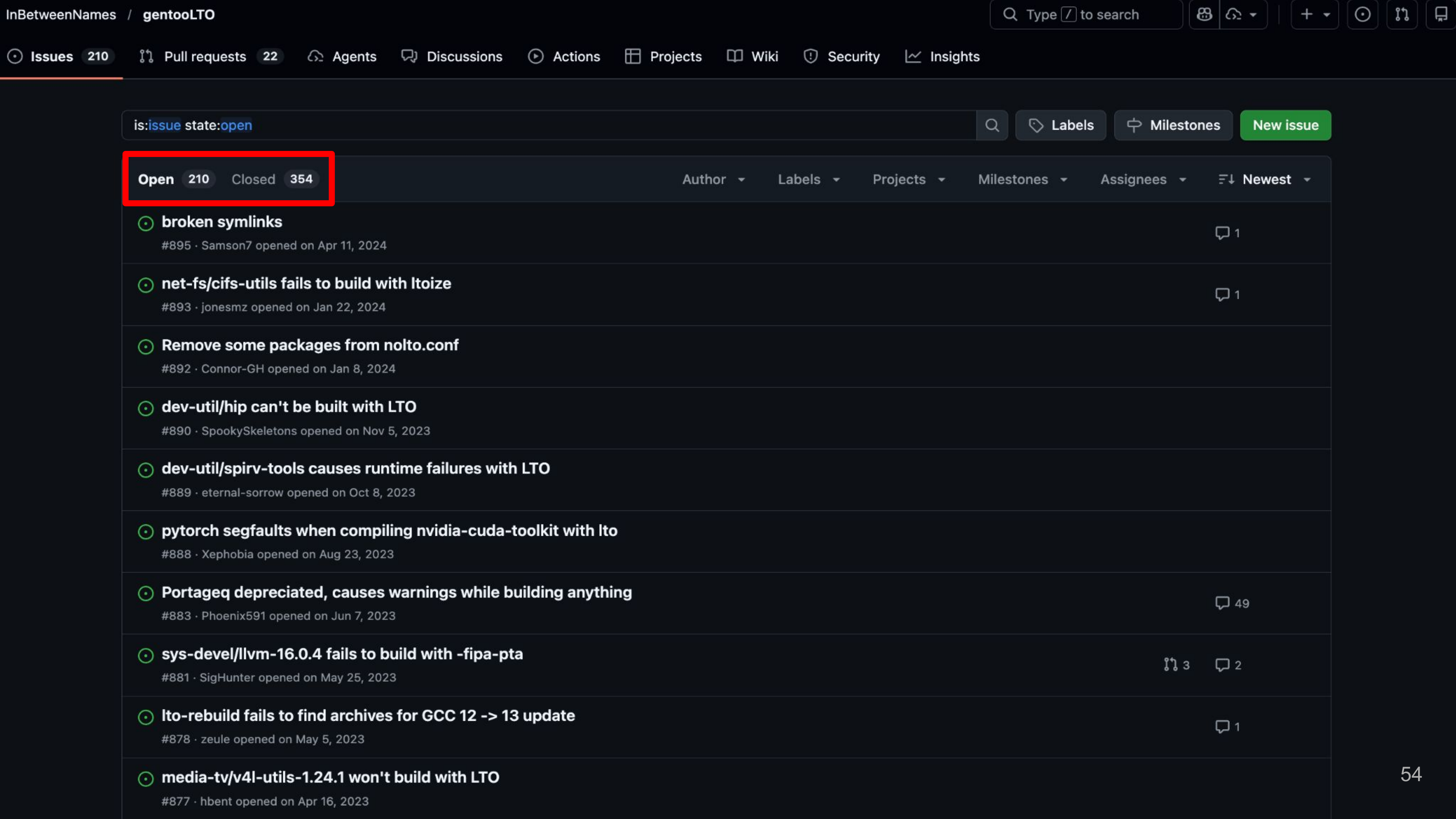
✓ **Consider enabling more aggressive optimizations for the Dist profile: Fat LTO and codegen-units = 1** enhancement 1
dhth/bmm#4 · by zamazan4ik was closed on Feb 22, 2025 · Updated on Feb 22, 2025

○ **Consider enabling more aggressive optimizations for the Dist profile: Fat LTO and codegen-units = 1** enhancement 1 2
solidiquis/grits#15 · zamazan4ik opened on Jan 22, 2025 · Updated on Jan 22, 2025

✓ **Consider enabling more aggressive optimizations for the Dist profile: Fat LTO and codegen-units = 1** 1
mitsuhiko/systemfd#33 · by zamazan4ik was closed on Jan 22, 2025 · Updated on Jan 22, 2025

