

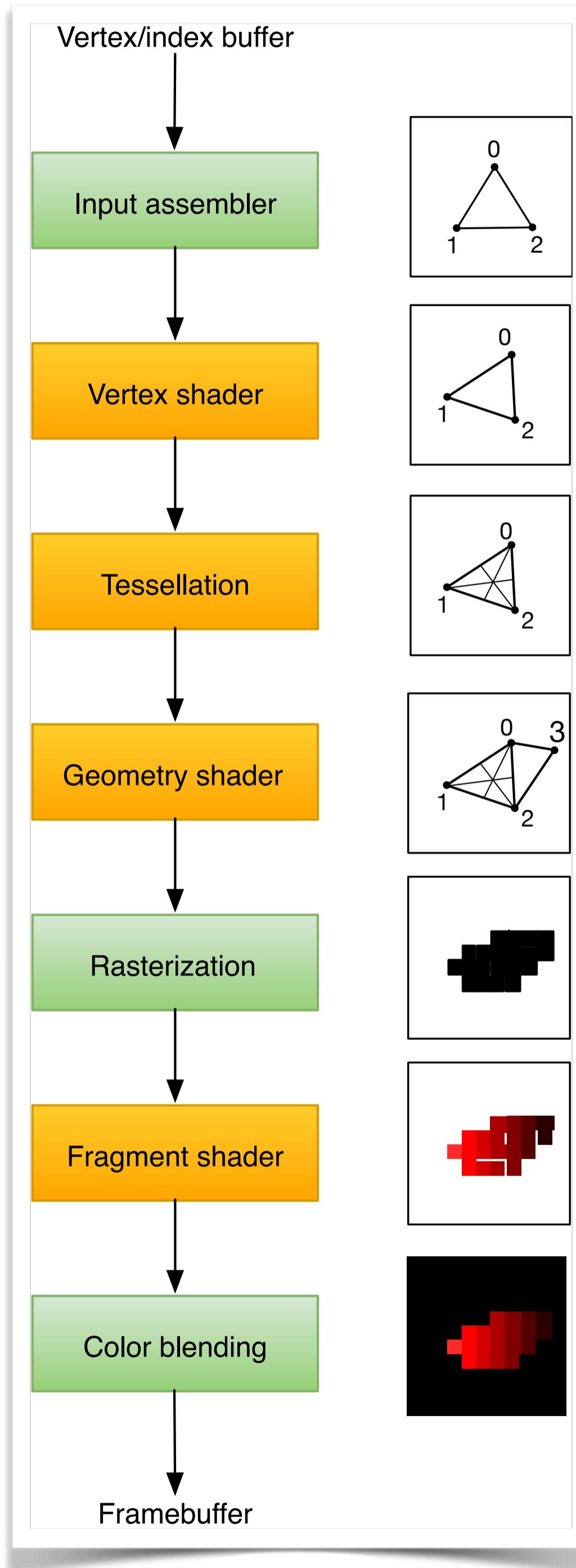
PIE: A System for Programming Interactive Pipelines

Gabriël Konat

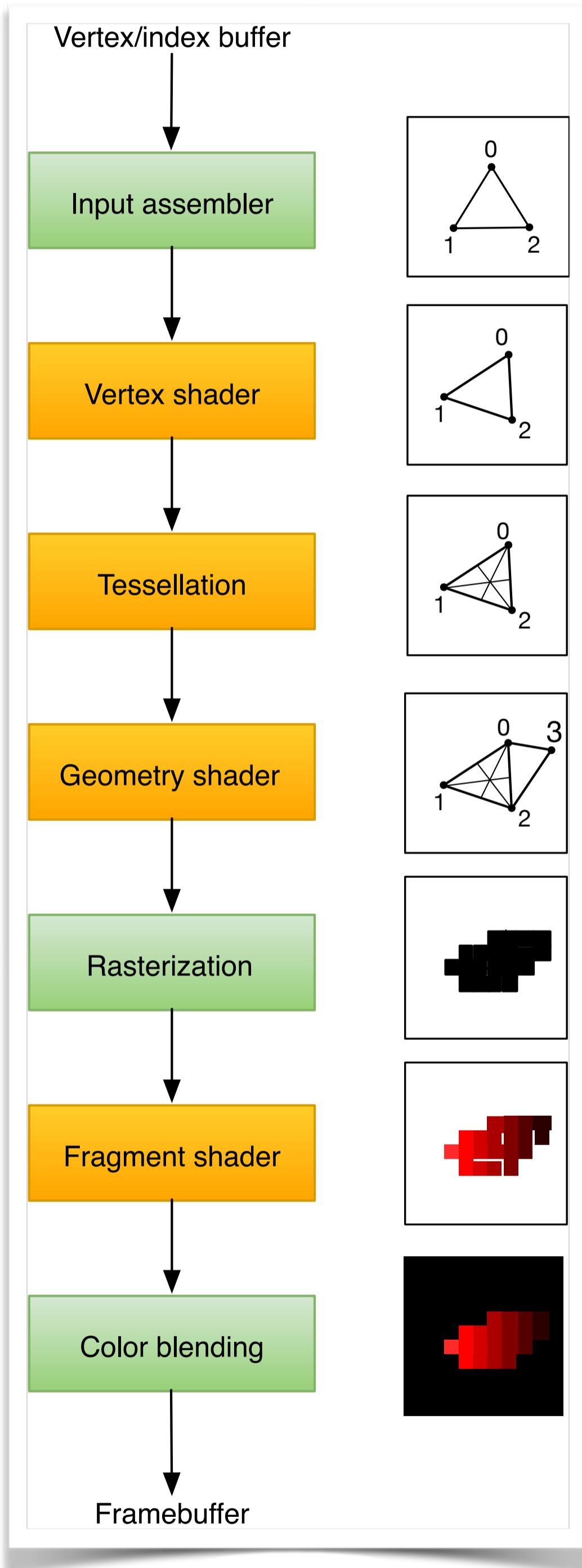
Join work with: Michael Steindorfer, Sebastian Erdweg, Eelco Visser



Pipeline: Flow of Computation



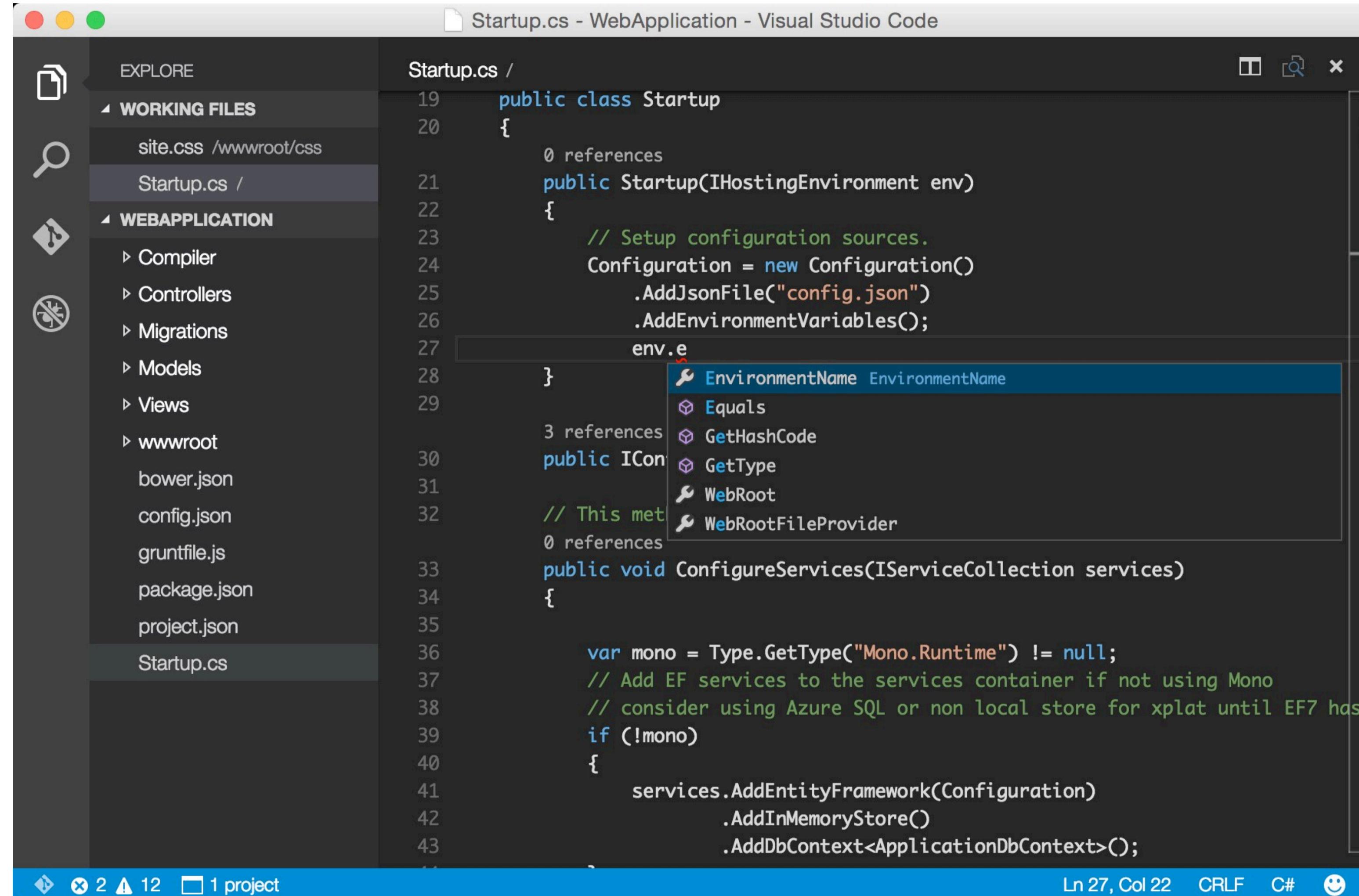
Pipeline: Flow of Computation



```
IDIR =../include  
CC=gcc  
CFLAGS=-I$(IDIR)  
  
ODIR=obj  
LDIR =../lib  
  
LIBS=-lm  
  
_DEPS = hellomake.h  
DEPS = $(patsubst %,$(IDIR)/%, $(DEPS))  
  
_OBJ = hellomake.o hellofunc.o  
OBJ = $(patsubst %,$(ODIR)/%, $(OBJ))  
  
$(ODIR)/%.o: %.c $(DEPS)  
    $(CC) -c -o $@ $^ $(CFLAGS)  
  
hellomake: $(OBJ)  
    gcc -o $@ $^ $(CFLAGS) $(LIBS)  
  
.PHONY: clean  
  
clean:  
    rm -f $(ODIR)/*.* ~ core $(INCDIR)/*~
```

Interactive Pipeline: Immediate Feedback

Interactive Pipeline: Immediate Feedback



A screenshot of the Visual Studio Code interface. The title bar says "Startup.cs - WebApplication - Visual Studio Code". The left sidebar shows a file tree with "WORKING FILES" containing "site.css" and "Startup.cs", and a "WEBAPPLICATION" section with "Compiler", "Controllers", "Migrations", "Models", "Views", "wwwroot", "bower.json", "config.json", "gruntfile.js", "package.json", "project.json", and another "Startup.cs". The main editor area shows the "Startup.cs" code:

```
19  public class Startup
20  {
21      // ...
22      Configuration = new Configuration()
23          .AddJsonFile("config.json")
24          .AddEnvironmentVariables();
25
26      env.e
27      } // EnvironmentName EnvironmentName
28      Equals
29      GetHashCode
30      GetType
31      WebRoot
32      WebRootFileProvider
33
34  }
35
36  var mono = Type.GetType("Mono.Runtime") != null;
37  // ...
38  if (!mono)
39  {
40      services.AddEntityFramework(Configuration)
41          .AddInMemoryStore()
42          .AddDbContext<ApplicationDbContext>();
43
44  }
```

The cursor is at line 27, column 22, where "e" is highlighted. A tooltip shows the code completion suggestions: "EnvironmentName EnvironmentName", "Equals", "GetHashCode", "GetType", "WebRoot", and "WebRootFileProvider". The status bar at the bottom shows "Ln 27, Col 22 CRLF C# 😊".

Interactive Pipeline: Immediate Feedback

The screenshot shows a window with four code editors:

- func_body.sdf3**:

```
8 context-free syntax // PIE function body
9
10 PieFuncBody = Exp
11
12 context-free syntax // Expression composition
13
14 Block.Block      = <{<{Exp ";"}>}>
15 Block.EmptyBlock = <{}>
16 Exp              = <<Block>>
17
18 Exp = <(<Exp>)> {bracket, prefer} // Prefer bracket over tuple literal with
19
20 context-free syntax // Unary expressions
21
22 Expr ::= ...
```
- parse-short.pie**:

```
1 func normalize(file: path, includeDirs: path*) -> path = {
2   requires file; [requires dir with extension "sdf" | dir <- includeDirs];
3   val normFile = file.replaceExtension("norm");
4   val depFile = file.replaceExtension("dep");
5   exec(["sdf2normalized"] + "$file" + ["-I$dir" | dir <- includeDirs] +
6     "-o$normFile" + "-d$depFile");
7   [requires dep by hash | dep <- extract-deps(depFile)];
8   generates normFile; normFile
9 }
10 func extract-deps(depFile: path) -> path* = foreign class
11 func generate-table(normFiles: path*, outputFile: path) -> path = {
12   [requires file by hash | file <- normFiles];
13   exec(["sdf2table"] + ["$file" | file <- normFiles] + "-o$outputFile");
14   generates outputFile; outputFile
15 }
```
- cfg.sdf3**:

```
17
18 context-free syntax // Workspace config
19
20 Section.WorkspaceSec =
21 <workspace {
22   <{WorkspaceOpt "\n"}*>
23 }>
24 WorkspaceOpt.LangSpec = <langspec <Path>>
25 WorkspaceOpt.SpxLang = <spxlang <Path>>
26 WorkspaceOpt.SpxLangSpec = <spxlangs <Path>>
27
28 context-free syntax // Language specification config
29
30 Section.LangSpecSec =
31 <langs <Path>>
```
- test.cfg**:

```
1 workspace {
2   spxlang /Users/gohla/metaborg/repo/pipeline/cfg/langspec
3   spxlang /Users/gohla/spofax/master/repo/spofax-releng/sdf/org.metaborg.met
4   spxlang /Users/gohla/spofax/master/repo/spofax-releng/esv/org.metaborg.met
5
6   langspec minimal/langspec.cfg
7   langspec maximal/langspec.cfg
8 }
9
10 langspec {
11   identification {
12     file extensions: min, max
13   }
14   information {
15     name: minimal
16 }
```

Interactive Pipeline: Immediate Feedback

```
func_body.sdf3
8 context-free syntax // PIE function body
9
10 PieFuncBody = Exp
11
12 context-free syntax // Expression composition
13
14 Block.Block      = <{{Exp ";"}}+>
15 Block.EmptyBlock = <{}>
16 Exp              = <<Block>>
17
18 Exp = <(<Exp>)> {bracket, prefer} // Prefer bracket over tuple literal with
19
20 context-free syntax // Unary expressions
21
22 Expr ::= ... - --Expr2_Signed

parse-short.pie
1 func normalize(file: path, includeDirs: path*) -> path = {
2   requires file; [requires dir with extension "sdf" | dir <- includeDirs];
3   val normFile = file.replaceExtension("norm");
4   val depFile = file.replaceExtension("dep");
5   exec(["sdf2normalized"] + "$file" + ["-I$dir" | dir <- includeDirs] +
6     "-o$normFile" + "-d$depFile");
7   [requires dep by hash | dep <- extract-deps(depFile)];
8   generates normFile; normFile
9 }
10 func extract-deps(depFile: path) -> path* = foreign class
11 func generate-table(normFiles: path*, outputFile: path) -> path = {
12   [requires file by hash | file <- normFiles];
13   exec(["sdf2table"] + ["$file" | file <- normFiles] + "-o$outputFile");
14   generates outputFile; outputFile
15 }

cfg.sdf3
17
18 context-free syntax // Workspace config
19
20 Section.WorkspaceSec =
21 <workspace {
22   <{WorkspaceOpt "\n"}*>
23 }>
24 WorkspaceOpt.LangSpec = <langspec <Path>>
25 WorkspaceOpt.SpxLang = <spxlang <Path>>
26 WorkspaceOpt.SpxLangSpec = <spxlangs <Path>>
27
28 context-free syntax // Language specification config
29
30 Section.LangSpecSec =
31 <langs <Path>>

test.cfg
1 workspace {
2   spxlang /Users/gohla/metaborg/repo/pipeline/cfg/langspec
3   spxlang /Users/gohla/spofax/master/repo/spofax-releng/sdf/org.metaborg.met
4   spxlang /Users/gohla/spofax/master/repo/spofax-releng/esv/org.metaborg.met
5
6   langspec minimal/langspec.cfg
7   langspec maximal/langspec.cfg
8 }
9
10 langspec {
11   identification {
12     file extensions: min, max
13   }
14   information {
15     name: minimal
16 }
```

Incrementalization: only recompute what has been affected by a change

Developing interactive pipelines is

Complicated

Error Prone

High Development Cost

when programmers *manually reason*
about incrementality

Requirements

A *system* for
developing
interactive pipelines,
that are:

Incremental

Correct

Persistent

Expressive

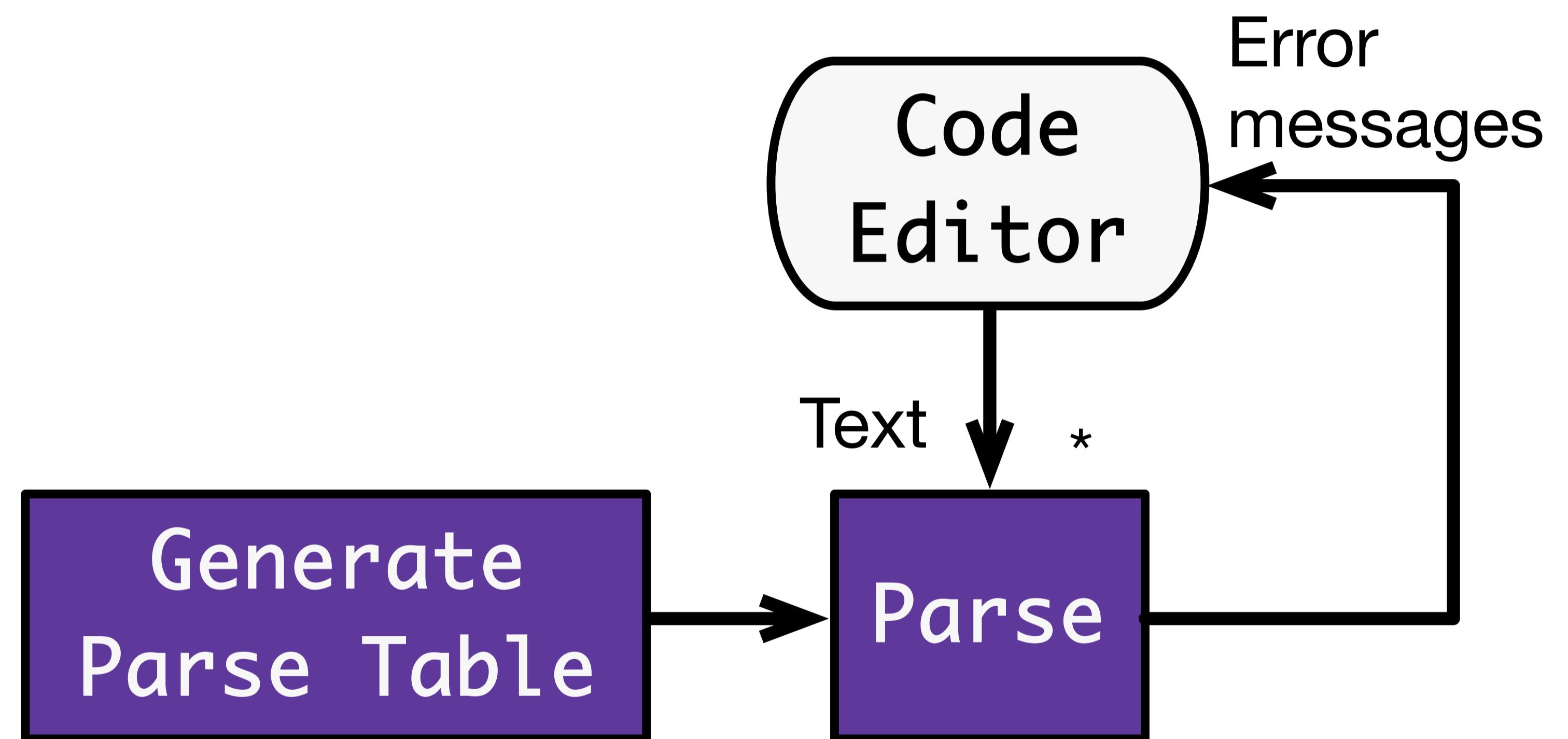
Easy to develop

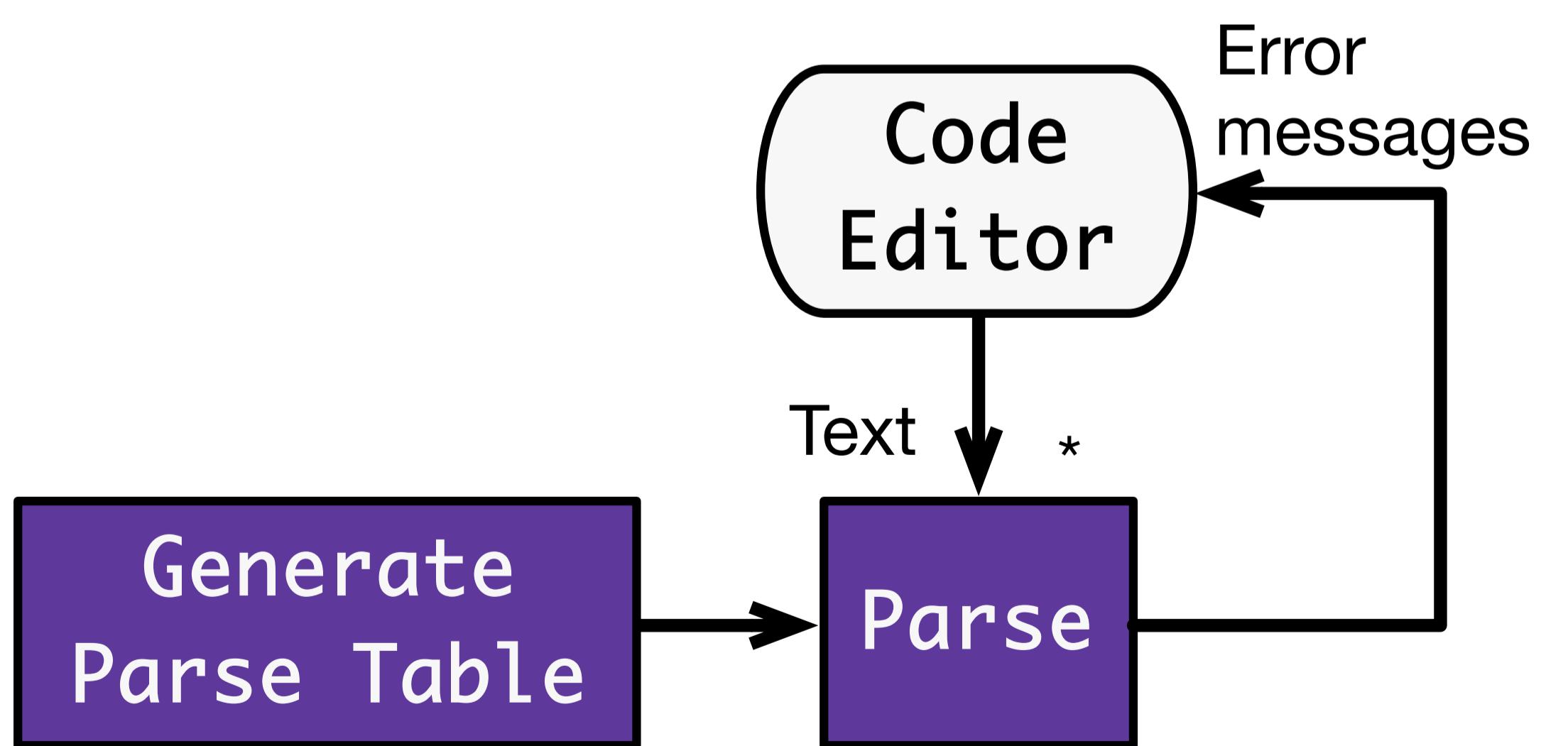
PIE: Pipelines for Interactive Environments

