



SemVer in Rust: breakage, tooling, and edge cases

Predrag Gruevski

 <https://predr.ag/>  @predrag@hachyderm.io  PredragGruevski  obi1kenobi



SemverChecks

Predrag Gruevski

 <https://predr.ag/>

 @predrag@hachyderm.io

 [PredragGruevski](#)

 [obi1kenobi](#)

SemVer is communication

Merged

Weekly `cargo update` of dependencies #642

github-actions merged 1 commit into `main` from `cargo_update` 2 days ago



obi1kenobi commented 2 days ago

Owner ...

Automation to keep dependencies in `Cargo.lock` current.

The following is the output from `cargo update` :

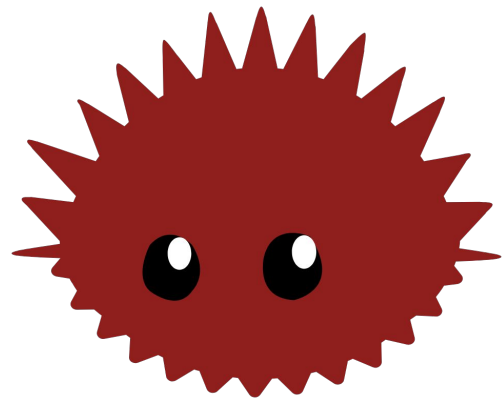
```
Updating anstream v0.6.7 -> v0.6.11
Updating async-compression v0.4.5 -> v0.4.6
Updating bitflags v2.4.1 -> v2.4.2
Updating clap v4.4.16 -> v4.4.18
Updating clap-verbosity-flag v2.1.1 -> v2.1.2
Updating clap_builder v4.4.16 -> v4.4.18
Updating gix-config-value v0.14.3 -> v0.14.4
Updating gix-path v0.10.3 -> v0.10.4
Updating gix-sec v0.10.3 -> v0.10.4
Updating gix-trace v0.1.6 -> v0.1.7
Updating gix-utils v0.1.8 -> v0.1.9
Updating h2 v0.3.23 -> v0.3.24
Updating hermit-abi v0.3.3 -> v0.3.4
Updating libz-ng-sys v1.1.14 -> v1.1.15
Updating linux-raw-sys v0.4.12 -> v0.4.13
Updating predicates v3.0.4 -> v3.1.0
Updating proc-macro2 v1.0.76 -> v1.0.78
Updating rayon v1.8.0 -> v1.8.1
Updating rayon-core v1.12.0 -> v1.12.1
Updating regex v1.10.2 -> v1.10.3
Updating regex-automata v0.4.3 -> v0.4.4
Updating smallvec v1.12.0 -> v1.13.1
Updating smol_str v0.2.0 -> v0.2.1
Updating unicode-bidi v0.3.14 -> v0.3.15
Updating uuid v1.6.1 -> v1.7.0
```



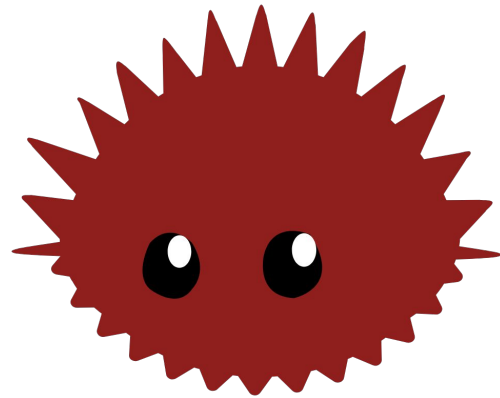
cargo update ...

✓ d6d5a51

**SemVer is so hard,
no mere mortals can uphold it.**

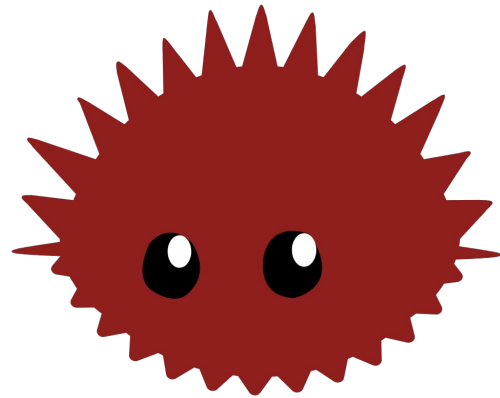


SemVer is so hard,
no mere mortals can uphold it.



Computers are no mere mortals.
They are *really good at SemVer.*

SemVer is so hard,
no mere mortals can uphold it.



Computers are no mere mortals.
They are *really good at SemVer.*

**Here's how hard
SemVer is in Rust**

Falsehoods we believed about SemVer



Falsehoods we believed about SemVer



- Crates always adhere to SemVer

Version 0.5.1 breaks SemVer guarantees #285

✓ Closed

jonasbb opened this issue on Nov 25, 2018 · 5 comments

Version 0.5.1 breaks SemVer guarantees #285

✓ Closed

jonasbb opened this issue on Nov 25, 2018 · 5 comments

Backwards incompatibility for ArgMatches and UnwindSafe #3876

✓ Closed

🔄 2 tasks done

doivosevic opened this issue on Jun 27, 2022 · 7 comments

Version 0.5.1 breaks SemVer guarantees #285

✓ Closed

jonasbb opened this issue on Nov 25, 2018 · 5 comments

Backwards incompatibility for ArgMatches and UnwindSafe #3876

✓ Closed

🔄 2 tasks done

doivosevic opened this issue on Jun 27, 2022 · 7 comments

SemVer breaking change caused by mio upgrade to 0.8 #4512

✓ Closed

ecton opened this issue on Feb 17, 2022 · 3 comments

Version 0.5.1 breaks SemVer guarantees #285

✓ Closed

jonasbb opened this issue on Nov 25, 2018 · 5 comments

Backwards incompatibility for ArgMatches and UnwindSafe #3876

✓ Closed

🔄 2 tasks done

doivosevic opened this issue on Jun 27, 2022 · 7 comments

SemVer breaking change caused by mio upgrade to 0.8 #4512

✓ Closed

ecton opened this issue on Feb 17, 2022 · 3 comments

Semver violation in 0.18.12 — please re-export `git2` since it's part of public API #91

✓ Closed

obi1kenobi opened this issue on Jan 23 · 2 comments

Version 0.5.1 breaks SemVer guarantees #285

✓ Closed

jonasbb opened this issue on Nov 25, 2018 · 5 comments

Backwards incompatibility for ArgMatches and UnwindSafe #3876

✓ Closed

🔄 2 tasks done

doivosevic opened this issue on Jun 27, 2022 · 7 comments

tracing 0.1.38 included accidentally-breaking change from added Drop impl #2578

🔄 Open

jonhoo opened this issue on Apr 28 · 19 comments

Semver violation in 0.18.12 — please re-export `git2` since it's part of public API #91

✓ Closed

obi1kenobi opened this issue on Jan 23 · 2 comments

Version 0.5.1 breaks SemVer guarantees #285

✓ Closed

jonasbb opened this issue on Nov 25, 2018 · 5 comments

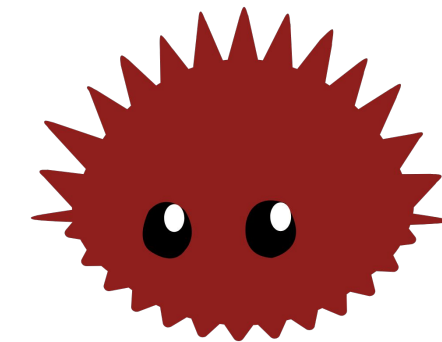
Backwards incompatibility for ArgMatches and UnwindSafe #3876

✓ Closed

🔄 2 tasks done

doivosevic opened this issue on Jun 27, 2022 · 7 comments

tracing 0.1.38 included **accidentally-breaking** change from added Drop impl #2578



🕒 Open

jonhoo opened this issue on Apr 28 · 19 comments

Semver violation in 0.18.12 — please re-export `git2` since it's part of public API #91

✓ Closed

obi1kenobi opened this issue on Jan 23 · 2 comments

Version 0.5.1 breaks SemVer guarantees #285

✓ Closed

jonasbb opened this issue on Nov 25, 2018 · 5 comments

Backwards incompatibility for ArgMatches and UnwindSafe #3876

✓ Closed

🔄 2 tasks done

doivosevic opened this issue on Jun 27, 2022 · 7 comments

tracing 0.1.38 included accidentally-breaking change from added Drop impl #2578

🔄 Open

jonhoo opened this issue on Apr 28 · 19 comments

Semver violation in 0.18.12 — please re-export `git2` since it's part of public API #91

✓ Closed

obi1kenobi opened this issue on Jan 23 · 2 comments

Version 0.5.1 breaks SemVer guarantees #285

🔒 Closed

jonasbb opened this issue on Nov 25, 2018 · 5 comments

Backwards incompatibility for ArgMatches and UnwindSafe #3876

🔒 Closed

🔄 2 tasks done

doivosevic opened this issue on Jun 27, 2022 · 7 comments

tracing 0.1.38 included accidentally-breaking change from added Drop impl #2578

🔓 Open

jonhoo opened this issue on Apr 28 · 19 comments

Semver violation in 0.18.12 — please re-export `git2` since it's part of public API #91

