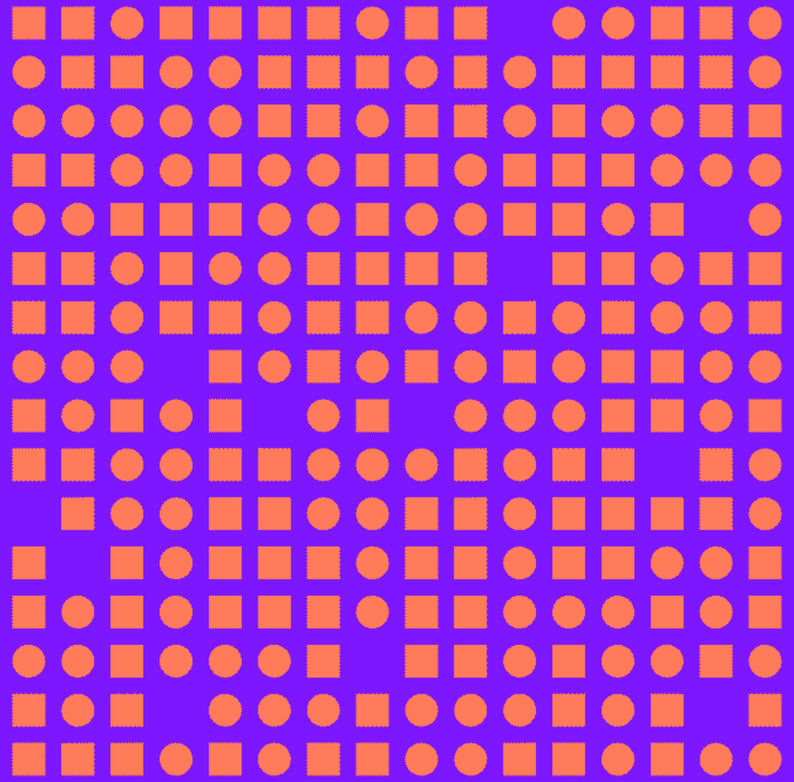
TypeFox

OCX 26 OPEN COMMUNITY EXPERIENCE

OPEN COMMUNITY FOR TOOLING

Learning from 10 Years of Building Programming Languages

Mark Sujew – OCX 2026



Who am I?

Mark Sujew

THEIA

TypeFox

langium

HAW HAMBURG



Some Background

- TypeFox founded in 2016
  - Almost named “XtextSource”
- I started working with Xtext in 2016
- Joined TypeFox in 2021
- Started the Langium Project
  - Now at millions of downloads/week





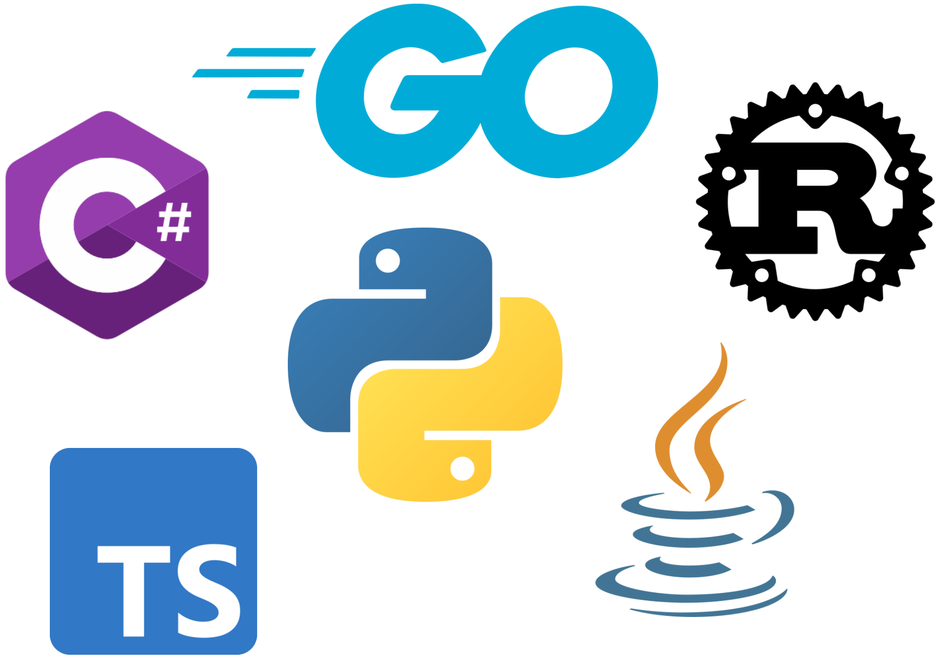






Why Are We Building New Languages?

