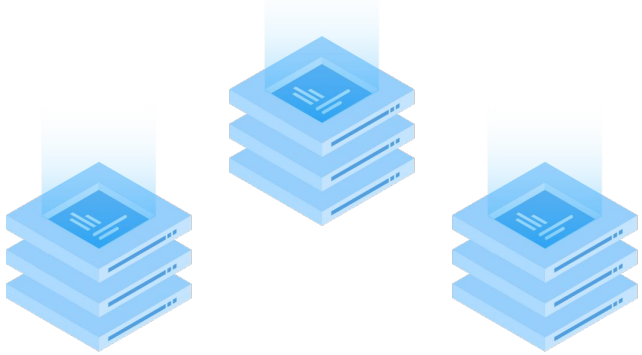
Rust Practice in TiKV

Wenxuan Shi

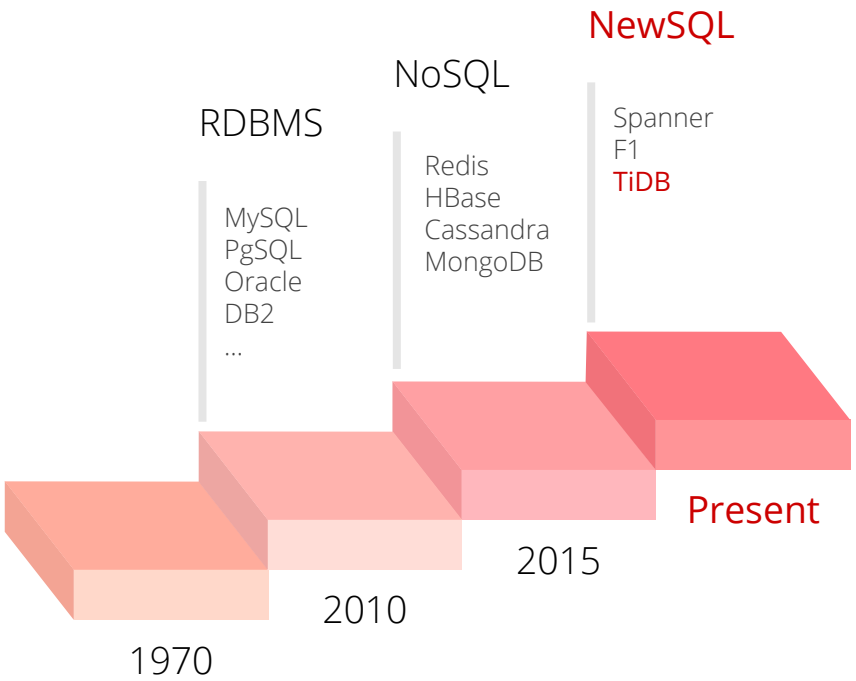
PingCAP TiDB



About Me & PingCAP

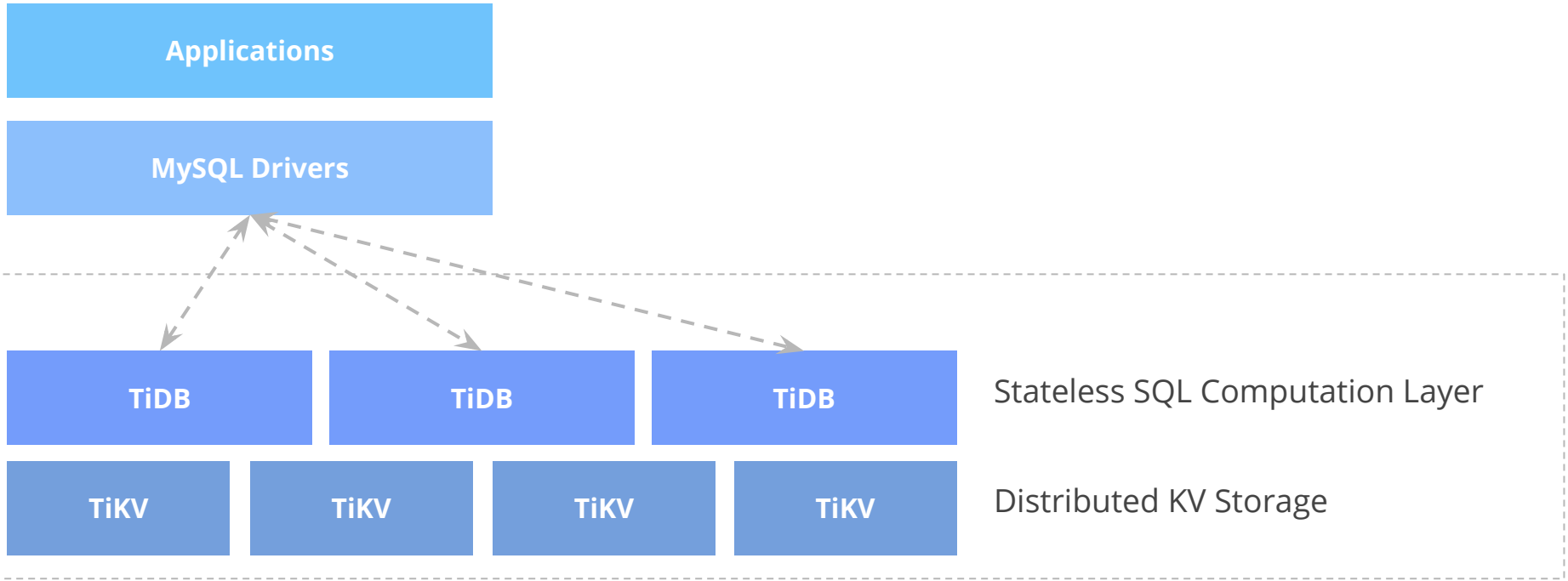
