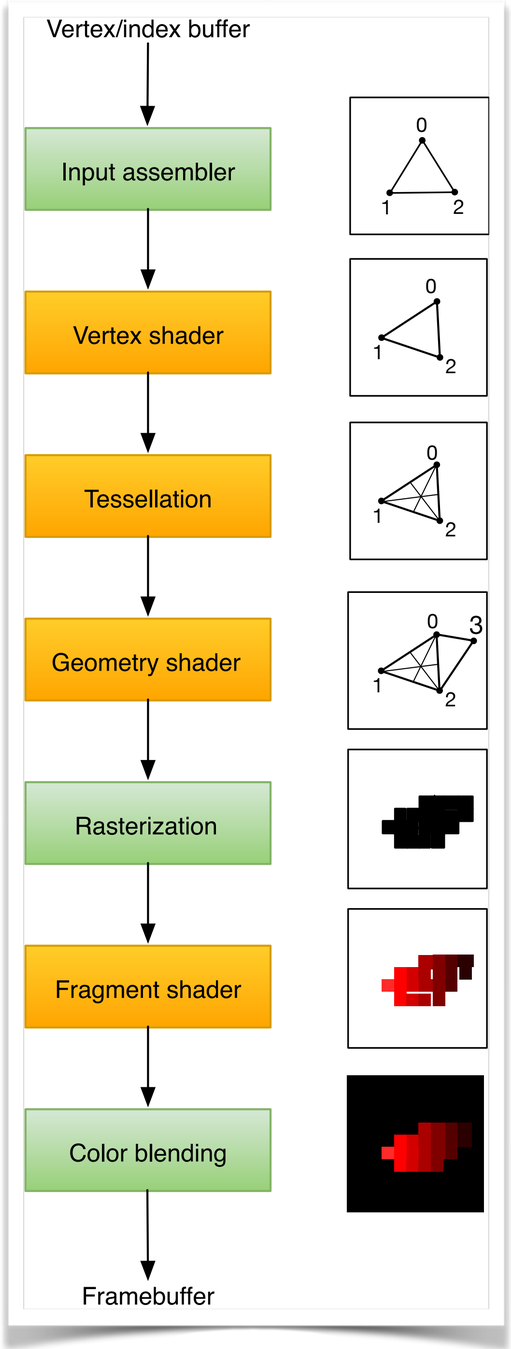
PIE: A System for Programming Interactive Pipelines

Gabriël Konat

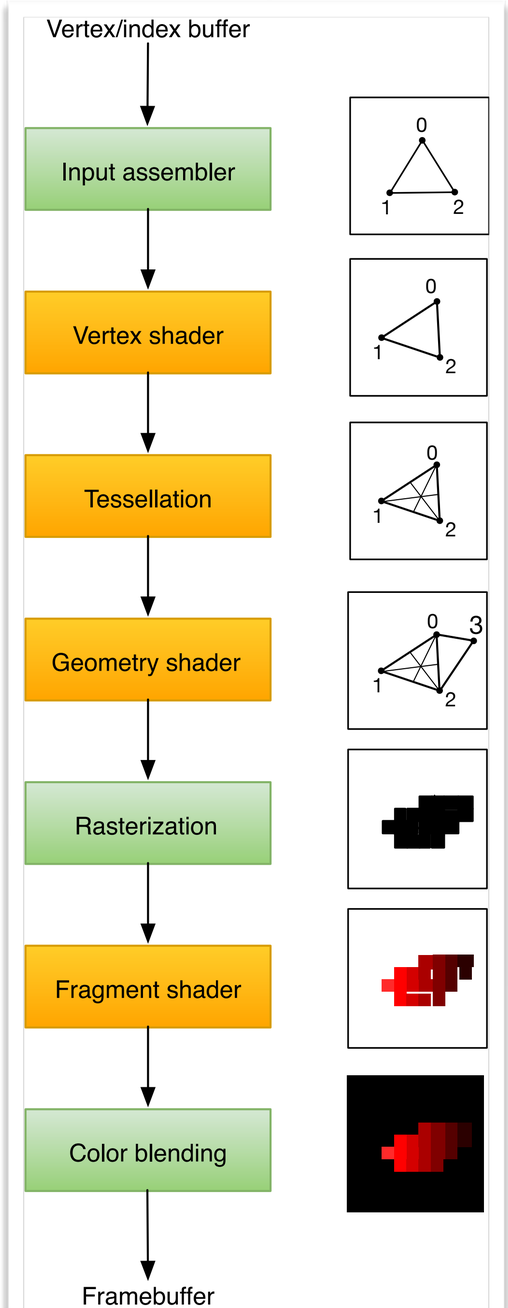
Join work with: Michael Steindorfer, Sebastian Erdweg, Eelco Visser

TU Delft

Pipeline: Flow of Computation



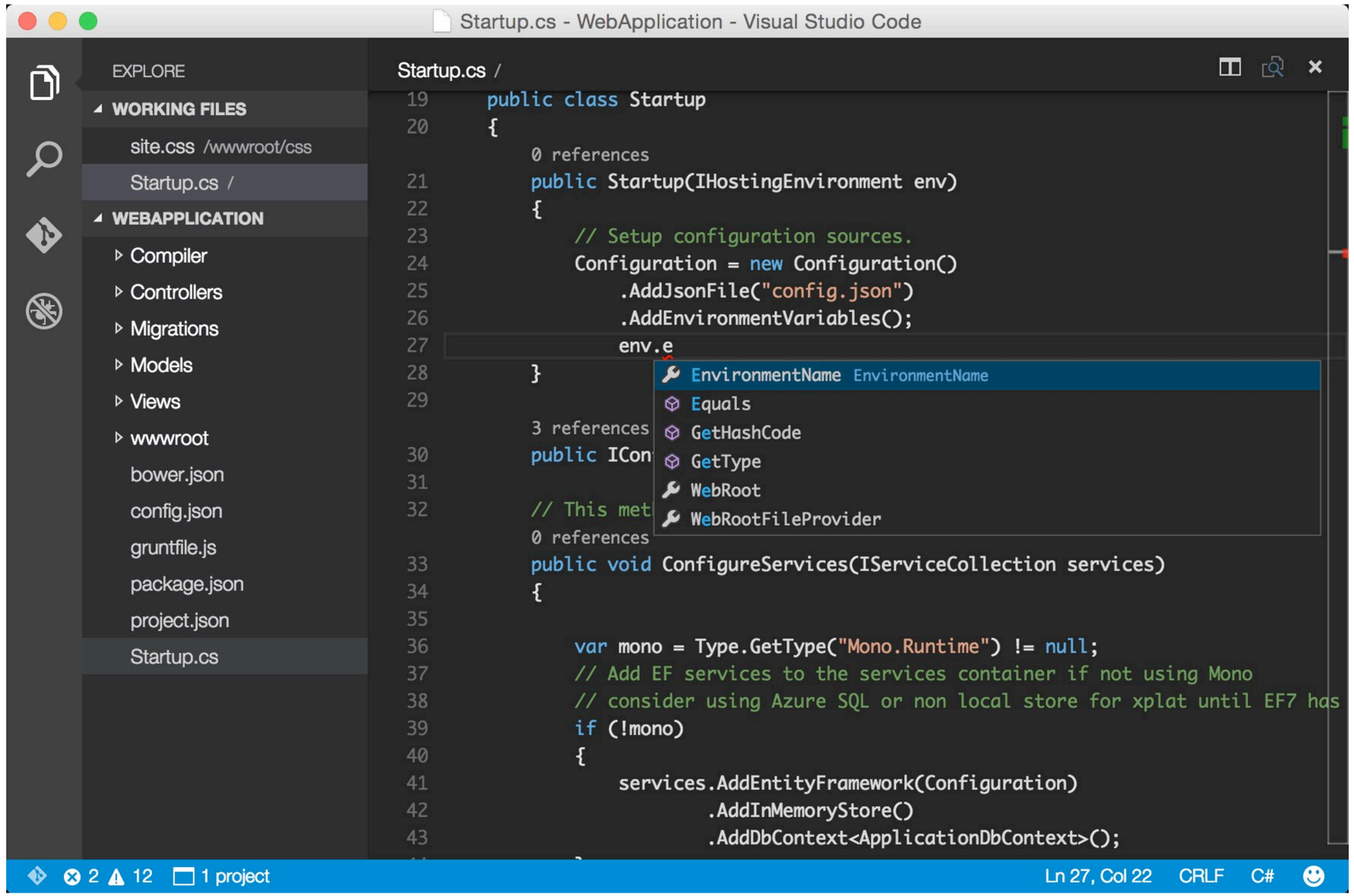
Pipeline: Flow of Computation



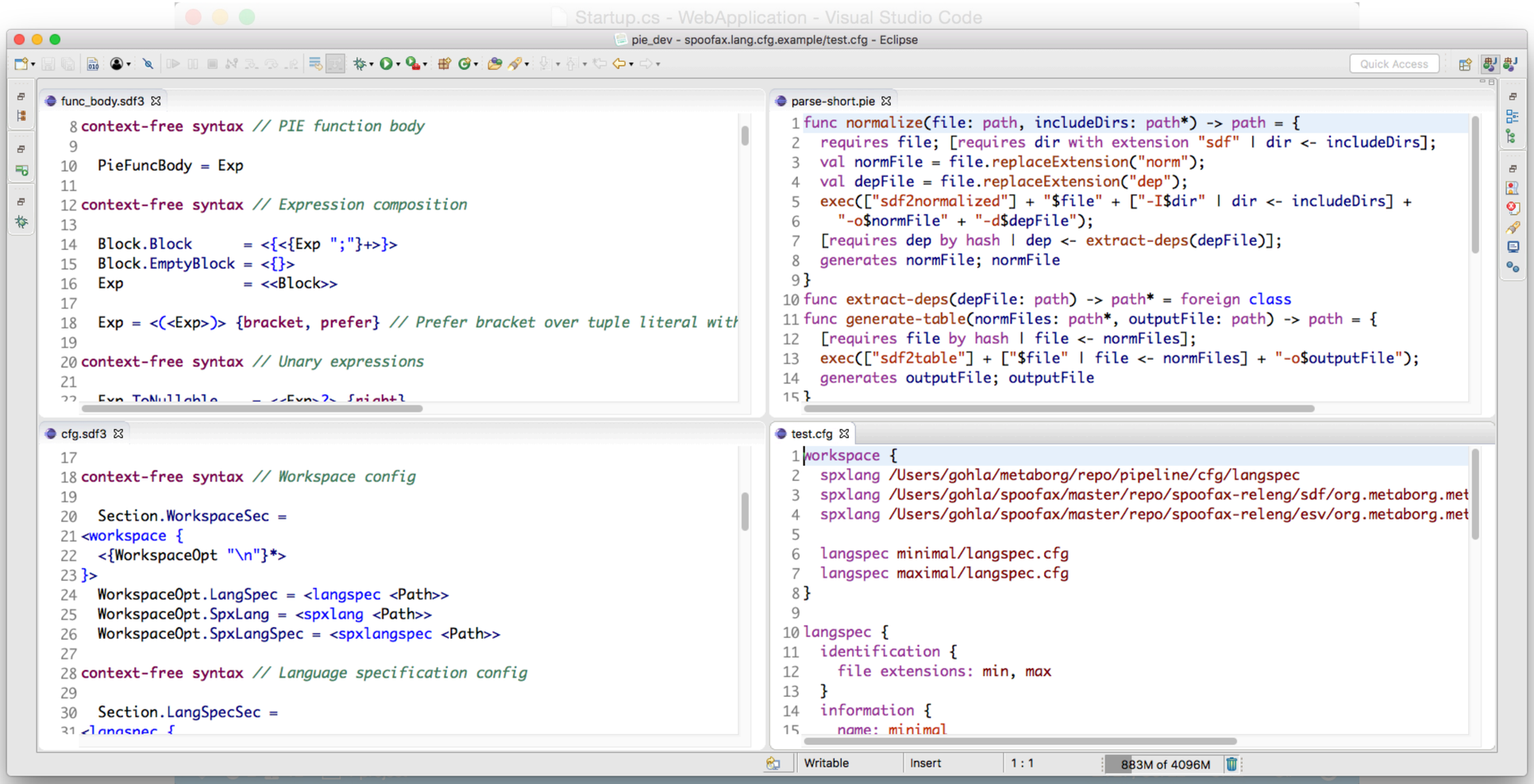

```
IDIR =../include
CC=gcc
CFLAGS=-I$(IDIR)

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)/*.o *~ core $(INCDIR)/*~
```

Interactive Pipeline: Immediate Feedback

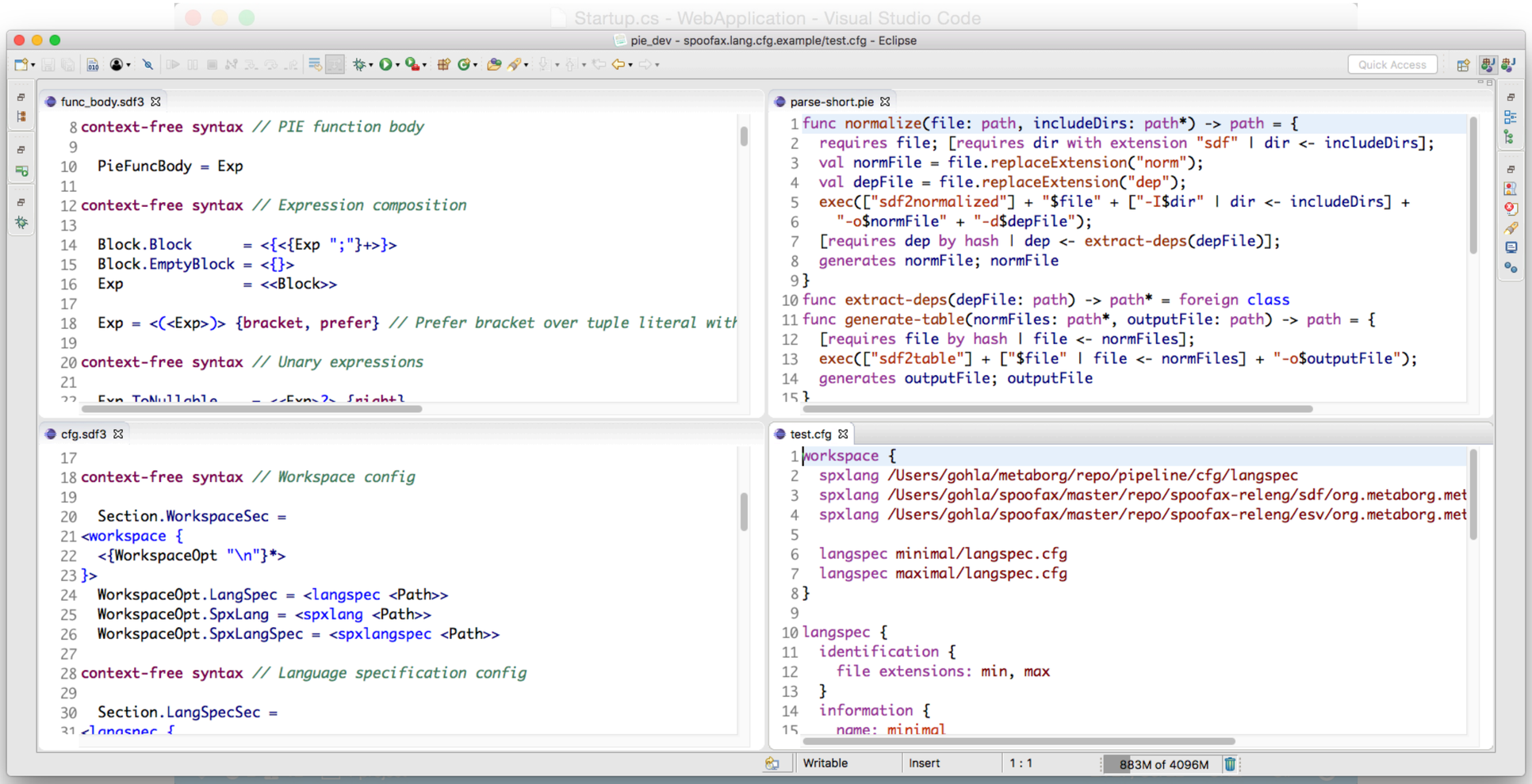
Interactive Pipeline: Immediate Feedback



