Ruby => Go -> Rust ::<> - Embracing modern paradigms

@sudhindraRao
Development Manager @JFrog

Sudhindra Rao

- 20 years in Software
- Process Control Engineer for a few years
- Rubyist since 2006
- Golang Skeptic -> Embracer
- **Learning Rust**
- Developer of class from time to time
- **Current Status: Dev Manager @JFrog**

(Uses Spaces not Tabs)

Follow me: @sudhindraRao

sudhindraRao.com

Key concepts in Programming

- Data Encapsulation
- Separation of Concerns
- Loose coupling
- Identifying patterns
- Design for readability
- Test First - Test Driven Development
- Architecture that is easy to change

Programming Patterns

- Convention over configuration
- Testability of Code
- Error handling and communicating with the user
- Patterns from the 12 factor app
- Naming conventions for variables, methods, functions, objects
- Code formatting patterns
- Unit testing, Integration testing, Acceptance testing

Programming in Golang

- Language designed to address specific problems
- Universal Formatting with gofmt
- Stdlib rules!
- Clean syntax
- Functional Programming
- No side-effects
- Strong primitives that are easy to learn

Programming in Rust

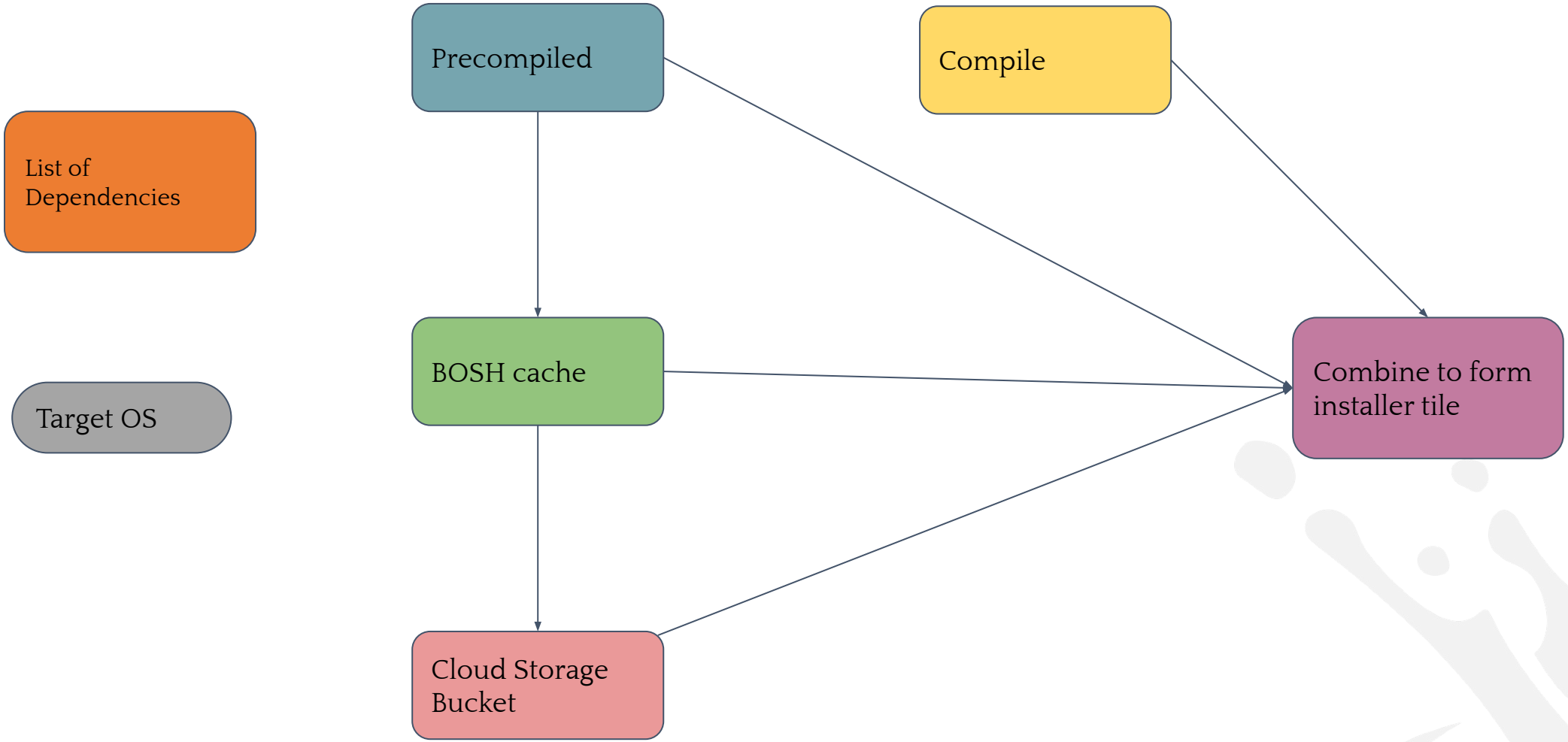
- Designed to solve specific problems - **Systems programming for security**
- Universal Formatting with rustfmt - **YAY!**
- Stdlib rules! - **Yes and Maybe?**
- Clean syntax - **Familiar syntax**
- Functional Programming - **YAY!!**
- No side-effects - **YAY!! - (borrow_checker) - No Garbage Collection!**
- Strong primitives that are easy to learn - **Not so easy to learn!**

