Developing (experimental) Rust SDK and a Beam engine for IoT devices

Sho Nakatani
TOYOTA Motor Corporation
@laysakura (GitHub)

BEAM SUMMIT
NYC 2023

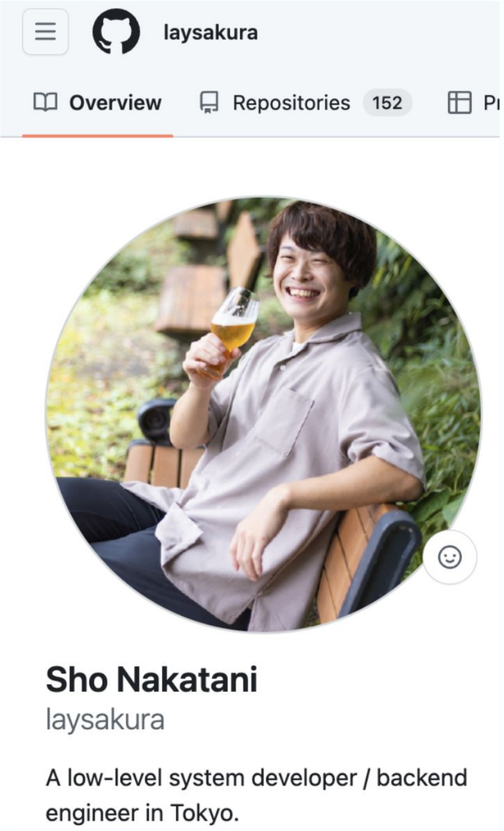
Goal

- About Beam Rust SDK: Make it the 5th Beam SDK
  - Sharing the motivation behind its development
  - Presenting the current status of the project
  - Encouraging collaboration and gathering contributors
- About SpringQL:
  - Providing a brief overview of SpringQL, a stream processor specifically designed for IoT devices

Beam, Rust, and Me

About Me

- Research and Development in stream processing for cloud and IoT devices
  - Implementing SpringQL in Rust (GitHub repo)
- Recognizing Beam as standard stream processing model for the next 10 years
  - Desire to support the Beam model for SpringQL
- Active involvement in the development of Beam Rust SDK since February 2023



Sho Nakatani
laysakura

A low-level system developer / backend engineer in Tokyo.

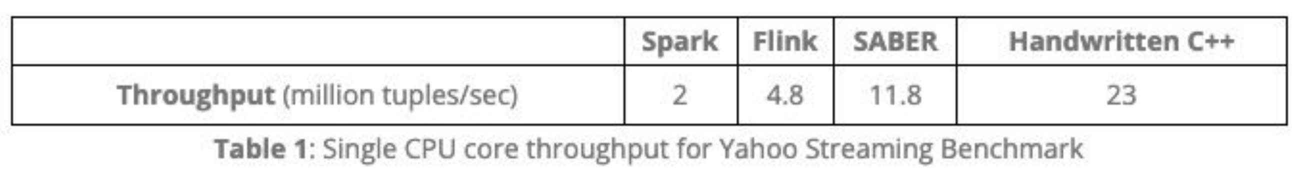
Agenda

- Rust SDK Development **(17 minutes)**
  - Motivation
  - Design
  - Rust-specific challenges
  - History and future prospects
- Introduction to SpringQL & Integration with Beam **(3 minutes)**

Rust SDK: Motivation

Motivation for Rust SDK

- For Pipeline Construction (or Programming)
  - Leveraging Rust's statically-typed nature and generics
  - Meeting the demand from Rustaceans for a dedicated Beam Rust SDK
- For Worker
  - Memory safety
  - Performant
    - Comparing to Go: More lightweight runtimes (e.g. no garbage collection)
    - (My interest) High performance single-node SPEs with Beam model?
      - Relevant Research: Scabbard, SABER/LightSaber, StreamBox
      - “Do We Need Distributed Stream Processing?” (blog post)
        - *"a single multicore server can provide better throughput than a multi-node cluster for many streaming applications"*



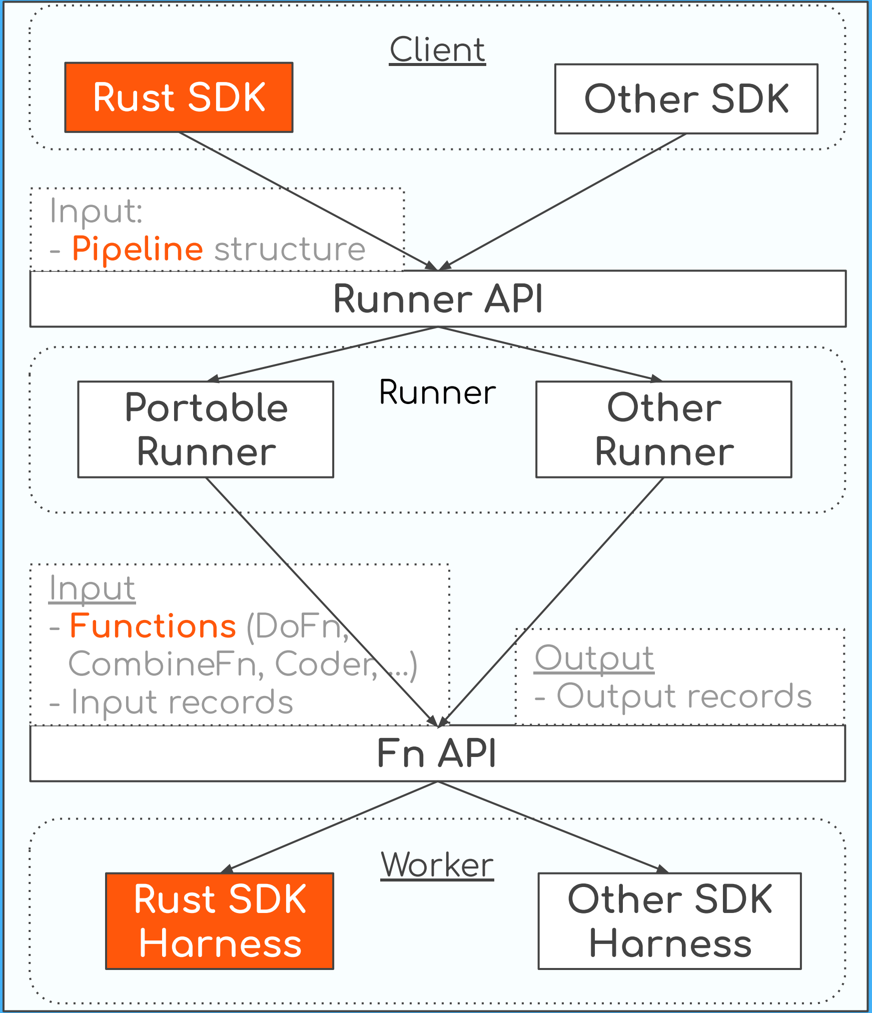
	Spark	Flink	SABER	Handwritten C++
Throughput (million tuples/sec)	2	4.8	11.8	23

Table 1: Single CPU core throughput for Yahoo Streaming Benchmark

Rust SDK: Design

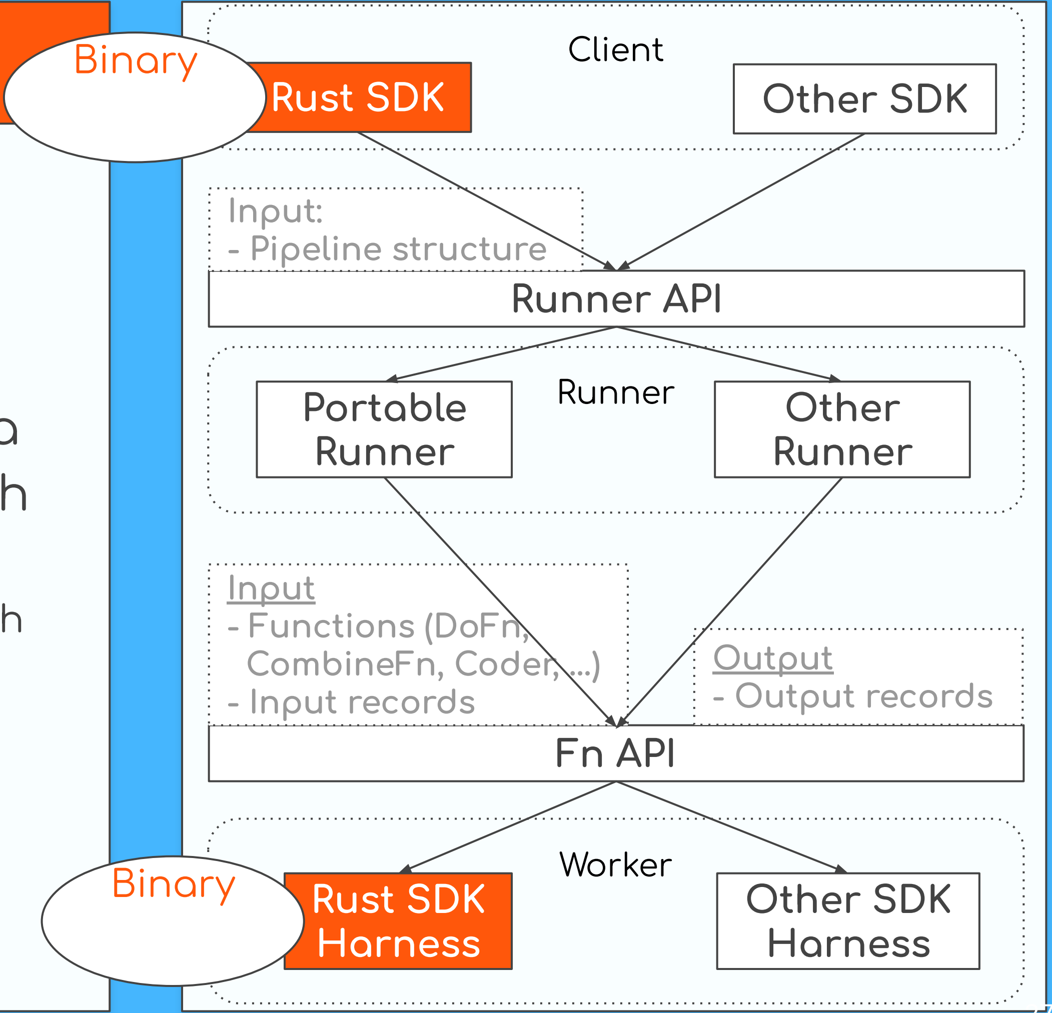
Where Rust SDK Works

- Rust SDK works in:
  - Client to construct pipelines
  - Workers to execute Rust-specific functions
- An application is built as a binary statically linked with the Rust SDK
  - Binaries are deployed to both Client and Workers
  - Different binaries are built from the same app (source)