Interactive Pipeline: Immediate Feedback



Interactive Pipeline: Immediate Feedback



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:

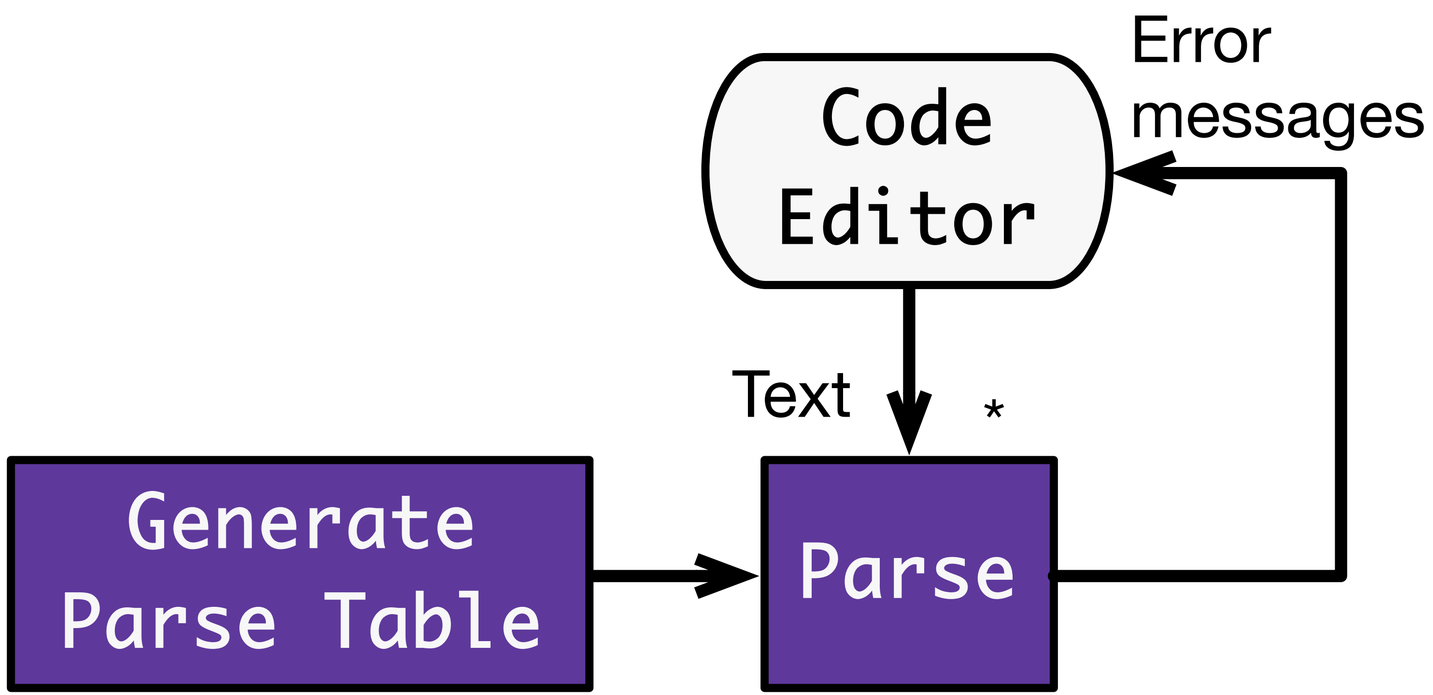
- Incremental
- Correct
- Persistent
- Expressive
- Easy to develop

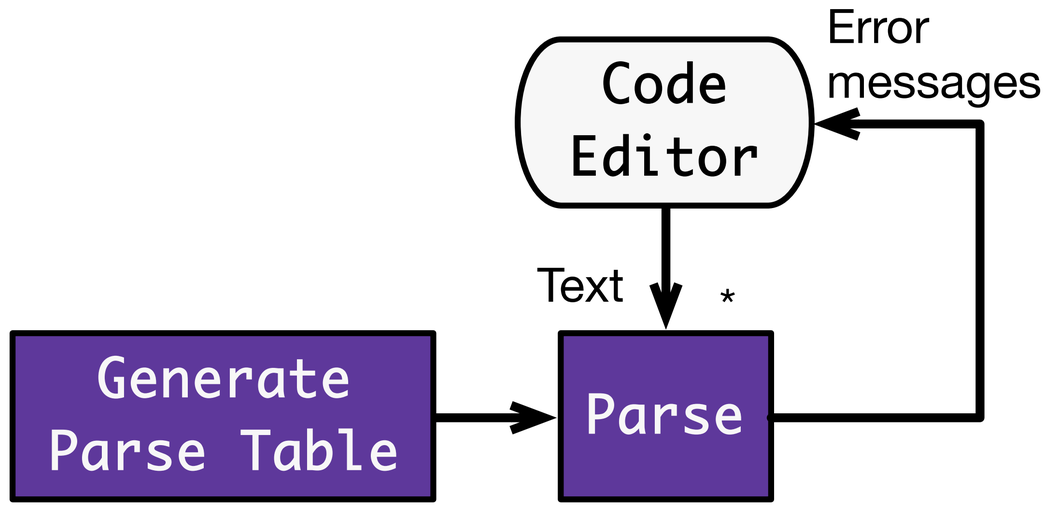
PIE: Pipelines for Interactive Environments

DSL

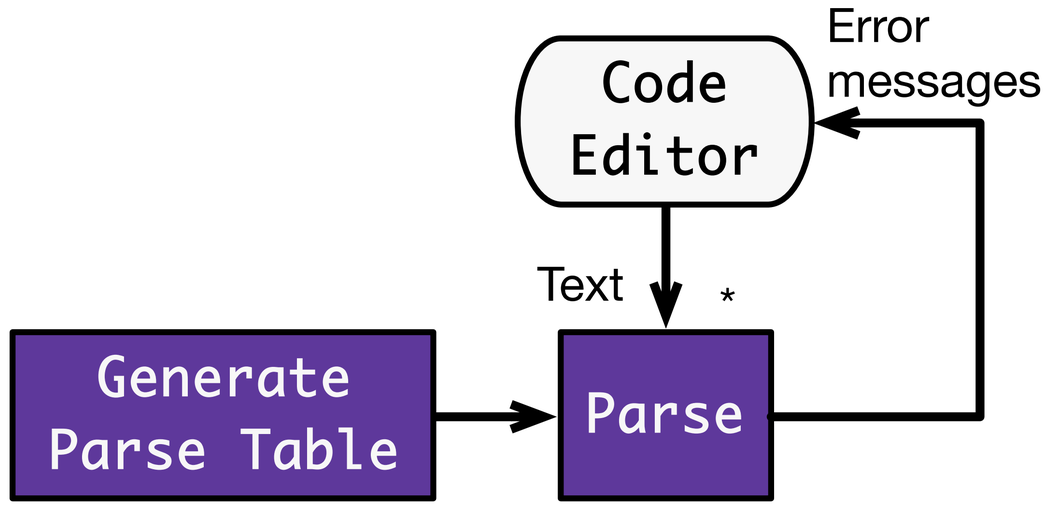
API

Runtime


Code
Editor
Error
messages
Text
*
Generate
Parse Table
Parse

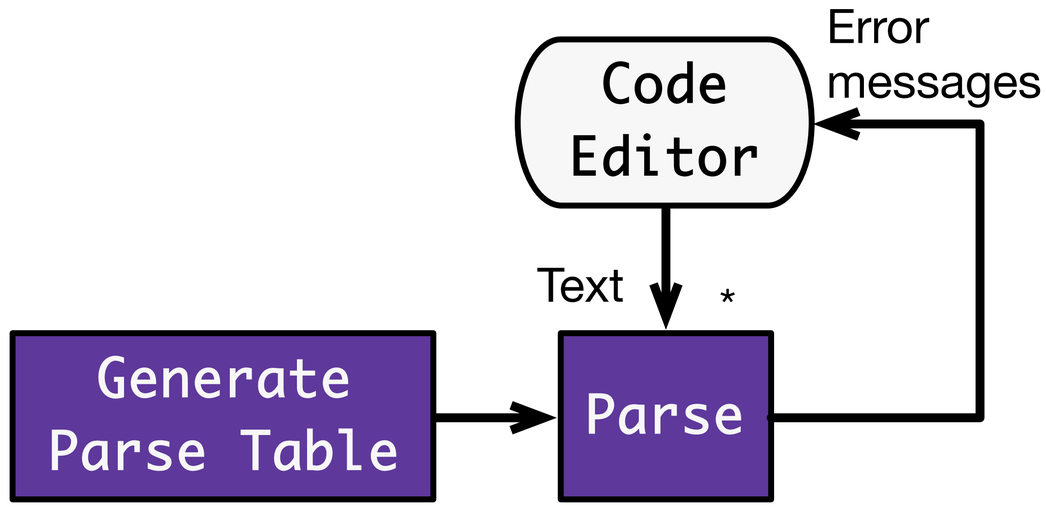

Code
Editor
Error
messages
Text
*
Generate
Parse Table
Parse

PIE DSL code


Error
messages
Code
Editor
Text
*
Generate
Parse Table
Parse

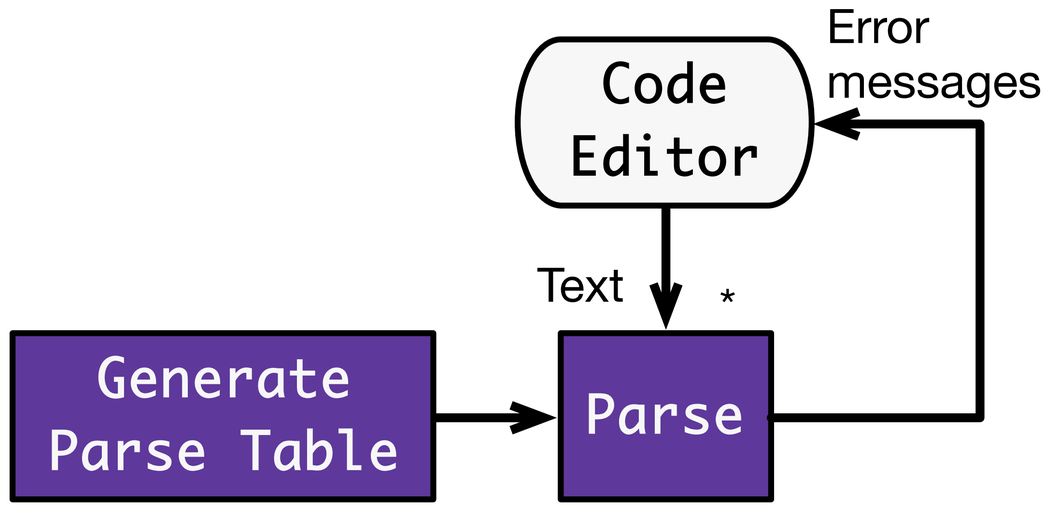
PIE DSL code

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

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

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



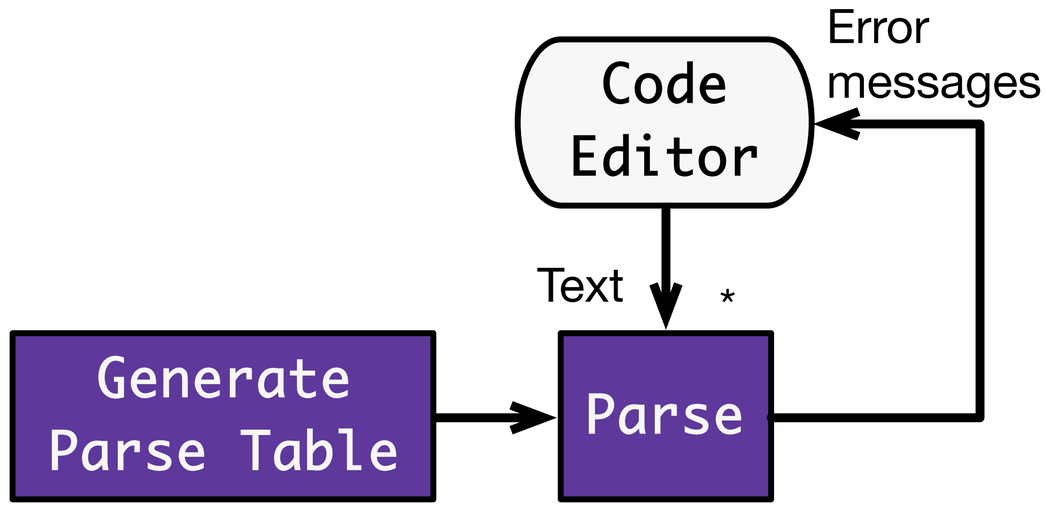
PIE DSL code

```
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*)
```



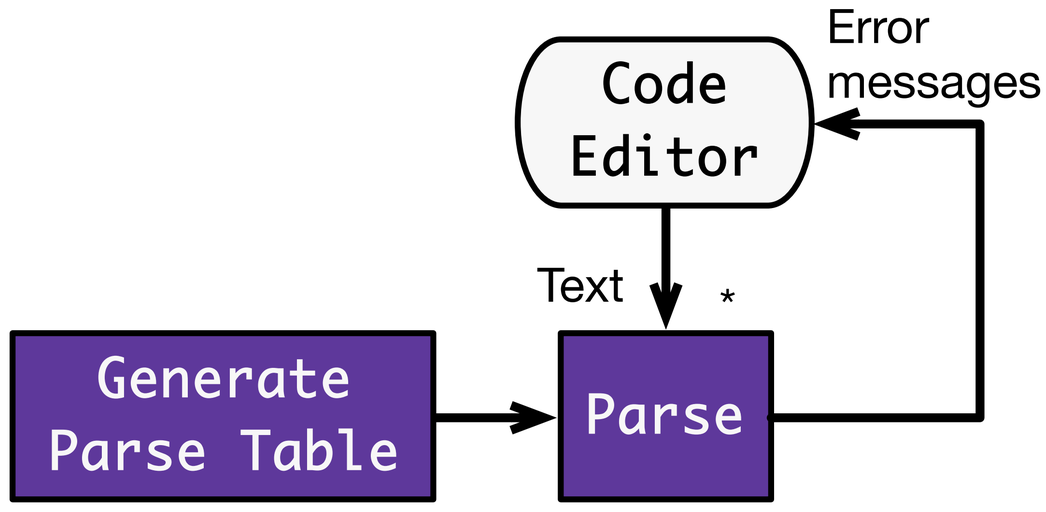
PIE DSL code

```
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*)
```



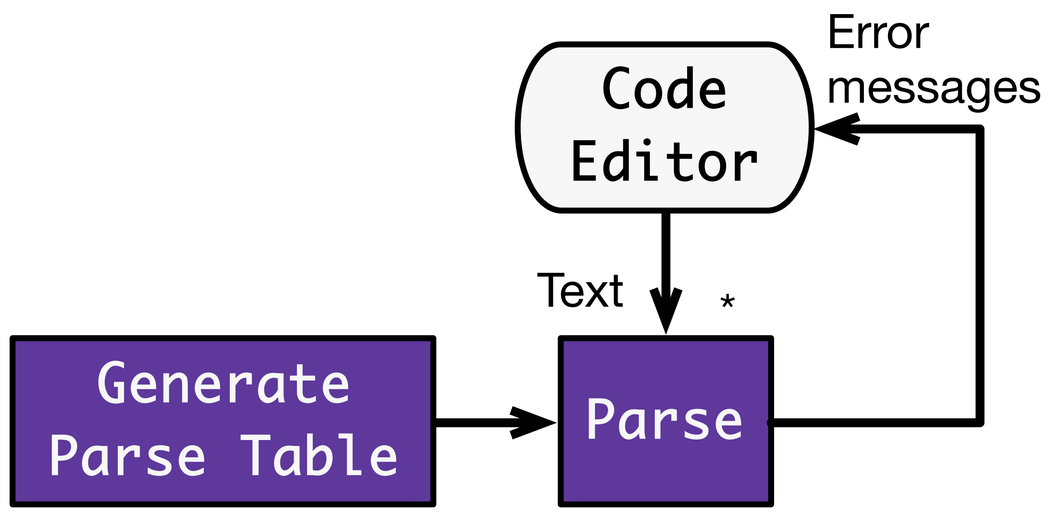
PIE DSL code

```
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*)
```



