Rust vs Go

Topology Engineering Team

Description
We need to build live-object-sdk and live-object-vm. We decided to use Rust for live-object-sdk but haven't decided which language to use for live-object-vm.

Candidates are Rust and Go, and detailed pros and cons can be found here.

It's noteworthy that all of these trade-offs originated from the fact that gram is written in Go while Wasmer is in Rust. Hence, our next question is this: is gram better to migrate to Rust?

More context behind the initial decision
0. Why was Go selected?
- Both Go and Protocol Buffers are developed by Google so they go really well. Protocol Buffers for Rust is not officially supported yet (it has been a year since they started to work on this) but there are community built libraries such as Prost.
- go-libp2p is slightly more implemented than rust-libp2p, though both of them are conceived as supporting all we need.
- Jihoon was under the pressure of not producing any code even after we found what to build two months ago. Rust vs Go seemed a toss up and could be changed later.

1. What has changed?
- Protocol Buffers can be replaced with JSON. It was meant to be used as IDL (Interface Definition Language) for Live Object but JSON will be a more developer-friendly choice. All blockchains use JSON when describing smart contract's interface.
- Team learned how others implement their WASM based VM. Among them, Cosmos is the closest example to us. Cosmos app chain is written in Go and CosmWasm is in Rust. But the mixture of Go and Rust inevitably increased complexity in terms of readability and maintainability.

If we're considering migration, now is the right time to decide.

Considerations
0. Goodness for developing major components such as database and WASM based VM. We are already building major components in Go but how easy is it to do it in Rust? Are we just adding dependencies and good to go?

- Database: rocksdb and libmdbx are the options. Both are efficient, embeddable, and have a rust binding. (rocksdb and libmdbx)

Node impl	Database
Near	rocksdb
Polkadot	rocksdb and its own implementation

Solana	its own implementation
zkSync	rocksdb
reth	libmdbx

- JSON-RPC: jsonrpsee and actix are the options.

- gRPC: tonic and Prost are the options.

- WASM: Wasmer is written in Rust, well-maintained and many projects are using it.

1. Goodness for developing a distributed system (what are the important properties? robustness, performance, etc)
Power of Go
- Go's goroutine and channel are convenient.
- Go has GC, taking some of the burden off developers. In Rust, developers have to pass a memory safety check of ownership concept in compilation time.

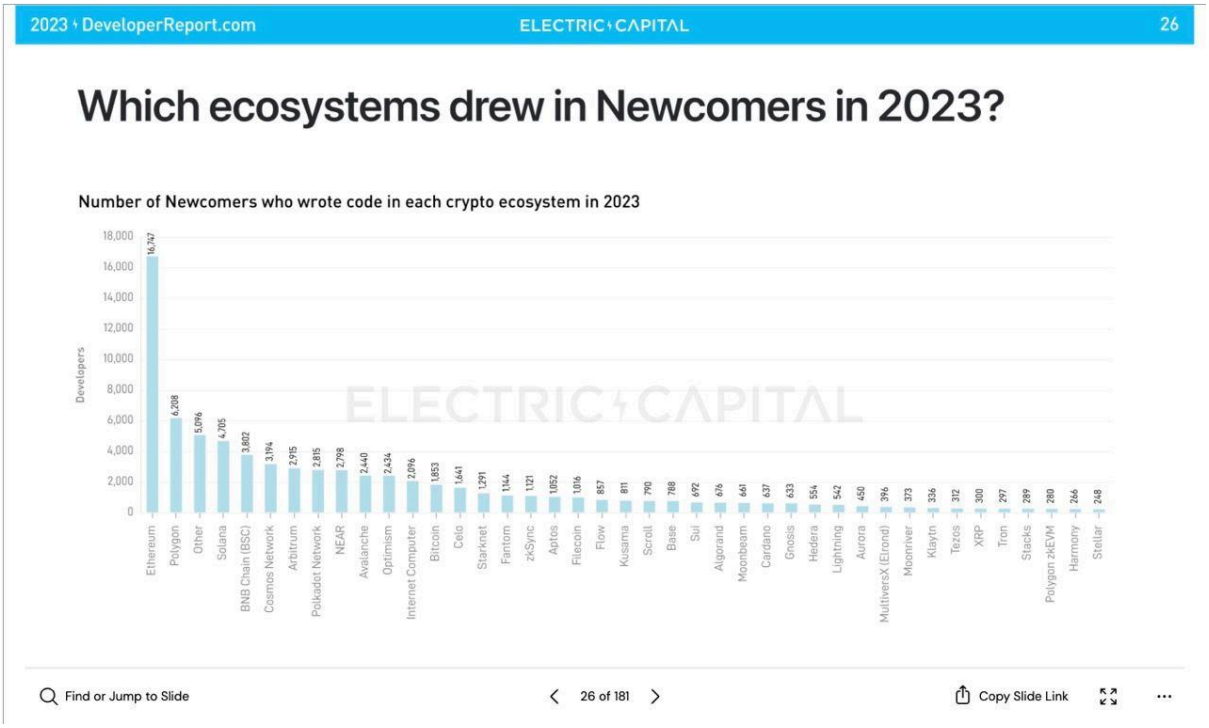
Power of Rust
- Rust's enum, iterator, and async are powerful.
- Rust enforces memory safety through ownership concepts and doesn't have nil. Rust doesn't suffer from memory leaks, dangling pointers, or segmentation faults.
- Rust can handle errors in a more sophisticated way if designed properly.

