LNP/BP Standards Association

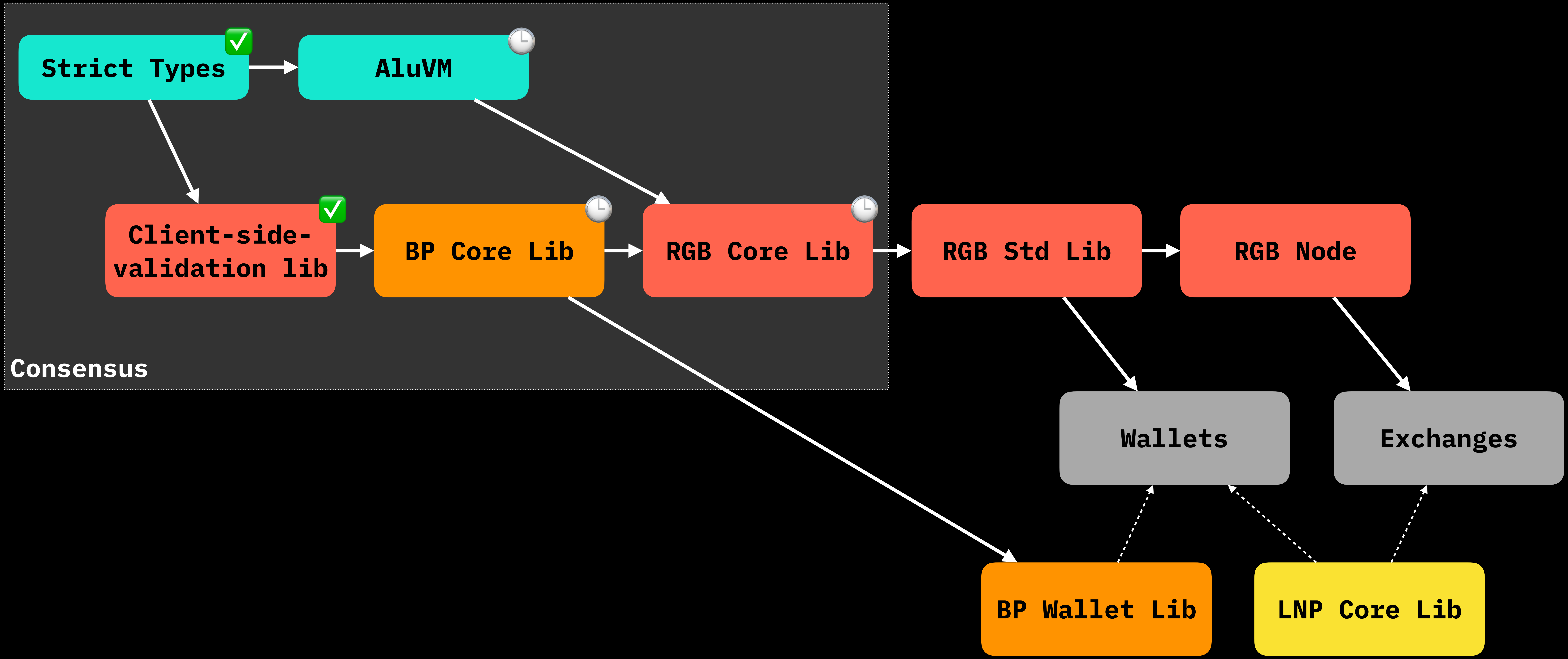
RGB release roadmap updates

Dr Maxim Orlovsky
Chief engineering officer at **LNP/BP Standards Association**,

@dr_orlovsky

FBDE A843 3DDC 1E69 FA90 C35E FFC0 2509 47E5 C6F7

v0.10 release roadmap



strict-types.org

github.com/strict-types

Strict Types

- **Formal verification**
- **New language** (part of Contractum), functional like **Haskell/Idris**, but close to the bare metal like **Rust**
  - autogenerated right from the rust code!
- **Generates** consensus **specs** in automatic fashion!
- Able to **prove** that an implementation follows the spec
- Able to **detect consensus-breaking** changes in new versions and **generate migrations**

Rust

```rust
#[derive(
  StrictType, StrictDumb,
  StrictEncode, StrictDecode
)]
#[strict_type(tags = order)]
enum Assoc {
  #[strict_type(dumb)]
  Index(u8),
  Fields {
    name: TinyString,
    value: u64
  },
  Three
}
```

Strict types

```
data Assoc ::
    index U8 |
    fields (name String,
            value U64) |
    three
```

Compiled

```
----- BEGIN STRICT TYPE LIB -----
Id: 9PAgDBAAAGt41sxDmkmXksGHYbVuz4N2zcFiyPnVqQbv
Checksum: mama-jumbo-sinatra

C1N0cmljdFR5cGVzADMAAA1CdWlsZEZyYWdtZW50DUJ1aWxk
RnJhZ21lbnQEAgAFaWRlbnQABQEBBUlkZW504TuN1tWttWWv
FkVuW4Q4cSncj+UuooL09iZrqs7bbQBBmRpZ2l0cwAFAQEFS
WRlbnThO43W1a21Za8WRW5bhDhxKdyP5T+6igvT2Jmuqzttt
ApEZXBlbmRlbmN5CkRlcGVuZGVuY3kGAwJpZAEJVHlwZUxpY
klk5tZzBPpj+Vr6BbBThcI4gFCKEoTeNrr16VLLPaTbIEbmF
tZQEHTGliTmFtZci7OpO8xnSnrmi2hQxVsmC+zQa9pS3hmnj
OpQxBtLA3ZlcgEGU2VtVmVykAEc2f8Fyx1Ercnhl0Ktvczm6
VI6bq+Rn9oXXvzv34MRW51bVZhcmlhbnRzDEVudW1WYXJpYW
50cwUBAAkBB1ZhcmlhbnQ30qMQ

----- END STRICT TYPE LIB -----
```