...aren't there enough already?



General Purpose Languages



Domain Specific Languages

What Even is a Programming Language?

...and what do I need to build one?

- Necessity for most:
  - Scanner
  - Parser
  - Symbol Table
  - Scoping/Linking
- Nice to have:
  - Type System
  - Problem Markers
  - IDE Support



Language Engineering Frameworks

The Promise of Xtext & Langium





- High-Level Grammar Language
- Describe Grammar & Structure
- The Framework provides everything else:
  - Scanner & Parser
  - Syntax Tree Structure
  - Scoping & Linking Infrastructure
  - IDE Integration



Language Engineering Frameworks

The Promise of Xtext & Langium






```
grammar Arithmetics

entry Module:
    'module' name=ID
    (statements+=Statement)*;

Statement:
    Definition | Evaluation;

Definition:
    'def' name=ID
    ('(' args+=DeclaredParameter (',' args+=DeclaredParameter)* ')')?
    ':' expr=Expression ';';

hidden terminal WS: /\s+/;
terminal ID: /[_a-zA-Z][\w_]*/;
```



Language Engineering Frameworks

How do they help us?

- Scanner
- Parser
- Symbol Table
- Scoping/Linking

- Type System
- Problem Markers
- IDE Support

Full Support

Partial Support

Infrastructure Only



Case Studies

EXS and PL/I with Langium

Case Study: EXS

Safety-Critical DSL for Train Networks

- Low-Level Language
- Verbose Syntax
- Single-File Projects
- Huge Files

```
BEGIN_NAMESPACE
  Namespace1
  BEGIN_RULE
    MyRule PROCESSING
    BEGIN_RULE_FRAGMENT
      Frag
      BEGIN_PRECONDITION
        CHECK_PRECONDITION @Condition
      END_PRECONDITION
      BEGIN_ACTION
        BEGIN_VARIABLE
          One INTERMEDIARY TYPE_INTEGER
        END_VARIABLE
        ASSIGN_CONST_INT @MyRule.Frag.One 1
        ADD @Counter @Counter @MyRule.Frag.One
      END_ACTION
    END_RULE_FRAGMENT
  END_RULE
END_NAMESPACE
```

DOI: 10.1109/FDL68117.2025.11165270

Case Study: EXS

Technical Solution

- Compiler/Runtime/Debugger in C
- IDE Support:
  - VS Code Extension
  - Langium Language Server

```
BEGIN_NAMESPACE
  Namespace1
  BEGIN_RULE
    MyRule PROCESSING
    BEGIN_RULE_FRAGMENT
      Frag
      BEGIN_PRECONDITION
        CHECK_PRECONDITION @Condition
      END_PRECONDITION
      BEGIN_ACTION
        BEGIN_VARIABLE
          One INTERMEDIARY TYPE_INTEGER
        END_VARIABLE
        ASSIGN_CONST_INT @MyRule.Frag.One 1
        ADD @Counter @Counter @MyRule.Frag.One
      END_ACTION
    END_RULE_FRAGMENT
  END_RULE
END_NAMESPACE
```

DOI: 10.1109/FDL68117.2025.11165270

Case Study: EXS

Technical Limitations

- Single-threaded Language Server
- Large Files lock up *everything*
- Non-Cancellable Parsing

```
BEGIN_NAMESPACE
  Namespace1
  BEGIN_RULE
    MyRule PROCESSING
    BEGIN_RULE_FRAGMENT
      Frag
      BEGIN_PRECONDITION
        CHECK_PRECONDITION @Condition
      END_PRECONDITION
      BEGIN_ACTION
        BEGIN_VARIABLE
          One INTERMEDIARY TYPE_INTEGER
        END_VARIABLE
        ASSIGN_CONST_INT @MyRule.Frag.One 1
        ADD @Counter @Counter @MyRule.Frag.One
      END_ACTION
    END_RULE_FRAGMENT
  END_RULE
END_NAMESPACE
```

DOI: 10.1109/FDL68117.2025.11165270

Case Study: PL/I

Mainframe Programming Language (1964)

