Ryan.Kelly@harrison.ai

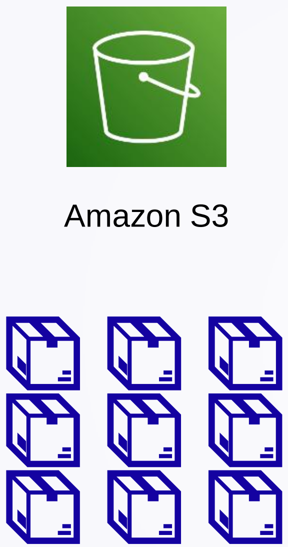
@rfkelly
https://rfk.id.au/

harrison.ai

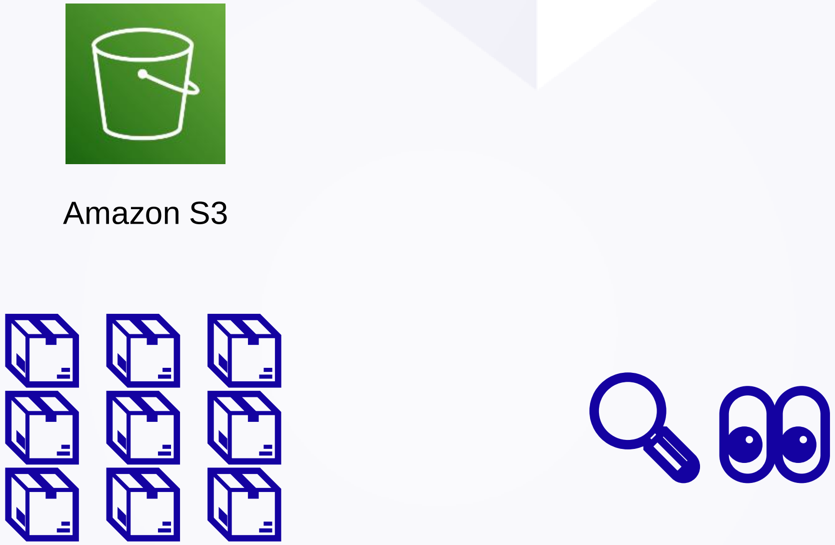
harrison.ai

# Indexing Petabytes of Data with Rust and AWS

10th October 2022



A few PB of de-identified medical image data in millions tar archives



Amazon S3

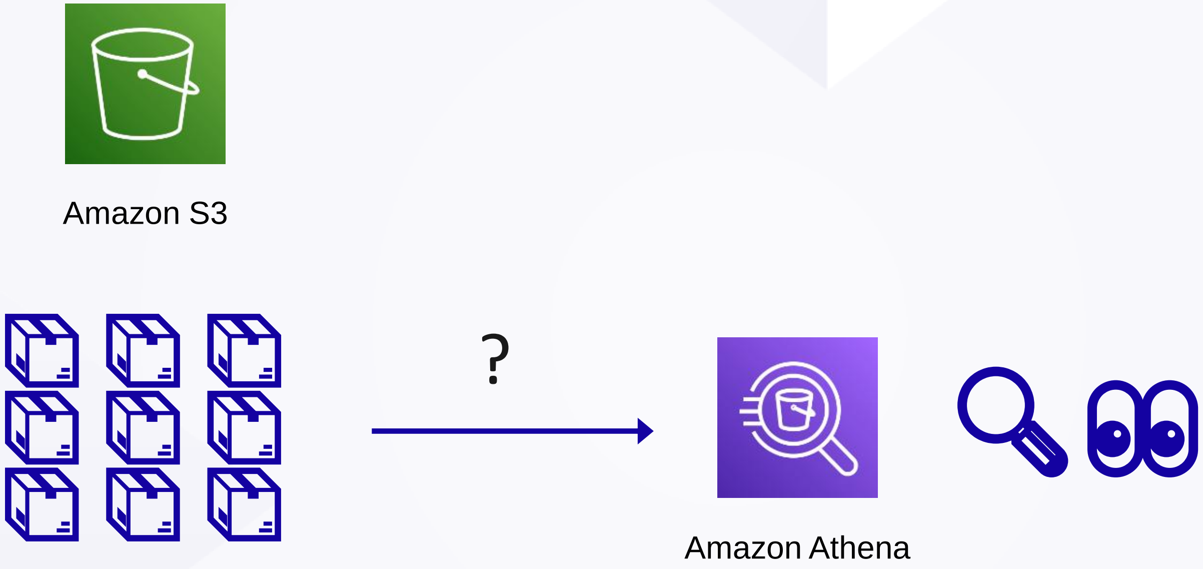
A few PB of de-identified medical image data in millions tar archives

