MARK SUJEW

Software Engineer

TypeFox

How and Why to build compilers and language servers in Go



Why all this?

What is the use case for building a new language, compiler, editor, and more?



Business Use Cases

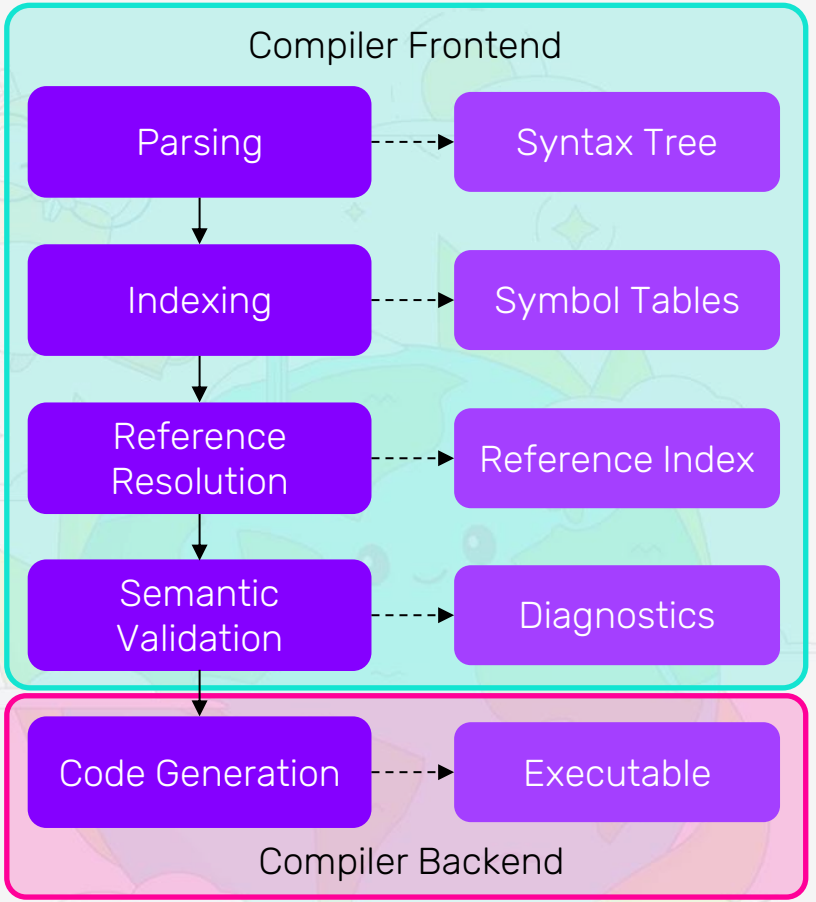
How can you earn a living doing this?



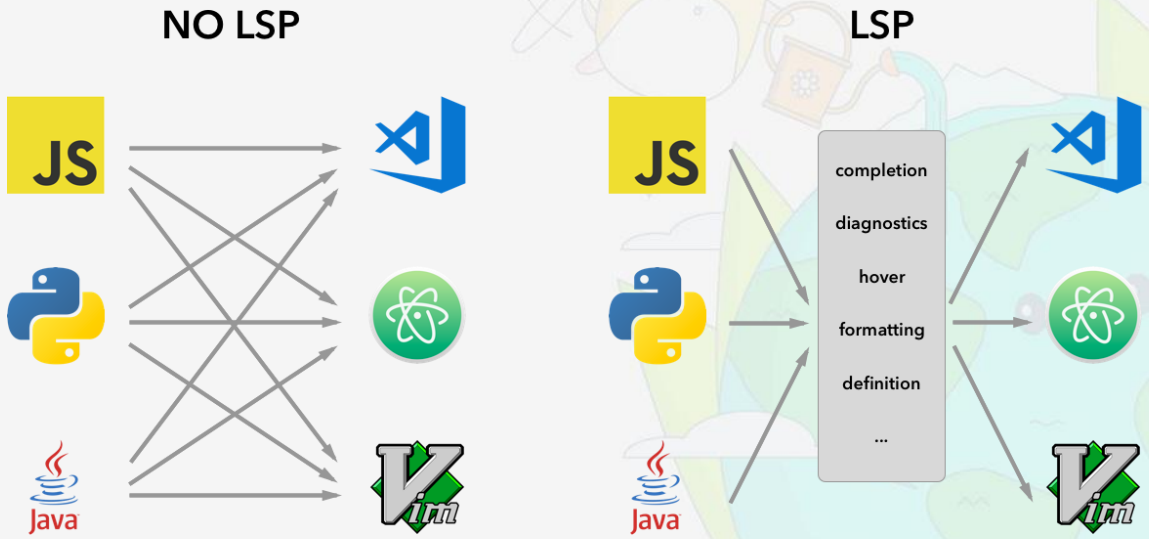
- Tools for Developers
- Low-Code for Domain Experts
- Legacy Modernization