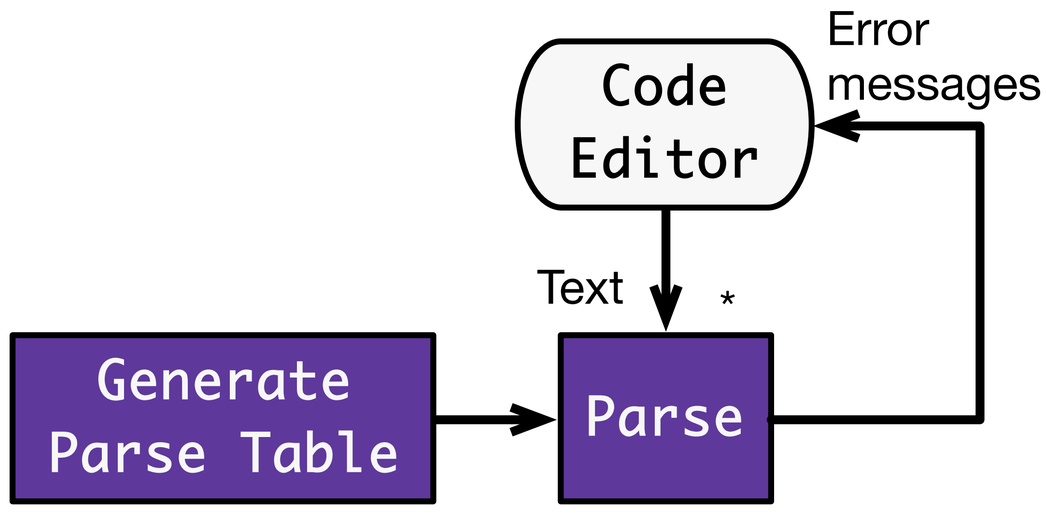
PIE DSL code

```
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*)
```



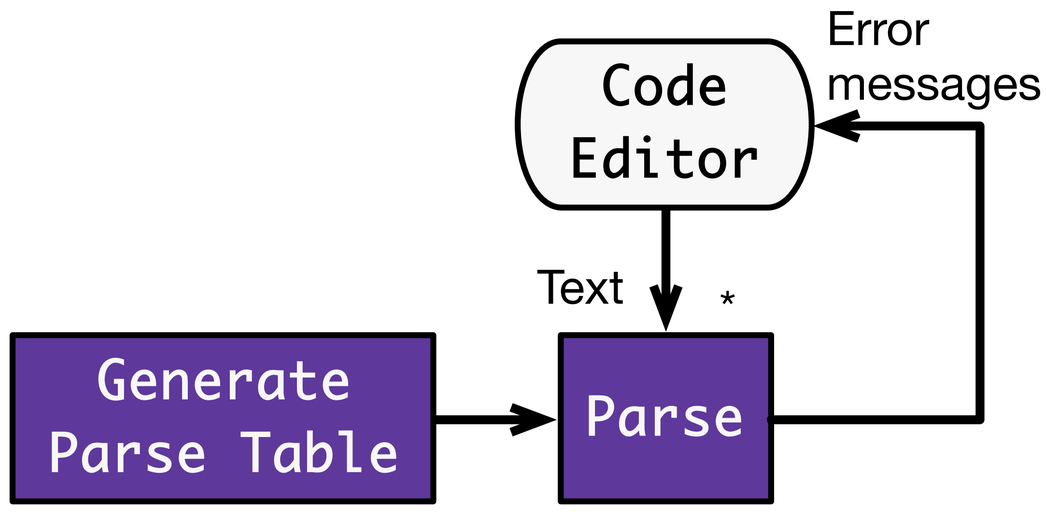
PIE DSL code

```
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*)
```



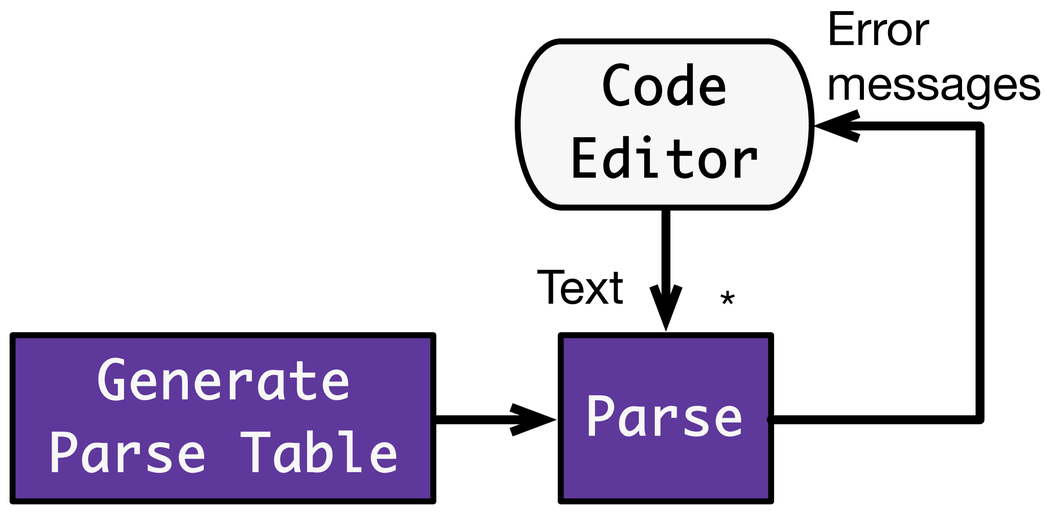
PIE DSL code

```
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*)
```



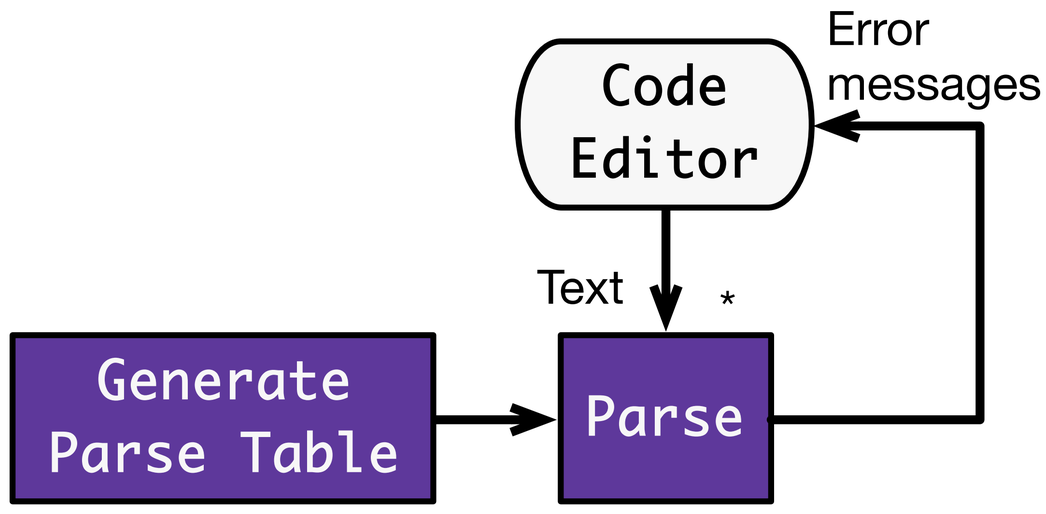
PIE DSL code

```
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*)
```



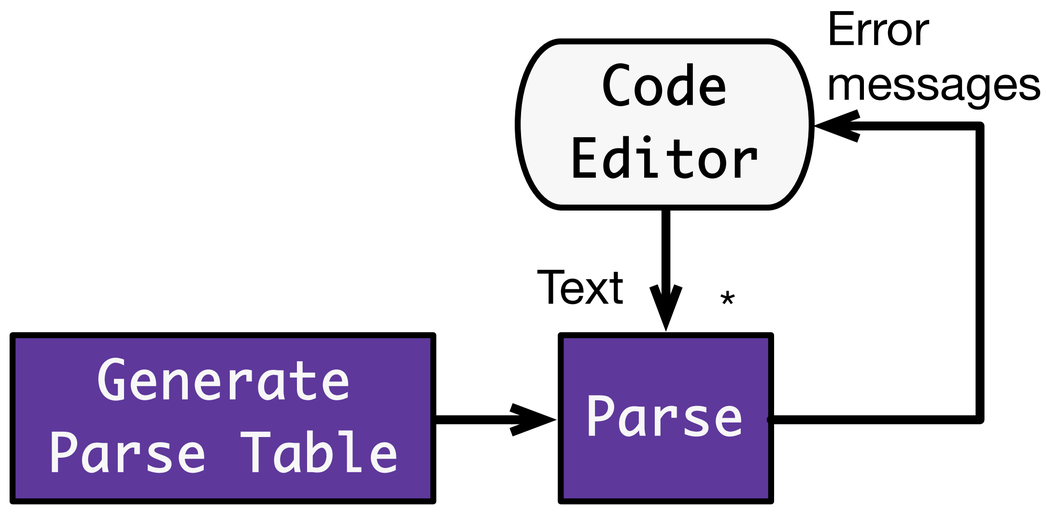
PIE DSL code

```
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
```

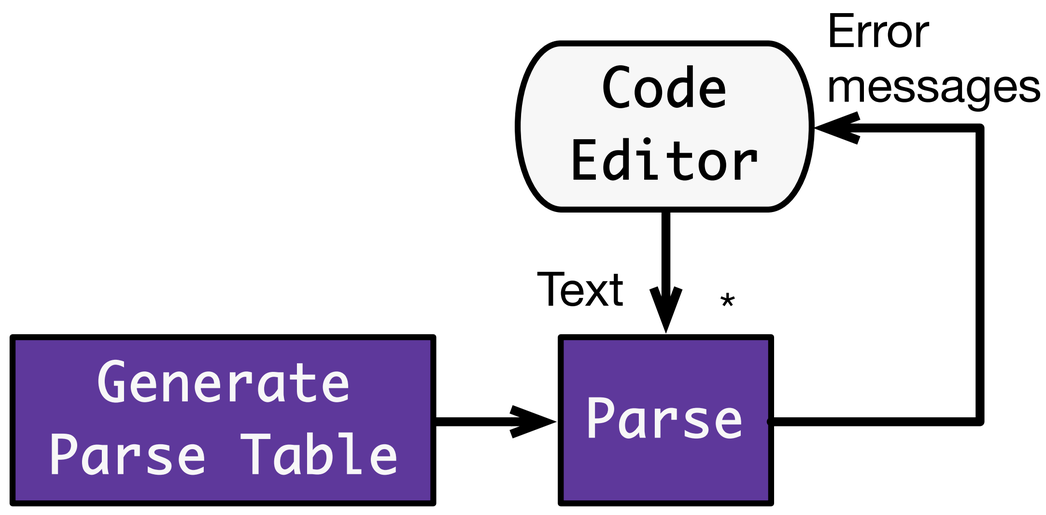


PIE DSL code

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

func compile-parse-table(defFile: pat
  requires defFile;
  val tblFile = defFile.replaceExtensio
  val depFile = defFile.replaceExtens
  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(depFile)];
  generates tblFile;
}
func parse(text: string, pt: path) -> (Ast, Msg*)
```

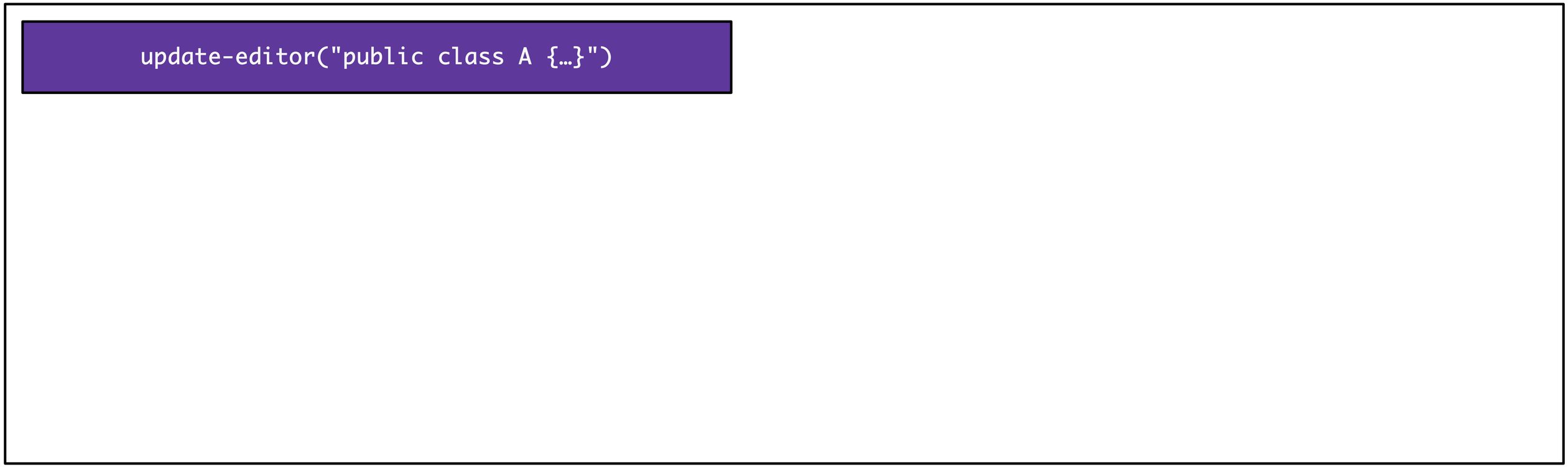
```
class Parse : Func<Parse.Input, Parse.Output> {
  data class Input(val text: String, val table: PPath)
  data class Output(val ast: Ast, val msgs: List<Msg>)
  override fun ExecContext.exec(input: Input): Output {
    require(input.table)
    val result = Api.parse(input.table, input.text)
    return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

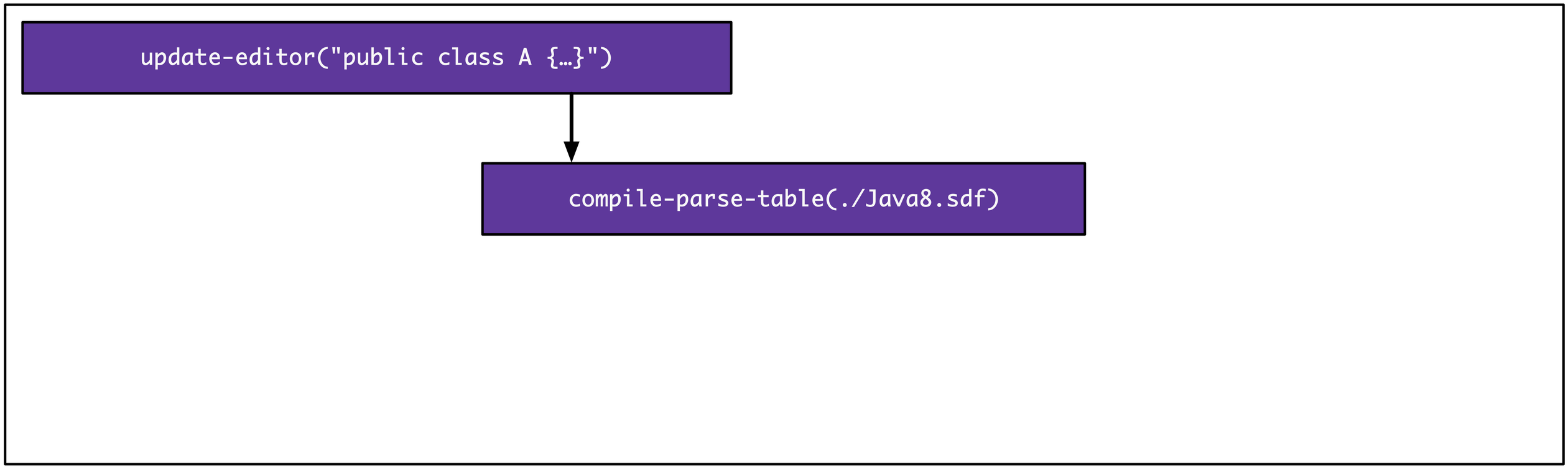
Execute and build dependency graph




```
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(depFile)];
  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)
  }
}
```