DSL

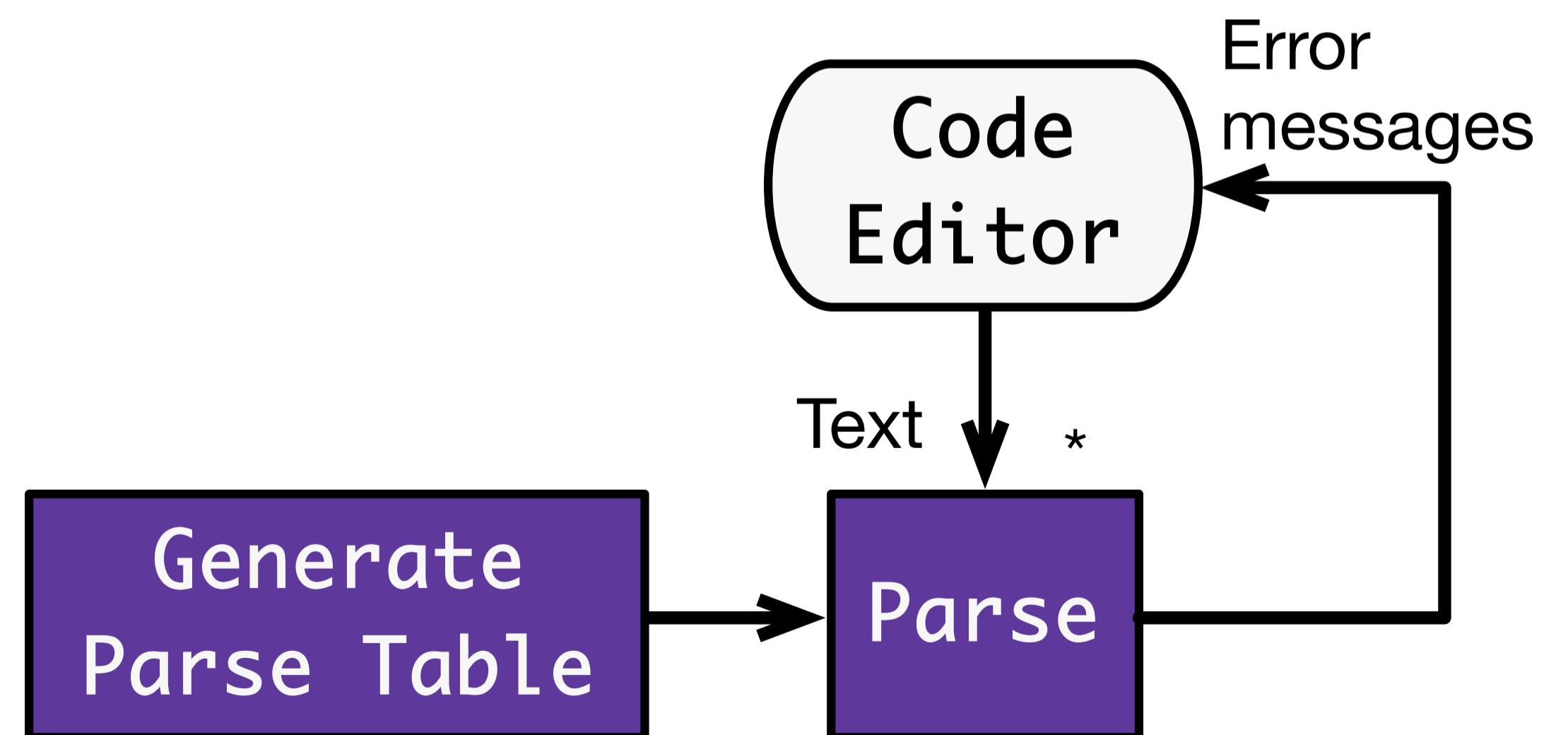
API

Runtime





PIE DSL code

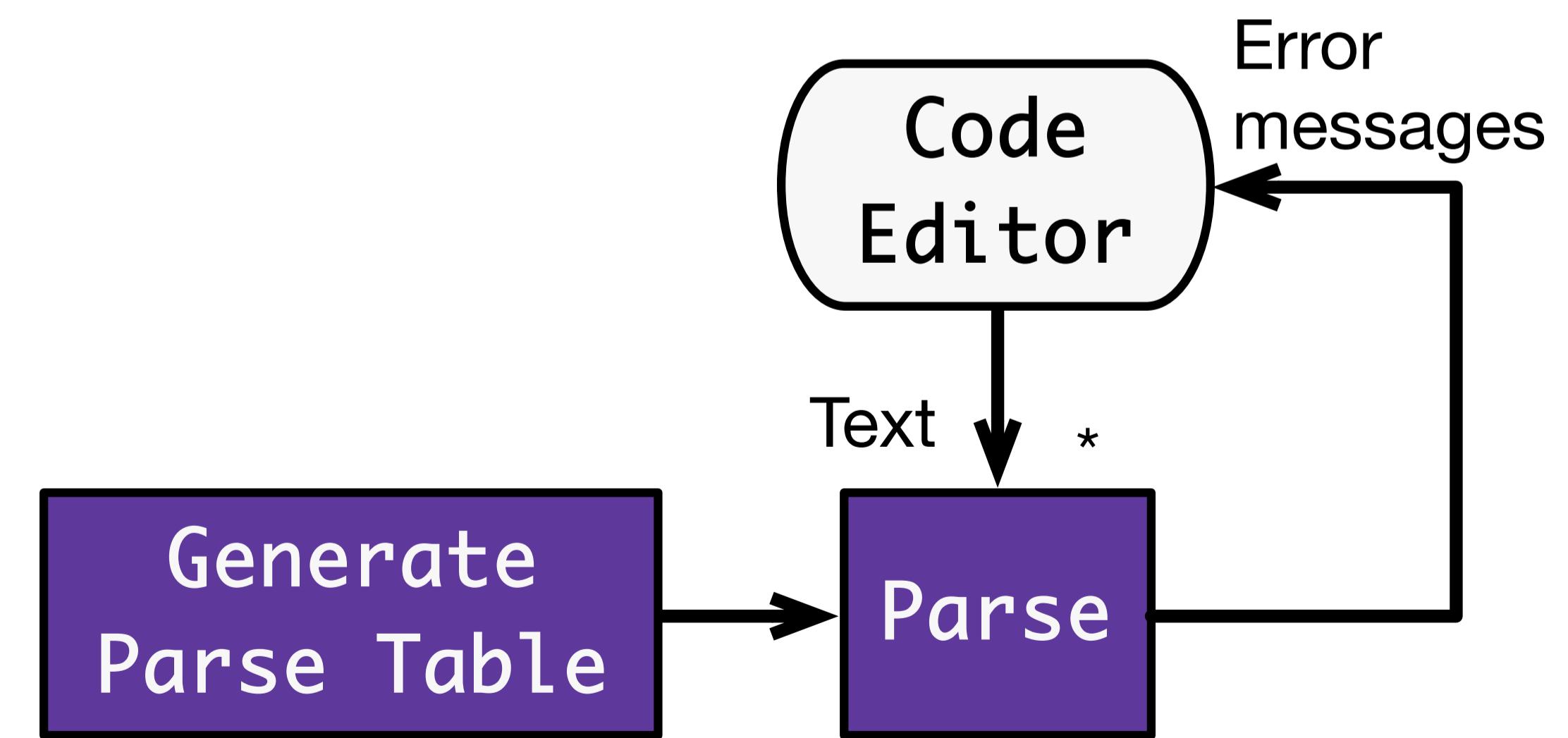


PIE DSL code

```
func update-editor(text: string) -> Msg*
```

```
func compile-parse-table(defFile: path) -> path
```

```
func parse(text: string, pt: path) -> (Ast, Msg*)
```

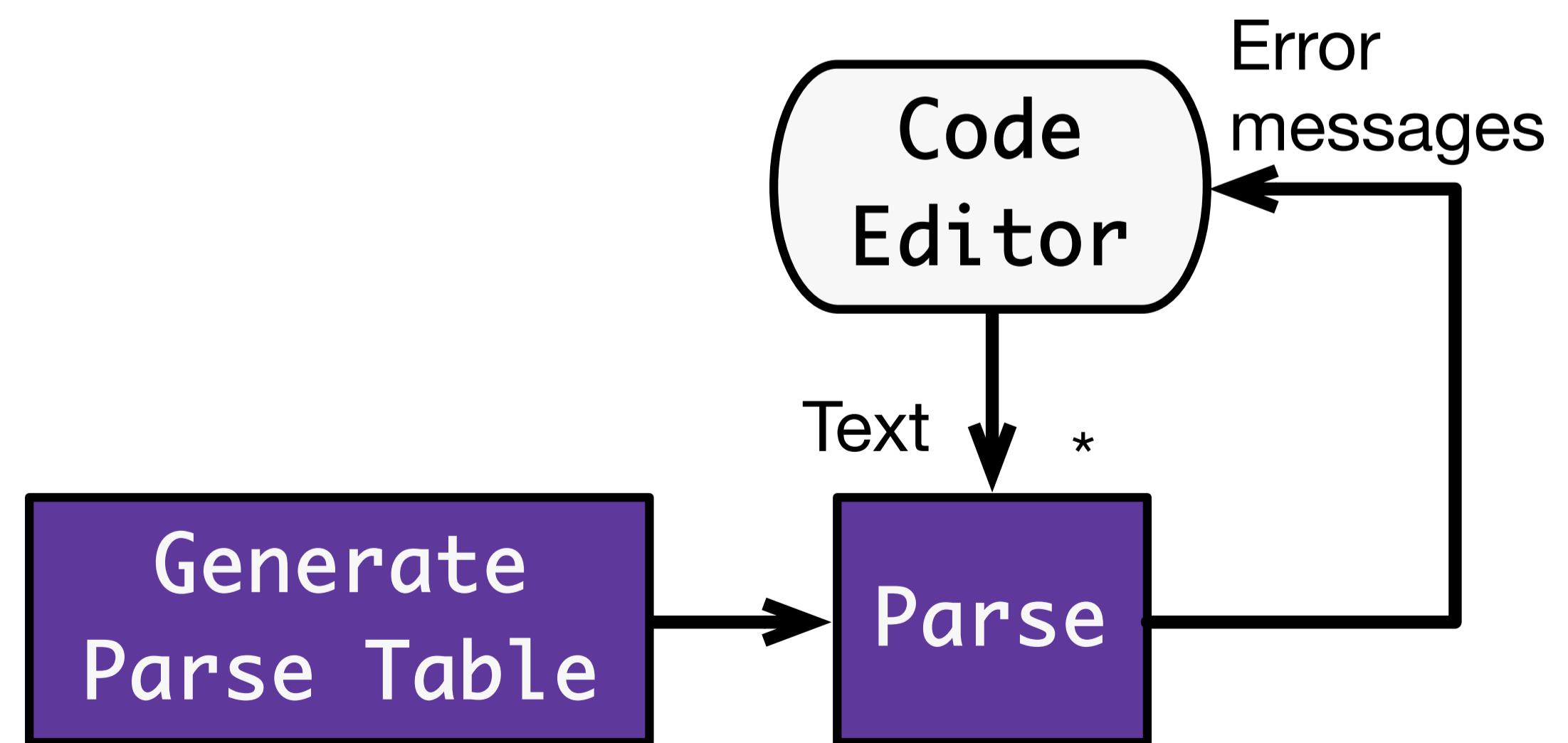


PIE DSL code

```
func update-editor(text: string) -> Msg* = {
    val syntaxDef = ./Java8.sdf;
}

func compile-parse-table(defFile: path) -> path

func parse(text: string, pt: path) -> (Ast, Msg*)
```



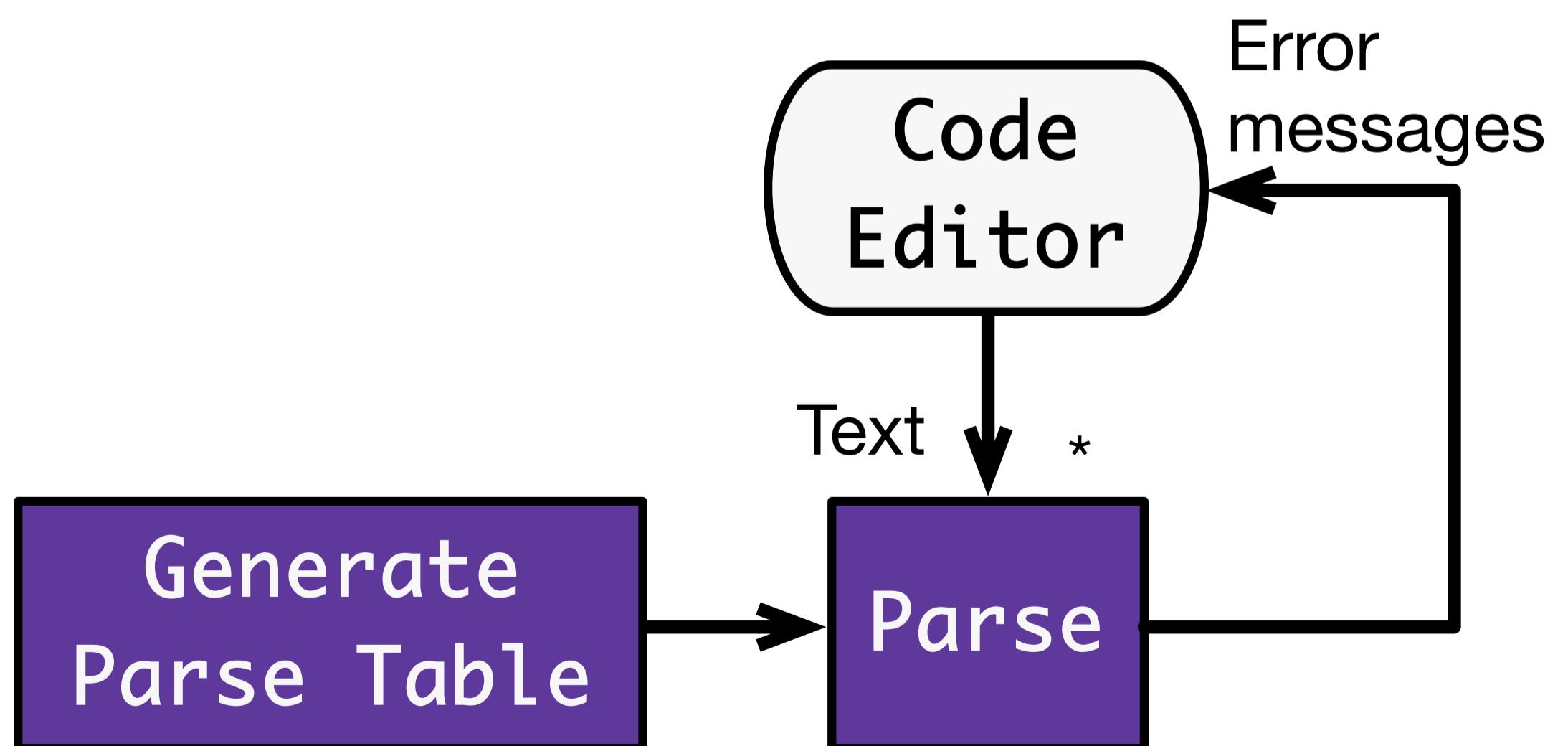
PIE DSL code

```
func update-editor(text: string) -> Msg* = {
    val syntaxDef = ./Java8.sdf;
    val parseTable = compile-parse-table(syntaxDef);

}

func compile-parse-table(defFile: path) -> path

func parse(text: string, pt: path) -> (Ast, Msg*)
```

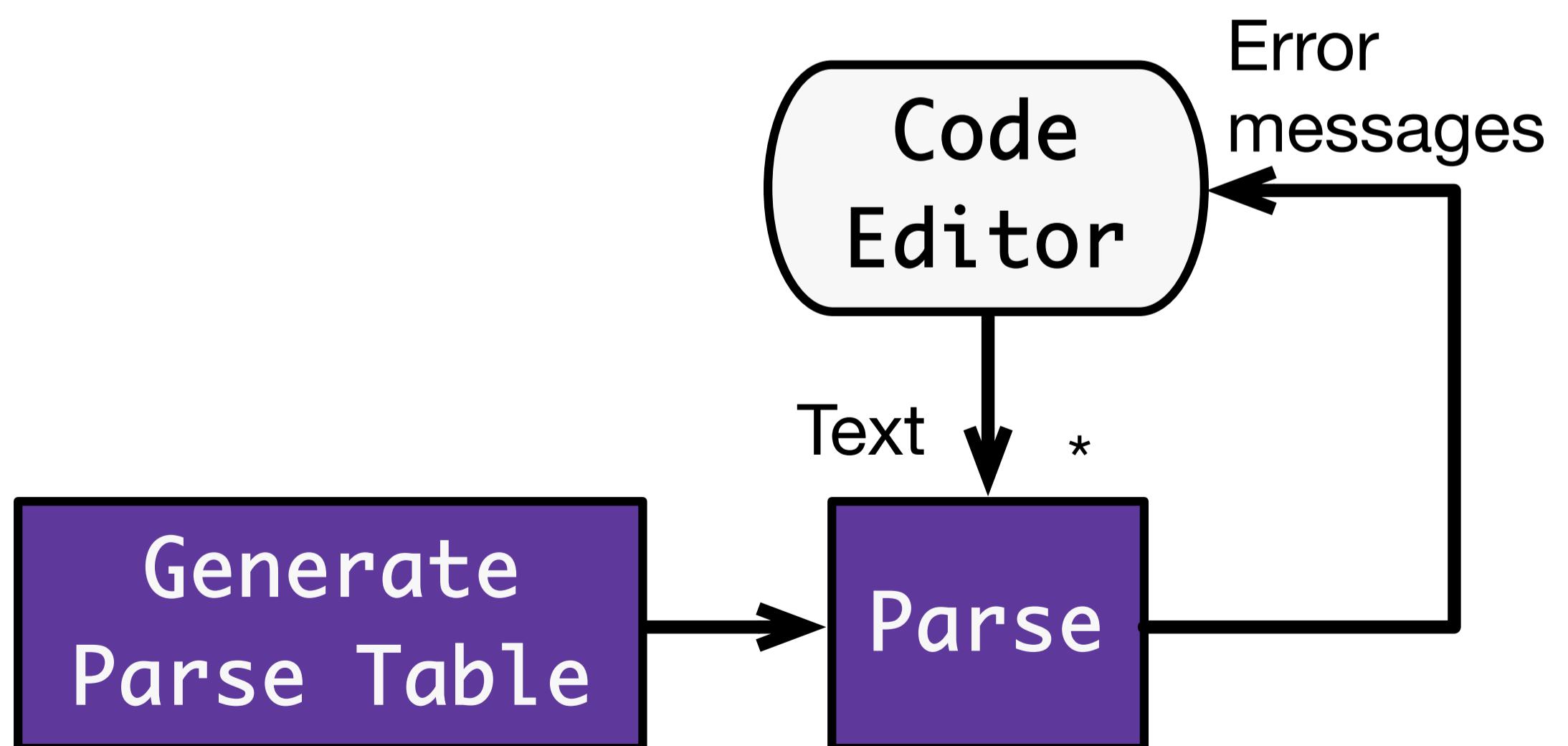


PIE DSL code

```
func update-editor(text: string) -> Msg* = {
    val syntaxDef = ./Java8.sdf;
    val parseTable = compile-parse-table(syntaxDef);
    val (_, messages) = parse(text, parseTable);
    messages
}

func compile-parse-table(defFile: path) -> path

func parse(text: string, pt: path) -> (Ast, Msg*)
```

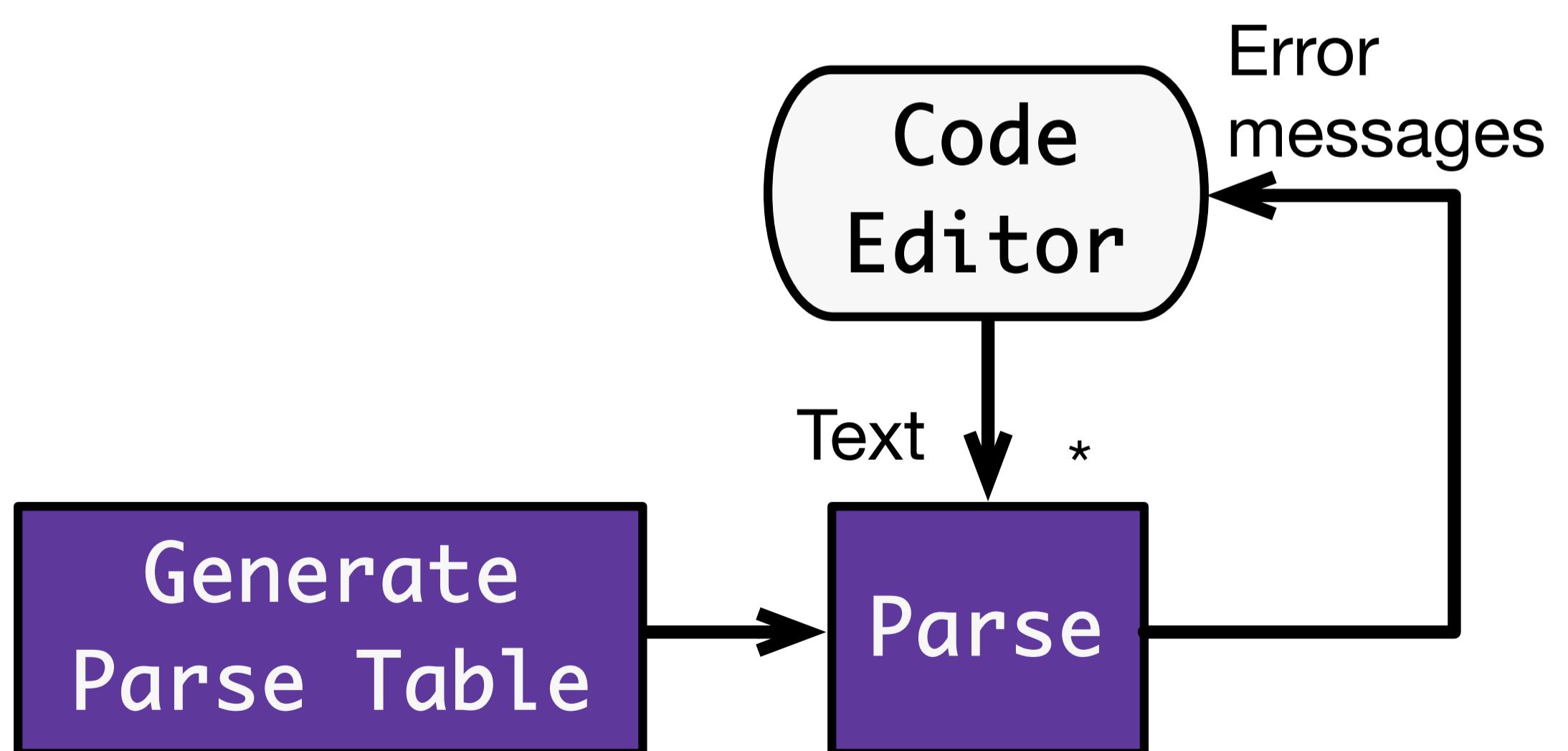


PIE DSL code

```
func update-editor(text: string) -> Msg* = {
    val syntaxDef = ./Java8.sdf;
    val parseTable = compile-parse-table(syntaxDef);
    val (_, messages) = parse(text, parseTable);
    messages
}

func compile-parse-table(defFile: path) -> path = {
    exec(["sdf2table"] + "$defFile" + "-o$tblFile"
        + "-d$depFile");
}

func parse(text: string, pt: path) -> (Ast, Msg*)
```