How Does a Compiler Work?

- Takes Code as Input
- Distinct Compiler Phases
- Generates Different Intermediate Artifacts
- Produces Executable!



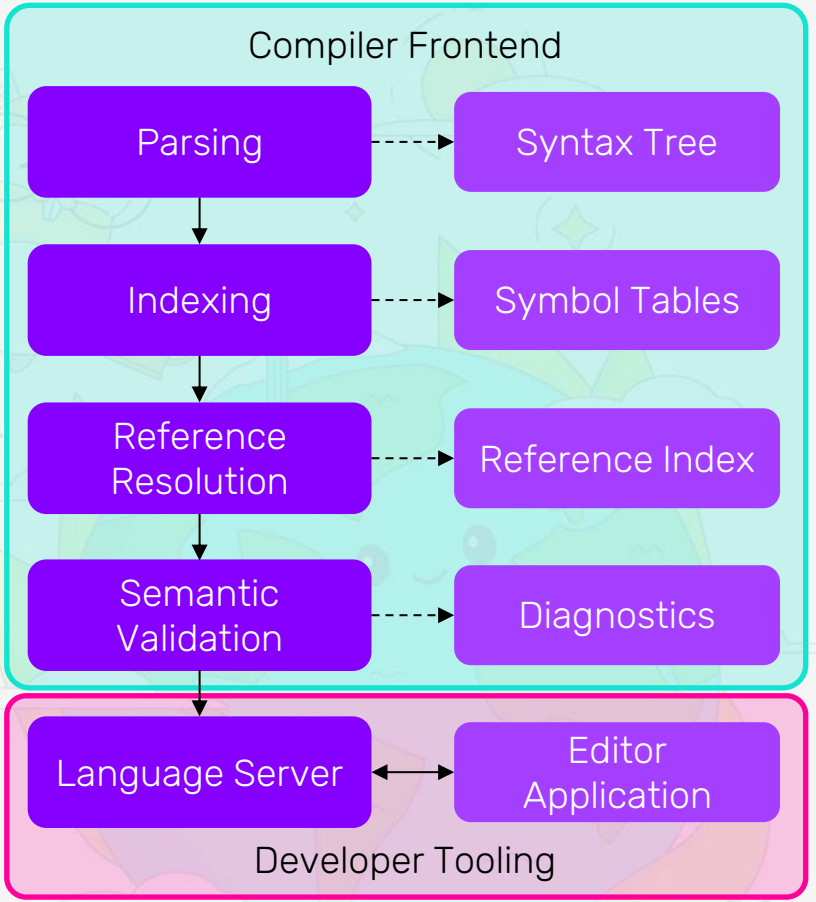
What is a Language Server?



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

How Does a Language Server Work?

- Separate Binary Executable
- Uses Compiler as a Library
- Produces Editor Information!



