Improving LLVM Backend Development with a New TableGen Language Server

2025 AsiaLLVM Developers' Meeting

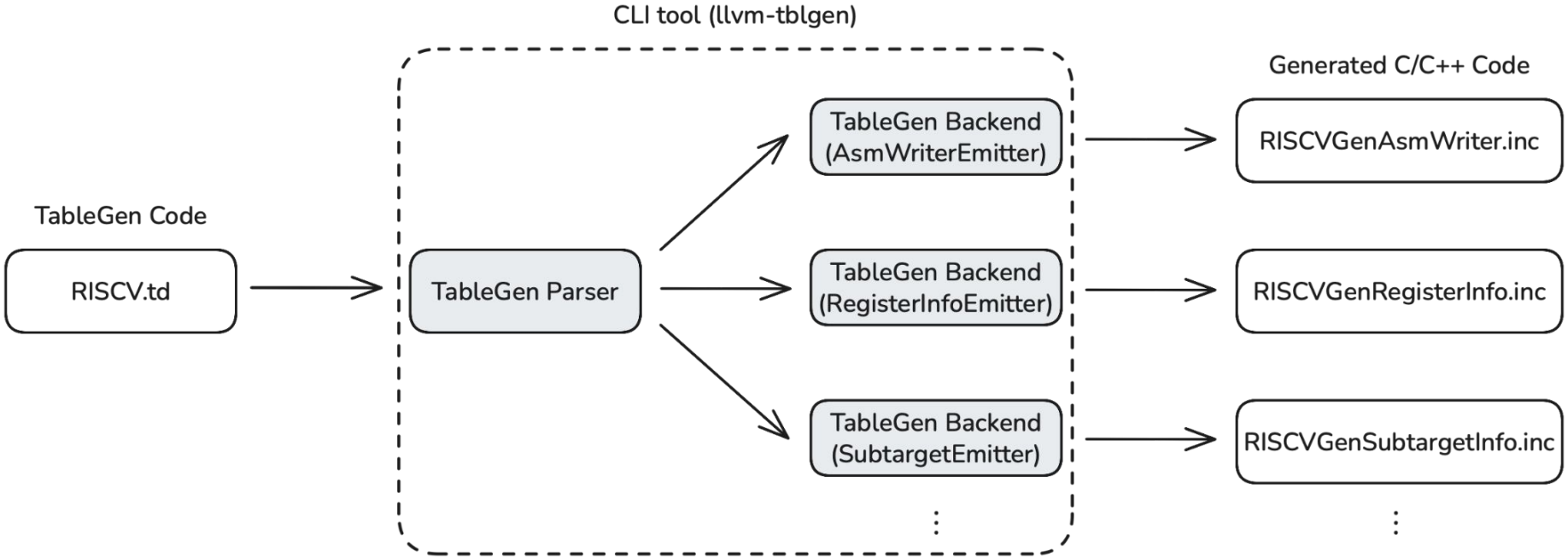
Shin Ando

What is TableGen?

- A **Domain Specific Language (DSL)** used within the LLVM project
- Used for:
  - LLVM backend definitions
  - MLIR dialect definitions
  - Clang CLI options
- Some stats:
  - Over 1,500 files
  - Over 875 KLOC

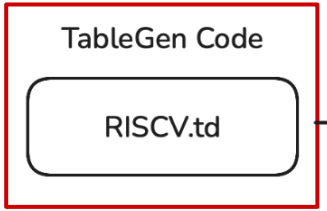
How TableGen works

Example: RISC-V Backend



How TableGen works

Example: RISC-V Backend




```
def ADD  : ALU_rr<0b0000000, 0b000, "add", Commutable=1>,
           Sched<[WriteIALU, ReadIALU, ReadIALU]>;
def SUB  : ALU_rr<0b0100000, 0b000, "sub">,
           Sched<[WriteIALU, ReadIALU, ReadIALU]>;
```