- TiKV Infrastructure Engineer

- HTAP & NewSQL Database
- CNCF
- Open source

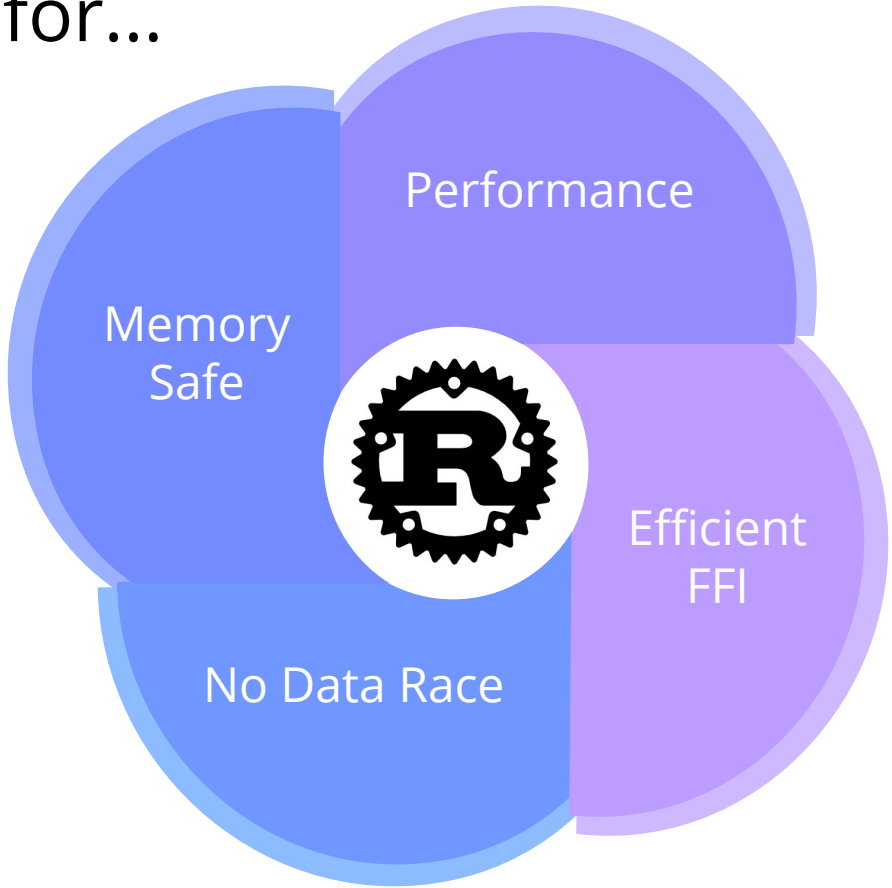


PingCAP

Architecture



We Use Rust for...