My experiments with Go

A Kubernetes platform

- A complete Kubernetes Platform
- Support for multiple operating systems (also Windows)
- Support for many clouds
- Day 2 operations support

Look - Find - Package



Concurrency the old way

```
for i := 0; i < workers; i++ {
  go func(deps tile.Dependency, args ProgArgs) {
    for {
      select {
      case release := <-releases:
        err := app.processRelease(release, deps, args)
        result := Result{
          release: release,
          err:     err,
        }
        results <- result
      case <-stop:
        return
      }
    }
  }(deps, args)
}
```

```
func (app App) processRelease(...) error {
    app.downloader.Download(...)
...
```

```
func (app App) Download(...) error {
    app.downloader.Compile(...)
...
```

```
func (app App) Compile(...) error {
    app.downloader.Publish(...)
...
```

```
func (app App) Publish(...) error {
    app.downloader.CalculateSha(...)
...
```

CD.FOUNDATION

Termination Condition

```
stopWorkers := func() {
  for i := 0; i < workers; i++ {
    stop <- true
  }
  close(stop)
}
```

```
for result := range results {
  if result.err != nil {
    errorResults = append(errorResults, result)
  }
  remaining--

  if remaining == 0 {
    stopWorkers()
    close(results)
    close(releases)
  }
}
```

Old code - concurrency issues

- Had Goroutines
- Had a couple of channels for signaling state
- Had sync mechanisms like waitGroups
- Started breaking as soon as we introduced another OS

Goroutines

- *Do not communicate by sharing memory; instead, share memory by communicating.*
- Concurrently executing
- Similar to threads - But cheaper
- Channel is a communication mechanism between goroutines

- Pipelines to chain concurrency
- No 'external' bookkeeping for the system

What about WaitGroups and Synchronization

Did you defer cleanup

Don't trust, your weakness with threads

Synchronization with scattered booleans is a bad idea

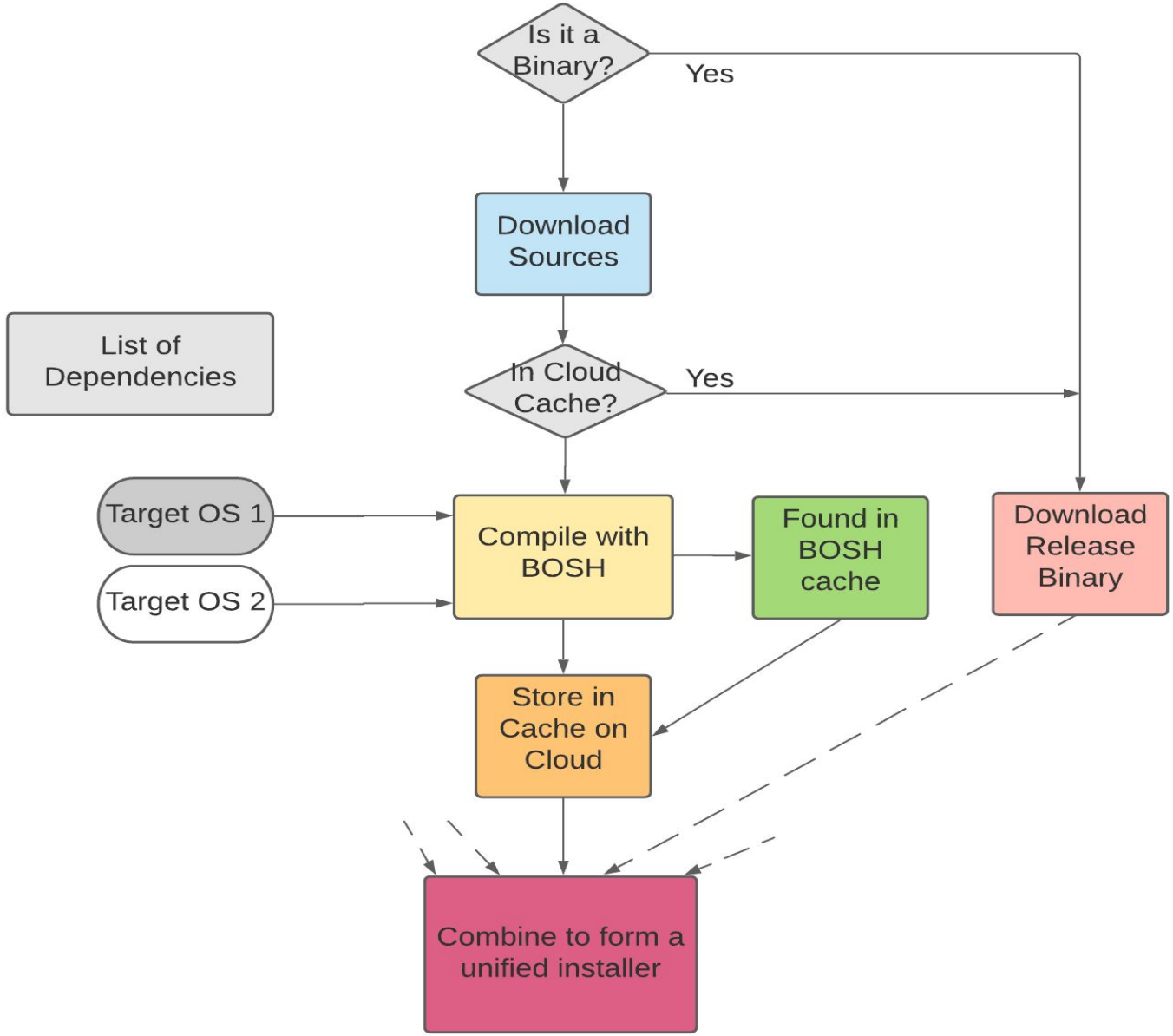
Don't forget to call wg.Done()