Example: an idea about LTO for Rust **by default**

- Bring the LTO benefits for the whole Rust ecosystem at once
  - Luckily, LTO in Rust is much safer to enable compared to C++



Open 210 Closed 354

broken symlinks
#895 · Samson7 opened on Apr 11, 2024 — 1

net-fs/cifs-utils fails to build with ltoize
#893 · jonesmz opened on Jan 22, 2024 — 1

Remove some packages from nolto.conf
#892 · Connor-GH opened on Jan 8, 2024

dev-util/hip can't be built with LTO
#890 · SpookySkeletons opened on Nov 5, 2023

dev-util/spirv-tools causes runtime failures with LTO
#889 · eternal-sorrow opened on Oct 8, 2023

pytorch segfaults when compiling nvidia-cuda-toolkit with lto
#888 · Xephobia opened on Aug 23, 2023

Portageq depreciated, causes warnings while building anything
#883 · Phoenix591 opened on Jun 7, 2023 — 49

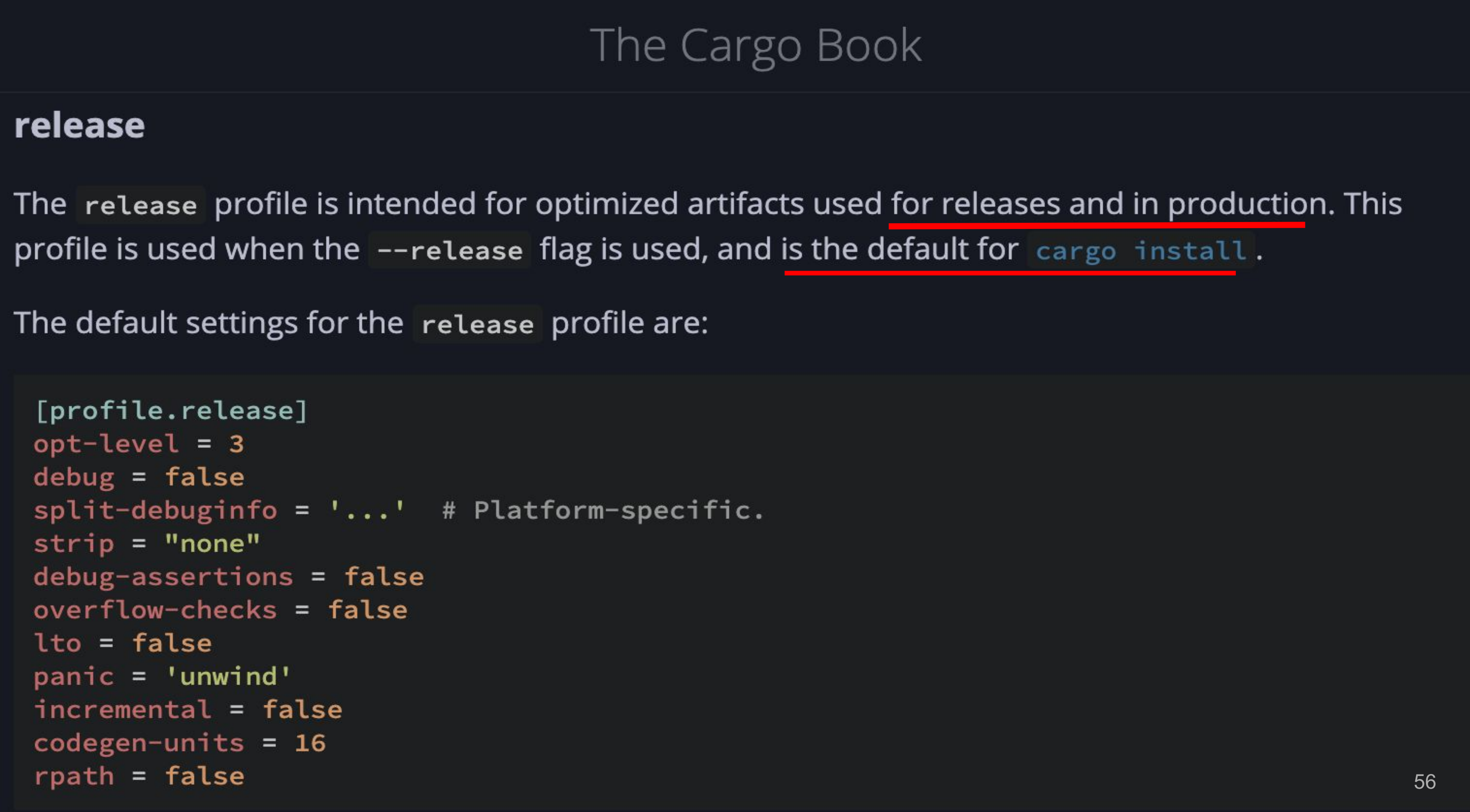
sys-devel/llvm-16.0.4 fails to build with -fipa-pta
#881 · SigHunter opened on May 25, 2023 — 3 — 2