- Unusual Semantics
- Statements for Everything
- Behavior differs from Spec
- Macro Preprocessor
- Highly Configurable
- Huge Legacy Code Bases

```
DCL APPLE_COUNT  FIXED BIN(31) INIT(0);
DCL CARROT_COUNT FIXED BIN(31) INIT(0);
DCL CHERRY_COUNT FIXED BIN(31) INIT(0);
DCL TOMATO_COUNT FIXED BIN(31) INIT(0);
DCL INPFILE FILE RECORD INPUT;
DCL OUTFILE FILE RECORD OUTPUT;

OPEN FILE (INPFILE);
OPEN FILE (OUTFILE);

READ FILE (INPFILE) INTO (REC_INP);
DO WHILE (^EOF);
  SELECT(REC_INP.NAME);
    WHEN('APPLE')  APPLE_COUNT=APPLE_COUNT+REC_INP.COUNT;
    WHEN('CARROT') CARROT_COUNT=CARROT_COUNT+REC_INP.COUNT;
    WHEN('CHERRY') CHERRY_COUNT=CHERRY_COUNT+REC_INP.COUNT;
    WHEN('TOMATO') TOMATO_COUNT=TOMATO_COUNT+REC_INP.COUNT;
    OTHERWISE
      PUT SKIP LIST('Unexpected order ',REC_INP.NAME);
  END;
  READ FILE (INPFILE) INTO (REC_INP);
END;
```



Case Study: PL/I

Macro Preprocessor

```
%DO_READ: PROC (FILE, INTO) STATEMENT;
  ANSWER("DCL " || FILE || " FILE RECORD INPUT;");
  ANSWER("OPEN FILE (" || FILE || ");");
  ANSWER("READ FILE (" || FILE || ") INTO (" || INTO || ");");
END;
%ACT DO_READ;

DO_READ(SOME_FILE, SOME_CONTENT);
DO_READ FILE(OTHER_FILE) INTO(OTHER_CONTENT);
```



```
DCL SOME_FILE FILE RECORD INPUT;
OPEN FILE (SOME_FILE);
READ FILE (SOME_FILE) INTO (SOME_CONTENT);
DCL OTHER_FILE FILE RECORD INPUT;
OPEN FILE (OTHER_FILE);
READ FILE (OTHER_FILE) INTO (OTHER_CONTENT);
```



Case Study: PL/I

Configurability

```
DO WHILE (¬EOF);
  ...
END;


*PROCESS NOT('^');
DO WHILE (^EOF);
  ...
END;


*PROCESS NOT('!');
DO WHILE (!EOF);
  ...
END;

*PROCESS NOT('!^');
DO WHILE (!THIS | ^THAT);
  ...
END;
```

```
*PROCESS RULES(MULTICLOSE);
MAIN: PROC OPTIONS(MAIN);
  DO WHILE(TRUE);
    IF (I < 5) THEN DO;
      I = I + 1;
    END; // optional!
  END; // optional!
END MAIN;

// the same as:

MAIN: PROC OPTIONS(MAIN);
  DO WHILE(TRUE);
    IF (I < 5) THEN DO;
      I = I + 1;
END MAIN;
```



Case Study: PL/I

Technical Limitations

- Large Files → Similar issues as EXS
- Lots of Workarounds for Lexing
- Even more for the Preprocessor
- Not a Type-2 Language (!)

```
DCL APPLE_COUNT  FIXED BIN(31) INIT(0);
DCL CARROT_COUNT FIXED BIN(31) INIT(0);
DCL CHERRY_COUNT FIXED BIN(31) INIT(0);
DCL TOMATO_COUNT FIXED BIN(31) INIT(0);
DCL INPFILE FILE RECORD INPUT;
DCL OUTFILE FILE RECORD OUTPUT;

OPEN FILE (INPFILE);
OPEN FILE (OUTFILE);

READ FILE (INPFILE) INTO (REC_INP);
DO WHILE (^EOF);
  SELECT(REC_INP.NAME);
    WHEN('APPLE')  APPLE_COUNT=APPLE_COUNT+REC_INP.COUNT;
    WHEN('CARROT') CARROT_COUNT=CARROT_COUNT+REC_INP.COUNT;
    WHEN('CHERRY') CHERRY_COUNT=CHERRY_COUNT+REC_INP.COUNT;
    WHEN('TOMATO') TOMATO_COUNT=TOMATO_COUNT+REC_INP.COUNT;
    OTHERWISE
      PUT SKIP LIST('Unexpected order ',REC_INP.NAME);
  END;
  READ FILE (INPFILE) INTO (REC_INP);
END;
```