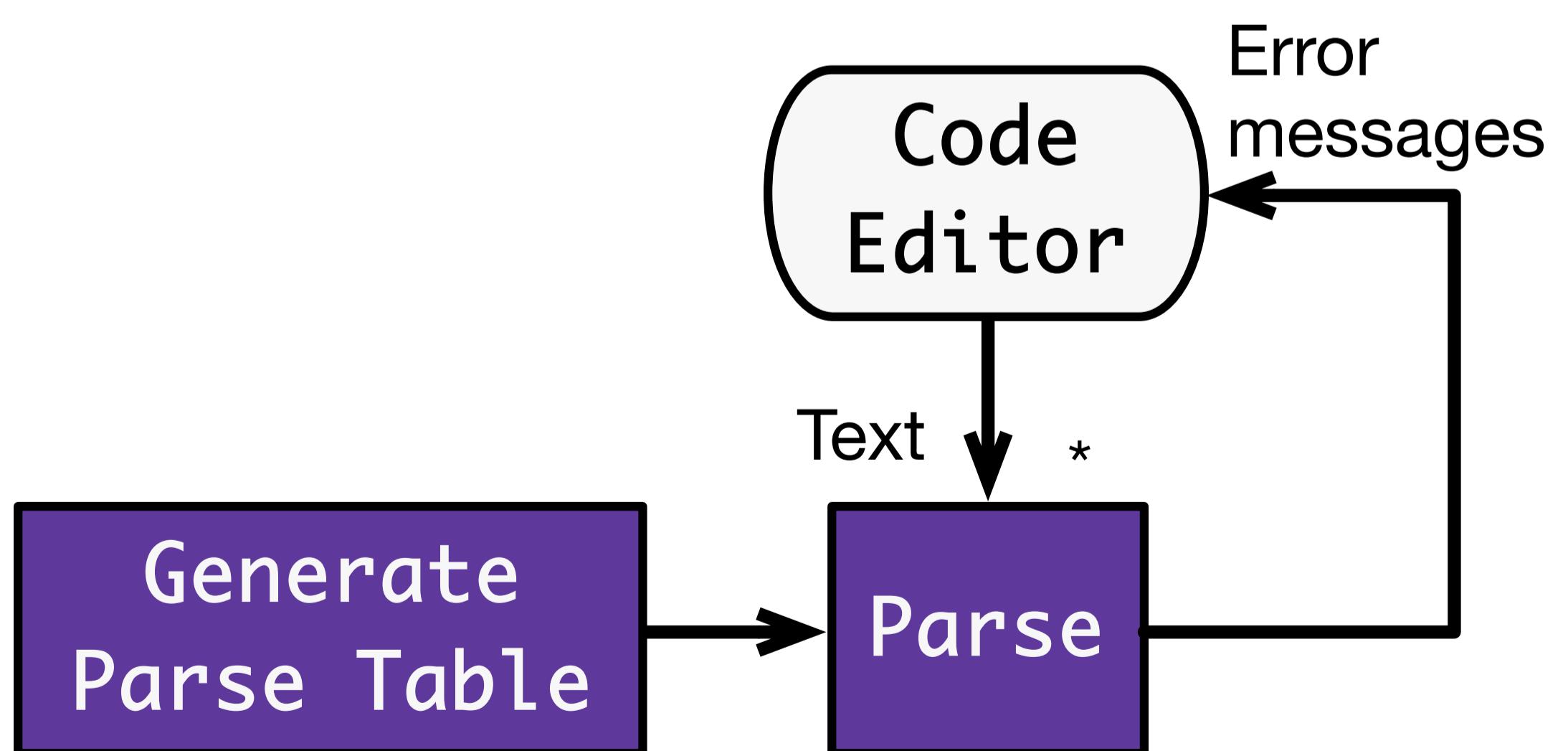
PIE DSL code

```
func update-editor(text: string) -> Msg* = {
    val syntaxDef = ./Java8.sdf;
    val parseTable = compile-parse-table(syntaxDef);
    val (_, messages) = parse(text, parseTable);
    messages
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;

    exec(["sdf2table"] + "$defFile" + "-o$tblFile"
        + "-d$depFile");
}

func parse(text: string, pt: path) -> (Ast, Msg*)
```

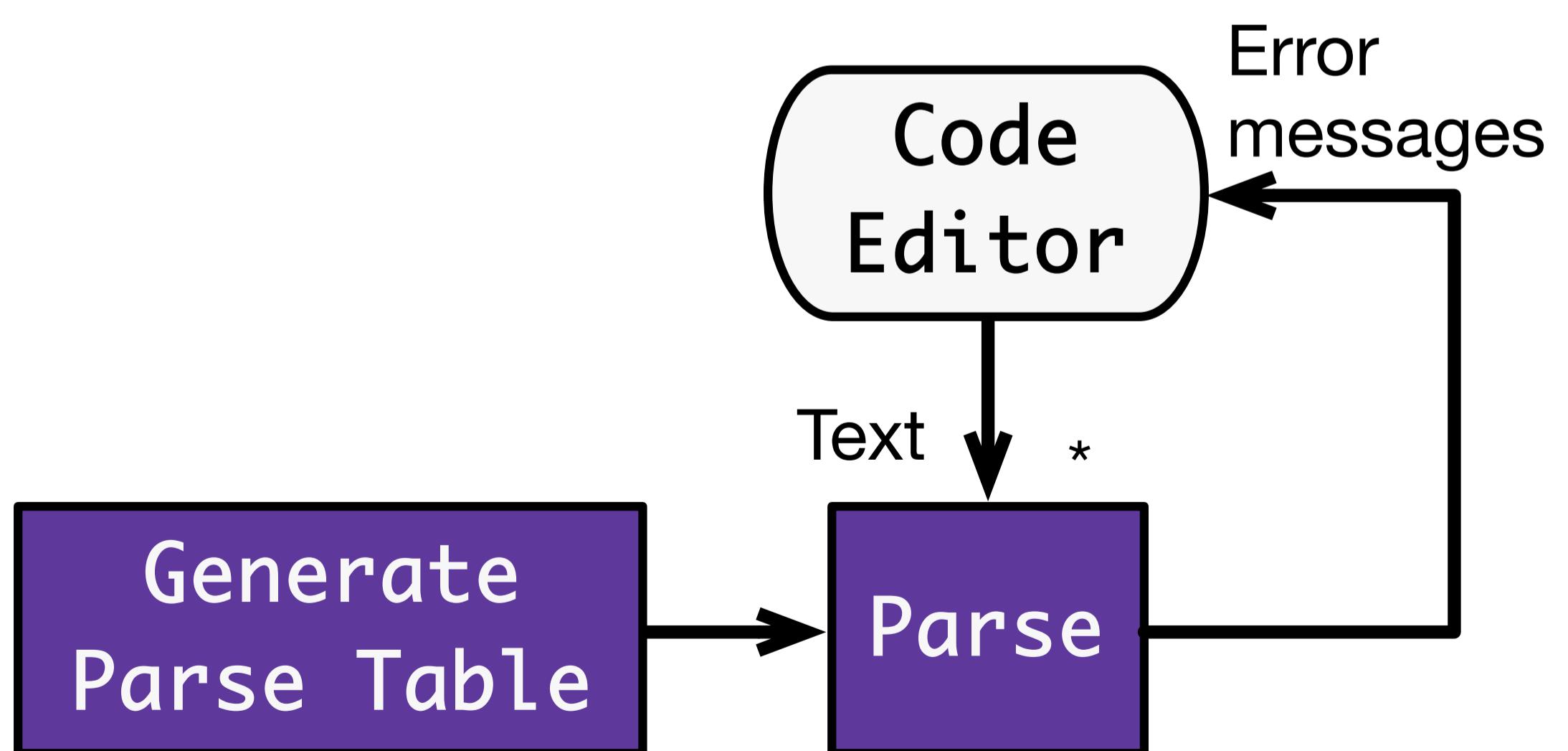


PIE DSL code

```
func update-editor(text: string) -> Msg* = {
    val syntaxDef = ./Java8.sdf;
    val parseTable = compile-parse-table(syntaxDef);
    val (_, messages) = parse(text, parseTable);
    messages
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    val tblFile = defFile.replaceExtension("tbl");
    val depFile = defFile.replaceExtension("dep");
    exec(["sdf2table"] + "$defFile" + "-o$tblFile"
        + "-d$depFile");
}

func parse(text: string, pt: path) -> (Ast, Msg*)
```

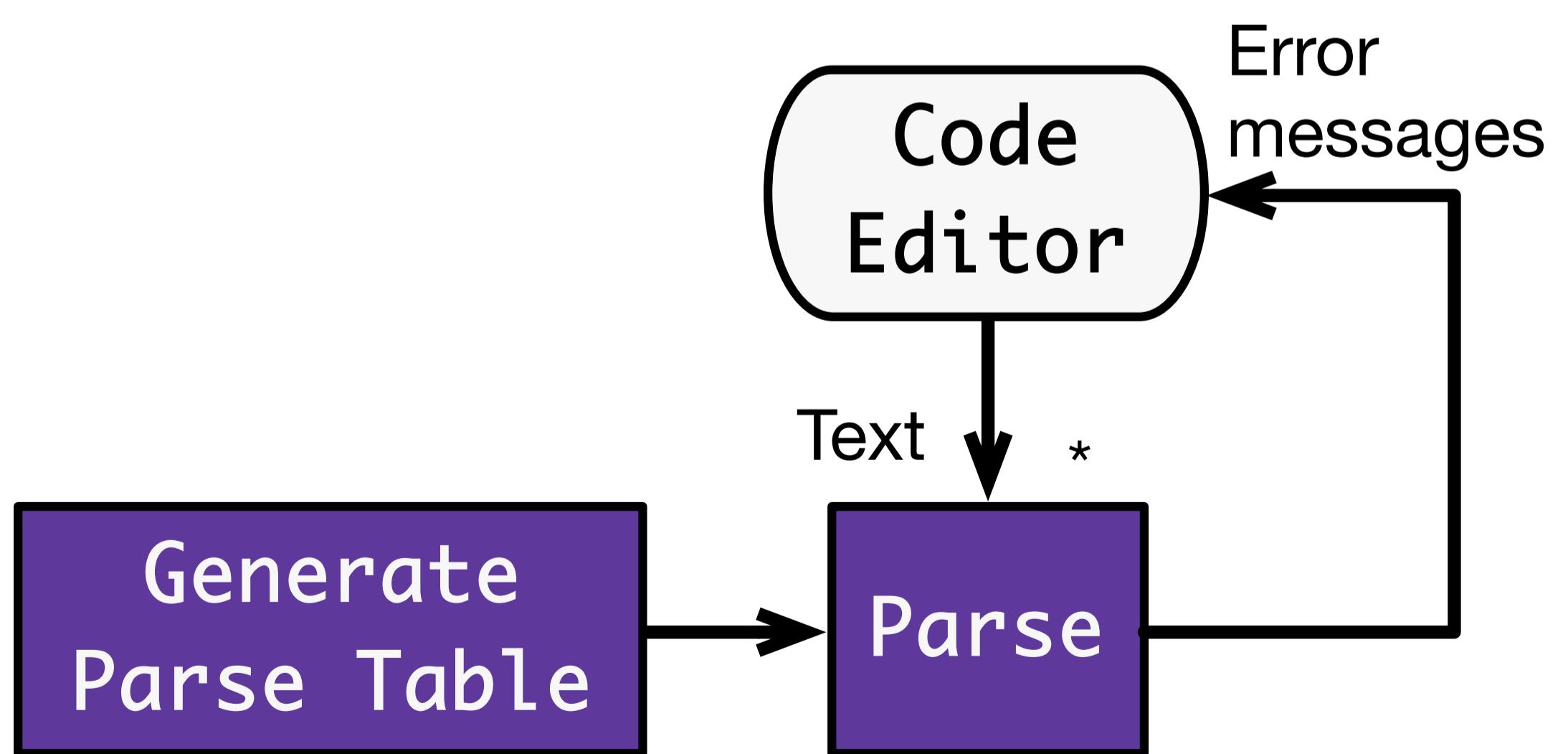


PIE DSL code

```
func update-editor(text: string) -> Msg* = {
    val syntaxDef = ./Java8.sdf;
    val parseTable = compile-parse-table(syntaxDef);
    val (_, messages) = parse(text, parseTable);
    messages
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    val tblFile = defFile.replaceExtension("tbl");
    val depFile = defFile.replaceExtension("dep");
    exec(["sdf2table"] + "$defFile" + "-o$tblFile"
        + "-d$depFile");
    [requires dep | dep <- extract-deps(depFile)];
}

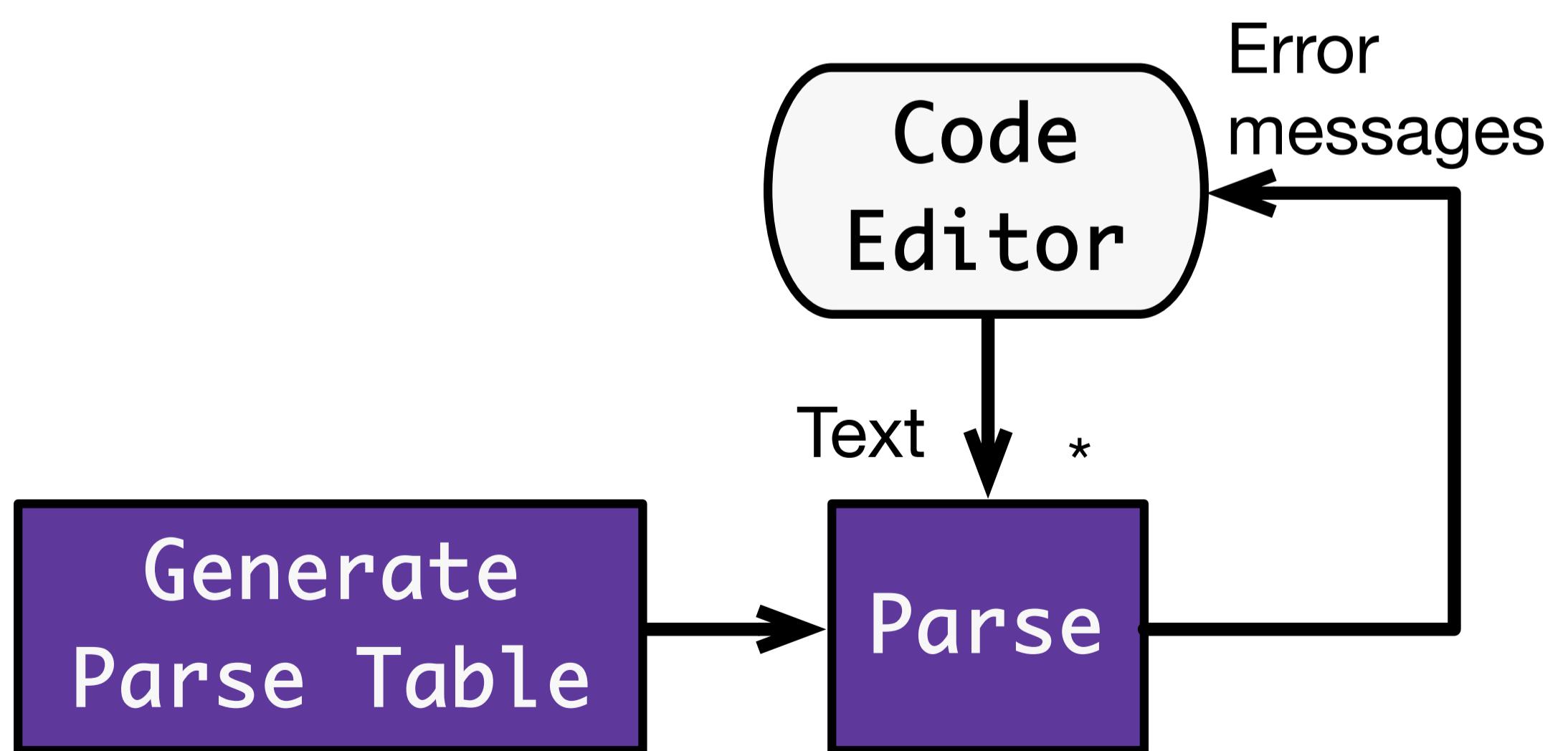
func extract-deps(depFile: path) -> path*
func parse(text: string, pt: path) -> (Ast, Msg*)
```



PIE DSL code

```
func update-editor(text: string) -> Msg* = {
    val syntaxDef = ./Java8.sdf;
    val parseTable = compile-parse-table(syntaxDef);
    val (_, messages) = parse(text, parseTable);
    messages
}

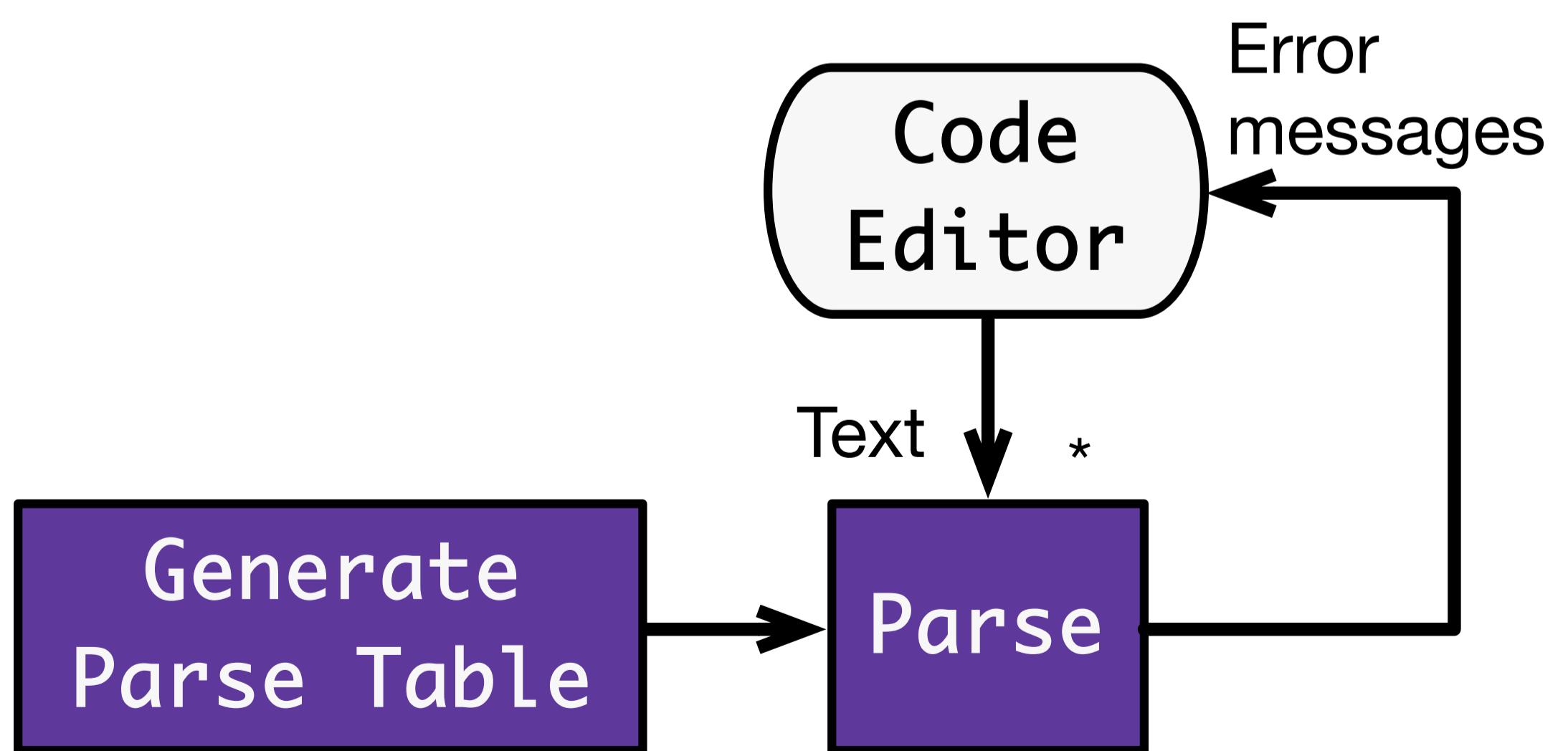
func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    val tblFile = defFile.replaceExtension("tbl");
    val depFile = defFile.replaceExtension("dep");
    exec(["sdf2table"] + "$defFile" + "-o$tblFile"
        + "-d$depFile");
    [requires dep | dep <- extract-deps(depFile)];
    generates tblFile;
    tblFile
}
func extract-deps(depFile: path) -> path*
func parse(text: string, pt: path) -> (Ast, Msg*)
```



PIE DSL code

```
func update-editor(text: string) -> Msg* = {
    val syntaxDef = ./Java8.sdf;
    val parseTable = compile-parse-table(syntaxDef);
    val (_, messages) = parse(text, parseTable);
    messages
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    val tblFile = defFile.replaceExtension("tbl");
    val depFile = defFile.replaceExtension("dep");
    exec(["sdf2table"] + "$defFile" + "-o$tblFile"
        + "-d$depFile");
    [requires dep | dep <- extract-deps(depFile)];
    generates tblFile;
    tblFile
}
func extract-deps(depFile: path) -> path*
    = foreign jvm spoofax.sdf#extractDeps
func parse(text: string, pt: path) -> (Ast, Msg*)
    = foreign jvm spoofax.sdf.Parse
```



PIE DSL code

```
func update-editor(text: string) -> Msgs {
    val syntaxDef = ./Java8.sdf;
    val parseTable = compile-parse-table();
    val (_, messages) = parse(text, parseTable);
    messages
}

func compile-parse-table(defFile: path) {
    requires defFile;
    val tblFile = defFile.replaceExtension("tbl");
    val depFile = defFile.replaceExtension("dep");
    exec(["sdf2table"] + "$defFile" + "-o$tblFile"
        + "-d$depFile");
    [requires dep | dep <- extract-deps(depFile)];
    generates tblFile;
    tblFile
}

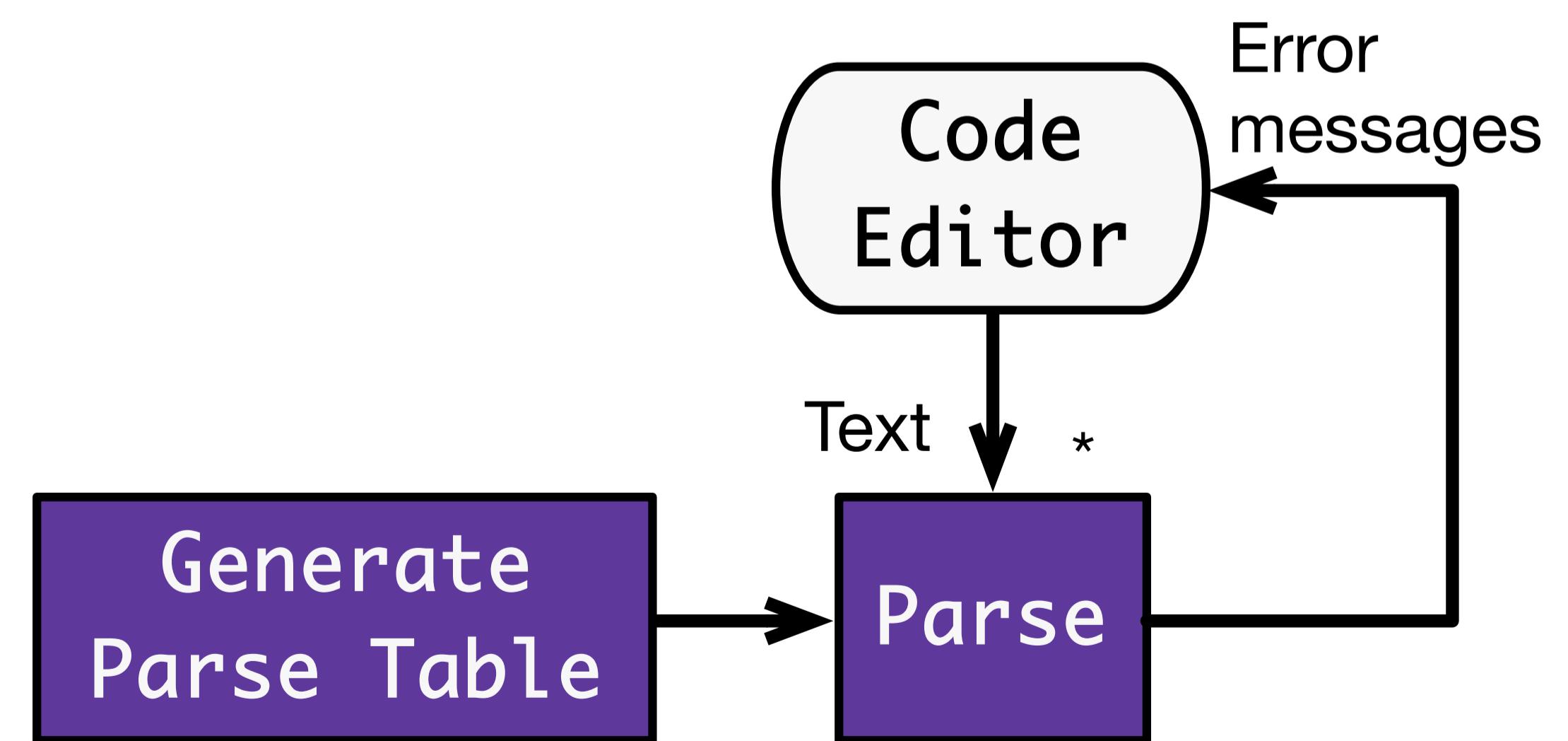
func extract-deps(depFile: path) -> path*
= foreign jvm spoofax.sdf#extractDeps

func parse(text: string, pt: path) -> (Ast, Msg*)
= foreign jvm spoofax.sdf.Parse
```