lto-rebuild fails to find archives for GCC 12 -> 13 update
#878 · zeule opened on May 5, 2023 — 1

media-tv/v4l-utils-1.24.1 won't build with LTO
#877 · hbent opened on Apr 16, 2023

Example: an idea about LTO for Rust **by default**

- Not only for Rust but for dependent ecosystems too like Rust-based Python packages with maturin (and other deps)
- There are multiple issues here:
  - People use Relese profile during the development phase



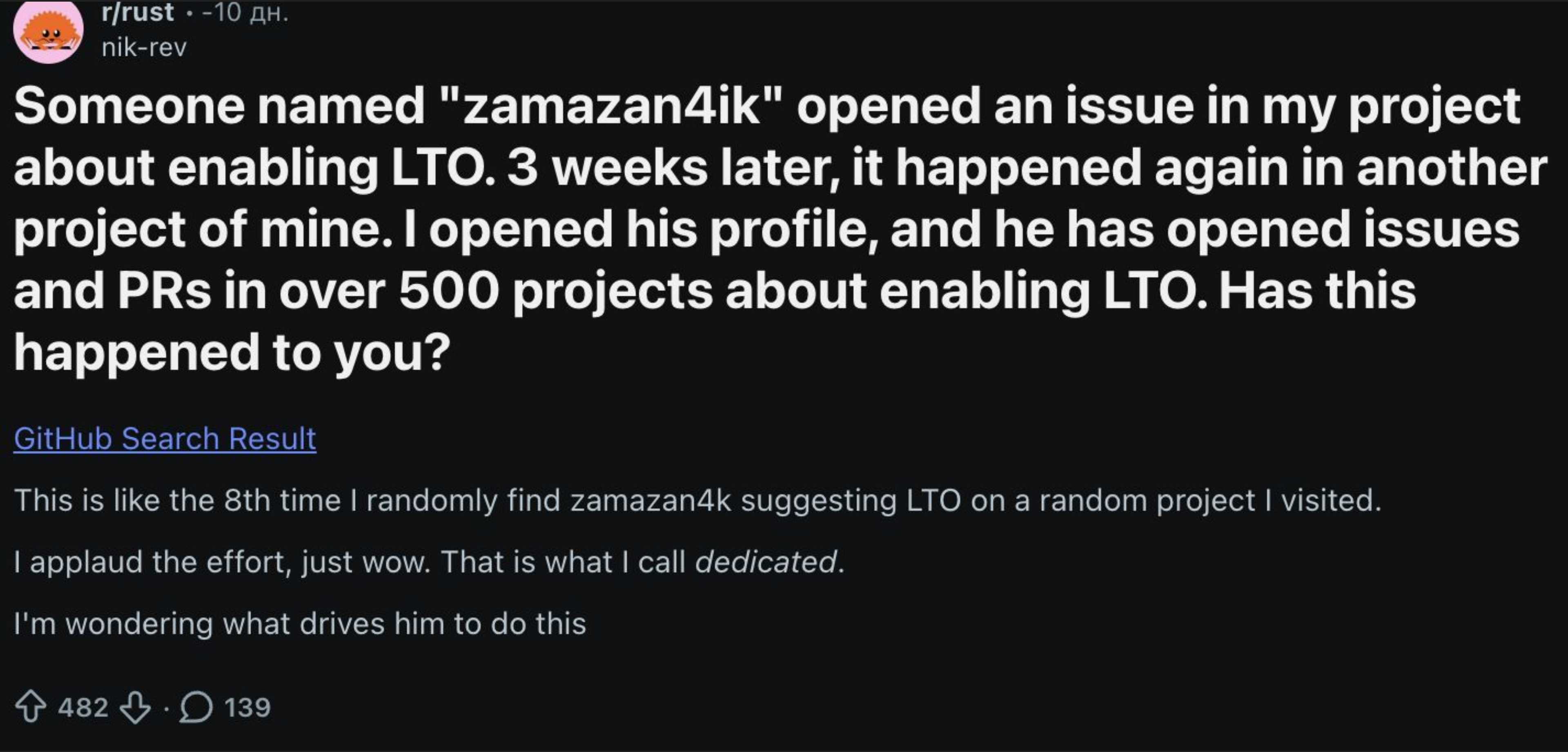
The Cargo Book

release

The `release` profile is intended for optimized artifacts used for releases and in production. This profile is used when the `--release` flag is used, and is the default for `cargo install`.