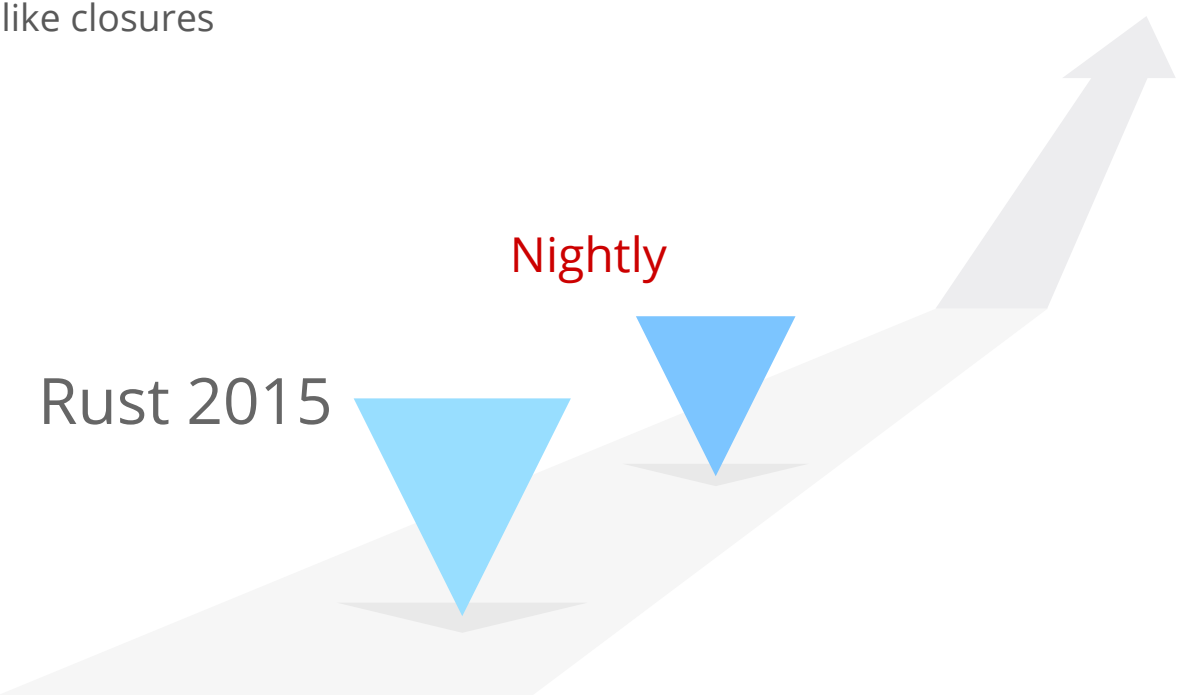


We Use Rust for...

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

```
#[inline]
fn async_snapshot(
    engine: E,
    ctx: &kvrpcpb::Context,
) -> impl Future<Item = E::Snap, Error = Error> {
    let (callback, future) = ::util::future::paired_future_callback();
    let val = engine.async_snapshot(ctx, callback);
    future::result(val)
        .and_then(|_| ...)
        .and_then(|_| ...)
        .map_err(|_| ...)
}
```

What's the type??