Strict types is protobufs for functional programming

Basics

- Generalized abstract data types (**GADT**)
- Each type is **confined**: has a well-defined bounds
  - Like minimal and maximal number of elements in an array/map/set
- Each type is able to describe itself (**reflection**, previously absent in rust)
- Each type is able to describe its **memory layout**
  - Two types may be analyzed on memory layout cross-compatibility
- Each type has a **semantic id**
  - Two types with the same memory layout may be a semantically-different!

Client-side-validation & BP core

- Migration to the new strict type system
  Reduction of codebase: 12’000 lines less!

- New commitment workflow based on strict types
  - Meklization of all array elements
  - Streamlined and automatically enforced commitment rules

- Removal of rust-bitcoin and rust-miniscript dependencies

- Compilation to WASM target

RGB Core Lib v0.10

- Contract **global state**
- Arbitrary **complex & composable data types** as a part of contract owned and global state
- Full **AluVM** support; removal of legacy hardcoded validation rules
- Significantly **reduced number of dependencies** and the size of consensus-critical code base
  - removal of dependency on Grin Bulletproofs implementation
- **Ossification** begins: new releases will be bugfix-only

  Removal of rust-bitcoin & miniscript dependencies removes the need to update consensus code twice a year just to maintain compatibility
- Compilation to **WASM** target

No ~~forks~~ in RGB (client-side-validation in general):

you can’t ~~split chain~~ since there is no chain

No ~~network splits~~ in RGB, since there is no P2P network

…but there might be protocol changes and potential asset losses due to them, so something should be there

Types of RGB consensus changes

- Something invalid becomes valid: **fast-forward** (not a ~~**hardfork**~~!)
  - *Existing owners are not affected*
  - *New beneficiaries must upgrade wallets*
- Something valid becomes invalid: **pushback** (not a ~~**softfork**~~!)
  - *Existing owners may loose assets if they update the wallet*
  - *In fact a new protocol, not RGBv1 anymore*
  - *Can happen only through issuers re-issing assets on a new protocol - and users using two wallets (for RGBv1 and new protocol)*

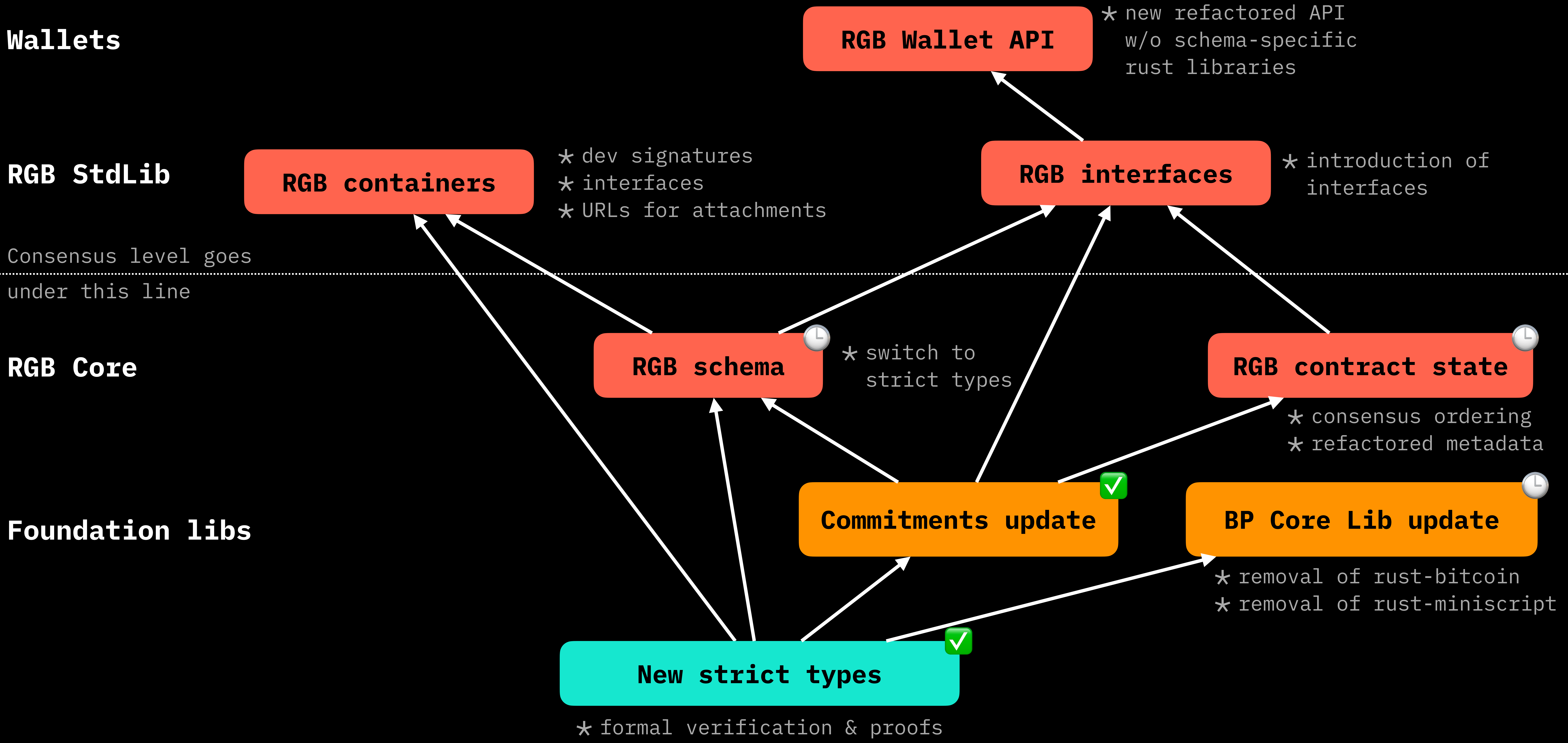
- **RGBv0.10** will be a **pushback**
- If successful, it would become **RGBv1**
- All future features added to RGB must be **fast-forwards** only
- Pushbacks might happen only due to a critical bugfix, and will happen via asset issuers re-issuing the assets on a new protocol (RGBv1_fix)
- **RGBv0.10 will require assets creators to re-issue assets for all current holders**

