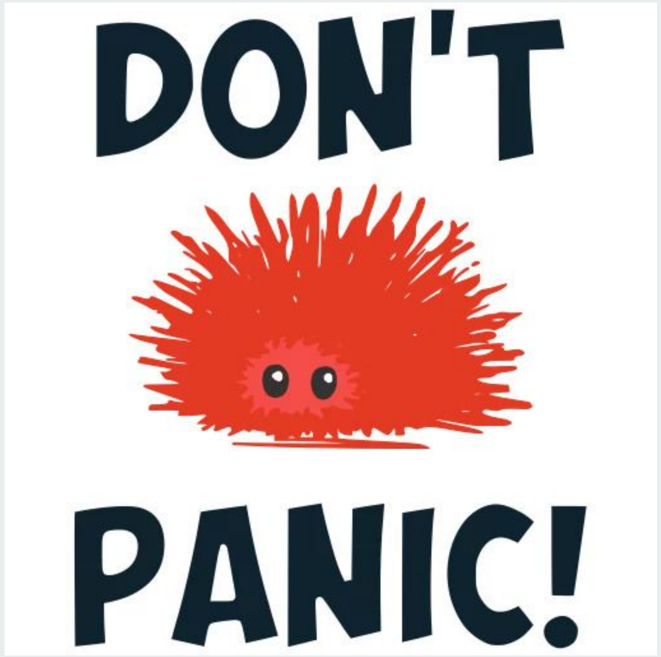
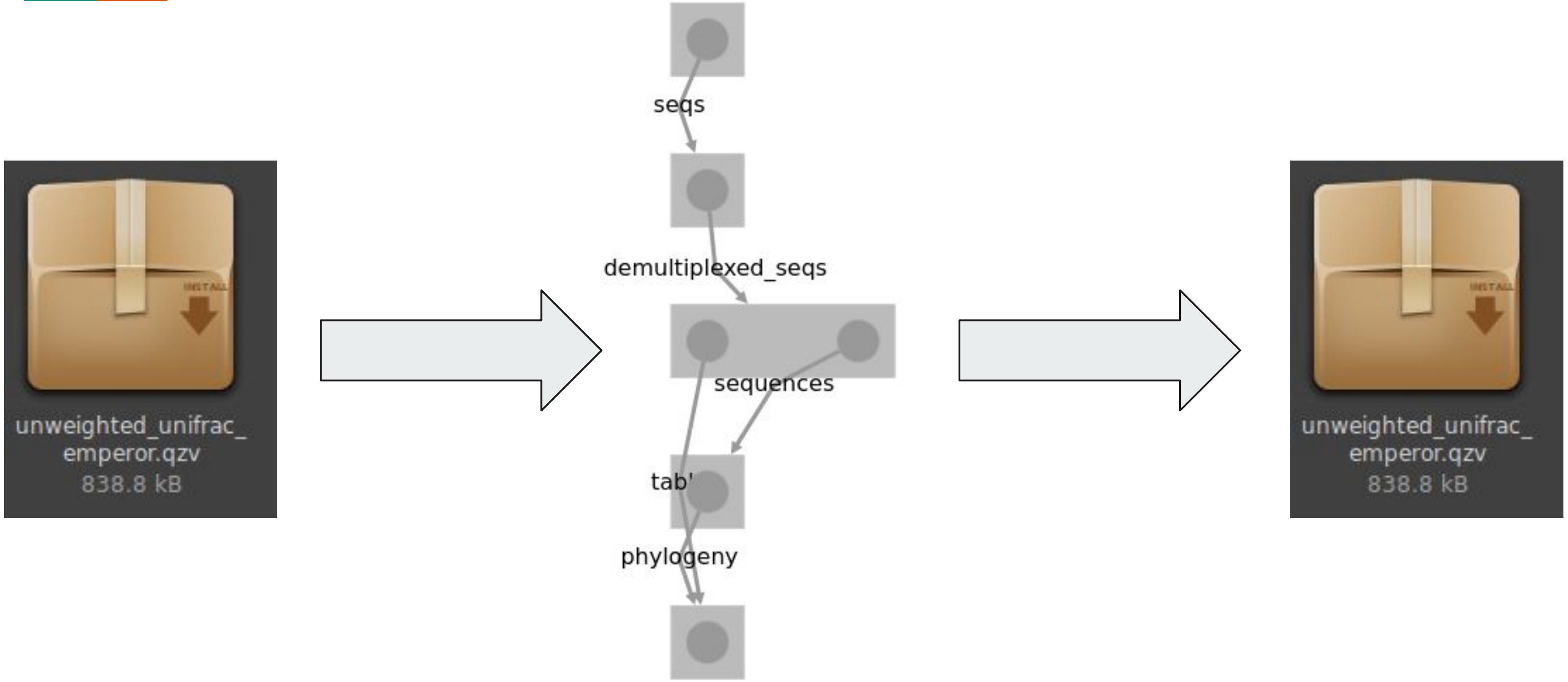
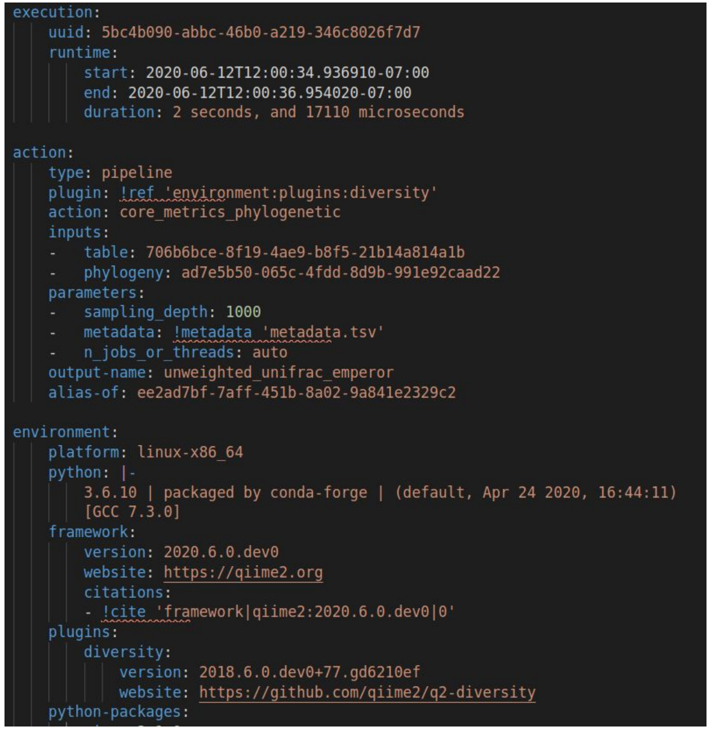


A ramble through data deserialization in Rust, with Chris Keefe

Long-term goals

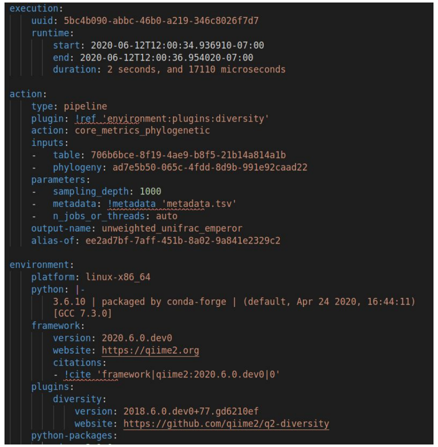

unweighted_unifrac_emperor.qzv
838.8 kB
seqs
demultiplexed_seqs
sequences
tab'
phylogeny
unweighted_unifrac_emperor.qzv
838.8 kB