Identify concurrency

- Network Latency
- Cloud Latency
- Compilation Latency
- Resource Latency
- OS specific Latency*

Untangle

The Algorithm



Building the Pipeline

```
numberOfResultsExpected := len(tileDependencies.Releases)
releasesToDownload := make(chan CandidateRelease, numberOfResultsExpected)
releasesToCompile := make(chan CandidateRelease, numberOfResultsExpected)
releasesToPublish := make(chan CandidateRelease, numberOfResultsExpected)
results := make(chan CandidateRelease, numberOfResultsExpected)

for i := 0; i < a.maxConcurrentDownloaders; i++ {
  go a.downloadWorker.Create(releasesToDownload, releasesToCompile, results)
}

for i := 0; i < a.maxConcurrentCompilers; i++ {
  go a.compileWorker.Create(releasesToCompile, releasesToPublish, results)
}

for i := 0; i < a.maxConcurrentPublishers; i++ {
  go a.publishWorker.Create(releasesToPublish, results)
}
```

Termination Condition

```
for candidate := range results {
  if candidate.Result.Err != nil {
    allErrors = append(allErrors, candidate.Result.Err.Error())
  }
  numberOfResultProcessed++
  if numberOfResultProcessed == numberOfResultsExpected {
    break
  }
}

close(releasesToCompile)
close(releasesToPublish)
close(results)
```

Objects through the pipeline

```
type CandidateRelease struct {
 Release tile.Release
 Result  Result
}
```

```
type Result struct {
 PathToDownloadedSourceTarball   string
 PathsToCompiledTarball          map[string]string
 PathToCompiledTarballsToPublish []string
 Err                             error
}
```

```
and a few more objects...
```

CD.FOUNDATION

Success!

- Build time 6 hours -> 2 hours
- Error traceability for each binary
- Fast failures means fast feedback

Don’t forget the basics

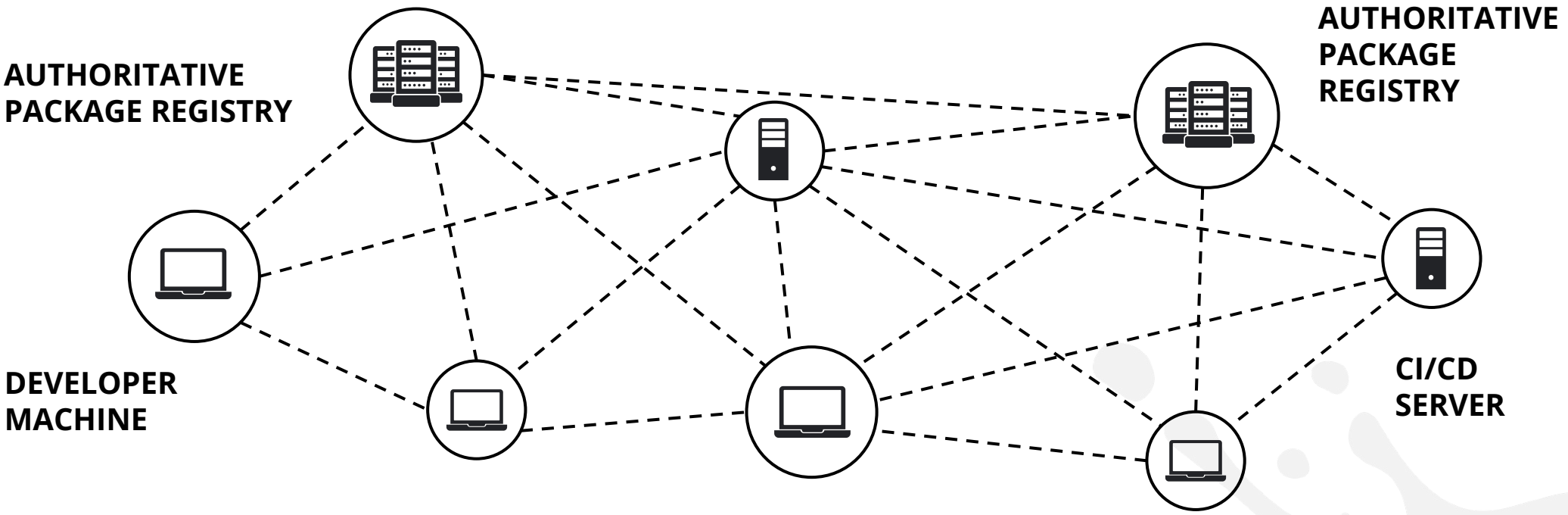
- Test Driven Development(TDD)
- Malleable Code
- Separation of Concerns - Identify responsibility
- Encapsulation

