

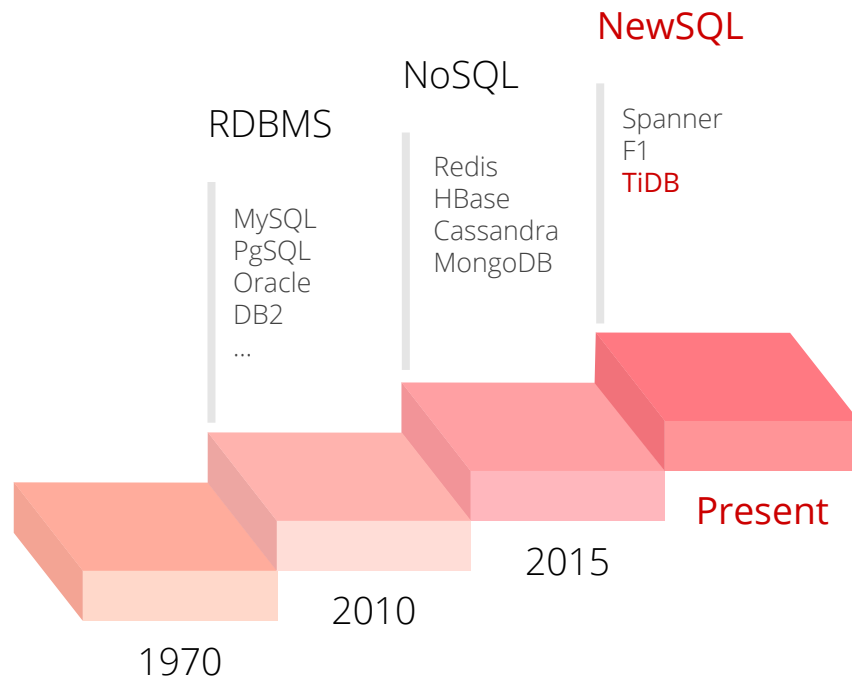
Rust Practice in TiKV

Wenxuan Shi

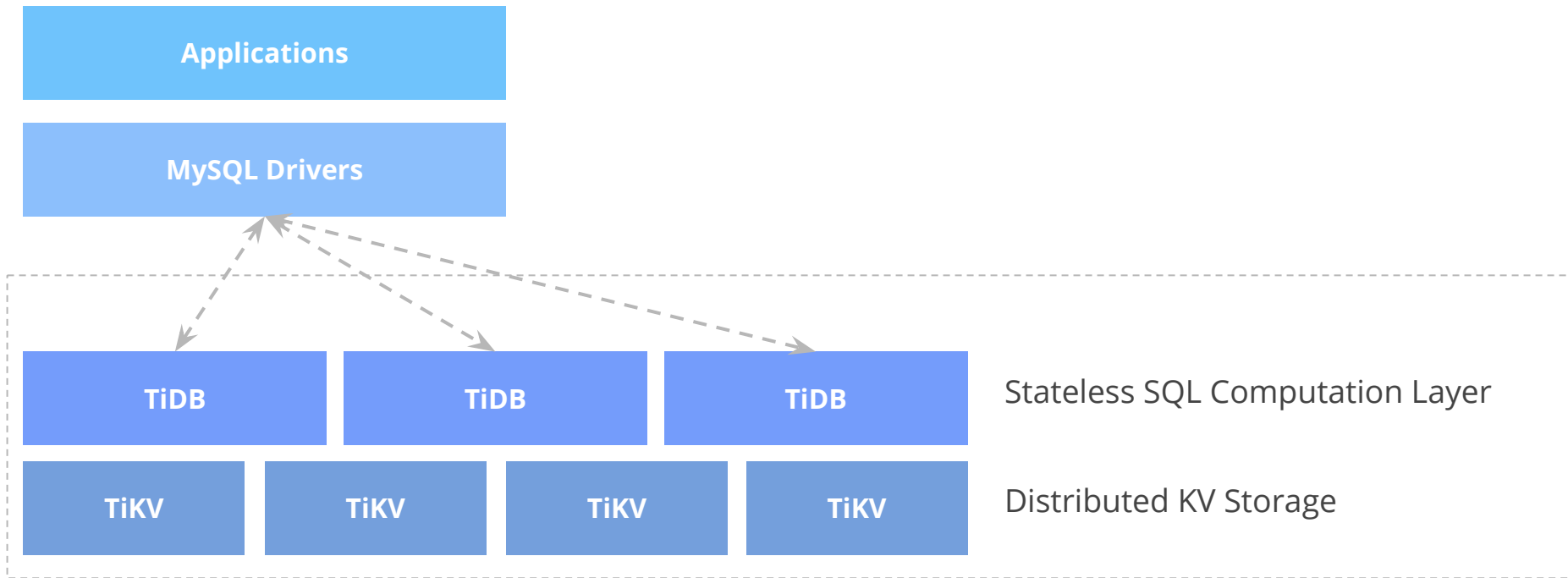


About Me & PingCAP