Short-term goals



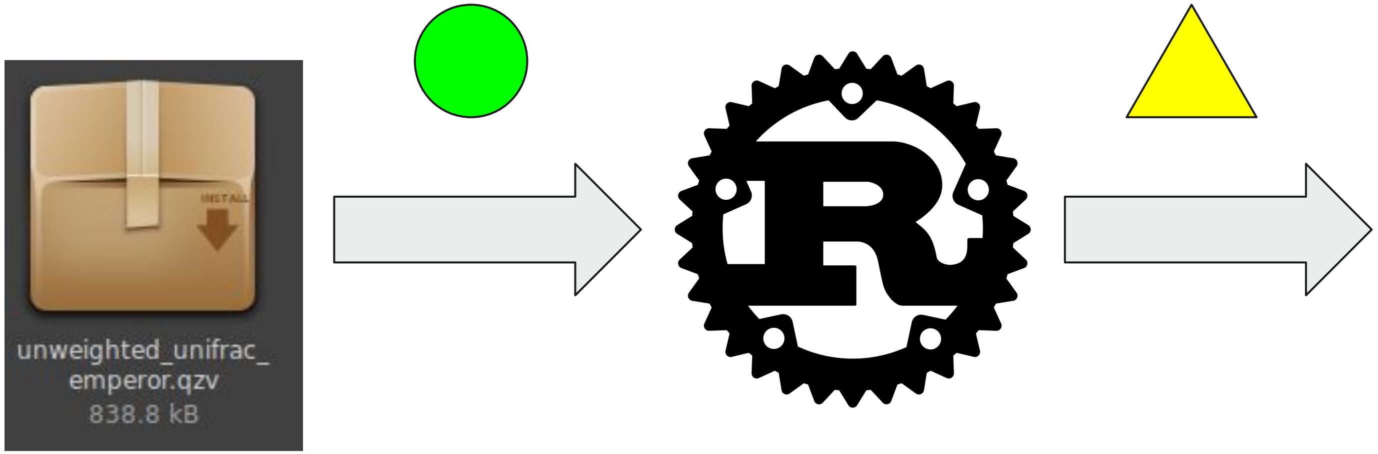
```
execution:
    uuid: 5bc4b090-abbc-46b0-a219-346c8026f7d7
    runtime:
        start: 2020-06-12T12:00:34.936910-07:00
        end: 2020-06-12T12:00:36.954020-07:00
        duration: 2 seconds, and 17110 microseconds

action:
    type: pipeline
    plugin: !ref 'environment:plugins:diversity'
    action: core_metrics_phylogenetic
    inputs:
    -   table: 706b6bce-8f19-4ae9-b8f5-21b14a814a1b
    -   phylogeny: ad7e5b50-065c-4fdd-8d9b-991e92caad22
    parameters:
    -   sampling_depth: 1000
    -   metadata: !metadata 'metadata.tsv'
    -   n_jobs_or_threads: auto
    output-name: unweighted_unifrac_emperor
    alias-of: ee2ad7bf-7aff-451b-8a02-9a841e2329c2

environment:
    platform: linux-x86_64
    python: |-
        3.6.10 | packaged by conda-forge | (default, Apr 24 2020, 16:44:11)
        [GCC 7.3.0]
    framework:
        version: 2020.6.0.dev0
        website: https://qiime2.org
        citations:
        - !cite 'framework|qiime2:2020.6.0.dev0|0'
    plugins:
        diversity:
            version: 2018.6.0.dev0+77.gd6210ef
            website: https://github.com/qiime2/q2-diversity
    python-packages:
```

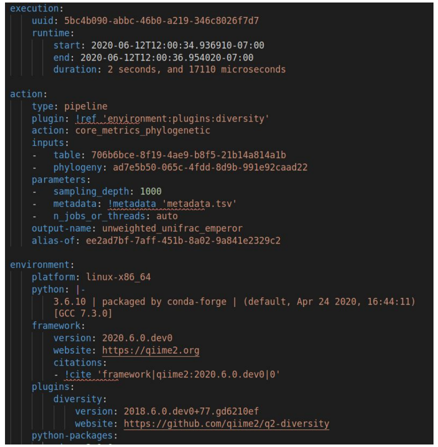
Short-term goals



```
execution:
    uuid: 5bc4b090-abbc-46b0-a219-346c8026f7d7
    runtime:
        start: 2020-06-12T12:00:34.936910-07:00
        end: 2020-06-12T12:00:36.954020-07:00
        duration: 2 seconds, and 17110 microseconds

action:
    type: pipeline
    plugin: !ref 'environment:plugins:diversity'
    action: core_metrics_phylogenetic
    inputs:
    -   table: 706b6bce-8f19-4ae9-b8f5-21b14a814a1b
    -   phylogeny: ad7e5b50-065c-4fdd-8d9b-991e92caad22
    parameters:
    -   sampling_depth: 1000
    -   metadata: !metadata 'metadata.tsv'
    -   n_jobs_or_threads: auto
    output-name: unweighted_unifrac_emperor
    alias-of: ee2ad7bf-7aff-451b-8a02-9a841e2329c2

environment:
    platform: linux-x86_64
    python: |-
        3.6.10 | packaged by conda-forge | (default, Apr 24 2020, 16:44:11)
        [GCC 7.3.0]
    framework:
        version: 2020.6.0.dev0
        website: https://qiime2.org
        citations:
        - !cite 'framework|qiime2:2020.6.0.dev0|0'
    plugins:
        diversity:
            version: 2018.6.0.dev0+77.gd6210ef
            website: https://github.com/qiime2/q2-diversity
    python-packages:
```

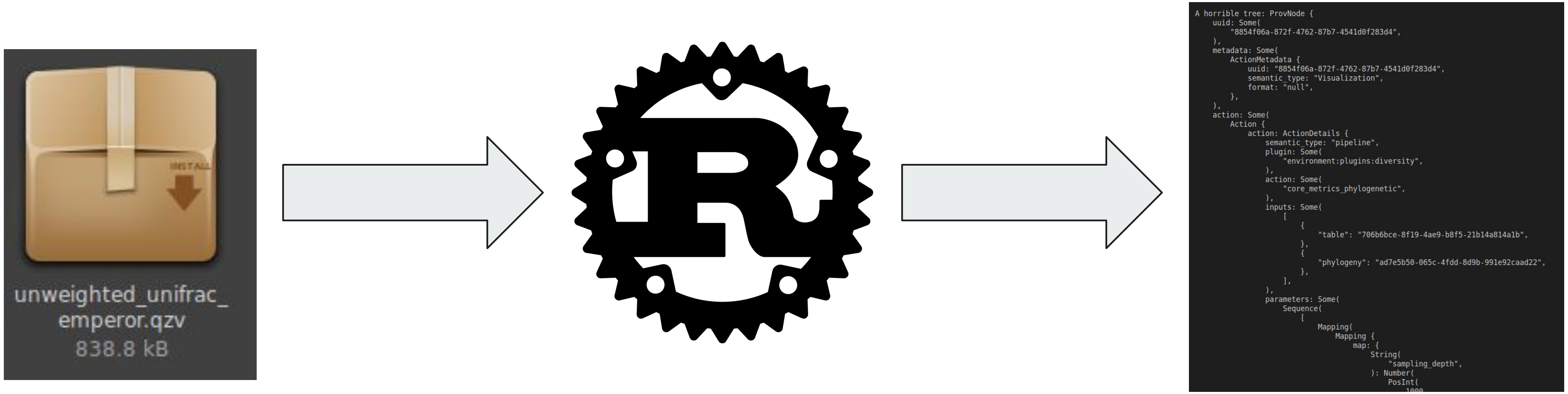
We're nearly there!