🔒 Closed

obi1kenobi opened this issue on Jan 23 · 2 comments



Open

tracing 0.1.38 included accidentally-breaking change from added Drop impl #2578
jonhoo opened this issue on Apr 28, 2023 · 19 comments

jasl mentioned this issue on Apr 29, 2023
Revert "Bump tracing from 0.1.37 to 0.1.38" paritytech/cargo-contract#1096 Merged

fooooooooooooooooo mentioned this issue on Apr 29, 2023
Can't build 0.23.0 cargo-bins/cargo-binstall#1019 Closed

71 mentioned this issue on Apr 30, 2023
cargo: downgrade tracing from yanked 0.1.38 to 0.1.37 martinvonz/jj#1563 Merged
4 tasks

JamesHinshelwood mentioned this issue on May 1, 2023
Revert tracing from 0.1.38 back to 0.1.37 Zilliqa/zq2#119 Merged

rillian mentioned this issue on May 2, 2023
Downgrade yanked tracing 0.1.38 to 0.1.37 brave/star-randsrv#83 Merged

ilslv mentioned this issue on May 4, 2023
Consider entering Span on Drop for Instrumented #2541 Open

wizard-28 added a commit to pacstall/pacbot that referenced this issue on May 6, 2023
build(tracing): 0.1.38 -> 0.1.37 ... Verified ✓ c8fa820

Turbo87 mentioned this issue on May 10, 2023
Revert "Update Rust crate tracing to v0.1.38 (#6387)" rust-lang/crates.io#6458 Merged

zecakeh mentioned this issue on May 10, 2023
Downgrade tracing to 0.1.37 matrix-org/matrix-authentication-service#1178 Merged



**SemVer violations are
miscommunication**



Falsehoods we believed about SemVer



- ~~• Crates always adhere to SemVer~~

Falsehoods we believed about SemVer

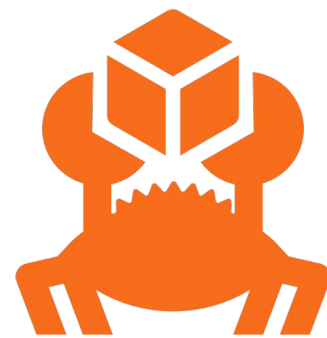


- ~~Crates always adhere to SemVer~~
- Careful coding is enough to avoid violating SemVer

**1 in 6 of the top 1000 crates
have broken SemVer at least once**

Joint work with Tomasz Nowak, Mieszko Grodzicki, Bartosz Smolarczyk, Michał Staniewski
<https://predr.ag/blog/semver-violations-are-common-better-tooling-is-the-answer/>

**1 in 6 of the top 1000 crates
have broken SemVer at least once**

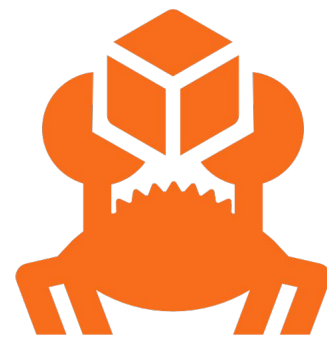


SemverChecks

Joint work with Tomasz Nowak, Mieszko Grodzicki, Bartosz Smolarczyk, Michał Staniewski

<https://predr.ag/blog/semver-violations-are-common-better-tooling-is-the-answer/>

1 in 6 of the **top 1000 crates**
have broken SemVer at least once

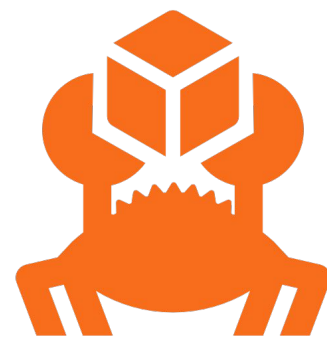


SemverChecks

Joint work with Tomasz Nowak, Mieszko Grodzicki, Bartosz Smolarczyk, Michał Staniewski

<https://predr.ag/blog/semver-violations-are-common-better-tooling-is-the-answer/>

**>3% of the 14000 scanned releases
had at least one SemVer violation**



SemverChecks

Joint work with Tomasz Nowak, Mieszko Grodzicki, Bartosz Smolarczyk, Michał Staniewski

<https://predr.ag/blog/semver-violations-are-common-better-tooling-is-the-answer/>



obi1kenobi commented 2 days ago

Owner ...

Automation to keep dependencies in Cargo.lock current.

The following is the output from cargo update :

```
Updating anstream v0.6.7 -> v0.6.11
Updating async-compression v0.4.5 -> v0.4.6
Updating bitflags v2.4.1 -> v2.4.2
Updating clap v4.4.16 -> v4.4.18
Updating clap-verbosity-flag v2.1.1 -> v2.1.2
Updating clap_builder v4.4.16 -> v4.4.18
Updating gix-config-value v0.14.3 -> v0.14.4
Updating gix-path v0.10.3 -> v0.10.4
Updating gix-sec v0.10.3 -> v0.10.4
Updating gix-trace v0.1.6 -> v0.1.7
Updating gix-utils v0.1.8 -> v0.1.9
Updating h2 v0.3.23 -> v0.3.24
Updating hermit-abi v0.3.3 -> v0.3.4
Updating libz-ng-sys v1.1.14 -> v1.1.15
Updating linux-raw-sys v0.4.12 -> v0.4.13
Updating predicates v3.0.4 -> v3.1.0
Updating proc-macro2 v1.0.76 -> v1.0.78
Updating rayon v1.8.0 -> v1.8.1
Updating rayon-core v1.12.0 -> v1.12.1
Updating regex v1.10.2 -> v1.10.3
Updating regex-automata v0.4.3 -> v0.4.4
Updating smallvec v1.12.0 -> v1.13.1
Updating smol_str v0.2.0 -> v0.2.1
Updating unicode-bidi v0.3.14 -> v0.3.15
Updating uuid v1.6.1 -> v1.7.0
```



Statistically, there's a semver violation somewhere in here...

Falsehoods we believed about SemVer



- ~~• Crates always adhere to SemVer~~
- ~~• Careful coding is enough to avoid violating SemVer~~

Falsehoods we believed about SemVer



- ~~Crates always adhere to SemVer~~
- ~~Careful coding is enough to avoid violating SemVer~~
- Breaking changes always require major versions



master ▾

rfcs / text / 1105-api-evolution.md

↑ Top

Preview

Code

Blame

Raw



Detailed design

For clarity, in the rest of the RFC, we will use the following terms:

- **Major change:** a change that requires a major semver bump.
- **Minor change:** a change that requires only a minor semver bump.
- **Breaking change:** a change that, *strictly speaking*, can cause downstream code to fail to compile.

What we will see is that in Rust today, almost any change is technically a breaking change. For example, given the way that globs currently work, *adding any public item* to a library can break its clients (more on that later). But not all breaking changes are equal.

So, this RFC proposes that **all major changes are breaking, but not all breaking changes are major.**



master

rfcs / text / 1105-api-evolution.md

↑ Top

Preview

Code

Blame

Raw



Detailed design

For clarity, in the rest of the RFC, we will use the following terms:

- **Major change:** a change that requires a major semver bump.
- **Minor change:** a change that requires only a minor semver bump.
- **Breaking change:** a change that, *strictly speaking*, can cause downstream code to fail to compile.

What we will see is that in Rust today, almost any change is technically a breaking change. For example, given the way that globs currently work, *adding any public item* to a library can break its clients (more on that later). But not all breaking changes are equal.

So, this RFC proposes that **all major changes are breaking, but not all breaking changes are major.**



master

rfcs / text / 1105-api-evolution.md

↑ Top

Preview

Code

Blame

Raw



Detailed design

For clarity, in the rest of the RFC, we will use the following terms:

- **Major change:** a change that requires a major semver bump.
- **Minor change:** a change that requires only a minor semver bump.
- **Breaking change:** a change that, *strictly speaking*, can cause downstream code to fail to compile.

What we will see is that in Rust today, almost any change is technically a breaking change. For example, given the way that globs currently work, *adding any public item* to a library can break its clients (more on that later). But not all breaking changes are equal.

So, this RFC proposes that **all major changes are breaking, but not all breaking changes are major.**



master

rfcs / text / 1105-api-evolution.md

↑ Top

Preview

Code

Blame

Raw



Detailed design

For clarity, in the rest of the RFC, we will use the following terms:

- **Major change:** a change that requires a major semver bump.
- **Minor change:** a change that requires only a minor semver bump.
- **Breaking change:** a change that, *strictly speaking*, can cause downstream code to fail to compile.

What we will see is that in Rust today, almost any change is technically a breaking change. For example, given the way that globs currently work, *adding any public item* to a library can break its clients (more on that later). But not all breaking changes are equal.

So, this RFC proposes that **all major changes are breaking, but not all breaking changes are major.**



master

rfcs / text / 1105-api-evolution.md

↑ Top

Preview

Code

Blame

Raw



Detailed design

For clarity, in the rest of the RFC

- Major change: a change
- Minor change: a change
- Breaking change: a change

What we will see is that in Rust

example, given the way that g

clients (more on that later). But not all breaking changes are equal.