Lexers & Parsers

- Take Strings as Input
- Lexers produce Tokens
- Parsers produce Trees

```
// Input string
"println(42)"

// Output tokens
[
    "println",
    "(",
    "42",
    ")"
]
```

Lexers & Parsers

- Take Strings as Input
- Lexers produce Tokens
- Parsers produce Trees

```
// Output syntax tree
{
    "type": "FunctionCall",
    "ref": "println",
    "arguments": [{
        "type": "NumberLiteral",
        "value": 42
    }]
}
```

Syntax Trees

- Tree Structure
- Abstraction from Text
- Map back to Text
- See also `go/ast`

```
// Go Struct
type NumberLiteral struct {
    value float64
}
// Abstract Representation
{
    "value": 42
}
// Concrete Representation
{
    "image": "42",
    "offset": 12,
    "length": 2,
    "property": "value"
}
```

Symbol Tables & Scopes

- Canonical Names
- Aliases
- Stored *Locally*
- Many Scope Implementations

```
type Symbol struct {
    Name string
    Node AstNode
}

type SymbolTable struct {
    Symbols []Symbol
}

type Scope interface {
    Resolve(name string) *Symbol
}
```

Symbol Tables & Scopes Examplified

```
interface Item {
    value: string
}

function main() {
    let content: Item
    let newValue = "hello"
    content.value = newValue
}
```

Item
value

main
content
newValue

Symbol Tables & Scopes Examplified

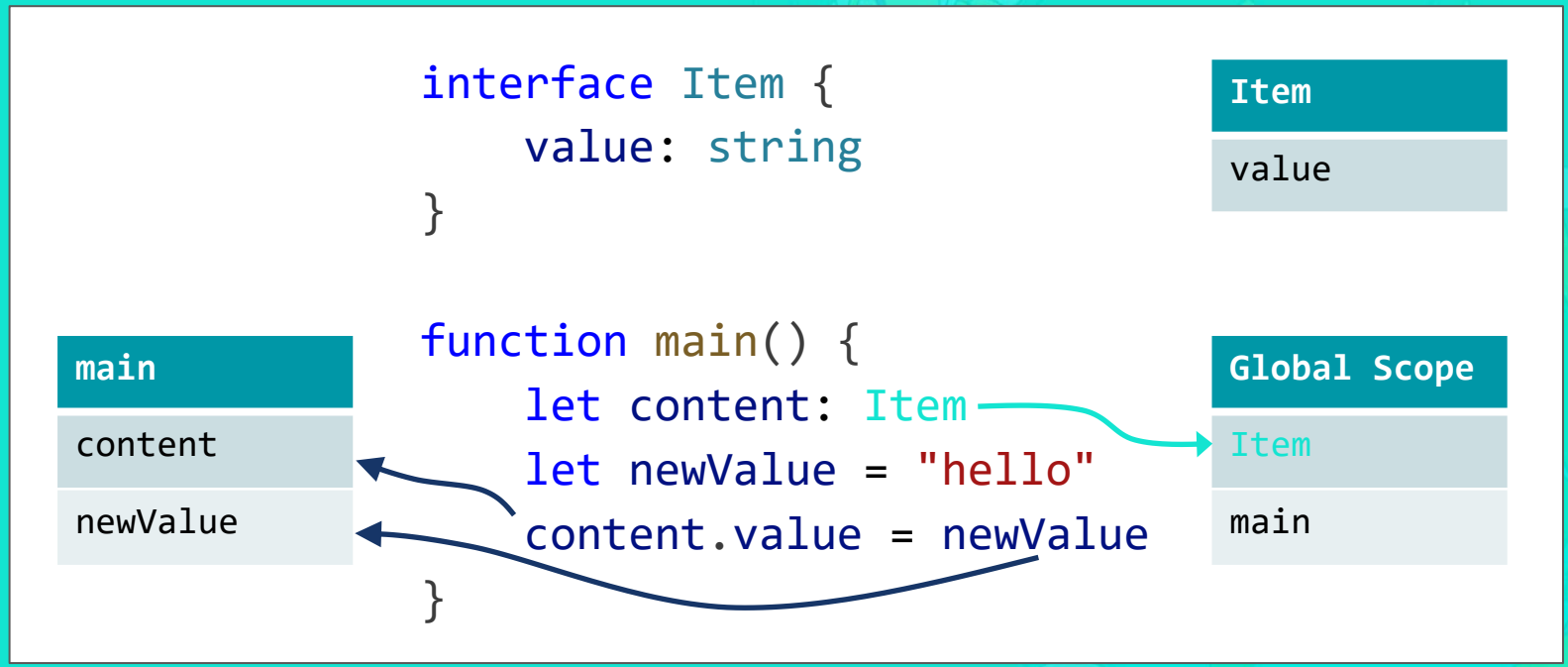
```
interface Item {
    value: string
}

function main() {
    let content: Item
    let newValue = "hello"
    content.value = newValue
}
```