Amazon S3

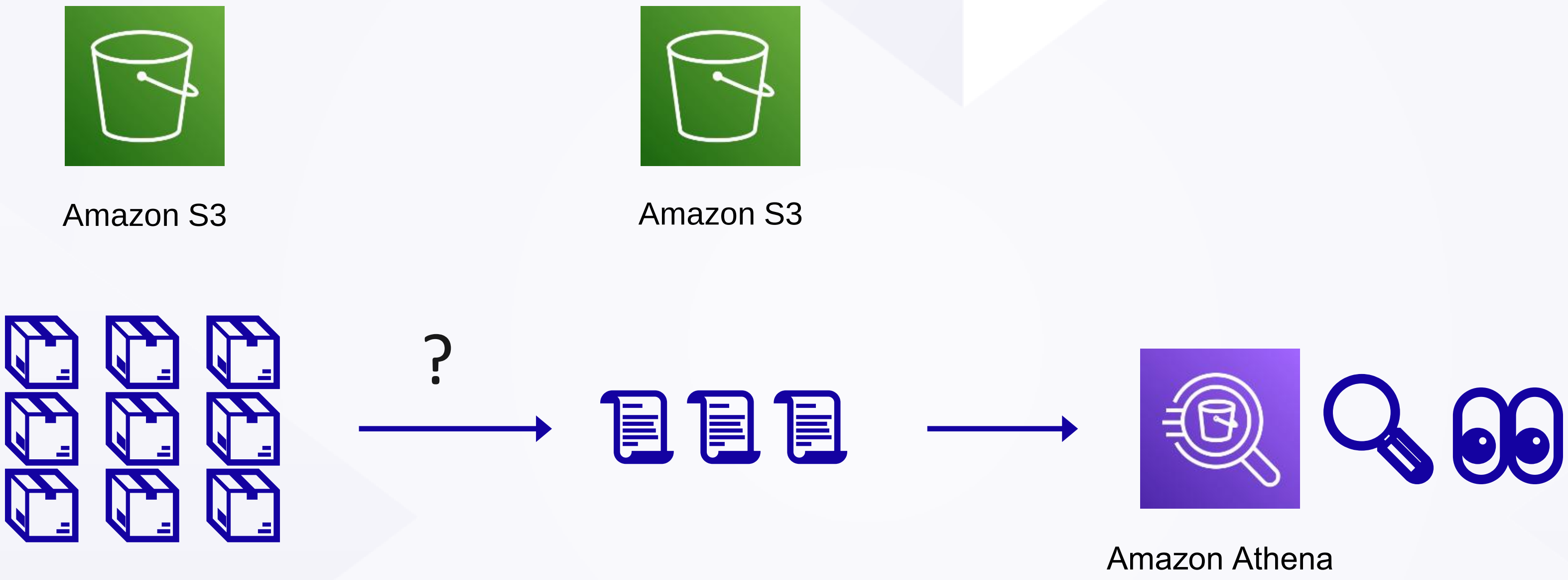
📦 📦 📦
📦 📦 📦
📦 📦 📦

🔍 👀
🤷

A few PB of de-identified medical image data in millions tar archives



A few PB of de-identified medical image data in millions tar archives



Amazon S3

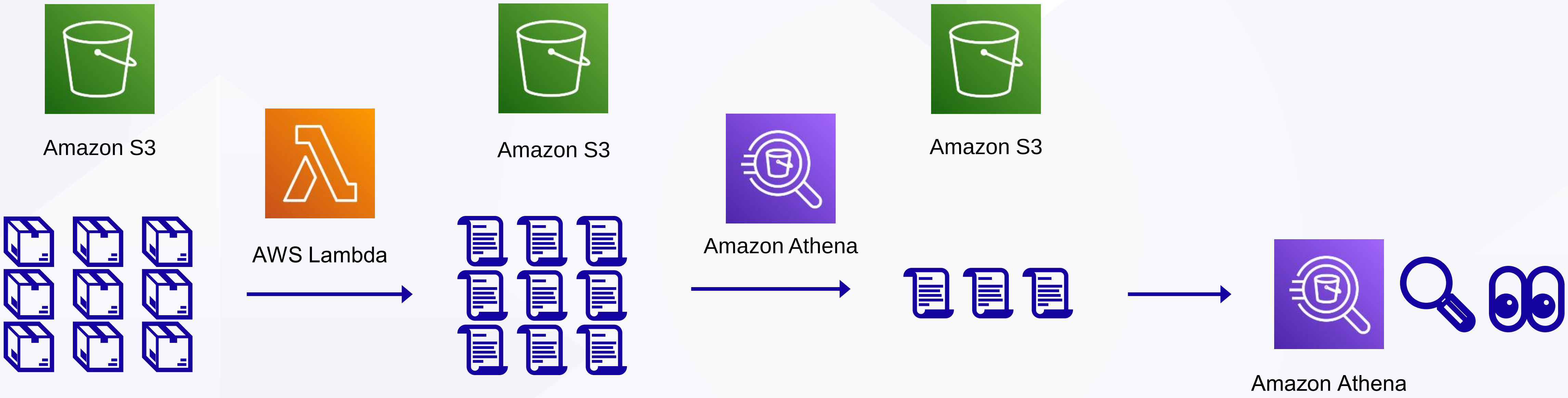
Amazon S3

?

Amazon Athena

A few PB of de-identified medical image data in millions tar archives

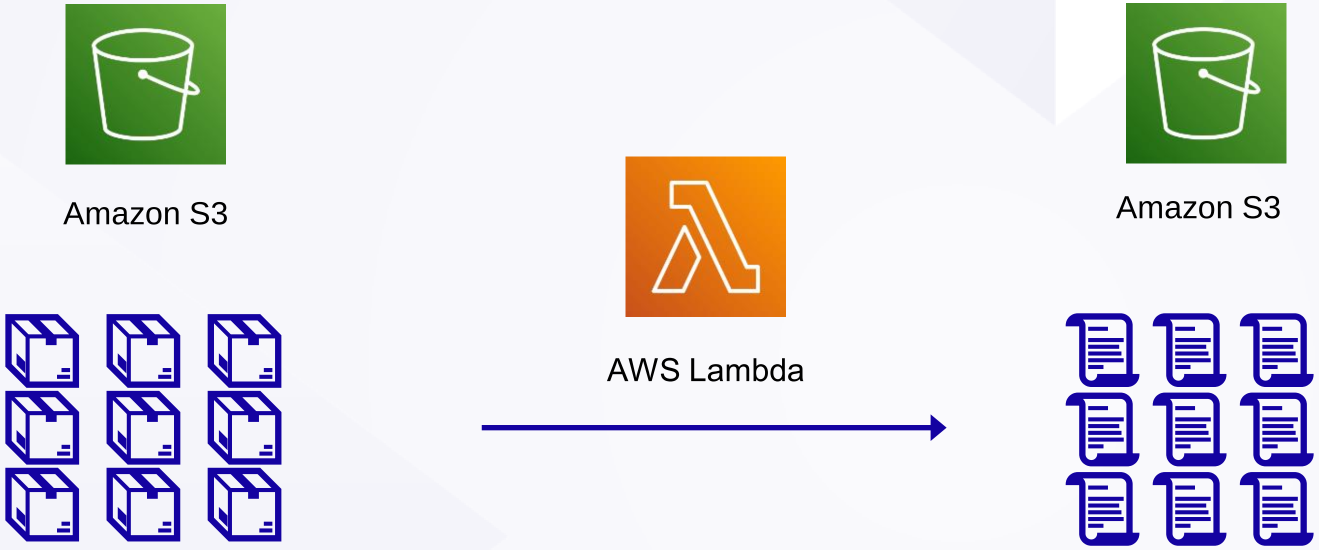
.parquet files summarizing available data