Foreign JVM code

```
fun extractDeps(depFile: PPath) = Api.extractSdfDeps(depFile)

class Parse : Func<Parse.Input, Parse.Output> {
    data class Input(val text: String, val table: PPath)
    data class Output(val ast: Ast, val msgs: List<Msg>)
    override fun ExecContext.exec(input: Input): Output {
        require(input.table)
        val result = Api.parse(input.table, input.text)
        return Output(result.ast, result.msgs)
    }
}
```



Execution

```
func update-editor(text: string) -> Msg * =  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path =  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tbIFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg *)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

Execute and build dependency graph

```
func update-editor(text: string) -> Msg * =  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path =  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tbIFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg *)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

Execute and build dependency graph

Legend

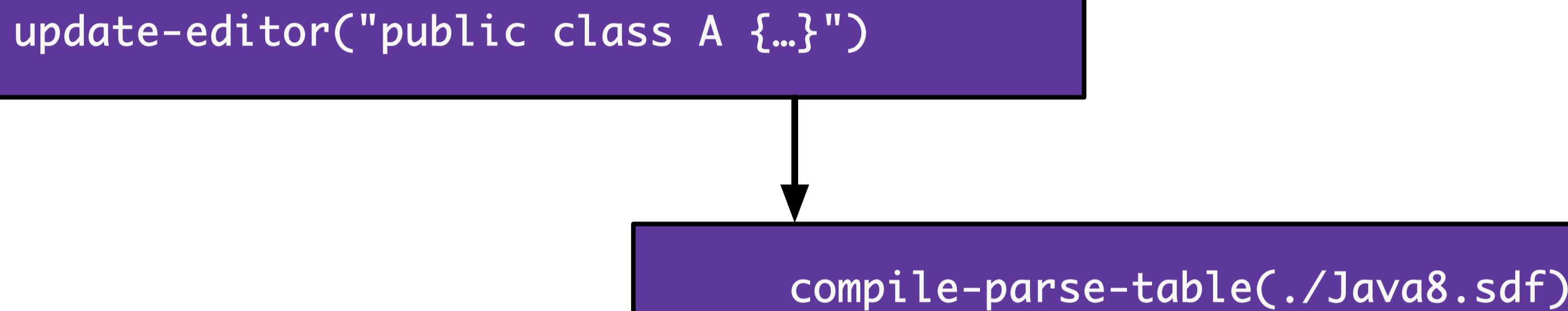
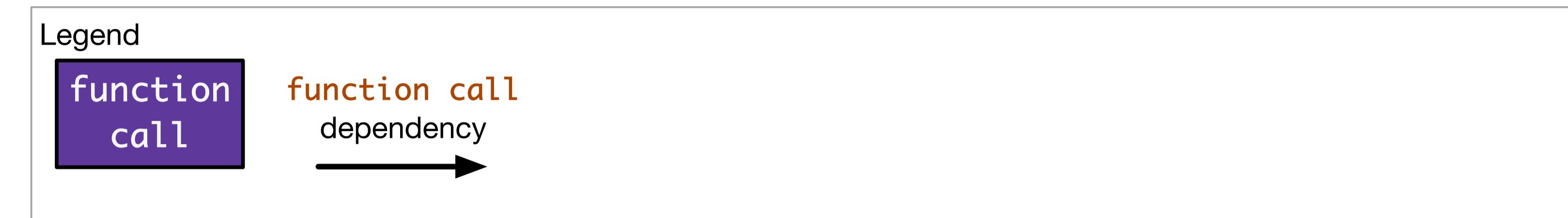
function
call

update-editor("public class A {...}")

```
func update-editor(text: string) -> Msg * =  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path =  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tbFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

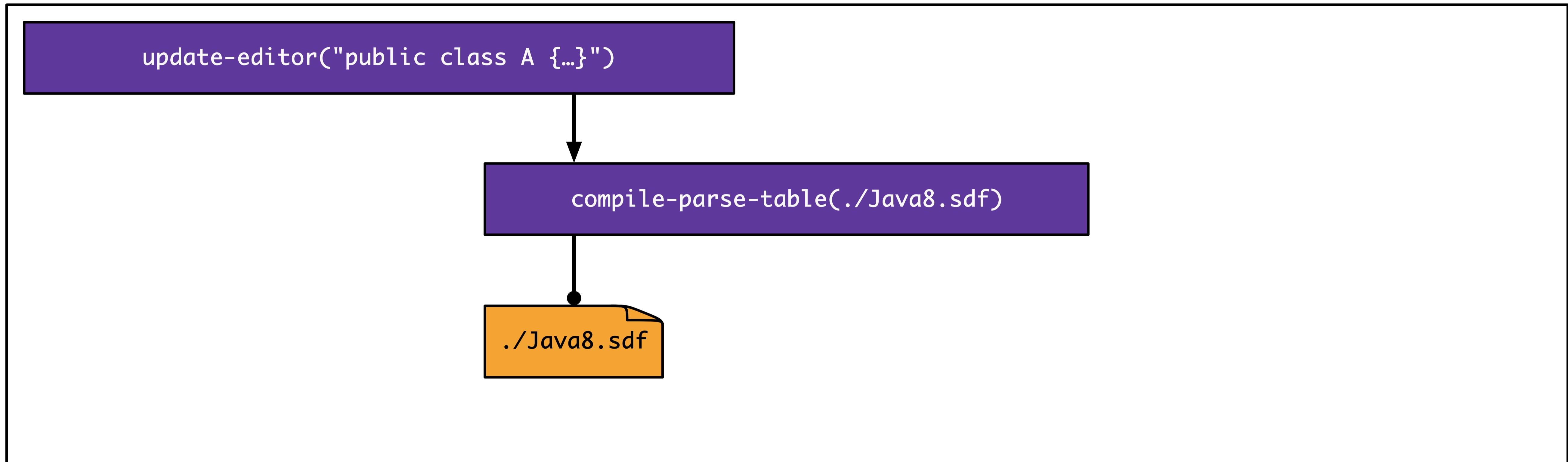
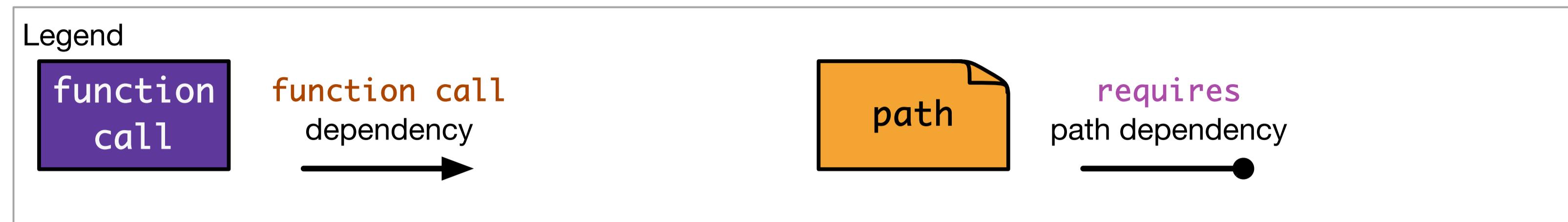
Execute and build dependency graph



```
func update-editor(text: string) -> Msg * =  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path =  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

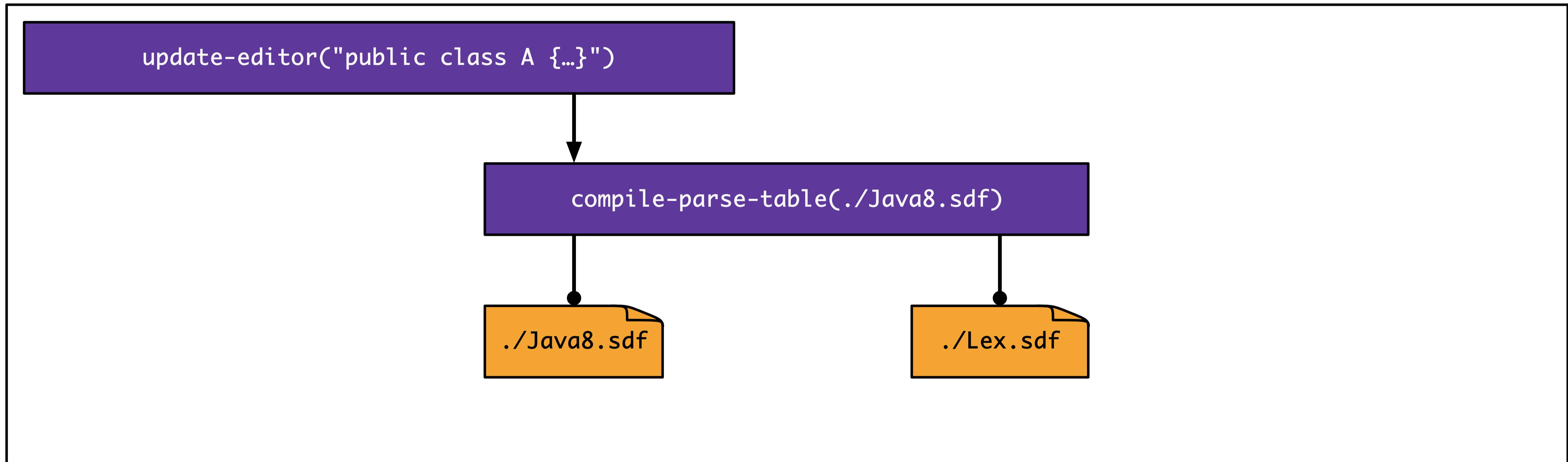
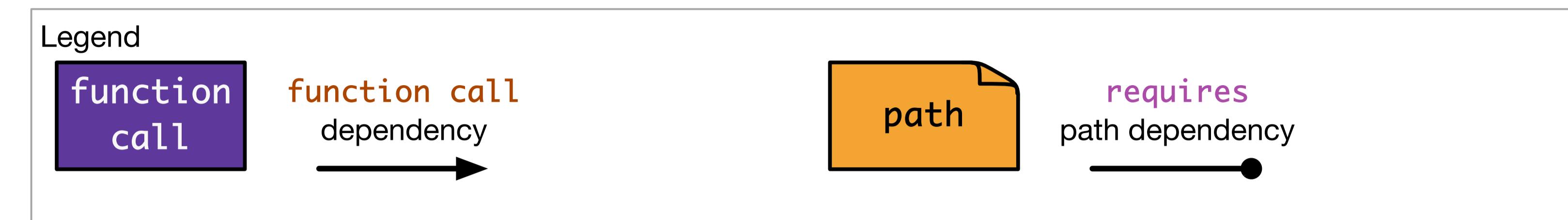
Execute and build dependency graph



```
func update-editor(text: string) -> Msg * =  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path =  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

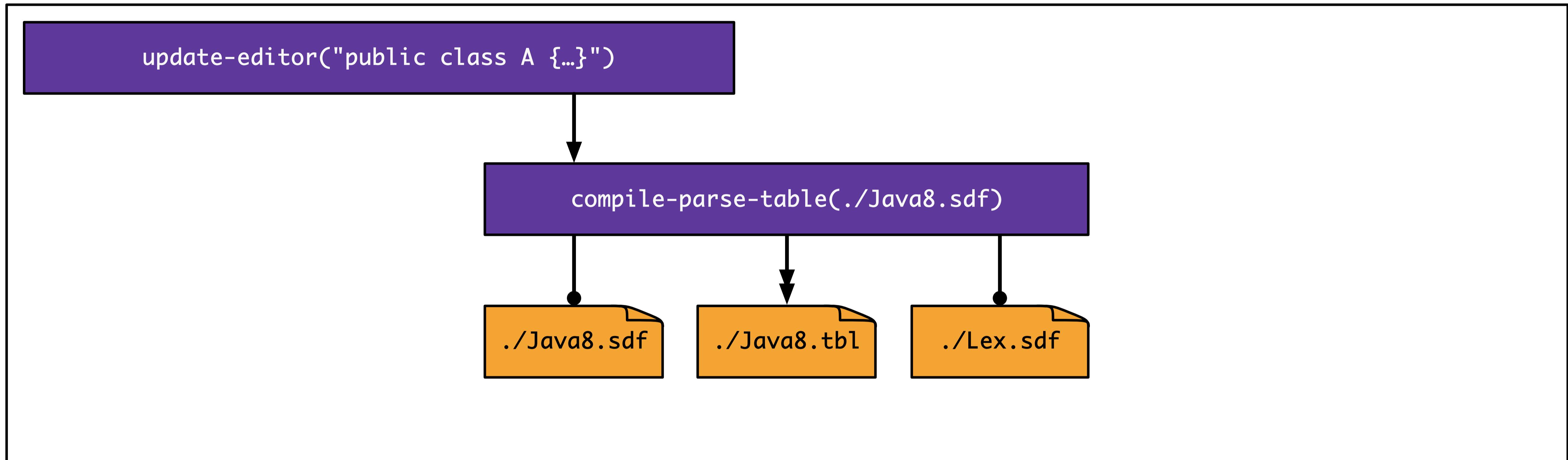
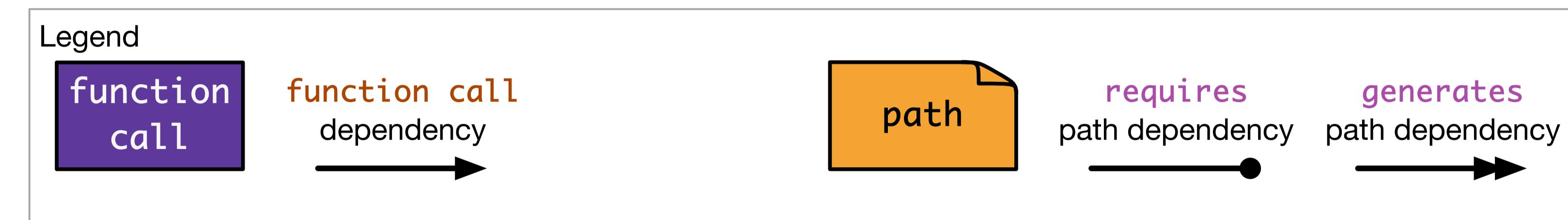
Execute and build dependency graph



```
func update-editor(text: string) -> Msg * =  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path =  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tbFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

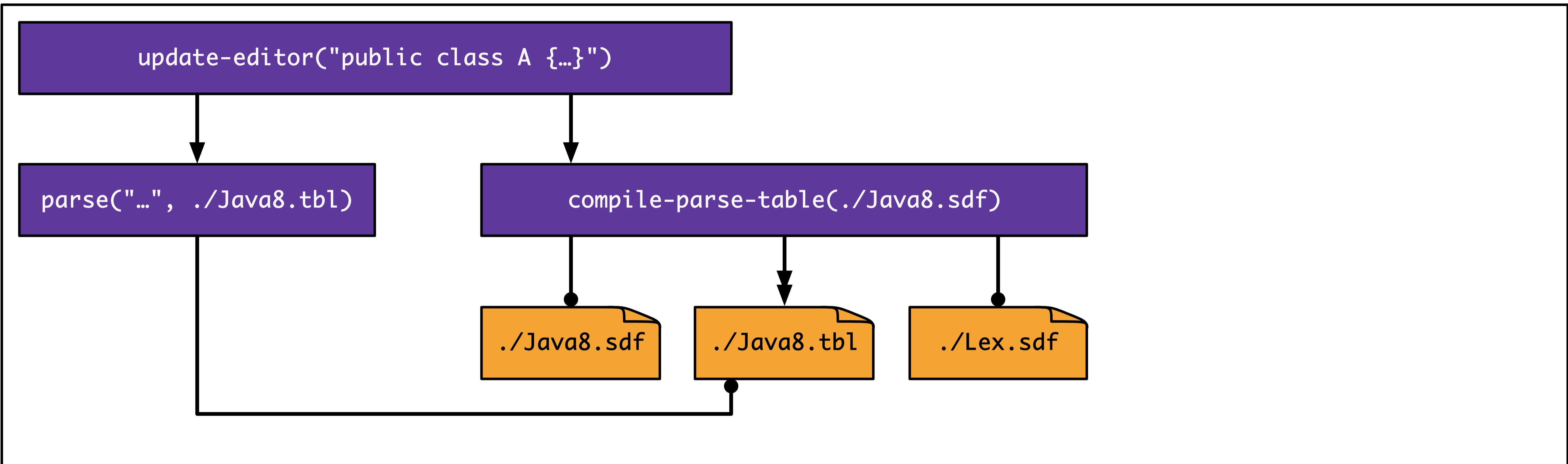
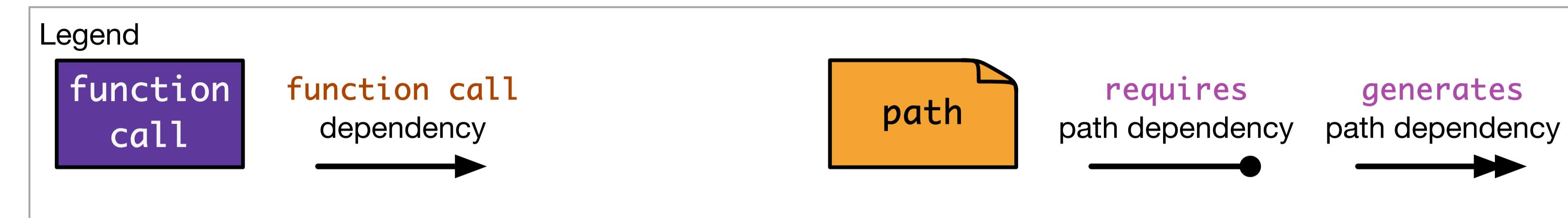
Execute and build dependency graph



```
func update-editor(text: string) -> Msg *={  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

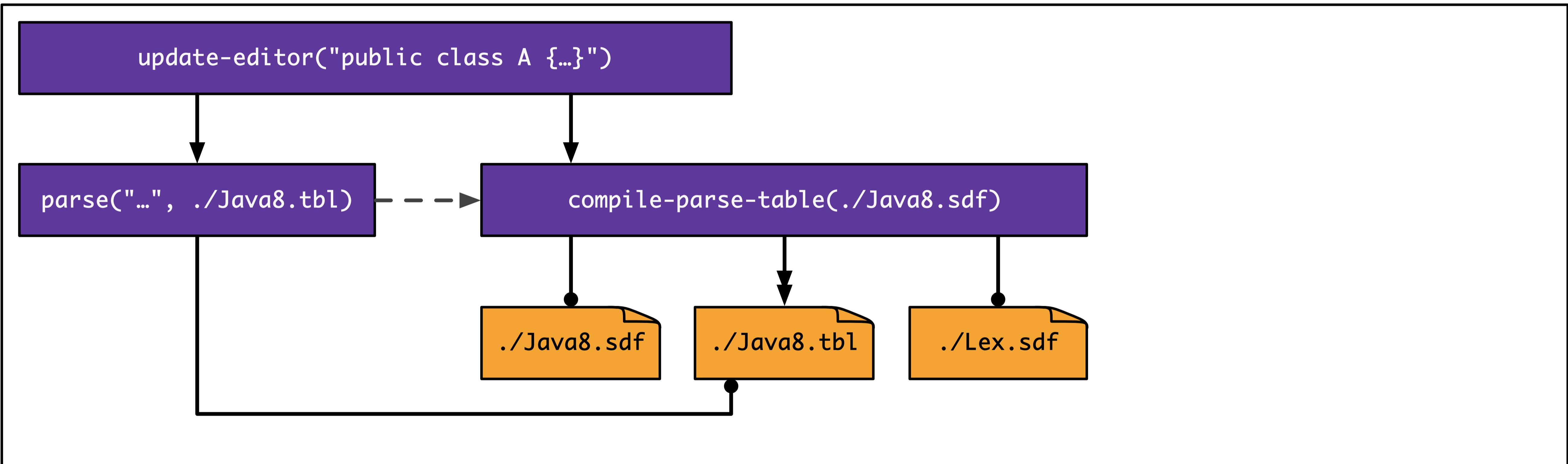
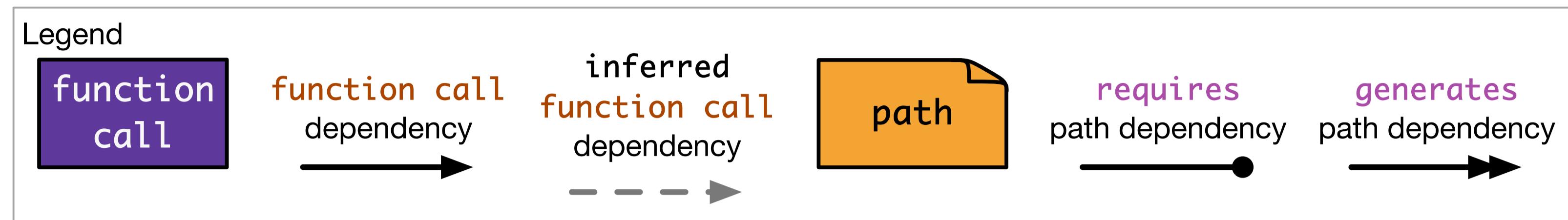
Execute and build dependency graph



```
func update-editor(text: string) -> Msg *={  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