Rust Version

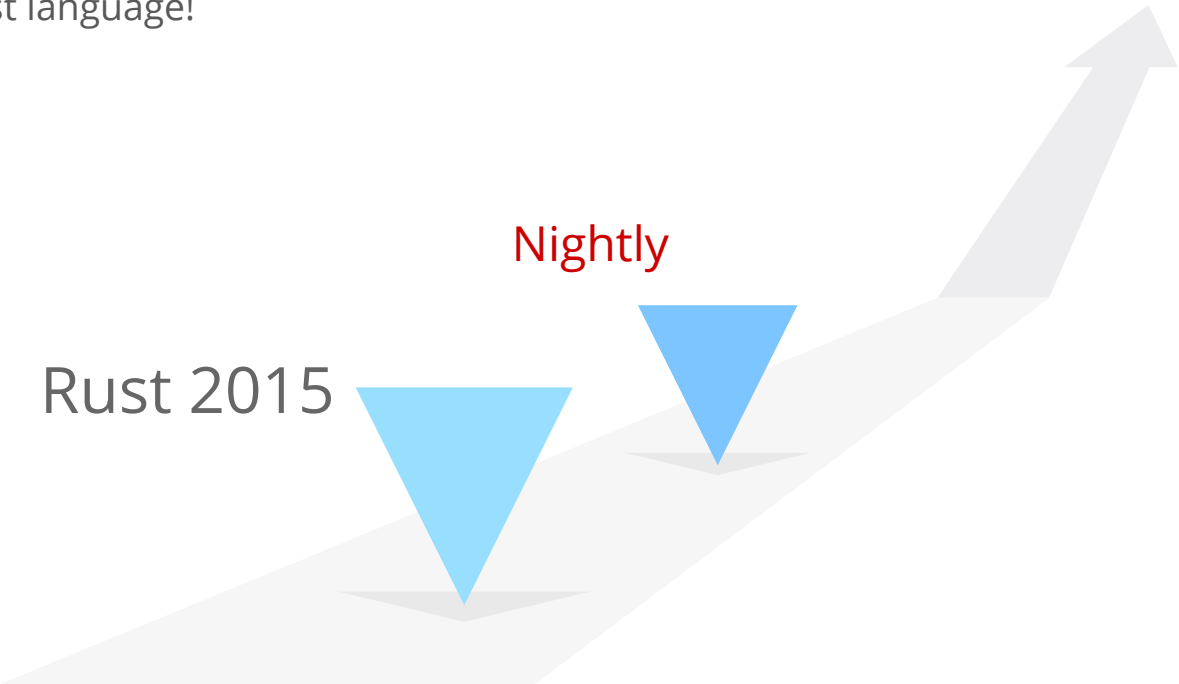
```
#[inline]
fn async_snapshot(
    engine: E,
    ctx: &kvrpcpb::Context,
) -> impl Future<Item = E::Snap, Error = Error> {
    let (callback, future) = ::util::future::paired_future_callback();
    let val = engine.async_snapshot(ctx, callback);
    future::result(val)
        .and_then(|_| ...)
        .and_then(|_| ...)
        .map_err(|_| ...)
}
```

I don't care!

PingCAP

Rust Version

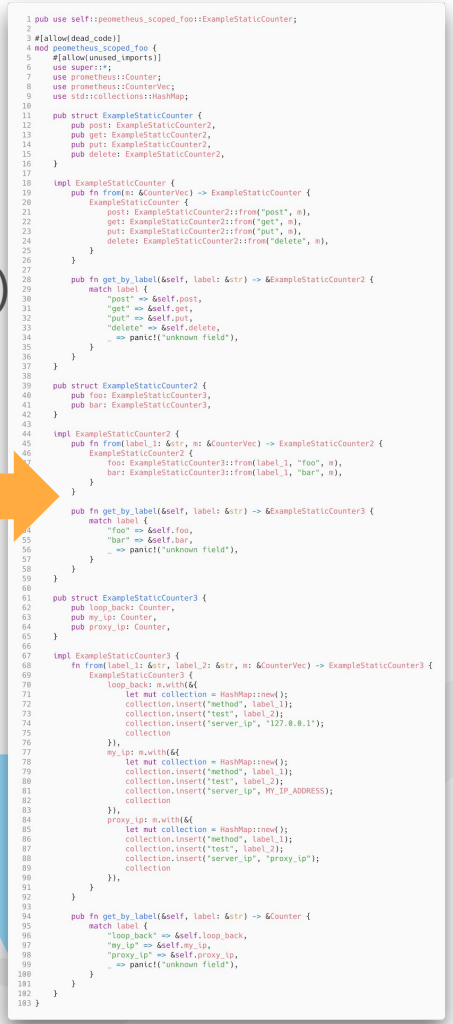
- Procedural Macro / Custom Derive (Rust 1.30)
  - Meta programming via Rust language!



Rust Version

- Proce 1.30)
  - M

```
make_static_metric! {
    pub struct ExampleStaticCounter: Counter {
        "method" => {
            post,
            get,
            put,
            delete,
        },
        "test" => {
            foo,
            bar,
        },
        "server_ip" => {
            loop_back: "127.0.0.1",
            my_ip: MY_IP_ADDRESS,
            proxy_ip,
        },
    }
}
```