A few PB of de-identified medical image data in millions tar archives

.jsonl files listing contents of each archive

.parquet files summarizing available data



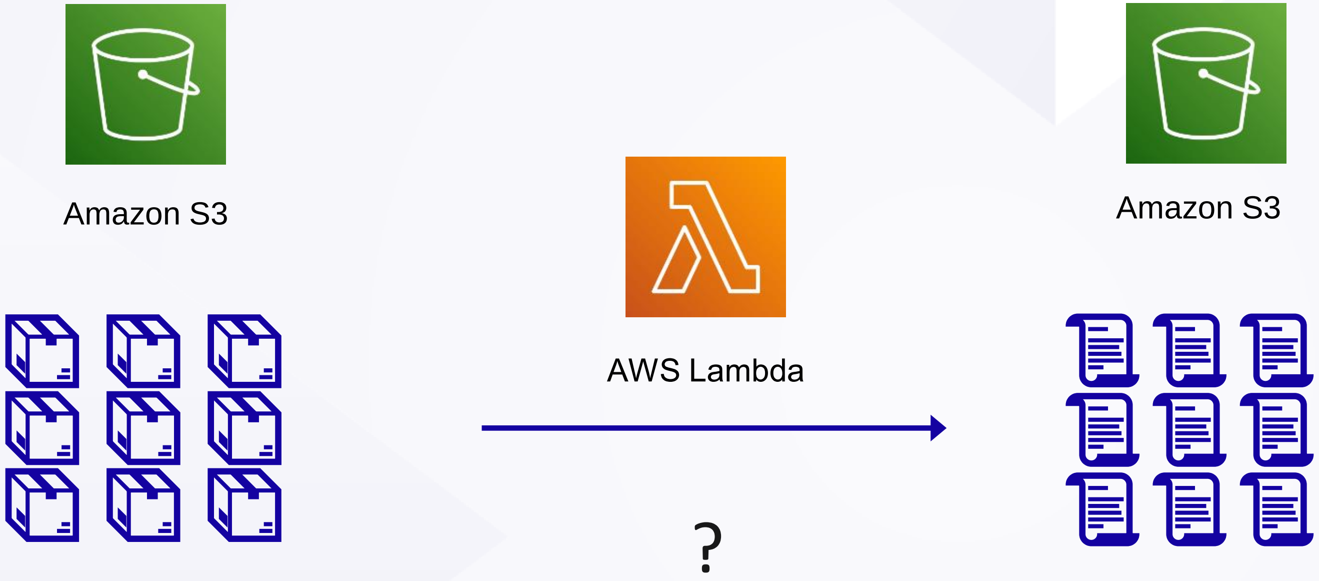
Amazon S3

AWS Lambda

Amazon S3

A few PB of de-identified medical image data in millions tar archives

.jsonl files listing contents of each archive



Amazon S3

AWS Lambda

?

Amazon S3

A few PB of de-identified medical image data in millions tar archives

.jsonl files listing contents of each archive

# A Python Baseline

```python
import json
import tarfile

def index_tarball(input_path, output_path):
    with tarfile.open(input_path) as tarball:
        with open(output_path, "w") as output:
            for member in tarball:
                row = json.dumps({
                    "archive": input_path,
                    "filename": member.name,
                    "size": member.size
                })
                output.write(row)
                output.write("\n")
```

# A Python Baseline

```python
import json
import tarfile


def index_tarball(input_path, output_path):
    with tarfile.open(input_path) as tarball:
        with open(output_path, "w") as output:
            for member in tarball:
                row = json.dumps({
                    "archive": input_path,
                    "filename": member.name,
                    "size": member.size
                })
                output.write(row)
                output.write("\n")
```

IRL, this would be rich metadata about the file, e.g. opaque patient and study identifiers

# A Basic Rust Equivalent

```
[dependencies]
anyhow = "1"
tar = "0.4"
serde = { version = "1", features=["derive"] }
serde_json = "1"
```

# A Basic Rust Equivalent

```rust
use anyhow::{Context, Result};
use serde::Serialize;
use std::fs::File;
use std::io::prelude::*;

#[derive(Serialize)]
struct IndexEntry {
    archive: String,
    filename: String,
    size: u64,
}
```

# A Basic Rust Equivalent

```rust
fn index_tarball(input_path: &str, output_path: &str) -> Result<()> {
    let mut tarball = tar::Archive::new(File::open(input_path)?);
    let mut output = File::create(output_path)?;
    for entry in tarball.entries()? {
        let entry = entry?;
        let row = serde_json::to_string(&IndexEntry {
            archive: input_path.into(),
            filename: entry.path()?.to_str().context("non-utf8 path")?.into(),
            size: entry.size(),
        })?;
        writeln!(output, "{}", row)?;
    }
    Ok(())
}
```

# A Basic Rust Equivalent

```rust
fn index_tarball(input_path: &str, output_path: &str) -> Result<()> {
    let mut tarball = tar::Archive::new(File::open(input_path)?);
    let mut output = File::create(output_path)?;
    for entry in tarball.entries()? {
        let entry = entry?;
        let row = serde_json::to_string(&IndexEntry {
            archive: input_path.into(),
            filename: entry.path()?.to_str().context("non-utf8 path")?.into(),
            size: entry.size(),
        })?;
        writeln!(output, "{}", row)?;
    }
    Ok(())
}
```