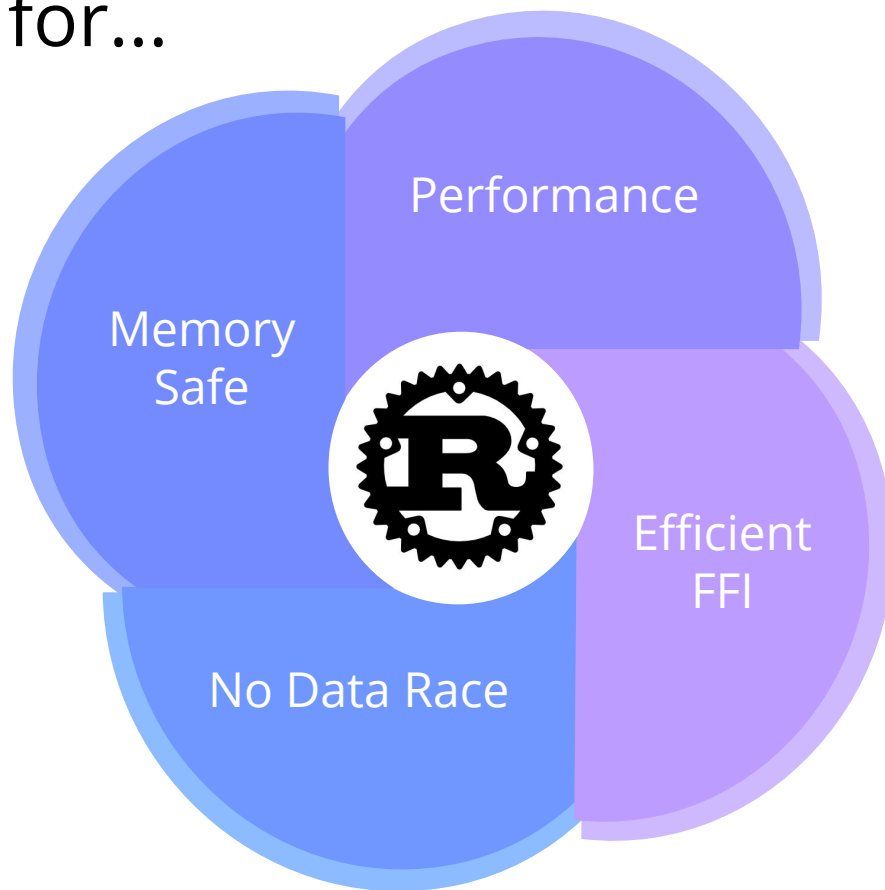
- TiKV Infrastructure Engineer
- HTAP & NewSQL Database
- CNCF
- Open source



Architecture



We Use Rust for...