- Refactor
- Go back to basics
- Prioritize Tech Debt
- Optimize at the right time
- Ask for Support

~~My~~ Experiments with Rust

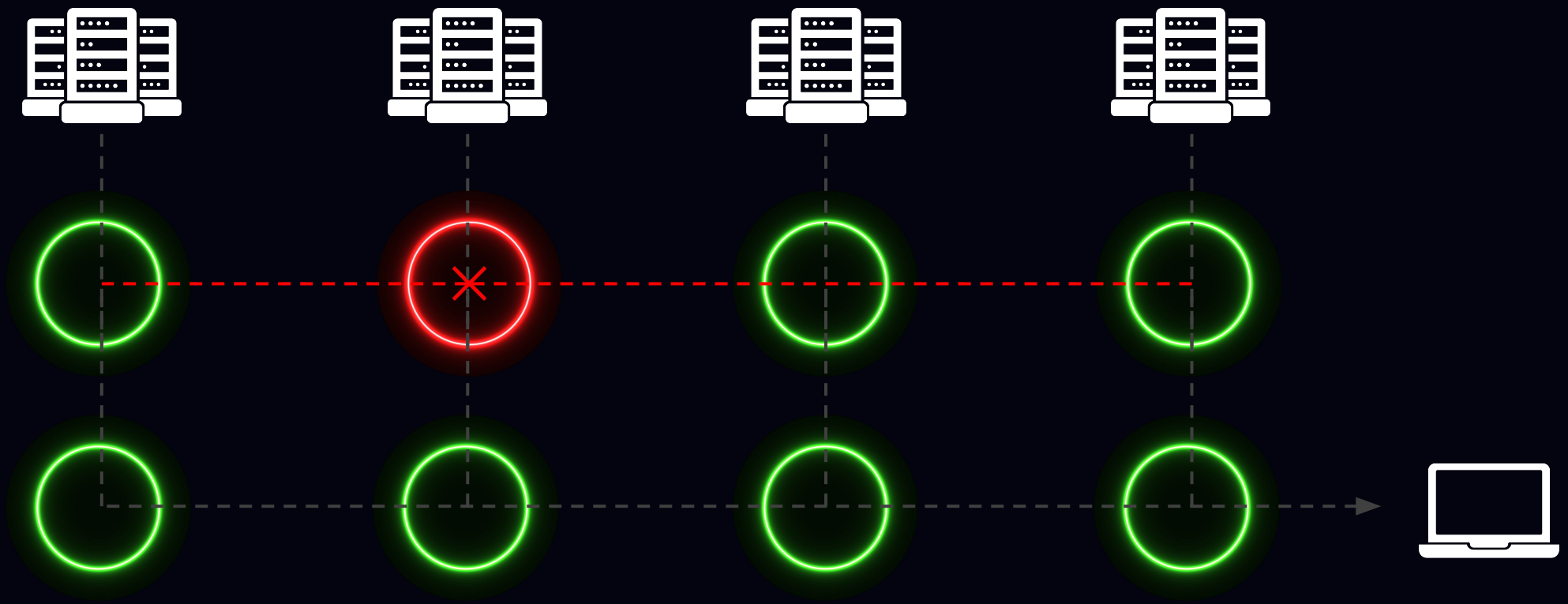
Our

Pyrsia Binary Distribution Network

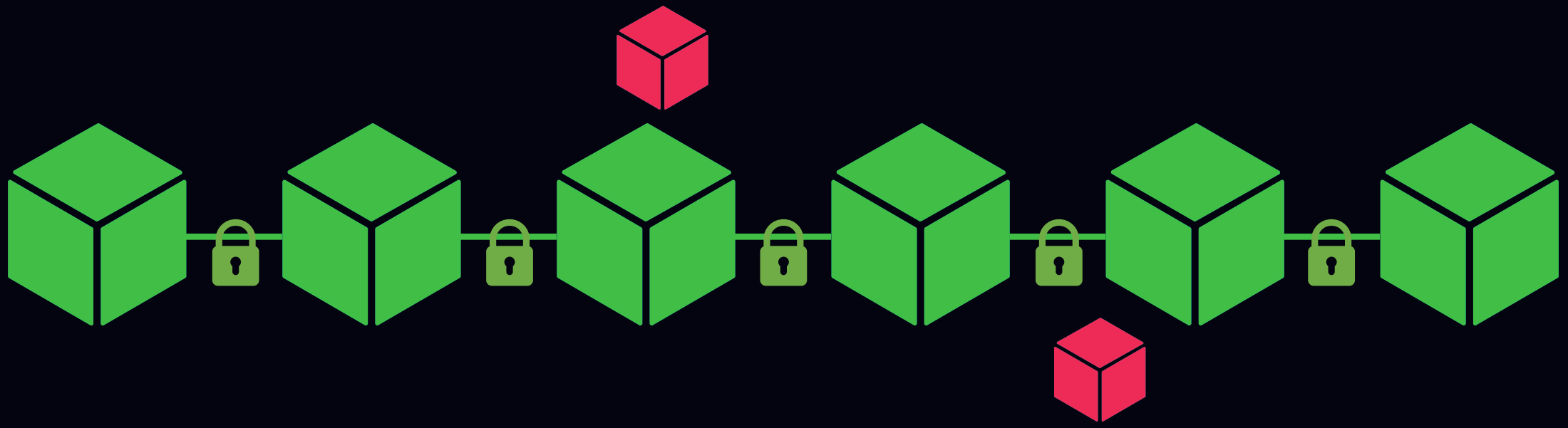


CD.FOUNDATION

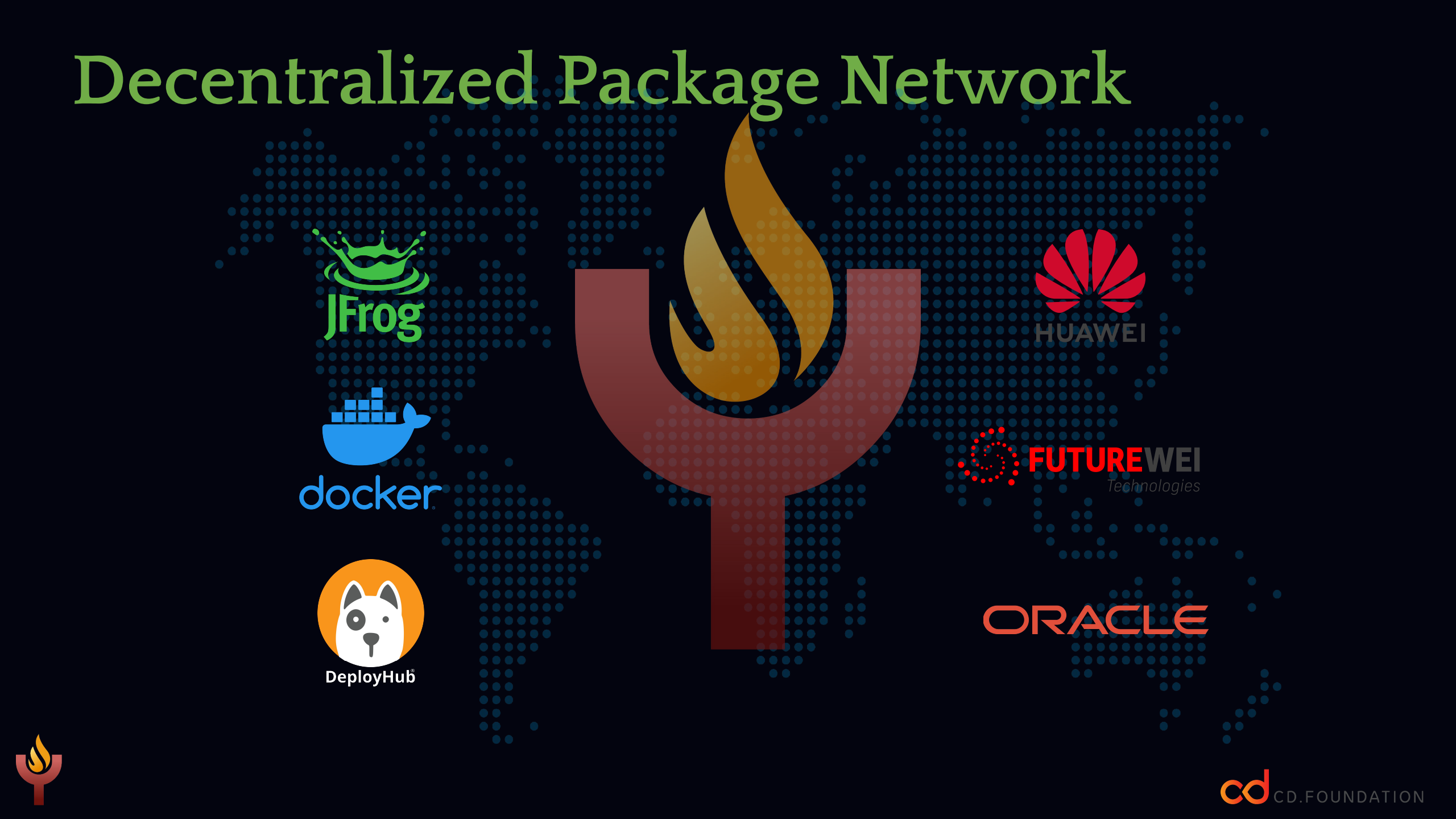
Multi-node verification of Builds



Immutable Transparency Ledger



Decentralized Package Network



Challenges in Building Pyrsia

- Peer to Peer distribution model - libp2p in Rust
- Immutable ledger with novel consensus algorithm - AlephBFT
- Managing Artifacts (not files) for distribution and storage
- MultiNode Build Processing
- Event driven mechanisms to keep the metadata updated

cohost.org/jubilee @workingjubilee · Nov 6

Replying to @Subject_CsK

tbf, often the error messages are dogshit and are cryptic breadcrumbs at best for a lot of language intepreters/compilers, or are genuinely irrelevant warnings, so people coming to Rust with preconditioning to ignore the errors is understandable to me.

1 24

def __conrad__(self, kyne): return kyne + self

@Subject_CsK

Replying to @workingjubilee

Even the linter is incredible. I was able to port a big chunk of a python lib to rust (for like 40x performance gains) without even really understanding what borrowing is

4:33 PM · Nov 6, 2022 · Twitter Web App

6 Retweets **1** Quote Tweet **27** Likes

Patterns Observed in Rust

- Managing Concurrency
- Error Handling
- Borrow Checker

Async programming in Rust

- Rust book Chapter 16 - Fearless Concurrency
- thread::spawn
- std::sync::mpsc - Channels
- Mutex::New
- Rc<T> - Reference Counting
- Arc<T> - Atomic Reference Counting