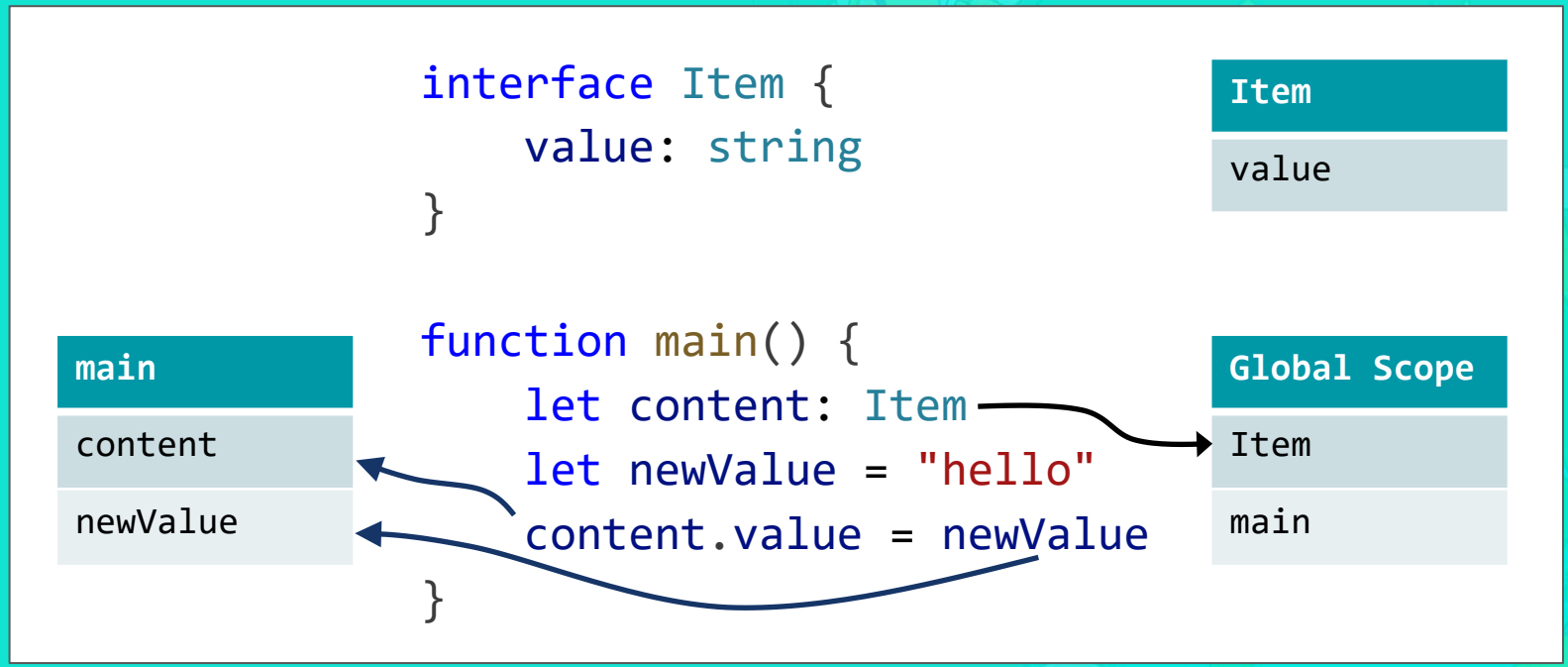
Item
value

main
content
newValue

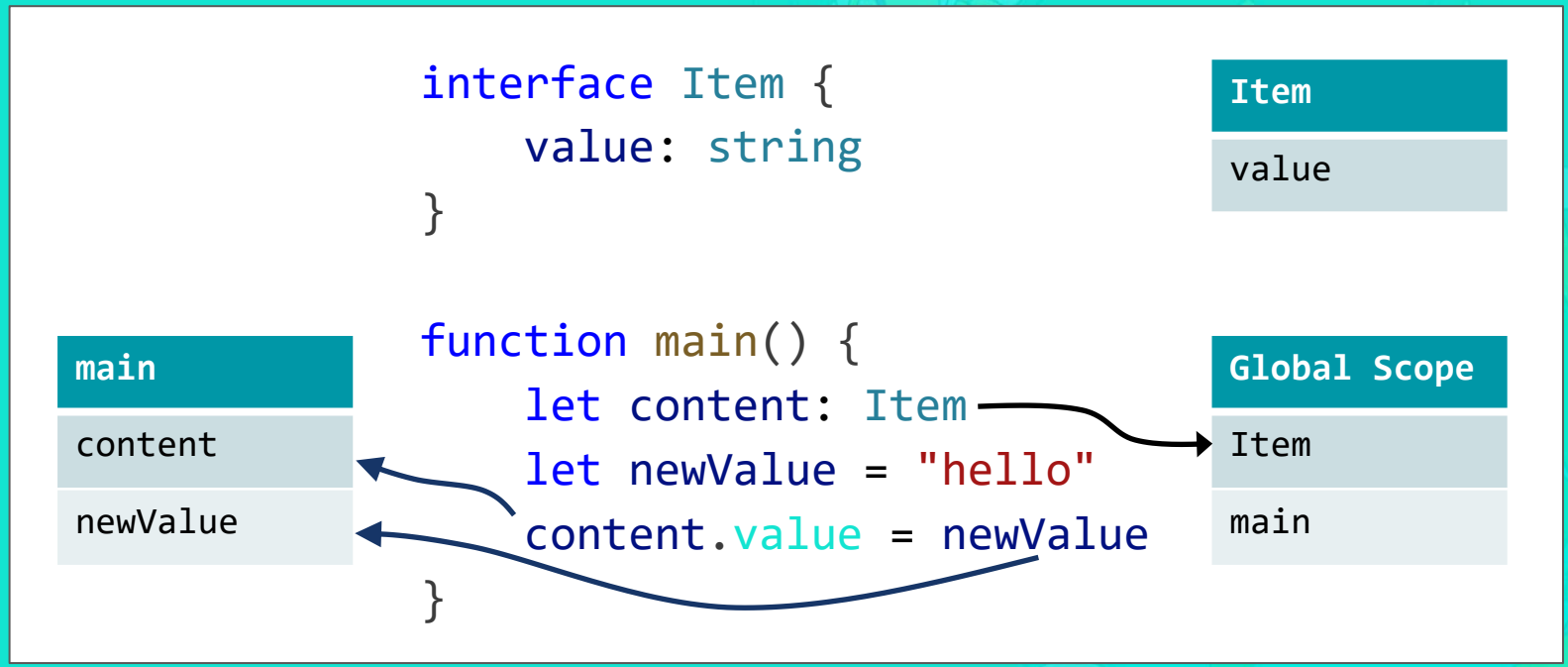
Symbol Tables & Scopes Examplified



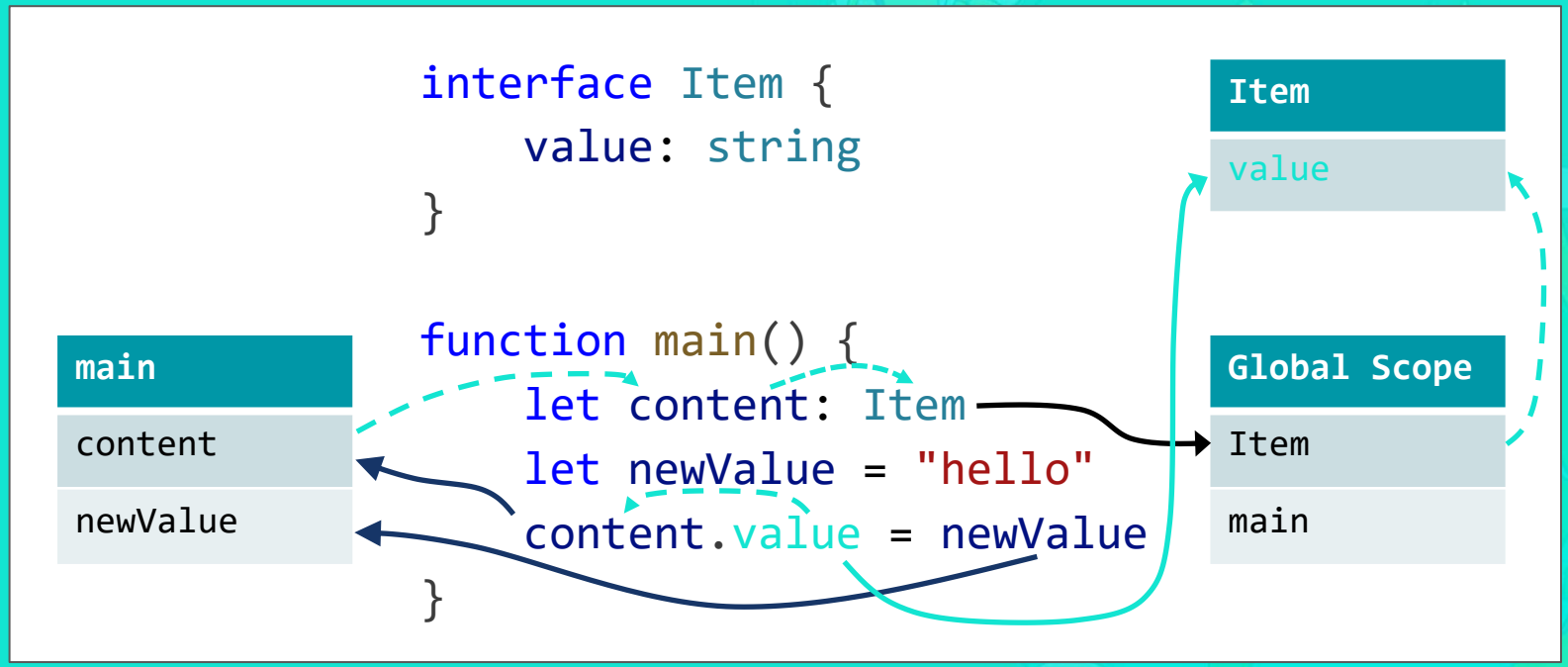
Symbol Tables & Scopes Examplified



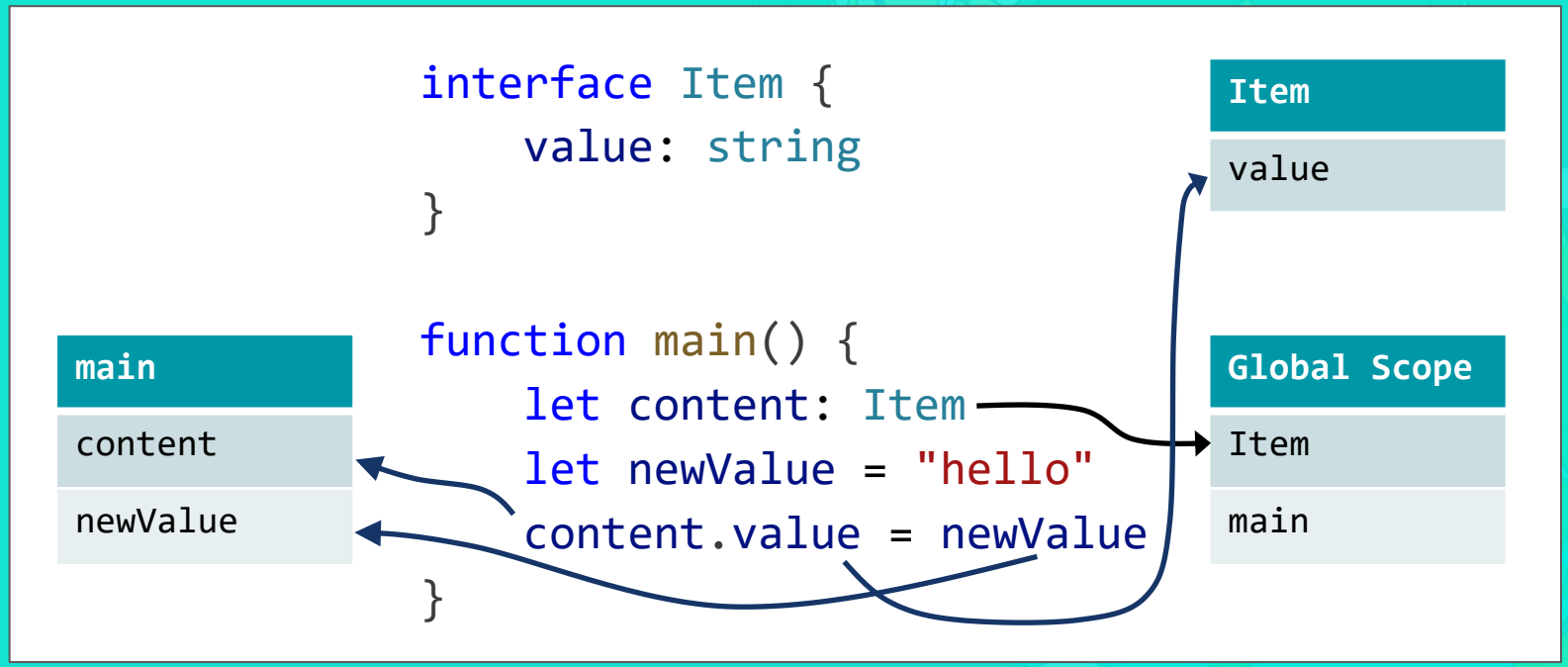
Symbol Tables & Scopes Examplified



Symbol Tables & Scopes Examplified

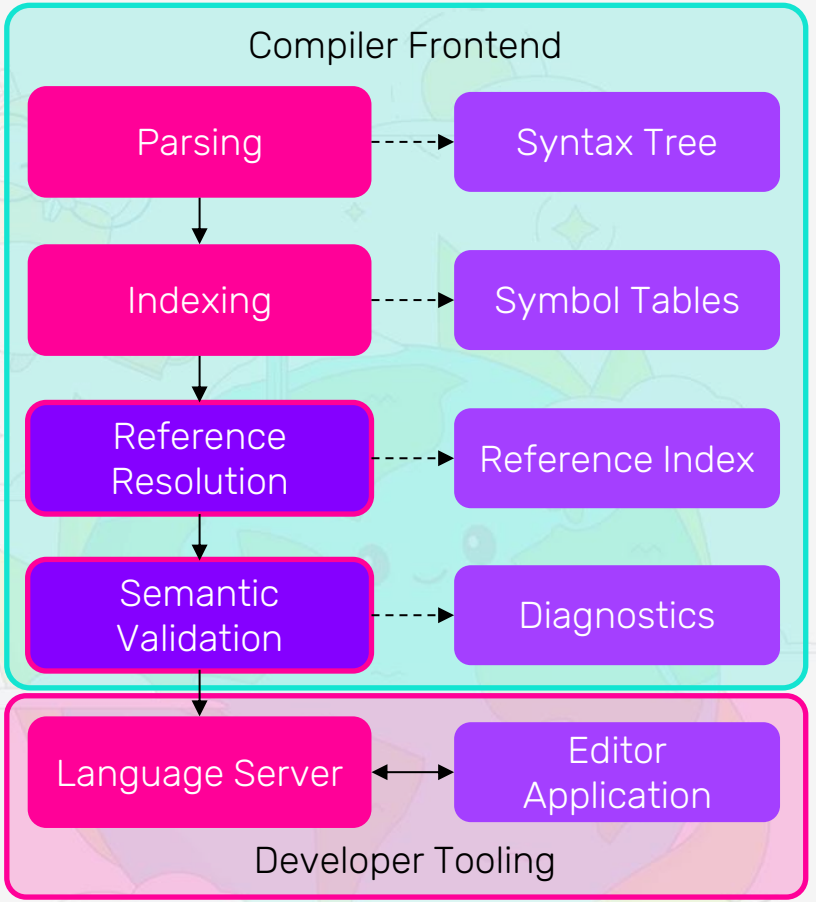