The default settings for the `release` profile are:

```toml
[profile.release]
opt-level = 3
debug = false
split-debuginfo = '...'  # Platform-specific.
strip = "none"
debug-assertions = false
overflow-checks = false
lto = false
panic = 'unwind'
incremental = false
codegen-units = 16
rpath = false
```



r/rust • -10 дн.
nik-rev

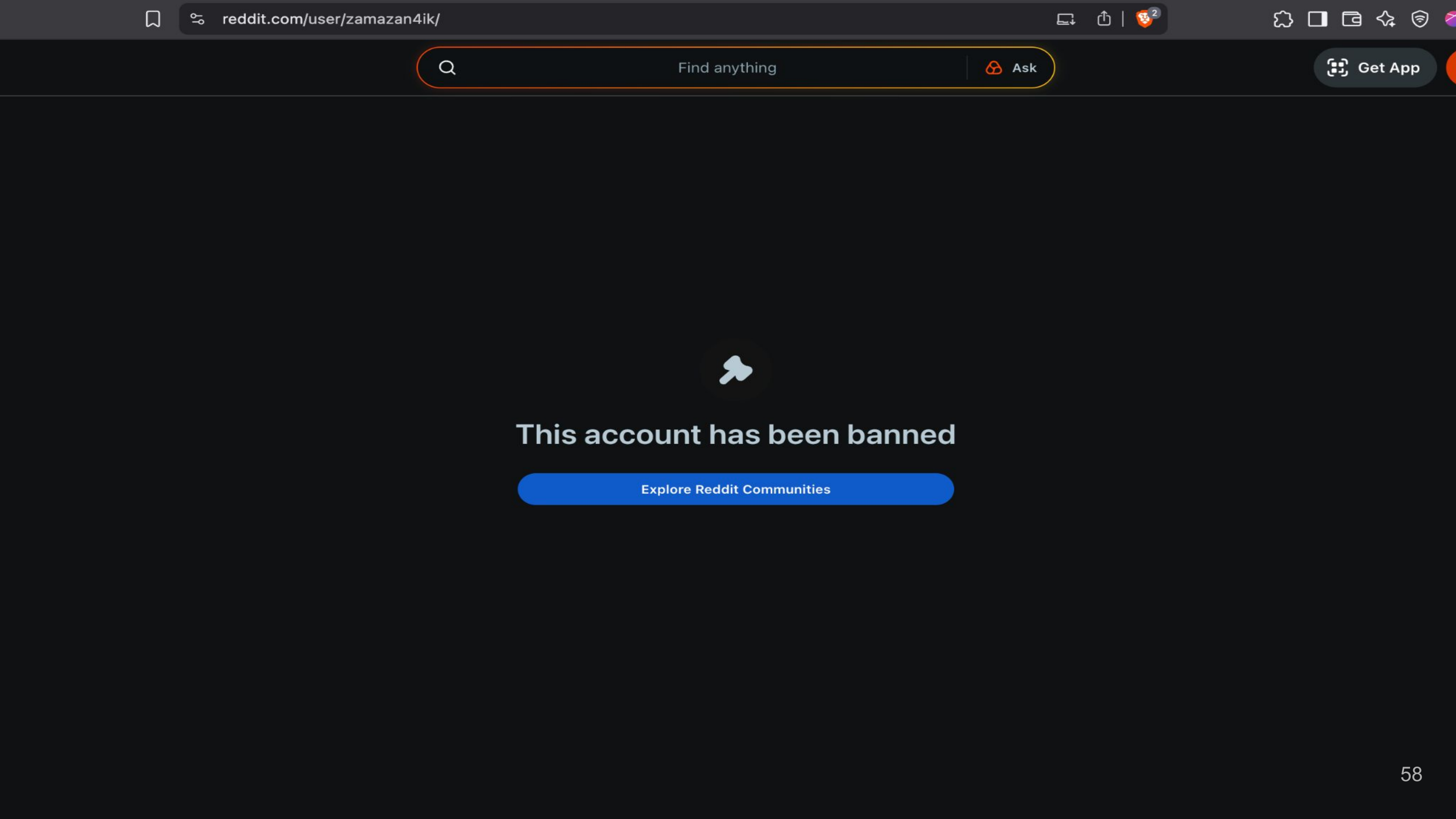
Someone named "zamazan4ik" opened an issue in my project about enabling LTO. 3 weeks later, it happened again in another project of mine. I opened his profile, and he has opened issues and PRs in over 500 projects about enabling LTO. Has this happened to you?