Rust won’t let a potential error pass unhandled

# Gut-check performance comparison

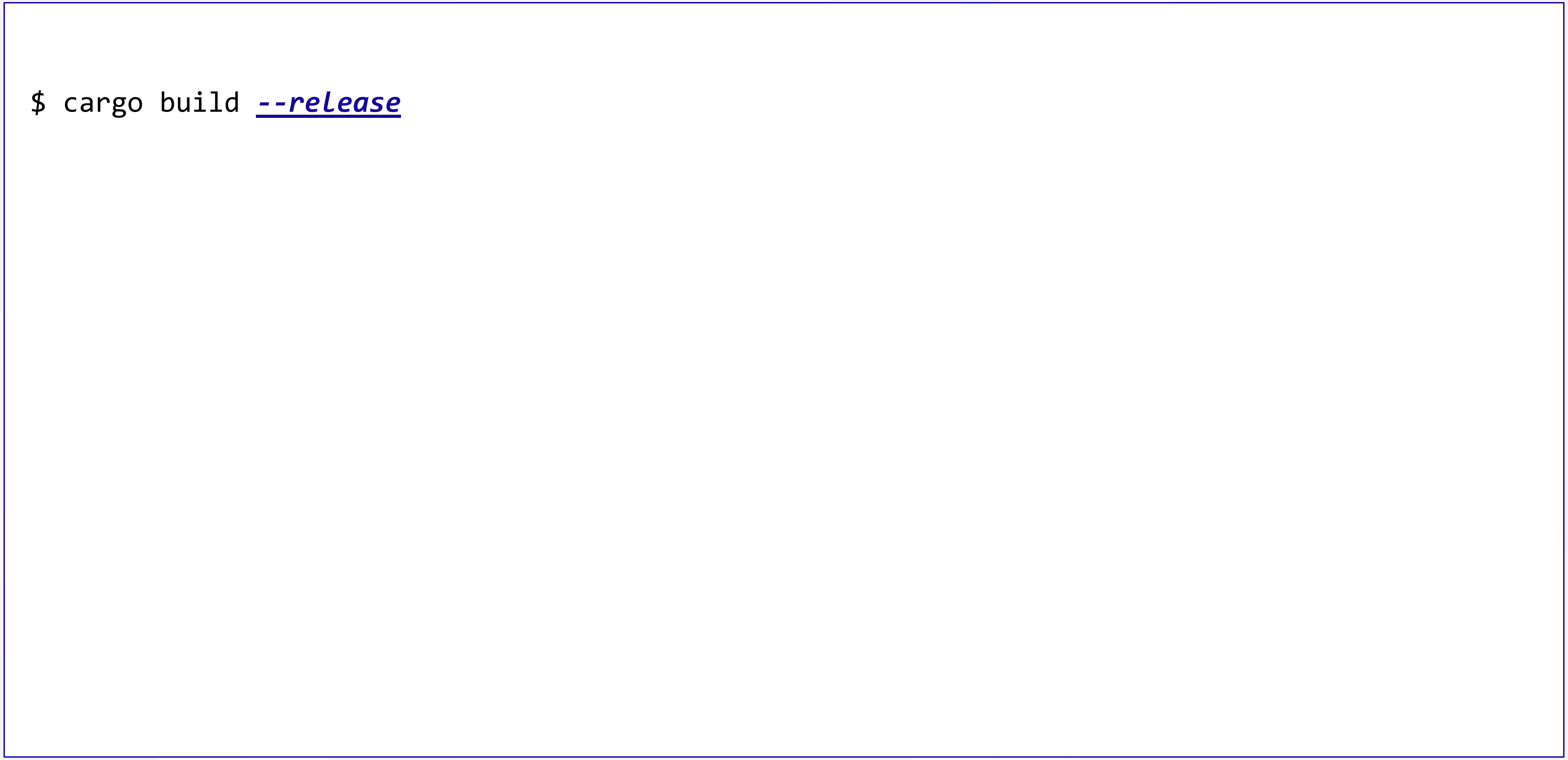
```
$ hyperfine "python ./src/0_naive_python/index-tarballs.py ./input"
Benchmark 1: python ./src/0_naive_python/index-tarballs.py ./input
  Time (mean ± σ):     345.7 ms ±  10.3 ms    [User: 328.8 ms, System: 16.9 ms]
  Range (min … max):   334.4 ms … 360.3 ms    10 runs
```

# Gut-check performance comparison

```
$ hyperfine "python ./src/0_naive_python/index-tarballs.py ./input"
Benchmark 1: python ./src/0_naive_python/index-tarballs.py ./input
  Time (mean ± σ):     345.7 ms ±  10.3 ms    [User: 328.8 ms, System: 16.9 ms]
  Range (min … max):   334.4 ms … 360.3 ms    10 runs

$ hyperfine "cargo run --bin index-tarballs-1 ./input"
Benchmark 1: cargo run --bin index-tarballs-1
  Time (mean ± σ):     658.9 ms ±  11.6 ms    [User: 387.9 ms, System: 231.4 ms]
  Range (min … max):   644.4 ms … 676.8 ms    10 runs
```

# Gut-check performance comparison



```
$ cargo build --release
```

# Gut-check performance comparison

```
$ cargo build --release


$ hyperfine "python ./src/0_naive_python/index-tarballs.py ./input"
Benchmark 1: python ./src/0_naive_python/index-tarballs.py ./input
  Time (mean ± σ):     330.1 ms ±   8.4 ms    [User: 314.8 ms, System: 15.1 ms]
  Range (min … max):   321.1 ms … 342.6 ms    10 runs
```