Symbol Tables & Scopes Examplified



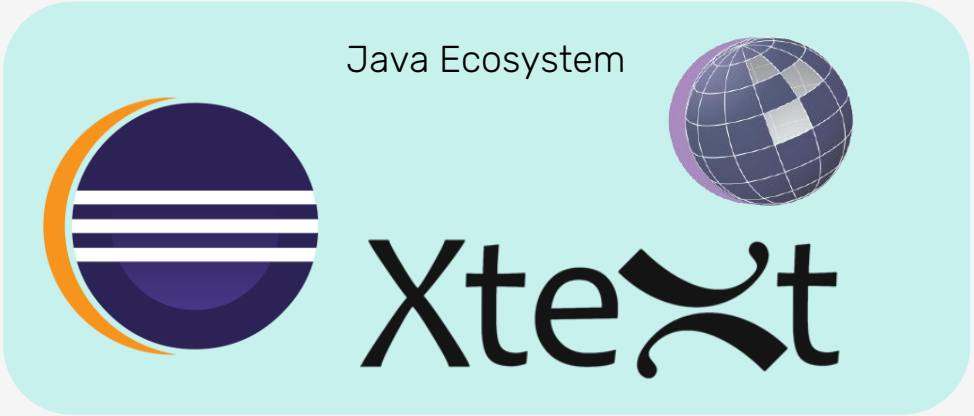
Language Engineering Toolkits

- Write ENBF-Style Grammars
- Receive Out-of-the-Box Functionality
- Reduce Effort and Knowledge Requirement



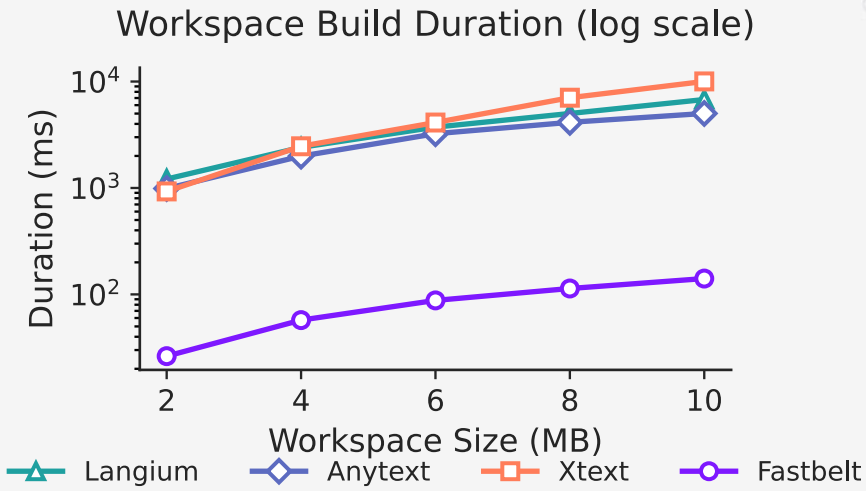
Changing Ecosystems

How to build developer tools in the 2010's





Faster Ecosystems





Mark Sujew and Steven Smyth. 2026. ***Fastbelt: An Architecture for Efficient Language Engineering***. In Proceedings of the ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion 2026), October 4–10, 2026, Málaga.

https://doi.org/10.1145/3837062.3838878

Why not Rust?

Everyone is using Rust in language tooling, why have we decided against it?







So why Go?

- Natural mapping of grammar to types
- Fast enough
- It’s really simple

```
type VariableDeclaration interface {
    core.AstNode
    Statement
    NamedElement
    Type() TypeReference
    Value() Expression
}
// Composable implementation
type VariableDeclarationImpl struct {
    core.AstNodeBase
    StatementData
    NamedElementData
    VariableDeclarationData
}
```

Go Caveats

- Weak Pattern Matching
- LSP Type System
- WASM Performance
- Nil-Handling

```
// this TypeScript definition...
export interface ServerCapabilities {
    hoverProvider?: boolean |
                    HoverOptions;
}

// ...turns into this Go struct
type ServerCapabilities struct {
    HoverProvider any
}
```

Thank you!



MARK SUJEW

mark.sujew@typefox.io