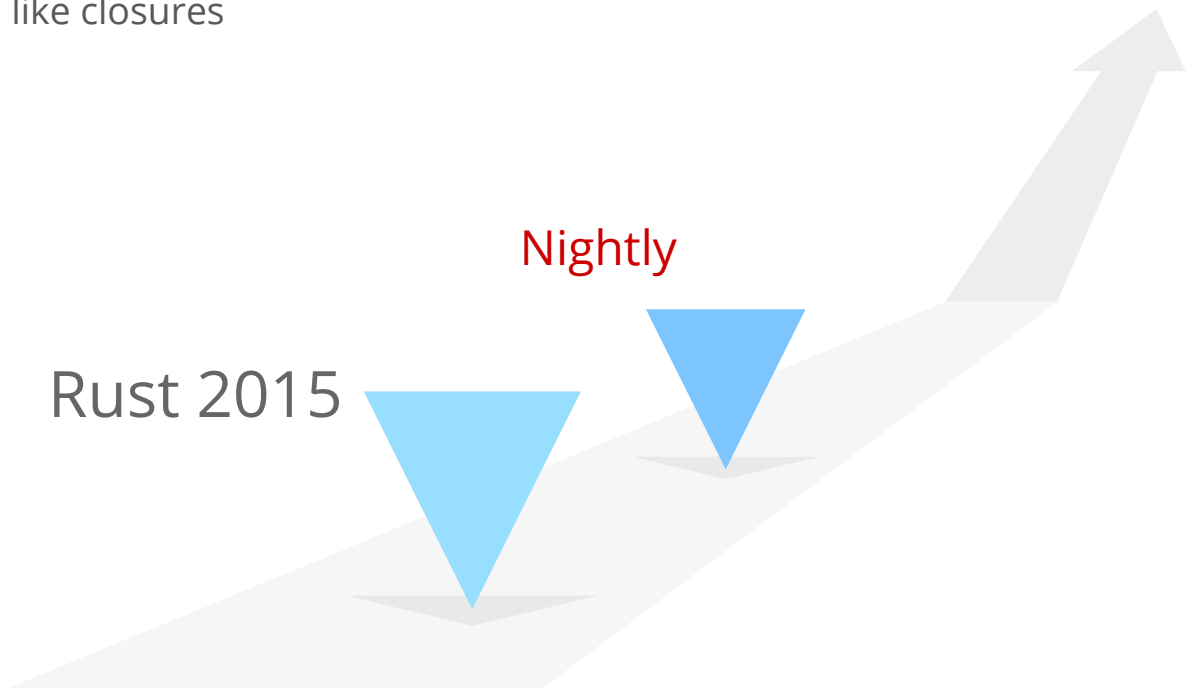


We Use Rust for...

Language	Performance	Coding Difficulty	Rigor	Safeness
C / C++	Very High	Difficult	Rigorous	No
Golang	Normal	Easy	Free	Some
Rust	Very High	Difficult	Very Rigorous	High

Rust Version

- Impl Trait (Rust 1.26)
 - Return unnameable types, like closures



Rust Version

```
1 #[inline]
2 fn async_snapshot(
3     engine: E,
4     ctx: &kvrpcpb::Context,
5 ) -> impl Future<Item = E::Snap, Error = Error> {
6     let (callback, future) = ::util::future::paired_future_callback();
7     let val = engine.async_snapshot(ctx, callback);
8     future::result(val)
9         .and_then(|_| ...)
10        .and_then(|_| ...)
11        .map_err(|_| ...)
12 }
```

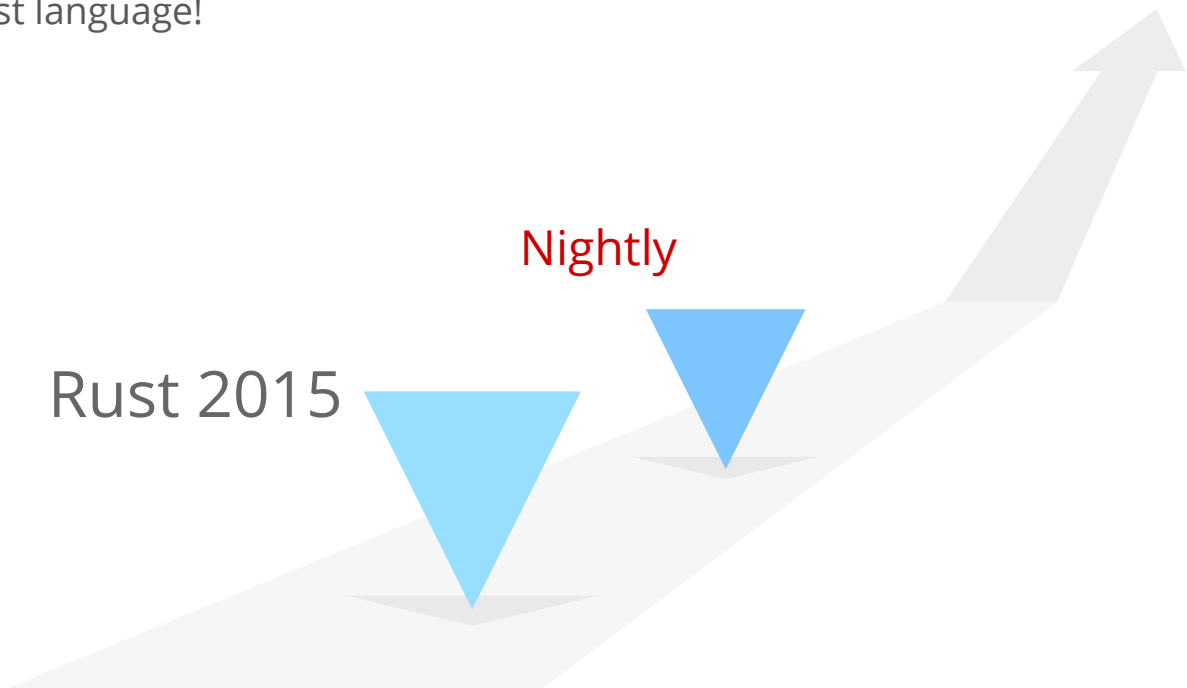
What's the type??

Rust Version

```
1 #[inline]
2 fn async_snapshot(
3     engine: E,                                I don't care!
4     ctx: &kvrpcpb::Context,
5 ) -> impl Future<Item = E::Snap, Error = Error> {
6     let (callback, future) = ::util::future::paired_future_callback();
7     let val = engine.async_snapshot(ctx, callback);
8     future::result(val)
9         .and_then(|_| ...)
10        .and_then(|_| ...)
11        .map_err(|_| ...)
12 }
```


Rust Version

- Procedural Macro / Custom Derive (Rust 1.30)
 - Meta programming via Rust language!