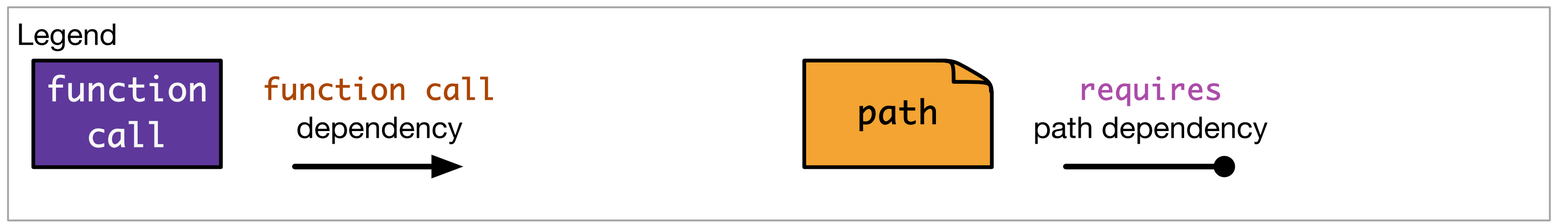
Execute and build dependency graph

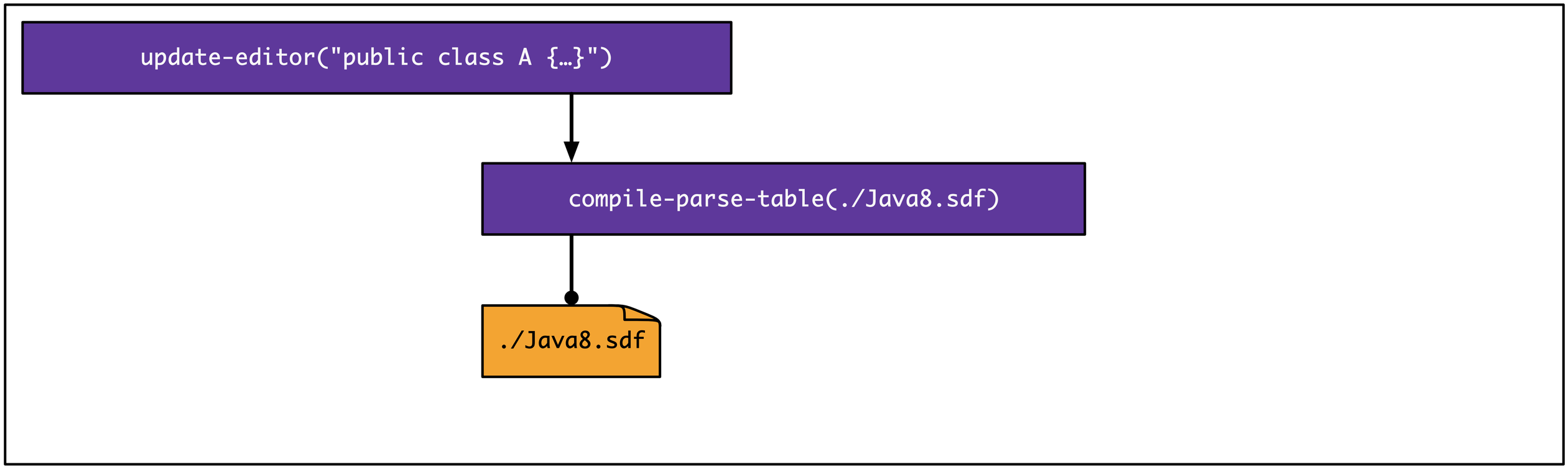



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

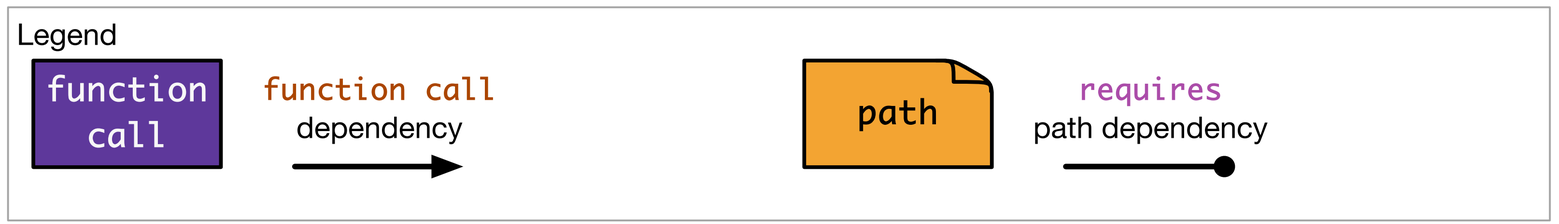


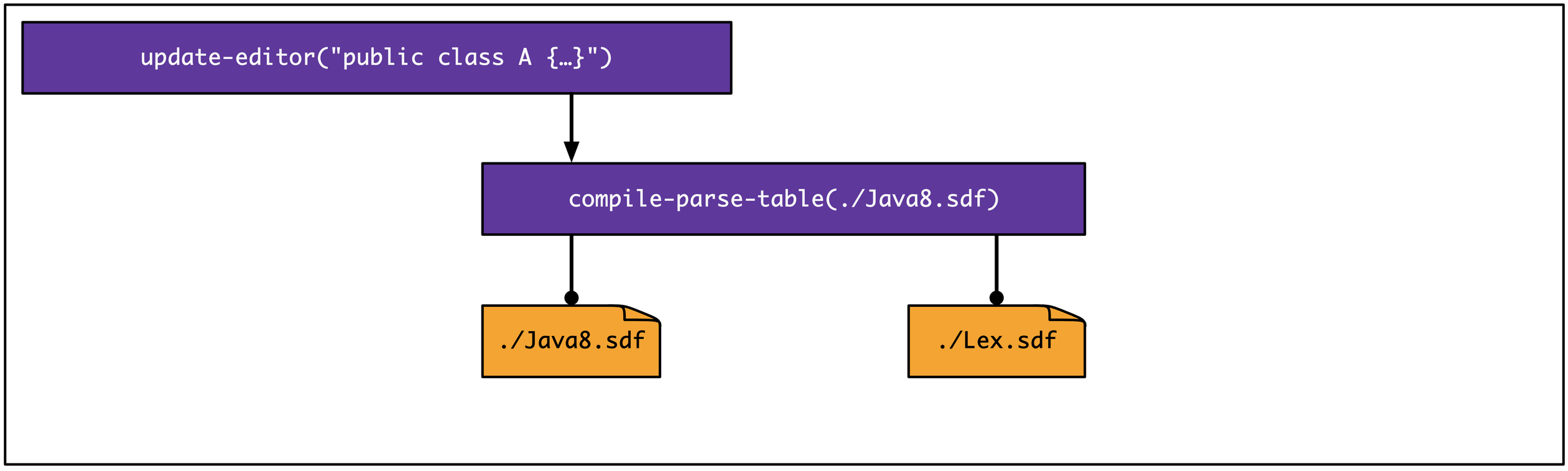



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

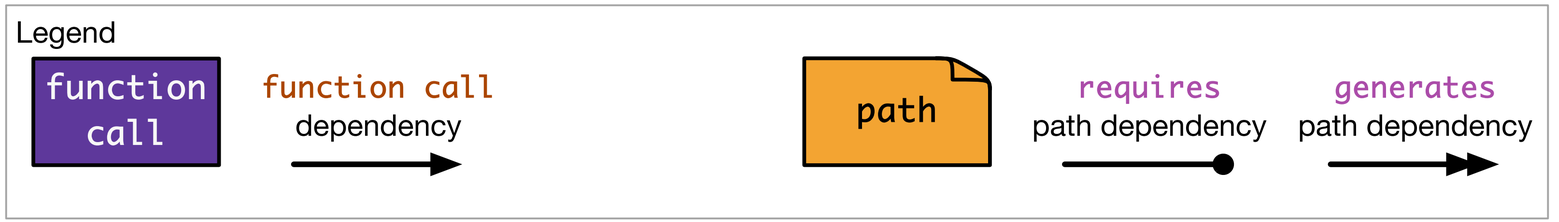


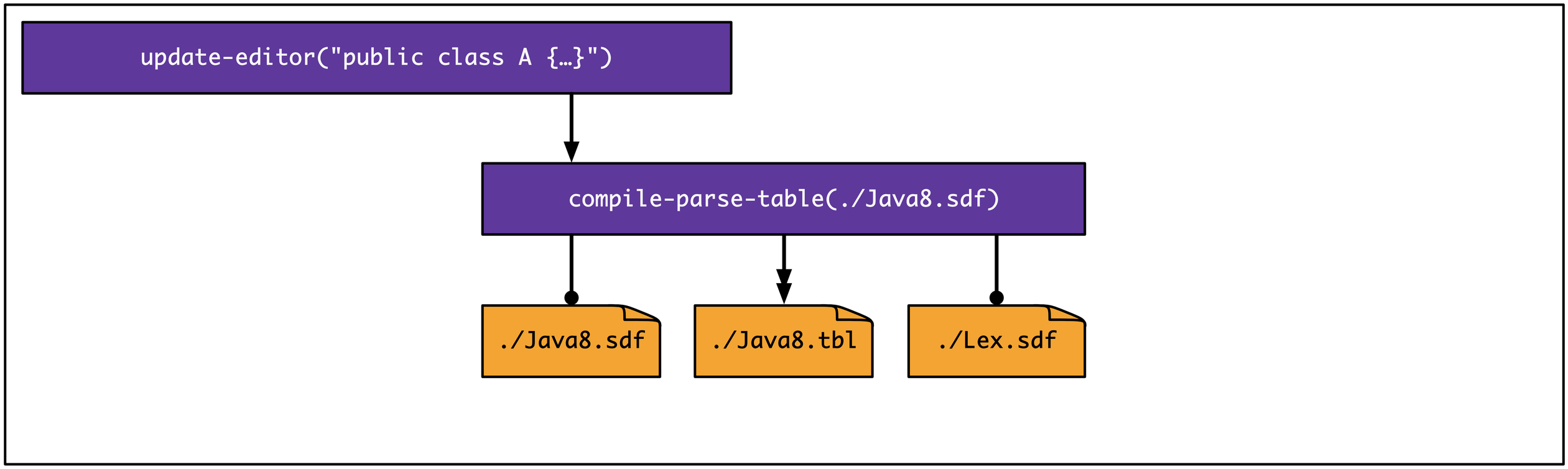



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

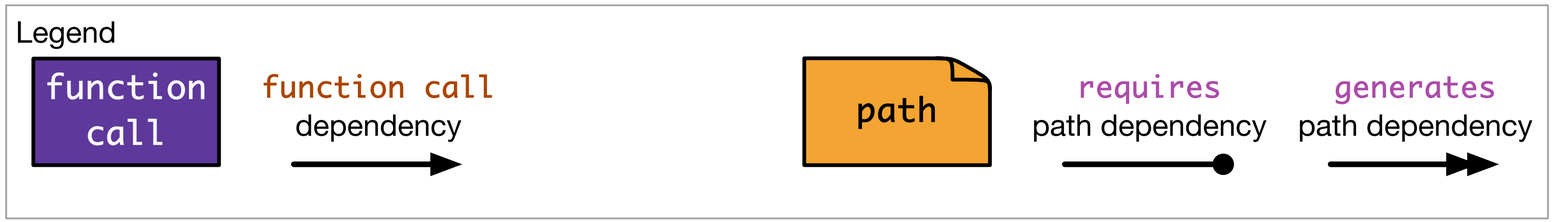


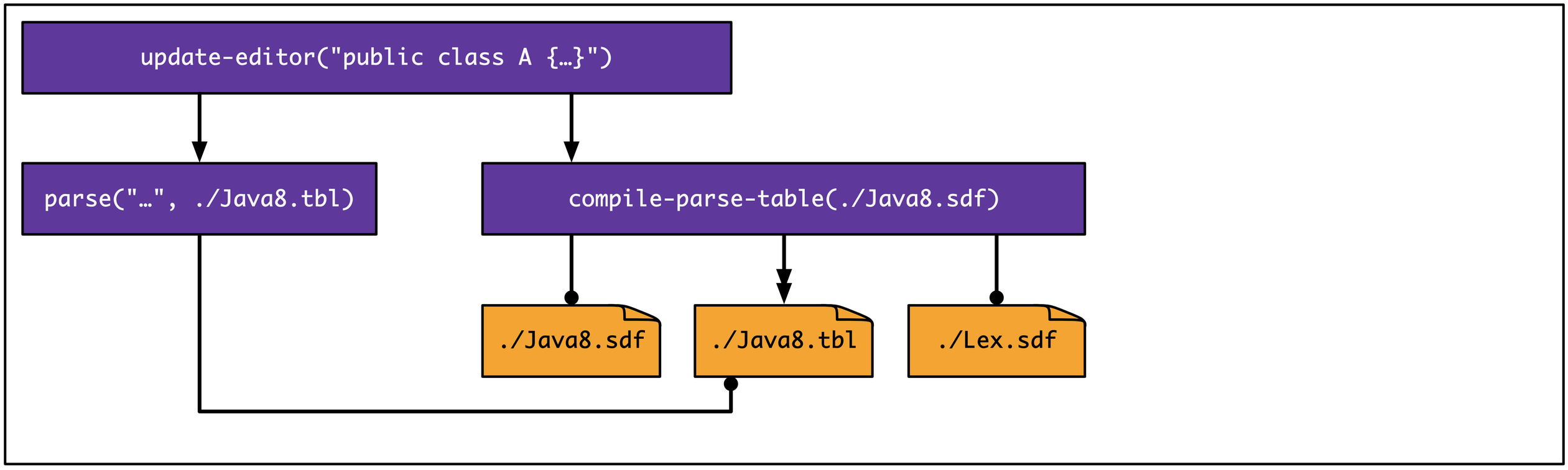



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

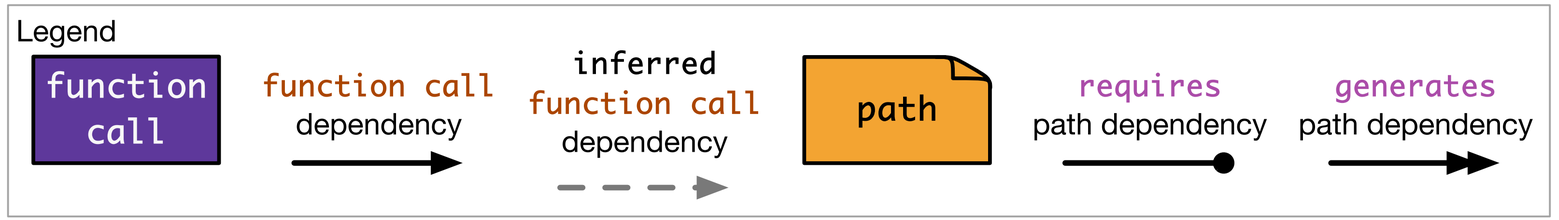


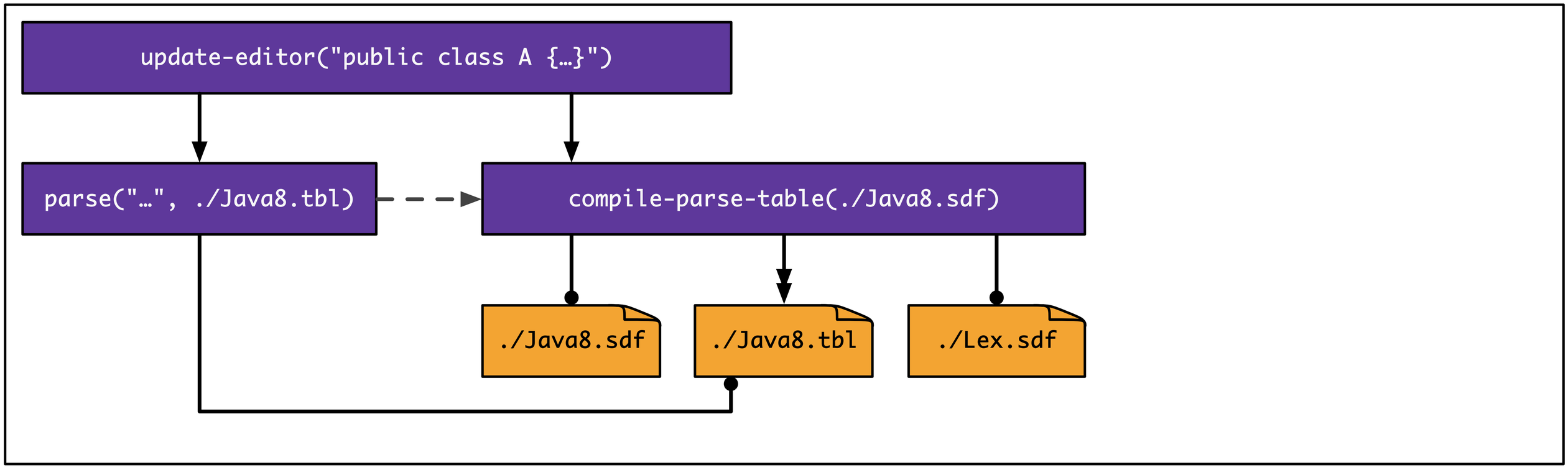