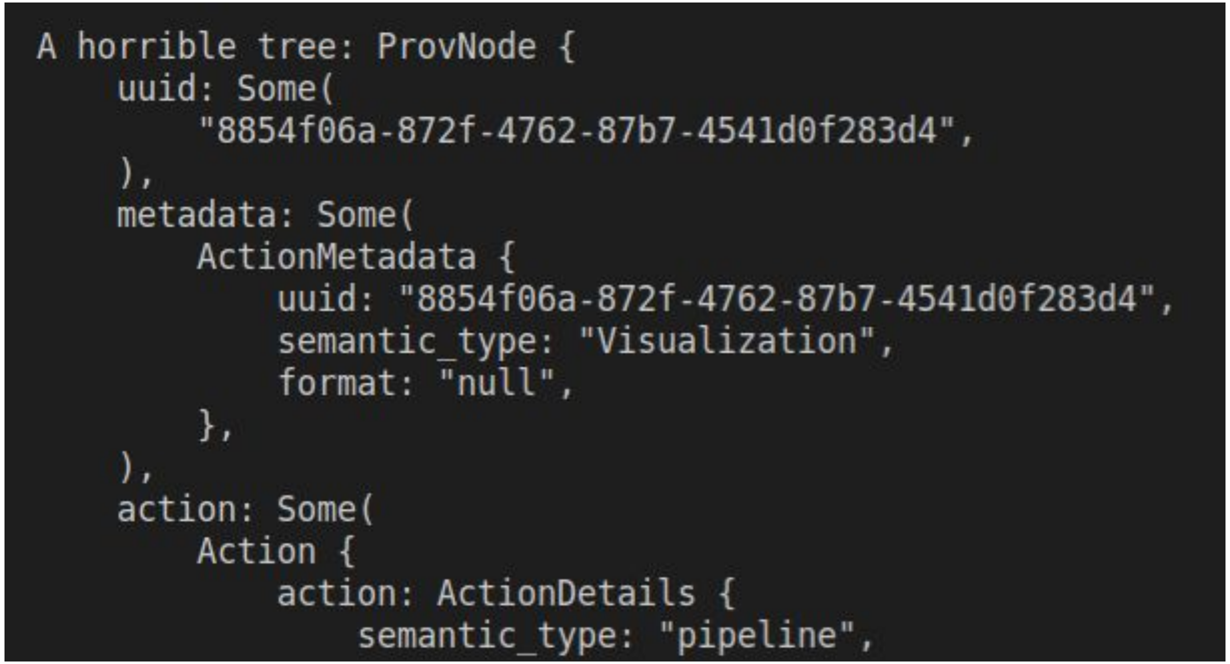
```
execution:
    uuid: 5bc4b090-abbc-46b0-a219-346c8026f7d7
    runtime:
        start: 2020-06-12T12:00:34.936910-07:00
        end: 2020-06-12T12:00:36.954020-07:00
        duration: 2 seconds, and 17110 microseconds

action:
    type: pipeline
    plugin: !ref 'environment:plugins:diversity'
    action: core_metrics_phylogenetic
    inputs:
    -   table: 706b6bce-8f19-4ae9-b8f5-21b14a814a1b
    -   phylogeny: ad7e5b50-065c-4fdd-8d9b-991e92caad22
    parameters:
    -   sampling_depth: 1000
    -   metadata: !metadata 'metadata.tsv'
    -   n_jobs_or_threads: auto
    output-name: unweighted_unifrac_emperor
    alias-of: ee2ad7bf-7aff-451b-8a02-9a841e2329c2

environment:
    platform: linux-x86_64
    python: |-
        3.6.10 | packaged by conda-forge | (default, Apr 24 2020, 16:44:11)
        [GCC 7.3.0]
    framework:
        version: 2020.6.0.dev0
        website: https://qiime2.org
        citations:
        - !cite 'framework|qiime2:2020.6.0.dev0|0'
    plugins:
        diversity:
            version: 2018.6.0.dev0+77.gd6210ef
            website: https://github.com/qiime2/q2-diversity
    python-packages:
```

We’re nearly there!



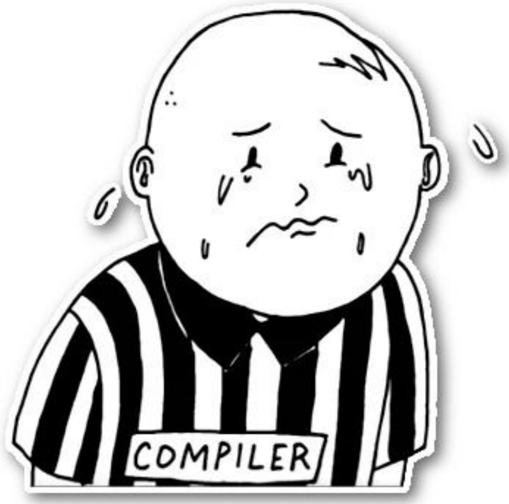
```
A horrible tree: ProvNode {
    uuid: Some(
        "8854f06a-872f-4762-87b7-4541d0f283d4",
    ),
    metadata: Some(
        ActionMetadata {
            uuid: "8854f06a-872f-4762-87b7-4541d0f283d4",
            semantic_type: "Visualization",
            format: "null",
        },
    ),
    action: Some(
        Action {
            action: ActionDetails {
                semantic_type: "pipeline",
                plugin: Some(
                    "environment:plugins:diversity",
                ),
                action: Some(
                    "core_metrics_phylogenetic",
                ),
                inputs: Some(
                    [
                        {
                            "table": "706b6bce-8f19-4ae9-b8f5-21b14a814a1b",
                        },
                        {
                            "phylogeny": "ad7e5b50-065c-4fdd-8d9b-991e92caad22",
                        },
                    ],
                ),
                parameters: Some(
                    Sequence(
                        [
                            Mapping(
                                Mapping {
                                    map: {
                                        String(
                                            "sampling_depth",
                                        ): Number(
                                            PosInt(
                                                1000
```

We're nearly there!



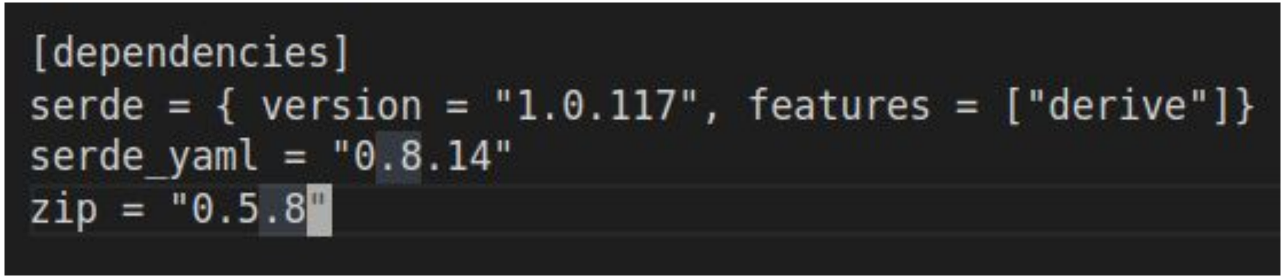
```
A horrible tree: ProvNode {
    uuid: Some(
        "8854f06a-872f-4762-87b7-4541d0f283d4",
    ),
    metadata: Some(
        ActionMetadata {
            uuid: "8854f06a-872f-4762-87b7-4541d0f283d4",
            semantic_type: "Visualization",
            format: "null",
        },
    ),
    action: Some(
        Action {
            action: ActionDetails {
                semantic_type: "pipeline",
```

Lesson 1: Writing Rust takes time.



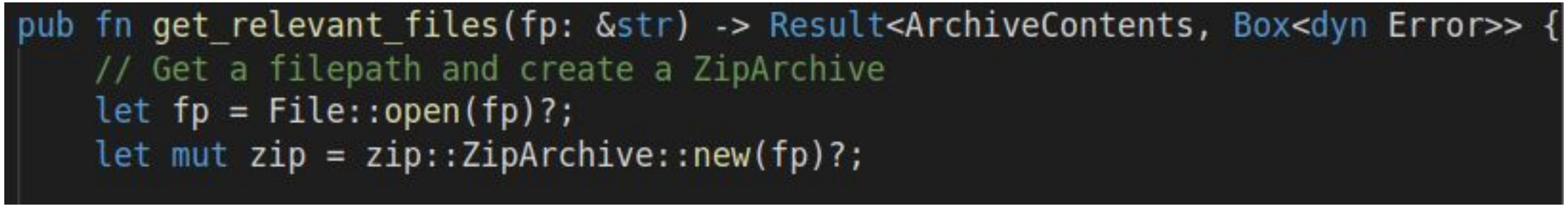
Lesson 1: Writing Rust takes time.

Lesson 2: Rust has neat libraries!



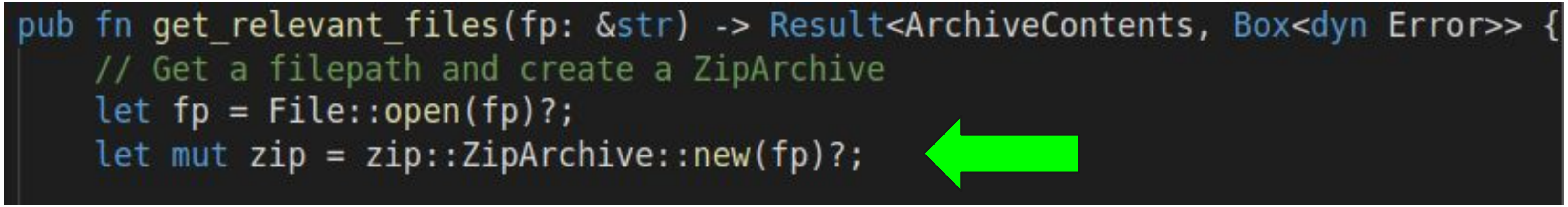
```toml
[dependencies]
serde = { version = "1.0.117", features = ["derive"]}
serde_yaml = "0.8.14"
zip = "0.5.8"
```

Lesson 2a: Create a ZipArchive...