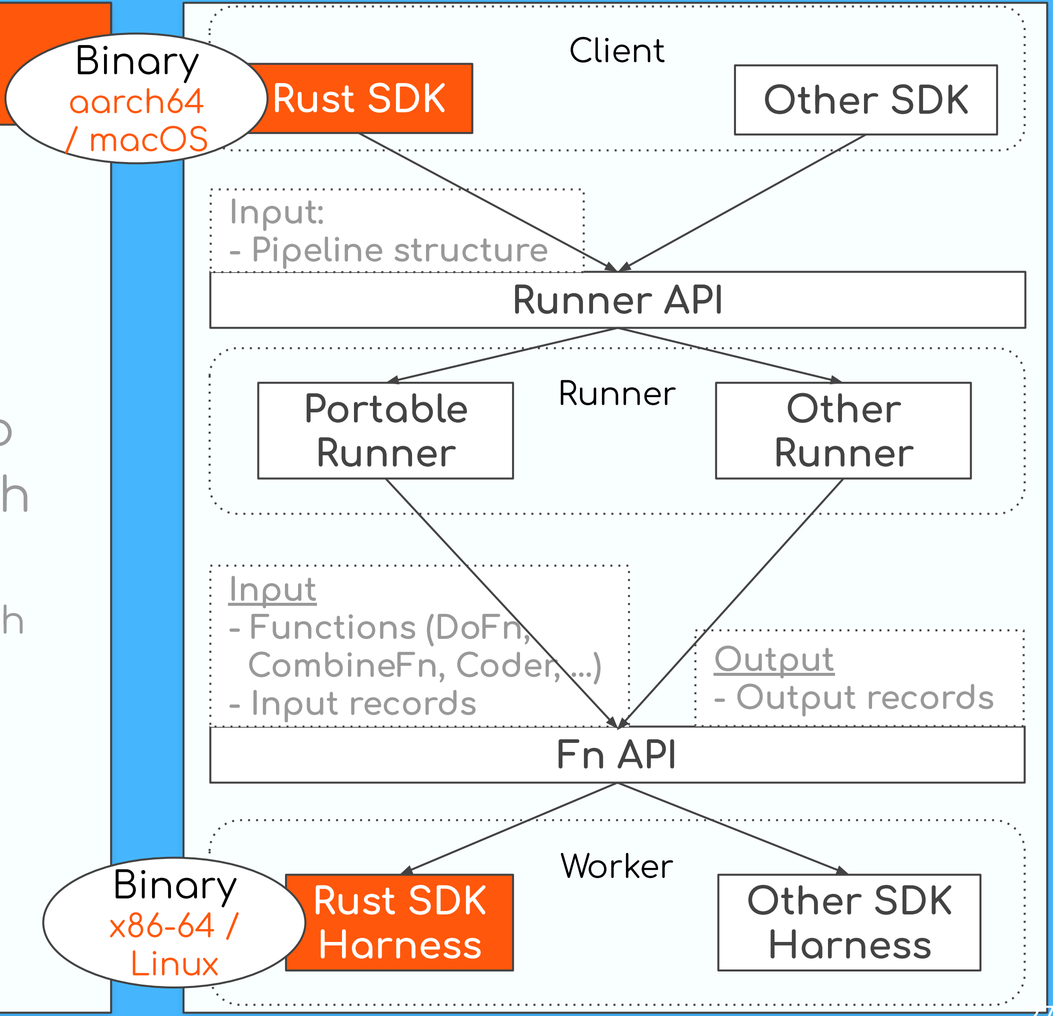
Where Rust SDK Works

- Rust SDK works in:
  - Client to construct pipelines
  - Workers to execute Rust-specific functions
- An application is built as a binary statically linked with the Rust SDK
  - Binaries are deployed to both Client and Workers
  - Different binaries are built from the same app (source)



Where Rust SDK Works

- Rust SDK works in:
  - Client to construct pipelines
  - Workers to execute Rust-specific functions
- An application is built into binary statically linked with Rust SDK
  - Binaries are deployed to both Client and Workers
  - Different binaries are built from the same app (source)



Design Concepts

- Mainly influenced by TypeScript (features) and Go (compilation & deployment)
- Statically-typed pipeline construction
- Removal of Pipeline APIs (explained later)
- Asynchronous execution of workers

Note: The design concepts may require further synchronization with other contributors.

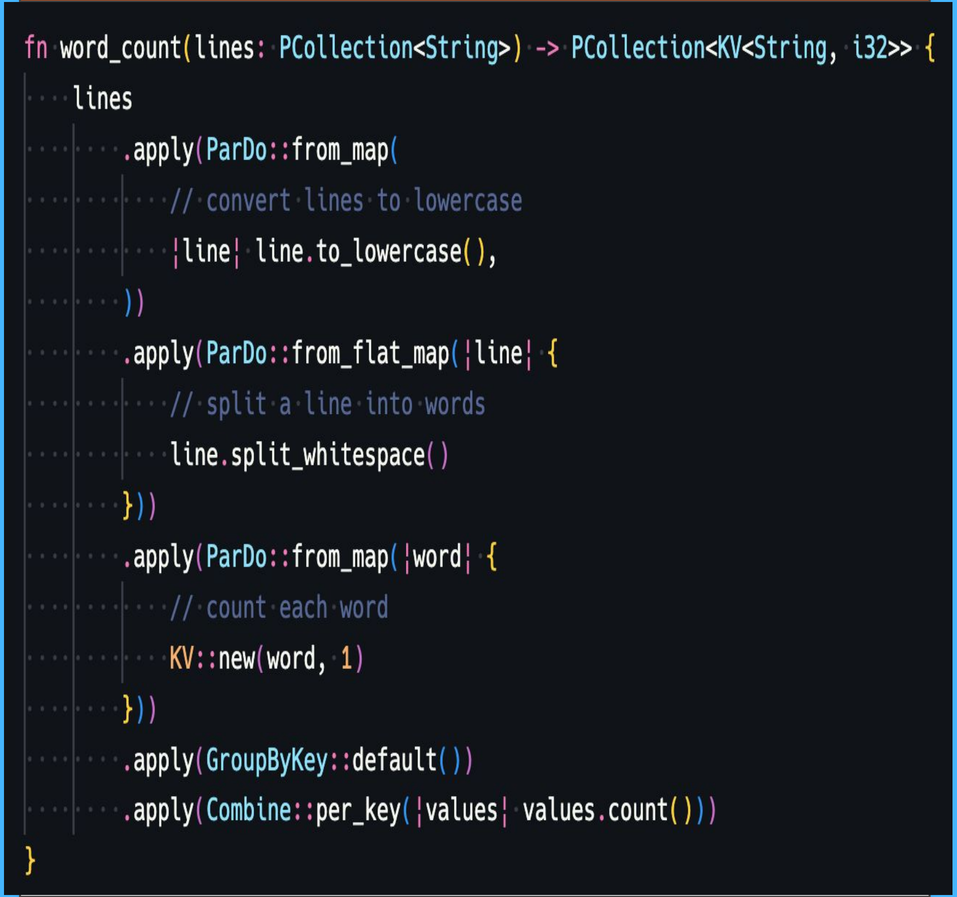
Design Concepts

- Mainly influenced by TypeScript (features) and Go (compilation & deployment)
- Statically-typed pipeline construction
- Removal of Pipeline APIs
- Asynchronous execution of workers

Note: The design concepts may require further synchronization with other contributors.

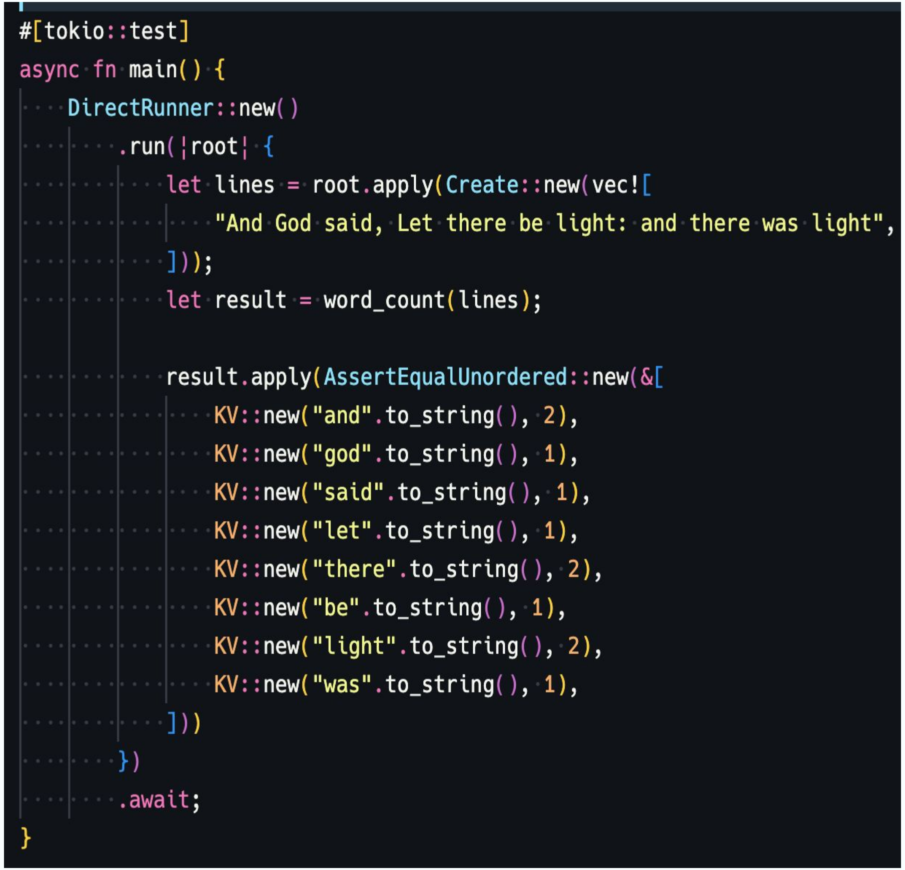
Show the concepts via a word-count pipeline

Word-count pipeline...



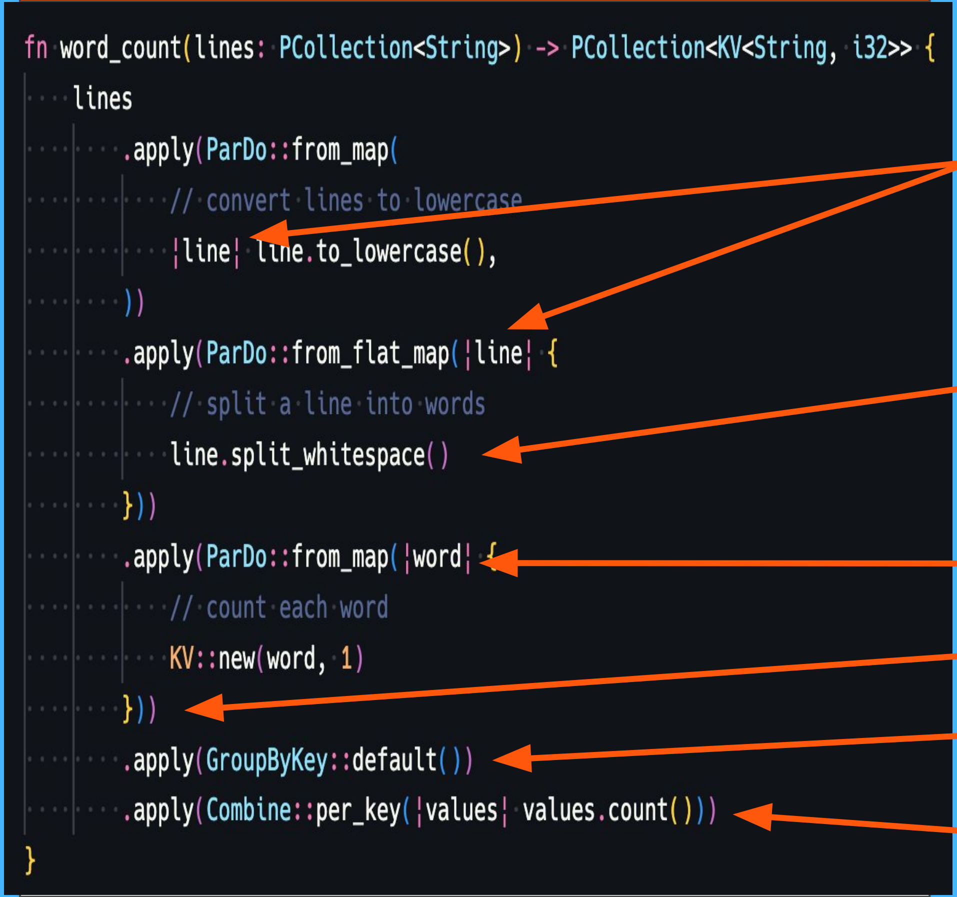
```rust
fn word_count(lines: PCollection<String>) -> PCollection<KV<String, i32>> {
    lines
        .apply(ParDo::from_map(
            // convert lines to lowercase
            |line| line.to_lowercase(),
        ))
        .apply(ParDo::from_flat_map(|line| {
            // split a line into words
            line.split_whitespace()
        }))
        .apply(ParDo::from_map(|word| {
            // count each word
            KV::new(word, 1)
        }))
        .apply(GroupByKey::default())
        .apply(Combine::per_key(|values| values.count()))
}
```

and its usage from DirectRunner



```rust
#[tokio::test]
async fn main() {
    DirectRunner::new()
        .run(|root| {
            let lines = root.apply(Create::new(vec![
                "And God said, Let there be light: and there was light",
            ]));
            let result = word_count(lines);

            result.apply(AssertEqualUnordered::new(&[
                KV::new("and".to_string(), 2),
                KV::new("god".to_string(), 1),
                KV::new("said".to_string(), 1),
                KV::new("let".to_string(), 1),
                KV::new("there".to_string(), 2),
                KV::new("be".to_string(), 1),
                KV::new("light".to_string(), 2),
                KV::new("was".to_string(), 1),
            ]))
        })
        .await;
}
```