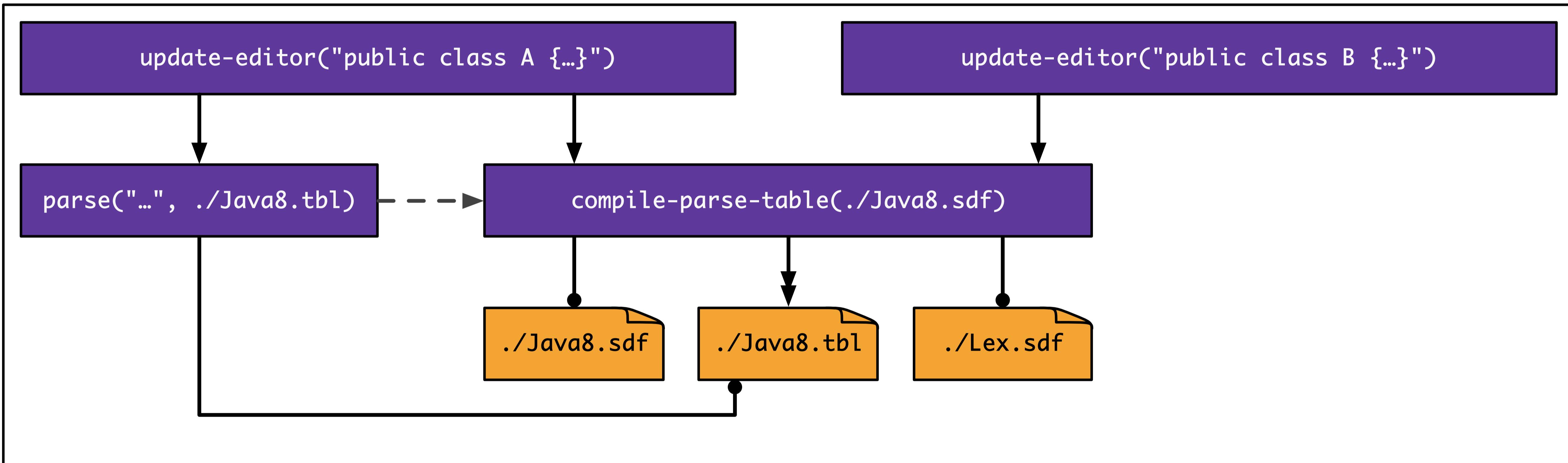
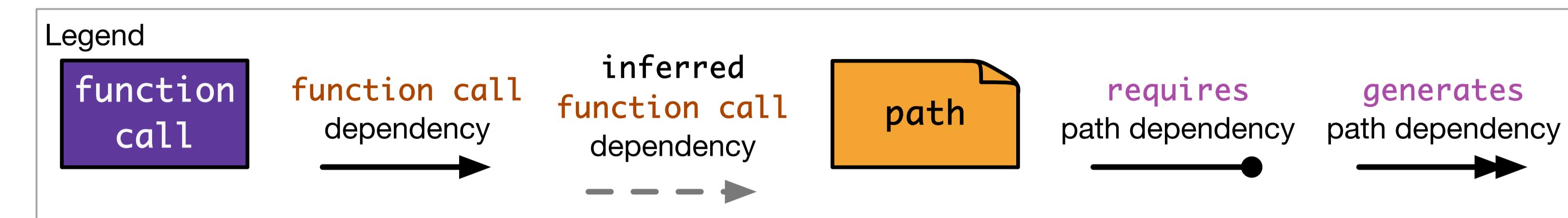
Execute and build dependency graph



```
func update-editor(text: string) -> Msg *={  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

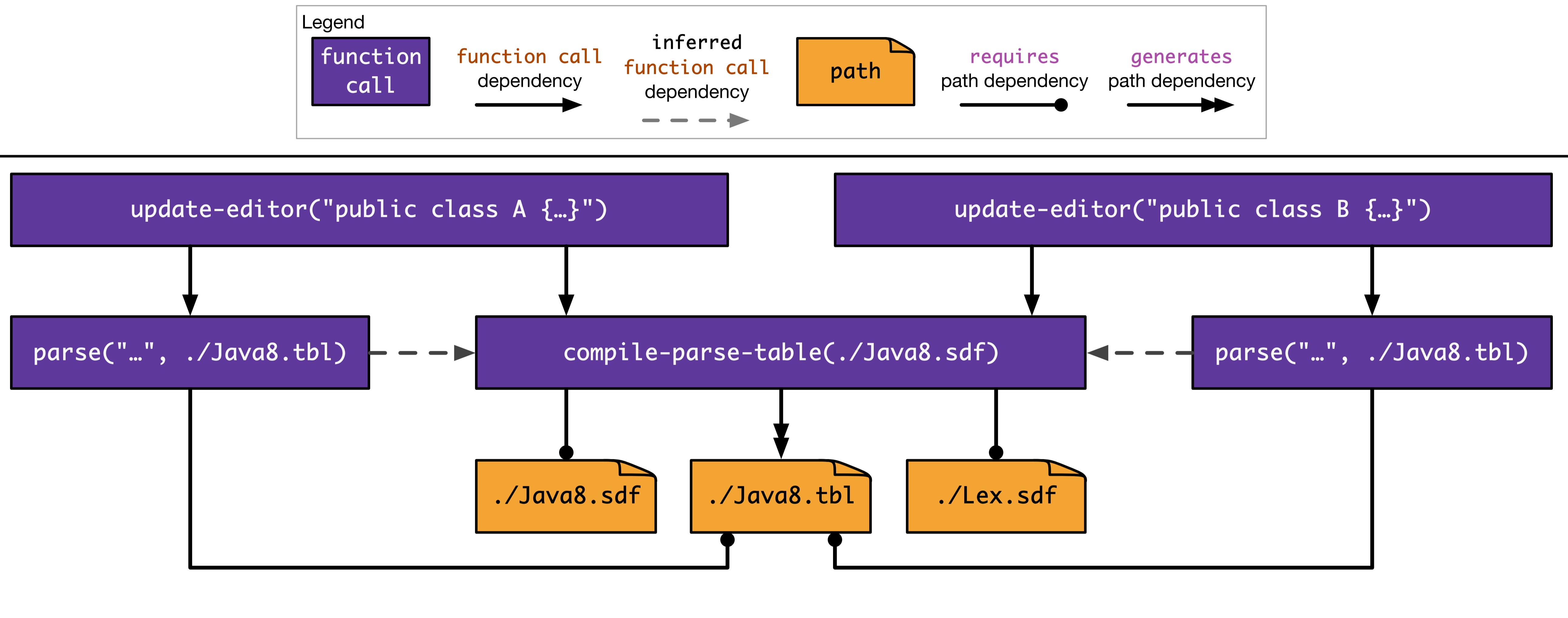
Execute and build dependency graph



```
func update-editor(text: string) -> Msg* =  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

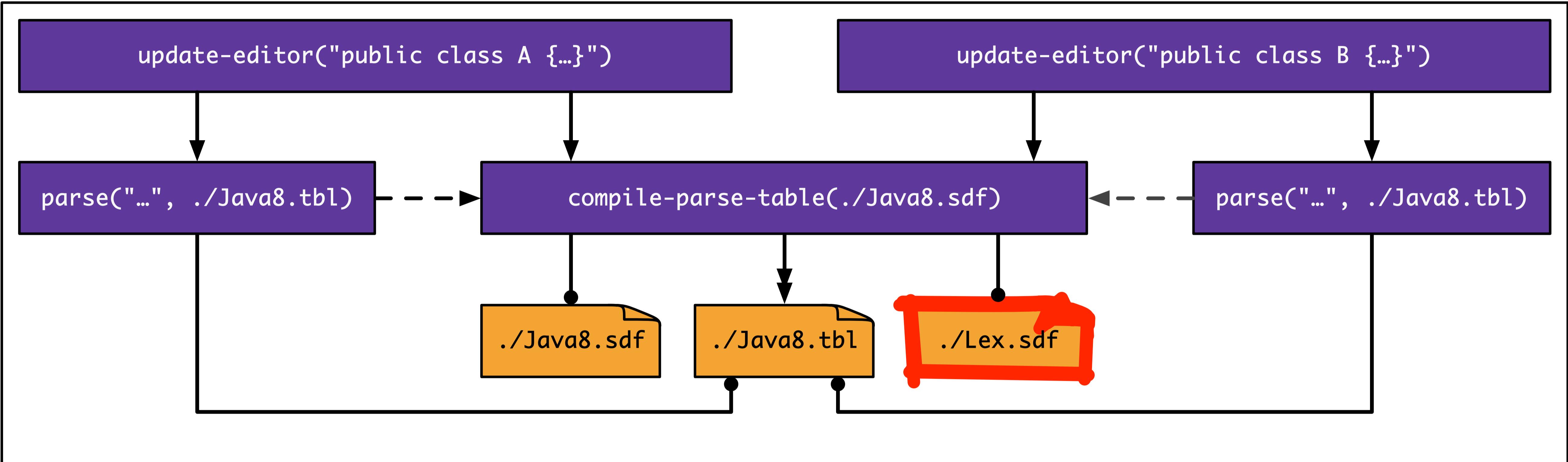
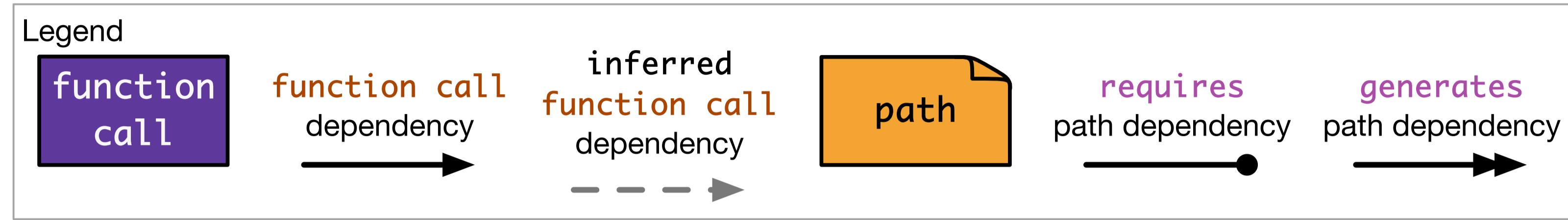
Execute and build dependency graph



```
func update-editor(text: string) -> Msg *={  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

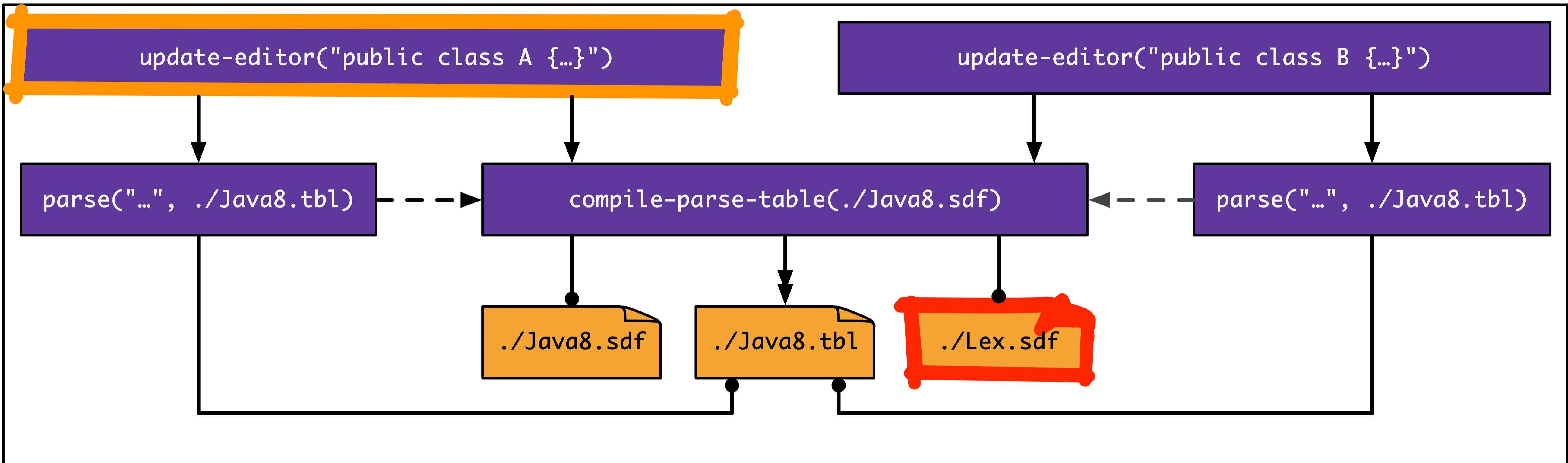
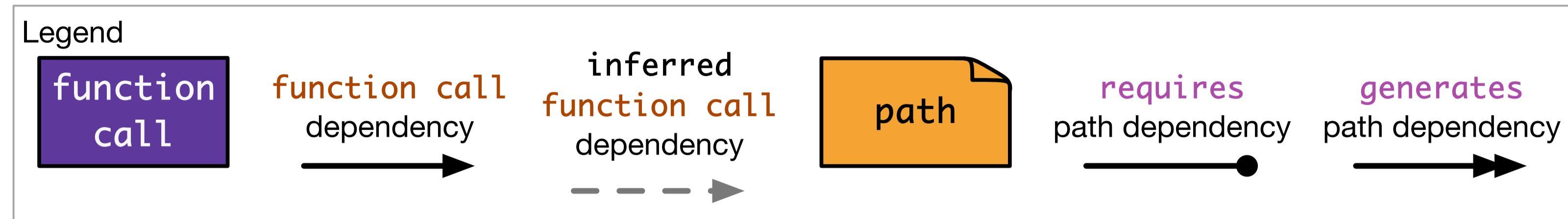
Incremental re-execution with changes



```
func update-editor(text: string) -> Msg *={  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

Incremental re-execution with changes



```

func update-editor(text: string) -> Msg * = {
    val parseTable = compile-parse-table(syntaxDef);
    parse(text, parseTable);
}

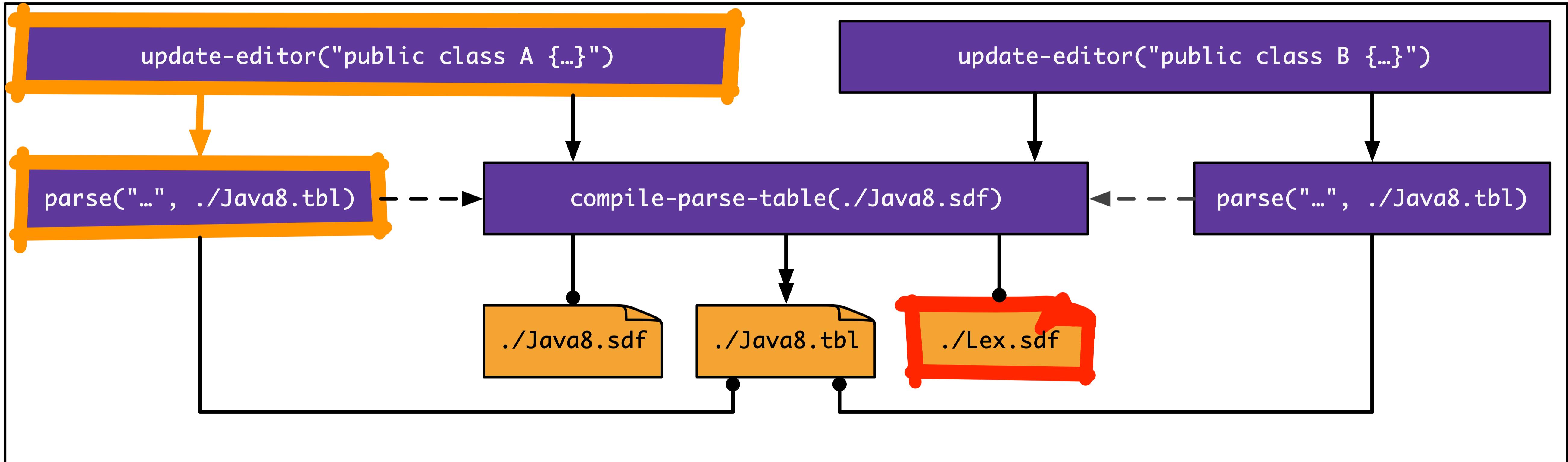
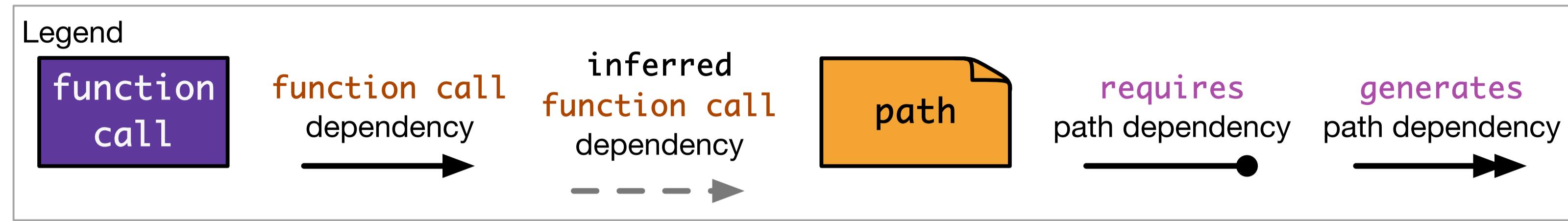
func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    [requires dep | dep <- extract-deps(defFile)];
    generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg *)
  
```

```

class Parse : Func<Parse.Input, Parse.Output> {
    data class Input(val text: String, val table: PPath)
    data class Output(val ast: Ast, val msgs: List<Msg>)
    override fun ExecContext.exec(input: Input): Output {
        require(input.table)
        val result = Api.parse(input.table, input.text)
        return Output(result.ast, result.msgs)
    }
}
  
```

Incremental re-execution with changes



```

func update-editor(text: string) -> Msg * =  

  val parseTable = compile-parse-table(syntaxDef);  

  parse(text, parseTable);  

}  

func compile-parse-table(defFile: path) -> path = {  

  requires defFile;  

  [requires dep | dep <- extract-deps(defFile)];  

  generates tblFile;  

}  

func parse(text: string, pt: path) -> (Ast, Msg *)
  
```

```

class Parse : Func<Parse.Input, Parse.Output> {  

  data class Input(val text: String, val table: PPath)  

  data class Output(val ast: Ast, val msgs: List<Msg>)  

  override fun ExecContext.exec(input: Input): Output {  

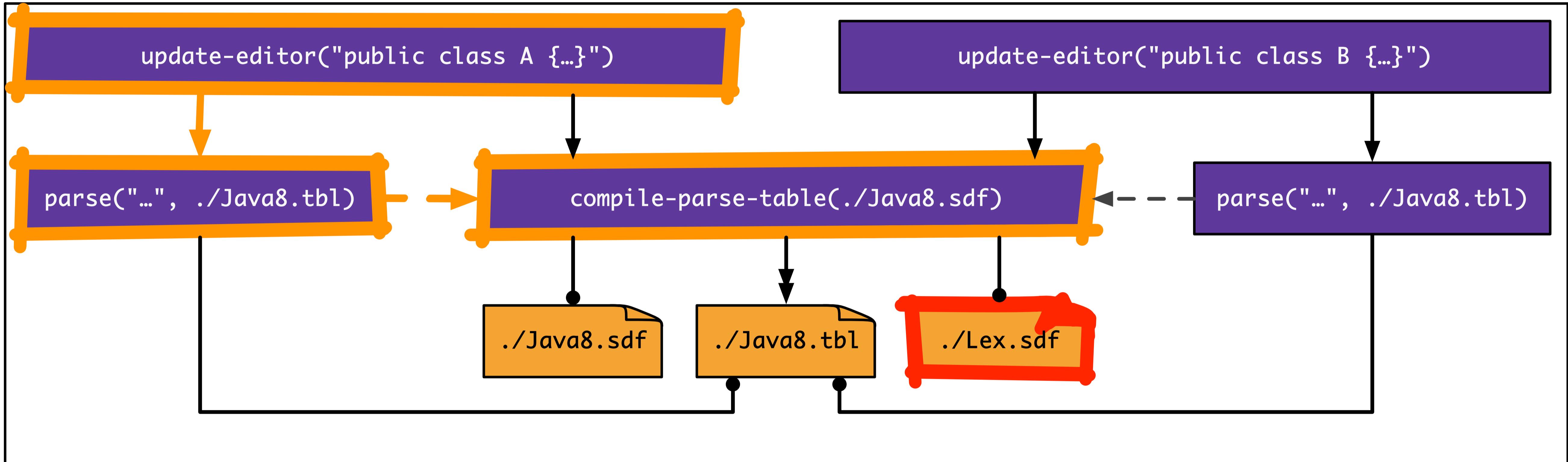
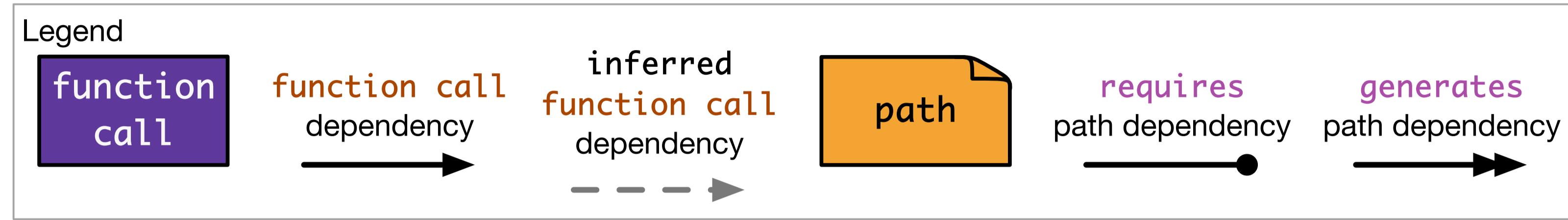
    require(input.table)  

    val result = Api.parse(input.table, input.text)  

    return Output(result.ast, result.msgs)  

  }
}
  
```

Incremental re-execution with changes



```

func update-editor(text: string) -> Msg * = {
    val parseTable = compile-parse-table(syntaxDef);
    parse(text, parseTable);
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    [requires dep | dep <- extract-deps(defFile)];
    generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg *)

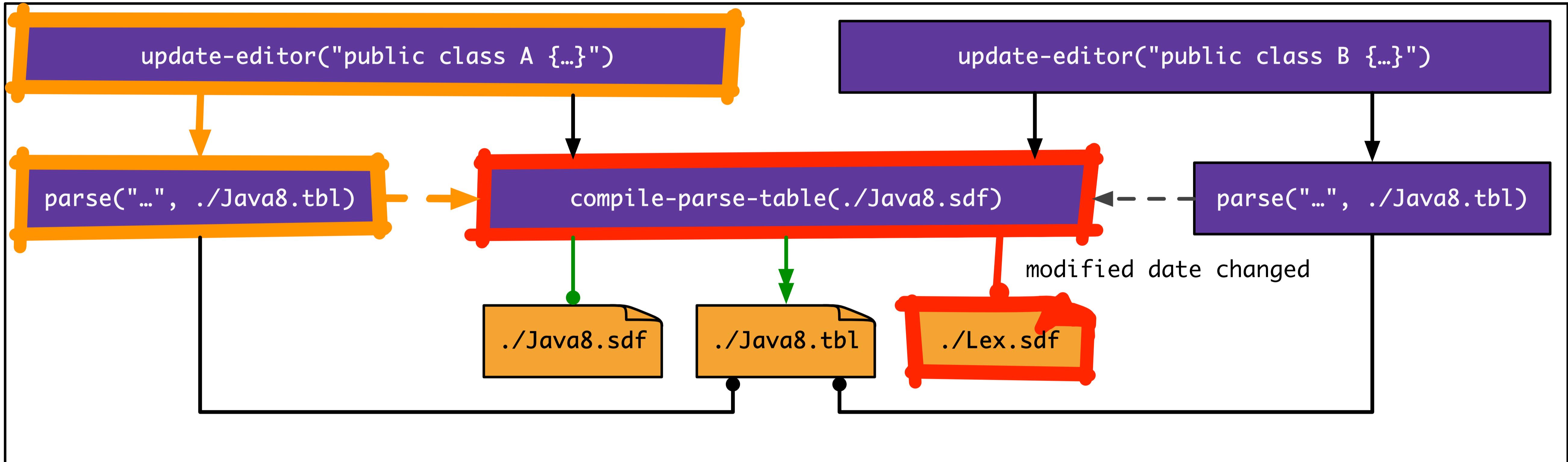
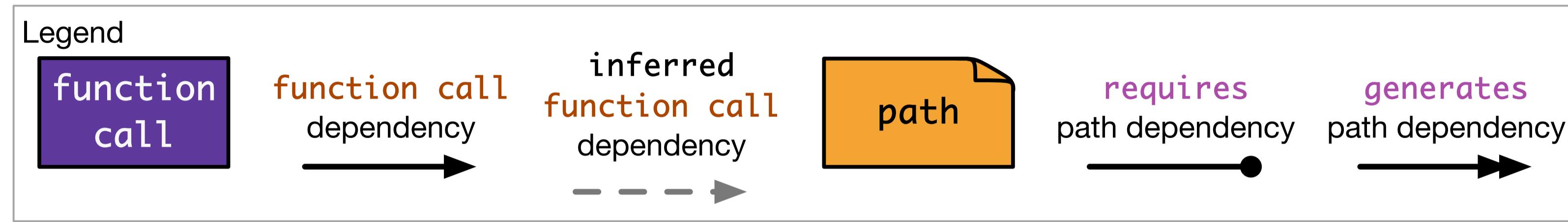
```

```

class Parse : Func<Parse.Input, Parse.Output> {
    data class Input(val text: String, val table: PPath)
    data class Output(val ast: Ast, val msgs: List<Msg>)
    override fun ExecContext.exec(input: Input): Output {
        require(input.table)
        val result = Api.parse(input.table, input.text)
        return Output(result.ast, result.msgs)
    }
}