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

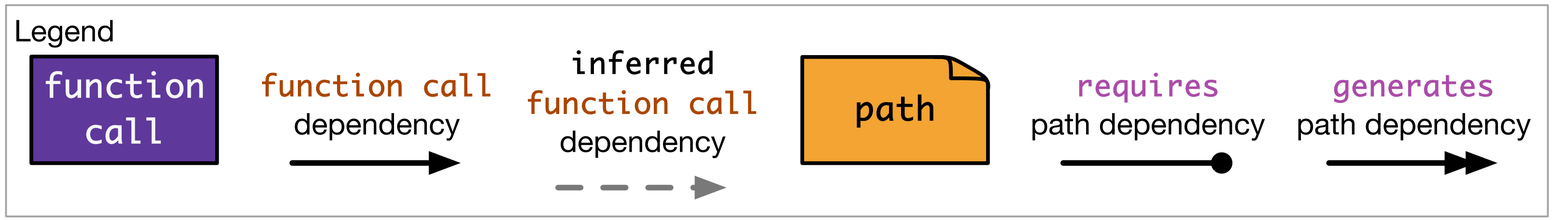


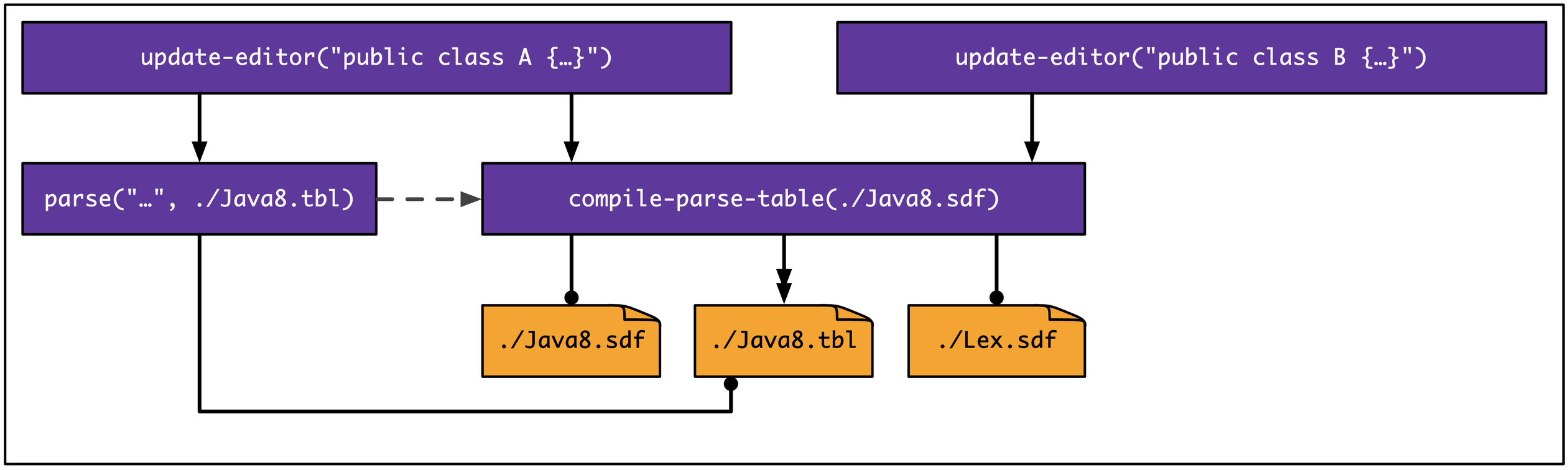



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

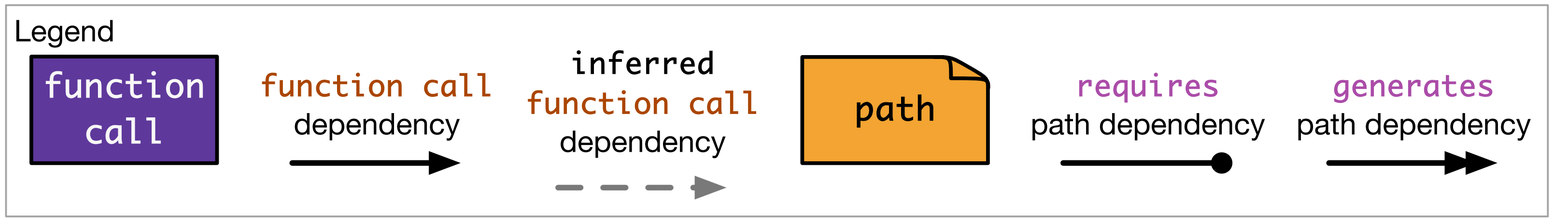


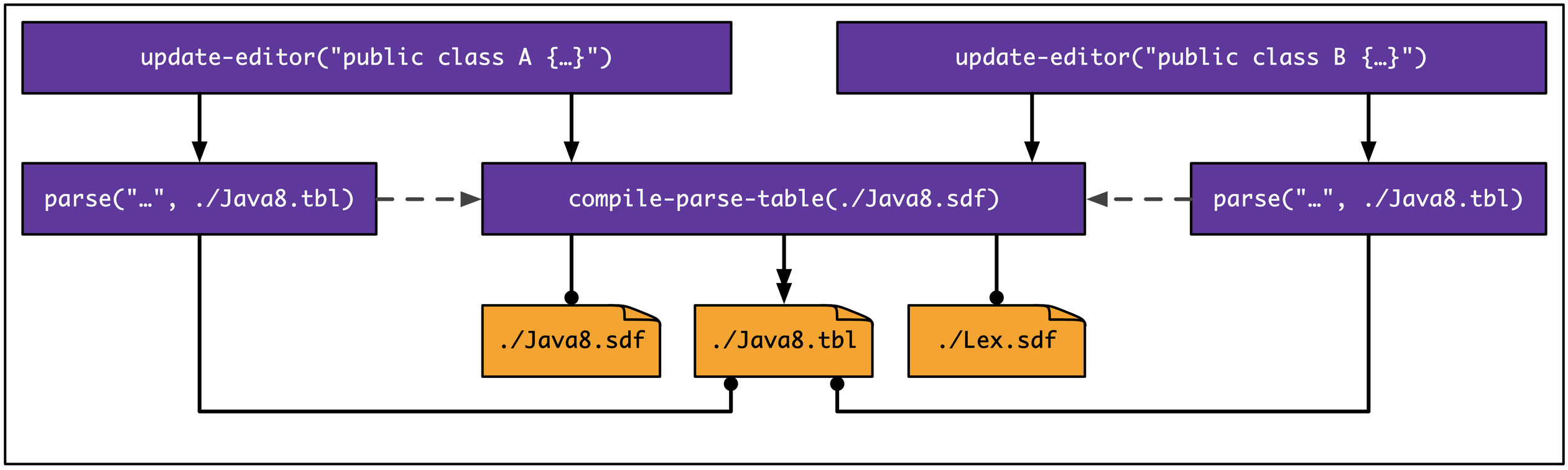



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

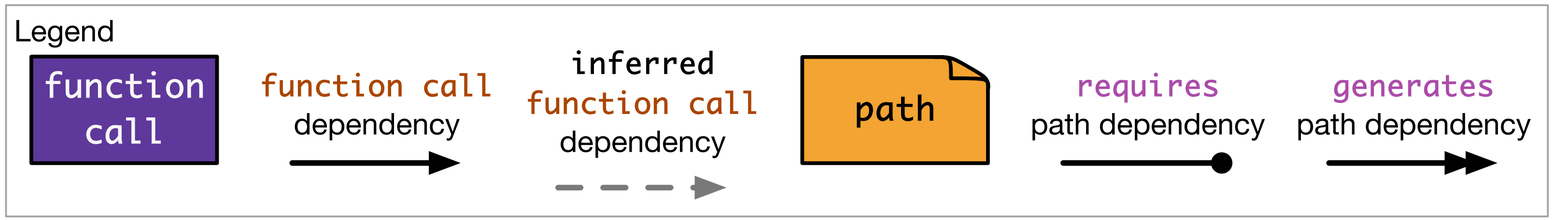


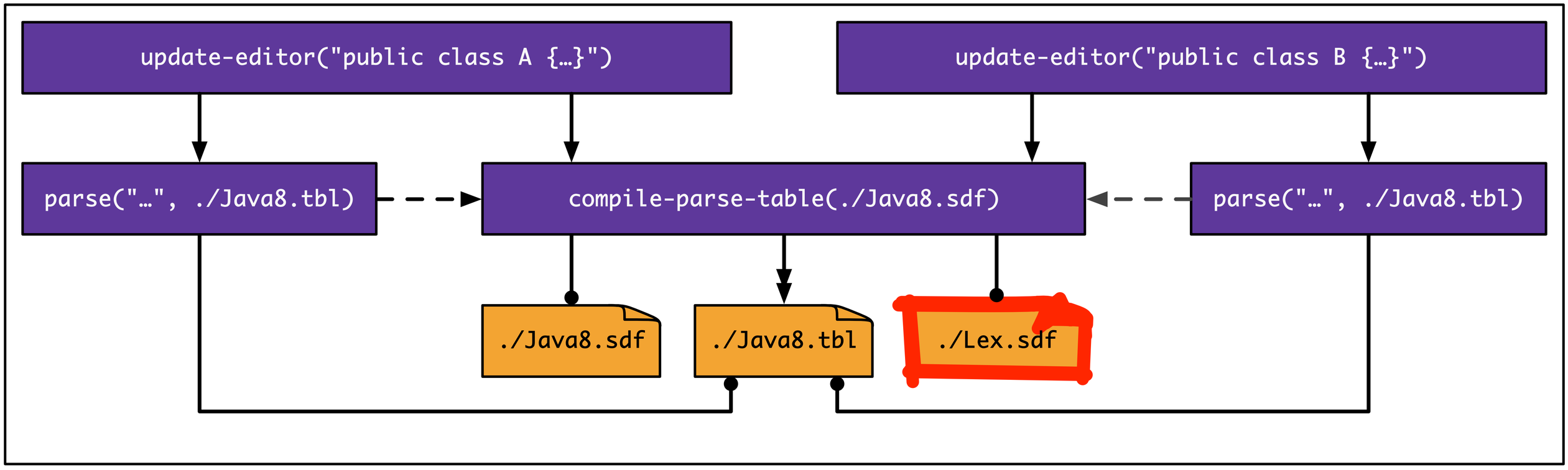



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

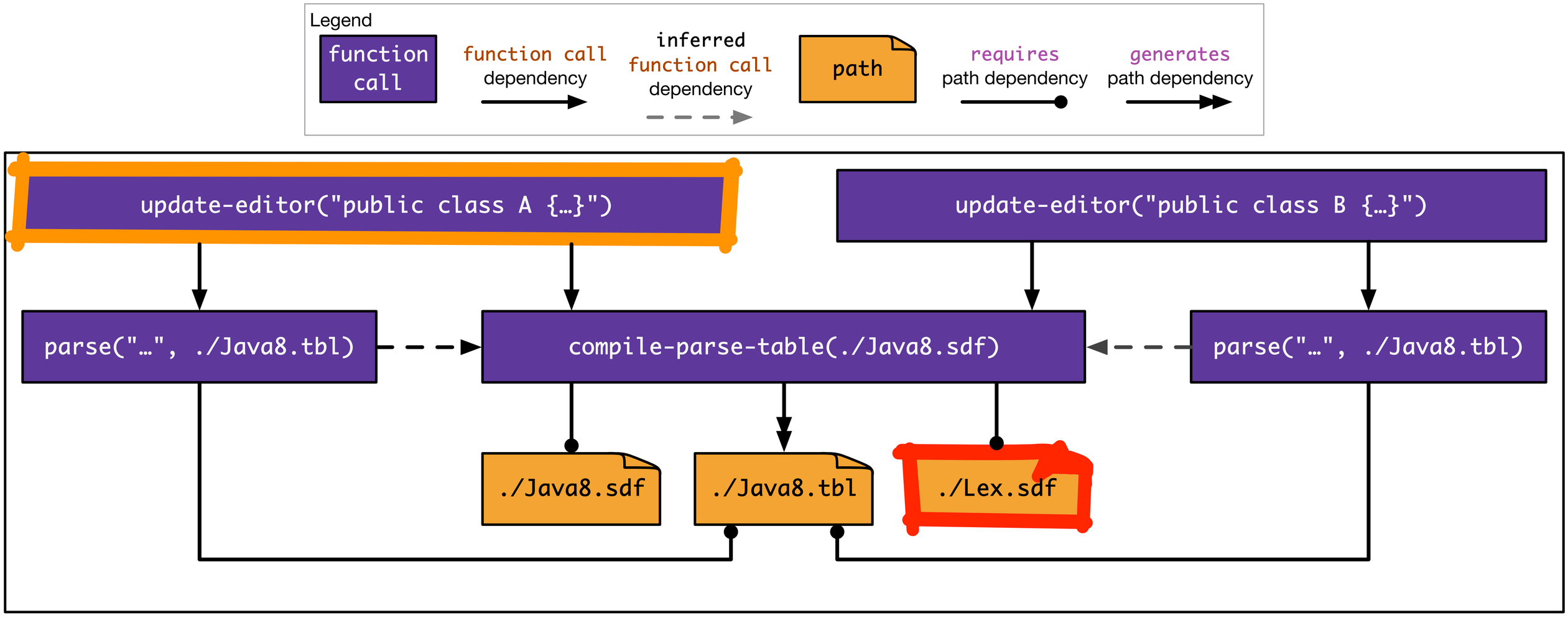





```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

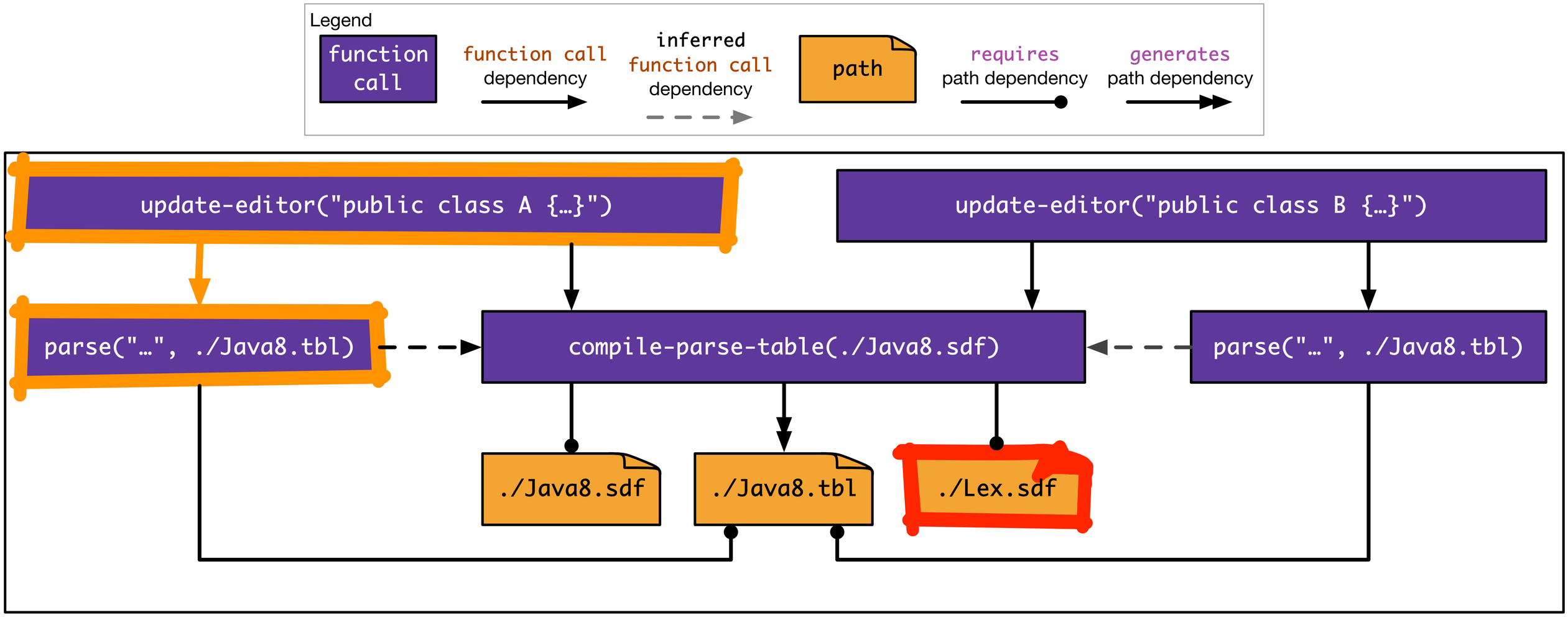
Incremental re-execution with changes