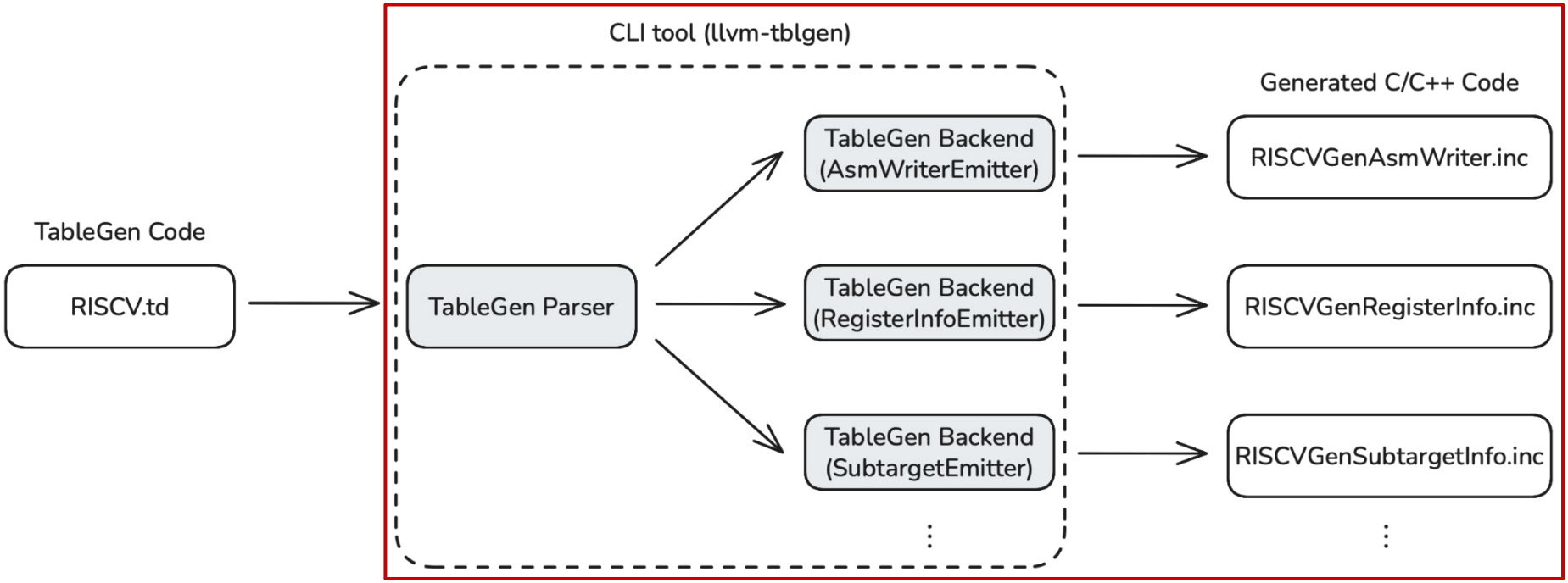
llvm/lib/Target/RISCV/RISCVInstrInfo.td

Defines RISC-V registers, instructions, and encodings, etc.

How TableGen works

Example: RISC-V Backend

Parses .td files,
and runs backends to generate code



Language Server Protocol (LSP)

- Protocol that exposes code analysis features to editors / IDEs
- Enables code completion, go to definition, etc.

https://code.visualstudio.com/api/language-extensions/language-server-extension-guide

Challenges: tblgen-lsp-server

- tblgen-lsp-server is an official TableGen language server implemented as part of MLIR since 2022

Current Challenges:

- **Limited language features**
- **Performance & Stability**
- **Cannot handle the “include anti-pattern”**