Word-count pipeline…



```
fn word_count(lines: PCollection<String>) -> PCollection<KV<String, i32>> {
    lines
        .apply(ParDo::from_map(
            // convert lines to lowercase
            |line| line.to_lowercase(),
        ))
        .apply(ParDo::from_flat_map(|line| {
            // split a line into words
            line.split_whitespace()
        }))
        .apply(ParDo::from_map(|word| {
            // count each word
            KV::new(word, 1)
        }))
        .apply(GroupByKey::default())
        .apply(Combine::per_key(|values| values.count()))
}
```

and its usage from DirectRunner

Statically-typed (w/ automatic type-inference)

line: **String**

line.split_whitespace(): **Vec<String>**
→ flat-mapped into **String**

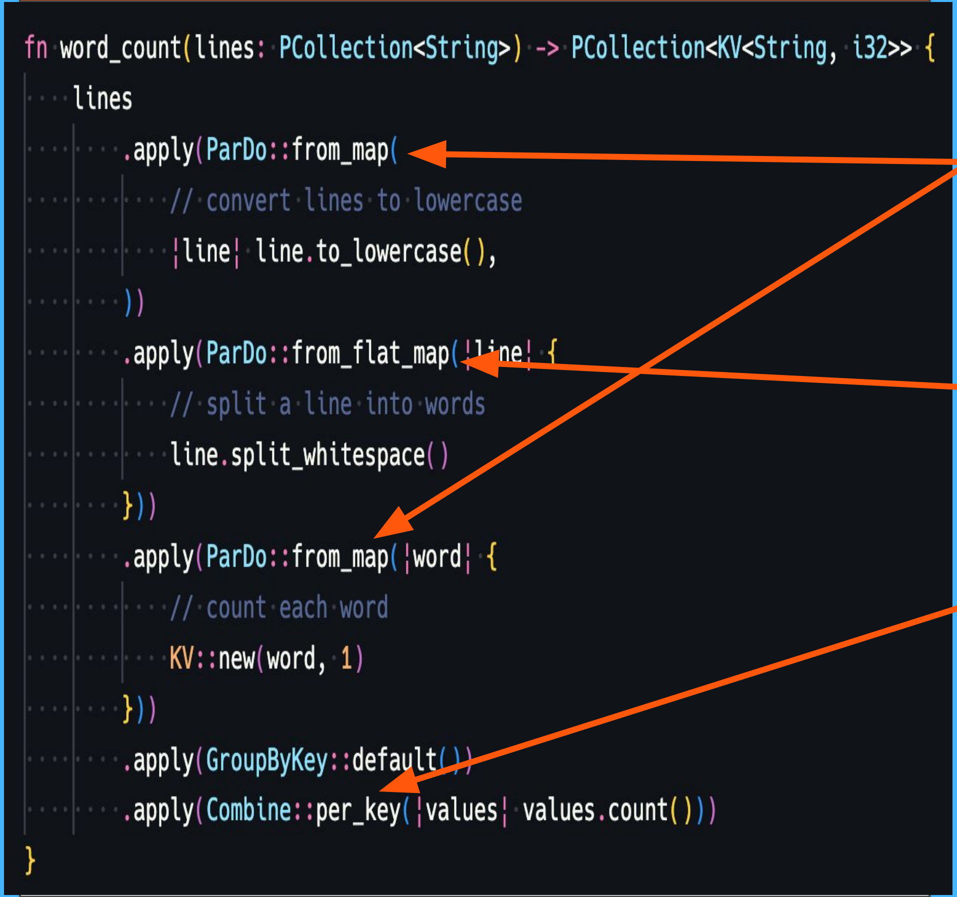
word: **String**

(output PCollection): **KV<String, i32>**

(output PCollection): **KV<String, Vec<i32>>**

(output PCollection): **KV<String, i32>**

Word-count pipeline...



```
fn word_count(lines: PCollection<String>) -> PCollection<KV<String, i32>> {
    lines
        .apply(ParDo::from_map(
            // convert lines to lowercase
            |line| line.to_lowercase(),
        ))
        .apply(ParDo::from_flat_map(|line| {
            // split a line into words
            line.split_whitespace()
        }))
        .apply(ParDo::from_map(|word| {
            // count each word
            KV::new(word, 1)
        }))
        .apply(GroupByKey::default())
        .apply(Combine::per_key(|values| values.count()))
}
```

and its usage from DirectRunner

Statically-typed (w/ generics)

```
fn from_map<F, In, Out>(func: F) -> ParDo
  where
    F: Fn(&In) -> Out,
    In: ElemType, Out: ElemType

fn from_flat_map<F, In, Out>(func: F) -> ParDo
  where
    F: Fn(&In) -> Vec<Out>,
    In: ElemType, Out: ElemType

fn per_key<F, In, Out>(func: F) -> Combine
  where
    F: Fn(&In) -> Vec<Out>,
    In: ElemType, Out: ElemType
```

Word-count pipeline…

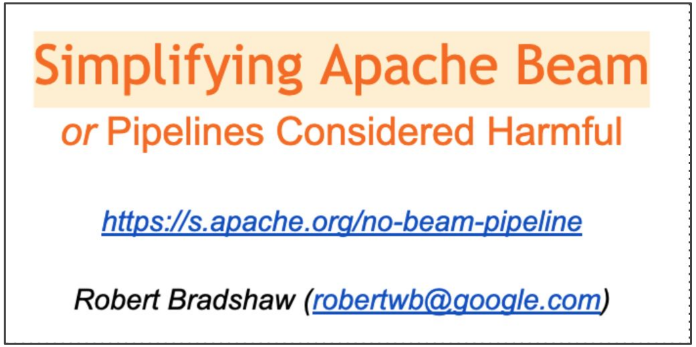
and its usage from DirectRunner

Runner.run() instead of Pipeline.run()

- Same API as TypeScript SDK.

  Runner.run() introduce pipeline root (PValue)

- Proposed in a design doc.

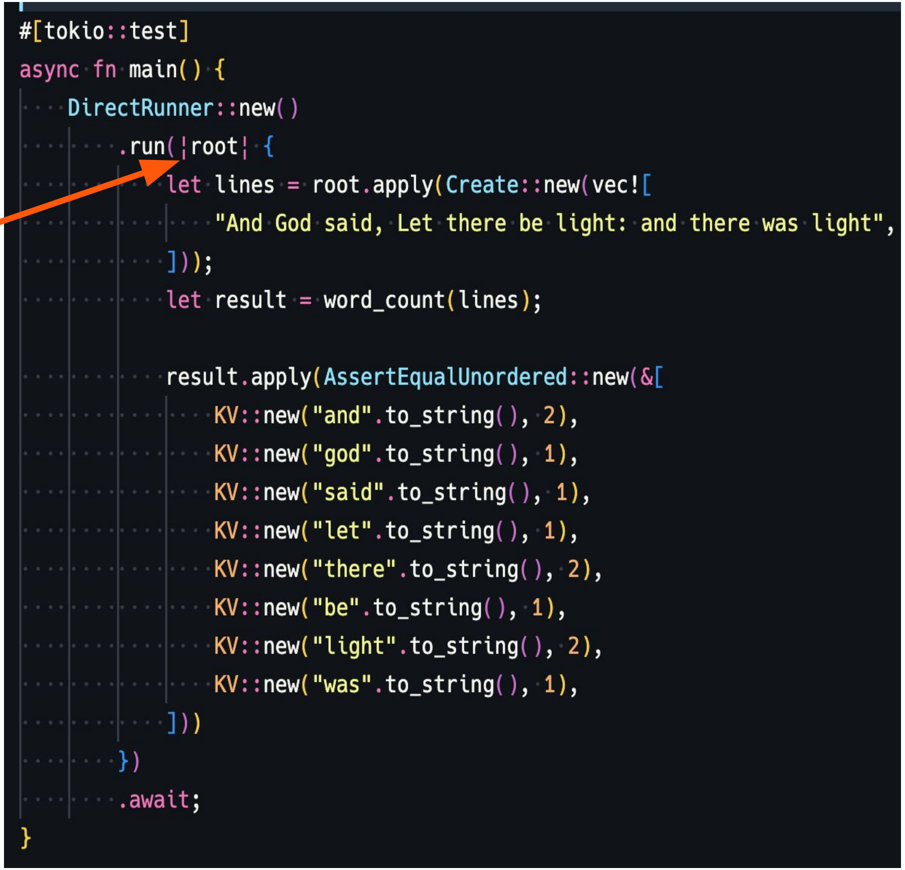


Simplifying Apache Beam

or Pipelines Considered Harmful

https://s.apache.org/no-beam-pipeline

Robert Bradshaw (robertwb@google.com)



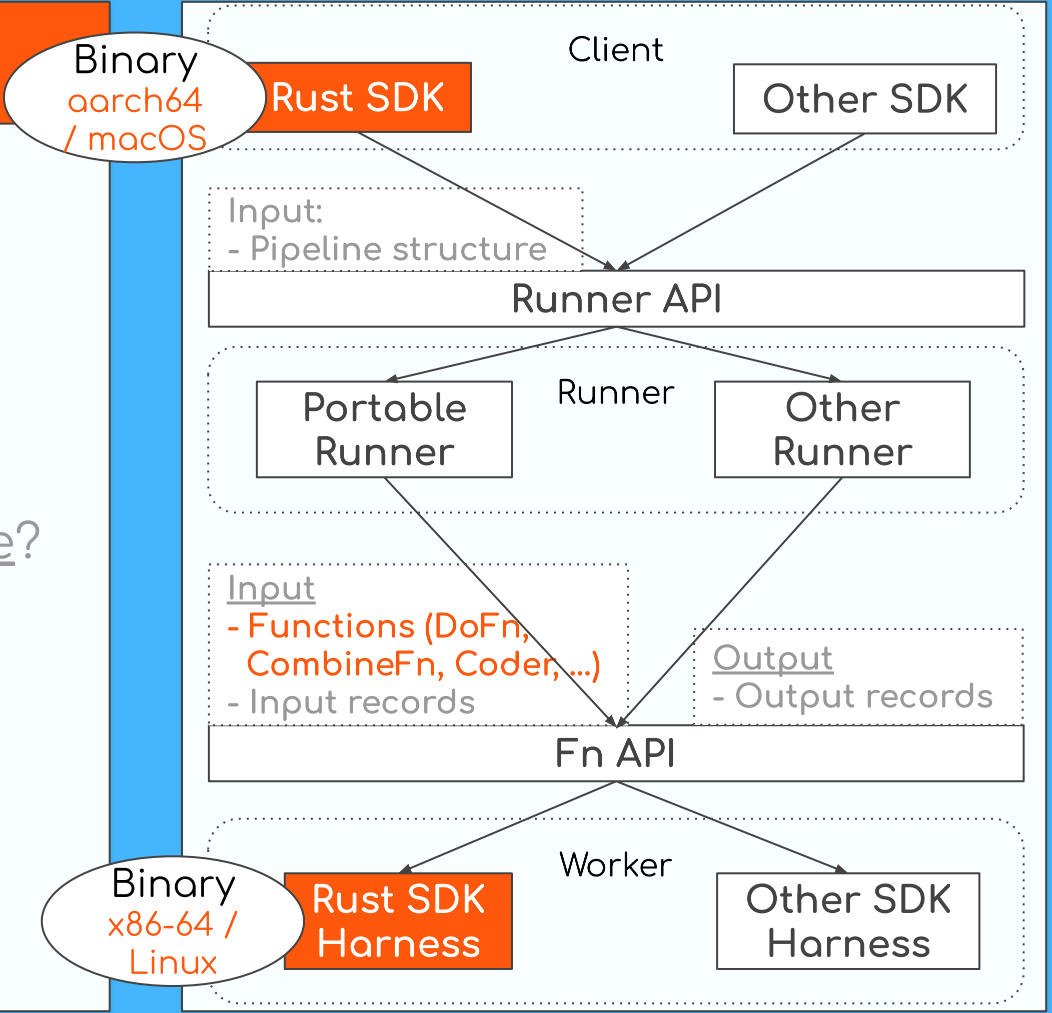
```
#[tokio::test]
async fn main() {
    DirectRunner::new()
        .run(|root| {
            let lines = root.apply(Create::new(vec![
                "And God said, Let there be light: and there was light",
            ]));
            let result = word_count(lines);

            result.apply(AssertEqualUnordered::new(&[
                KV::new("and".to_string(), 2),
                KV::new("god".to_string(), 1),
                KV::new("said".to_string(), 1),
                KV::new("let".to_string(), 1),
                KV::new("there".to_string(), 2),
                KV::new("be".to_string(), 1),
                KV::new("light".to_string(), 2),
                KV::new("was".to_string(), 1),
            ]))
        })
        .await;
}
```