PingCAP

Rust Vers

- Procedur
  - Meta p

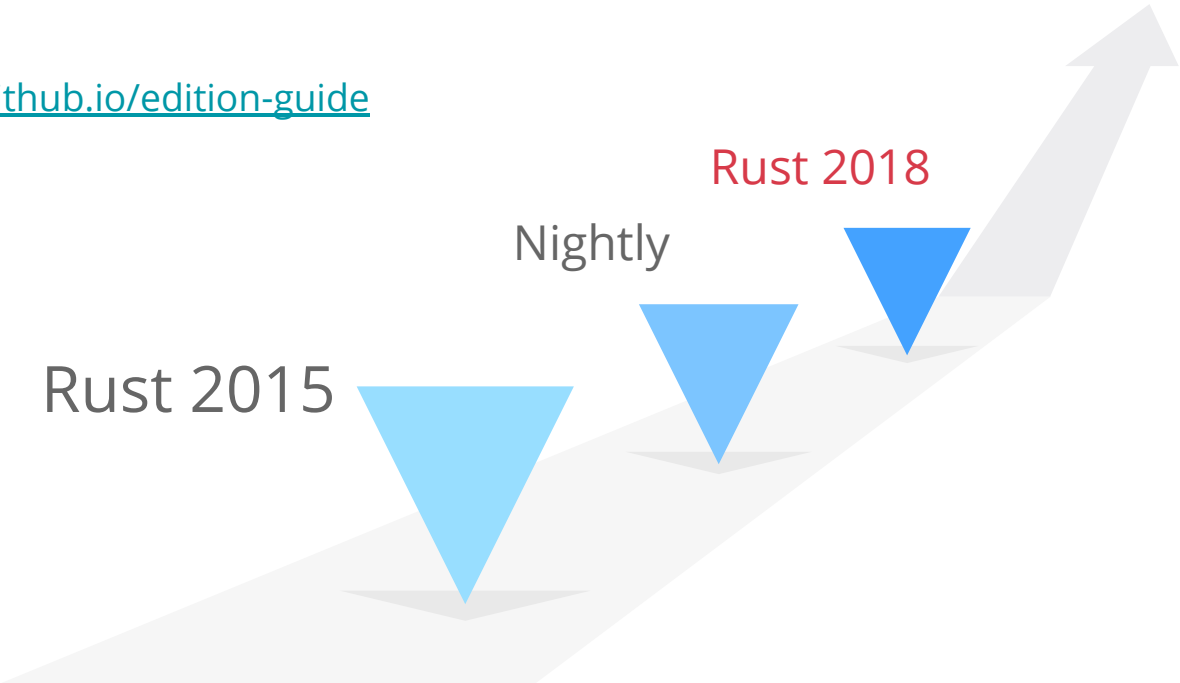
```rust
pub use self::peometheus_scoped_foo::ExampleStaticCounter;

#[allow(dead_code)]
mod peometheus_scoped_foo {
    #[allow(unused_imports)]
    use super::*;
    use prometheus::Counter;
    use prometheus::CounterVec;
    use std::collections::HashMap;

    pub struct ExampleStaticCounter {
        pub post: ExampleStaticCounter2,
        pub get: ExampleStaticCounter2,
        pub put: ExampleStaticCounter2,
        pub delete: ExampleStaticCounter2,
    }

    impl ExampleStaticCounter {
        pub fn from(m: &CounterVec) -> ExampleStaticCounter {
            ExampleStaticCounter {
                post: ExampleStaticCounter2::from("post", m),
                get: ExampleStaticCounter2::from("get", m),
                put: ExampleStaticCounter2::from("put", m),
                delete: ExampleStaticCounter2::from("delete", m),
            }
        }

        pub fn get_by_label(&self, label: &str) -> &ExampleStaticCounter2 {
            match label {
                "post" => &self.post,
                "get" => &self.get,
                "put" => &self.put,
                "delete" => &self.delete,
                _ => panic!("unknown field"),
            }
        }
    }

```

PingCAP

Rust Version

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

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



PingCAP

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

```rust
pub struct Foo {
    x: [f32; 3]
}

pub fn foo(x: &mut Foo, p: &mut [f32]) {
    for i in 0..3 {
        x.x[i] += p[i];
    }
}
```

Clippy suggests...