GitHub Search Result

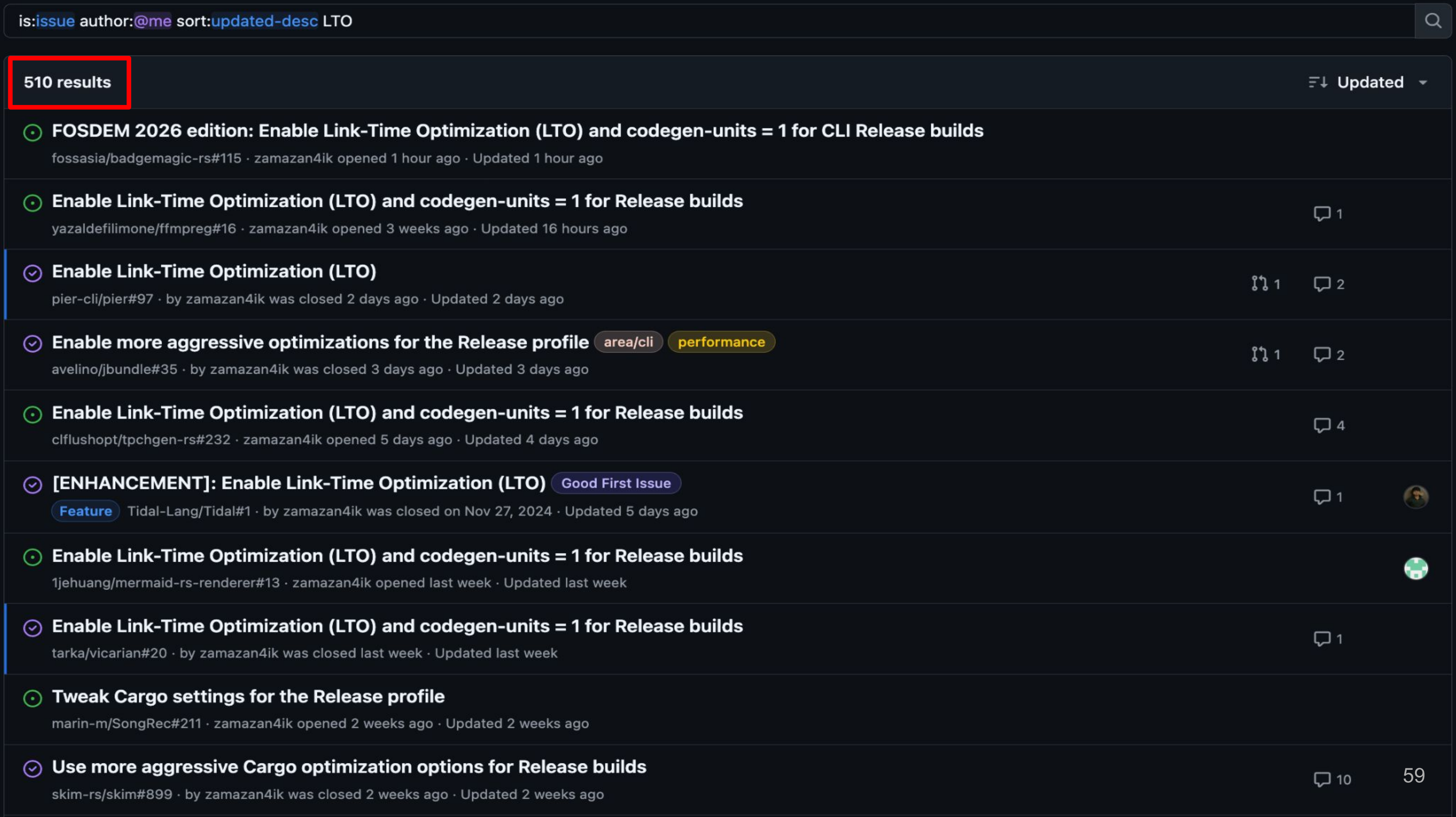
This is like the 8th time I randomly find zamazan4k suggesting LTO on a random project I visited.

I applaud the effort, just wow. That is what I call *dedicated*.