Rust Version

- Proc
- M

```
1 make_static_metric! {
2     pub struct ExampleStaticCounter: Counter {
3         "method" => {
4             post,
5             get,
6             put,
7             delete,
8         },
9         "test" => {
10            foo,
11            bar,
12        },
13        "server_ip" => {
14            loop_back: "127.0.0.1",
15            my_ip: MY_IP_ADDRESS,
16            proxy_ip,
17        },
18    }
19 }
```

1.30)

```
1 pub use self::pepmetheus_scoped_foo::ExampleStaticCounter;
2
3 #![allow(dead_code)]
4 #mod pepmetheus_scoped_foo {
5     #![allow(unused_imports)]
6     use super::*;
7     use prometheus::Counter;
8     use prometheus::CounterVec;
9     use std::collections::HashMap;
10
11     pub struct ExampleStaticCounter {
12         pub post: ExampleStaticCounter2,
13         pub get: ExampleStaticCounter,
14         pub put: ExampleStaticCounter,
15         pub delete: ExampleStaticCounter,
16     }
17
18     impl ExampleStaticCounter {
19         pub fn from(&counterVec) -> ExampleStaticCounter {
20             ExampleStaticCounter {
21                 post: ExampleStaticCounter2::from("post", m),
22                 get: ExampleStaticCounter2::from("get", m),
23                 put: ExampleStaticCounter2::from("put", m),
24                 delete: ExampleStaticCounter2::from("delete", m),
25             }
26         }
27
28         pub fn get_by_label(&self, label: &str) -> &ExampleStaticCounter2 {
29             match label {
30                 "post" => &self.post,
31                 "get" => &self.get,
32                 "put" => &self.put,
33                 "delete" => &self.delete,
34                 _ => panic!("unknown field"),
35             }
36         }
37     }
38
39     pub struct ExampleStaticCounter2 {
40         pub foo: ExampleStaticCounter,
41         pub bar: ExampleStaticCounter,
42     }
43
44     impl ExampleStaticCounter2 {
45         pub fn from(label_1: &str, m: &CounterVec) -> ExampleStaticCounter2 {
46             ExampleStaticCounter2 {
47                 foo: ExampleStaticCounter::from(label_1, "foo", m),
48                 bar: ExampleStaticCounter::from(label_1, "bar", m),
49             }
50         }
51
52         pub fn get_by_label(&self, label: &str) -> &ExampleStaticCounter3 {
53             match label {
54                 "foo" => &self.foo,
55                 "bar" => &self.bar,
56                 _ => panic!("unknown field"),
57             }
58         }
59     }
60
61     pub struct ExampleStaticCounter3 {
62         pub loop_back: Counter,
63         pub my_ip: Counter,
64         pub proxy_ip: Counter,
65     }
66
67     impl ExampleStaticCounter3 {
68         fn from(label_1: &str, label_2: &str, m: &CounterVec) -> ExampleStaticCounter3 {
69             ExampleStaticCounter3 {
70                 loop_back: m.with64(
71                     let mut collection = HashMap::new();
72                     collection.insert("method", label_1);
73                     collection.insert("test", label_2);
74                     collection.insert("server_ip", "127.0.0.1");
75                     collection
76                 ),
77                 my_ip: m.with64(
78                     let mut collection = HashMap::new();
79                     collection.insert("method", label_1);
80                     collection.insert("test", label_2);
81                     collection.insert("server_ip", MY_IP_ADDRESS);
82                     collection
83                 ),
84                 proxy_ip: m.with64(
85                     let mut collection = HashMap::new();
86                     collection.insert("method", label_1);
87                     collection.insert("test", label_2);
88                     collection.insert("server_ip", "proxy_ip");
89                     collection
90                 ),
91             }
92         }
93     }
94
95     pub fn get_by_label(&self, label: &str) -> &Counter {
96         match label {
97             "loop_back" => &self.loop_back,
98             "my_ip" => &self.my_ip,
99             "proxy_ip" => &self.proxy_ip,
100             _ => panic!("unknown field"),
101         }
102     }
103 }
```