Rust SDK:
Rust-specific Challenges

How to share functions?

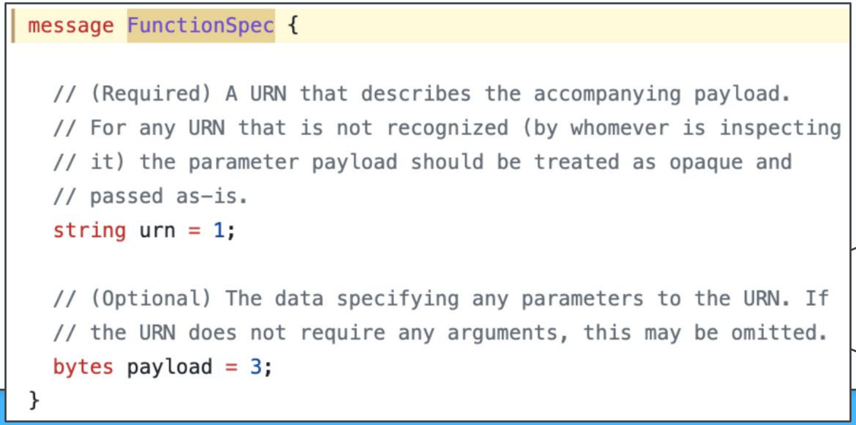
(between client and worker)

- Functions (and closures)
  - User-defined ParDo, CombineFn, Coder, ...
- Both binaries contain the same functions, but how does a worker determine which functions to execute?

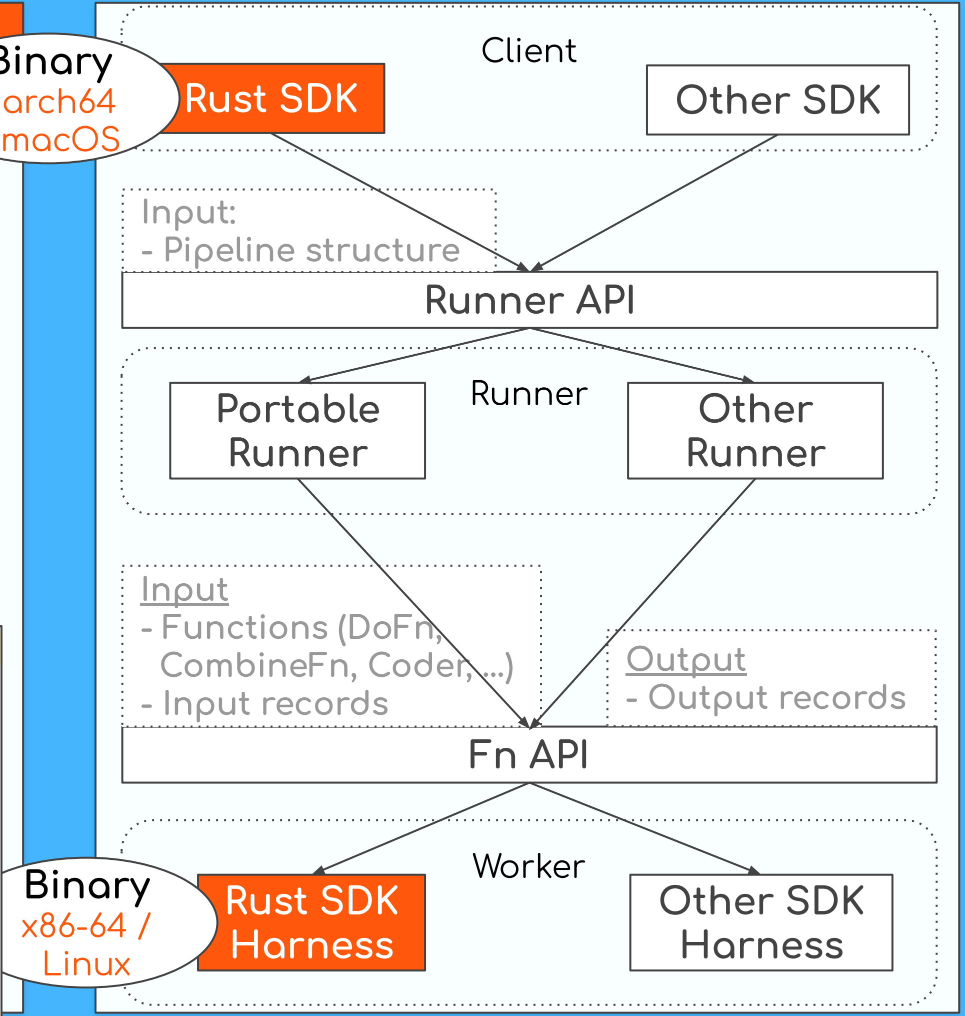


How to share functions?
(between client and worker)

- Functions (and closures)
  - User-defined ParDo, CombineFn, Coder, ...
- Both binaries contain the same functions, but how does a worker determine which functions to execute?
  - From Fn API, worker receives:

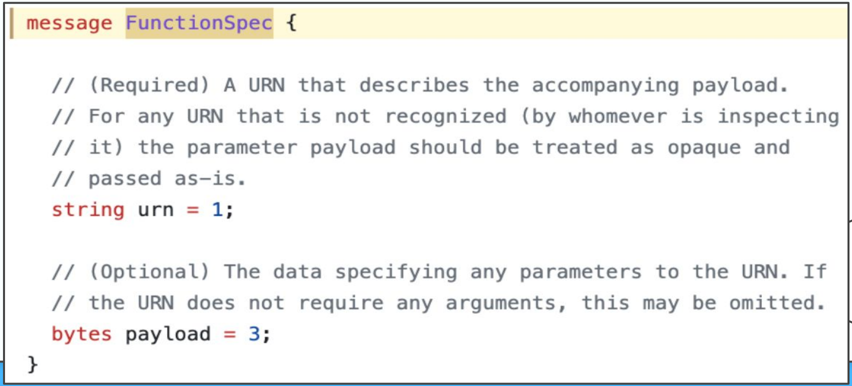


```
message FunctionSpec {

  // (Required) A URN that describes the accompanying payload.
  // For any URN that is not recognized (by whomever is inspecting
  // it) the parameter payload should be treated as opaque and
  // passed as-is.
  string urn = 1;

  // (Optional) The data specifying any parameters to the URN. If
  // the URN does not require any arguments, this may be omitted.
  bytes payload = 3;
}
```

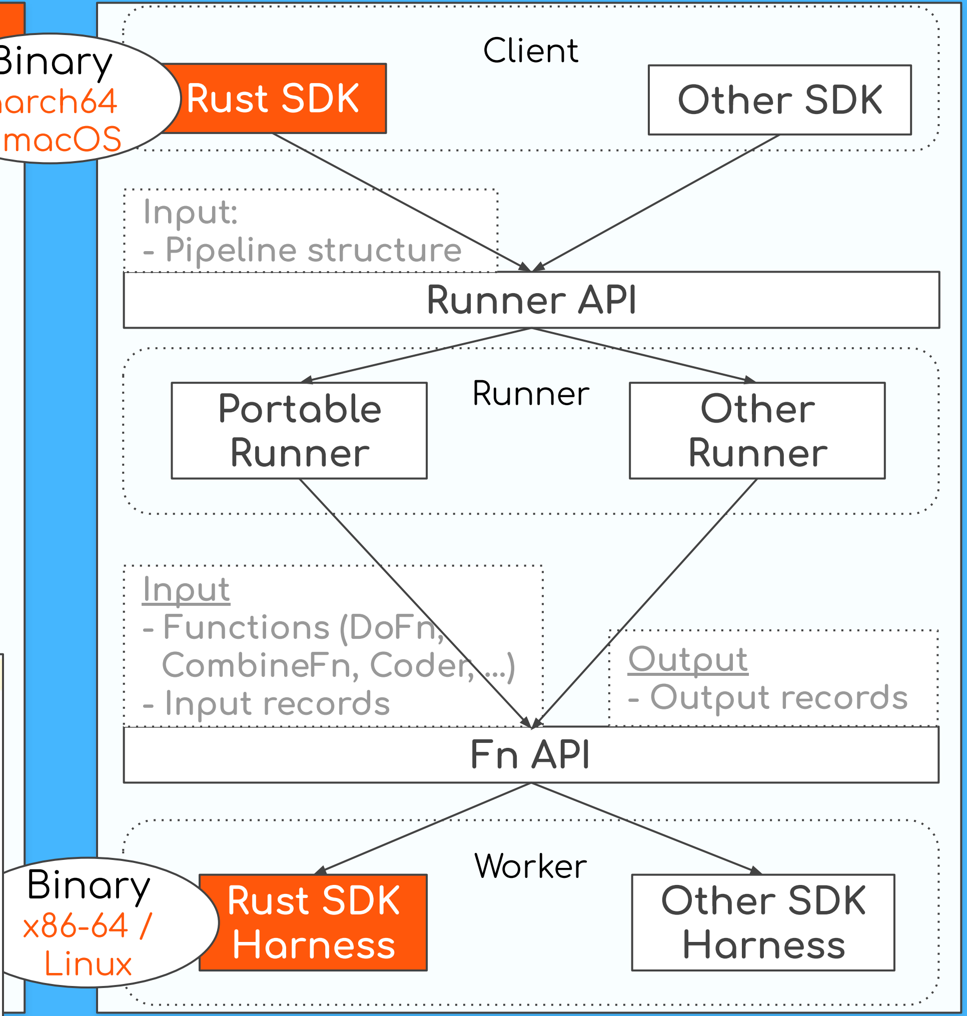


How to share functions?
(between client and worker)

- How does a worker decide which function to execute?
- Deserialize function body from payload?
  - Cannot serialize functions in Rust (especially for generic ones).
    - See discussion in a design doc for detail