Learnings

The Good Parts of these Frameworks

- Rapid Setup/Iteration
  - Scaffolding
  - Grammar Change Propagation
  - Type Safety
  - Out-Of-The-Box Features
- Infrastructure
  - Scoping/Linking
  - IDE Support
  - Testing
  - Ecosystem



Learnings

The Not-So-Good Parts

- Automatic Type Inference
- Performance
  - **Memory Pressure**
  - **Inefficient Structures** (CST)
- Parser Flexibility (lack thereof)
- Runtime Constraints
  - **Parallelization**
  - **Deployment**



Learnings

How to do it better?

- Automatic Type Inference

  → Explicit Typing

- Memory Pressure

  → **Use Less Memory (?)**

- Parser Flexibility

  → Better Hooks into Internals

- Runtime Constraints

  → Native Binary



The Memory Problem

How to do it better?

```
((statemachine) (TrafficLight)
((events)
    (switchCapacity)
    (next))

((initialState) (PowerOff))

((state) (PowerOff)
    ((switchCapacity) (=>) (RedLight))
(end))

((state) (RedLight)
    ((switchCapacity) (=>) (PowerOff))
    ((next) (=>) (GreenLight))
(end)))
```

Every (...) is one logical Concrete Syntax Tree Block



The Memory Problem

How to do it better?

- Frameworks generate AST+CST
  - CST yields 80% of memory load
- AST can be completely derived from CST
- … then why keep both in memory?
- Solution: **Token-Enriched AST**

```
((statemachine) (TrafficLight)
((events)
    (switchCapacity)
    (next))

((initialState) (PowerOff))

((state) (PowerOff)
    ((switchCapacity) (=>) (RedLight))
(end))

((state) (RedLight)
    ((switchCapacity) (=>) (PowerOff))
    ((next) (=>) (GreenLight))
(end)))
```