So, this RFC proposes that **all major changes are breaking, but not all breaking changes are major.**

- Adding new items to a module



master

rfcs / text / 1105-api-evolution.md

↑ Top

Preview

Code

Blame

Raw



Detailed design

For clarity, in the rest of the RFC

- Major change: a change
- Minor change: a change
- Breaking change: a change

What we will see is that in Rust

example, given the way that global

clients (more on that later). But not all breaking changes are equal.

So, this RFC proposes that **all major changes are breaking, but not all breaking changes are major.**

- Adding new items to a module
- Changes that break type inference (requiring type annotations in downstream code)



master

rfcs / text / 1105-api-evolution.md

↑ Top

Preview

Code

Blame

Raw



Detailed design

For clarity, in the rest of the RFC

- Major change: a change
- Minor change: a change
- Breaking change: a change

- Adding new items to a module
- Changes that break type inference (requiring type annotations in downstream code)
- Reverting accidental API changes

What we will see is that in Rust

example, given the way that

clients (more on that later). But not all breaking changes are equal.

So, this RFC proposes that **all major changes are breaking, but not all breaking changes are major.**



master

rfcs / text / 1105-api-evolution.md

↑ Top

Preview

Code

Blame

Raw



Detailed design

For clarity, in the rest of the RFC

- Major change: a change
- Minor change: a change
- Breaking change: a change

What we will see is that in Rust

example, given the way that global

clients (more on that later). But not all breaking changes are equal.

- Adding new items to a module
- Changes that break type inference (requiring type annotations in downstream code)
- Reverting accidental API changes
- Critical soundness or security fixes, subject to the maintainer's judgment call

So, this RFC proposes that **all major changes are breaking, but not all breaking changes are major.**

Falsehoods we believed about SemVer



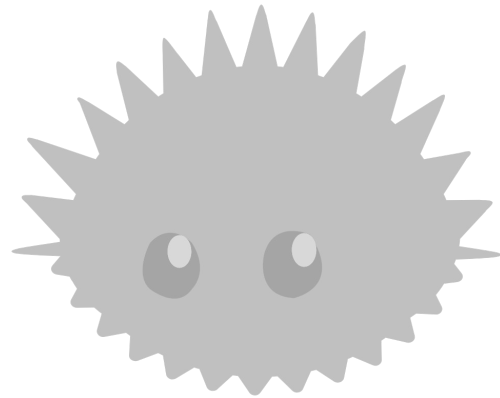
- ~~Crates always adhere to SemVer~~
- ~~Careful coding is enough to avoid violating SemVer~~
- ~~Breaking changes always require major versions~~

SemVer's rules are complex!

SemVer's rules are complex!

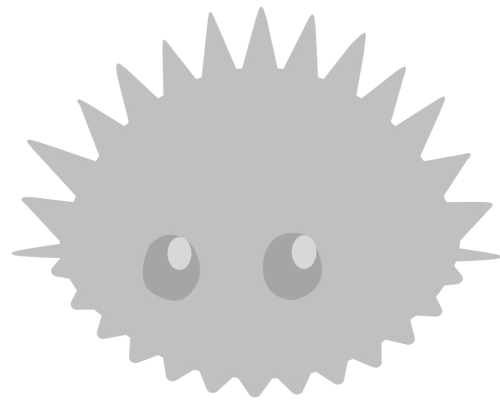
Automation can help!

SemVer is so hard,
no mere mortals can uphold it.



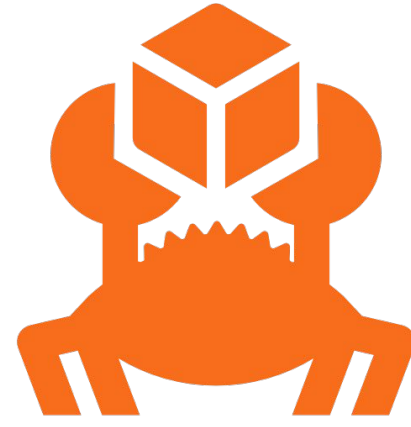
Computers are no mere mortals.
They are *really good at SemVer.*

SemVer is so hard,
no mere mortals can uphold it.

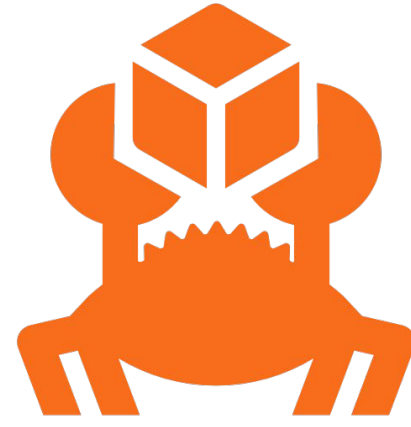


Computers are no mere mortals.
They are *really good at SemVer.*

They are best where we
do poorly, and vice versa.



SemverChecks

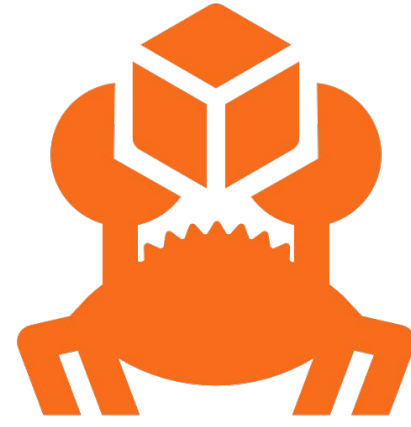


SemverChecks



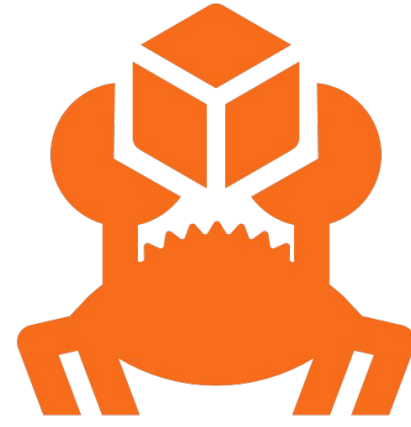
Cargo





SemverChecks

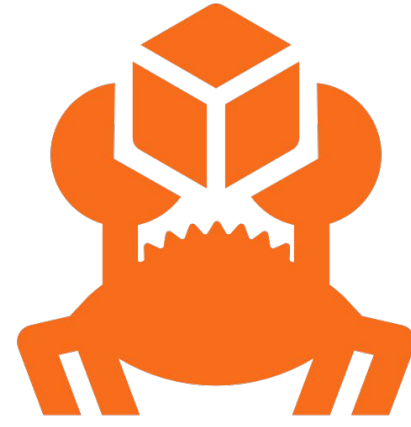
cargo semver-checks && cargo publish



SemverChecks

`cargo semver-checks && cargo publish`

Detects the version bump,
then scans for API changes
inappropriate for that bump.



SemverChecks

```
cargo semver-checks && cargo publish
```

```
cargo install cargo-semver-checks --locked
```

Release-plz

Release Rust crates from CI with a Release PR



crates.io v0.3.42 CI passing docker

Release-plz helps you release your Rust packages by automating:

- CHANGELOG generation (with [git-cliff](#)).
- Creation of GitHub/Gitea releases.
- Publishing to a cargo registry (`crates.io` by default).
- Version bumps in `Cargo.toml` .

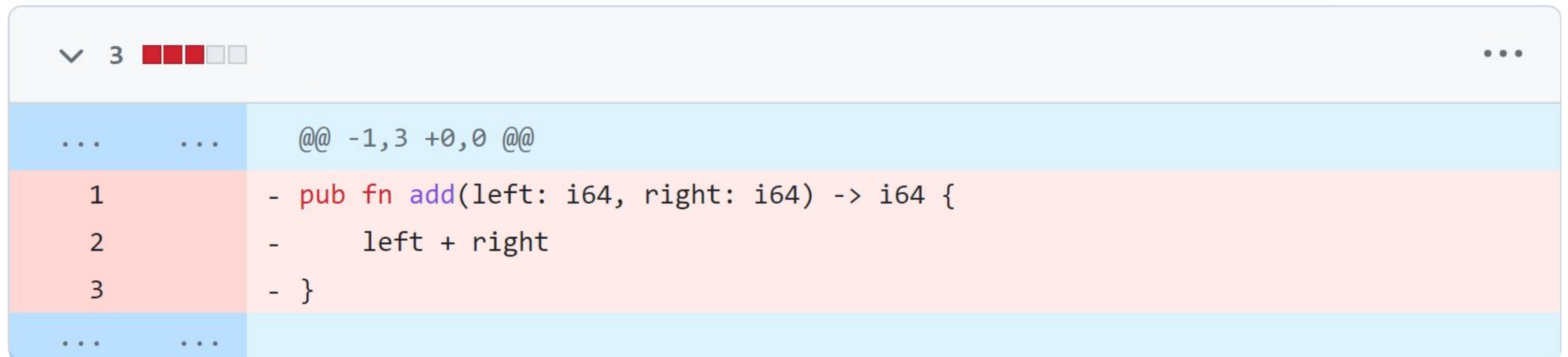
Release-plz updates your packages with a release Pull Request based on:

- Your git history, following [Conventional commits](#).
 - API breaking changes (detected by [cargo-semver-checks](#)).
-

Examples

Example #1: pub fn gets deleted

Showing 1 changed file with 0 additions and 3 deletions.