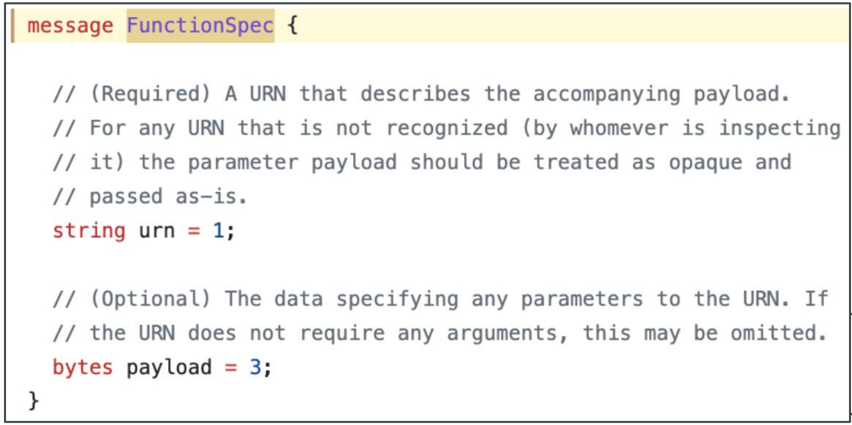
```
message FunctionSpec {

  // (Required) A URN that describes the accompanying payload.
  // For any URN that is not recognized (by whomever is inspecting
  // it) the parameter payload should be treated as opaque and
  // passed as-is.
  string urn = 1;

  // (Optional) The data specifying any parameters to the URN. If
  // the URN does not require any arguments, this may be omitted.
  bytes payload = 3;
}
```



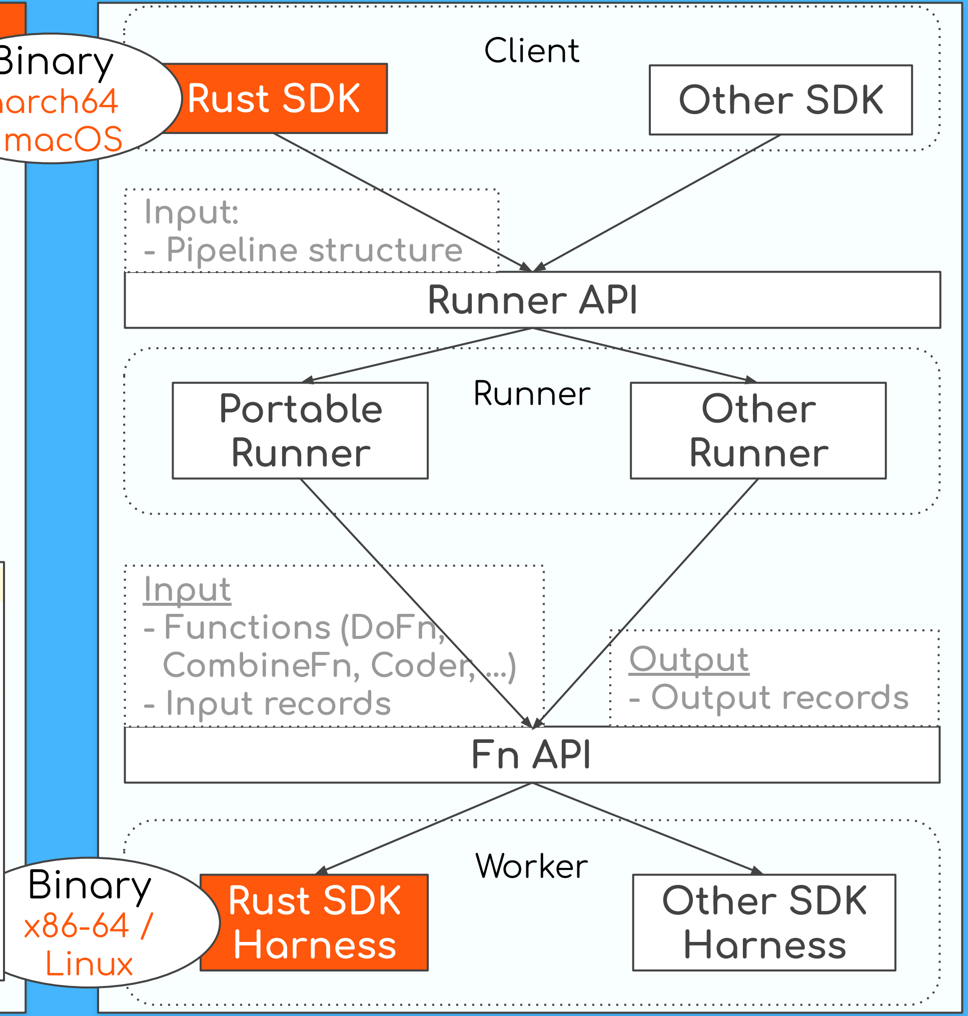
How to share functions?

(between client and worker)

- How does a worker decide which function to execute?
- Function symbols in URN?
  - **No reflection** in Rust (cannot call function from its symbol)
  - Closures are unnamed
  - Different from Go SDK



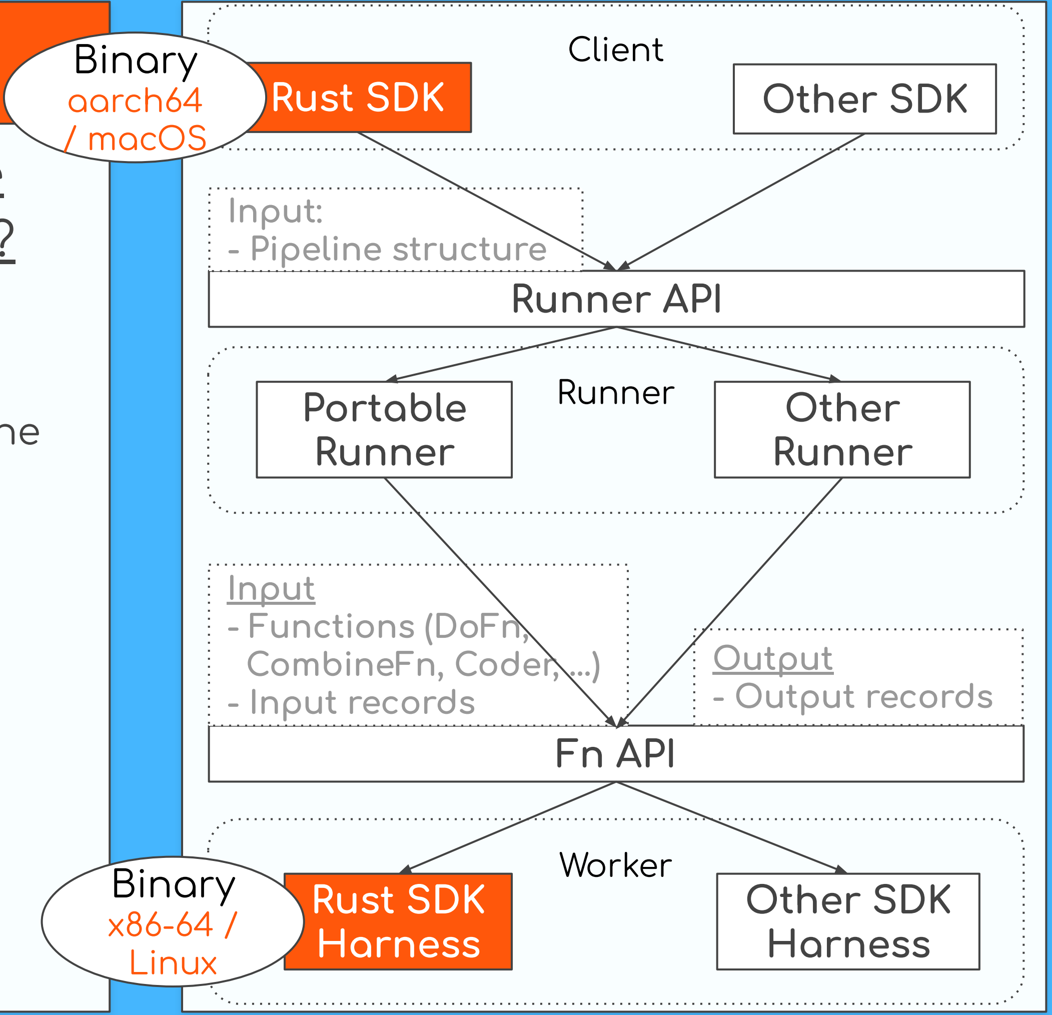
```
message FunctionSpec {

  // (Required) A URN that describes the accompanying payload.
  // For any URN that is not recognized (by whomever is inspecting
  // it) the parameter payload should be treated as opaque and
  // passed as-is.
  string urn = 1;

  // (Optional) The data specifying any parameters to the URN. If
  // the URN does not require any arguments, this may be omitted.
  bytes payload = 3;
}
```



How to share functions?

(between client and worker)

- <u>How does a worker decide which function to execute?</u>
- Registering such map?
  “URN → function pointer“
  - Init function might register the map
    - *Note that function pointers differ in Client and Worker (different binary)*
  - Requires macro and further implementation efforts, but seems not a bad idea

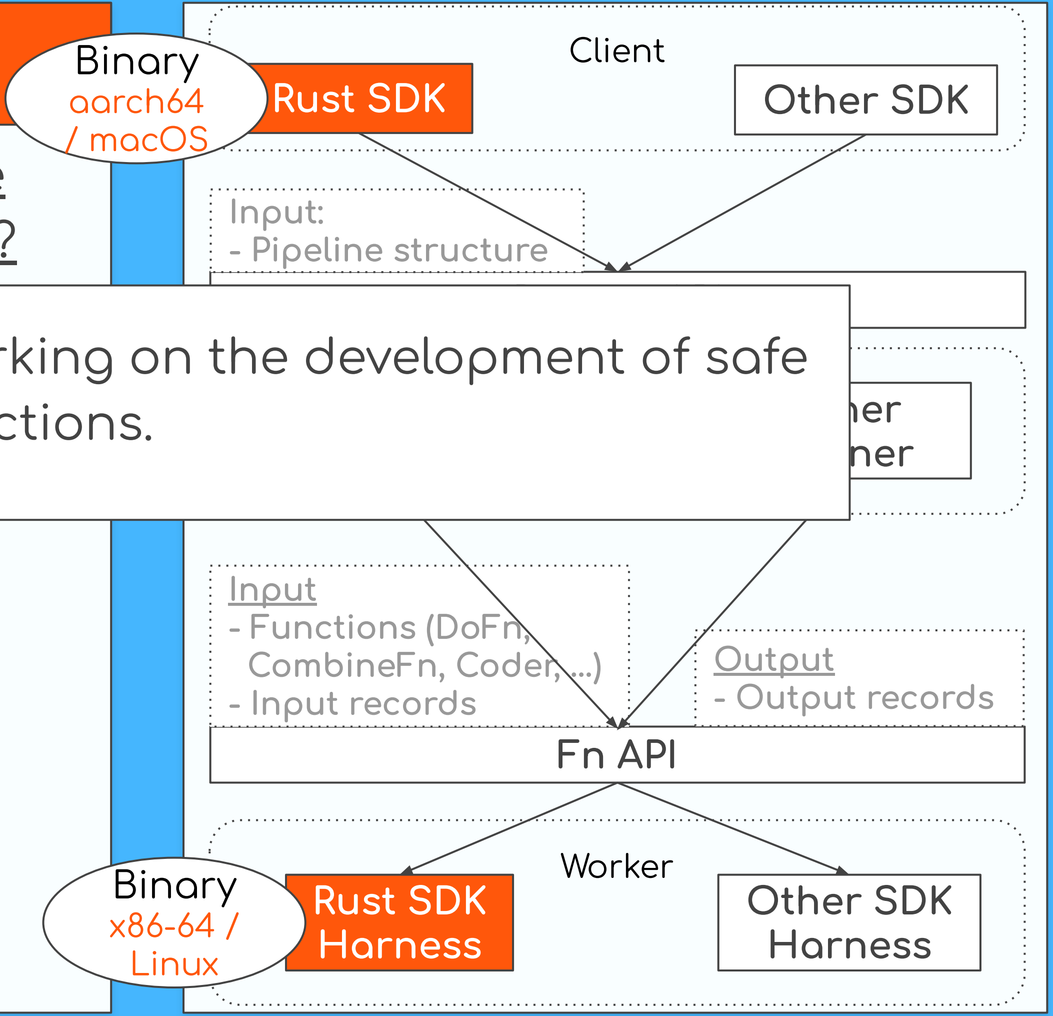


How to share functions?