```
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(depFile)];
  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)
  }
}
```

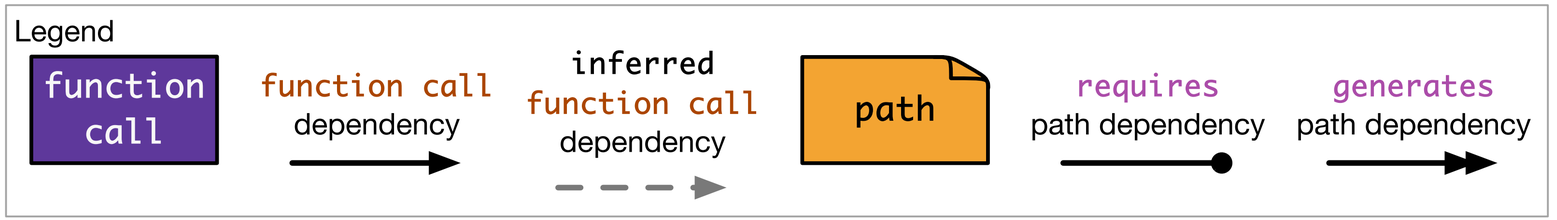
Incremental re-execution with changes

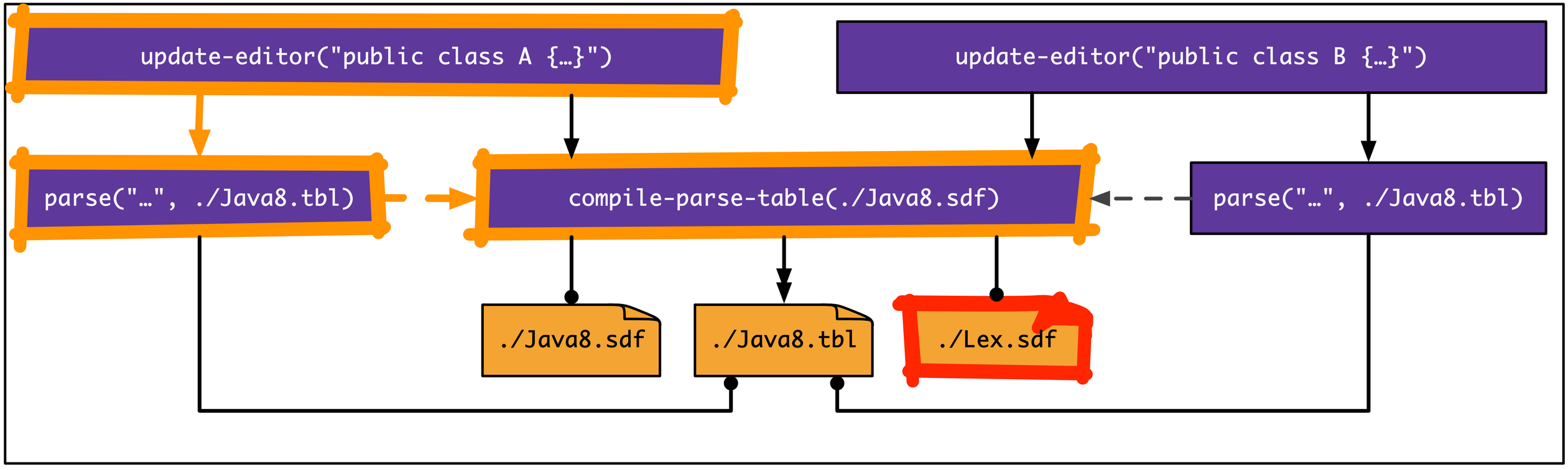



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

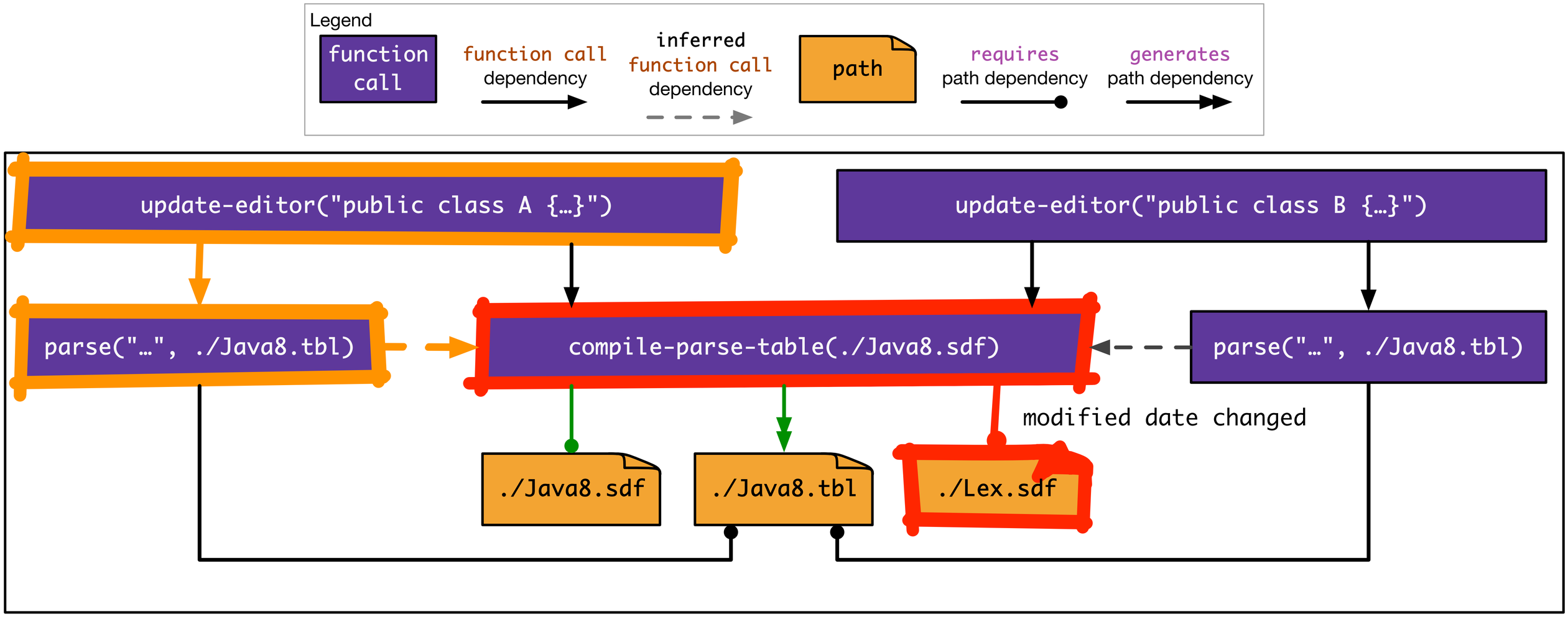





```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

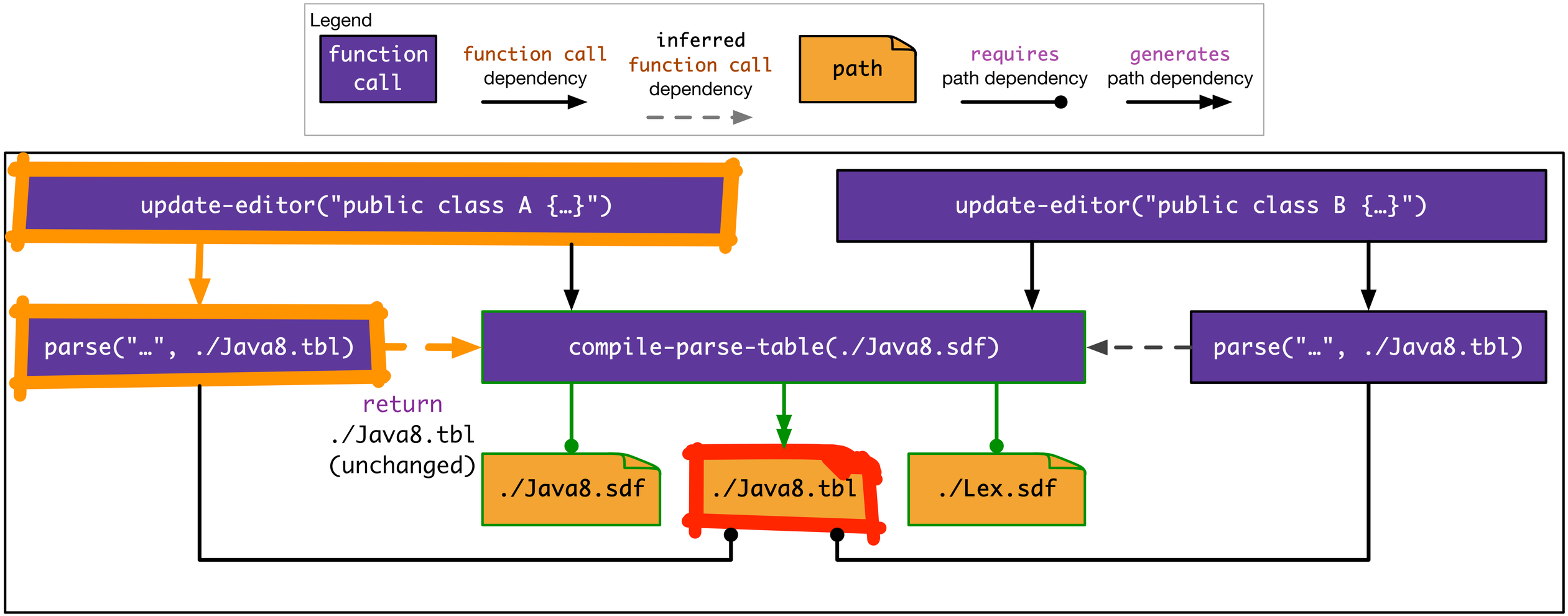
Incremental re-execution with changes




```
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(depFile)];
  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)
  }
}
```

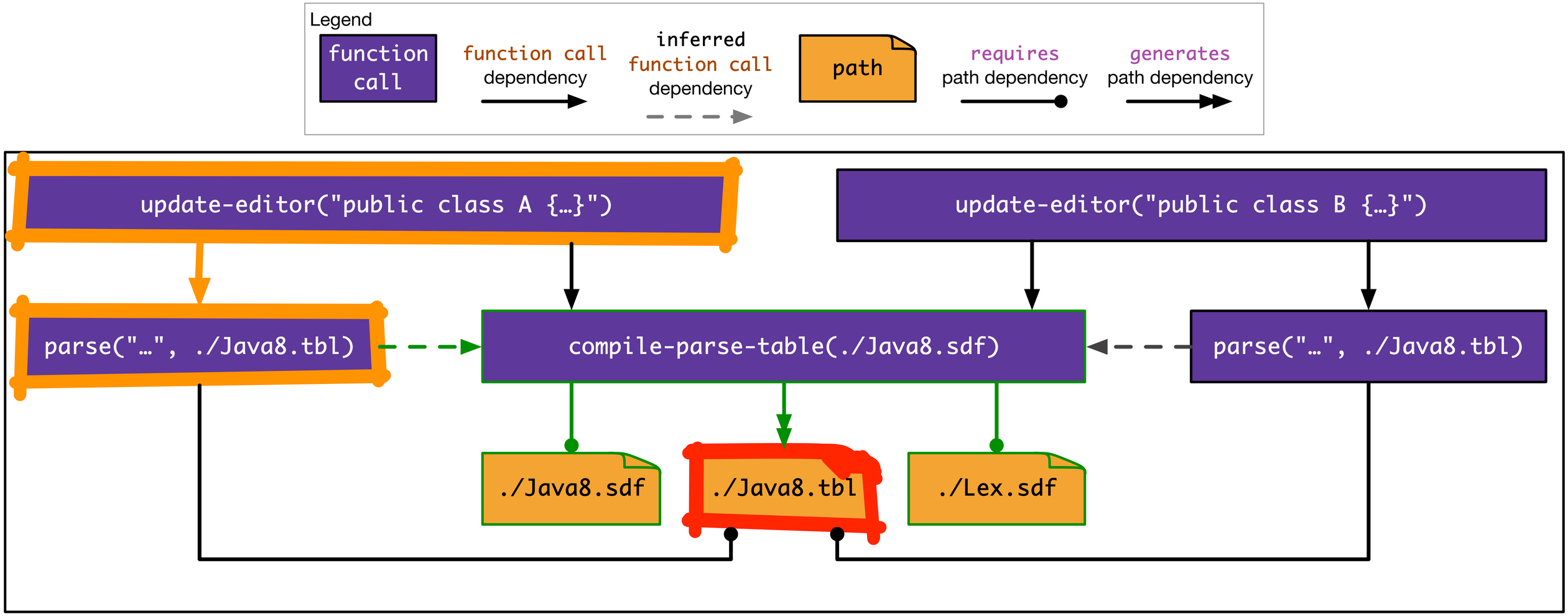
Incremental re-execution with changes




```
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(depFile)];
  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)
  }
}
```