The screenshot shows a diff viewer interface. At the top, there is a header with a dropdown arrow, the number '3', and a progress indicator consisting of three red squares followed by two grey squares. Below this, the diff content is displayed in a table-like format. The first row is a light blue header row containing '...', '...', and '@@ -1,3 +0,0 @@'. The next three rows are highlighted in light red, indicating deletions. These rows contain line numbers '1', '2', and '3' in the left column, and the corresponding code lines in the right column: '- pub fn add(left: i64, right: i64) -> i64 {', '- left + right', and '- }'. The final row is another light blue header row containing '...', '...', and '@@'. The entire diff content is enclosed in a light grey rounded rectangle.

```
... .. @@ -1,3 +0,0 @@
1 - pub fn add(left: i64, right: i64) -> i64 {
2 -     left + right
3 - }
... .. @@
```

https://github.com/obi1kenobi/semver-examples/compare/main...easy_01

```
$ cargo semver-checks
  Parsing easy_01 v0.1.0 (current)
  Parsing easy_01 v0.1.0 (baseline)
  Checking easy_01 v0.1.0 -> v0.1.0 (no change)
  Completed [ 0.011s] 58 checks; 57 passed, 1 failed, 0 unnecessary
```

```
--- failure function_missing: pub fn removed or renamed ---
```

Description:

A publicly-visible function cannot be imported by its prior path. A `pub use` may have been removed, or the function itself may have been renamed or removed entirely.

Failed in:

```
function easy_01::add, previously in file semver-examples/easy_01/old/src/lib.rs:1
```

```
  Final [ 0.012s] semver requires new major version: 1 major and 0 minor checks failed
```



Deletions of `pub` items are always a major breaking change

Falsehoods we believed about SemVer



- ~~Crates always adhere to SemVer~~
- ~~Careful coding is enough to avoid violating SemVer~~
- ~~Breaking changes always require major versions~~
- Deletions of `pub` items are always a major breaking change

```
mod private {  
    // This function cannot be imported from  
    // outside this crate.  
    //  
    // Deleting it is not a breaking change.  
    pub fn example() {}  
}
```



```
mod private {  
    // This function cannot be imported from  
    // outside this crate.  
    //  
    // Deleting it is not a breaking change.  
    pub fn example() {}  
}  
  
#[doc(hidden)]  
pub mod macro_helpers {  
    // This function is public, but  
    // explicitly documented as not public API  
    // via #[doc(hidden)] on its module.  
    //  
    // Deleting it is not a semver major change.  
    pub fn example() {}  
}
```

```
#[doc(hidden)]  
pub mod macro_helpers {  
    pub fn example() {}  
}  
  
// Oops! Now the function is public API,  
// since users can import it as  
// `use this_crate::example;`  
// without ever touching any non-public API.  
pub use macro_helpers::example;
```

Falsehoods we believed about SemVer



- ~~Crates always adhere to SemVer~~
- ~~Careful coding is enough to avoid violating SemVer~~
- ~~Breaking changes always require major versions~~
- Deletions of `pub` items are always a major breaking change

Falsehoods we believed about SemVer



- ~~Crates always adhere to SemVer~~
- ~~Careful coding is enough to avoid violating SemVer~~
- ~~Breaking changes always require major versions~~
- ~~Deletions of `pub` items are always a major breaking change~~

Example #2: adding a field to a struct

```
1      1      pub struct Foo {
2      2          pub first: i64,
3      3          pub second: bool,
4      4      +  pub third: Option<String>,
5      5      }
6      6
7      7      impl Foo {
8      8          pub fn new(first: i64, second: bool) -> Self {
9      9              Self {
10     10                  first,
11     11                  second,
12     12      +          third: None, // set to a default value
13     13              }
14     14          }
15     15      }
```




Adding fields to a struct can only be breaking via changes to its methods

Falsehoods we believed about SemVer



- ~~• Crates always adhere to SemVer~~
- ~~• Careful coding is enough to avoid violating SemVer~~
- ~~• Breaking changes always require major versions~~
- ~~• Deletions of `pub` items are always a major breaking change~~
- Adding fields to a struct can only be breaking via changes to its methods

Example #2: adding a field to a struct

```
1      1      pub struct Foo {
2      2          pub first: i64,
3      3          pub second: bool,
4      4      +  pub third: Option<String>,
5      5      }
6      6
7      7      impl Foo {
8      8          pub fn new(first: i64, second: bool) -> Self {
9      9              Self {
10     10                  first,
11     11                  second,
12     12      +          third: None, // set to a default value
13     13              }
14     14          }
15     15      }
```

Example #2: adding a field to a struct

```
1 1 pub struct Foo {
2 2     pub first: i64,
3 3     pub second: bool,
4 4 +   pub third: Option<String>,
4 5 }
5 6
6 7 impl Foo {
7 8     pub fn new(first: i64, second: bool) -> Self {
8 9         Self {
9 10            first,
10 11           second,
12 12 +         third: None, // set to a default value
11 13        }
12 14     }
13 15 }
```

Not marked
#[non_exhaustive]

Example #2: adding a field to a struct

```
1 1 pub struct Foo {
2 2     pub first: i64,
3 3     pub second: bool,
4 4 + pub third: Option<String>,
4 5 }
5 6
6 7 impl Foo {
7 8     pub fn new(first: i64, second: bool) -> Self {
8 9         Self {
9 10             first,
10 11             second,
12 12 +             third: None, // set to a default value
11 13         }
12 14     }
13 15 }
```

Not marked
#[non_exhaustive]

All prior fields were pub

Example #2: adding a field to a struct

```
1 1 pub struct Foo {
2 2     pub first: i64,
3 3     pub second: bool,
4 4 + pub third: Option<String>,
4 5 }
5 6
6 7 impl Foo {
7 8     pub fn new(first: i64, second: bool) -> Self {
8 9         Self {
9 10            first,
10 11           second,
12 12 +          third: None, // set to a default value
11 13        }
12 14     }
13 15 }
```

Not marked
#[non_exhaustive]

All prior fields were pub

```
use upstream::Foo;

fn main() {
    // Users were allowed
    // to construct `Foo` *directly*.
    //
    // This is now broken
    // since it doesn't specify
    // any value for `third`.
    let value = Foo {
        first: 0,
        second: false,
    };
}
```

```
$ cargo semver-checks
```

```
  Parsing med_01 v0.1.0 (current)
```

```
  Parsing med_01 v0.1.0 (baseline)
```

```
  Checking med_01 v0.1.0 -> v0.1.0 (no change)
```

```
Completed [ 0.010s] 58 checks; 57 passed, 1 failed, 0 unnecessary
```

```
--- failure constructible_struct_adds_field: externally-constructible struct adds field ---
```

Description:

A pub struct constructible with a struct literal has a new pub field. Existing struct literals must be updated to include the new field.

Failed in:

```
field Foo::third, in file semver-examples/med_01/new/src/lib.rs:4
```

```
Final [ 0.010s] semver requires new major version: 1 major and 0 minor checks failed
```

Falsehoods we believed about SemVer



- ~~• Crates always adhere to SemVer~~
- ~~• Careful coding is enough to avoid violating SemVer~~
- ~~• Breaking changes always require major versions~~
- ~~• Deletions of `pub` items are always a major breaking change~~
- Adding fields to a struct can only be breaking via changes to its methods

Falsehoods we believed about SemVer



- ~~Crates always adhere to SemVer~~
- ~~Careful coding is enough to avoid violating SemVer~~
- ~~Breaking changes always require major versions~~
- ~~Deletions of `pub` items are always a major breaking change~~
- ~~Adding fields to a struct can only be breaking via changes to its methods~~

Example #3: “internal-only changes”

```
hard_01/old/src/lib.rs
```

```
@@ -1,11 +1,14 @@
```

1 <code>#[derive(Clone)]</code> 2 <code>struct Foo {</code> 3 <code>- value: &'static str,</code> 4 <code>}</code> 5 6 <code>fn make_foo() -> Foo {</code> 7 <code> Foo {</code> 8 <code>- value: "some string",</code> 9 <code> }</code> 10 <code>}</code> 11	1 <code>+ use std::rc::Rc;</code> 2 <code>+</code> 3 <code>#[derive(Clone)]</code> 4 <code>struct Foo {</code> 5 <code>+ value: Rc<str>, // support non-static strings;</code> 6 <code>+ // ref-count for cheap cloning</code> 7 <code>}</code> 8 9 <code>fn make_foo() -> Foo {</code> 10 <code> Foo {</code> 11 <code>+ value: Rc::from("some string"),</code> 12 <code> }</code> 13 <code>}</code> 14
---	--



“If I didn’t touch it, I didn’t break it.”

Falsehoods we believed about SemVer



- ~~Crates always adhere to SemVer~~
- ~~Careful coding is enough to avoid violating SemVer~~
- ~~Breaking changes always require major versions~~
- ~~Deletions of `pub` items are always a major breaking change~~
- ~~Adding fields to a struct can only be breaking via changes to its methods~~
- “If I didn’t touch it, I didn’t break it.”

```
$ cargo semver-checks
  Parsing hard_01 v0.1.0 (current)
  Parsing hard_01 v0.1.0 (baseline)
  Checking hard_01 v0.1.0 -> v0.1.0 (no change)
  Completed [ 0.010s] 58 checks; 57 passed, 1 failed, 0 unnecessary
```

```
--- failure auto_trait_impl_removed: auto trait no longer implemented ---
```

Description:

A public type has stopped implementing one or more auto traits. This can break downstream code that depends on those traits being implemented.