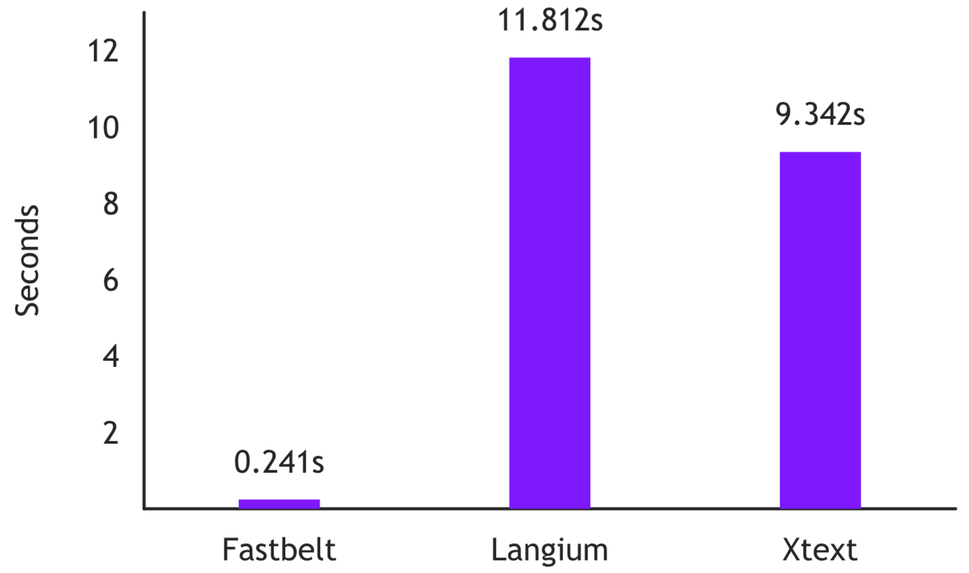
Fastbelt

High-Speed DSL Framework for Go

Fastbelt

In Short

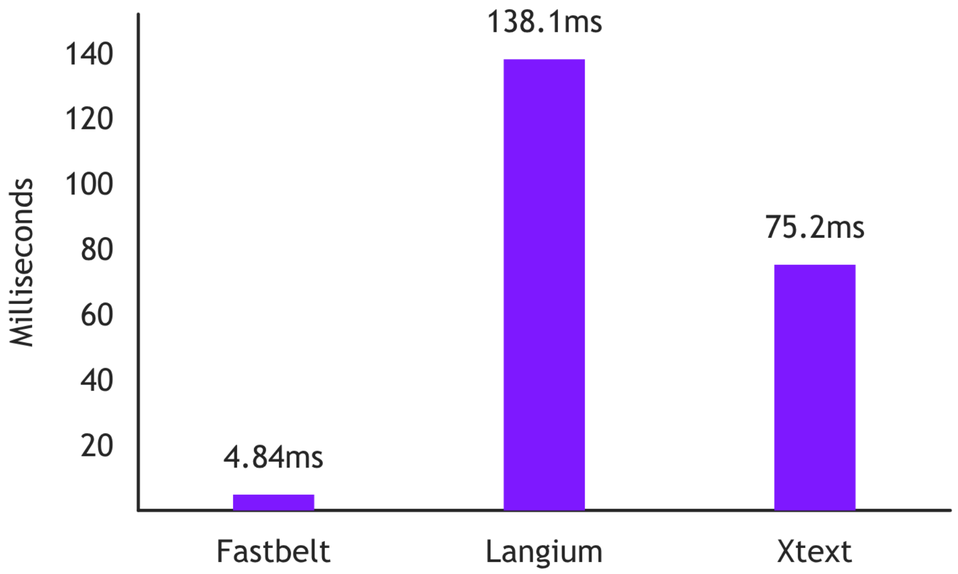
- Everything Xtext & Langium do
- Native Compilation
- Heavy Multi-Threading
- **Much** Faster



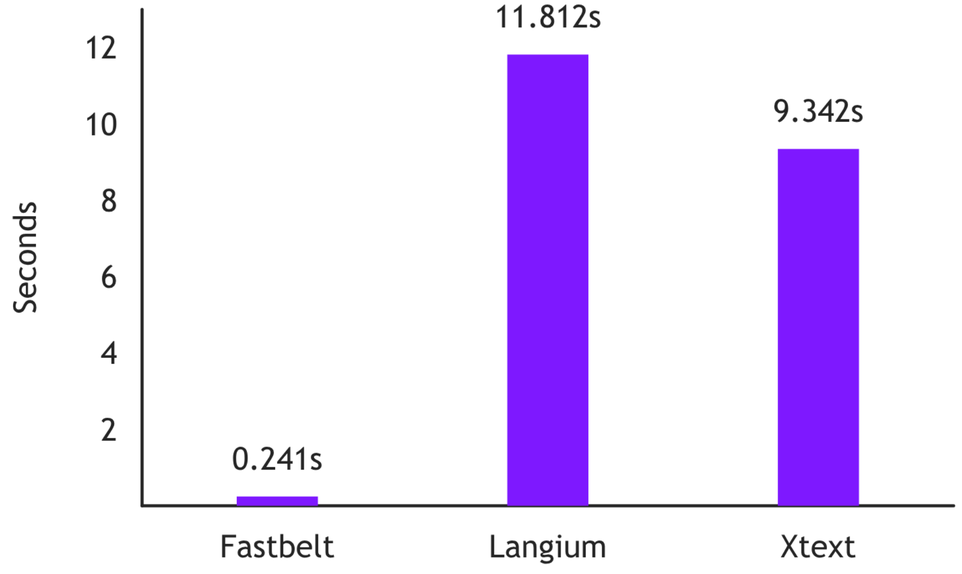
Large workspace build (12MB)