Rust Vers

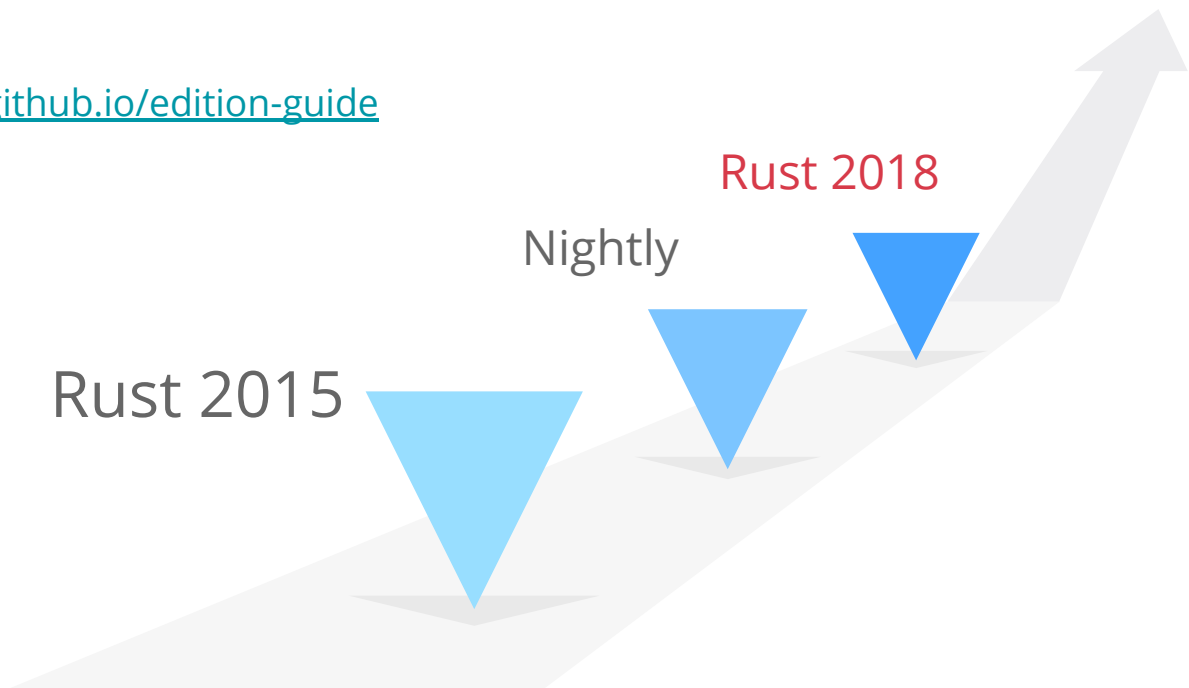
- Procedur
 - Meta p

```
1 pub use self::peometheus_scoped_foo::ExampleStaticCounter;
2
3 #[allow(dead_code)]
4 mod peometheus_scoped_foo {
5     #[allow(unused_imports)]
6     use super::*;
7     use prometheus::Counter;
8     use prometheus::CounterVec;
9     use std::collections::HashMap;
10
11     pub struct ExampleStaticCounter {
12         pub post: ExampleStaticCounter2,
13         pub get: ExampleStaticCounter2,
14         pub put: ExampleStaticCounter2,
15         pub delete: ExampleStaticCounter2,
16     }
17
18     impl ExampleStaticCounter {
19         pub fn from(m: &CounterVec) -> ExampleStaticCounter {
20             ExampleStaticCounter {
21                 post: ExampleStaticCounter2::from("post", m),
22                 get: ExampleStaticCounter2::from("get", m),
23                 put: ExampleStaticCounter2::from("put", m),
24                 delete: ExampleStaticCounter2::from("delete", m),
25             }
26         }
27
28         pub fn get_by_label(&self, label: &str) -> &ExampleStaticCounter2 {
29             match label {
30                 "post" => &self.post,
31                 "get" => &self.get,
32                 "put" => &self.put,
33                 "delete" => &self.delete,
34                 _ => panic!("unknown field"),
35             }
36         }
37     }
38 }
```

Rust Version

- We are moving forward to to Rust 2018!
- Learn more:

<https://rust-lang-nursery.github.io/edition-guide>



Code Style & Quality

- Formatter: rustfmt
<https://github.com/rust-lang/rustfmt>
- (Advanced) linter: rust-clippy
<https://github.com/rust-lang/rust-clippy>

Code Style & Quality

- Of course not all lint rules are suitable for us, like...

The ideal form

```
1 pub struct Foo {  
2     x: [f32; 3]  
3 }  
4  
5 pub fn foo(x: &mut Foo, p: &mut [f32]) {  
6     for i in 0..3 {  
7         x.x[i] += p[i];  
8     }  
9 }
```

Clippy suggests...

```
1 pub fn foo(x: &mut Foo, p: &mut [f32]) {  
2     for (i, p) in p.iter().enumerate().take(3) {  
3         x.x[i] += p;  
4     }  
5 }
```

Workspaces

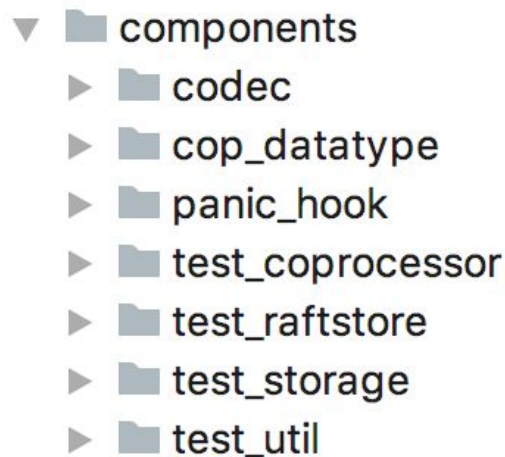
```
# Cargo.toml  
  
[workspace]  
  
members = [  
    "adder",  
    "add-one",  
]
```