Failed in:

```
type Bar is no longer Send, in file semver-examples/hard_01/new/src/lib.rs:16
```

```
type Bar is no longer Sync, in file semver-examples/hard_01/new/src/lib.rs:16
```

```
  Final [ 0.010s] semver requires new major version: 1 major and 0 minor checks failed
```

```
hard_01/old/src/lib.rs
@@ -1,11 +1,14 @@
...
1 + use std::rc::Rc;
2 +
1 3  #[derive(Clone)]
2 4  struct Foo {
3  -     value: &'static str,
5  +     value: Rc<str>, // support non-static strings;
6  +                       // ref-count for cheap cloning
4  7  }
5  8
6  9  fn make_foo() -> Foo {
7  10     Foo {
8  -     value: "some string",
11 +     value: Rc::from("some string"),
9  12     }
10 13     }
11 14
```

Failed in:

type Bar is no longer Send,
type Bar is no longer Sync,

```
hard_01/old/src/lib.rs
@@ -1,11 +1,14 @@
...
1 + use std::rc::Rc;
2 +
1 3  #[derive(Clone)]
2 4  struct Foo {
3  -     value: &'static str,
5  +     value: Rc<str>, // support non-static strings;
6  +                       // ref-count for cheap cloning
4  7  }
5  8
6  9  fn make_foo() -> Foo {
7  10     Foo {
8  -     value: "some string",
11 +     value: Rc::from("some string"),
9  12     }
10 13     }
11 14
```

Failed in:

type Bar is no longer Send,
type Bar is no longer Sync,

I didn't touch it, so I didn't break it
... right? 😅


```
hard_01/old/src/lib.rs
... @@ -1,11 +1,14 @@
1 + use std::rc::Rc;
2 +
1 3  #[derive(Clone)]
2 4  struct Foo {
3  -     value: &'static str,
5  +     value: Rc<str>, // support non-static strings;
6  +                       // ref-count for cheap cloning
4  7  }
5  8
6  9  fn make_foo() -> Foo {
7  10     Foo {
8  -     value: "some string",
11 +     value: Rc::from("some string"),
9  12     }
10 13 }
11 14
```

#[derive(Clone)]
struct Foo {

type Bar is in here

Failed in:
type Bar is no longer Send,
type Bar is no longer Sync,

I didn't touch it, so I didn't break it
... right? 😅

```
15  #[derive(Clone)]
16  pub struct Bar {
17      int: i64,
18      foo: Foo,
19  }
20
21  pub fn barify() -> Bar {
22      Bar {
23          int: 0,
24          foo: make_foo(),
25      }
26  }
```

Bar is public, so its implemented traits are public.

Bar contains a Foo.


```
15  #[derive(Clone)]
16  pub struct Bar {
17      int: i64,
18      foo: Foo,
19  }
20
21  pub fn barify() -> Bar {
22      Bar {
23          int: 0,
24          foo: make_foo(),
25      }
26  }
```

Bar is public, so its implemented traits are public.

Bar contains a Foo.

Auto-traits: traits that are automatically implemented for us *whenever possible*.

A type implements an auto-trait if all its constituents also implement the trait.

```
15     #[derive(Clone)]
16     pub struct Bar {
17         int: i64,
18         foo: Foo,
19     }
20
```

Bar is public, so its implemented traits are public.

Bar contains a Foo.

Auto-traits: traits that are automatically implemented for us *whenever possible*.

A type implements an auto-trait if all its constituents also implement the trait.

&'static str is both Send and Sync.
Rc<str> is neither.

```
+
#[derive(Clone)]
struct Foo {
-     value: &'static str,
+     value: Rc<str>, // support non-static strings;
+                 // ref-count for cheap cloning
}
```

```
26 }
```


Falsehoods we believed about SemVer

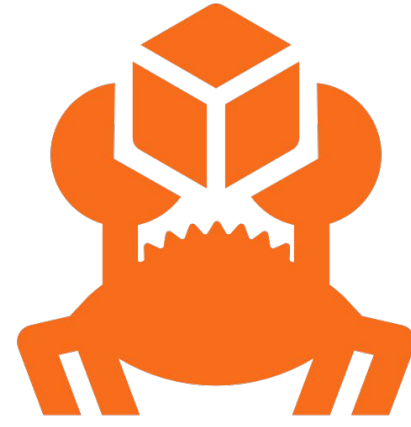


- ~~Crates always adhere to SemVer~~
- ~~Careful coding is enough to avoid violating SemVer~~
- ~~Breaking changes always require major versions~~
- ~~Deletions of `pub` items are always a major breaking change~~
- ~~Adding fields to a struct can only be breaking via changes to its methods~~
- “If I didn’t touch it, I didn’t break it.”

Falsehoods we believed about SemVer



- ~~Crates always adhere to SemVer~~
- ~~Careful coding is enough to avoid violating SemVer~~
- ~~Breaking changes always require major versions~~
- ~~Deletions of `pub` items are always a major breaking change~~
- ~~Adding fields to a struct can only be breaking via changes to its methods~~
- ~~“If I didn’t touch it, I didn’t break it.”~~



SemverChecks

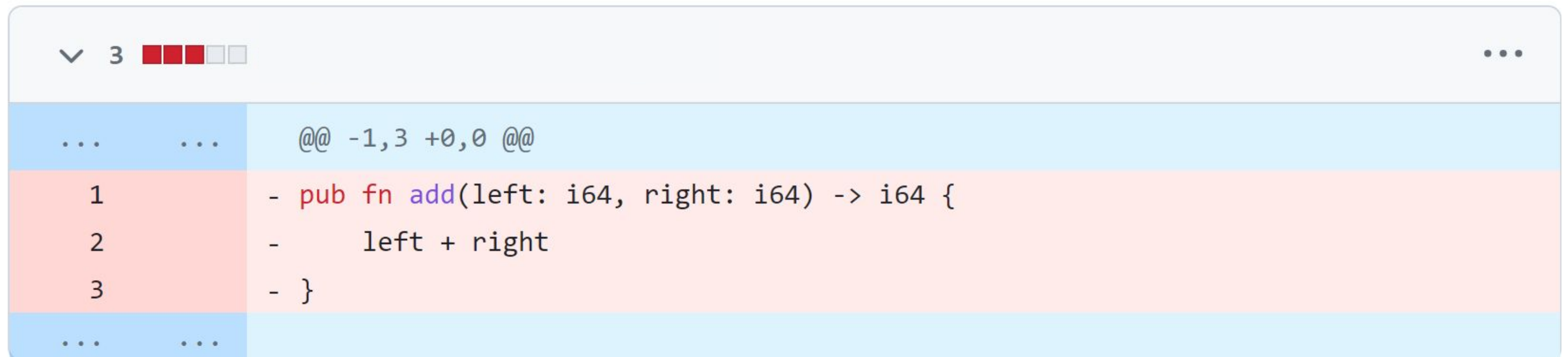
`cargo semver-checks && cargo publish`

Detects the version bump,
then scans for API changes
inappropriate for that bump.

How does this work?

Example: pub fn gets deleted

Showing 1 changed file with 0 additions and 3 deletions.



The screenshot shows a diff viewer interface. At the top, there is a dropdown menu with a downward arrow, the number '3', and a progress indicator consisting of three red squares followed by two grey squares. To the right of this is a three-dot menu icon. Below this header, the diff content is displayed in a table-like structure. The first row is a light blue header row containing '...', '...', and '@@ -1,3 +0,0 @@'. The next three rows are highlighted in light red, indicating deletions. These rows contain line numbers '1', '2', and '3' in the left column, and the corresponding code lines: '- pub fn add(left: i64, right: i64) -> i64 {', '- left + right', and '- }'. The final row is another light blue header row containing '...', '...', and '@@'. The code is color-coded: 'pub' is red, 'fn' is purple, and 'add' is blue.

```
... .. @@ -1,3 +0,0 @@
1 - pub fn add(left: i64, right: i64) -> i64 {
2 - left + right
3 - }
... .. @@
```

Example: pub fn gets deleted

Breaking change if all of these are true:

Example: pub fn gets deleted

Breaking change if all of these are true:

& Previously, the function was `pub`

Example: pub fn gets deleted

Breaking change if all of these are true:

& Previously, the function was `pub`

& Another crate could have imported and used it

Example: pub fn gets deleted

Breaking change if all of these are true:

- & Previously, the function was `pub`
- & Another crate could have imported and used it
- & That import did not rely on any `#[doc(hidden)]` items

Example: pub fn gets deleted

Breaking change if all of these are true:

- & Previously, the function was `pub`
- & Another crate could have imported and used it
- & That import did not rely on any `#[doc(hidden)]` items
- & Now, the same import name no longer satisfies the above

Example: pub fn gets deleted

Find all functions such that:

- & Previously, the function was `pub`
- & Another crate could have imported and used it
- & That import did not rely on any `#[doc(hidden)]` items
- & Now, the same import name no longer satisfies the above

Example: pub fn gets deleted

Find all functions such that:

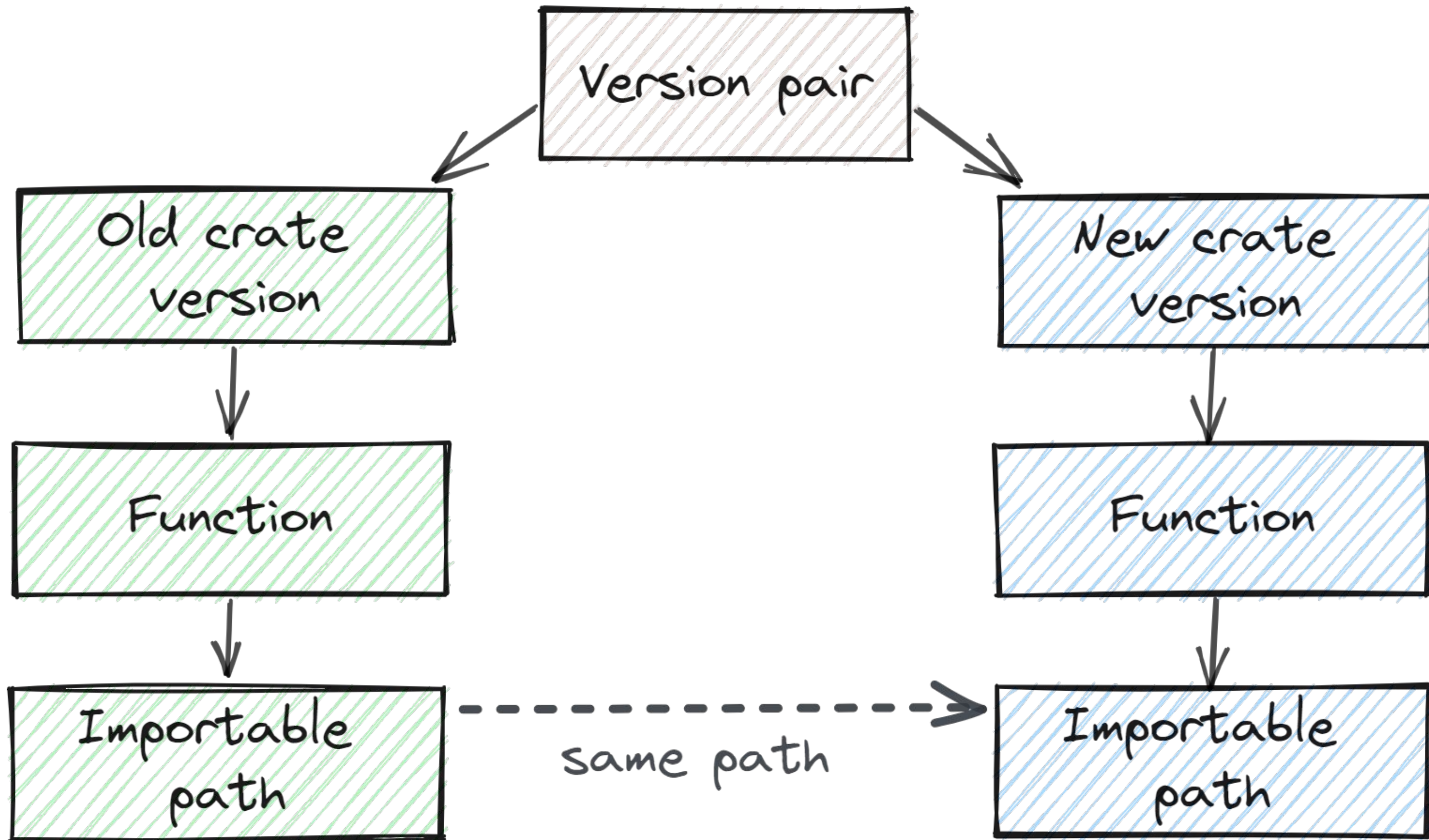
& Previously, the function was `pub`

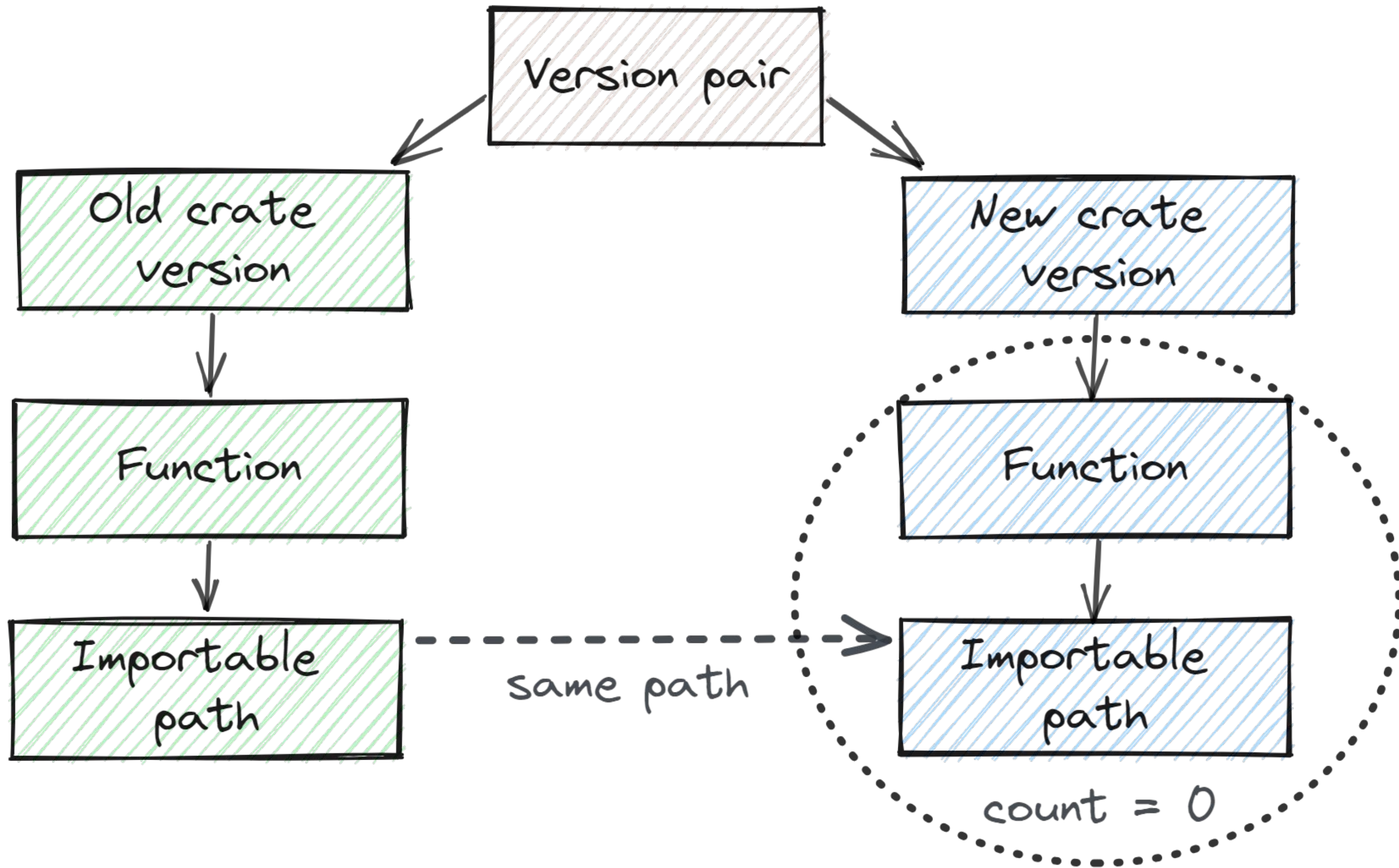
& Another crate could have imported and used it

& That import did not rely on any `#[doc(hidden)]` items

& Now, the same import name no longer satisfies the above

This sounds like a database query...





Database query!

Find all functions such that:

& Previously, the function was `pub`

& Another crate could have imported and used it

& That import did not rely on any `#[doc(hidden)]` items

& Now, the same import name no longer satisfies the above

This sounds like a database query...

Database query!

Find all functions such that:

& Previously, the function was `pub`

& Another crate could have imported and used it

& That import did not rely on any `#[doc(hidden)]` items

& Now, the same import name no longer satisfies the above

This is a database query!

```
{
  CrateDiff {
    baseline {
      item {
        ... on Function {
          visibility_limit @filter(op: "=", value: ["$public"])
          | | | | @output
          name @output

          importable_path {
            path @output @tag
            public_api @filter(op: "=", value: ["$true"])
          }
        }
      }
    }
    current @fold
    @transform(op: "count")
    @filter(op: "=", value: ["$zero"]) {
      item {
        ... on Function {
          visibility_limit @filter(op: "=", value: ["$public"])

          importable_path {
            path @filter(op: "=", value: ["%path"])
          }
        }
      }
    }
  }
}
```

Database query!