Fastbelt

High-Speed DSL Framework for Go



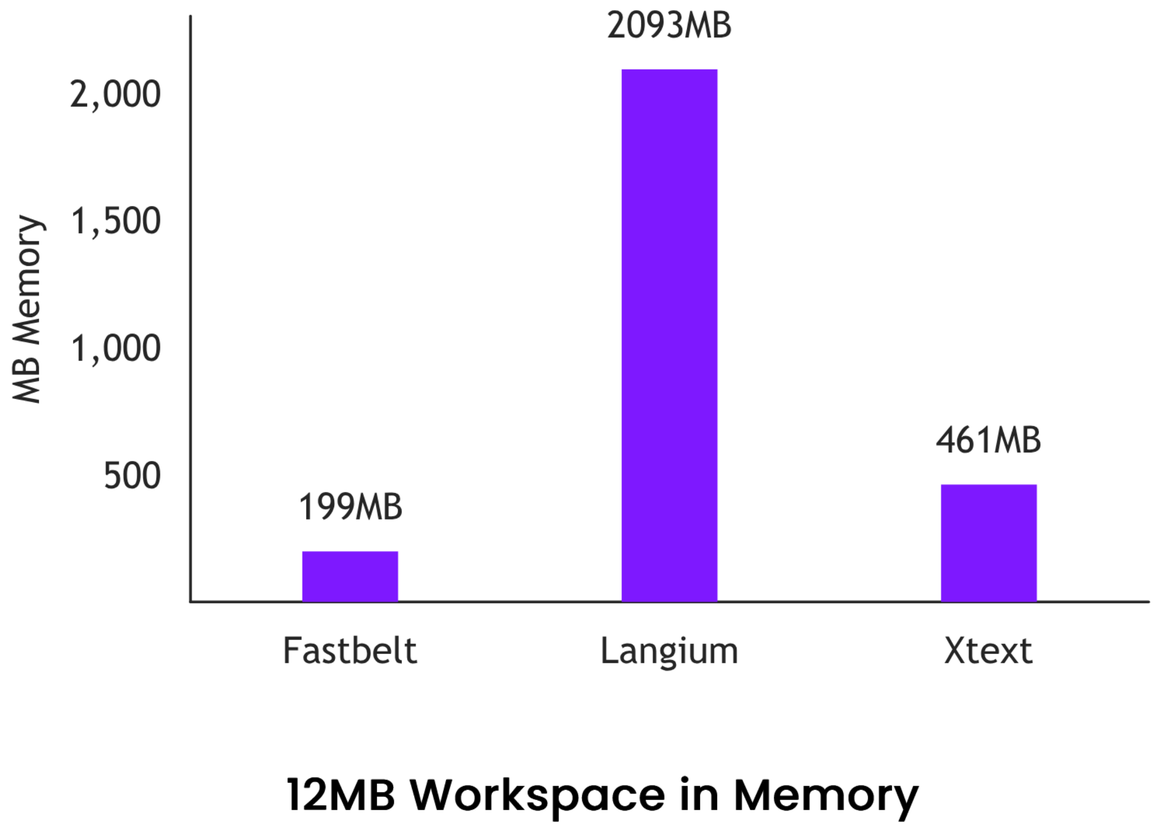
Small workspace build (250KB)



Large workspace build (12MB)