Future RGB fast-forwards (late 2023)

- More AluVM opcodes:
  - support for **reflection on bitcoin blockchain** data
  - **cross-contract interaction** and state access
- Zk update with bulletproofs
- Zk history rollups

RGB for wallets - release v0.10

- Running full multithreaded node will not be required for mobiles anymore
- **Unified JSON-based API** for all contract types & interfaces (fungible assets, NFTs, identities, DAOs etc)
- Ability to add **custom schema and contracts without RGB changes**
- Ability to **backup** stash during runtime
- Simplified **invoicing**
- Shorter, simpler and safer contract ids (Baid58 format)
- **Signatures** on assets, contracts, schema and other RGB-related code


Wallets
RGB Wallet API
⋆ new refactored API w/o schema-specific rust libraries
RGB StdLib
RGB containers
⋆ dev signatures
⋆ interfaces
⋆ URLs for attachments
RGB interfaces
⋆ introduction of interfaces
Consensus level goes under this line
RGB Core
RGB schema
⋆ switch to strict types
RGB contract state
⋆ consensus ordering
⋆ refactored metadata
Foundation libs
Commitments update ✅
BP Core Lib update
⋆ removal of rust-bitcoin
⋆ removal of rust-miniscript
New strict types ✅
⋆ formal verification & proofs

What's next after RGBv0.10 and RGBv1?

- Updating **wallet** & exchange integration stack (RGB Std Lib, RGB Node)
  Feb 2022

- Release main types of RGB contract interfaces for wallets
  (**RGB20** assets, RGB21 **NFTs**, RGB22 **identity**,
  RGB23 decentralized **naming**, RGB24 **DAO**)
  Mar-Apr 2023

- RGB **Lightning** integration
  Mar 2023-Aug 2023

- Lightning **DEX** supporting RGB
  by the end of the year

- Work on RGB toolchain (**Contractum** language)
  by the end of the year

Strict types

strict-types.org

github.com/strict-types

Strict Types

- **Formal verification**
- **New language** (part of Contractum), functional like **Haskell/Idris**, but close to the bare metal like **Rust**
  - autogenerated right from the rust code!
- **Generates** consensus **specs** in automatic fashion!
- Able to **prove** that an implementation follows the spec
- Able to **detect consensus-breaking** changes in new versions and **generate migrations**

Generalized abstract data types (GADT)

Product types

Tuples:

```
struct NewType(OldType);

struct KeyVal(u16, TinyString);
```

```
data NewType :: OldType

data KeyVal :: u16, String
```

Structs:

```
struct Fields {
  name: TinyString,
  value: u64
}
```

```
data Fields :: name String, value U64
```

Generalized abstract data types (GADT)

Sum types

Enums:

```
enum Variants {
  One,
  Two,
  Three
}
```

```
data Variants :: one | two | three
```

Associated enums:

```
enum Assoc {
  Index(u8),
  Fields { name: TinyString, value: u64 },
  Three
}
```

```
data Assoc ::
    index U8 |
    fields (name String, value U64) |
    three
```

Simple derivation

Rust

```rust
#[derive(
  StrictType, StrictDumb,
  StrictEncode, StrictDecode
)]
#[strict_type(tags = order)]
enum Assoc {
  #[strict_type(dumb)]
  Index(u8),
  Fields {
    name: TinyString,
    value: Option<u64>
  },
  Three
}
```

Strict types

```
data Assoc ::
    index U8 |
    fields (name String,
            value U64?) |
    three
```