Find all functions such that:

& Previously, the function was `pub`

& Another crate could have imported and used it

& That import did not rely on any `#[doc(hidden)]` items

& Now, the same import name no longer satisfies the above

This is a database query!

```
{
  CrateDiff {
    baseline {
      item {
        ... on Function {
          visibility_limit @filter(op: "=", value: ["$public"])
          | @output
          name @output

          importable_path {
            path @output @tag
            public_api @filter(op: "=", value: ["$true"])
          }
        }
      }
    }
    current @fold
    @transform(op: "count")
    @filter(op: "=", value: ["$zero"]) {
      item {
        ... on Function {
          visibility_limit @filter(op: "=", value: ["$public"])

          importable_path {
            path @filter(op: "=", value: ["%path"])
          }
        }
      }
    }
  }
}
```

Database query!

Find all functions such that:

& Previously, the function was `pub`

& Another crate could have imported and used it

& That import did not rely on any `#[doc(hidden)]` items

& Now, the same import name no longer satisfies the above

This is a database query!

```
{
  CrateDiff {
    baseline {
      item {
        ... on Function {
          visibility_limit @filter(op: "=", value: ["$public"])
                              @output
          name @output

          importable_path {
            path @output @tag
            public_api @filter(op: "=", value: ["$true"])
          }
        }
      }
    }
    current @fold
      @transform(op: "count")
      @filter(op: "=", value: ["$zero"]) {
        item {
          ... on Function {
            visibility_limit @filter(op: "=", value: ["$public"])

            importable_path {
              path @filter(op: "=", value: ["%path"])
            }
          }
        }
      }
  }
}
```


Database query!

Find all functions such that:

& Previously, the function was `pub`

& Another crate could have imported and used it

& That import did not rely on any `#[doc(hidden)]` items

& Now, the same import name no longer satisfies the above

This is a database query!

```
{
  CrateDiff {
    baseline {
      item {
        ... on Function {
          visibility_limit @filter(op: "=", value: ["$public"])
                              @output
          name @output

          importable_path {
            path @output @tag
            public_api @filter(op: "=", value: ["$true"])
          }
        }
      }
    }
  }
  current @fold
  @transform(op: "count")
  @filter(op: "=", value: ["$zero"]) {
    item {
      ... on Function {
        visibility_limit @filter(op: "=", value: ["$public"])

        importable_path {
          path @filter(op: "=", value: ["%path"])
        }
      }
    }
  }
}
```

Database query!

Find all functions such that:

& Previously, the function was `pub`

& Another crate could have imported and used it

& That import did not rely on any `#[doc(hidden)]` items

& Now, the same import name no longer satisfies the above

This is a database query!

```
{
  CrateDiff {
    baseline {
      item {
        ... on Function {
          visibility_limit @filter(op: "=", value: ["$public"])
                              @output
          name @output
        }
      }
    }
    importable_path {
      path @output @tag
      public_api @filter(op: "=", value: ["$true"])
    }
  }
}

current @fold
@transform(op: "count")
@filter(op: "=", value: ["$zero"]) {
  item {
    ... on Function {
      visibility_limit @filter(op: "=", value: ["$public"])

      importable_path {
        path @filter(op: "=", value: ["%path"])
      }
    }
  }
}
```

Database query!

Find all functions such that:

& Previously, the function was `pub`

& Another crate could have imported and used it

& That import did not rely on any `#[doc(hidden)]` items

& Now, the same import name no longer satisfies the above

This is a database query!

```
{
  CrateDiff {
    baseline {
      item {
        ... on Function {
          visibility_limit @filter(op: "=", value: ["$public"])
                              @output
          name @output
        }
      }
    }
    importable_path {
      path @output @tag
      public_api @filter(op: "=", value: ["$true"])
    }
  }
}

current @fold
@transform(op: "count")
@filter(op: "=", value: ["$zero"]) {
  item {
    ... on Function {
      visibility_limit @filter(op: "=", value: ["$public"])

      importable_path {
        path @filter(op: "=", value: ["%path"])
      }
    }
  }
}
```