Adding Tokio for Concurrency

```
56    async fn main() {
57        pretty_env_logger::init();
58
59  -     // Initiate the document store with it's first document
60  -     let index_one = "index_one";
61  -     let field1 = "most_significant_field";
62  -     // The actual first index is not necessary here, it is preserved in the
      documentStore
63  -     IndexSpec::new(index_one, vec![field1]);
64  -
65  -     let matches: ArgMatches = App::new("Pyrsia Node")
66  -         .version("0.1.0")
67  -         .author(clap::crate_authors!(", "))
68  -         .about("Application to connect to and participate in the Pyrsia network")
69  -         .arg(
70  -             Arg::new("host")
71  -                 .short('H')
72  -                 .long("host")
73  -                 .value_name("HOST")
74  -                 .default_value(DEFAULT_HOST)
75  -                 .takes_value(true)
76  -                 .required(false)
77  -                 .multiple_occurrences(false)
78  -                 .help("Sets the host address to bind to for the Docker API"),
79  -         )
80  -         .arg(
81  -             Arg::new("port")
82  -                 .short('p')
83  -                 .long("port")
84  -                 .value_name("PORT")
85  -                 .default_value(DEFAULT_PORT)
86  -                 .takes_value(true)
87  -                 .required(false)
88  -                 .multiple_occurrences(false)
89  -                 .help("Sets the port to listen to for the Docker API"),
90  -         )
91  -         .arg(
92  -             Arg::new("peer")
93  -                 //.short("p")
```

```
33    async fn main() {