```

Incremental re-execution with changes



```

func update-editor(text: string) -> Msg * = {
    val parseTable = compile-parse-table(syntaxDef);
    parse(text, parseTable);
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    [requires dep | dep <- extract-deps(defFile)];
    generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg *)

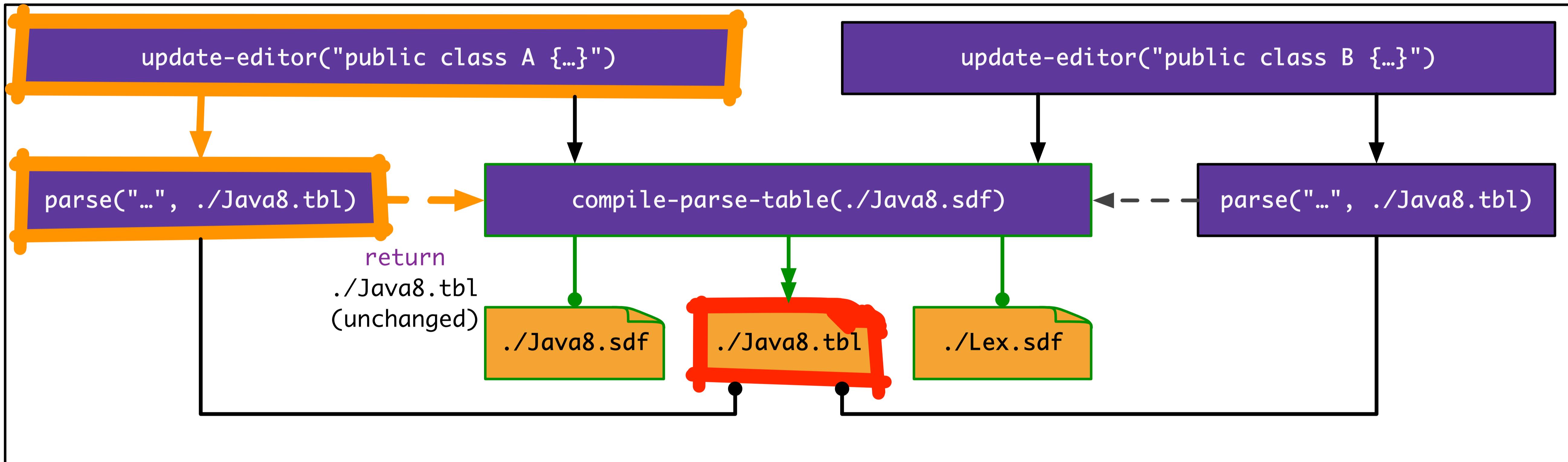
```

```

class Parse : Func<Parse.Input, Parse.Output> {
    data class Input(val text: String, val table: PPath)
    data class Output(val ast: Ast, val msgs: List<Msg>)
    override fun ExecContext.exec(input: Input): Output {
        require(input.table)
        val result = Api.parse(input.table, input.text)
        return Output(result.ast, result.msgs)
    }
}

```

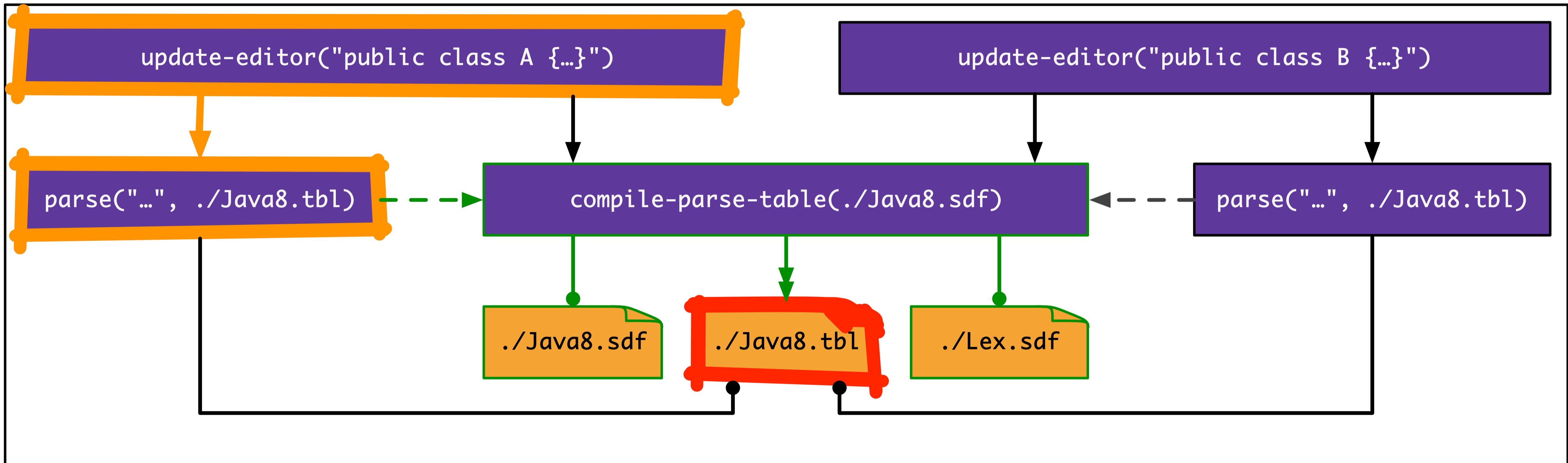
Incremental re-execution with changes



```
func update-editor(text: string) -> Msg *={  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg *)
```

```
class Parse : Func<Parse.Input, Parse.Output>{  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

Incremental re-execution with changes



```

func update-editor(text: string) -> Msg * = {
    val parseTable = compile-parse-table(syntaxDef);
    parse(text, parseTable);
}

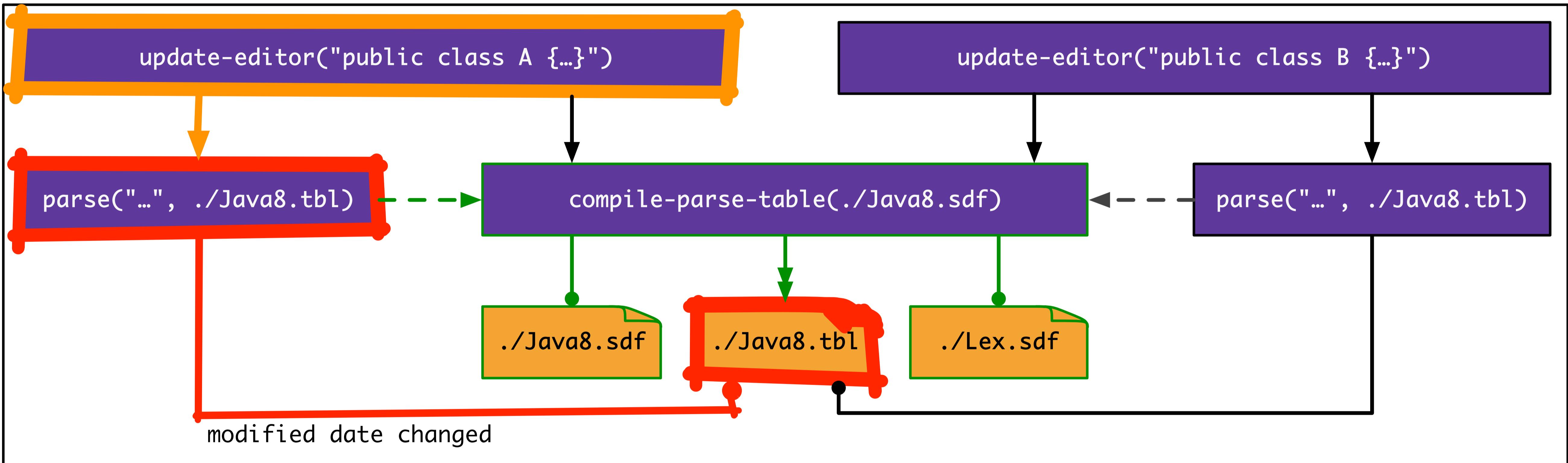
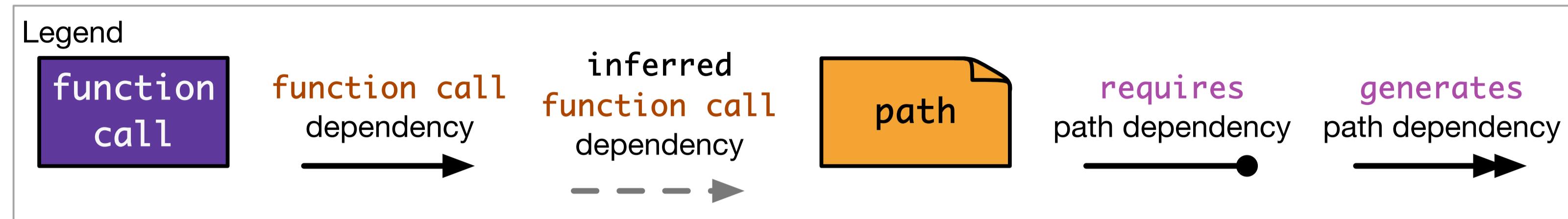
func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    [requires dep | dep <- extract-deps(defFile)];
    generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg *)
  
```

```

class Parse : Func<Parse.Input, Parse.Output> {
    data class Input(val text: String, val table: PPath)
    data class Output(val ast: Ast, val msgs: List<Msg>)
    override fun ExecContext.exec(input: Input): Output {
        require(input.table)
        val result = Api.parse(input.table, input.text)
        return Output(result.ast, result.msgs)
    }
}
  
```

Incremental re-execution with changes



```

func update-editor(text: string) -> Msg * = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}

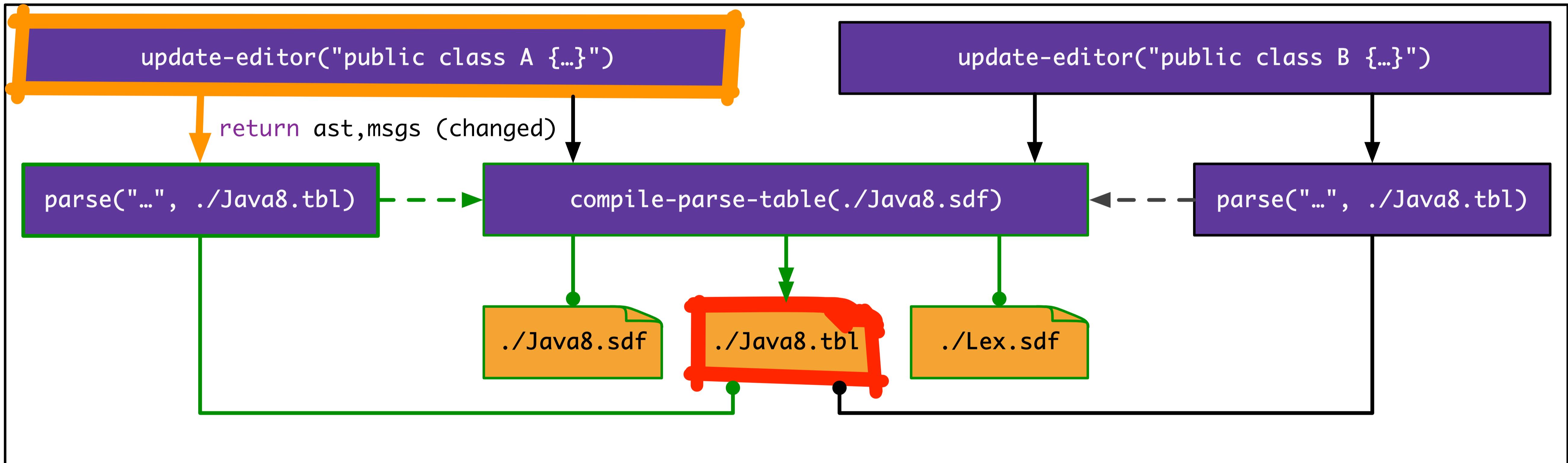
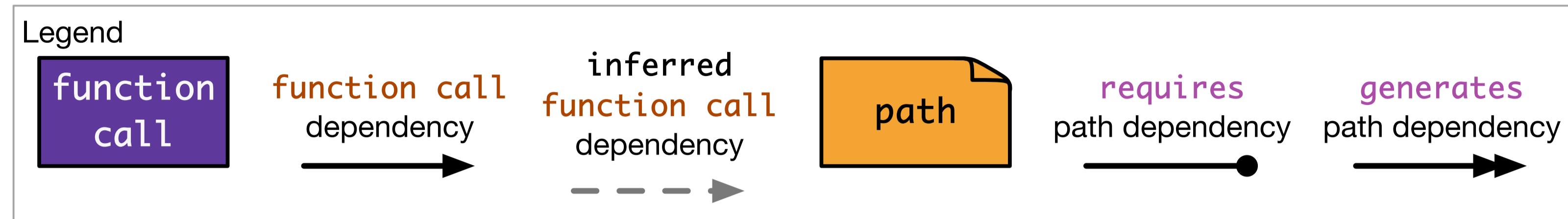
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(defFile)];
  generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg *)
  
```

```

class Parse : Func<Parse.Input, Parse.Output> {
  data class Input(val text: String, val table: PPath)
  data class Output(val ast: Ast, val msgs: List<Msg>)
  override fun ExecContext.exec(input: Input): Output {
    require(input.table)
    val result = Api.parse(input.table, input.text)
    return Output(result.ast, result.msgs)
  }
}
  
```

Incremental re-execution with changes



```

func update-editor(text: string) -> Msg * = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}

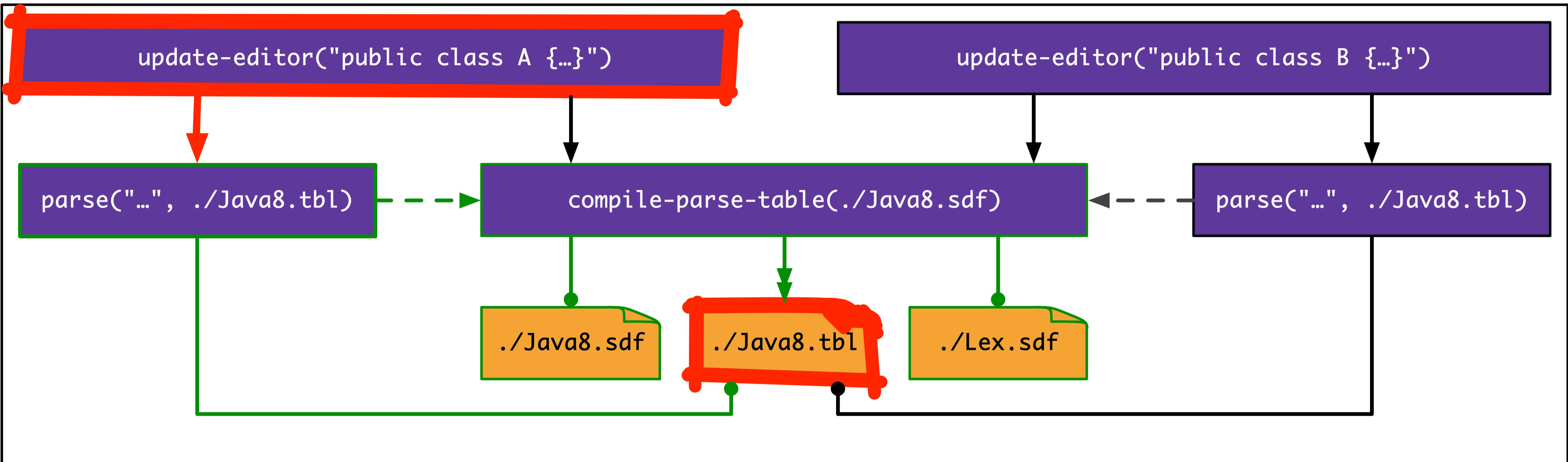
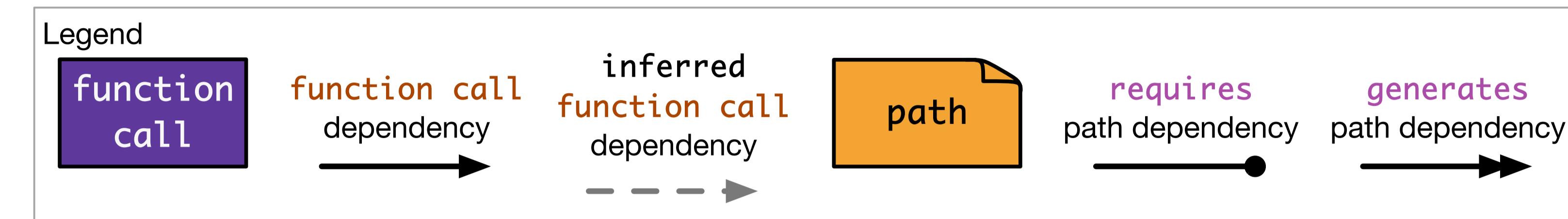
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(defFile)];
  generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg *)
  
```

```

class Parse : Func<Parse.Input, Parse.Output> {
  data class Input(val text: String, val table: PPath)
  data class Output(val ast: Ast, val msgs: List<Msg>)
  override fun ExecContext.exec(input: Input): Output {
    require(input.table)
    val result = Api.parse(input.table, input.text)
    return Output(result.ast, result.msgs)
  }
}
  
```

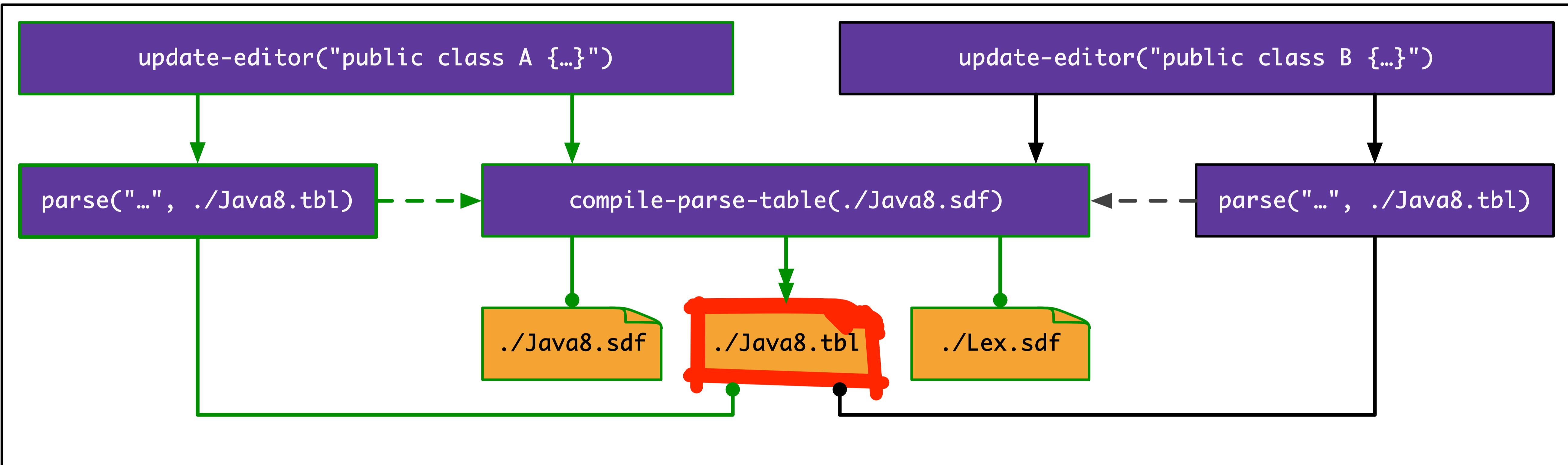
Incremental re-execution with changes



```
func update-editor(text: string) -> Msg *={  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output>{  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

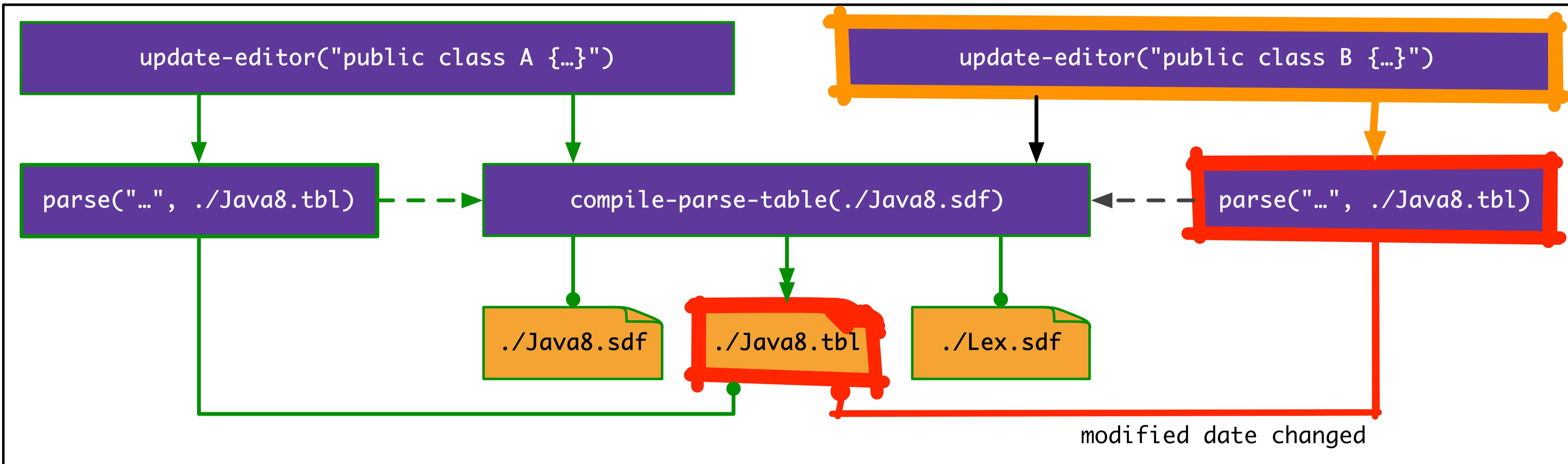
Incremental re-execution with changes



```
func update-editor(text: string) -> Msg *={  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

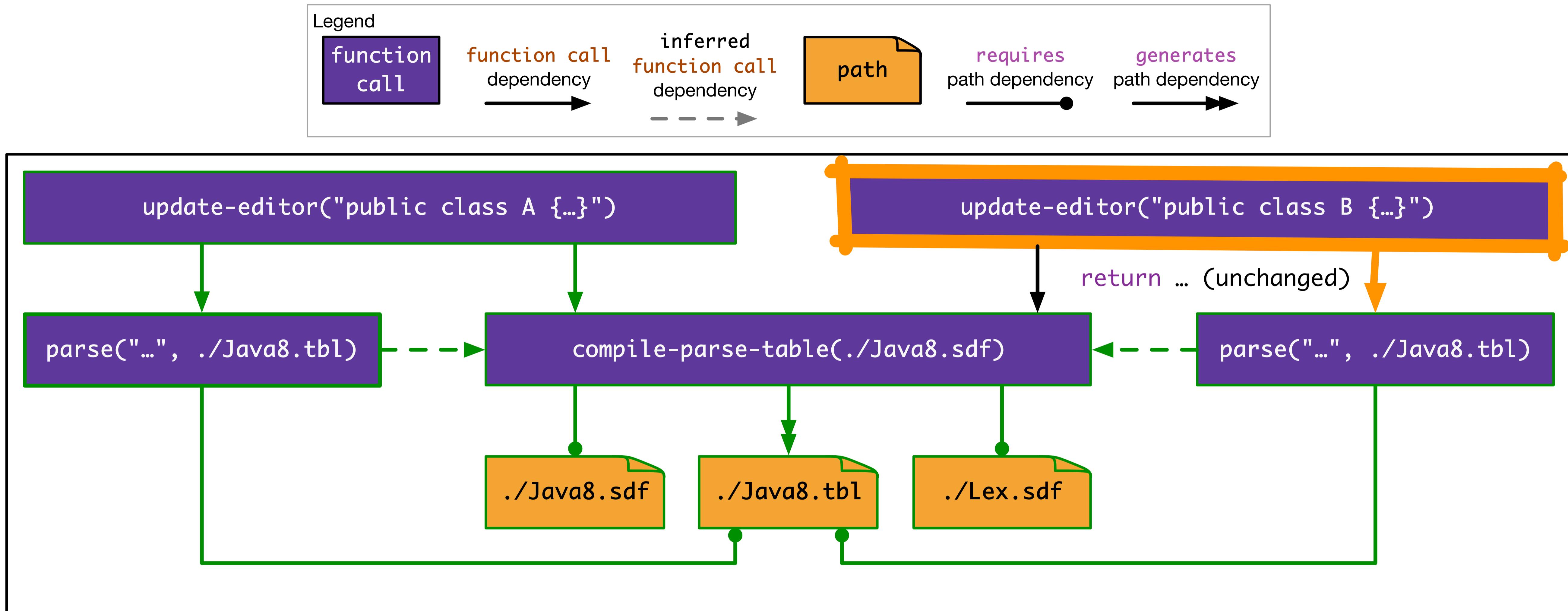
Incremental re-execution with changes



```
func update-editor(text: string) -> Msg *={  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