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

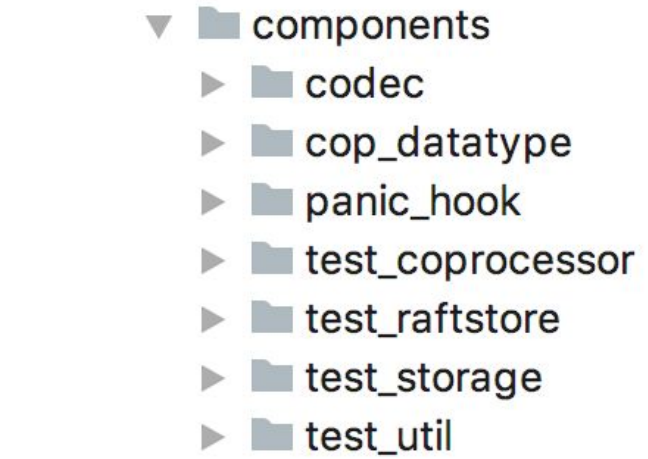
Workspaces

```toml
# 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
```

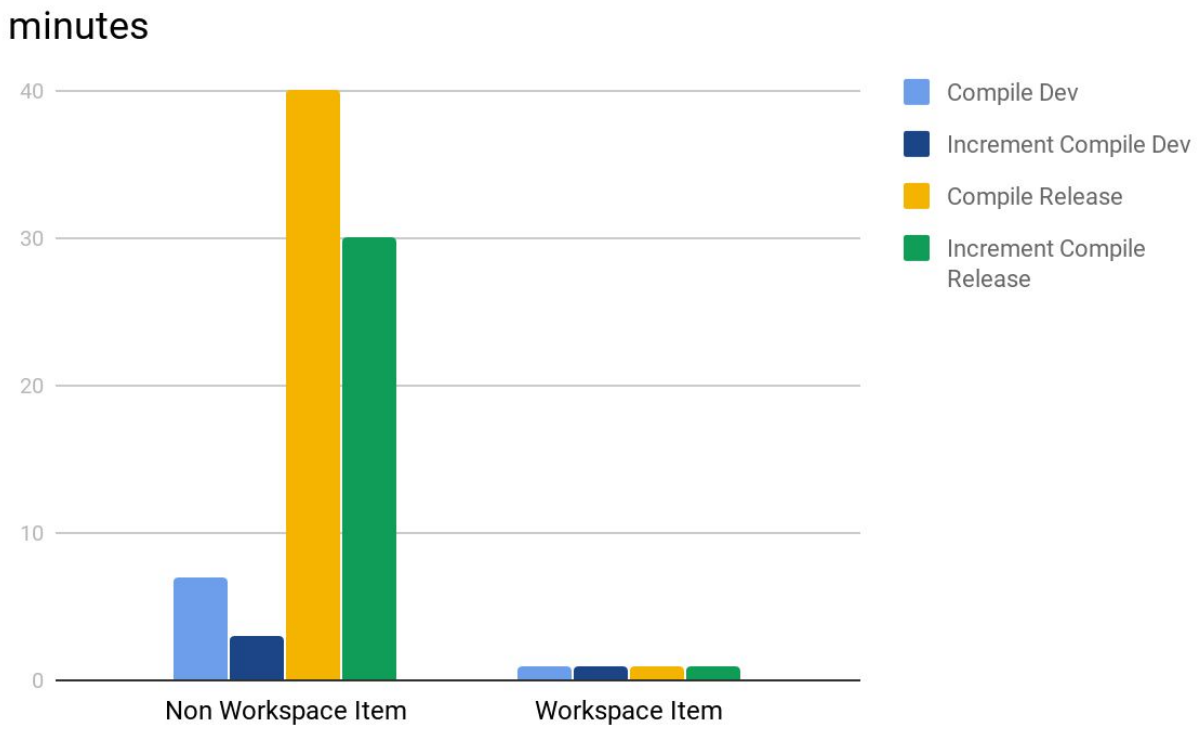
Workspaces

- 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



PingCAP

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

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

Macro way

```
macro_rules! map {
    ( $( $k:expr => $v:expr ),+ ) => {
        {
            let mut map = HashMap::default();
            $(map.insert($k, $v);)+
            map
        }
    };
}
```