34        pretty_env_logger::init();
35
36  +     let args = args::parser::PyrsiaNodeArgs::parse();
37  +
38  +     let (mut p2p_client, mut p2p_events, event_loop) = p2p::new().await.unwrap();
39  +
40  +     tokio::spawn(event_loop.run());
41  +
42  +     let final_peer_id = match args.peer {
43  +         Some(to_dial) => dial_other_peer(p2p_client.clone(), to_dial).await,
44  +         None => None,
45  +     };
```

Removing Mutexes

```
240
241     fn setup_artifact_service(
242         artifact_path: &Path,
243 -       blockchain_service: BlockchainService,
244         build_event_client: BuildEventClient,
245         p2p_client: Client,
246 - ) -> Result<Arc<Mutex<ArtifactService>>> {
247 -       let local_keypair =
248 -
    keypair_util::load_or_generate_ed25519(PathBuf::from(KEYPAIR_FILENAME.as_st
    r()));
249 -
250         let artifact_service = ArtifactService::new(
251             artifact_path,
252 -           local_keypair,
253 +           blockchain_service,
254             build_event_client,
255             p2p_client,
256         )?;
257
258 -       Ok(Arc::new(Mutex::new(artifact_service)))
259     }
```

```
262
263     fn setup_artifact_service(
264         artifact_path: &Path,
265 +       transparency_log_service: TransparencyLogService,
266         build_event_client: BuildEventClient,
267         p2p_client: Client,
268 + ) -> Result<ArtifactService> {
269         let artifact_service = ArtifactService::new(
270             artifact_path,
271 +           transparency_log_service,
272             build_event_client,
273             p2p_client,
274         )?;
275
276 +       Ok(artifact_service)
277     }
```