Incremental re-execution with changes



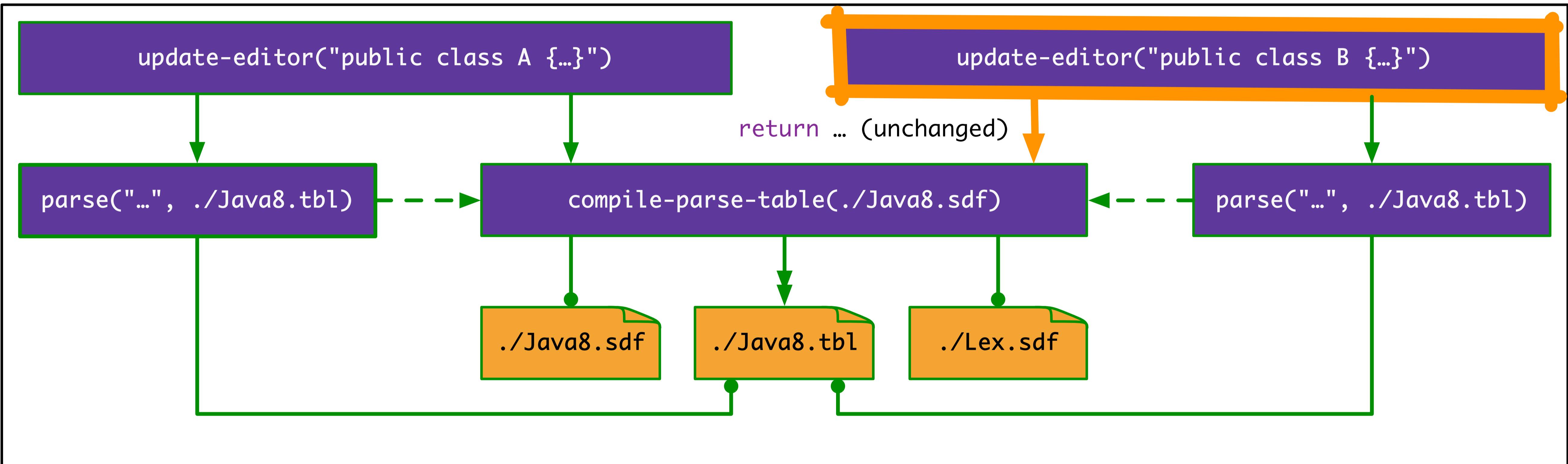
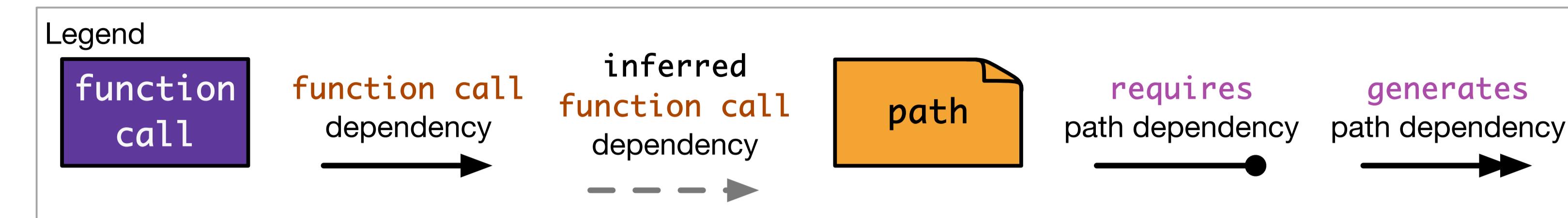
```
func update-editor(text: string) -> Msg* = {
    val parseTable = compile-parse-table(syntaxDef);
    parse(text, parseTable);
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    [requires dep | dep <- extract-deps(defFile)];
    generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

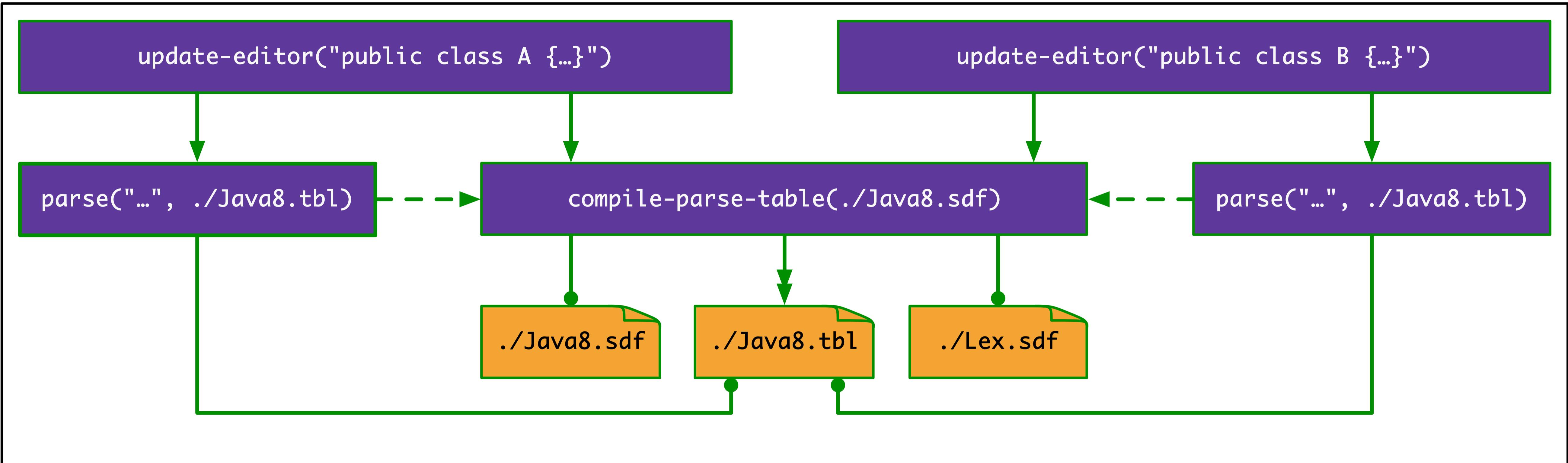
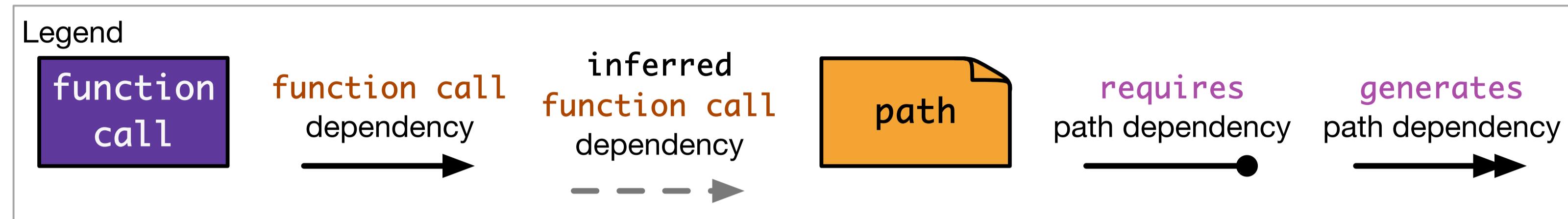
Incremental re-execution with changes



```
func update-editor(text: string) -> Msg *={  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

Incremental re-execution with changes



```
func update-editor(text: string) -> Msg *={  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

Incremental Performance Benchmarking

```
func main(jmhArgs: string*) -> path** = {
    val jar = build();
    val pkg = "io.usethesource.criterion";
    val javaSrcDir = ./src/main/java/io/usethesource/criterion;
    val benchmarks: (string, string, path*)* = [
        ("set", "$pkg.JmhSetBenchmarks.*\\$\"", [javaSrcDir + "/JmhSetBenchmarks.java"])
        , ("map", "$pkg.JmhMapBenchmarks.*\\$\"", [javaSrcDir + "/JmhMapBenchmarks.java"])
    ];
    val subjects: (string, string, path*)* = [
        ("clojure"      , "VF_CLOJURE"      , [./lib/clojure.jar      ])
        , ("champ"       , "VF_CHAMP"       , [./lib/champ.jar       ])
        // etc.
    ];
    [[run_benchmark(jar, jmhArgs, bench, subj) | bench <- benchmarks] | subj <- subjects]
}
func build() -> path = {
    val pomFile = ./pom.xml; requires pomFile;
    [requires file | file <- walk ./src with extensions ["java", "scala"]];
    exec(["mvn", "verify", "-f", "$pomFile"]);
    val jar = ./target/benchmarks.jar;
    generates jar; jar
}
func run_benchmark(jar: path, jmhArgs: string*, benchmark: (string, string, path*),
subject: (string, string, path*)) -> path = {
    ...
}
```

Spoofax-PIE: Live Language Development

// 3) Syntax-based styling

```
data SyntaxStyler = foreign java class {}  
data Styling = foreign java class {}  
func esv2styler(path) -> SyntaxStyler = foreign java class#method  
func esvStyle(Token*, SyntaxStyler) -> Styling = foreign java esv#style  
func style(tokens: Token*, langSpec: LangSpec) -> Styling = {  
    val mainFile = langSpec.styling();  
    requires mainFile;  
    val styler = esv2styler(mainFile);  
    esvStyle(tokens, styler)  
}
```

// 4) Combine parsing and styling to process strings and files

```
func processString(text: string, langSpec: LangSpec) -> (Msg*, Styling?) = {  
    val (ast, tokens, msgs) = parse(text, langSpec);  
    val styling = if(tokens != null) style(tokens, langSpec) else null;  
    (msgs, styling)  
}  
func processFile(file: path, langSpec: LangSpec) -> (Msg*, Styling?) = processString(read file, langSpec)
```

// 5) Keep files of an Eclipse project up-to-date

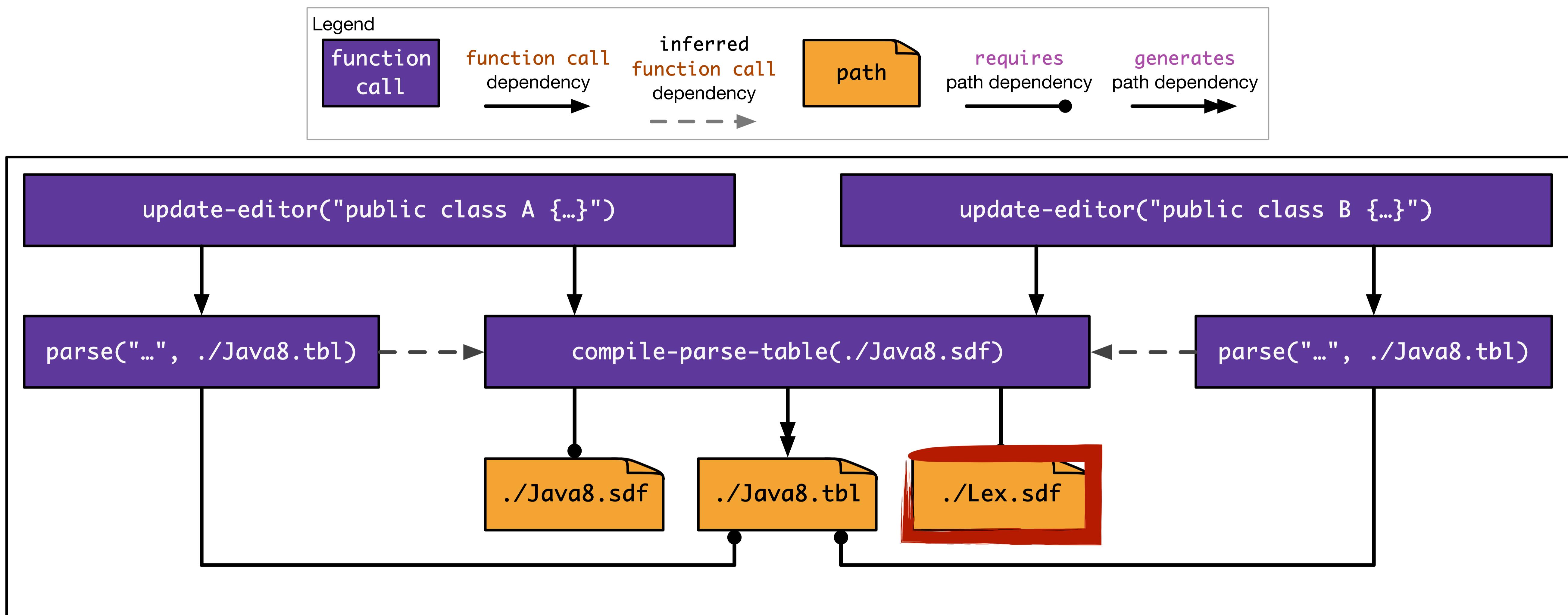
```
func updateProject(root: path, project: path) -> (path, Msg*, Styling?)* = {  
    val workspace = getWorkspace(root);  
    val relevantFiles = walk project with extensions workspace.extensions();  
    [updateFile(file, workspace) | file <- relevantFiles]  
}  
func updateFile(file: path, workspace: Workspace) -> (path, Msg*, Styling?) = {  
    val langSpec = workspace.langSpec(file);  
    val (msgs, styling) = processFile(file, langSpec); (file, msgs, styling)
```

On-demand execution

vs

Change-driven execution

Change-driven execution



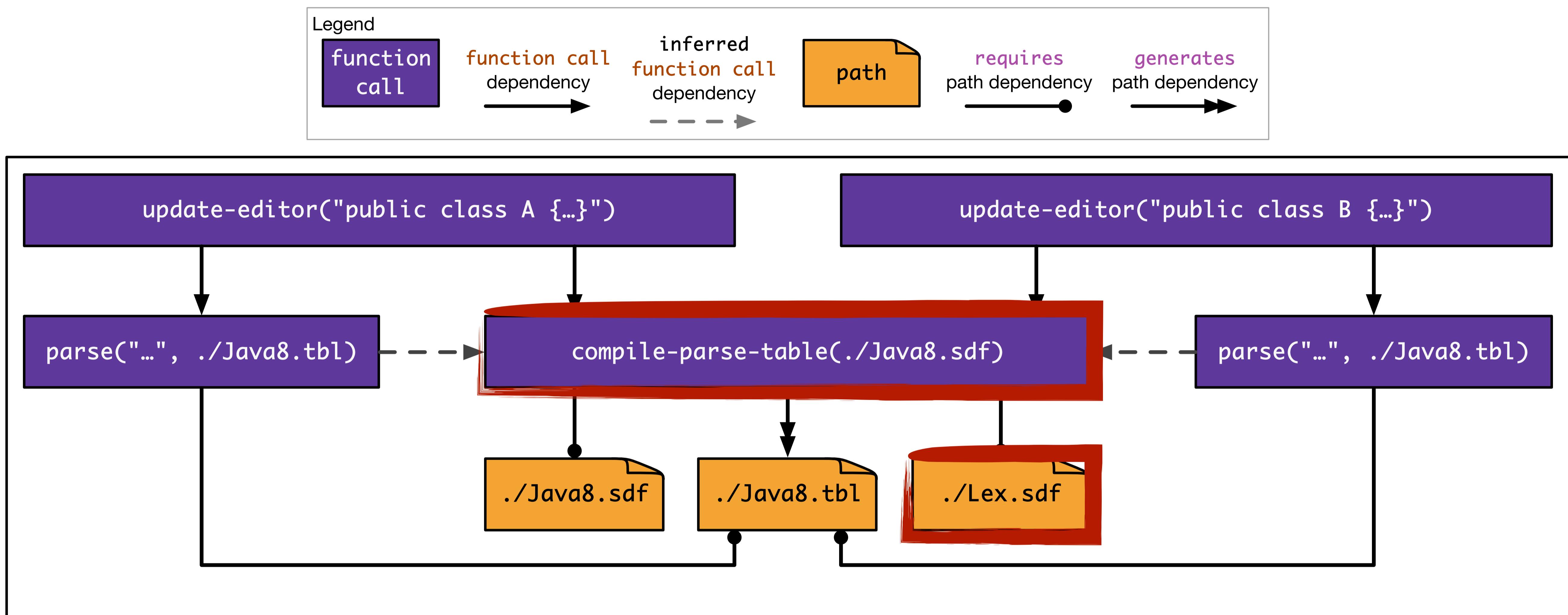
```
func update-editor(text: string) -> Msg* = {
    val parseTable = compile-parse-table(syntaxDef);
    parse(text, parseTable);
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    [requires dep | dep <- extract-deps(defFile)];
    generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {
    data class Input(val text: String, val table: PPath)
    data class Output(val ast: Ast, val msgs: List<Msg>)
    override fun ExecContext.exec(input: Input): Output {
        require(input.table)
        val result = Api.parse(input.table, input.text)
        return Output(result.ast, result.msgs)
    }
}
```

Change-driven execution



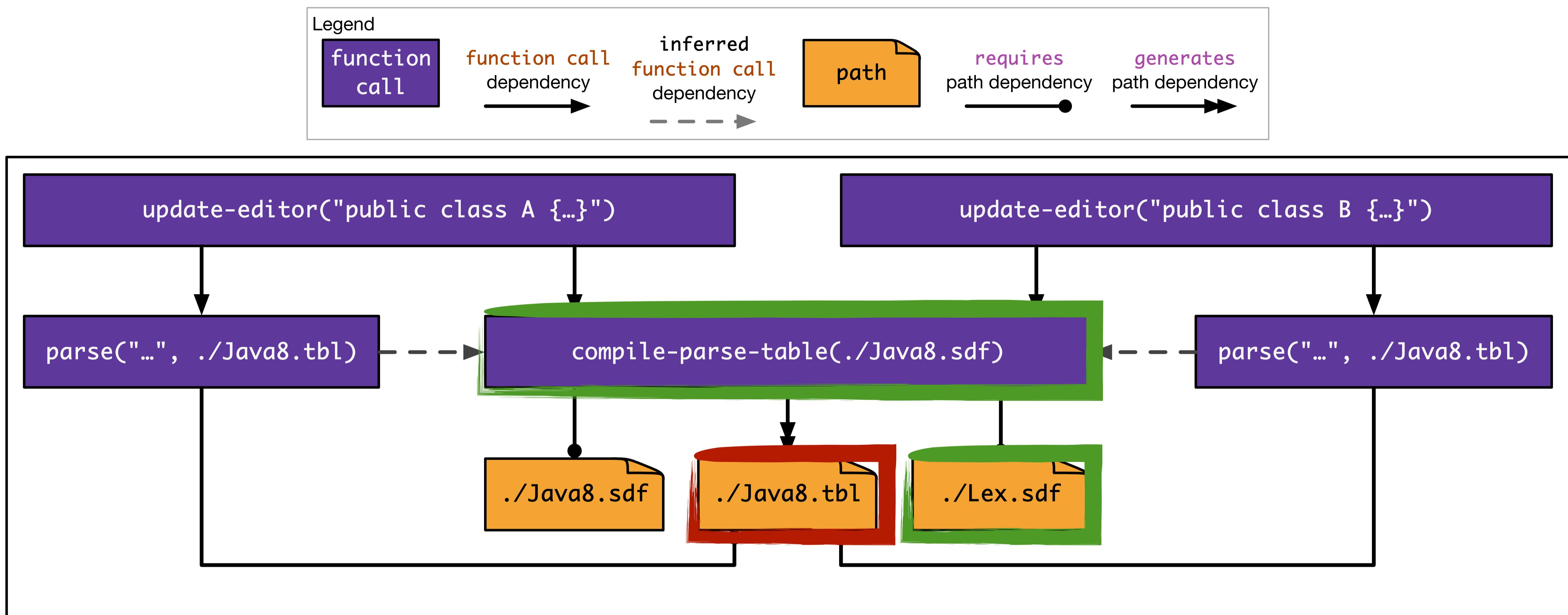
```
func update-editor(text: string) -> Msg* = {
    val parseTable = compile-parse-table(syntaxDef);
    parse(text, parseTable);
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    [requires dep | dep <- extract-deps(defFile)];
    generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

Change-driven execution



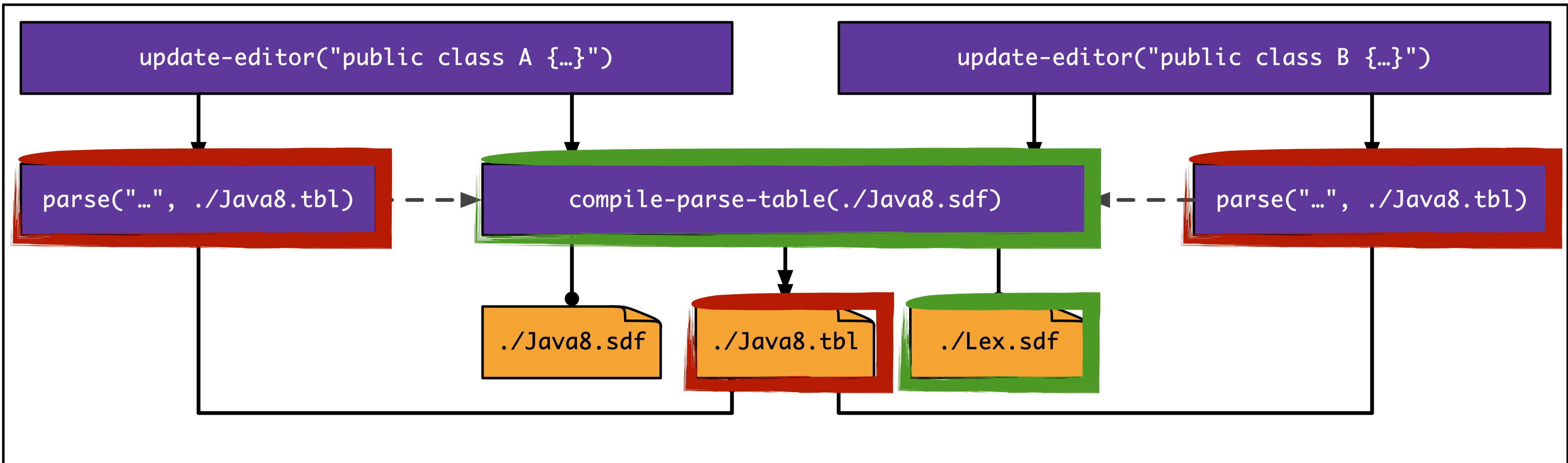
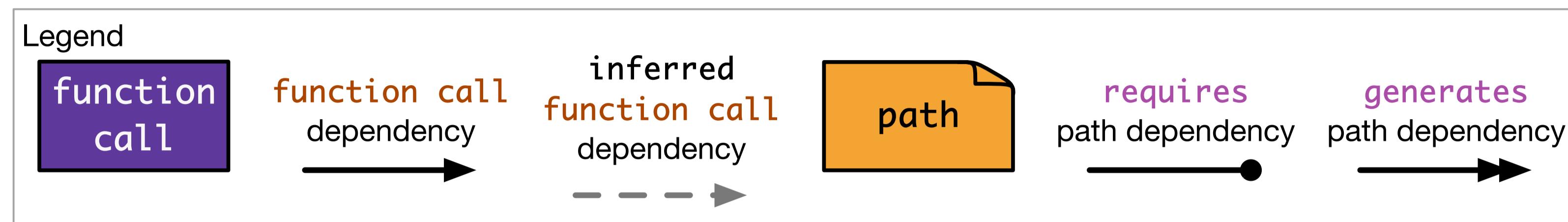
```
func update-editor(text: string) -> Msg* = {
    val parseTable = compile-parse-table(syntaxDef);
    parse(text, parseTable);
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    [requires dep | dep <- extract-deps(defFile)];
    generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {
    data class Input(val text: String, val table: PPath)
    data class Output(val ast: Ast, val msgs: List<Msg>)
    override fun ExecContext.exec(input: Input): Output {
        require(input.table)
        val result = Api.parse(input.table, input.text)
        return Output(result.ast, result.msgs)
    }
}
```