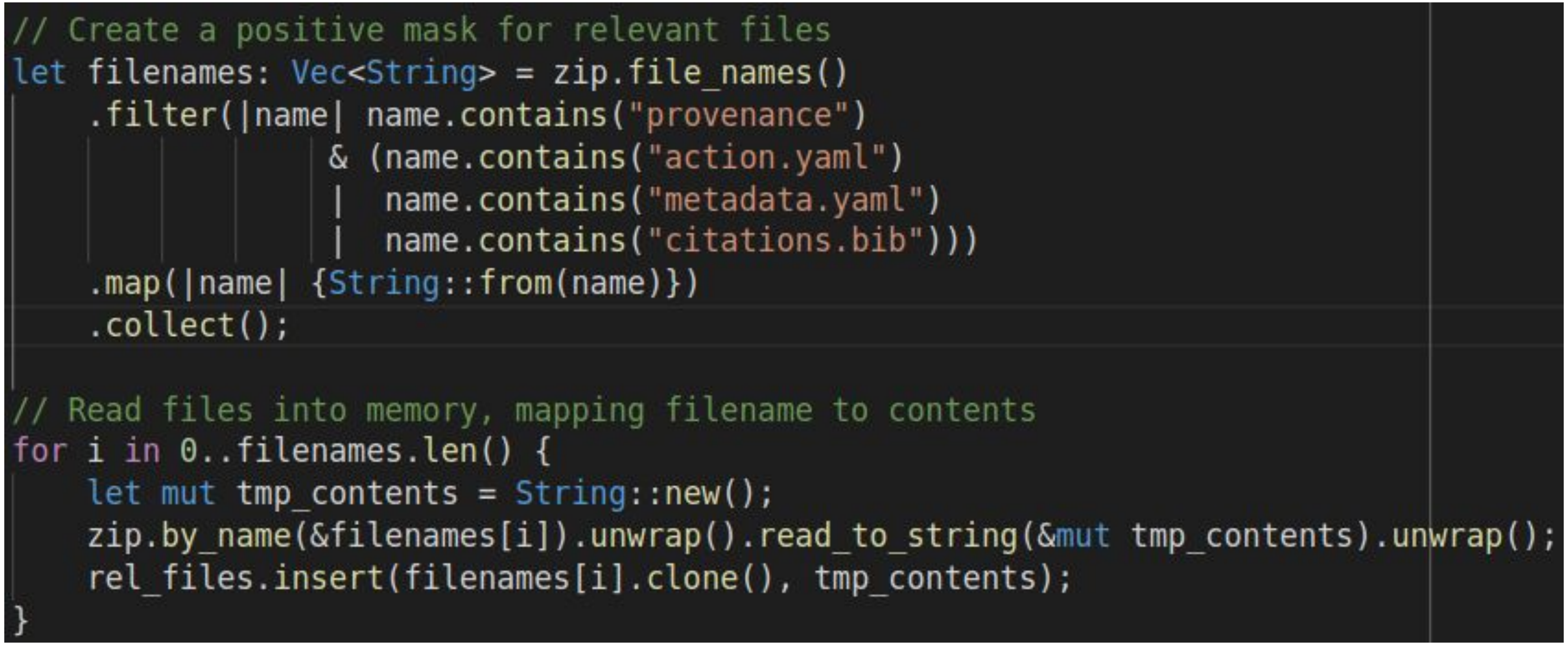
```
pub fn get_relevant_files(fp: &str) -> Result<ArchiveContents, Box<dyn Error>> {
    // Get a filepath and create a ZipArchive
    let fp = File::open(fp)?;
    let mut zip = zip::ZipArchive::new(fp)?;
```

Lesson 2a: Create a ZipArchive...



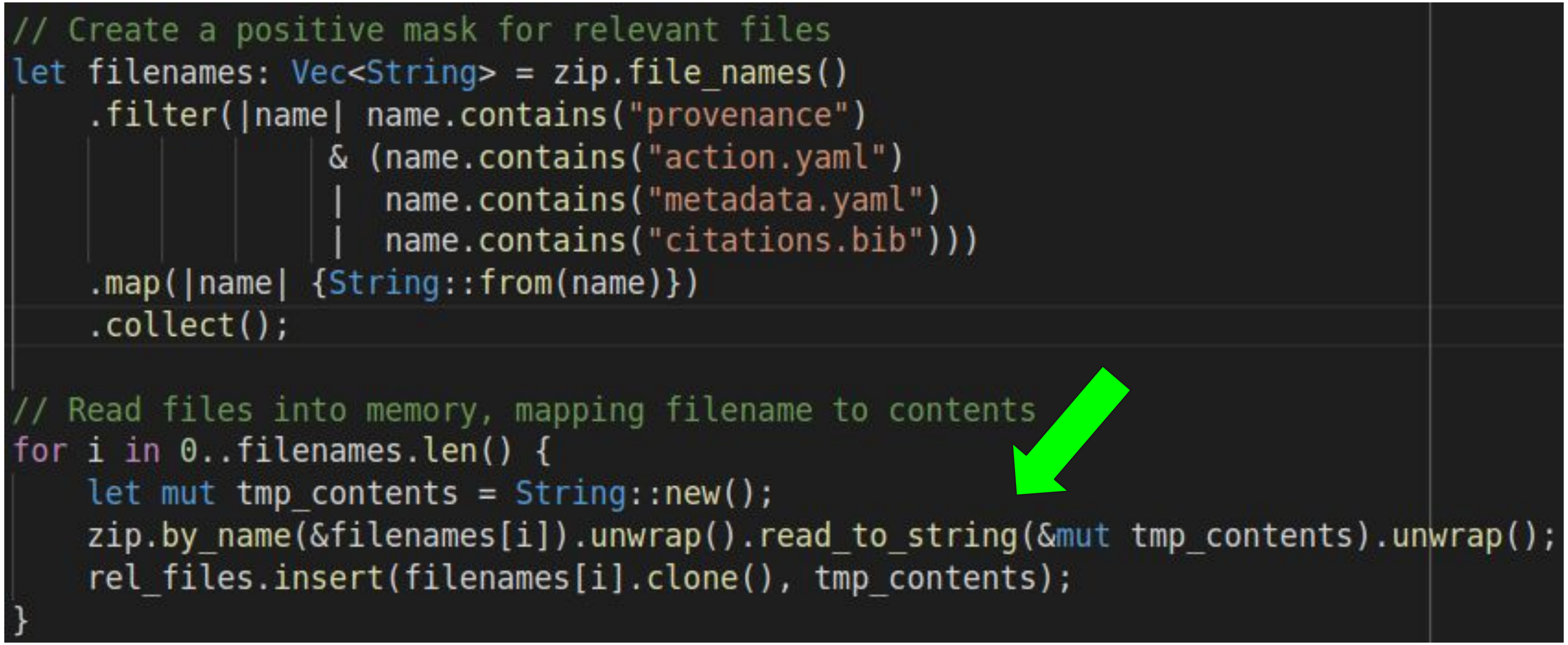
```
pub fn get_relevant_files(fp: &str) -> Result<ArchiveContents, Box<dyn Error>> {
    // Get a filepath and create a ZipArchive
    let fp = File::open(fp)?;
    let mut zip = zip::ZipArchive::new(fp)?;
```

Lesson 2a: … and read files into memory.



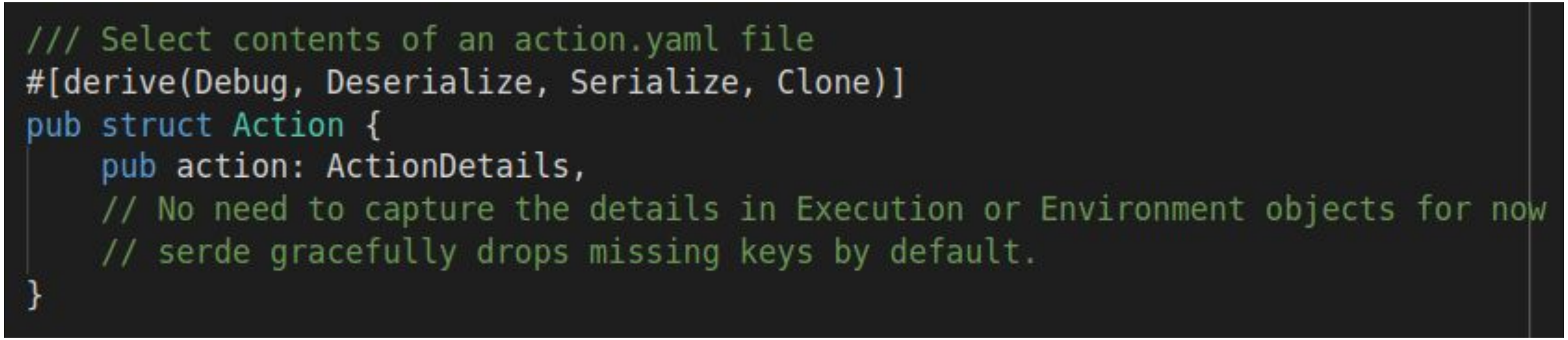
```rust
// Create a positive mask for relevant files
let filenames: Vec<String> = zip.file_names()
    .filter(|name| name.contains("provenance")
                 & (name.contains("action.yaml")
                 |  name.contains("metadata.yaml")
                 |  name.contains("citations.bib")))
    .map(|name| {String::from(name)})
    .collect();

// Read files into memory, mapping filename to contents
for i in 0..filenames.len() {
    let mut tmp_contents = String::new();
    zip.by_name(&filenames[i]).unwrap().read_to_string(&mut tmp_contents).unwrap();
    rel_files.insert(filenames[i].clone(), tmp_contents);
}
```

Lesson 2a: … and read files into memory.