Fastbelt

Memory Consumption



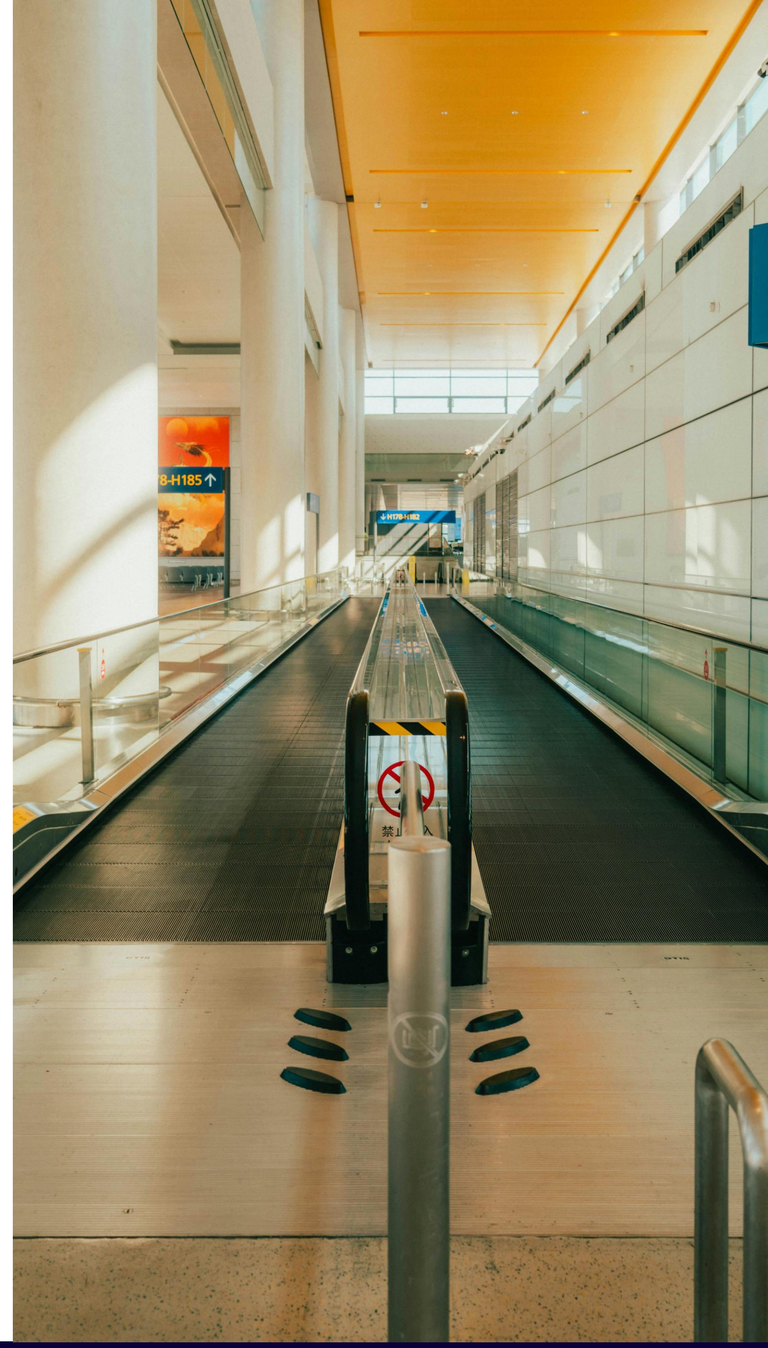
12MB Workspace in Memory

```
type StatemachineData struct {
    tokens    []*Token
    name      *Token
    events    []Event
    commands  []Command
    init      *Reference[State]
    states    []State
}
```

Fastbelt

Current Features

- Grammar Language
- Scanner & Parser Generator
- Linking & Scoping
- Diagnostics
- LSP Support (Definition & References)