Challenge #1: Limited Language Features

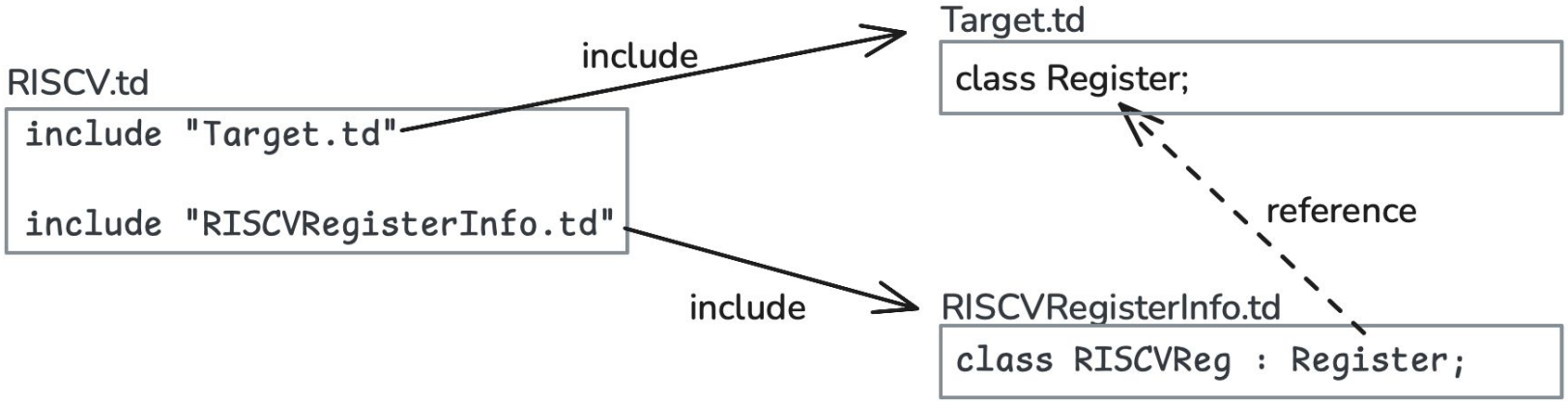
- **Limited set of language features implemented**
  - Supports: diagnostics, go to def, find refs, document link, hover
  - Missing: completion, outline, rename, inlay hint, etc.
- **No new feature development since 2023**
  - It appears that recent activity has focused on bug fixes and refactoring

Challenge #2: Performance & Stability

- **Response is slow for large TableGen code**
  - The server parses and analyzes all TableGen files for each edit
  - Responding can take about 1 second
- **The language server is unstable**
  - Opening a file that contains fatal errors can crash the server
  - Because it uses llvm-tblgen’s parser, which aborts on errors

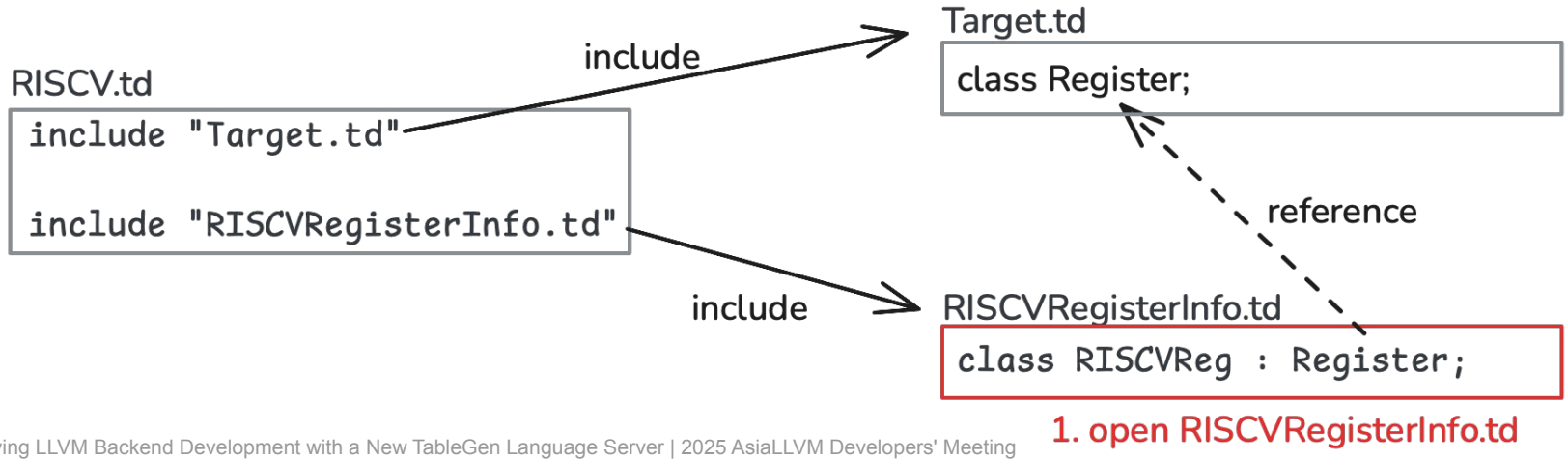
Challenge #3: Cannot Handle the “include anti-pattern” #10