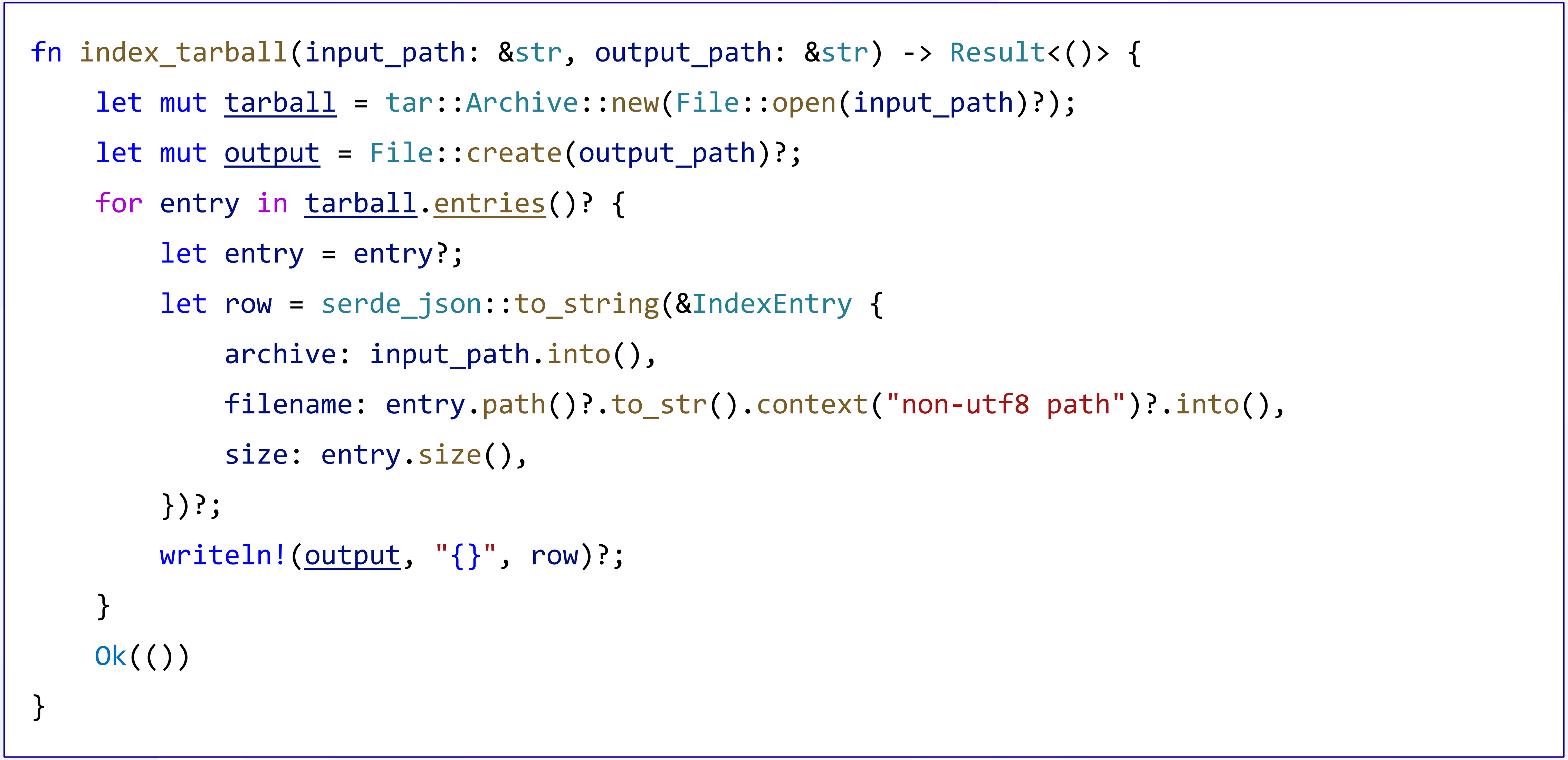
# Gut-check performance comparison

```
$ cargo build --release

$ hyperfine "python ./src/0_naive_python/index-tarballs.py ./input"
Benchmark 1: python ./src/0_naive_python/index-tarballs.py ./input
  Time (mean ± σ):     330.1 ms ±   8.4 ms    [User: 314.8 ms, System: 15.1 ms]
  Range (min … max):   321.1 ms … 342.6 ms    10 runs

$ hyperfine "./target/release/index-tarballs-1 ./input"
Benchmark 1: ./target/release/index-tarballs-1 ./input
  Time (mean ± σ):     215.7 ms ±   6.4 ms    [User: 29.6 ms, System: 186.1 ms]
  Range (min … max):   206.7 ms … 225.4 ms    14 runs
```

# Some Light Optimisation



```rust
fn index_tarball(input_path: &str, output_path: &str) -> Result<()> {
    let mut tarball = tar::Archive::new(File::open(input_path)?);
    let mut output = File::create(output_path)?;
    for entry in tarball.entries()? {
        let entry = entry?;
        let row = serde_json::to_string(&IndexEntry {
            archive: input_path.into(),
            filename: entry.path()?.to_str().context("non-utf8 path")?.into(),
            size: entry.size(),
        })?;
        writeln!(output, "{}", row)?;
    }
    Ok(())
}
```

# Some Light Optimisation

```rust
fn index_tarball(input_path: &str, output_path: &str) -> Result<()> {
    let mut tarball = tar::Archive::new(File::open(input_path)?);
    let mut output = File::create(output_path)?;
    for entry in tarball.entries()? {
        let entry = entry?;
        let row = serde_json::to_string(&IndexEntry {
            archive: input_path.into(),
            filename: entry.path()?.to_str().context("non-utf8 path")?.into(),
            size: entry.size(),
        })?;
        writeln!(output, "{}", row)?;
    }
    Ok(())
}
```

Unnecessary string copies

# Some Light Optimisation

```
fn index_tarball(input_path: &str, output_path: &str) -> Result<()> {
    let mut tarball = tar::Archive::new(File::open(input_path)?);
    let mut output = File::create(output_path)?;
    for entry in tarball.entries()? {
        let entry = entry?;
        let row = serde_json::to_string(&IndexEntry {
            archive: input_path,
            filename: entry.path()?.to_str().context("non-utf8 path")?,
            size: entry.size(),
        })?;
        writeln!(output, "{}", row)?;
    }
    Ok(())
}
```

# Some Light Optimisation

```
fn index_tarball(input_path: &str, output_path: &str) -> Result<()> {
    let mut tarball = tar::Archive::new(File::open(input_path)?);
    let mut output = File::create(output_path)?;
    for entry in tarball.entries()? {
        let entry = entry?;
        let row = serde_json::to_string(&IndexEntry {
            archive: input_path,
            filename: entry.path()?.to_str().context("non-utf8 path")?,
            size: entry.size(),
        })?;
        writeln!(output, "{}", row)?;
    }
    Ok(())
}
```

Unnecessary intermediate string

# Some Light Optimisation

```rust
fn index_tarball(input_path: &str, output_path: &str) -> Result<()> {
    let mut tarball = tar::Archive::new(File::open(input_path)?);
    let mut output = File::create(output_path)?;
    for entry in tarball.entries()? {
        let entry = entry?;
        serde_json::to_writer(&mut output, &IndexEntry {
            archive: input_path,
            filename: entry.path()?.to_str().context("non-utf8 path")?,
            size: entry.size(),
        })?;
        writeln!(output)?;
    }
    Ok(())
}
```