Directory Structure

```
├── Cargo.lock  
├── Cargo.toml  
├── add-one  
│   ├── Cargo.toml  
│   └── src  
│       └── lib.rs  
├── adder  
│   ├── Cargo.toml  
│   └── src  
│       └── main.rs  
└── target
```

Workspaces

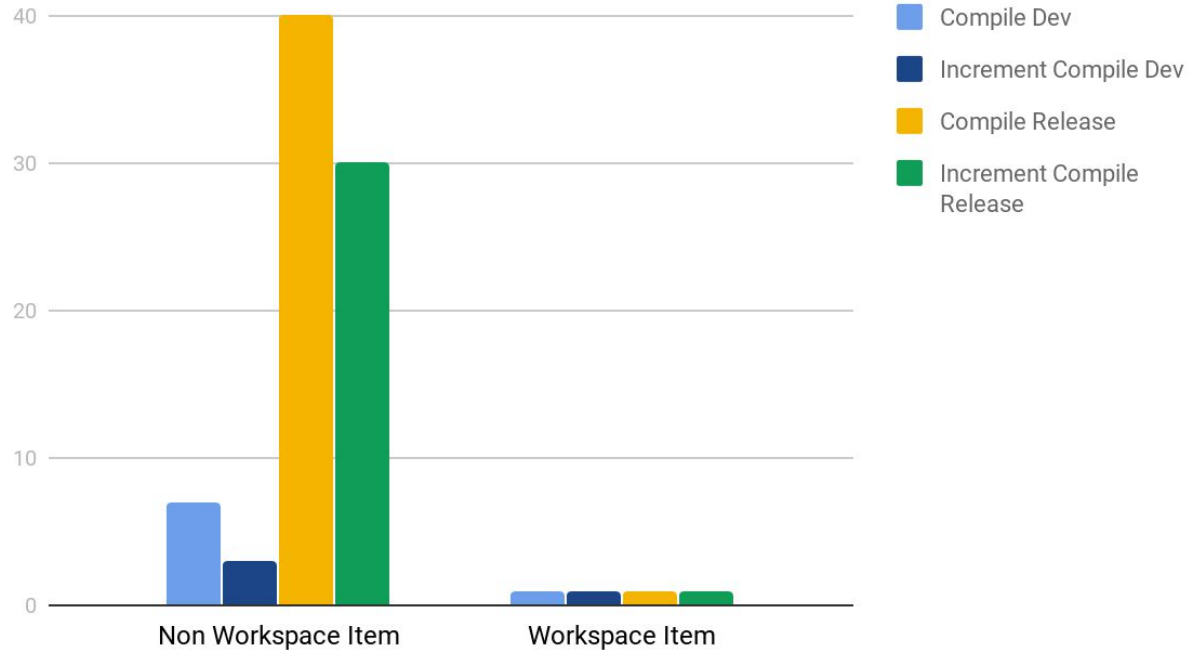
- More modular
- Speed up benchmark & test compiling
 - a. Prevent linking a large code base
 - b. Unchanged crates doesn't need to be recompiled
- Friendly to RLS



More coming!

Workspaces

minutes



Macros

- More DRY: Auto generate codes in some pattern
- Sometimes types and lifetimes are just hard to write :)
- Defects:
 - IDE unfriendly
 - Hard to debug when issues occurs inside macro
 - You can let compiler expand the macro

Macros

Standard way

```
1 let mut map = HashMap::new();
2 map.insert("a", 1);
3 map.insert("b", 2);
4 map.insert("c", 3);
```

Macro way

```
1 macro_rules! map {
2     ( $( $k:expr => $v:expr ),+ ) => {
3         {
4             let mut map = HashMap::default();
5             $(map.insert($k, $v);)+
6             map
7         }
8     };
9 }
```

```
1 let map = map!("a" => 1, "b" => 2, "c" => 3);
```

Procedural Macros

- Handling more complicated cases
- It have to be a standalone crate!
 - Use workspace to place it with other things together