- Cannot handle the “include anti-pattern” commonly seen in LLVM backends
- e.g. When opening RISCVRegisterInfo.td, the server cannot resolve Register



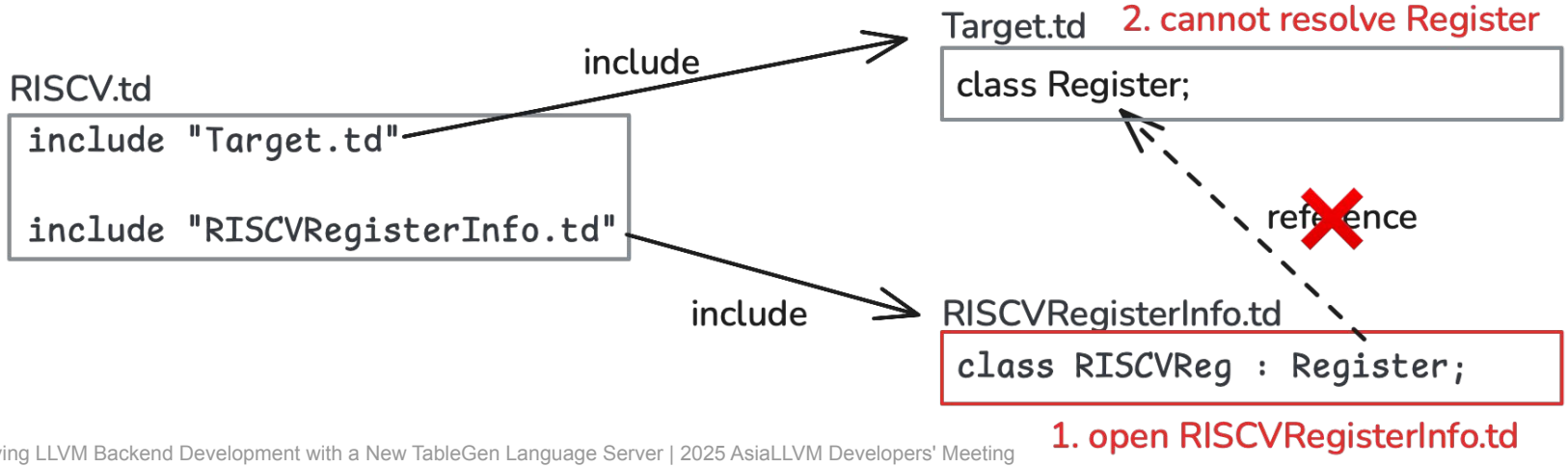
Challenge #3: Cannot Handle the “include anti-pattern” #11

- Cannot handle the “include anti-pattern” commonly seen in LLVM backends
- e.g. When opening RISCVRegisterInfo.td, the server cannot resolve Register



Challenge #3: Cannot Handle the “include anti-pattern” #12

- Cannot handle the “include anti-pattern” commonly seen in LLVM backends
- e.g. When opening RISCVRegisterInfo.td, the server cannot resolve Register



Needs for a TableGen Language Server

It should meet the following needs:

- **Modern, full-featured**
- **Fast, error-tolerant parsing & analysis**
- **Support for the “include anti-pattern”**

tablegen-lsp: A New TableGen Language Server

I’m developing a language server that meets the previous needs

tablegen-lsp: https://github.com/arata-nvm/tablegen-lsp

- More language features
- Incremental & error-tolerant parsing
- Support for “include anti-pattern”

* tablegen-lsp is in early stages and does not support some TableGen syntax

Key Feature #1: More Language Features

In addition to tblgen-lsp-server's language features, it implements the following:

```
1   class Foo1;
2   class Foo2;
3
4   class Bar : F
                 Foo1
                 Foo2
```

Completion

```
23  class HwMode<string FS, list<Predicate> Ps> {
24    // A string representing subtarget features that turn on this HW mode.
25    // For example, "+feat1,-feat2" will indicate that the mode is active
26    // when "feat1" is enabled and "feat2" is disabled at the same time.
27    // Any other features are not checked.
28    // When multiple modes are used, they should be mutually exclusive,
29    // otherwise the results are unpredictable.
30    string Features = FS;
31
32    // A list of predicates that turn on this HW mode.
33    list<Predicate> Predicates = Ps;
34  }
```

Folding

```
1169  class StandardPseudoInstruction : Instruction {
1170    let mayLoad :bit = false;
1171    let mayStore :bit = false;
1172    let isCodeGenOnly :bit = true;
1173    let isPseudo :bit = true;
1174    let hasNoSchedulingInfo :bit = true;
1175    let Namespace :string = "TargetOpcode";
1176  }
```

Inlay Hints

```
OUTLINE
  HwMode class
    FS string
    Ps list<Predicate>
    Features string
    Predicates list<Predicate>
  DefaultMode def
```