(between client and worker)

- How does a worker decide which function to execute?
- Regis
  “URN
  - In
    m
    - *Note that function pointers differ in Client and Worker (different binary)*
  - Requires macro and further implementation efforts, but seems not a bad idea

We are currently working on the development of safe serialization for functions.



Rust SDK: <u>Development history and future</u>

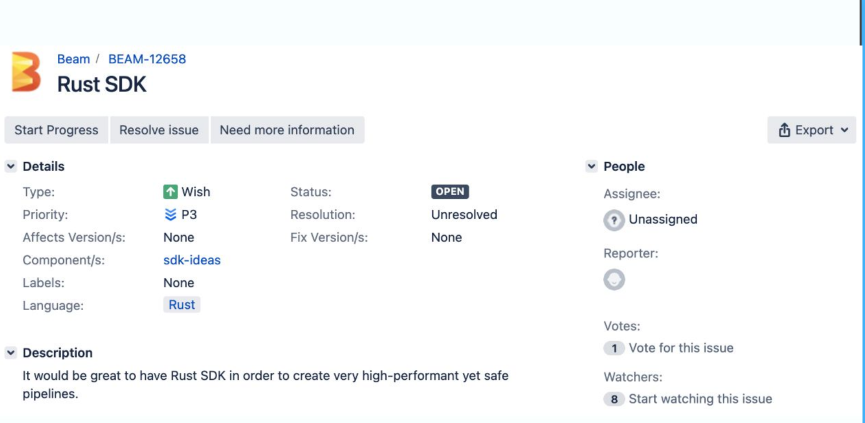
Why History?

- While I currently serve as the repository owner of the experimental Beam Rust SDK, I am not the project's original contributor.
- It is important for me to acknowledge and honor the contributions of past and current individuals involved in the project.

I apologize if I have unintentionally omitted mentioning any specific contributor names.

Started from a JIRA Ticket

- The Rust SDK issue was created in July 2021 on JIRA
- There was a recommendation to learn from the TypeScript SDK
- An initial concept of pipeline construction was shared in a Gist
- Contributor
  - jayendra13
- Advisers
  - kennknowles
  - robertwb
  - lostluck



Beam / BEAM-12658

Rust SDK

Details

Type: Wish

Priority: P3

Affects Version/s: None

Component/s: sdk-ideas

Labels: None

Language: Rust

Status: OPEN

Resolution: Unresolved

Fix Version/s: None

People

Assignee: Unassigned

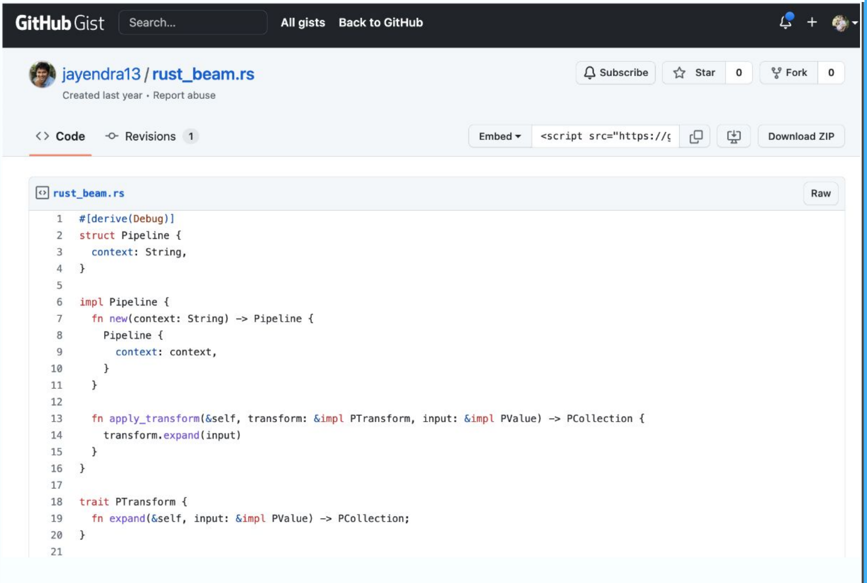
Reporter:

Votes: 1 Vote for this issue

Watchers: 8 Start watching this issue

Description

It would be great to have Rust SDK in order to create very high-performant yet safe pipelines.



jayendra13 / rust_beam.rs

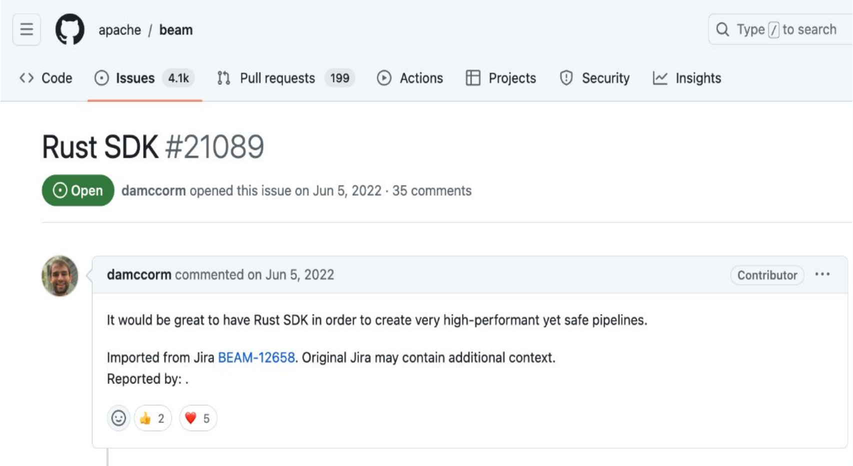
Created last year · Report abuse

rust_beam.rs

```rust
#[derive(Debug)]
struct Pipeline {
  context: String,
}

impl Pipeline {
  fn new(context: String) -> Pipeline {
    Pipeline {
      context: context,
    }
  }

  fn apply_transform(&self, transform: &impl PTransform, input: &impl PValue) -> PCollection {
    transform.expand(input)
  }
}

trait PTransform {
  fn expand(&self, input: &impl PValue) -> PCollection;
}

```

Issue Migrated to GitHub

- The issue on GitHub is still active to this day
- Experimental implementation repos:
  - kennknowles/beam [old]
  - ↓ (merged into)
  - nivaldoh/beam [old]
  - ↓ (forked to)
  - laysakura/beam [current]
- Organizer: brucearctor



Rust SDK #21089

Open damccorm opened this issue on Jun 5, 2022 · 35 comments

damccorm commented on Jun 5, 2022 Contributor

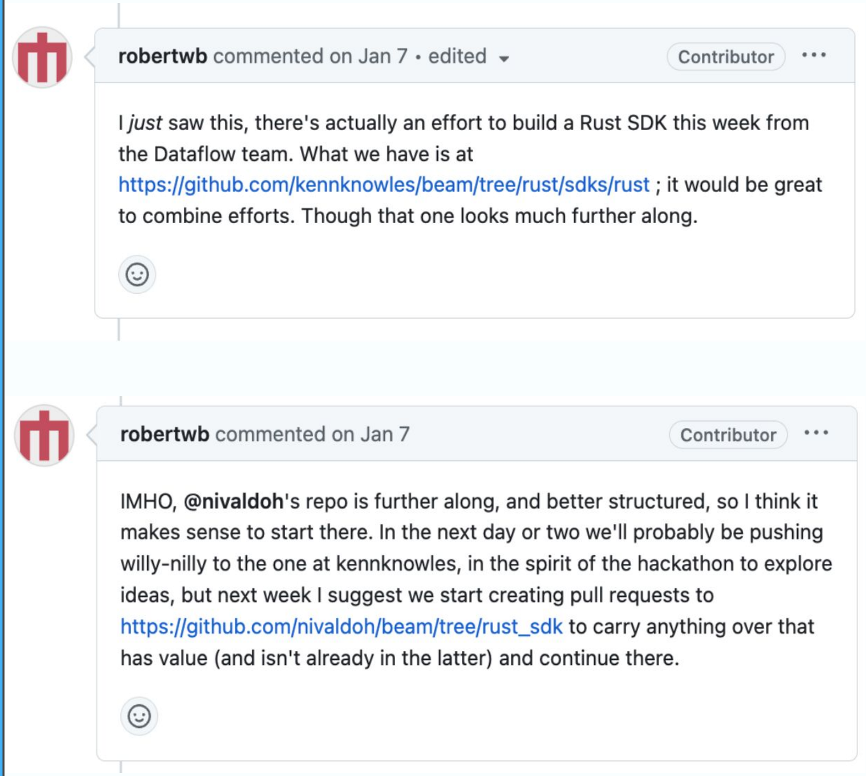
It would be great to have Rust SDK in order to create very high-performant yet safe pipelines.

Imported from Jira BEAM-12658. Original Jira may contain additional context.
Reported by: .

👍 2 ❤️ 5

[Old repo] kennknowles/beam

- Project initiation: January 2023
- The Google Cloud Dataflow team started a Rust SDK development
- Later merged into nivaldoh/beam repository
- Contributors
    - antonbobkov
    - robertwb
    - JayDosunmu
    - y1chi



robertwb commented on Jan 7 • edited

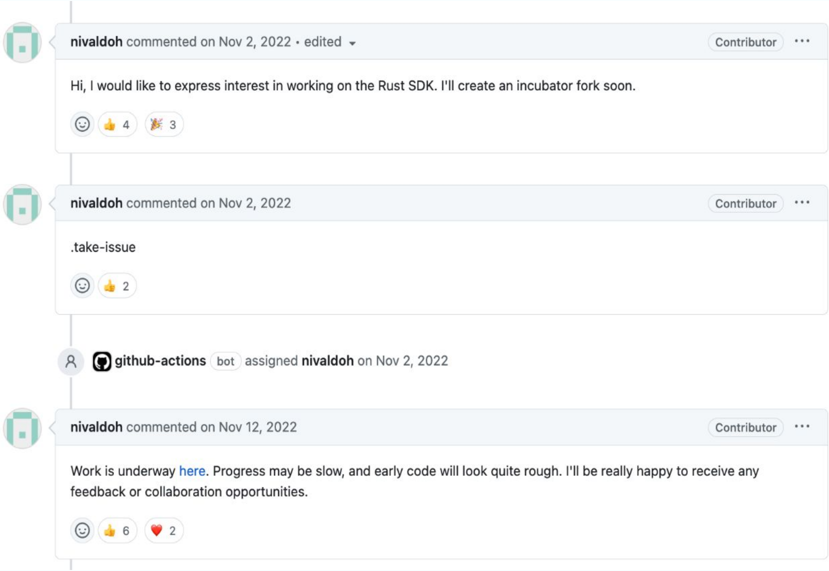
I *just* saw this, there's actually an effort to build a Rust SDK this week from the Dataflow team. What we have is at https://github.com/kennknowles/beam/tree/rust/sdks/rust ; it would be great to combine efforts. Though that one looks much further along.

robertwb commented on Jan 7

IMHO, **@nivaldoh**'s repo is further along, and better structured, so I think it makes sense to start there. In the next day or two we'll probably be pushing willy-nilly to the one at kennknowles, in the spirit of the hackathon to explore ideas, but next week I suggest we start creating pull requests to https://github.com/nivaldoh/beam/tree/rust_sdk to carry anything over that has value (and isn't already in the latter) and continue there.