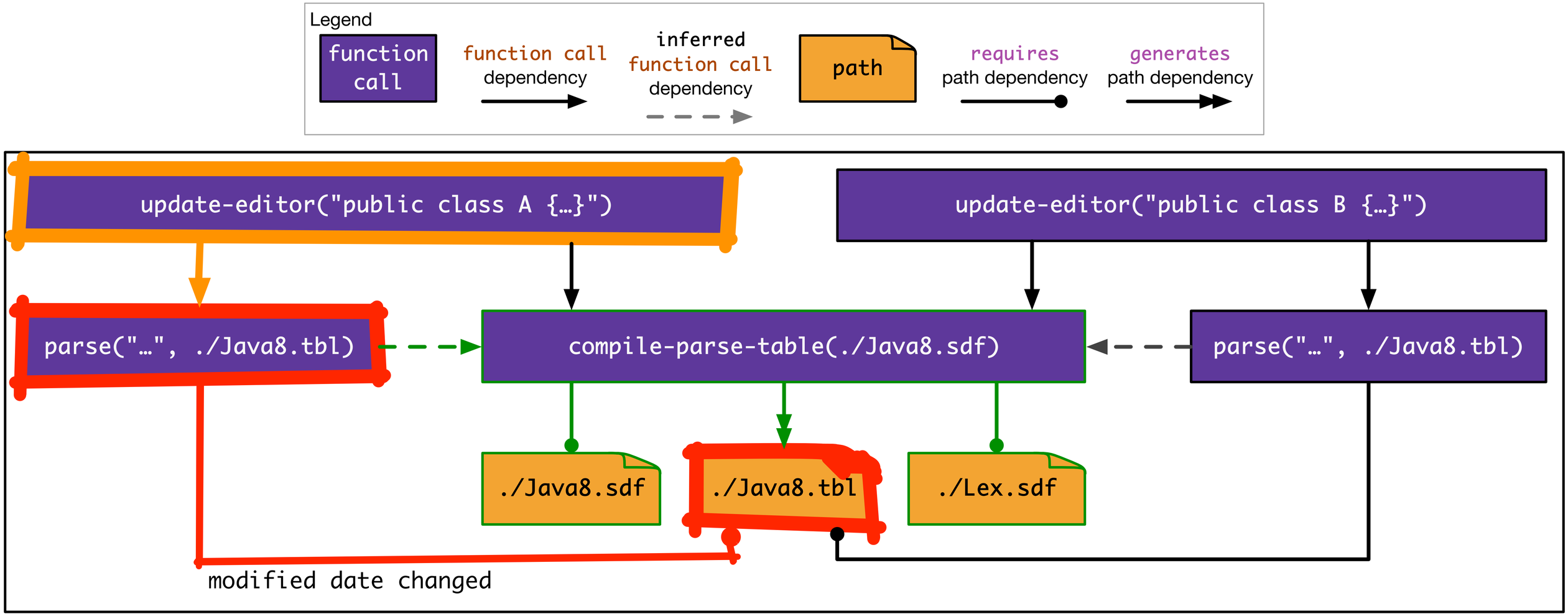
Incremental re-execution with changes




```
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(depFile)];
  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)
  }
}
```