I'm wondering what drives him to do this



This account has been banned



is:issue author:@me sort:updated-desc LTO

510 results

Updated

FOSDEM 2026 edition: Enable Link-Time Optimization (LTO) and codegen-units = 1 for CLI Release builds
fossasia/badgemagic-rs#115 · zamazan4ik opened 1 hour ago · Updated 1 hour ago

Enable Link-Time Optimization (LTO) and codegen-units = 1 for Release builds
yazaldefilimone/ffmpreg#16 · zamazan4ik opened 3 weeks ago · Updated 16 hours ago
1

Enable Link-Time Optimization (LTO)
pier-cli/pier#97 · by zamazan4ik was closed 2 days ago · Updated 2 days ago
1 2

Enable more aggressive optimizations for the Release profile area/cli performance
avelino/jbundle#35 · by zamazan4ik was closed 3 days ago · Updated 3 days ago
1 2

Enable Link-Time Optimization (LTO) and codegen-units = 1 for Release builds
clflushopt/tpchgen-rs#232 · zamazan4ik opened 5 days ago · Updated 4 days ago
4

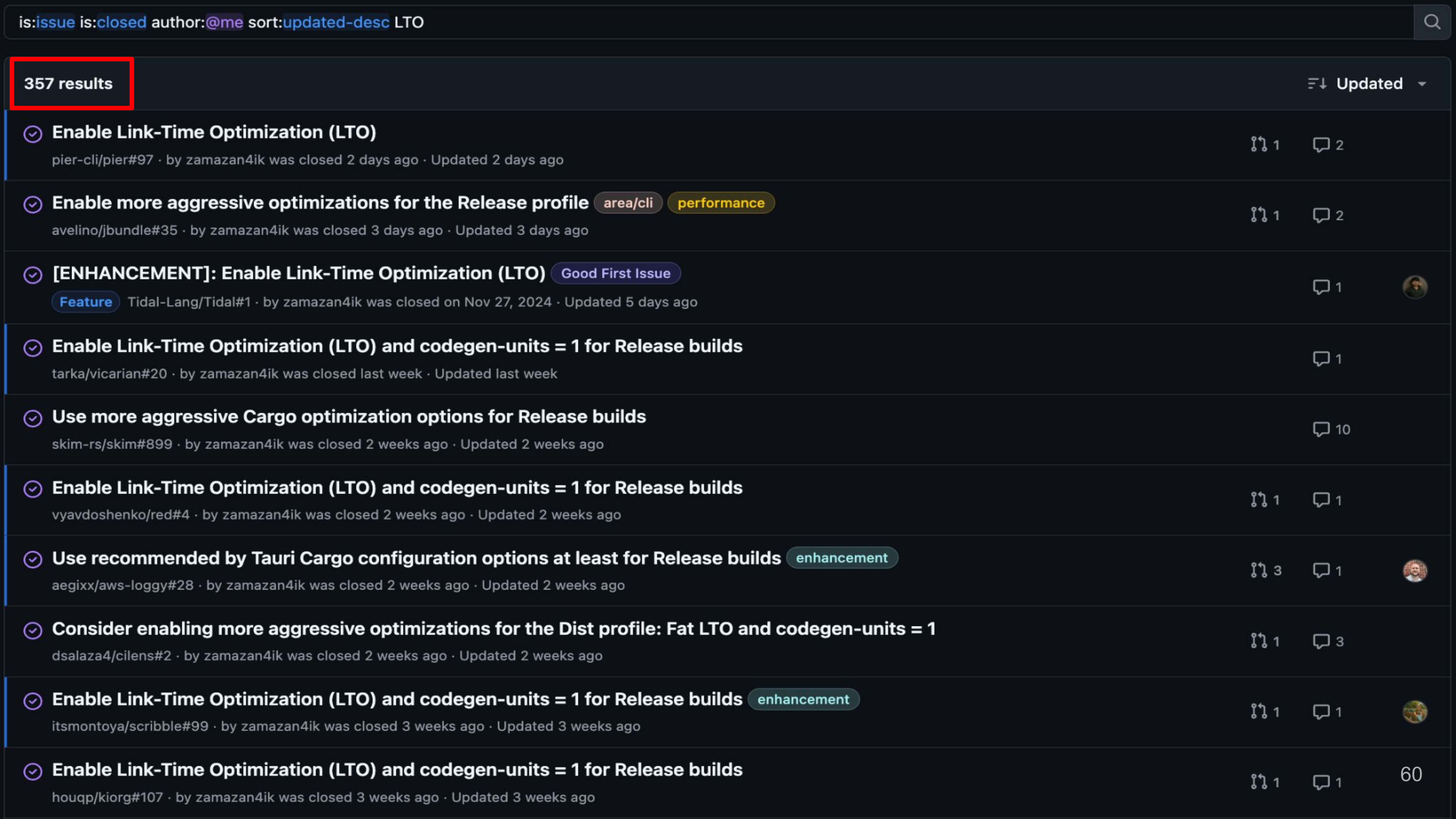
[ENHANCEMENT]: Enable Link-Time Optimization (LTO) Good First Issue
Feature Tidal-Lang/Tidal#1 · by zamazan4ik was closed on Nov 27, 2024 · Updated 5 days ago
1