# Some Light Optimisation

```rust
fn index_tarball(input_path: &str, output_path: &str) -> Result<()> {
    let mut tarball = tar::Archive::new(File::open(input_path)?);
    let mut output = File::create(output_path)?;
    for entry in tarball.entries()? {
        let entry = entry?;
        serde_json::to_writer(&mut output, &IndexEntry {
            archive: input_path,
            filename: entry.path()?.to_str().context("non-utf8 path")?,
            size: entry.size(),
        })?;
        writeln!(output)?;
    }
    Ok(())
}
```

Lots of small reads/writes

# Some Light Optimisation

```rust
fn index_tarball(input_path: &str, output_path: &str) -> Result<()> {
    let mut tarball = tar::Archive::new(BufReader::new(File::open(input_path)?));
    let mut output = BufWriter::new(File::create(output_path)?);
    for entry in tarball.entries()? {
        let entry = entry?;
        serde_json::to_writer(&mut output, &IndexEntry {
            archive: input_path,
            filename: entry.path()?.to_str().context("non-utf8 path")?,
            size: entry.size(),
        })?;
        writeln!(output)?;
    }
    Ok(())
}
```

# Gut-check performance comparison

```
$ hyperfine "python ./src/0_naive_python/index-tarballs.py ./input"
Benchmark 1: python ./src/0_naive_python/index-tarballs.py ./input
  Time (mean ± σ):     330.1 ms ±   8.4 ms    [User: 314.8 ms, System: 15.1 ms]
  Range (min … max):   321.1 ms … 342.6 ms    10 runs

$ hyperfine "./target/release/index-tarballs-2 ./input"
Benchmark 1: ./target/release/index-tarballs-2 ./input
  Time (mean ± σ):     130.5 ms ±   5.8 ms    [User: 13.0 ms, System: 117.5 ms]
  Range (min … max):   123.6 ms … 144.6 ms    20 runs
```

# So anyway…AWS?

# A Python Baseline

```python
import boto3


def index_tarball(s3client, bucket, input_key, output_key):
    input = s3client.get_object(Bucket=bucket, Key=input_key)["Body"]

    output = BytesIO()
    with tarfile.open(fileobj=input, mode="r|") as tarball:
        for member in tarball:
            row = json.dumps(
                {"archive": input_key, "filename": member.name, "size": member.size}
            )
            output.write(row.encode("utf-8"))
            output.write(b"\n")

    output.seek(0)
    s3client.put_object(
        Bucket=bucket,
        Key=output_key,
        Body=output,
    )
```

# A Rust Equivalent

```toml
[dependencies]
anyhow = "1"
async-tar = "0.4"
aws-config = "0.49"
aws-sdk-s3 = "0.19"
lambda_runtime = "0.6"
aws_lambda_events = "0.5"
futures = "0.3"
tokio = { version = "1", features=["full"] }
serde = { version = "1", features=["derive"] }
serde_json = "1"
```

# A Rust Equivalent

```
[dependencies]
anyhow = "1"
async-tar = "0.4"
aws-config = "0.49"
aws-sdk-s3 = "0.19"
lambda_runtime = "0.6"
aws_lambda_events = "0.5"
futures = "0.3"
tokio = { version = "1", features=["full"] }
serde = { version = "1", features=["derive"] }
serde_json = "1"
```

Need a different tar crate…

# A Rust Equivalent

...because it’s all going to be async

```rust
async fn index_tarball(
    client: &s3::Client,
    bucket: &str,
    input_key: &str,
    output_key: &str,
) -> Result<()> {

    // TODO: asyncify the previous code


}
```

# A Rust Equivalent

```
let tarball = async_tar::Archive::new(
    client
        .get_object()
        .bucket(bucket)
        .key(input_key)
        .send()
        .await?
        .body
        .map_err(|e| std::io::Error::new(std::io::ErrorKind::Other, e))
        .into_async_read(),
);
```