Procedural Macros

```
1 make_static_metric! {
2     pub struct ExampleStaticCounter: Counter {
3         "method" => {
4             post,
5             get,
6             put,
7             delete,
8         },
9         "test" => {
10            foo,
11            bar,
12        },
13        "server_ip" => {
14            loop_back: "127.0.0.1",
15            my_ip: MY_IP_ADDRESS,
16            proxy_ip,
17        },
18    }
19 }
```



```
1 pub use self::ipemetheus_scoped_foo::ExampleStaticCounter;
2
3 #![allow(dead_code)]
4 #not prometheus_scoped foo {
5     #allow(unused_imports)}
6 use super::*;
7 use prometheus::Counter;
8 use prometheus::CounterVec;
9 use std::collections::HashMap;
10
11 pub struct ExampleStaticCounter {
12     pub post: ExampleStaticCounter2,
13     pub get: ExampleStaticCounter,
14     pub put: ExampleStaticCounter,
15     pub delete: ExampleStaticCounter,
16 }
17
18 impl ExampleStaticCounter {
19     pub fn from(&counterVec) -> ExampleStaticCounter {
20         ExampleStaticCounter {
21             post: ExampleStaticCounter2::from("post", m),
22             get: ExampleStaticCounter2::from("get", m),
23             put: ExampleStaticCounter2::from("put", m),
24             delete: ExampleStaticCounter2::from("delete", m),
25         }
26     }
27
28     pub fn get_by_label(&self, label: &str) -> &ExampleStaticCounter2 {
29         match label {
30             "post" => &self.post,
31             "get" => &self.get,
32             "put" => &self.put,
33             "delete" => &self.delete,
34             _ => panic!("unknown field"),
35         }
36     }
37 }
38
39 pub struct ExampleStaticCounter2 {
40     pub foo: ExampleStaticCounter,
41     pub bar: ExampleStaticCounter,
42 }
43
44 impl ExampleStaticCounter2 {
45     pub fn from(label_1: &str, m: &CounterVec) -> ExampleStaticCounter2 {
46         ExampleStaticCounter2 {
47             foo: ExampleStaticCounter3::from(label_1, "foo", m),
48             bar: ExampleStaticCounter3::from(label_1, "bar", m),
49         }
50     }
51
52     pub fn get_by_label(&self, label: &str) -> &ExampleStaticCounter3 {
53         match label {
54             "foo" => &self.foo,
55             "bar" => &self.bar,
56             _ => panic!("unknown field"),
57         }
58     }
59 }
60
61 pub struct ExampleStaticCounter3 {
62     pub loop_back: Counter,
63     pub my_ip: Counter,
64     pub proxy_ip: Counter,
65 }
66
67 impl ExampleStaticCounter3 {
68     fn from(label_1: &str, label_2: &str, m: &CounterVec) -> ExampleStaticCounter3 {
69         ExampleStaticCounter3 {
70             loop_back: m.with64(
71                 let mut collection = HashMap::new();
72                 collection.insert("method", label_1);
73                 collection.insert("test", label_2);
74                 collection.insert("server_ip", "127.0.0.1");
75                 collection
76             ),
77             my_ip: m.with64(
78                 let mut collection = HashMap::new();
79                 collection.insert("method", label_1);
80                 collection.insert("test", label_2);
81                 collection.insert("server_ip", MY_IP_ADDRESS);
82                 collection
83             ),
84             proxy_ip: m.with64(
85                 let mut collection = HashMap::new();
86                 collection.insert("method", label_1);
87                 collection.insert("test", label_2);
88                 collection.insert("server_ip", "proxy_ip");
89                 collection
90             ),
91         }
92     }
93
94     pub fn get_by_label(&self, label: &str) -> &Counter {
95         match label {
96             "loop_back" => &self.loop_back,
97             "my_ip" => &self.my_ip,
98             "proxy_ip" => &self.proxy_ip,
99             _ => panic!("unknown field"),
100         }
101     }
102 }
103 }
```

Unsafe

- Efficient Codec

```
1 pub fn encode_f64_to_comparable u64(v: f64) -> u64 {
2     let u: u64 = unsafe { ::std::mem::transmute(v) };
3     if v.is_sign_positive() {
4         u | SIGN_MARK
5     } else {
6         !u
7     }
8 }
```

Unsafe

- Fast Memory Access (without bounding check)

```
1 pub fn encode_var_u64(buf: &mut [u8], mut v: u64) -> usize {
2     assert!(buf.len() >= MAX_VARINT64_LENGTH);
3     unsafe {
4         let mut ptr = buf.as_mut_ptr();
5         while v >= 0x80 {
6             // We have already checked buffer size at the beginning so that it is safe to
7             // directly access the buffer.
8             *ptr = 0x80 | (v & 0x7f) as u8;
9             ptr = ptr.offset(1);
10            v >>= 7;
11        }
12        *ptr = v as u8;
13        ptr.offset_from(buf.as_ptr()) as usize + 1
14    }
15 }
```

Unsafe

- Fast Memory Access (without bounding check)

```
1 pub fn encode_var_u64(buf: &mut [u8], mut v: u64) -> usize {
2     assert!(buf.len() >= MAX_VARINT64_LENGTH);
3     unsafe {
4         let mut ptr = buf.as_mut_ptr();
5         while v >= 0x80 {
6             // We have already checked buffer size at the beginning so that it is safe to
7             // directly access the buffer.
8             *ptr = 0x80 | (v & 0x7f) as u8;
9             ptr = ptr.offset(1),
10            v >>= 7;
11        }
12        *ptr = v as u8;
13        ptr.offset_from(buf.as_ptr()) as usize + 1
14    }
15 }
```