Enable Link-Time Optimization (LTO) and codegen-units = 1 for Release builds
1jehuang/mermaid-rs-renderer#13 · zamazan4ik opened last week · Updated last week

Enable Link-Time Optimization (LTO) and codegen-units = 1 for Release builds
tarka/vicarian#20 · by zamazan4ik was closed last week · Updated last week
1

Tweak Cargo settings for the Release profile
marin-m/SongRec#211 · zamazan4ik opened 2 weeks ago · Updated 2 weeks ago

Use more aggressive Cargo optimization options for Release builds
skim-rs/skim#899 · by zamazan4ik was closed 2 weeks ago · Updated 2 weeks ago
10



is:issue is:closed author:@me sort:updated-desc LTO

357 results

Enable Link-Time Optimization (LTO)
pier-cli/pier#97 · by zamazan4ik was closed 2 days ago · Updated 2 days ago
1 · 2

Enable more aggressive optimizations for the Release profile area/cli performance
avelino/jbundle#35 · by zamazan4ik was closed 3 days ago · Updated 3 days ago
1 · 2

[ENHANCEMENT]: Enable Link-Time Optimization (LTO) Good First Issue
Feature Tidal-Lang/Tidal#1 · by zamazan4ik was closed on Nov 27, 2024 · Updated 5 days ago
1

Enable Link-Time Optimization (LTO) and codegen-units = 1 for Release builds
tarka/vicarian#20 · by zamazan4ik was closed last week · Updated last week
1

Use more aggressive Cargo optimization options for Release builds
skim-rs/skim#899 · by zamazan4ik was closed 2 weeks ago · Updated 2 weeks ago
10

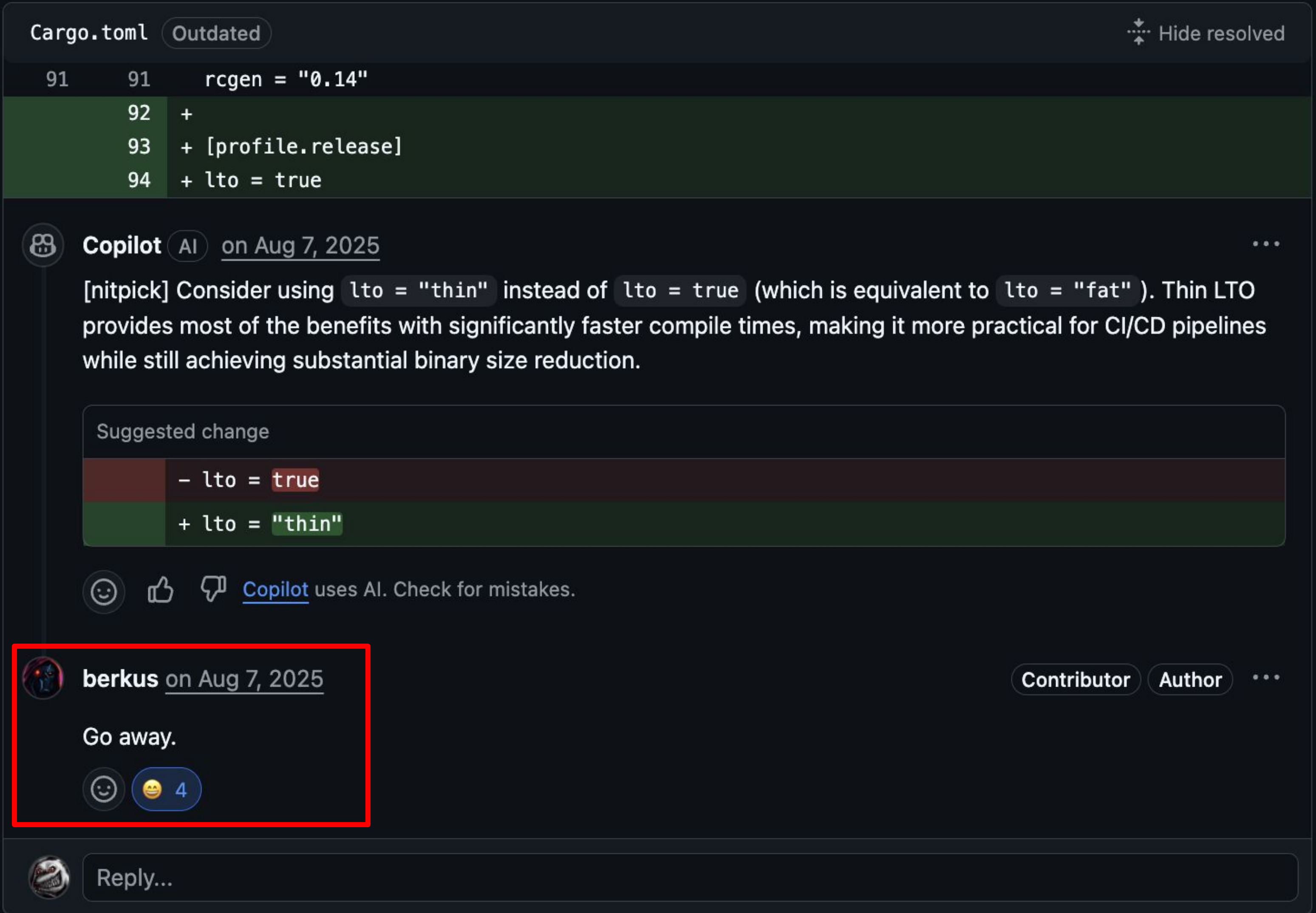
Enable Link-Time Optimization (LTO) and codegen-units = 1 for Release builds
vyavdoshenko/red#4 · by zamazan4ik was closed 2 weeks ago · Updated 2 weeks ago
1 · 1