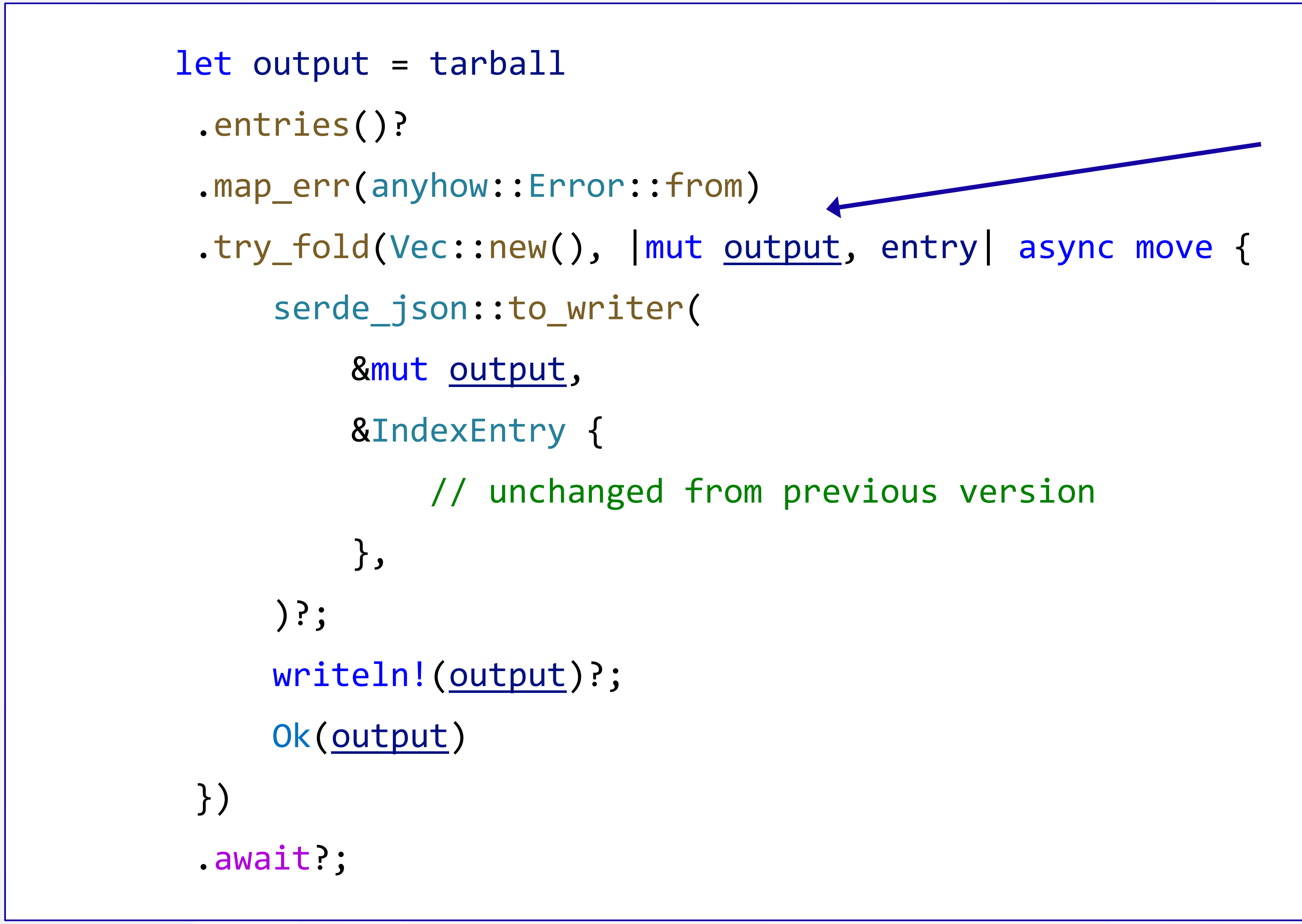
make it impl `AsyncRead`

# A Rust Equivalent

```rust
let mut output = Vec::new();
let mut entries = tarball.entries()?;
while let Some(entry) = entries.try_next().await? {
    serde_json::to_writer(
        &mut output,
        &IndexEntry {
            // unchanged from previous version
        },
    )?;
    writeln!(output)?;
}
```

This is now a `Stream`, not an `Iterator`

# A Rust Equivalent



```
let output = tarball
 .entries()?
 .map_err(anyhow::Error::from)
 .try_fold(Vec::new(), |mut output, entry| async move {
     serde_json::to_writer(
         &mut output,
         &IndexEntry {
             // unchanged from previous version
         },
     )?;
     writeln!(output)?;
     Ok(output)
 })
 .await?;
```

`TryStreamExt` has a lot of powerful helper methods

harrison.ai

# A Rust Equivalent

```
client
    .put_object()
    .bucket(bucket)
    .key(output_key)
    .body(output.into())
    .send()
    .await?;
```

# A Rust Lambda

```
use cobalt_aws::lambda::{run_message_handler, Error};

#[tokio::main]
async fn main() -> Result<(), Error> {
    run_message_handler(message_handler).await
}

// Not shown: impls to populate this from env vars
struct Context {
    client: s3::Client,
    bucket: String,
}

async fn message_handler(input_key: String, context: Arc<Context>) -> Result<()> {
    let output_key = … // Not shown, for brevity
    index_tarball(&context.client, &context.bucket, input_key, &output_key)
        .await?;
}
```

# Deploy via Docker Image

```
RUN mkdir /bin

RUN --mount=type=cache,target=/usr/local/cargo/registry \
    --mount=type=cache,target=/build/target \
    cargo build --profile release --target x86_64-unknown-linux-musl

RUN mv target/x86_64-unknown-linux-musl/index-tarballs-lambda /bin

ENTRYPOINT ["/bin/index-tarballs-lambda"]
```

# Deploy via Docker Image

```
RUN mkdir /bin

RUN --mount=type=cache,target=/usr/local/cargo/registry \
    --mount=type=cache,target=/build/target \
    cargo build --profile release --target aarch64-unknown-linux-musl

RUN mv target/x86_64-unknown-linux-musl/index-tarballs-lambda /bin

ENTRYPOINT ["/bin/index-tarballs-lambda"]
```

Amazon suggest that this arch is usually cheaper

# Rough Performance Numbers

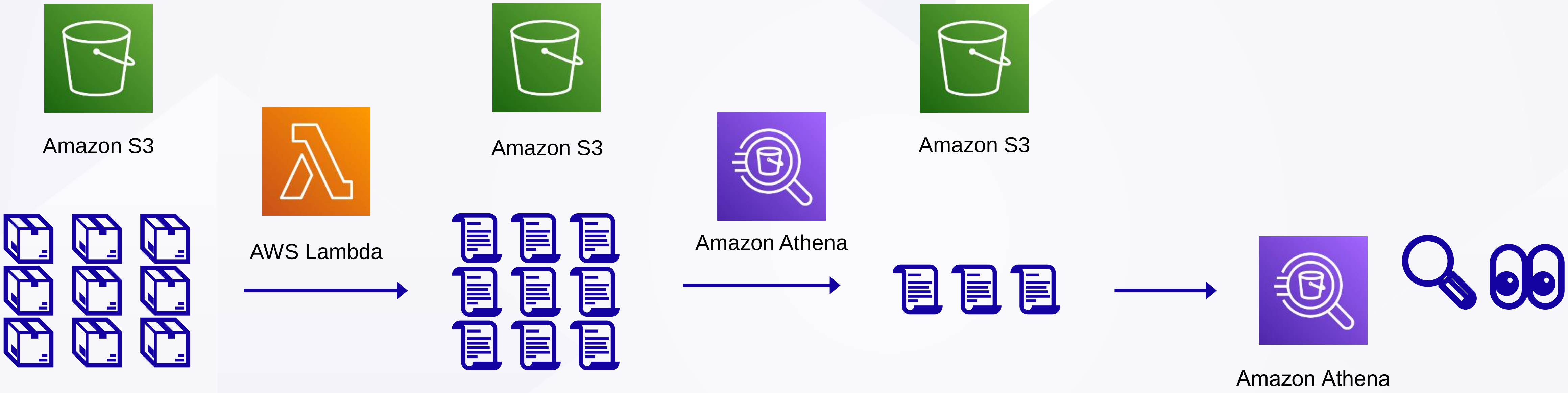
| | architecture = x86_64 | architecture = arm64 |
|---|---|---|
| Python | 14.5 seconds avg | 14.3 seconds avg |
| Rust | 7.3 seconds avg | 7.8 seconds avg |