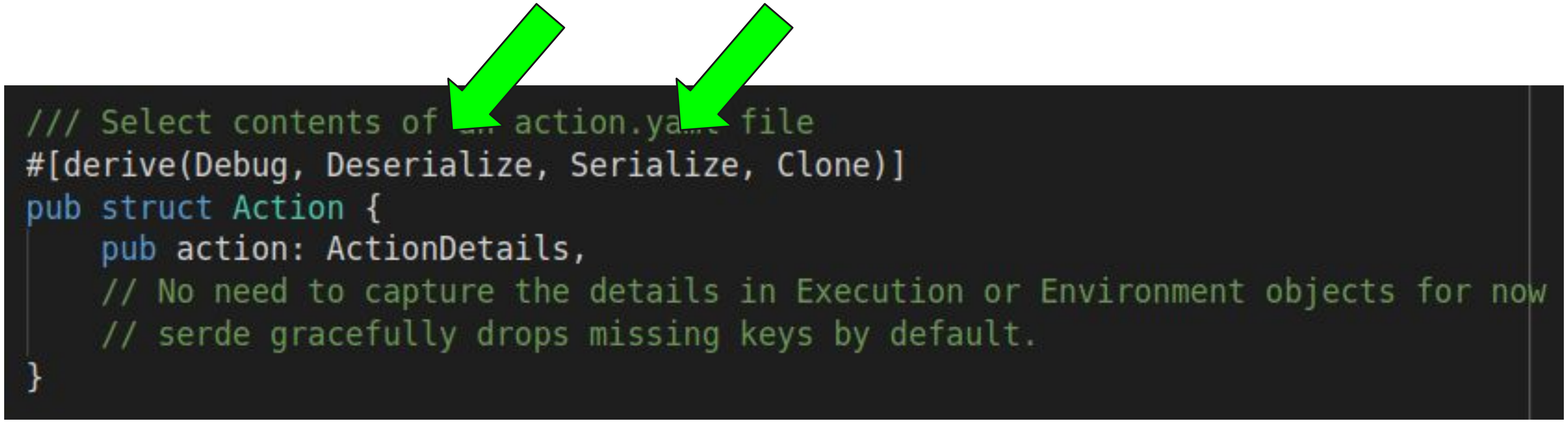
```rust
// Create a positive mask for relevant files
let filenames: Vec<String> = zip.file_names()
    .filter(|name| name.contains("provenance")
                & (name.contains("action.yaml")
                |  name.contains("metadata.yaml")
                |  name.contains("citations.bib")))
    .map(|name| {String::from(name)})
    .collect();

// Read files into memory, mapping filename to contents
for i in 0..filenames.len() {
    let mut tmp_contents = String::new();
    zip.by_name(&filenames[i]).unwrap().read_to_string(&mut tmp_contents).unwrap();
    rel_files.insert(filenames[i].clone(), tmp_contents);
}
```

Lesson 2b: Serde - define a struct...



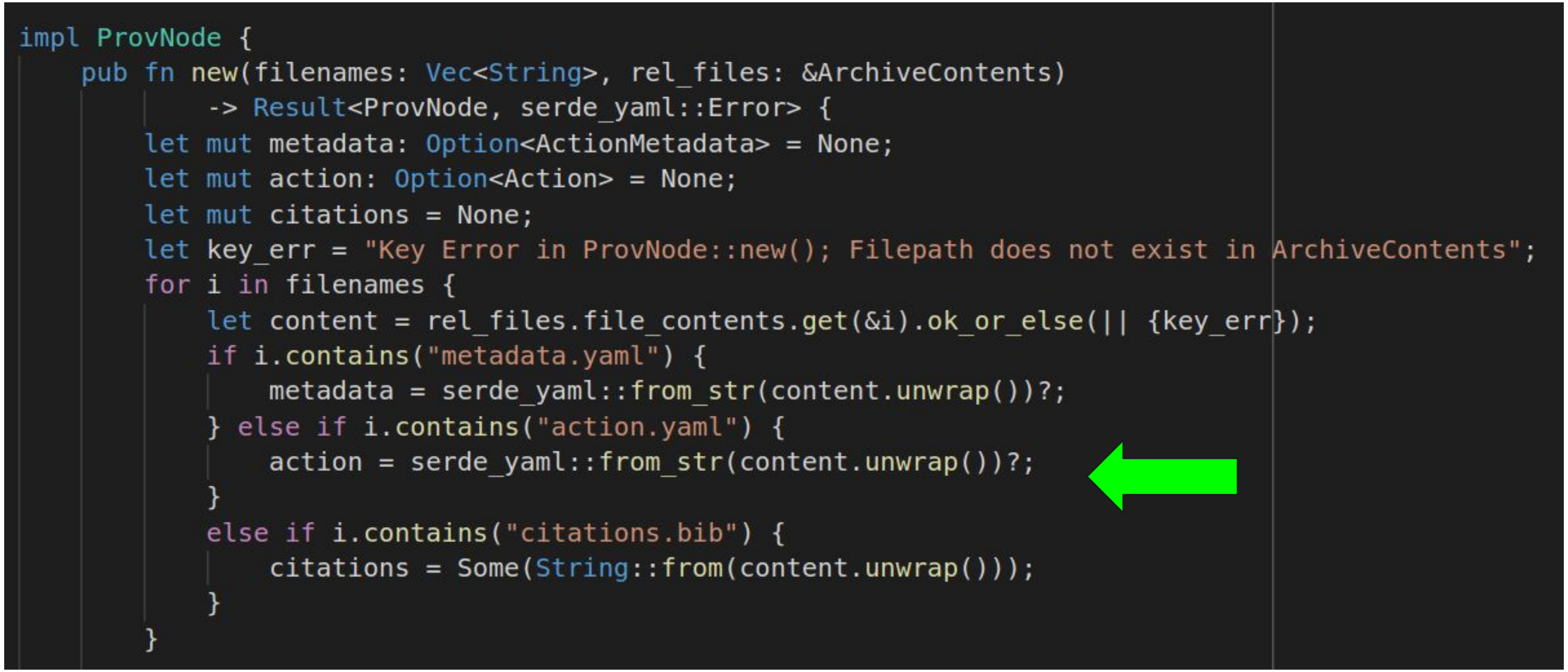
```rust
/// Select contents of an action.yaml file
#[derive(Debug, Deserialize, Serialize, Clone)]
pub struct Action {
    pub action: ActionDetails,
    // No need to capture the details in Execution or Environment objects for now
    // serde gracefully drops missing keys by default.
}
```

Lesson 2b: ...derive these Traits...