Use recommended by Tauri Cargo configuration options at least for Release builds enhancement
aegixx/aws-loggy#28 · by zamazan4ik was closed 2 weeks ago · Updated 2 weeks ago
3 · 1

Consider enabling more aggressive optimizations for the Dist profile: Fat LTO and codegen-units = 1
dsalaza4/cilens#2 · by zamazan4ik was closed 2 weeks ago · Updated 2 weeks ago
1 · 3

Enable Link-Time Optimization (LTO) and codegen-units = 1 for Release builds enhancement
itsmontoya/scribble#99 · by zamazan4ik was closed 3 weeks ago · Updated 3 weeks ago
1 · 1

Enable Link-Time Optimization (LTO) and codegen-units = 1 for Release builds
houqp/kiorg#107 · by zamazan4ik was closed 3 weeks ago · Updated 3 weeks ago
1 · 1



Cargo.toml Outdated

```
  rcgen = "0.14"
+
+ [profile.release]
+ lto = true
```

Copilot AI on Aug 7, 2025

[nitpick] Consider using `lto = "thin"` instead of `lto = true` (which is equivalent to `lto = "fat"`). Thin LTO provides most of the benefits with significantly faster compile times, making it more practical for CI/CD pipelines while still achieving substantial binary size reduction.

Suggested change

```
- lto = true
+ lto = "thin"
```

Copilot uses AI. Check for mistakes.

berkus on Aug 7, 2025 Contributor Author

Go away.

😄 4

More ideas!

- Advent of Compiler Optimizations from Matt Godbolt - what about other optimization domains?
  - E.g. “Advent of ClickHouse optimizations”, huh?
- Performance challenges database from **real** open-source project
  - Like 1BRC but real challenges for real workloads.
  - “1 billion requests challenge for Nginx”? :)
- Pushing faster tools by default
  - Like **grep** -> **ripgrep**
- "Software Performance" devroom at FOSDEM ← you are sitting here!

Software Performance nicely complements FOSDEM

- Language-specific devrooms like Rust, Go (even Python ;)
- Compiler devrooms like GCC and LLVM
- Domain-specific performance achievements from HPC, Big Data & Data Science, Databases, SDS, Bioinformatics, etc.
- Delivering performance improvements with Package management, Nix, and Distributions

All my work is actually about the same thing

- “Awesome PGO” project
  - An attempt to make PGO benefits more **accessible** for users
- It’s “Awesome LTO” spin-off
  - An attempt to make LTO benefits more **accessible** for the Rust ecosystem
- “Software Performance devroom”
  - An attempt to make rapid software more **accessible** for everyone

And I hope that it’s only the beginning.

Thank you! Questions?

- **Emails**:
  - zamazan4ik@tut.by (primary)
  - zamazan4ik@gmail.com (secondary)
- **Matrix**: @zamazan4ik:matrix.org
- **Telegram**: zamazan4ik
- **Discord**: zamazan4ik
- **GitHub / GitLab / Codeberg**: zamazan4ik