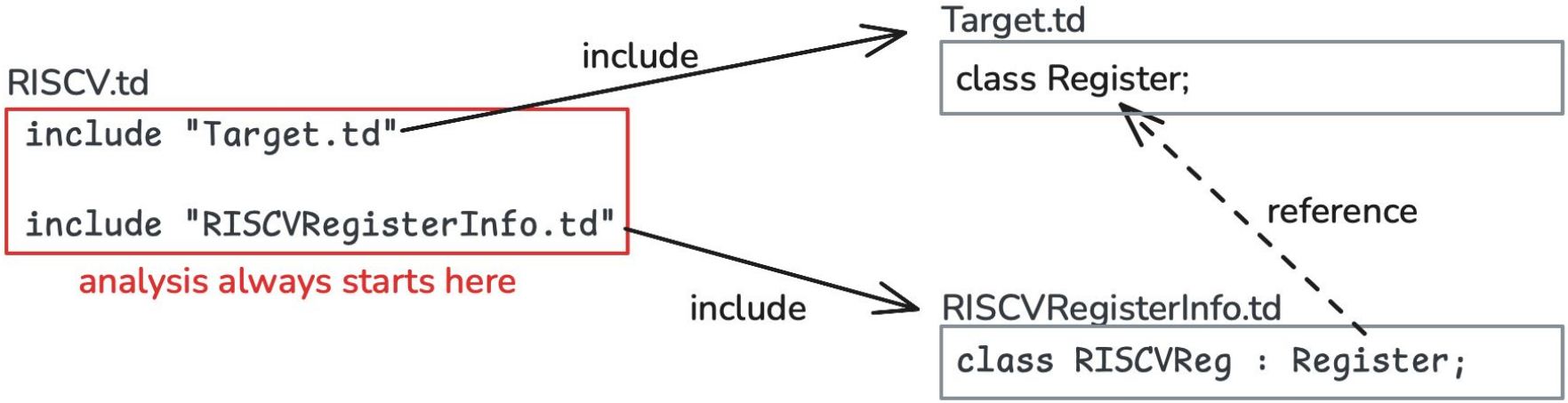
Outline

Key Feature #2: Incremental & error-tolerant parsing

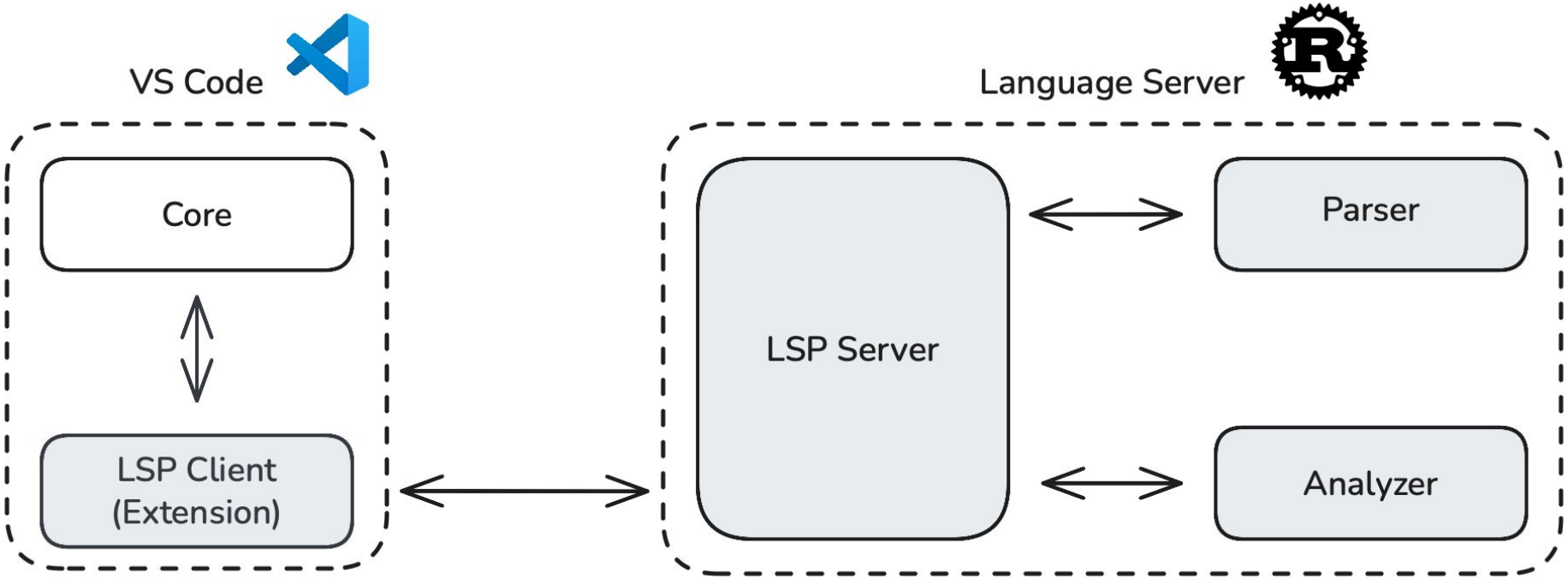
- Uses **rowan** and **salsa** libraries, the same libraries adopted by rust-analyzer
- **rowan**: implements red-green trees to build syntax trees even with syntax errors
  - red-green trees are lossless; preserve comments and whitespace
- **salsa**: enables incremental processing by recomputing only necessary parts when source code changes
  - It caches parse and analysis results

Key Feature #3: Support for “include anti-pattern”

- Users can specify a “**source root**” file
- Analysis always starts from the “**source root**”, regardless of the currently opened file in the editor



Architecture



Future Works

- **Support more TableGen syntax**
  - multiclass, defm, deftype, etc.
- **Integrate with LLVM's TableGen implementation**
  - Important for reducing behavioral differences
- **Implement more language features**
  - Rename, semantic highlight, signature help, etc.

Conclusion

- I’m developing a new TableGen language server
- A VSCode extension is available: https://marketplace.visualstudio.com/items?itemName=arata-nvm.tablegen-lsp
- Please try it out and share your feedback!