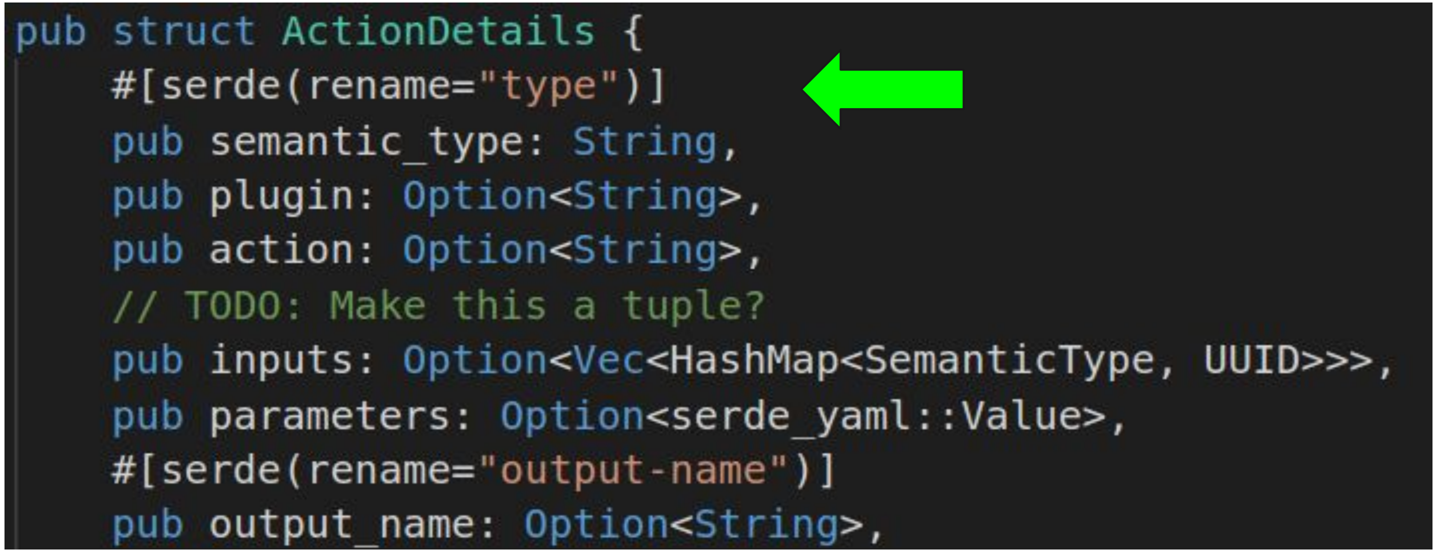
```rust
/// Select contents of an action.yaml file
#[derive(Debug, Deserialize, Serialize, Clone)]
pub struct Action {
    pub action: ActionDetails,
    // No need to capture the details in Execution or Environment objects for now
    // serde gracefully drops missing keys by default.
}
```

Lesson 2b: ... serde handles the rest.



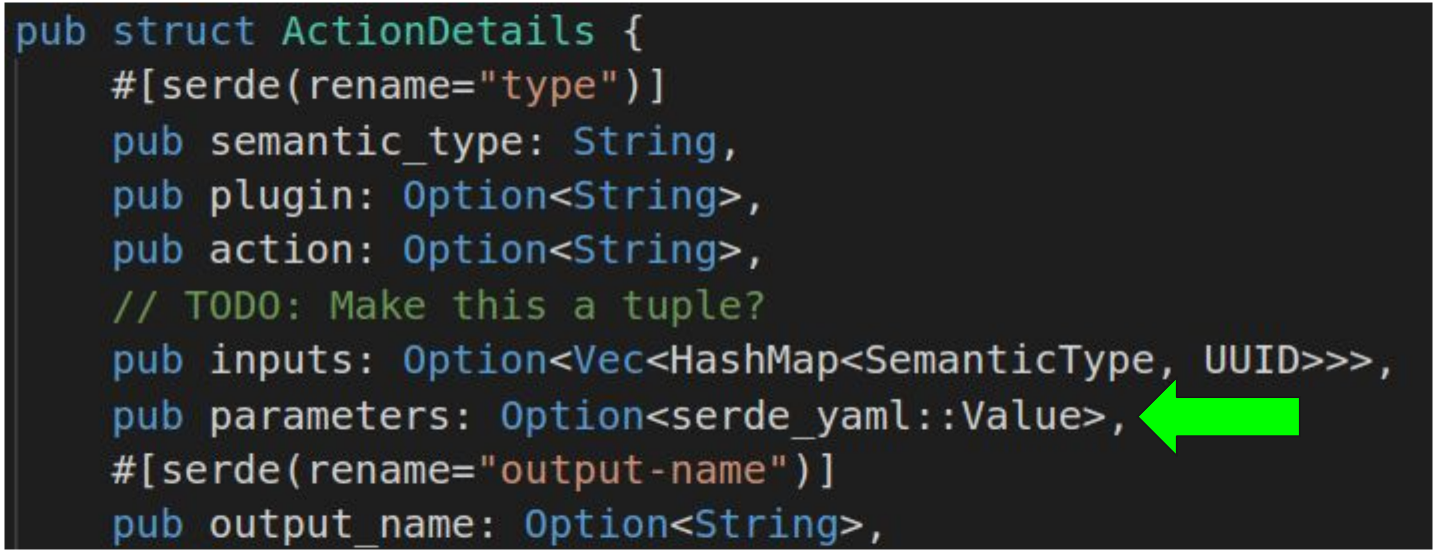
```
impl ProvNode {
    pub fn new(filenames: Vec<String>, rel_files: &ArchiveContents)
            -> Result<ProvNode, serde_yaml::Error> {
        let mut metadata: Option<ActionMetadata> = None;
        let mut action: Option<Action> = None;
        let mut citations = None;
        let key_err = "Key Error in ProvNode::new(); Filepath does not exist in ArchiveContents";
        for i in filenames {
            let content = rel_files.file_contents.get(&i).ok_or_else(|| {key_err});
            if i.contains("metadata.yaml") {
                metadata = serde_yaml::from_str(content.unwrap())?;
            } else if i.contains("action.yaml") {
                action = serde_yaml::from_str(content.unwrap())?;
            }
            else if i.contains("citations.bib") {
                citations = Some(String::from(content.unwrap()));
            }
        }
```

Lesson 2: Rust has neat libraries!



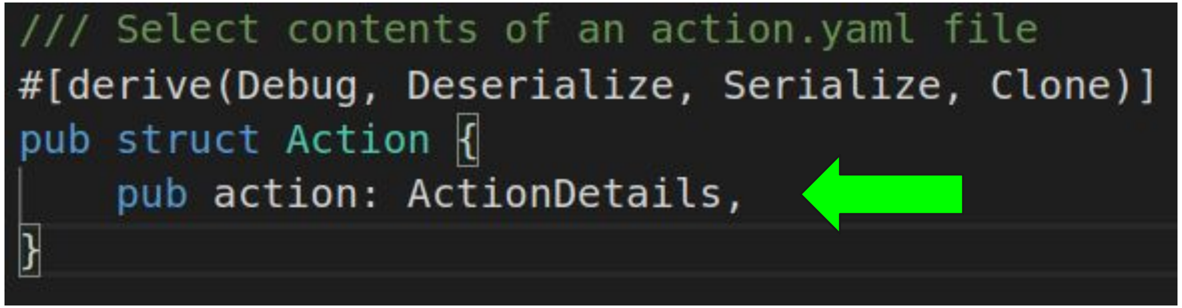
```rust
pub struct ActionDetails {
    #[serde(rename="type")]
    pub semantic_type: String,
    pub plugin: Option<String>,
    pub action: Option<String>,
    // TODO: Make this a tuple?
    pub inputs: Option<Vec<HashMap<SemanticType, UUID>>>,
    pub parameters: Option<serde_yaml::Value>,
    #[serde(rename="output-name")]
    pub output_name: Option<String>,
```

Lesson 2: Rust has neat libraries!

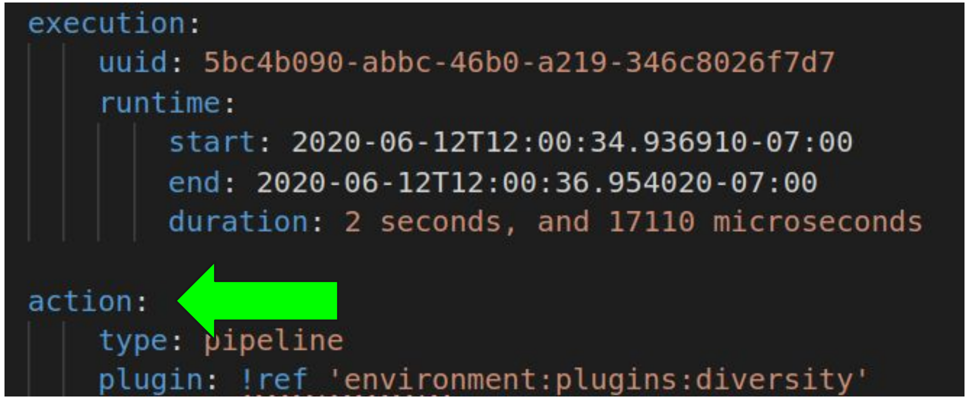


```
pub struct ActionDetails {
    #[serde(rename="type")]
    pub semantic_type: String,
    pub plugin: Option<String>,
    pub action: Option<String>,
    // TODO: Make this a tuple?
    pub inputs: Option<Vec<HashMap<SemanticType, UUID>>>,
    pub parameters: Option<serde_yaml::Value>,
    #[serde(rename="output-name")]
    pub output_name: Option<String>,
```

Lesson 2: Rust has neat libraries!