Unsafe

- LLVM Ininsics

```
1 pub fn try_decode_var_u64(buf: &[u8]) -> Result<(u64, usize)> {
2     unsafe {
3         let len = buf.len();
4         if ::std::intrinsics::likely(len >= MAX_VARINT64_LENGTH) {
5             // Fast path
6             .....
7         } else {
8             // Slow path
9             .....
10        }
11    }
12 }
```

Generics & Zero Cost Abstraction

Solution 1. Trait Object

```
fn get(store: Box<Store>, key: &[u8]) {}
```

Raft Store

RocksDB Store

Mock Store

Generics & Zero Cost Abstraction

Solution 2. Generic
(efficient)

```
fn get<S: Store>(store: S, key: &[u8]) {}
```

Raft Store

RocksDB Store

Mock Store

Generics & Zero Cost Abstraction

Solution 3. Impl Trait
(efficient)

```
fn get(store: impl Store, key: &[u8]) {}
```

Raft Store

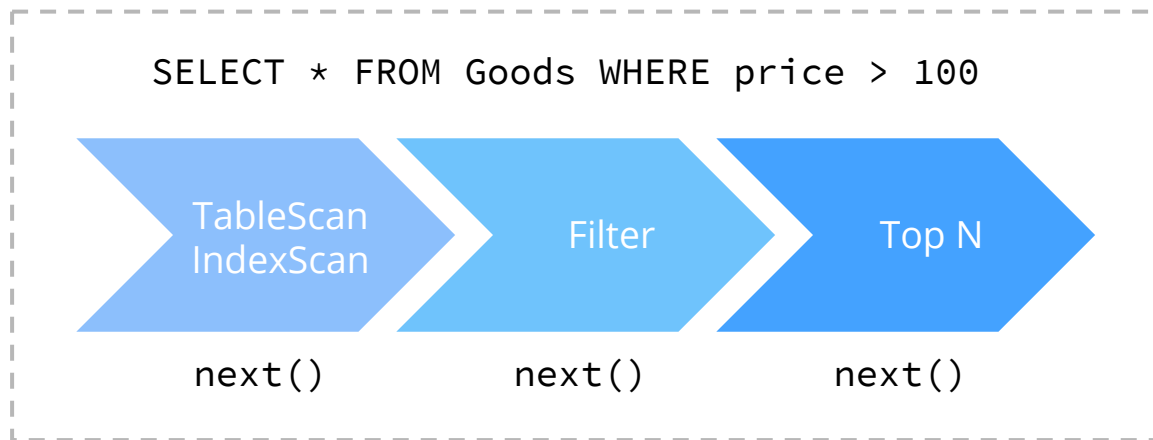
RocksDB Store

Mock Store

Generics & Zero Cost Abstraction

- Eliminate `Box<T>` in dynamic cases

```
trait Executor {  
  fn next(...);  
}
```



The combination is unknown at compile time

Generics & Zero Cost Abstraction

- Instead of JIT, we can...

```
struct TableScan;  
struct IndexScan;  
struct Filter<Src: Executor>;  
struct TopN<Src: Executor>;  
  
let common_pattern_1: TopN<Filter<TableScan>> = ...;  
let common_pattern_2: TopN<IndexScan> = ...;  
  
let uncommon_pattern: Box<Executor> = ...;
```

Generics & Zero Cost Abstraction

- Eliminate branches

```
1 fn decode_mem_comparable_bytes(  
2     src: &[u8],  
3     dest: &mut [u8],  
4     desc: bool,  
5 ) -> Result<(usize, usize)> {  
6     ...  
7     loop {  
8         let padding_size = if desc { 255 - ... } else { ... };  
9         ...  
10    }  
11 }
```

Generics & Zero Cost Abstraction

```
1 trait Helper {
2     fn parse_padding_size(...) -> usize;
3 }
4
5 struct AscendingHelper;
6 struct DescendingHelper;
7
8 impl Helper for AscendingHelper {
9     #[inline]
10    fn parse_padding_size(...) -> usize {
11        ...
12    }
13 }
14
15 impl Helper for DescendingHelper {
16     #[inline]
17    fn parse_padding_size(...) -> usize {
18        255 - ...
19    }
20 }
```

```
1 fn decode_mem_comparable_bytes_internal<T: Helper>(
2     src: &[u8],
3     dest: &mut [u8],
4     _helper: T,
5 ) -> Result<(usize, usize)> {
6     ...
7     loop {
8         let padding_size = T::parse_padding_size(...);
9         ...
10    }
11 }
```


Thank You !

We are hiring!

hire@pingcap.com

Contact me:

breezewish@pingcap.com