[Old repo] nivaldoh/beam

- Project initiation: November 2022
- Added:
  - Codes for pipeline construction (partial)
  - Worker codes (partial)
- Development activities ceased since February 2023
- Contributors
  - nivaldoh
  - sjvanrossum
  - laysakura (me)
  - Miuler



nivaldoh commented on Nov 2, 2022 • edited

Hi, I would like to express interest in working on the Rust SDK. I'll create an incubator fork soon.

nivaldoh commented on Nov 2, 2022

.take-issue

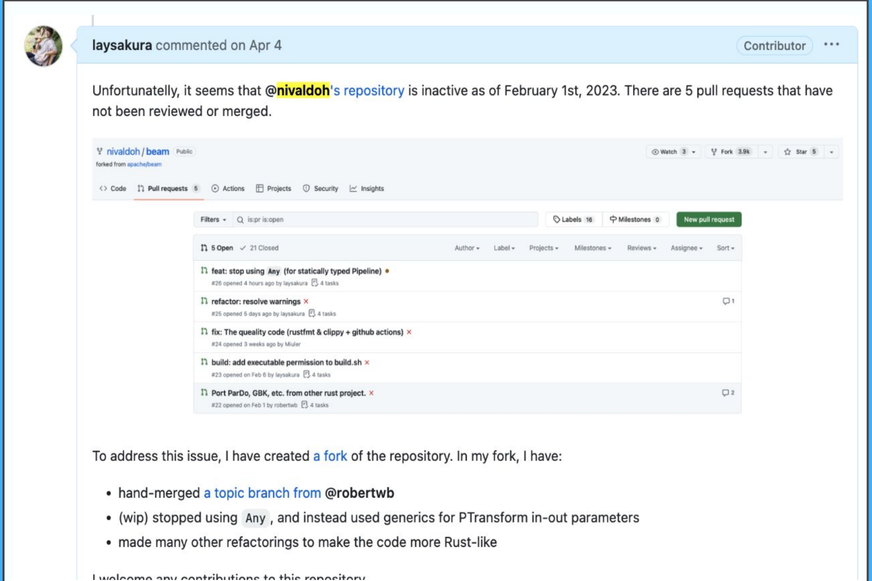
github-actions bot assigned nivaldoh on Nov 2, 2022

nivaldoh commented on Nov 12, 2022

Work is underway here. Progress may be slow, and early code will look quite rough. I'll be really happy to receive any feedback or collaboration opportunities.

[Current repo] laysakura/beam

- Project initiation: February 2023
- Forked from nivaldoh/beam
- Added:
    - Coder serialization (partial)
    - More worker codes (partial)
    - General function serialization (doing)
    - The Beam Programming Guide for Rust (doing)
- Contributors
    - dahlbaek
    - sjvanrossum
    - Kelvinyu1117
    - laysakura (me)



laysakura commented on Apr 4

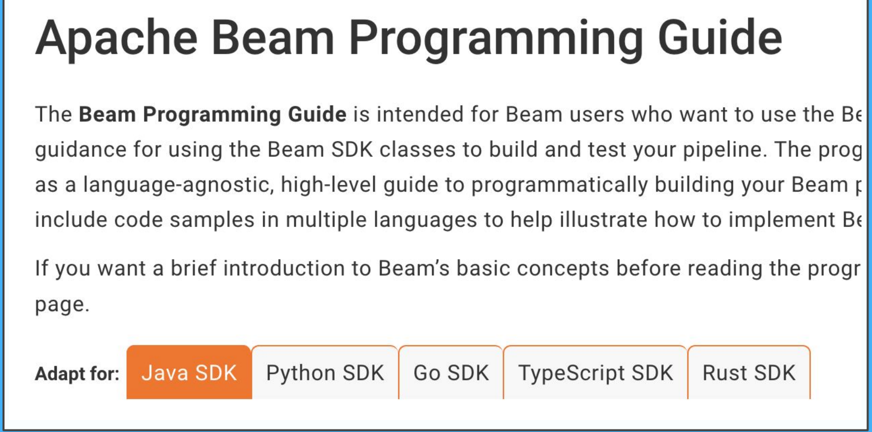
Unfortunatelly, it seems that @**nivaldoh**'s repository is inactive as of February 1st, 2023. There are 5 pull requests that have not been reviewed or merged.

- feat: stop using `Any` (for statically typed Pipeline)
  #26 opened 4 hours ago by laysakura 4 tasks
- refactor: resolve warnings
  #25 opened 5 days ago by laysakura 4 tasks 1
- fix: The quealty code (rustfmt & clippy + github actions)
  #24 opened 3 weeks ago by Miuler
- build: add executable permission to build.sh
  #23 opened on Feb 6 by laysakura 4 tasks
- Port ParDo, GBK, etc. from other rust project.
  #22 opened on Feb 1 by robertwb 4 tasks 2

To address this issue, I have created a fork of the repository. In my fork, I have:

- hand-merged a topic branch from @**robertwb**
- (wip) stopped using `Any`, and instead used generics for PTransform in-out parameters
- made many other refactorings to make the code more Rust-like

I welcome any contributions to this repository



Apache Beam Programming Guide

The **Beam Programming Guide** is intended for Beam users who want to use the Be
guidance for using the Beam SDK classes to build and test your pipeline. The prog
as a language-agnostic, high-level guide to programmatically building your Beam p
include code samples in multiple languages to help illustrate how to implement Be

If you want a brief introduction to Beam's basic concepts before reading the progr
page.

Adapt for: Java SDK | Python SDK | Go SDK | TypeScript SDK | Rust SDK

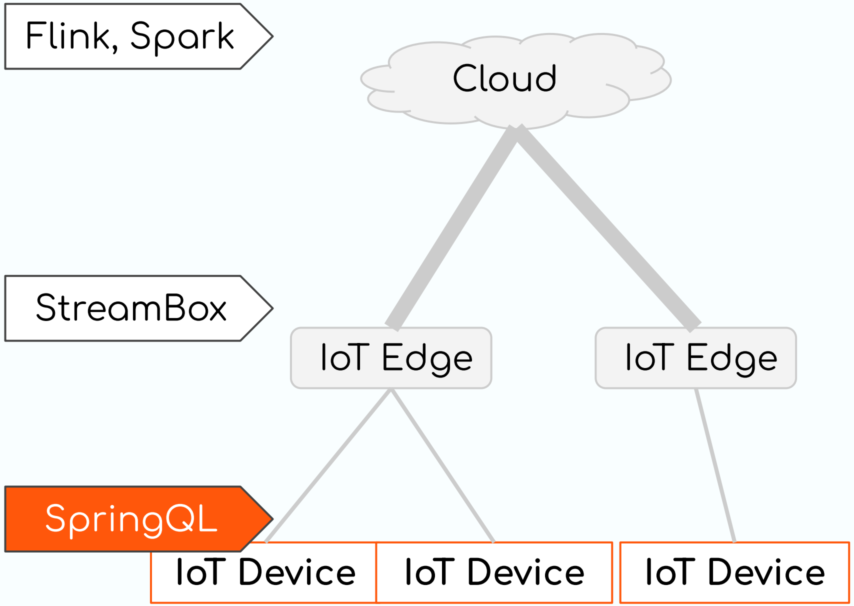
Future work

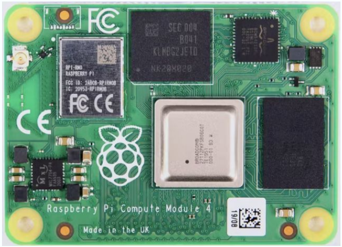
- Technically challenging implementations
  - Serialization/deserialization of functions (including closures), led by sjvanrossum
- Align design considerations for non-trivial features
  - Registration of user-defined objects (possibly through init function w/ macros)
  - Coders (custom coders, row coders, etc.)
  - Artifact staging service
- Completion of the Programming Guide and working examples
- **Call for more contributors!**
  - Will create good-first issues in laysakura/beam

SpringQL: Introduction and integration with Beam

SpringQL's Target

- Stream Processing Engine for IoT devices
    - Targeting middle-to-high end devices
        - Raspberry Pi
        - Connected vehicles
- Support semi-realtime stream processing
    - Input:
        - Sensor data
        - UI
    - Output:
        - Device actuation
        - Aggregated data (sent to edge/cloud)
        - UI (display, sound, ...)





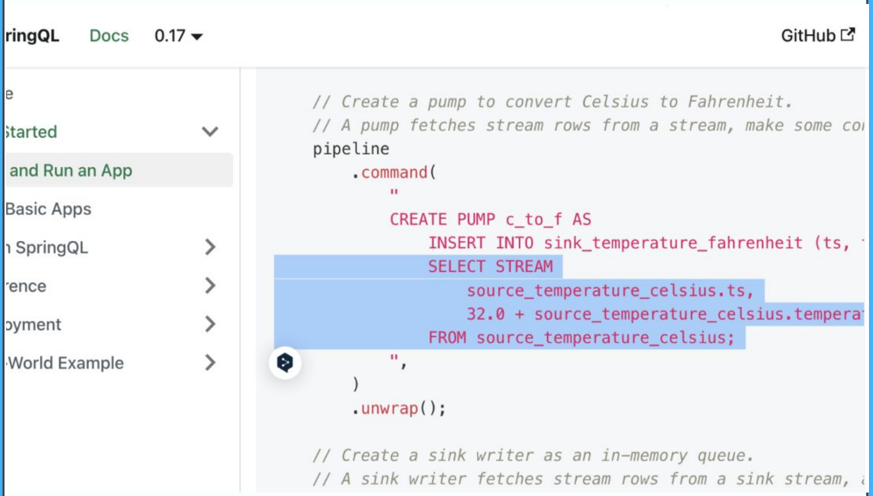
https://www.raspberrypi.com/products/compute-module-4/?variant=raspberry-pi-cm4001000

https://www.irasutoya.com/2019/10/blog-post_57.html

SpringQL's Current Status

- Implemented in Rust (repo)
- Distributed as libraries:
  - Rust (static)
  - C (static / dynamic)
- User interface
  - Client: Rust / C
  - Pipeline construction: SQL-like
  - Operation: Streaming SQL
- Problems
  - Difficulty in constructing DAGs using SQL-like language
  - Limited operations available through streaming SQL

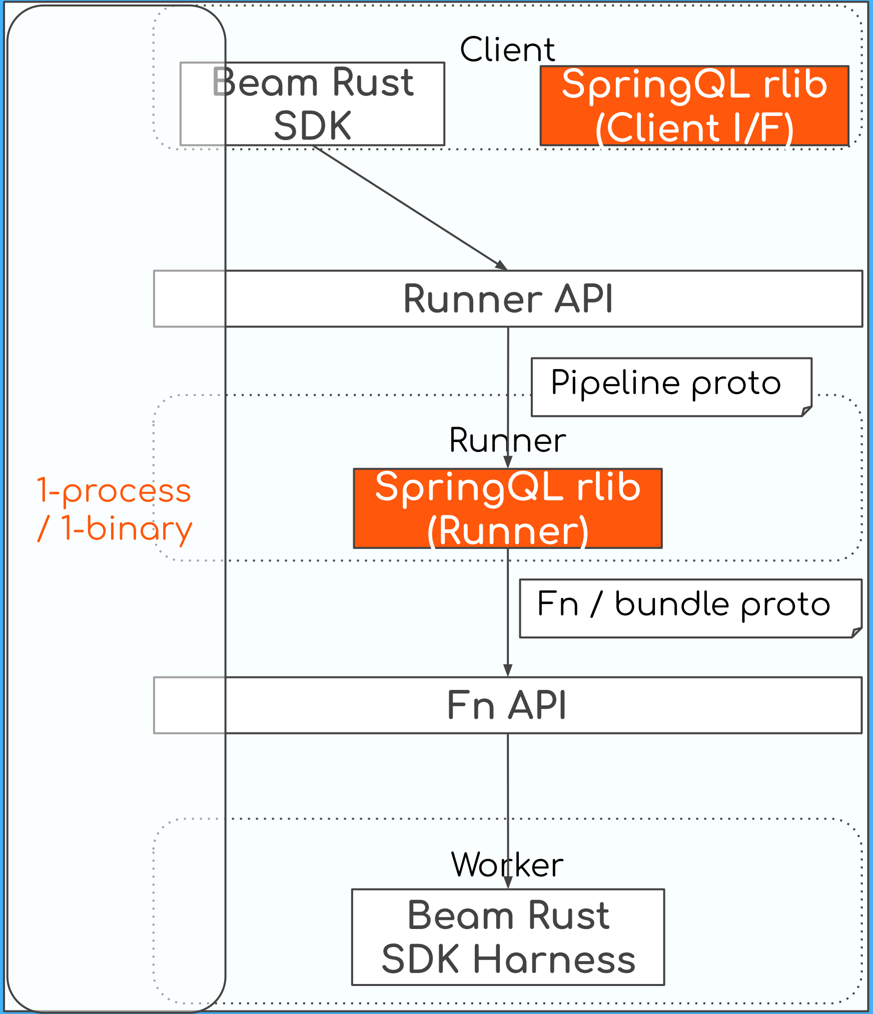
Desire to utilize Beam for U/I



```
// Create a pump to convert Celsius to Fahrenheit.
// A pump fetches stream rows from a stream, make some co
pipeline
    .command(
        "
        CREATE PUMP c_to_f AS
            INSERT INTO sink_temperature_fahrenheit (ts,
            SELECT STREAM
                source_temperature_celsius.ts,
                32.0 + source_temperature_celsius.tempera
            FROM source_temperature_celsius;
        ",
    )
    .unwrap();

// Create a sink writer as an in-memory queue.
// A sink writer fetches stream rows from a sink stream,
```