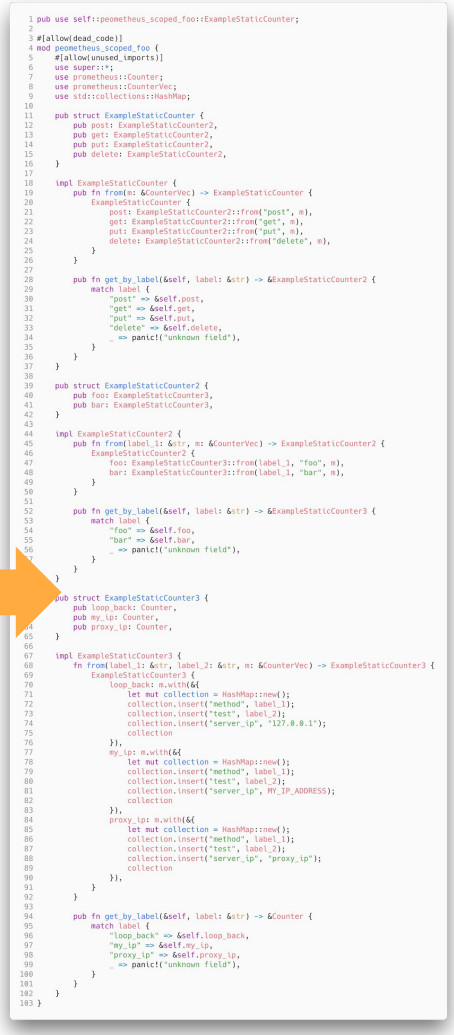
```
let map = map!("a" => 1, "b" => 2, "c" => 3);
```

Procedural Macros

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

Procedural Macros

```
make_static_metric! {
    pub struct ExampleStaticCounter: Counter {
        "method" => {
            post,
            get,
            put,
            delete,
        },
        "test" => {
            foo,
            bar,
        },
        "server_ip" => {
            loop_back: "127.0.0.1",
            my_ip: MY_IP_ADDRESS,
            proxy_ip,
        },
    }
}
```



PingCAP

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)

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

Unsafe

- Fast Memory Access (without bounding check)

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

PingCAP

Unsafe

- LLVM Intrinsics

```
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]) {}
```

Generics & Zero Cost Abstraction

Solution 2. Generic
(efficient)

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

Generics & Zero Cost Abstraction

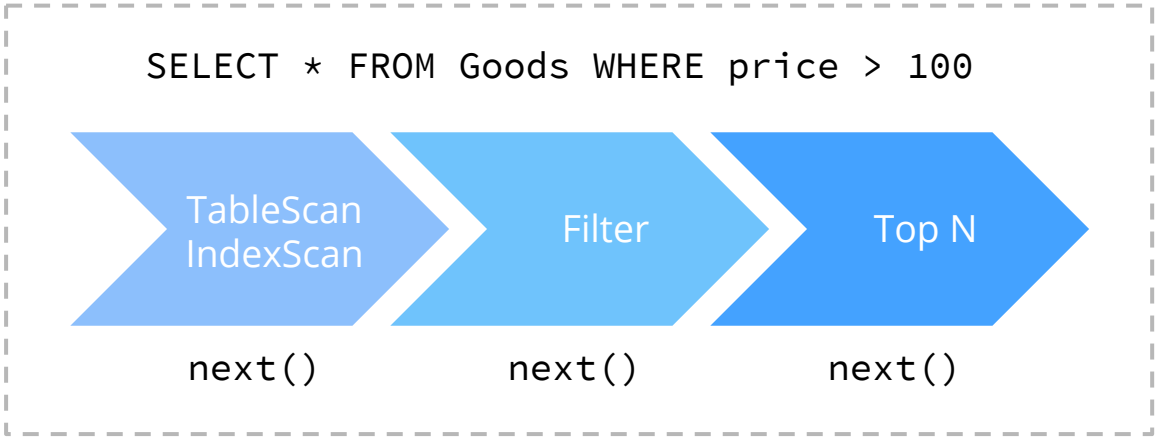
Solution 3. Impl Trait
(efficient)

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

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 }
```

PingCAP

Generics & Zero Cost Abstraction

```rust
trait Helper {
    fn parse_padding_size(...) -> usize;
}

struct AscendingHelper;
struct DescendingHelper;

impl Helper for AscendingHelper {
    #[inline]
    fn parse_padding_size(...) -> usize {
        ...
    }
}

impl Helper for DescendingHelper {
    #[inline]
    fn parse_padding_size(...) -> usize {
        255 - ...
    }
}
```

```rust
fn decode_mem_comparable_bytes_internal<T: Helper>(
    src: &[u8],
    dest: &mut [u8],
    _helper: T,
) -> Result<(usize, usize)> {
    ...
    loop {
        let padding_size = T::parse_padding_size(...);
        ...
    }
}
```

PingCAP

Thank You !

We are hiring!

hire@pingcap.com

Contact me:

breezewish@pingcap.com

PingCAP