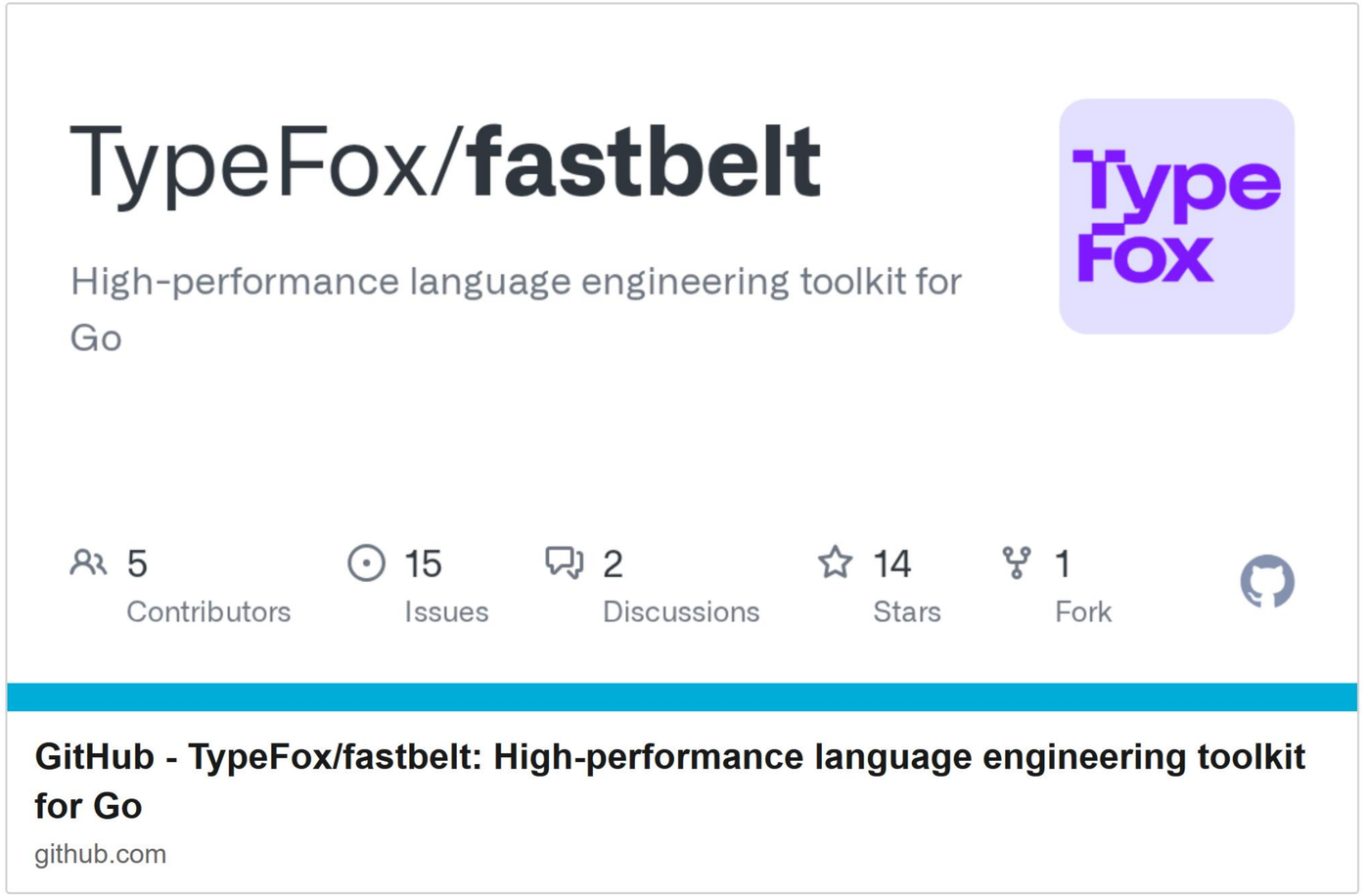


Fastbelt

Roadmap

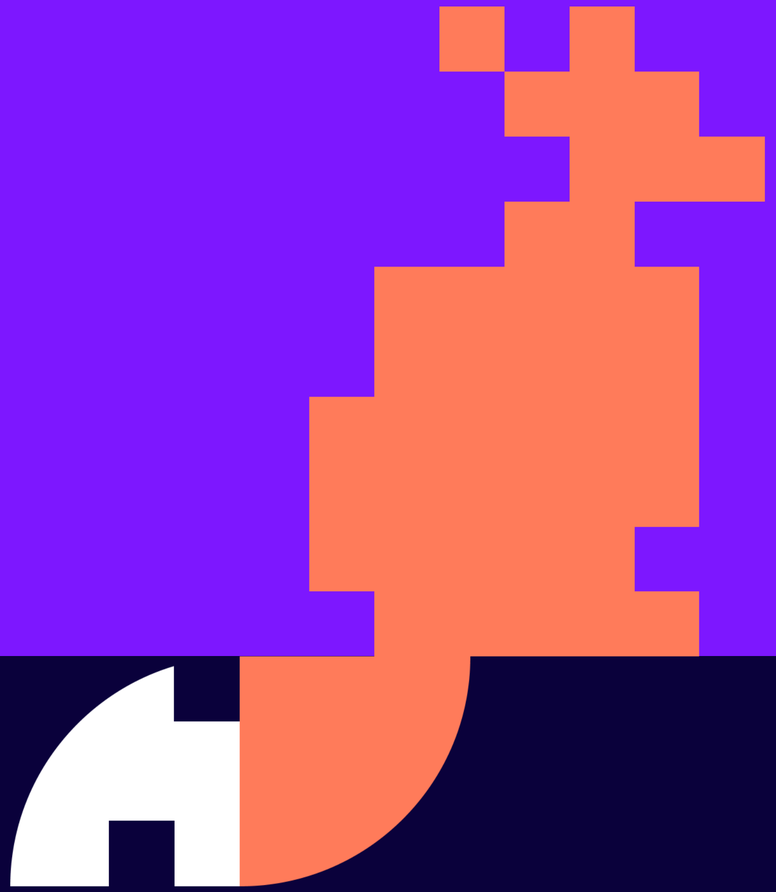


- Parser Recovery
- Improved Parser Lookahead
- Full LSP Support
- Multi-Language Support



Get Involved!

TypeFox



Questions?

OCX 26 OPEN COMMUNITY EXPERIENCE

OCT OPEN COMMUNITY FOR TOOLING