/// based on the provided artifact id.
pub async fn handle_request_artifact(
- artifact_service: Arc<Mutex<ArtifactService>>,
artifact_id: &str,
channel: ResponseChannel<ArtifactResponse>,
) -> anyhow::Result<()> {
debug!("Handling request artifact: {:?}", artifact_id);
- let mut artifact_service = artifact_service.lock().await;
let content =
artifact_service.get_artifact_locally(artifact_id).await?;
artifact_service
@@ -82,7 +82,8 @@ pub async fn handle_request_idle_metric(
}
/// based on the provided artifac
pub async fn handle_request_artif
+ mut artifact_service: Artifac
artifact_id: &str,
channel: ResponseChannel<Arti
) -> anyhow::Result<()> {
debug!("Handling request arti
let content =
artifact_service.get_artifact_loc
artifact_service
}

Using async trait

```
+ use async_trait::async_trait;
+ use futures::prelude::*;
+ use libp2p::core::upgrade::{read_length_prefixed, write_length_prefixed,
  ProtocolName};
+ use libp2p::request_response::RequestResponseCodec;
+ use std::io;
+
```

```
pub async fn handle_request_artifact(
    mut p2p_client: p2p::Client,
    hash: &str,
    channel: ResponseChannel<p2p::ArtifactResponse>,
) {
    let decoded_hash = hex::decode(&hash.get(7..).unwrap()).unwrap();
    match get_artifact(&decoded_hash, HashAlgorithm::SHA256) {
        Ok(content) => p2p_client.respond_artifact(content, channel).await,
        Err(e) => info!(
            "This node does not provide artifact {}. Error: {:?}",
            hash, e
        ),
    }
}
```

Elegant Error Handling

```
cmd = exec.Command("git", "-C", cloneTarget, "checkout", s.release.Tag)
output, err = cmd.CombinedOutput()
if err != nil {
    s.log.Println(string(output))      Ben Christel, 3 years ago • Consolidates logging cod
    return "", err
}
```

```
fn does_things() -> Result<u32, IoError> {
    let res1 = try!(canFail())
    let res2 = try!(canFail())
    return Ok(res1 + res2);
}
```

Error Handling in Rust

- Error handling in Rust can be clumsy if you can't use the question-mark operator. **To achieve happiness**, we need to return a `Result` which can accept any error. All errors implement the trait `std::error::Error`
- simple_error
- error_chain
- anyhow
- thiserror

Anyhow for Error Handling

```rust
const JSON_BODY: &'static str = r#"      Elliott Frisch, 2 months ago • Multiline raw string literals.
  { "version": "0.8" }
"#;

▶ Run | Debug
fn main() {
    let json_res: Result<JsonValue, Error> = json::parse(source: JSON_BODY);
    let version_result: Result<f32, String> = match json_res {
        Ok(json_value: JsonValue) => parse_version(json_value),
        Err(err: Error) => Err(format!("Parse error {err:?}")),
    };
    match version_result {
        Ok(vers: f32) => {
            println!("parsed version: {vers}")
        }
        Err(err: String) => {
            println!("Could not parse version {err:?}")
        }
    }
}

fn parse_version(json_value: json::JsonValue) -> Result<f32, String> {
    match &json_value["version"] {
        json::JsonValue::String(str: &String) => {
            // This is a turbofish.
            return match str.parse::<f32>() {
                Ok(v: f32) => Ok(v),
                Err(err: ParseFloatError) => Err(format!("Parse error {err:?}")),
            };
        }
        json::JsonValue::Number(numb: &Number) => {
            // Turbofish.
            return match numb.to_string().parse::<f32>() {
                Ok(v: f32) => Ok(v),
                Err(err: ParseFloatError) => Err(format!("Parse error {err:?}")),
            }
        }
        json::JsonValue::Short(numb: &Short) => {
            // Turbofish.
            return match numb.to_string().parse::<f32>() {
                Ok(v: f32) => Ok(v),
                Err(err: ParseFloatError) => Err(format!("Parse error {err:?}")),
```