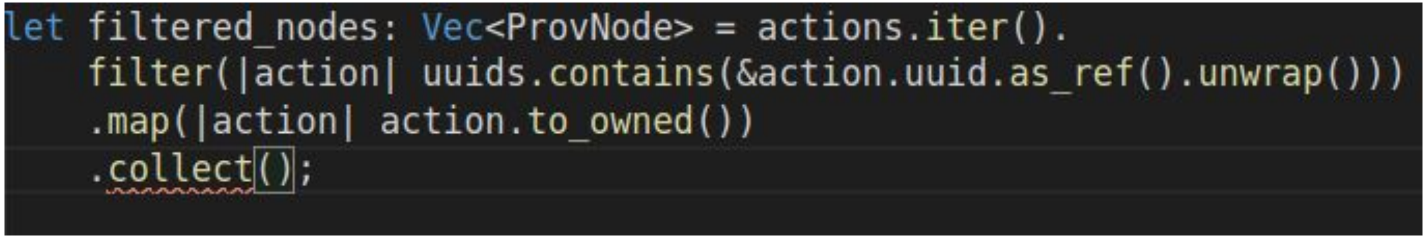


```
/// Select contents of an action.yaml file
#[derive(Debug, Deserialize, Serialize, Clone)]
pub struct Action {
    pub action: ActionDetails,
}
```



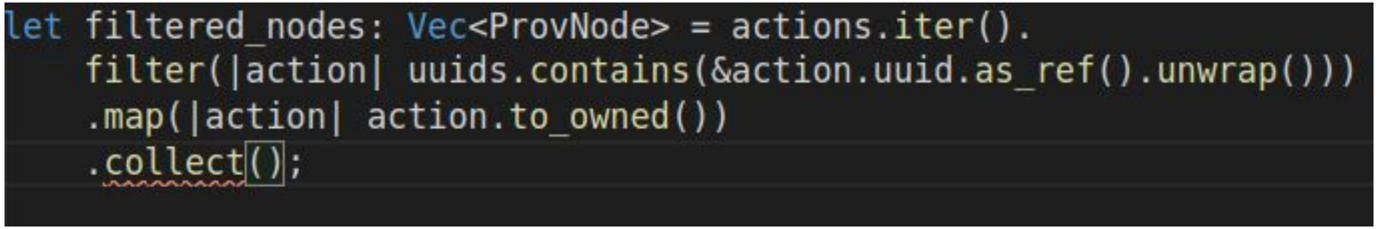
```
execution:
    uuid: 5bc4b090-abbc-46b0-a219-346c8026f7d7
    runtime:
        start: 2020-06-12T12:00:34.936910-07:00
        end: 2020-06-12T12:00:36.954020-07:00
        duration: 2 seconds, and 17110 microseconds

action:
    type: pipeline
    plugin: !ref 'environment:plugins:diversity'
```

Lesson 3: The Rust Compiler Tries...

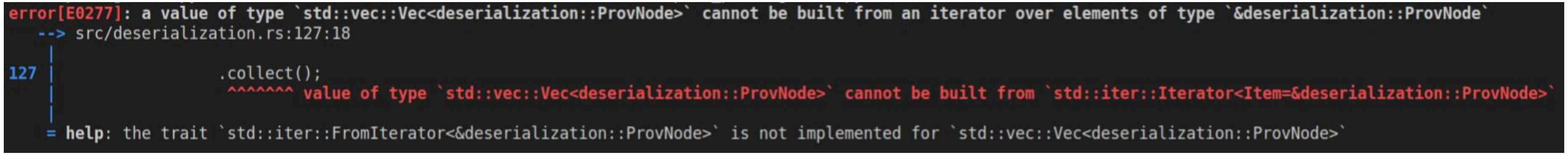


```
let filtered_nodes: Vec<ProvNode> = actions.iter().
    filter(|action| uuids.contains(&action.uuid.as_ref().unwrap()))
    .map(|action| action.to_owned())
    .collect();
```

Lesson 3: The Rust Compiler Misses Sometimes



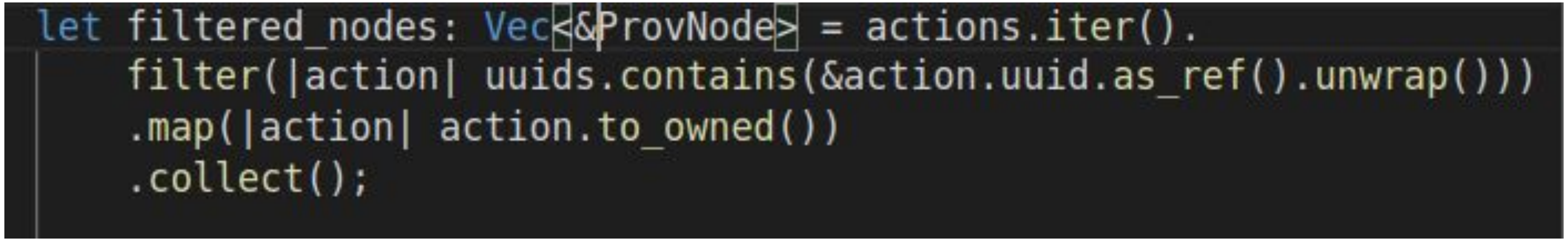
```
let filtered_nodes: Vec<ProvNode> = actions.iter().
    filter(|action| uuids.contains(&action.uuid.as_ref().unwrap()))
    .map(|action| action.to_owned())
    .collect();
```



```
error[E0277]: a value of type `std::vec::Vec<deserialization::ProvNode>` cannot be built from an iterator over elements of type `&deserialization::ProvNode`
   --> src/deserialization.rs:127:18
    |
127 |                 .collect();
    |                  ^^^^^^^ value of type `std::vec::Vec<deserialization::ProvNode>` cannot be built from `std::iter::Iterator<Item=&deserialization::ProvNode>`
    |
    = help: the trait `std::iter::FromIterator<&deserialization::ProvNode>` is not implemented for `std::vec::Vec<deserialization::ProvNode>`
```

(Vec<ProvNode> can’t be made from Iterator<&ProvNode>)

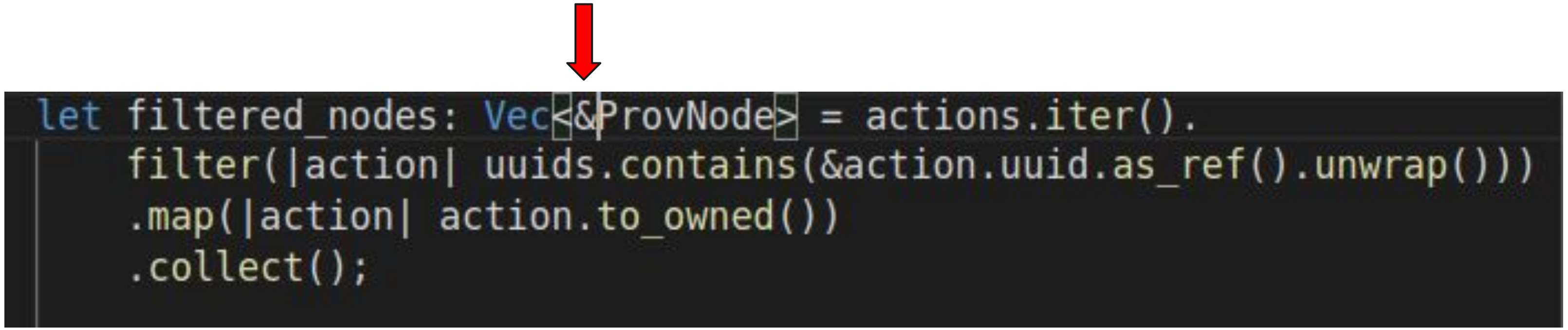
Lesson 3: The Rust Compiler Misses Sometimes



```
let filtered_nodes: Vec<&ProvNode> = actions.iter().
    filter(|action| uuids.contains(&action.uuid.as_ref().unwrap()))
    .map(|action| action.to_owned())
    .collect();
```

Oooh. I can fix that!