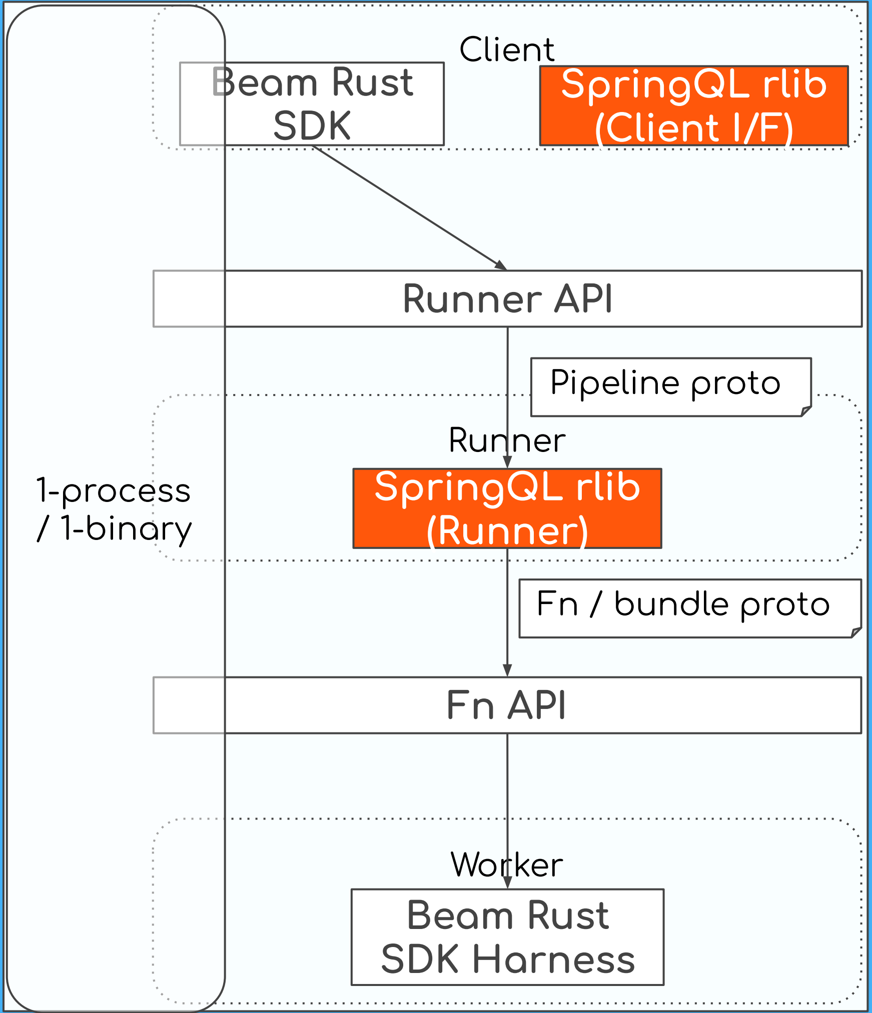
Initial Idea: Integration with Beam

- App, Beam SDK, and SpringQL library are all within the same process and binary
- SpringQL library serves as:
  - Client interface
  - Dedicated runner
- SpringQL Runner receives pipeline graph via Runner API in protobuf format
- SpringQL runner calls SDK Harness to execute UDFs
  - May use "LOOPBACK" SDK Harness (config doc)



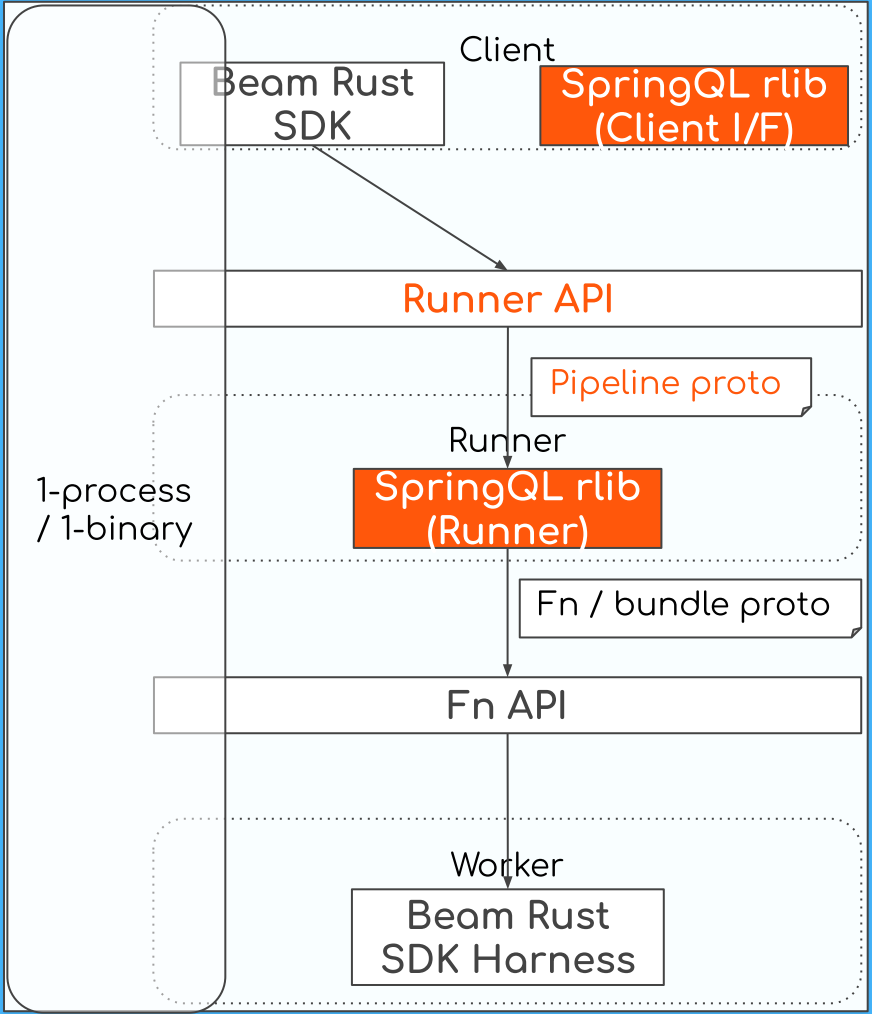
Initial Idea: Integration with Beam

- App, Beam SDK, and SpringQL library are all within the same process and binary
- SpringQL library serves as:
  - Client interface
  - Dedicated runner
- SpringQL Runner receives pipeline graph via Runner API in protobuf format
- SpringQL runner calls SDK Harness to execute UDFs
  - May use "LOOPBACK" SDK Harness (config doc)



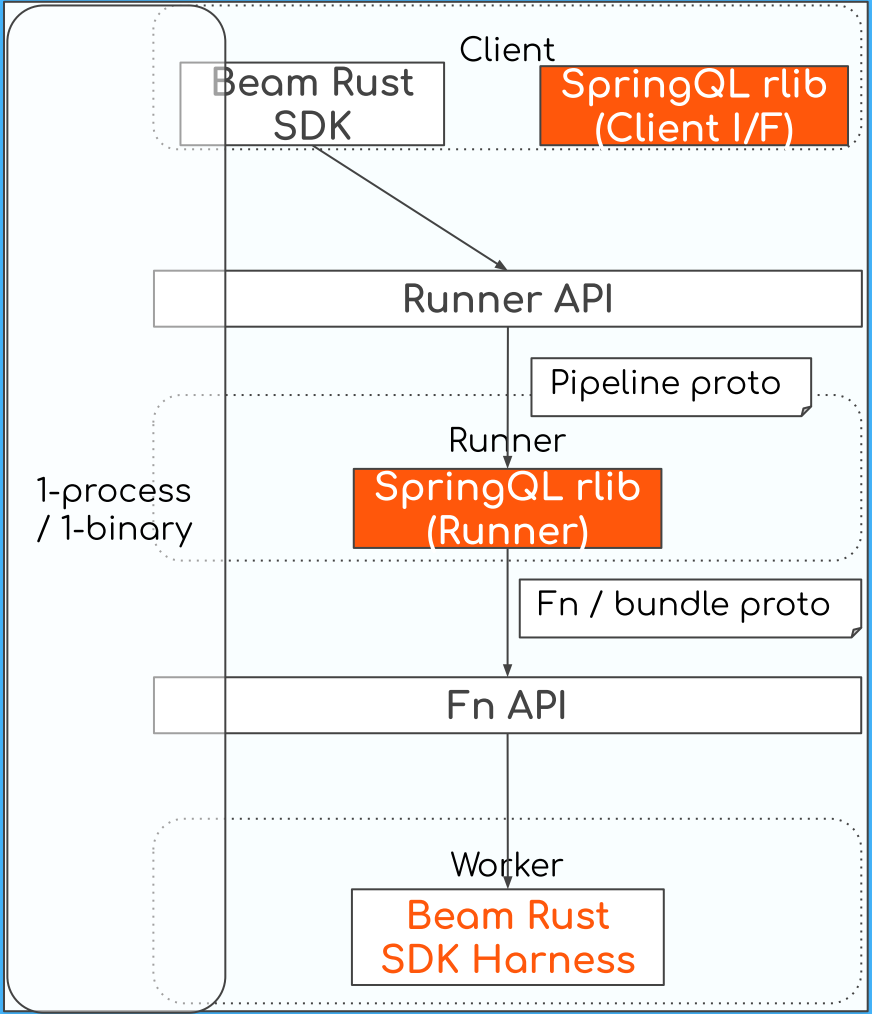
Initial Idea: Integration with Beam

- App, Beam SDK, and SpringQL library are all within the same process and binary
- SpringQL library serves as:
  - Client interface
  - Dedicated runner
- SpringQL Runner receives pipeline graph via Runner API in protobuf format
- SpringQL runner calls SDK Harness to execute UDFs
  - May use "LOOPBACK" SDK Harness (config doc)



Initial Idea: Integration with Beam

- App, Beam SDK, and SpringQL library are all within the same process and binary
- SpringQL library serves as:
  - Client interface
  - Dedicated runner
- SpringQL Runner receives pipeline graph via Runner API in protobuf format
- SpringQL runner calls SDK Harness to execute UDFs
  - May use “LOOPBACK” SDK Harness (config doc)



Summary

- About Beam Rust SDK
  - Motivation behind its development
  - Current status of the project
  - Call for contributions
- About SpringQL
  - SpringQL's target systems and architecture
  - Integration idea with Beam

Sho Nakatani

QUESTIONS?

BEAM SUMMIT
NYC 2023

Twitter: @laysakura
GitHub: laysakura