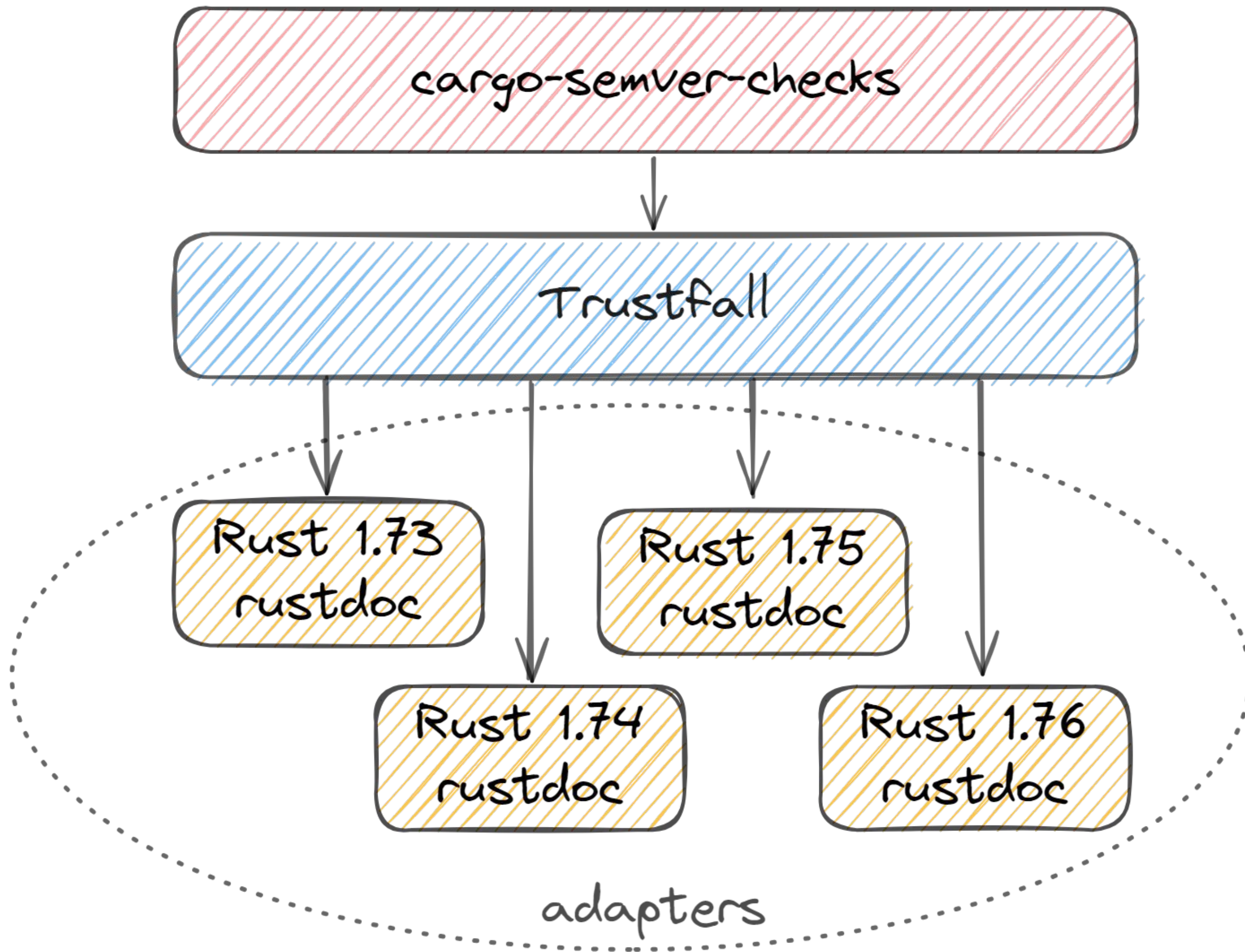

cargo-semver-checks

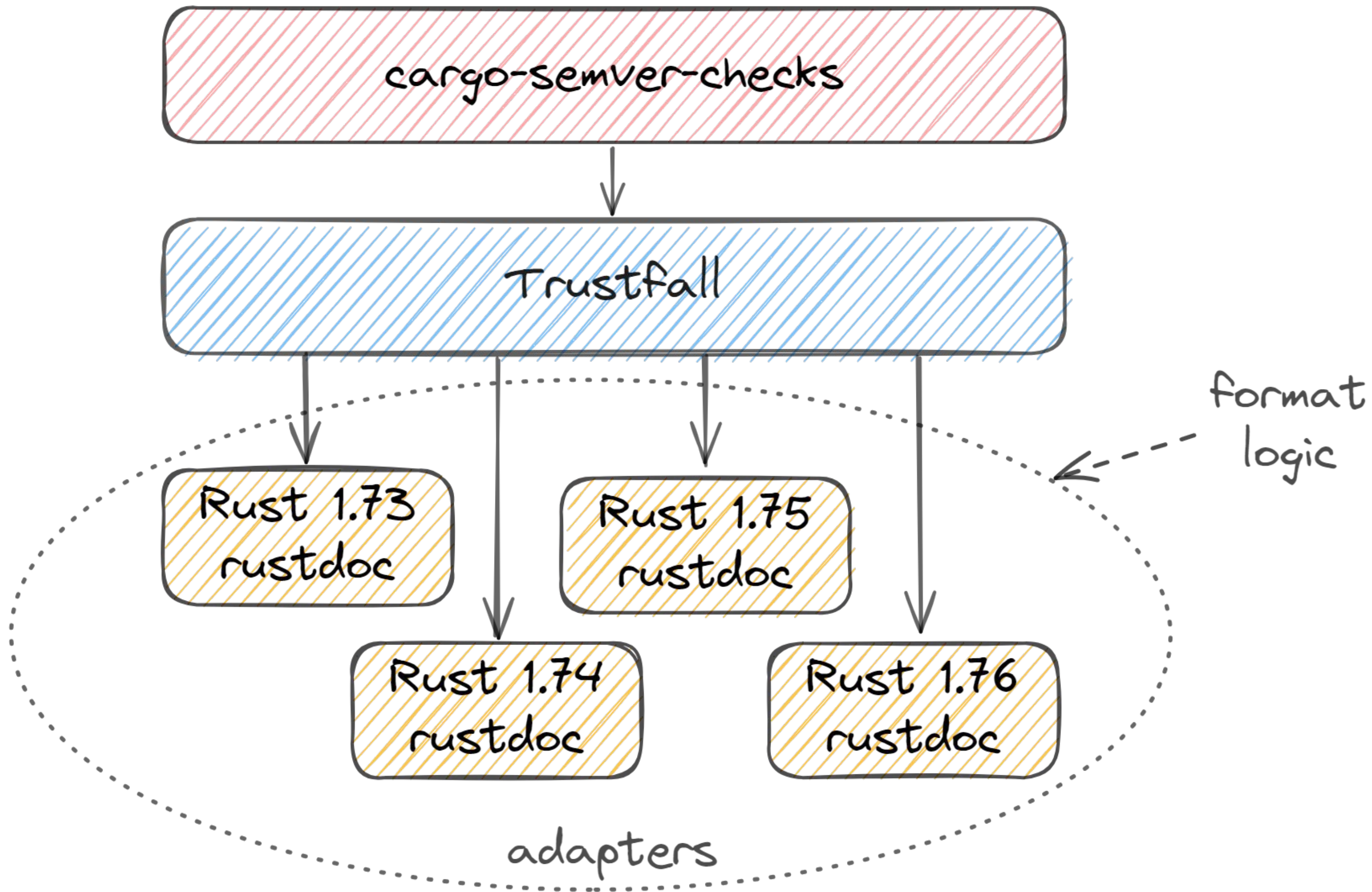
Rust 1.73
rustdoc

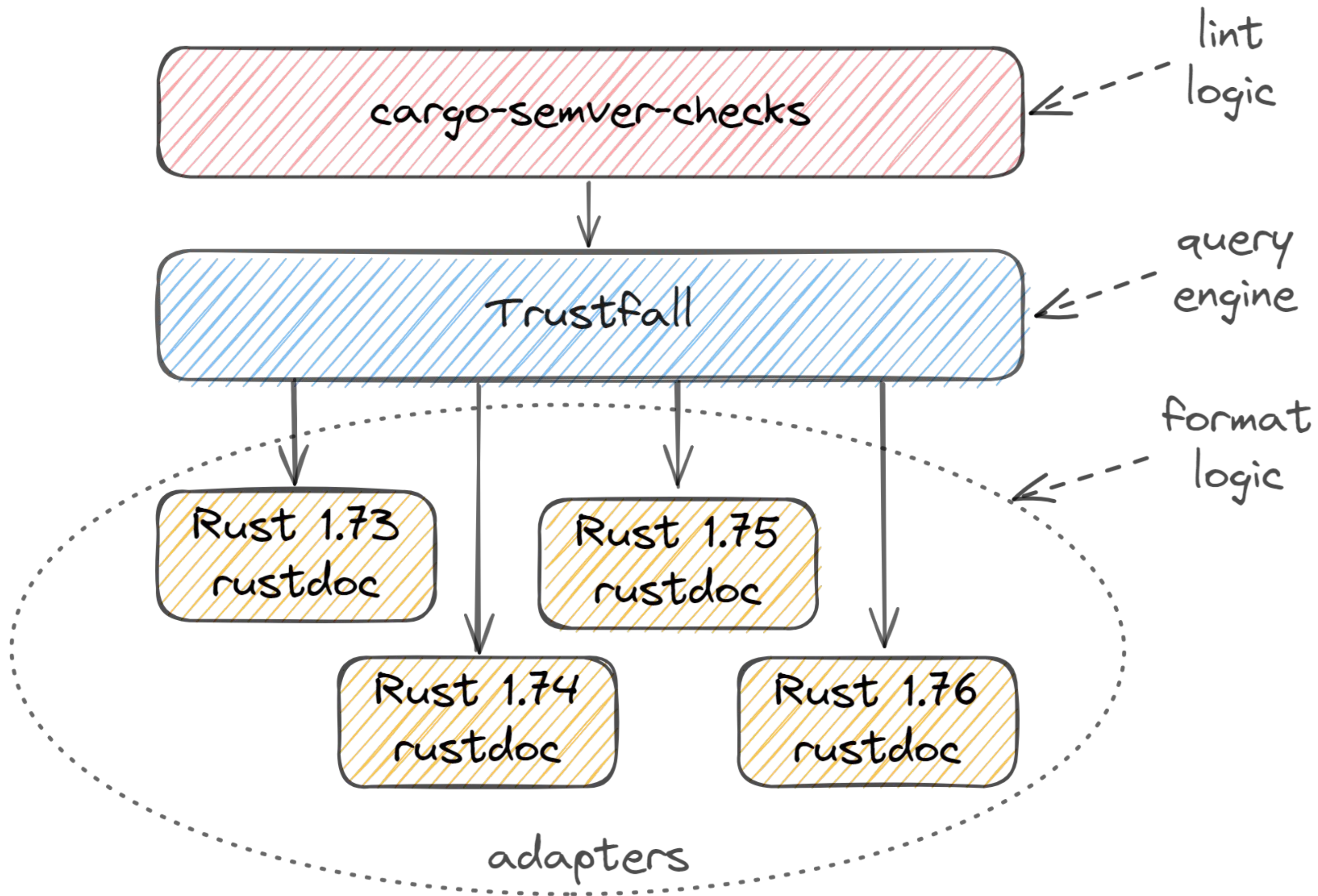
Rust 1.75
rustdoc

Rust 1.74
rustdoc

Rust 1.76
rustdoc







Trustfall: Turn everything into a database!

Represent data as a graph, then query any data sources

- Battle-tested: 7+ years in production
- Engine built in Rust; adapters can be Rust / Python / JS / WASM
- Query APIs, databases, arbitrary file formats – in-place & without ETL!

FOSS on GitHub: <https://github.com/obi1kenobi/trustfall>

Trustfall: Turn everything into a database!

Talks on Trustfall:

- “How to Query (Almost) Everything” – HYTRADBOI 2022
<https://www.hytradboi.com/2022/how-to-query-almost-everything>
- “How Database Tricks Sped up Rust Linting Over 2000x” – P99 CONF 2023
<https://www.youtube.com/watch?v=Fqo8r4bInsk>

Try Trustfall in our playgrounds:

- rustdoc JSON: <https://play.predr.ag/rustdoc>
- HackerNews REST APIs: <https://play.predr.ag/hackernews>

Trustfall makes cargo-semver-checks possible



Focus on linting & ergonomics, not rustdoc JSON format changes

Trustfall makes cargo-semver-checks possible



Focus on linting & ergonomics, not rustdoc JSON format changes

- 58 lints and growing – twice as many as a year ago

Trustfall makes cargo-semver-checks possible



Focus on linting & ergonomics, not rustdoc JSON format changes

- 58 lints and growing – twice as many as a year ago
- 32 contributors and growing – many new lints are first-time contributions!

Trustfall makes cargo-semver-checks possible



Focus on linting & ergonomics, not rustdoc JSON format changes

- 58 lints and growing – twice as many as a year ago
- 32 contributors and growing – many new lints are first-time contributions!
- Our users love us!



thomaseizinger commented on Nov 2, 2022

Contributor

Author



CI just caught an accidental breaking change! How good is this 🥰

<https://github.com/libp2p/rust-libp2p/actions/runs/3374939365/jobs/5601093378#step:8:21>

<https://github.com/libp2p/rust-libp2p/pull/3073#issuecomment-1299582893>

Toward fearless “cargo update”



SemVer is valuable, but impossible without automated help.
cargo-semver-checks is a solution with lots of happy users.

Toward fearless “cargo update”



SemVer is valuable, but impossible without automated help.
cargo-semver-checks is a solution with lots of happy users.

How you can help:

- Contribute code and lints to cargo-semver-checks

Toward fearless “cargo update”



SemVer is valuable, but impossible without automated help.
cargo-semver-checks is a solution with lots of happy users.

How you can help:

- Contribute code and lints to cargo-semver-checks
- Sponsor its development: <https://github.com/sponsors/obi1kenobi>

Toward fearless “cargo update”



SemVer is valuable, but impossible without automated help.
cargo-semver-checks is a solution with lots of happy users.

How you can help:

- Contribute code and lints to cargo-semver-checks
- Sponsor its development: <https://github.com/sponsors/obi1kenobi>
- Use cargo-semver-checks when others depend on your packages