Compiled

```
----- BEGIN STRICT TYPE LIB -----
Id: 9PAgDBAAAGt41sxDmkmXksGHYbVuz4N2zcFiyPnVqQbv
Checksum: mama-jumbo-sinatra

C1N0cmljdFR5cGVzADMAAA1CdWlsZEZyYWdtZW50DUJ1aWxk
RnJhZ21lbnQEAgAFaWRlbnQABQEBBUlkZW504TuN1tWttWWv
FkVuW4Q4cSncj+UuooL09iZrqs7bbQBBmRpZ2l0cwAFAQEFS
WRlbnThO43W1a21Za8WRW5bhDhxKdyP5T+6igvT2Jmuqzttt
ApEZXBlbmRlbmN5CkRlcGVuZGVuY3kGAwJpZAEJVHlwZUxpY
klk5tZzBPpj+Vr6BbBThcI4gFCKEoTeNrr16VLLPaTbIEbmF
tZQEHTGliTmFtZci7OpO8xnSnrmi2hQxVsmC+zQa9pS3hmnj
OpQxBtLA3ZlcgEGU2VtVmVykAEc2f8Fyx1Ercnhl0Ktvczm6
VI6bq+Rn9oXXvzv34MRW51bVZhcmlhbnRzDEVudW1WYXJpYW
50cwUBAAkBB1ZhcmlhbnQ30qMQ

----- END STRICT TYPE LIB -----
```

Semantic typing: type commits to its meaning

Rust

```rust
#[derive(
  StrictType, StrictDumb,
  StrictEncode, StrictDecode
)]
#[strict_type(tags = order)]
enum Assoc {
  #[strict_type(dumb)]
  Indexed(u8),
  Fields {
    name: TinyString,
    value: Option<u64>
  },
  Three
}
```

Strict types

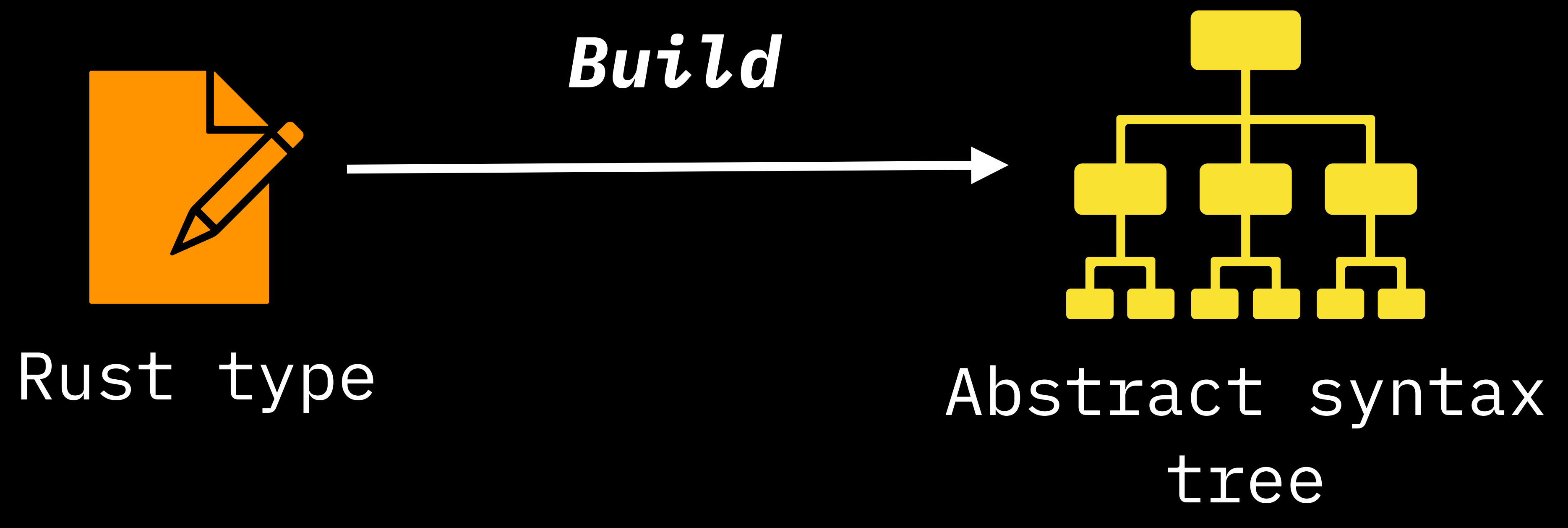
```
data Assoc ::
    indexed U8 |
    fields (name String,
            value U64?) |
    three
```