(Some say Go is more suitable for web service while Rust is more for system programming.)

Case study: Cosmos
- Cosmos chose Go because Rust wasn't an option for them.
- Tendermint started in 2014, Cosmos SDK started in 2017, and async was introduced to Rust in 2018.
- CosmWasm started in 2019 and Wasmer-Go's first stable version was introduced in 2021.

Case study: Near
- Near client is built with Rust.
- Near is also using Wasmer. (They forked it at their needs.)
- Near started in 2018.

2. Developer pool for future recruiting
Choosing a language that is actively used by blockchain core developers would facilitate our future recruiting.



https://www.developerreport.com/developer-report?s=which-ecosystems-drew-in-newcomers

Chain	Node is written in	Smart contract is in	Number of validators	Numbers of newcomers
Ethereum	Go	Solidity	3500	16747
Polygon	Go	Solidity	1000	6208
Solana	Rust	Rust	1770	4705
Cosmos	Go	Rust	180	3194
Near	Rust	Rust / Javascript	215	2798

Blockchain Frameworks	Language	Number of forks
Cosmos SDK	Go	3.3k
Polkadot SDK (ex-Substrate)	Rust	3k
OP Stack	Go	2.7k
Polygon Edge	Go	480

Blockchain nodes written in Go are mostly Geth (or Geth fork) or Tendermint. Nodes in Rust are written by different entities (Solana, Near, Sui, etc).

Smart contracts are written in mostly Solidity followed by Rust.

For spawning application-specific blockchain, Go (Cosmos SDK) and Rust (Polkadot SDK) are adopted similarly. When it comes to L2 on Ethereum, Go (OP Stack) is the most prominent option.

It's safe to say that there are more Go developers in blockchain core development, on which our focus is.

3. Development speed
- Rust requires some learning curve but no significant difference beyond it.

4. Migration cost
- Migration cost is expected around a week.

Conclusion
We'll use Rust for both live-object-sdk and live-object-vm. gram will migrate to Rust.

Rationales are:
- Both Rust and Go have good ecosystems.
- Rust has more expressiveness than Go, making it a better choice for our purpose.
- Zaki (from Cosmos) seconded this saying Rust is a clear winner for blockchain developers.
- There may be more blockchain core developers using Go, but it's hard to say there is a distinct gap.
- It's well-known that a language hurdle is not an issue.
- Migration cost is acceptable.

Appendix
We also put Elixir on the table but it was dropped for three reasons:
- It's an experimental choice.
- We need to write bridge code between Elixir and Wasmer (don't forget that this was the reason why we started this discussion).
- We'll need to write core libraries such as a database adapter by ourselves due to its small ecosystem.