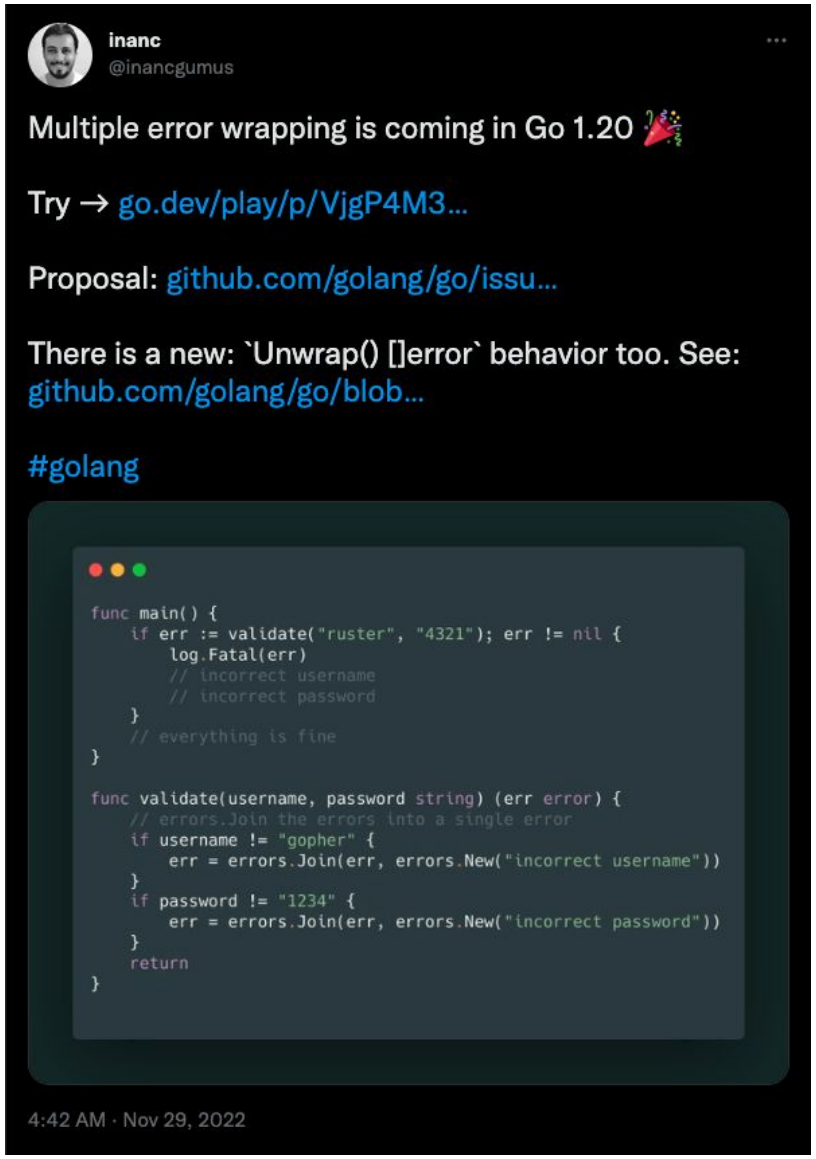
```rust
const JSON_BODY: &'static str = r#"      Elliott Frisch, 2 months ago • Multiline raw
  { "version": "" }
"#;
// https://github.com/dtolnay/anyhow
▶ Run | Debug
fn main() {
    let json_res: Result<JsonValue, Error> = json::parse(source: JSON_BODY);
    let version_result: Result<f32, Error> = match json_res {
        Ok(json_value: JsonValue) => parse_version(json_value),
        Err(err: Error) => Err(anyhow::anyhow!(err)),
    };
    match version_result {
        Ok(vers: f32) => {
            println!("parsed version: {vers}")
        }
        Err(err: Error) => {
            println!("Could not parse version {err:?}")
        }
    }
}

fn parse_version(json_value: json::JsonValue) -> anyhow::Result<f32> {
    Ok(json_value["version"].to_string().parse::<f32>()?)
}
```

[RustNYC talk by Elliott Frisch]

When writing a library don't use Anyhow, but when **using** a library highly recommended to use Anyhow.

Seems like a good idea



Embracing the Borrow Checker

- Expect a steep learning curve
- Initial efforts are going to be frustrating
- Encourage knowledge sharing and Pair programming
- After 1 year we still have areas that are challenging - Borrow Checker and Async programming

Pyrsia onboarding documentation

Key takeaways - 1 year of Rust

- Expect a steep learning curve
- Study with the Community
- Embrace the Evolution in Rust

Remember and apply the fundamentals

THANK YOU!

@sudhindraRao
sudhindraRao.com

@PyrsiaOSS
pyrsia.io

CD.FOUNDATION

References

Go References

- The Go Programming Language
- Effective Go
- Golang Playground
- https://jfrog.com/shownote/using-golang-concurrency-asynchronous-processing-06-2021/

Rust References

- The Rust Programming Language Book
- Rust for Rustaceans Book
- https://doc.rust-lang.org/book/ch09-02-recoverable-errors-with-result.html
- https://www.quora.com/What-is-anyhow-error-in-Rust
- https://stevedonovan.github.io/rust-gentle-intro/
- https://www.reddit.com/r/rust/comments/djzd5t/which_asyncconcurrency_crate_to_choose_from/
- https://news.ycombinator.com/item?id=28510257
- https://www.libhunt.com/r/crossbeam
- https://github.com/pyrsia/pyrsia/commit/11e1d1399951ffa2953e01b8df4c1022f3471a80
- https://github.com/pyrsia/pyrsia/pull/1062/files

Other References

- https://www.youtube.com/watch?v=JAD2s2nMr_0&ab_channel=BoulderRuby