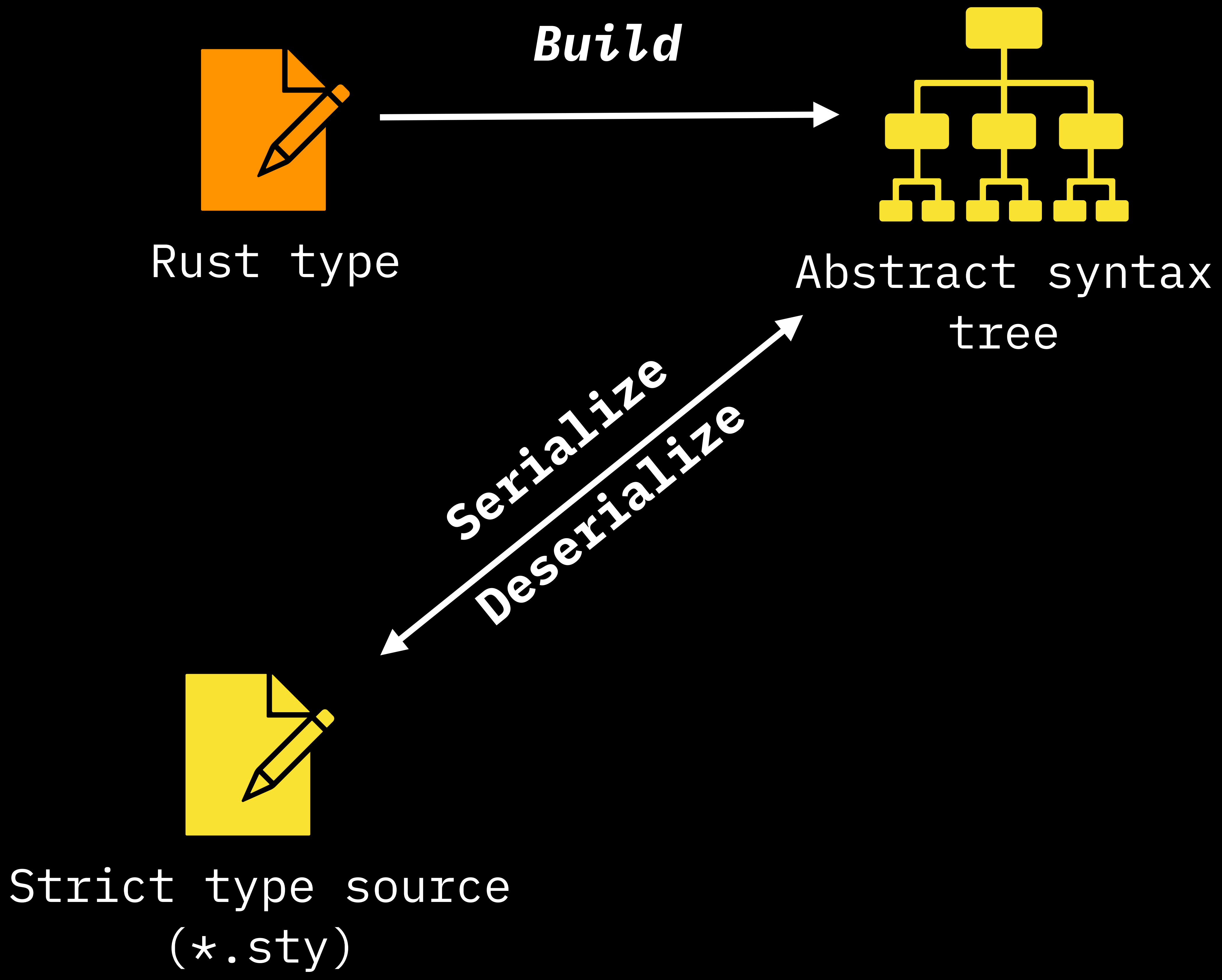
Compiled

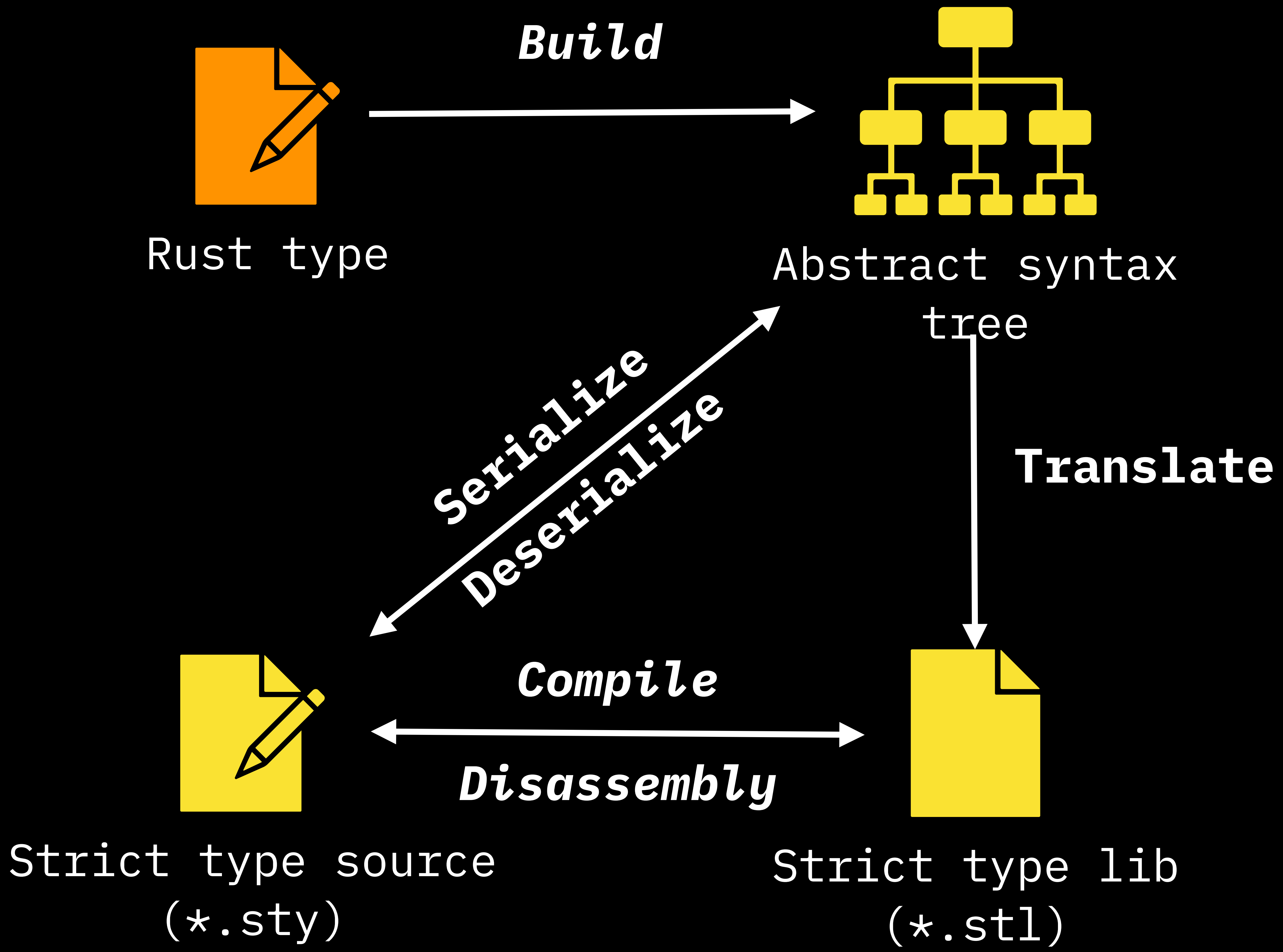
```
----- BEGIN STRICT TYPE LIB -----
Id: G5sL7FHaUo1oBPZ8CXFzqDA7vE3cUzruPMvUbpnBMh3A
Checksum: biology-laser-popcorn
```