Change-driven execution



```

func update-editor(text: string) -> Msg * = {
    val parseTable = compile-parse-table(syntaxDef);
    parse(text, parseTable);
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    [requires dep | dep <- extract-deps(defFile)];
    generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg *)

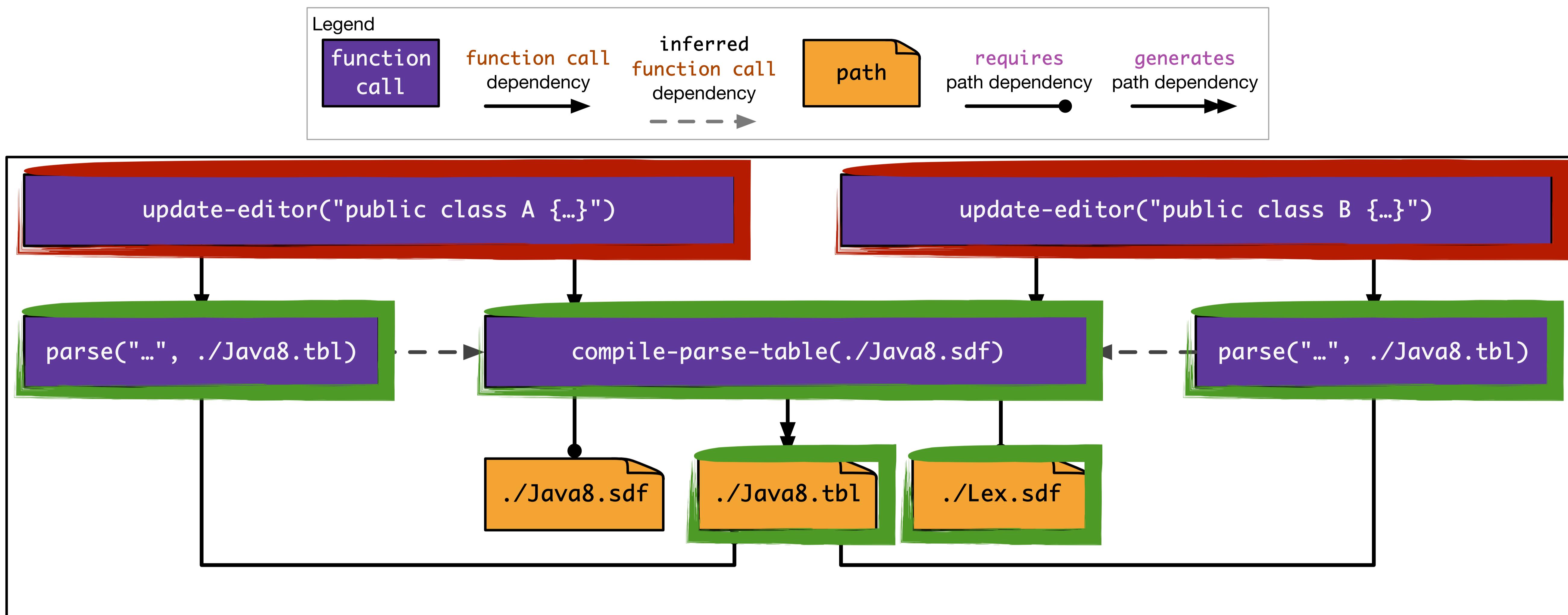
```

```

class Parse : Func<Parse.Input, Parse.Output> {
    data class Input(val text: String, val table: PPath)
    data class Output(val ast: Ast, val msgs: List<Msg>)
    override fun ExecContext.exec(input: Input): Output {
        require(input.table)
        val result = Api.parse(input.table, input.text)
        return Output(result.ast, result.msgs)
    }
}

```

Change-driven execution



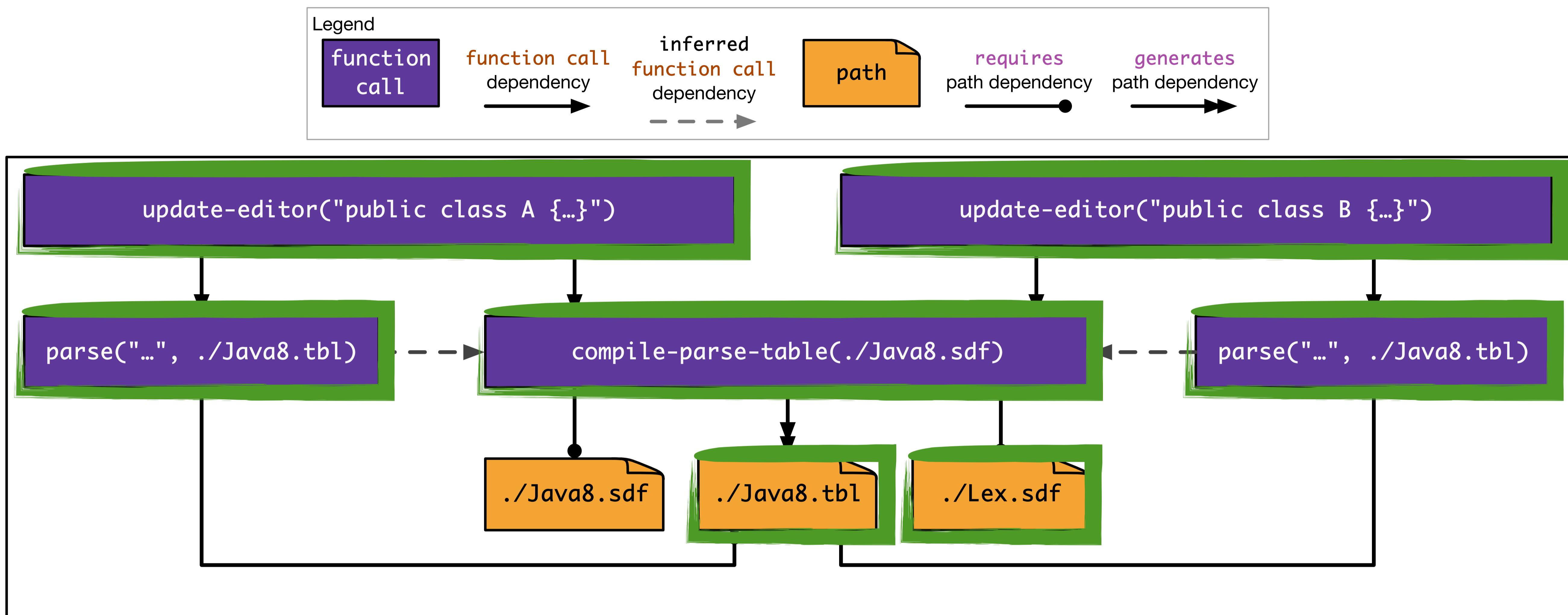
```
func update-editor(text: string) -> Msg* = {
    val parseTable = compile-parse-table(syntaxDef);
    parse(text, parseTable);
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    [requires dep | dep <- extract-deps(defFile)];
    generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {
    data class Input(val text: String, val table: PPath)
    data class Output(val ast: Ast, val msgs: List<Msg>)
    override fun ExecContext.exec(input: Input): Output {
        require(input.table)
        val result = Api.parse(input.table, input.text)
        return Output(result.ast, result.msgs)
    }
}
```

Change-driven execution



```
func update-editor(text: string) -> Msg* = {
    val parseTable = compile-parse-table(syntaxDef);
    parse(text, parseTable);
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    [requires dep | dep <- extract-deps(defFile)];
    generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {
    data class Input(val text: String, val table: PPath)
    data class Output(val ast: Ast, val msgs: List<Msg>)
    override fun ExecContext.exec(input: Input): Output {
        require(input.table)
        val result = Api.parse(input.table, input.text)
        return Output(result.ast, result.msgs)
    }
}
```

On-demand execution

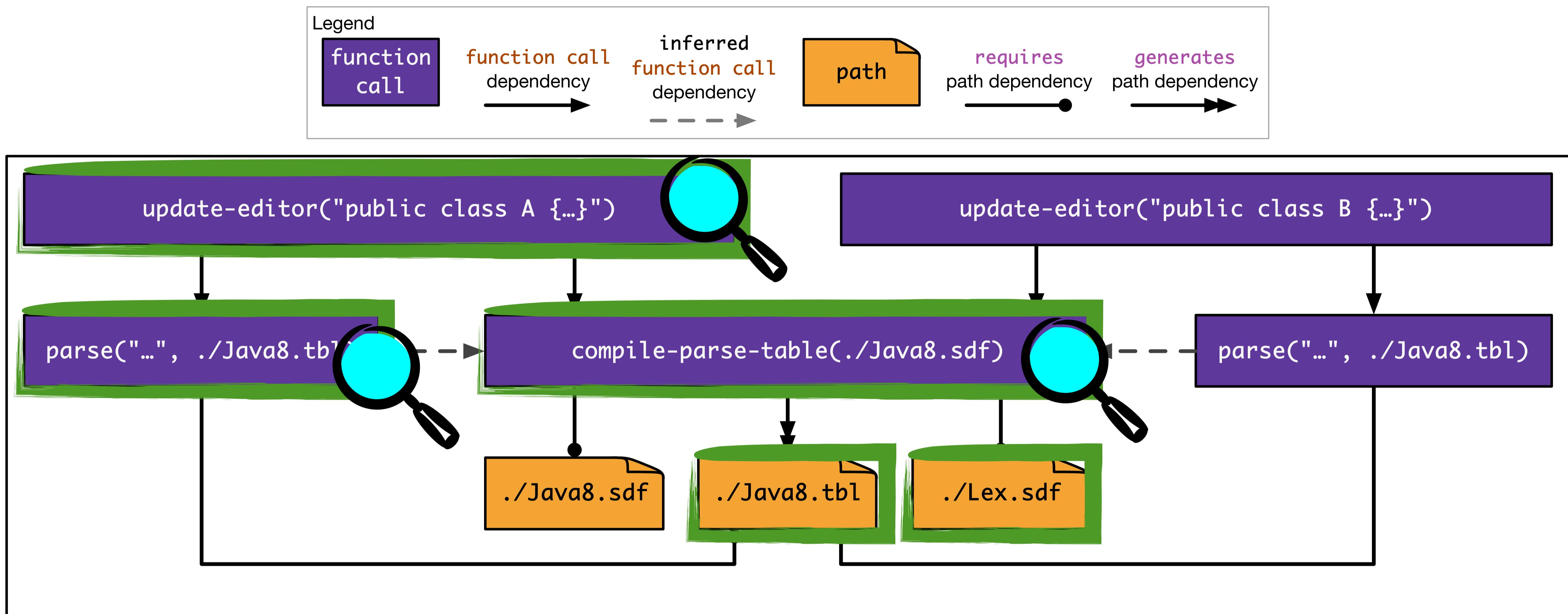
vs

Change-driven execution

vs

Observed change-driven exec

Observed change-driven execution



```
func update-editor(text: string) -> Msg * =  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path =  
    requires defFile;  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
}  
func parse(text: string, pt: path) -> (Ast, Msg*)
```

```
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

Developing interactive pipelines is

Complicated

Error Prone

High Development Cost

when programmers *manually reason*
about **incrementality**

9

Developing interactive pipelines is

Complicated

Error Prone

High Development Cost

when programmers *manually reason* about **incrementality**

9

PIE DSL code

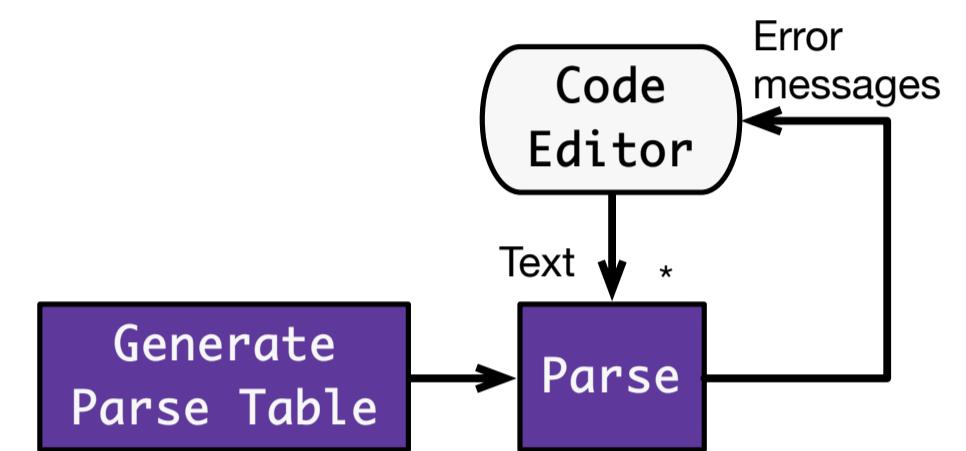
```
func update-editor(text: string) -> Msg* = {  
    val syntaxDef = ./Java8.sdf;  
    val parseTable = compile-parse-table(syntaxDef);  
    val (_, messages) = parse(text, parseTable);  
    messages  
}  
  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    val tblFile = defFile.replaceExtension("tbl");  
    val depFile = defFile.replaceExtension("dep");  
    exec(["sdf2table"] + "$defFile" + "-o$tblFile"  
        + "-d$depFile");  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
    tblFile  
}  
func extract-deps(depFile: path) -> path*  
= foreign jvm spoofax.sdf#extractDeps  
  
data Ast = foreign jvm spoofax.Ast;  
data Msg = foreign jvm spoofax.Msg;  
func parse(text: string, pt: path) -> (Ast, Msg*)  
= foreign jvm spoofax.sdf.Parse
```

27

Foreign JVM code

```
fun extractDeps(depFile: PPath) = Api.extractSdfDeps(depFile)  
  
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

```
public interface Ast { ... }  
public interface Msg { ... }
```



Developing interactive pipelines is

Complicated

Error Prone

High Development Cost

when programmers *manually reason* about **incrementality**

9

PIE DSL code

```
func update-editor(text: string) -> Msg* = {
    val syntaxDef = ./Java8.sdf;
    val parseTable = compile-parse-table(syntaxDef);
    val (_, messages) = parse(text, parseTable);
    messages
}

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    val tblFile = defFile.replaceExtension("tbl");
    val depFile = defFile.replaceExtension("dep");
    exec(["sdf2table"] + "$defFile" + "-o$tblFile"
        + "-d$depFile");
    [requires dep | dep <- extract-deps(defFile)];
    generates tblFile;
    tblFile
}
func extract-deps(defFile: path) -> path*
    = foreign jvm spoofax.sdf#extractDeps

data Ast = foreign jvm spoofax.Ast;
data Msg = foreign jvm spoofax.Msg;
func parse(text: string, pt: path) -> (Ast, Msg*)
    = foreign jvm spoofax.sdf.Parse
```

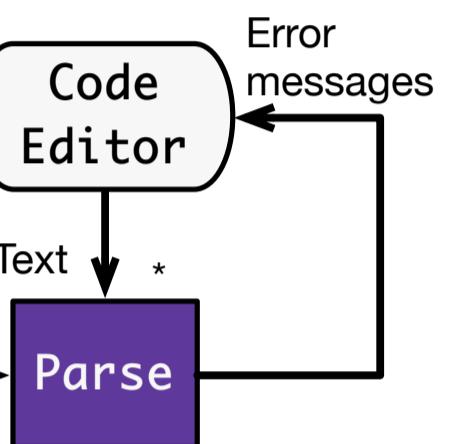
27

Foreign JVM code

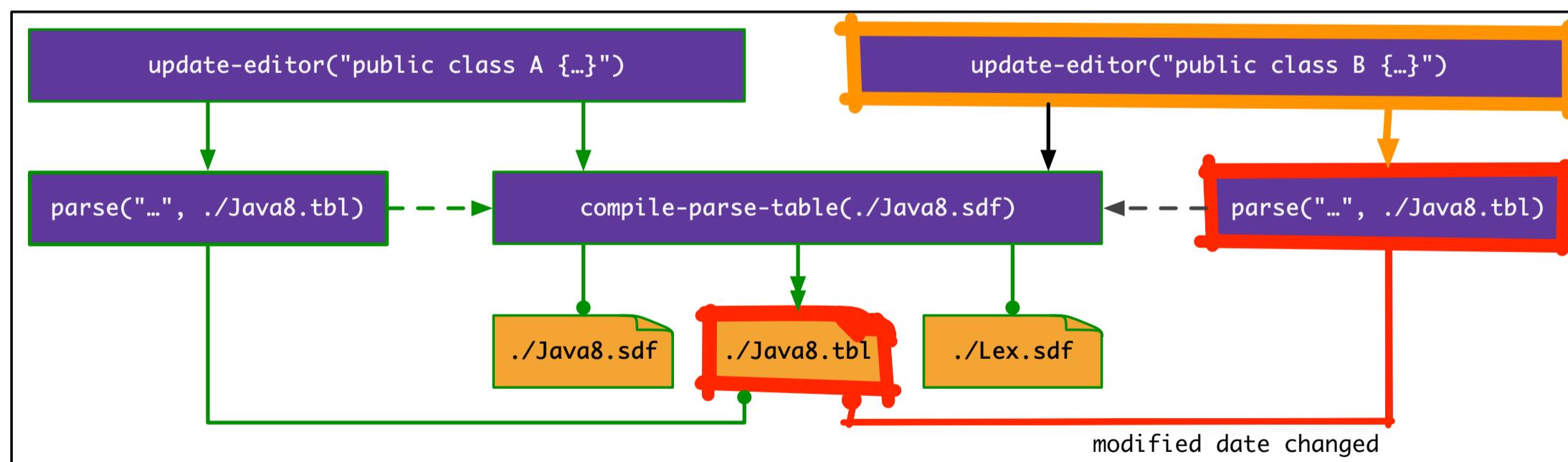
```
fun extractDeps(defFile: PPath) = Api.extractSdfDeps(defFile)

class Parse : Func<Parse.Input, Parse.Output> {
    data class Input(val text: String, val table: PPath)
    data class Output(val ast: Ast, val msgs: List<Msg>)
    override fun ExecContext.exec(input: Input): Output {
        require(input.table)
        val result = Api.parse(input.table, input.text)
        return Output(result.ast, result.msgs)
    }
}
```

```
public interface Ast { ... }
public interface Msg { ... }
```



Incremental re-execution with changes



```
func update-editor(text: string) -> Msg* = [
    val parseTable = compile-parse-table(syntaxDef);
    parse(text, parseTable);
]

func compile-parse-table(defFile: path) -> path = {
    requires defFile;
    [requires dep | dep <- extract-deps(defFile)];
    generates tblFile;
}

func parse(text: string, pt: path) -> (Ast, Msg*)
    = foreign jvm spoofax.sdf.Parse
```

```
class Parse : Func<Parse.Input, Parse.Output> {
    data class Input(val text: String, val table: PPath)
    data class Output(val ast: Ast, val msgs: List<Msg>)
    override fun ExecContext.exec(input: Input): Output {
        require(input.table)
        val result = Api.parse(input.table, input.text)
        return Output(result.ast, result.msgs)
    }
}
```

52

Developing interactive pipelines is

Complicated

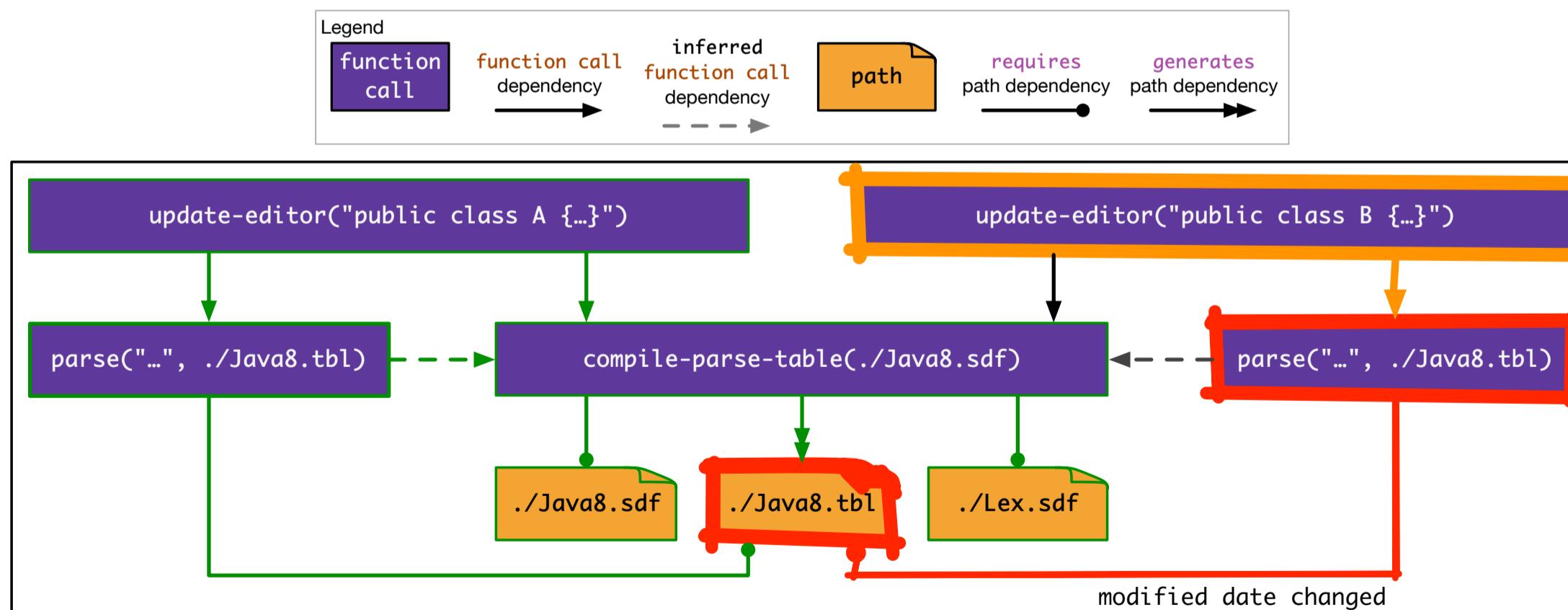
Error Prone

High Development Cost

when programmers *manually reason* about **incrementality**

9

Incremental re-execution with changes



```
func update-editor(text: string) -> Msg* = {  
    val syntaxDef = ./Java8.sdf;  
    val parseTable = compile-parse-table(syntaxDef);  
    parse(text, parseTable);  
}  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    val tblFile = defFile.replaceExtension("tbl");  
    val depFile = defFile.replaceExtension("dep");  
    exec(["sdf2table"] + "$defFile" + "-o$tblFile"  
        + "-d$depFile");  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
    tblFile  
}  
func extract-deps(depFile: path) -> path*  
= foreign jvm spoofax.sdf#extractDeps  
data Ast = foreign jvm spoofax.Ast;  
data Msg = foreign jvm spoofax.Msg;  
func parse(text: string, pt: path) -> (Ast, Msg*)  
= foreign jvm spoofax.sdf.Parse
```

52

PIE DSL code

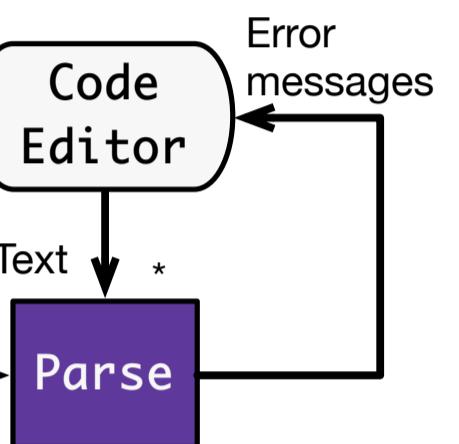
```
func update-editor(text: string) -> Msg* = {  
    val syntaxDef = ./Java8.sdf;  
    val parseTable = compile-parse-table(syntaxDef);  
    val (_, messages) = parse(text, parseTable);  
    messages  
}  
  
func compile-parse-table(defFile: path) -> path = {  
    requires defFile;  
    val tblFile = defFile.replaceExtension("tbl");  
    val depFile = defFile.replaceExtension("dep");  
    exec(["sdf2table"] + "$defFile" + "-o$tblFile"  
        + "-d$depFile");  
    [requires dep | dep <- extract-deps(defFile)];  
    generates tblFile;  
    tblFile  
}  
func extract-deps(depFile: path) -> path*  
= foreign jvm spoofax.sdf#extractDeps  
data Ast = foreign jvm spoofax.Ast;  
data Msg = foreign jvm spoofax.Msg;  
func parse(text: string, pt: path) -> (Ast, Msg*)  
= foreign jvm spoofax.sdf.Parse
```

27

Foreign JVM code

```
fun extractDeps(depFile: PPath) = Api.extractSdfDeps(depFile)  
class Parse : Func<Parse.Input, Parse.Output> {  
    data class Input(val text: String, val table: PPath)  
    data class Output(val ast: Ast, val msgs: List<Msg>)  
    override fun ExecContext.exec(input: Input): Output {  
        require(input.table)  
        val result = Api.parse(input.table, input.text)  
        return Output(result.ast, result.msgs)  
    }  
}
```

```
public interface Ast { ... }  
public interface Msg { ... }
```



Incremental



Correct



Persistent



Expressive



Easy to develop



65