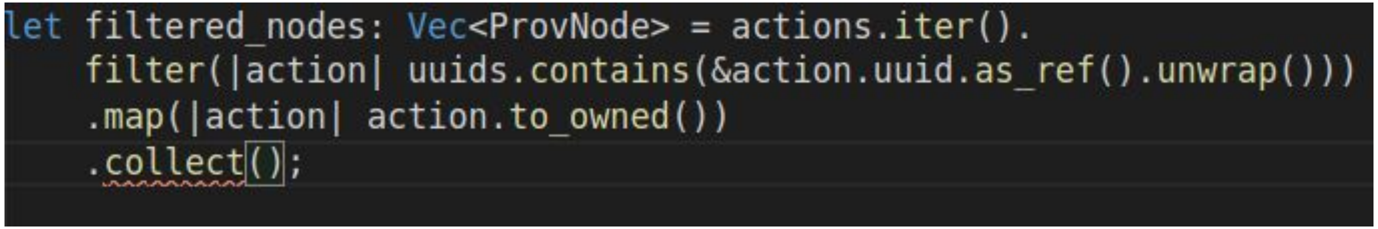
Lesson 3: The Rust Compiler Misses Sometimes



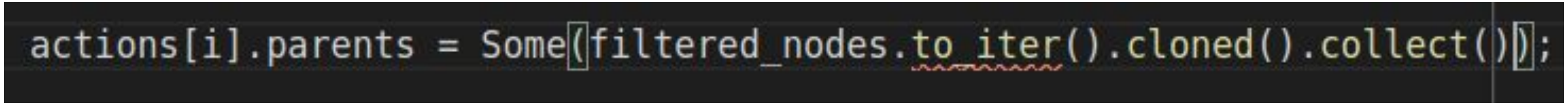
```
let filtered_nodes: Vec<&ProvNode> = actions.iter().
    filter(|action| uuids.contains(&action.uuid.as_ref().unwrap()))
    .map(|action| action.to_owned())
    .collect();
```

Oooh. I can fix that!

Lesson 3: The Rust Compiler Misses Sometimes



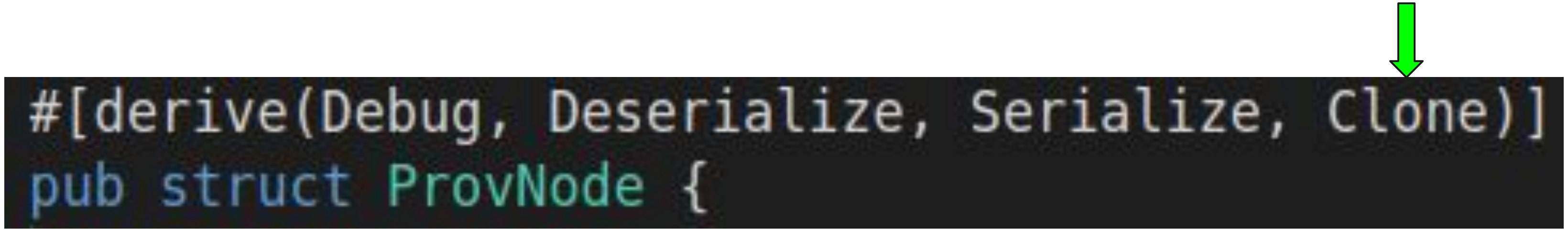
```
let filtered_nodes: Vec<ProvNode> = actions.iter().
    filter(|action| uuids.contains(&action.uuid.as_ref().unwrap()))
    .map(|action| action.to_owned())
    .collect();
```



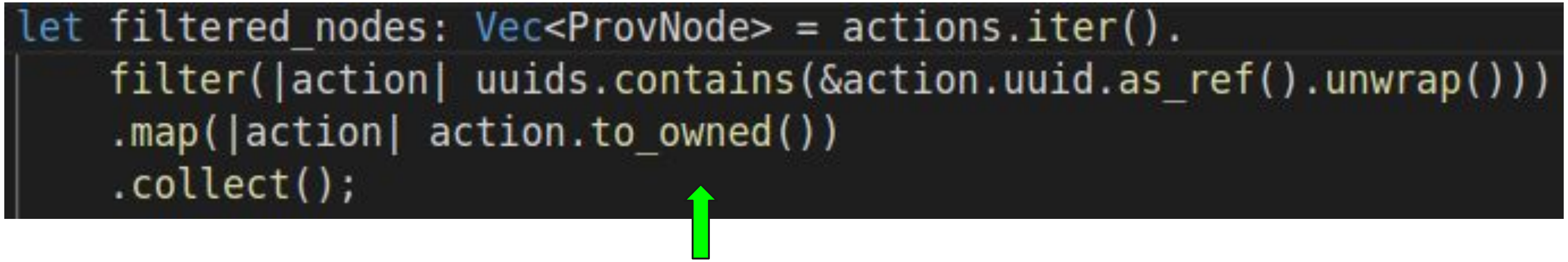
```
actions[i].parents = Some(filtered_nodes.to_iter().cloned().collect());
```

Even the duct tape isn’t helping.

Lesson 3: The Rust Compiler Misses Sometimes

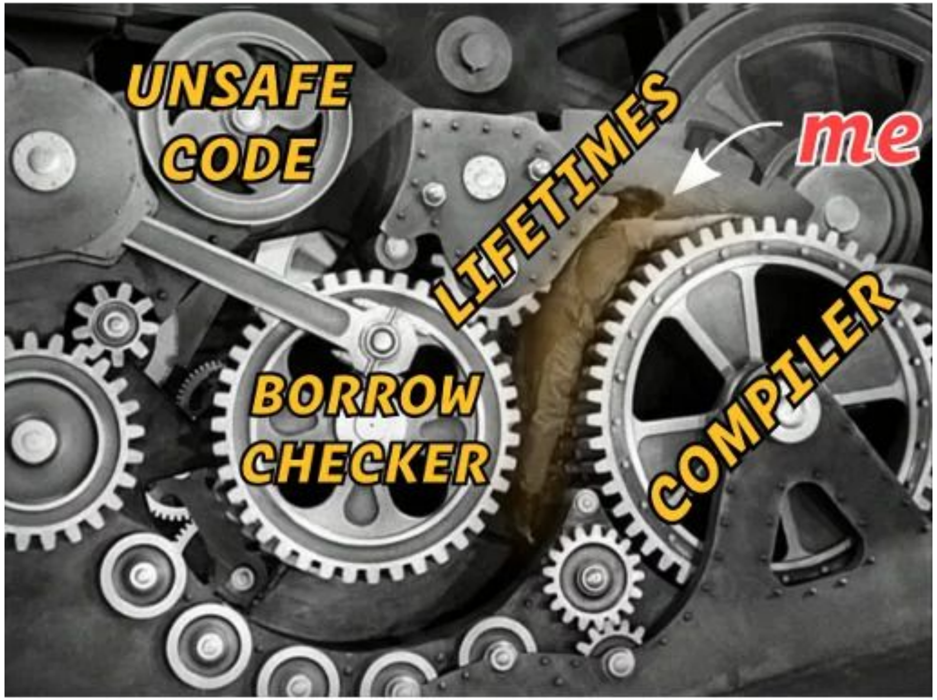


```rust
#[derive(Debug, Deserialize, Serialize, Clone)]
pub struct ProvNode {
```



```rust
let filtered_nodes: Vec<ProvNode> = actions.iter().
    filter(|action| uuids.contains(&action.uuid.as_ref().unwrap()))
    .map(|action| action.to_owned())
    .collect();
```

Wrapup: what we've achieved



What we’ve achieved

- Zip Archive Reading
- Data Modeling
- YAML deserialization
- Basic provenance tree building
- Modular structure
- Basic error reporting

To-do list:

- `build_tree()` buggy
- Data serialization
- Testing
- Robustness/Style

To-do list:

- `build_tree()` buggy
- Data serialization
- Testing
- Robustness/Style

Features/Future Work

- Runnable without a QIIME 2 install
- Compiling to WASM could let us do browser things
- UUID-based provenance tree diffs
- Parameter-based provenance tree diffs, etc
- Visualization of “nested” provenance data
- Full-analysis citations
- Full-analysis executable generation

Thanks to the rust community and all of you!

Image credits to the rust community via https://github.com/rochacbruno/rust_memes