GIw8U0kLhlXW1TcgAcKzps4oBAP8AE1VubmFtZWRGaWVsZHN
fS2V5VHkTVW5uYW1lZEZpZWxkc19LZXlUeQUBAAgBBUtleVR
5QrYSbC7jJiGszKH9EFkNSkrGEdaGVA4Xs042DHTi3TMBAP8
AFFVubmFtZWRGaWVsZHNfTGliUmVmFFVubmFtZWRGaWVsZHN
fTGliUmVmBQEACAEGTGliUmVmCLOjKIBPb66CELjy1OZzTHD
R85yeNspiKA+M7az9IBAP8AB1ZhcmlhbnQHVmFyaWFudAYCB
G5hbWUBCUZpZWxkTmFtZbxW3YEOJrtvkYH20lA1xY59BYr+8
WlnLSifjNHyP2h6A3RhZwAAARVWYXJpYW50SW5mb19Jbmxpb
mVSZWYVVmFyaWFudEluZm9fSW5saW5lUmVmBgIEbmFtZQEJR
mllbGROYW1lvFbdgQ4mu2+RgfbSUDXFj

```
----- END STRICT TYPE LIB -----
```


Build
Rust type
Abstract syntax tree


Build
Rust type
Abstract syntax tree
Serialize
Deserialize
Strict type source (*.sty)


Build
Rust type
Abstract syntax tree
Serialize
Deserialize
Translate
Compile
Disassembly
Strict type source (*.sty)
Strict type lib (*.stl)

Real-world example

https://github.com/strict-types/strict-types#strict-types-library

strict type system formally describing and proving itself

Parcel.toml

```toml
[dependencies]
StrictTypes = mama_jumbo_sinatra_9PAgDBAAAGt41sxDmkmXksGHYbVuz4N2zcFiyPnVqQbv
```