# Rough Performance Numbers

| | architecture = x86_64 | architecture = arm64 |
|---|---|---|
| Python | $241 / mil | $190 / mil |
| Rust | $121 / mil | $104 / mil |

# The IRL Version…

- Generated 6 different listings per tarball, with rich metadata
- Partitioned output files by prefix, to help Athena
- Used transparent zstd compression on input and output files
  - (thanks, `AsyncRead`/`AsyncWrite` traits!)
- Cost O($100) in lambda execution time



Amazon S3

AWS Lambda

Amazon S3

Amazon Athena

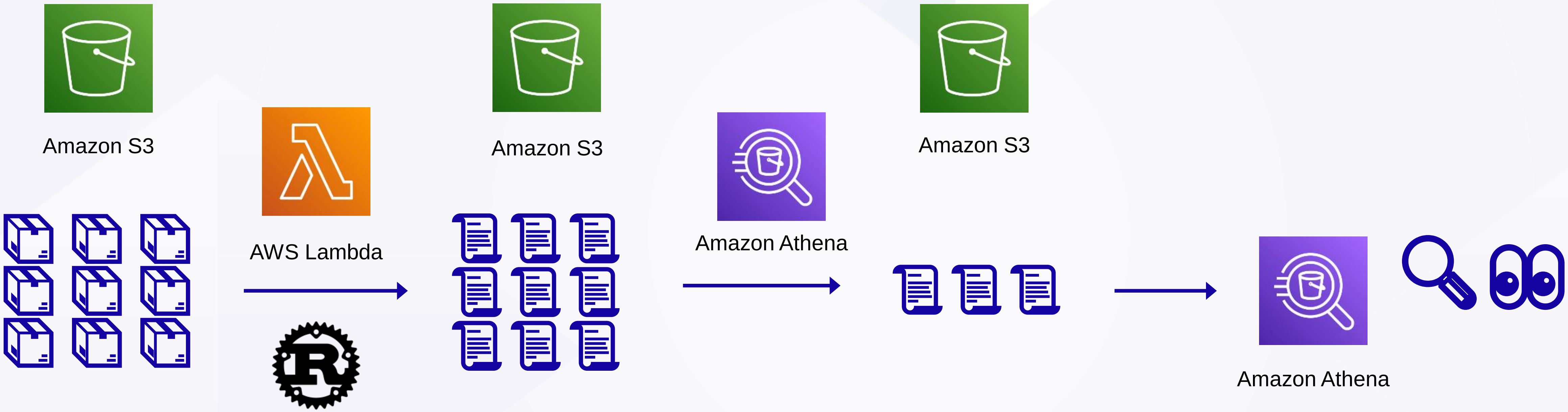
Amazon S3

Amazon Athena

A few PB of de-identified medical image data in millions tar archives

.jsonl files listing contents of each archive

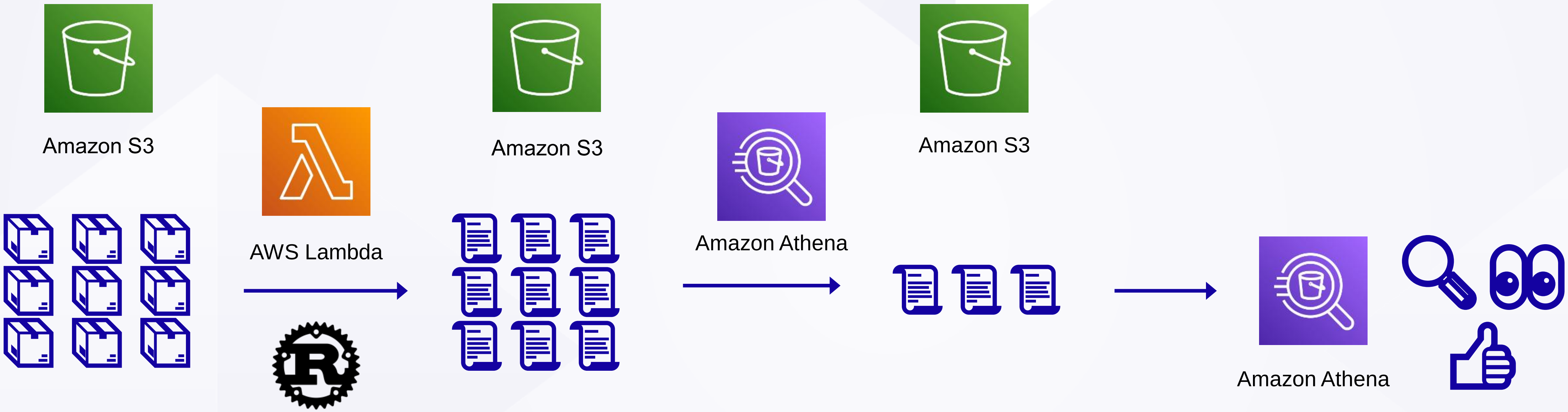
.parquet files summarizing available data



A few PB of de-identified medical image data in millions tar archives

.jsonl files listing contents of each archive

.parquet files summarizing available data



A few PB of de-identified medical image data in millions tar archives

.jsonl files listing contents of each archive

.parquet files summarizing available data

# Reflections

## Things we liked

- Runtime performance
- Low, stable memory usage
- Runtime robustness
  - “If it compiles, it works!”
- High-level abstractions
- Powerful async helpers
- Ease of build/packaging

## Things that were challenging

- Async ecosystem fragmentation
- Testing/mocking
  - plug: LocalStack
- Runtime debugging context
  - plug: tracing
- Optimisation opportunities are an attractive nuisance

# So, would we do it again?

# We’re making this a core competency

## And trying to open-source where we can

- Higher-level abstractions for working with AWS
  - https://github.com/harrison-ai/cobalt-aws/

- Docker-based build tooling
  - https://github.com/harrison-ai/dataeng-tooling-rust/

# thank you.

Ryan Kelly

https://rfk.id.au/

harrison.ai