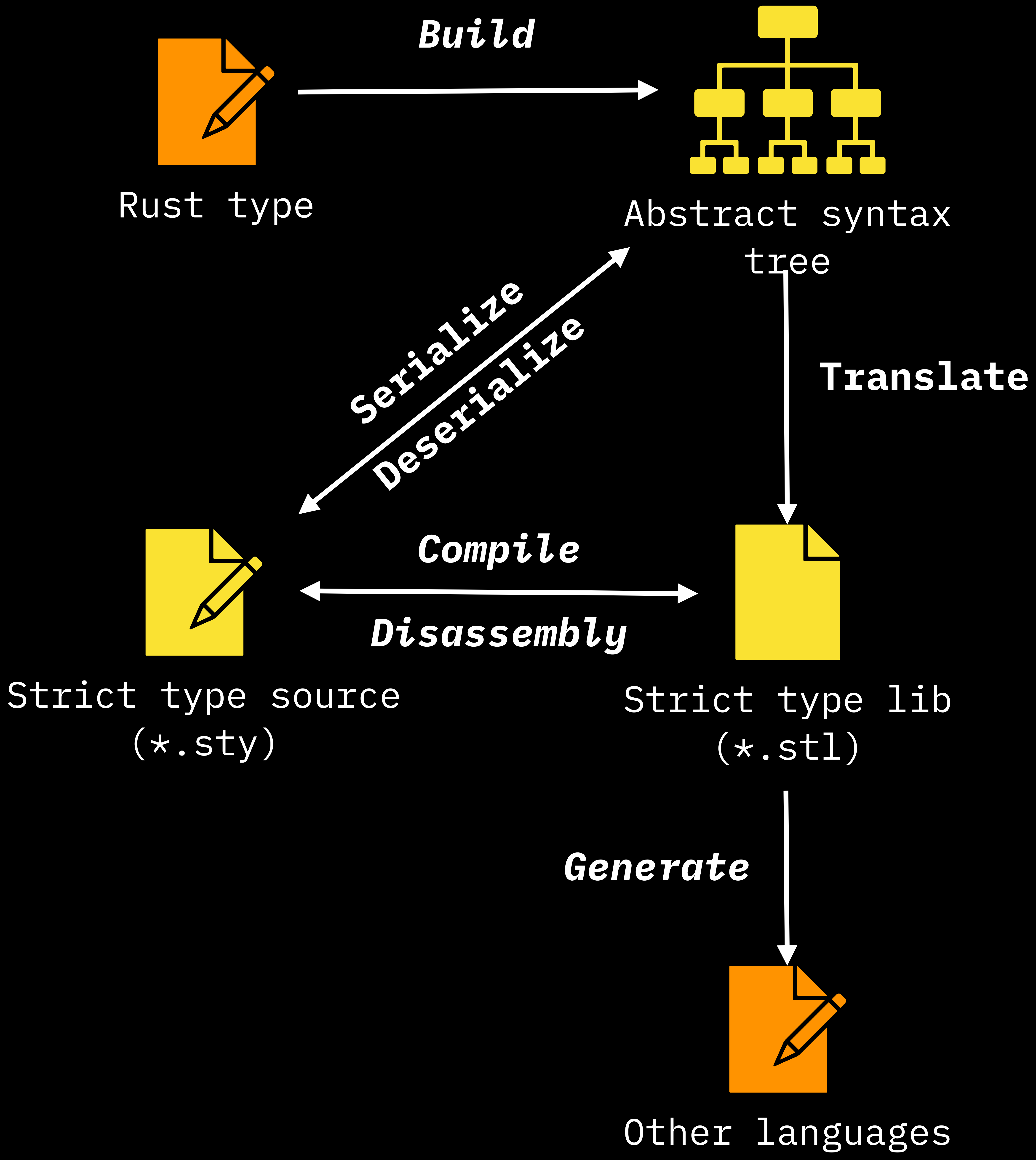
MyLibrary.sty

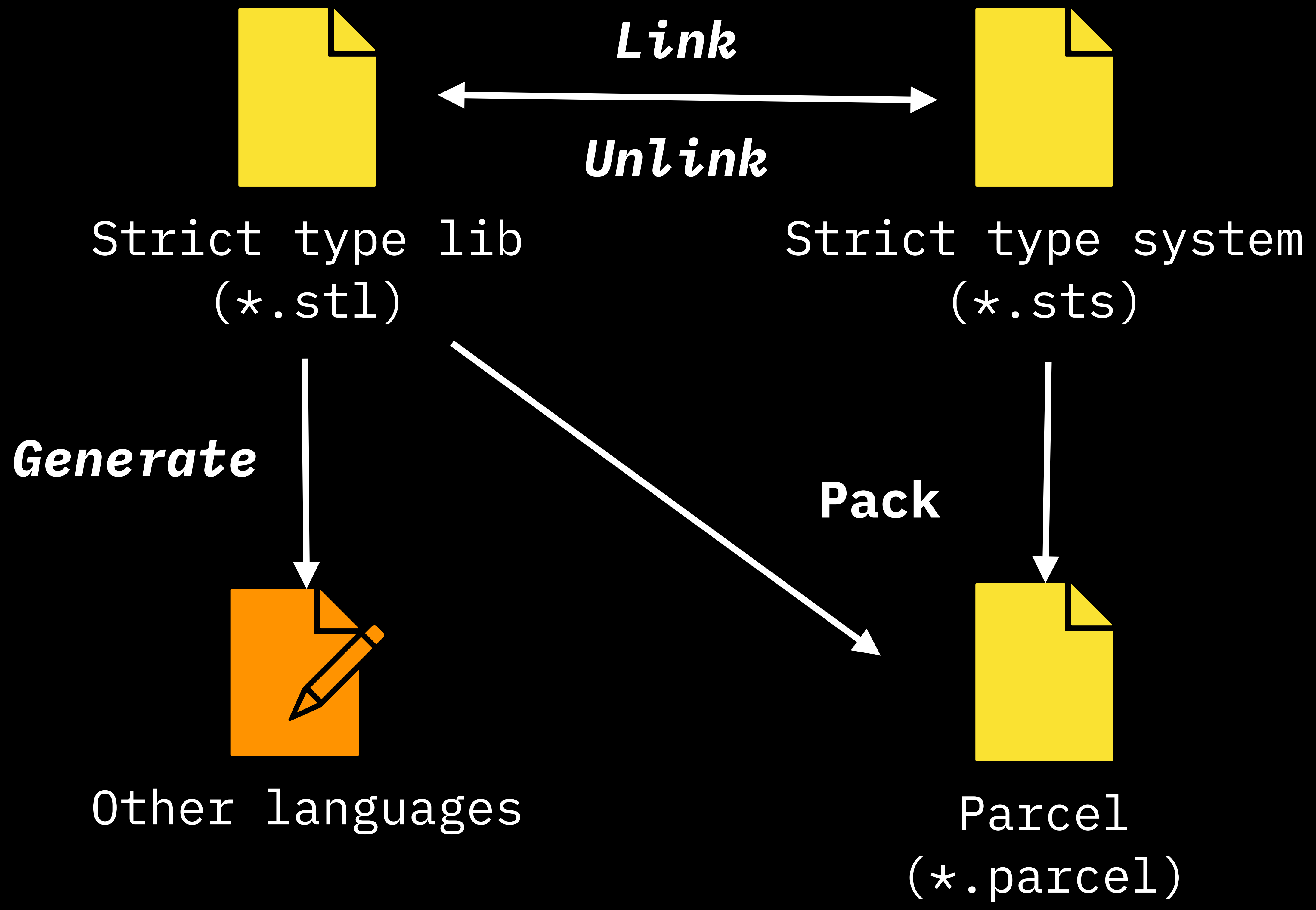
```
typelib MyLibrary

import StrictTypes -- we can be sure that this is the correct library
```

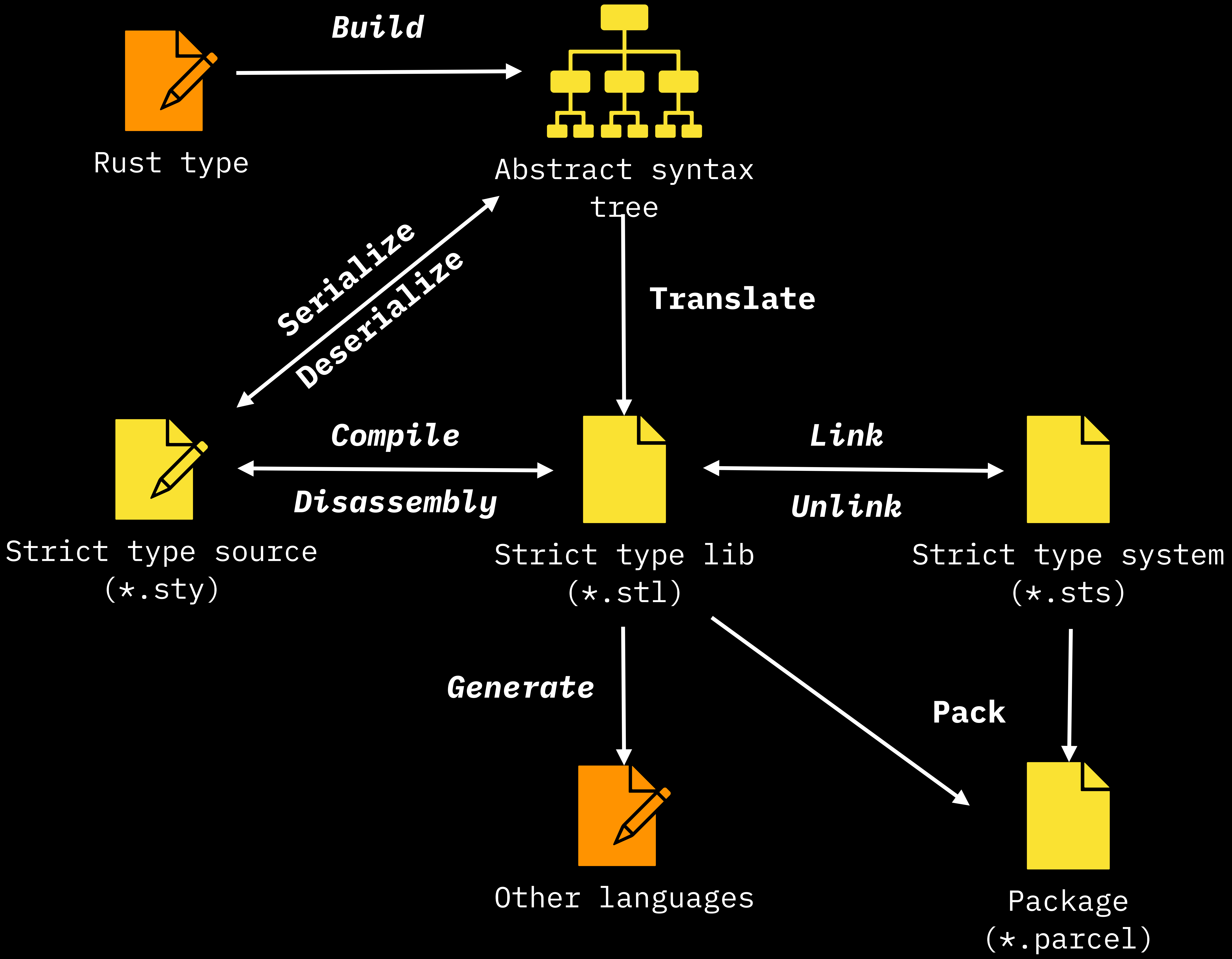
Initial set of libraries

- **CommitEncode**: client-side-validation primitives like LNPBP4
- **BP**: bitcoin protocol, including blockchain and single-use-seals
- **RGB**: all RGB consensus-level structures

not developed, but autogenerated from the real rust implementation, i.e. strictly equivalent to it!


Build
Rust type
Abstract syntax tree
Serialize
Deserialize
Translate
Compile
Disassembly
Strict type source (*.sty)
Strict type lib (*.stl)
Generate
Other languages


Link
Unlink
Strict type lib
(*.stl)
Strict type system
(*.sts)
Generate
Pack
Other languages
Parcel
(*.parcel)


Rust type
Build
Abstract syntax tree
Serialize
Deserialize
Translate
Strict type source (*.sty)
Compile
Disassembly
Strict type lib (*.stl)
Link
Unlink
Strict type system (*.sts)
Generate
Pack
Other languages
Package (*.parcel)

Toolchain

- **styx** general tool
  - Disassembler
  - Text armoring
  - Memory layout analyzer
  - Compatibility analyzer
  - JSON, YAML, URLEncode converter
- **styc** compiler (code -> binary)
- **styl** linker
- **styg** code generator for different languages (Rust, Swift, Kotlin, TypeScript)
- **parcel** package manager