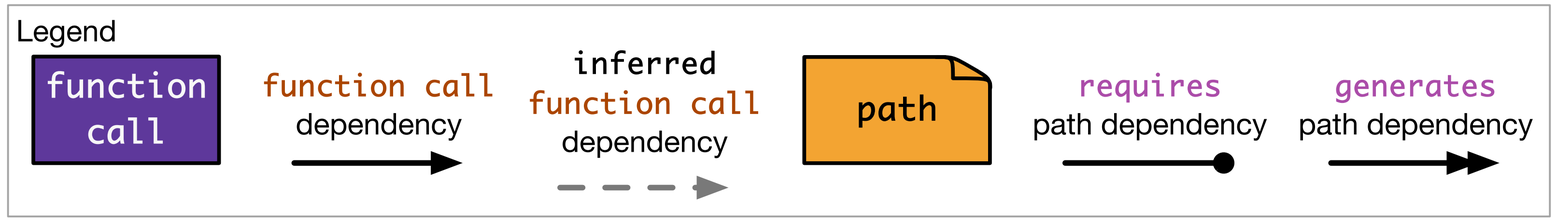
Incremental re-execution with changes

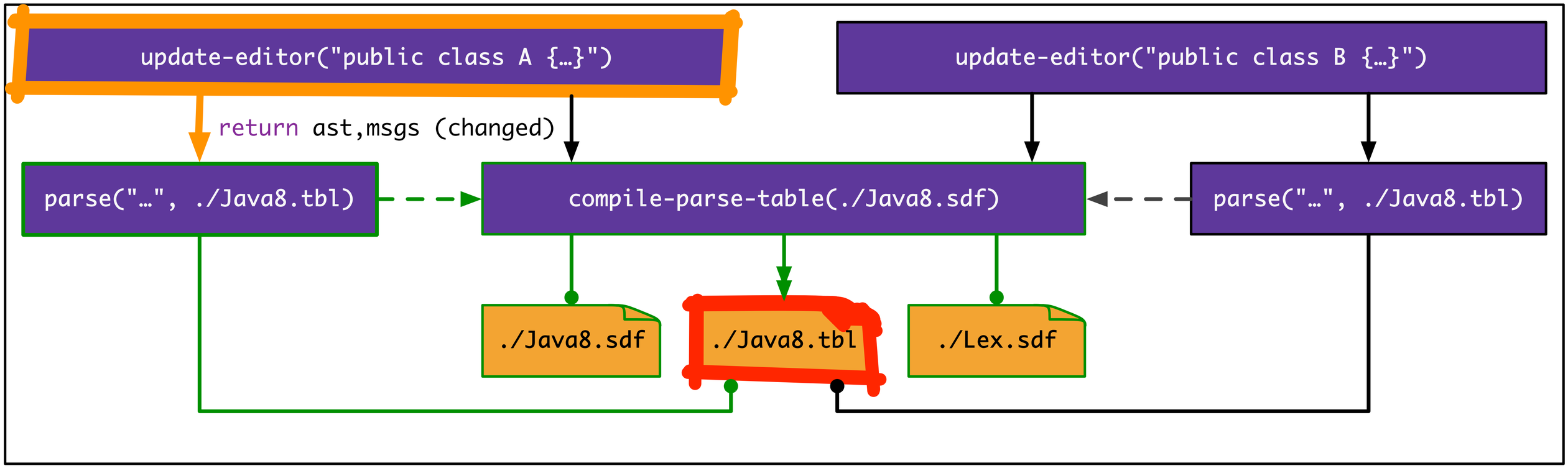



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

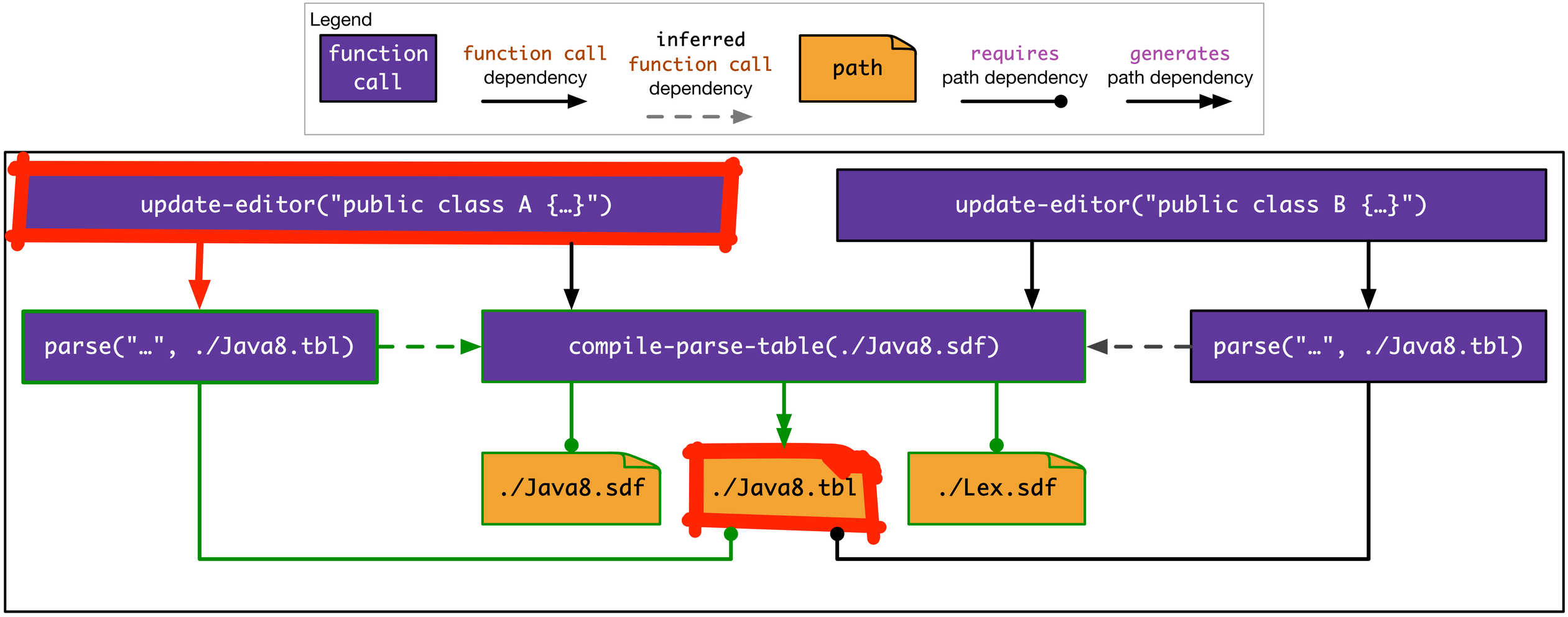





```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

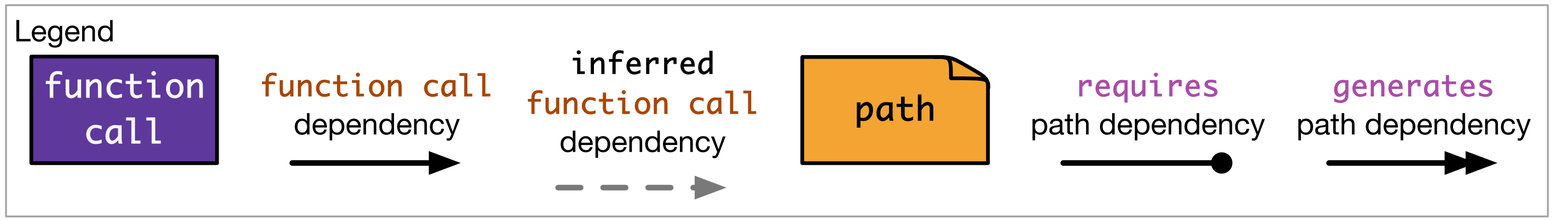
Incremental re-execution with changes

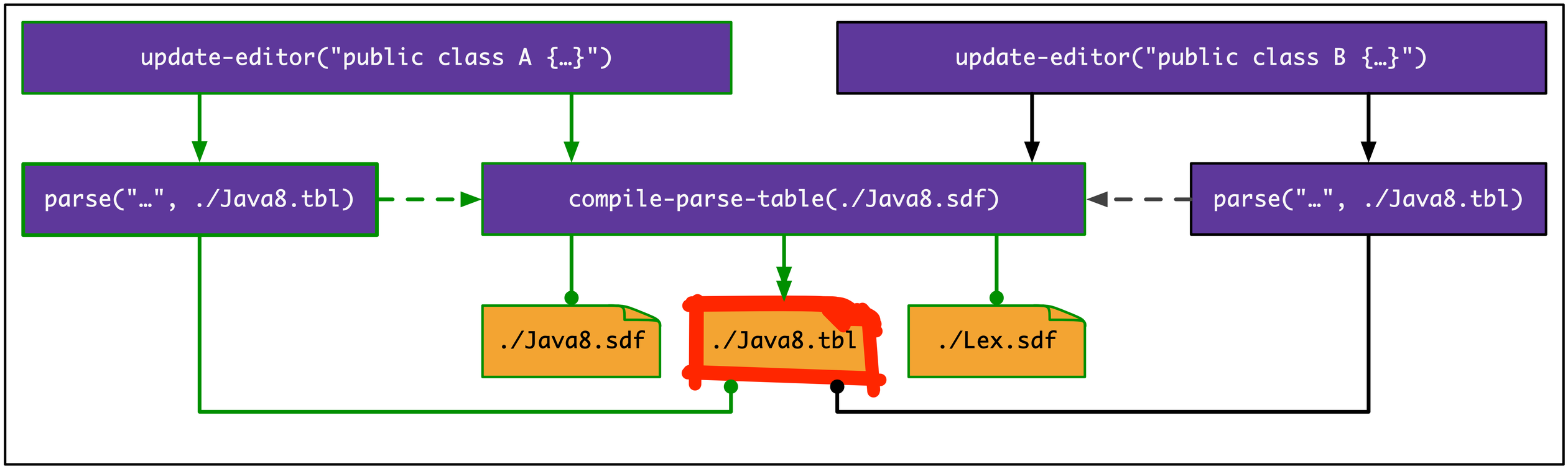



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

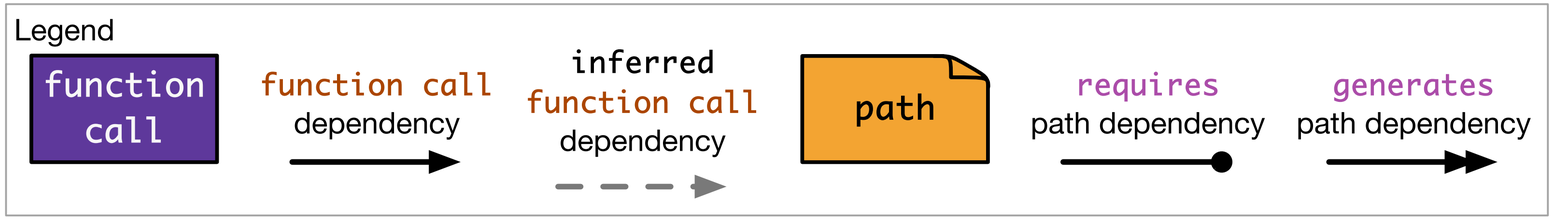


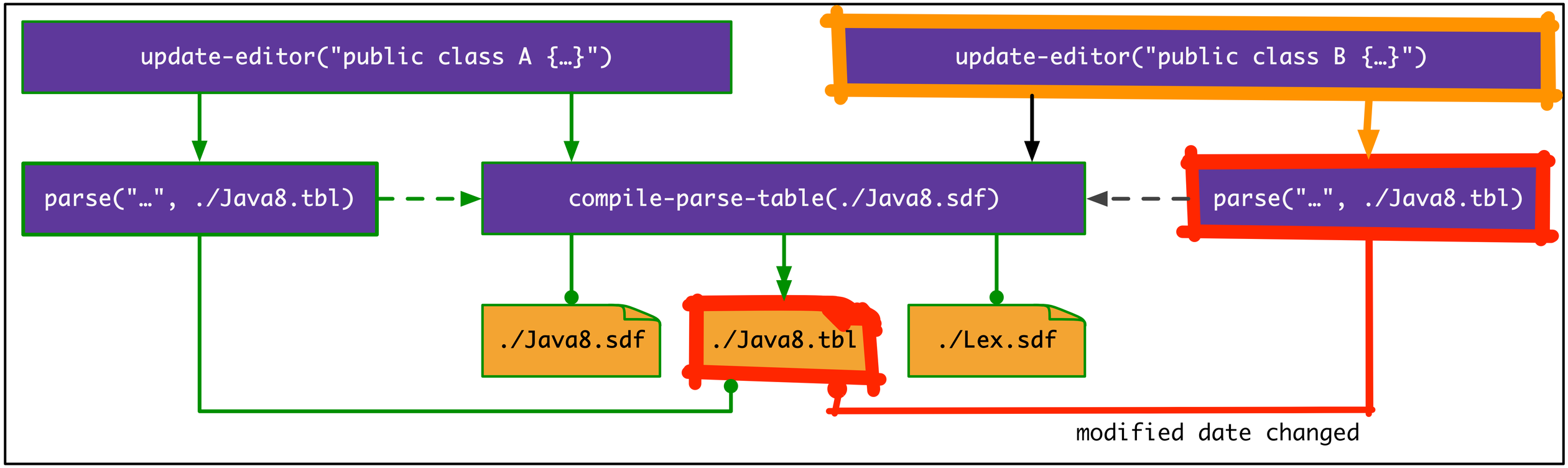



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

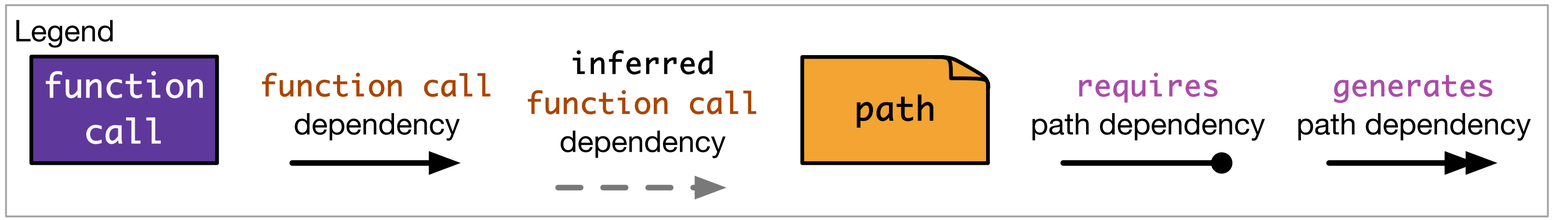


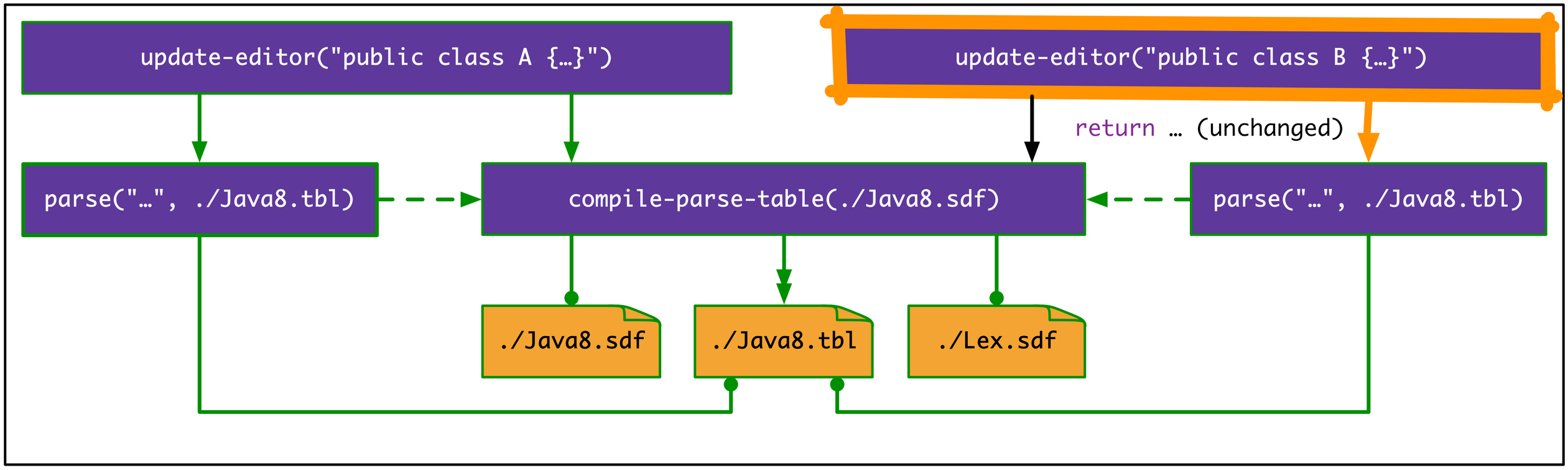



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

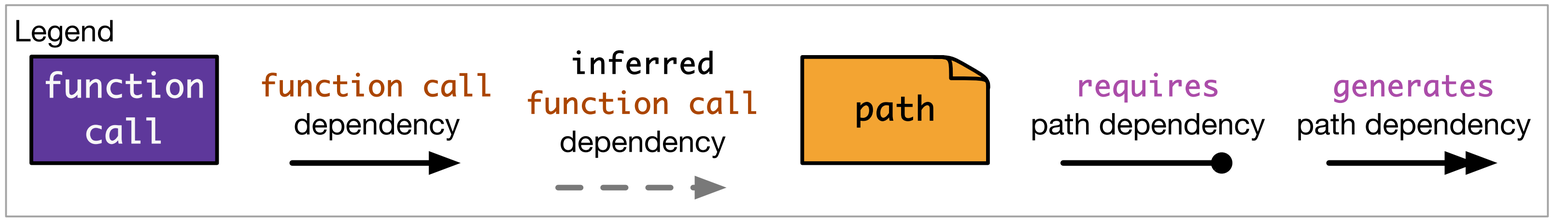


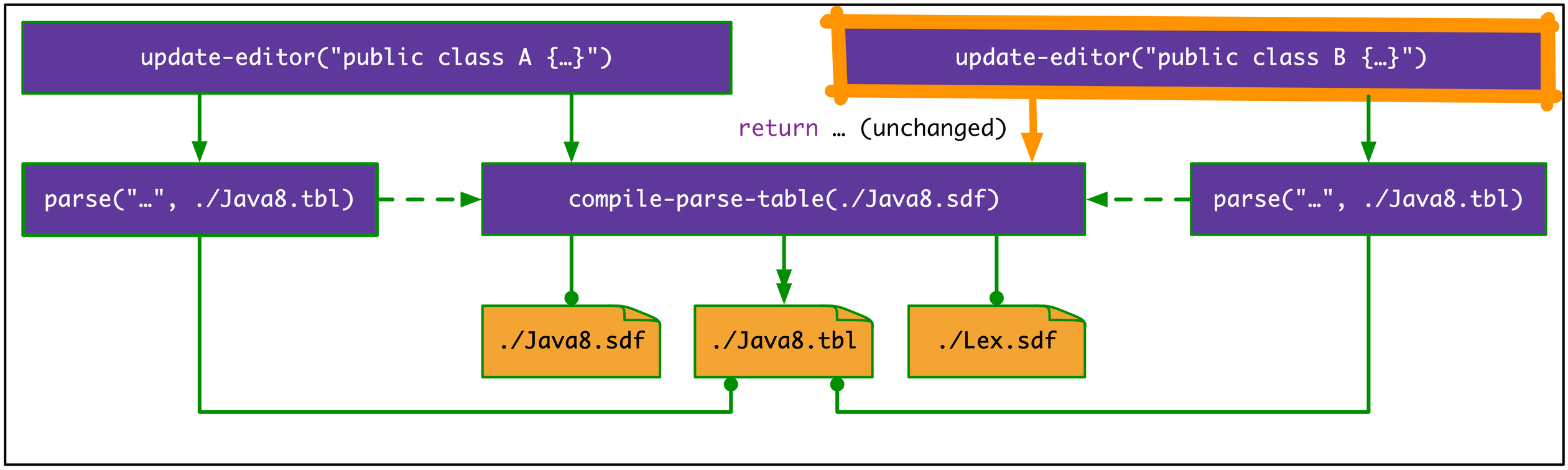



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

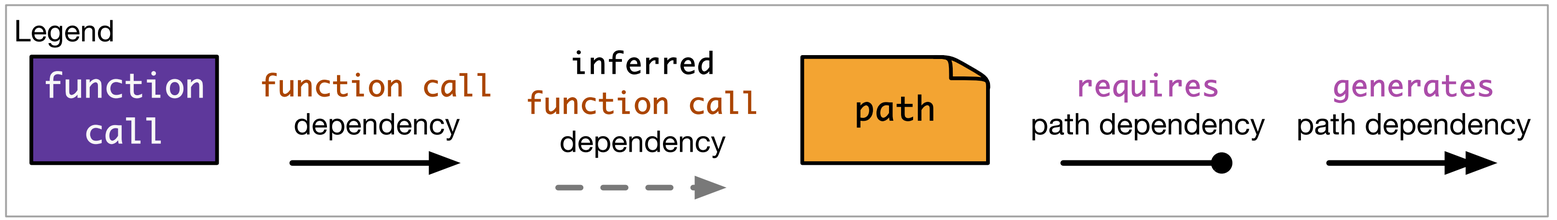


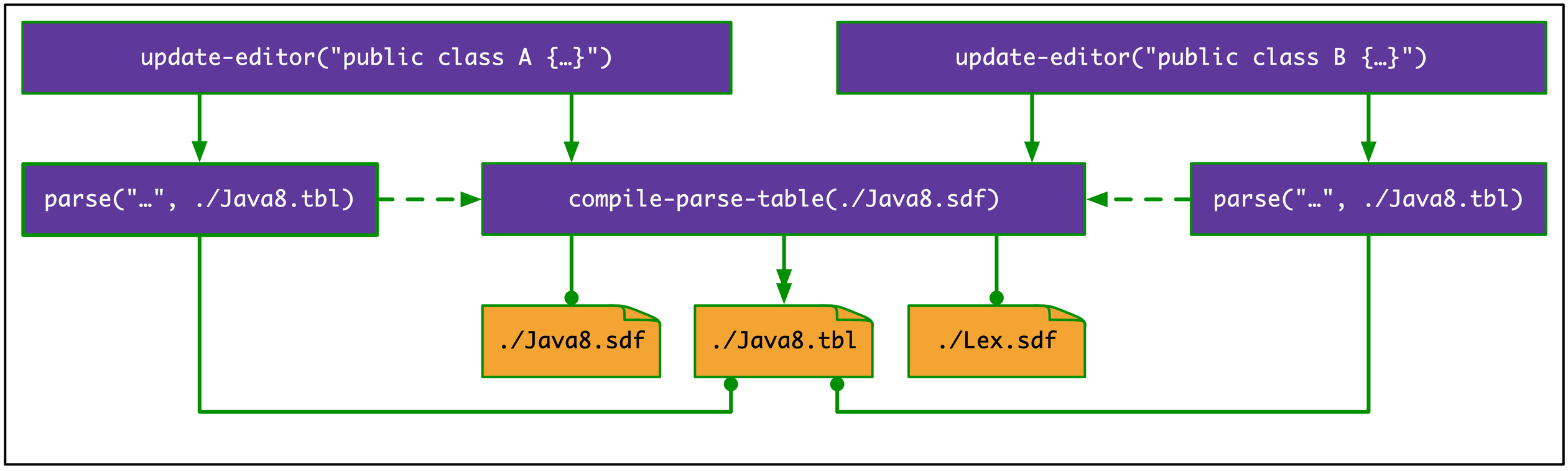



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

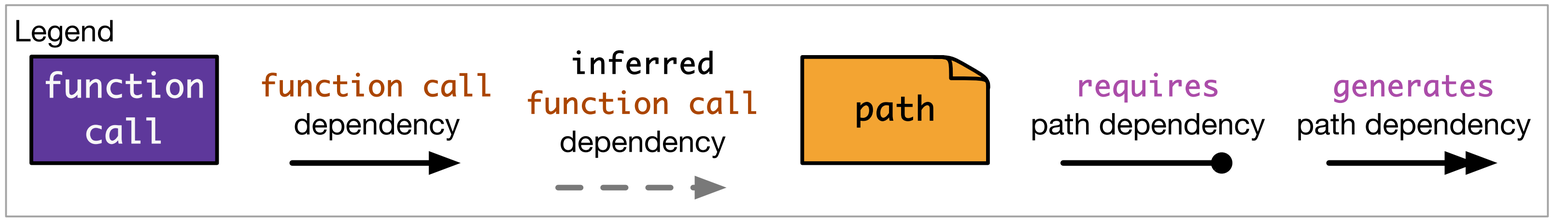
```
// 3) Syntax-based styling
data SyntaxStyler = foreign java class {}
data Styling = foreign java class {}
func esv2styler(path) -> SyntaxStyler = foreign java class#method
func esvStyle(Token*, SyntaxStyler) -> Styling = foreign java esv#style
func style(tokens: Token*, langSpec: LangSpec) -> Styling = {
  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)
```

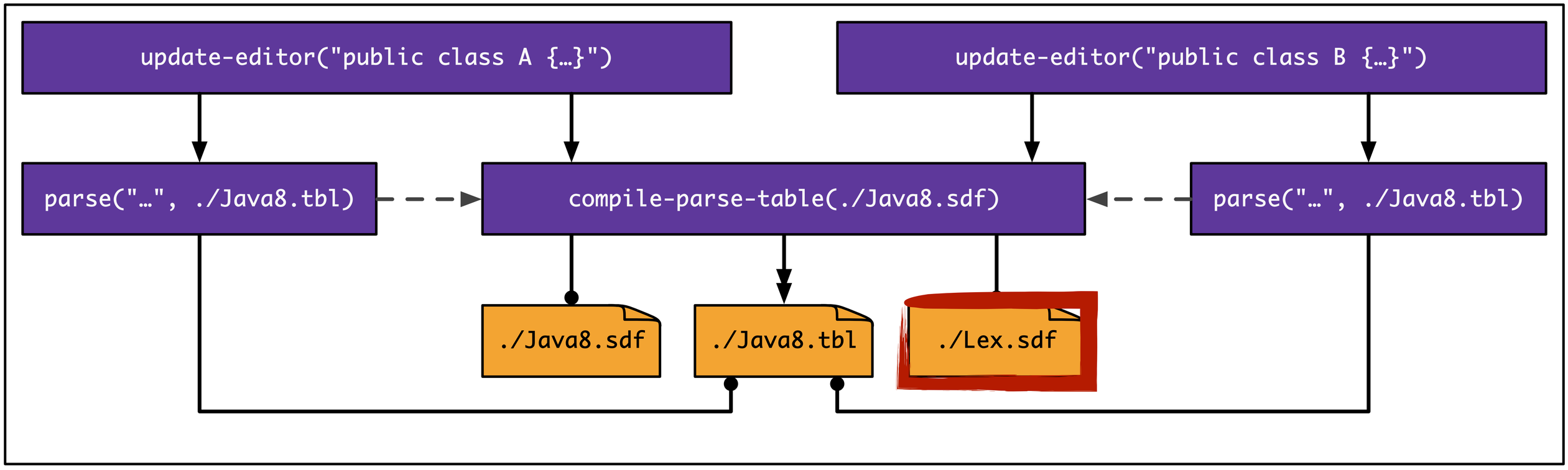
On-demand execution

vs

Change-driven execution

Change-driven execution

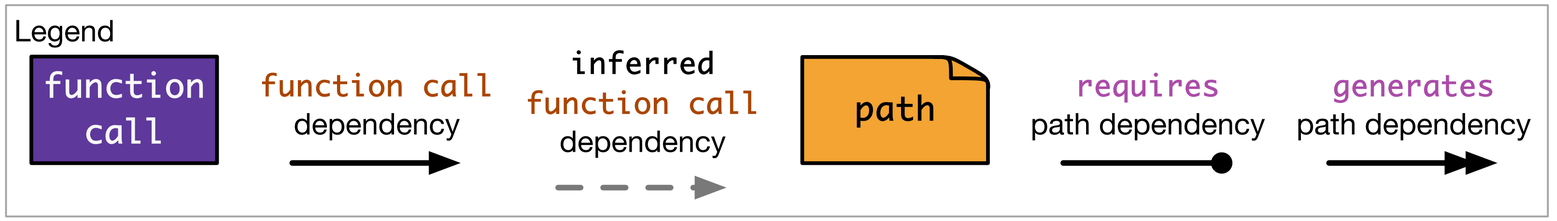


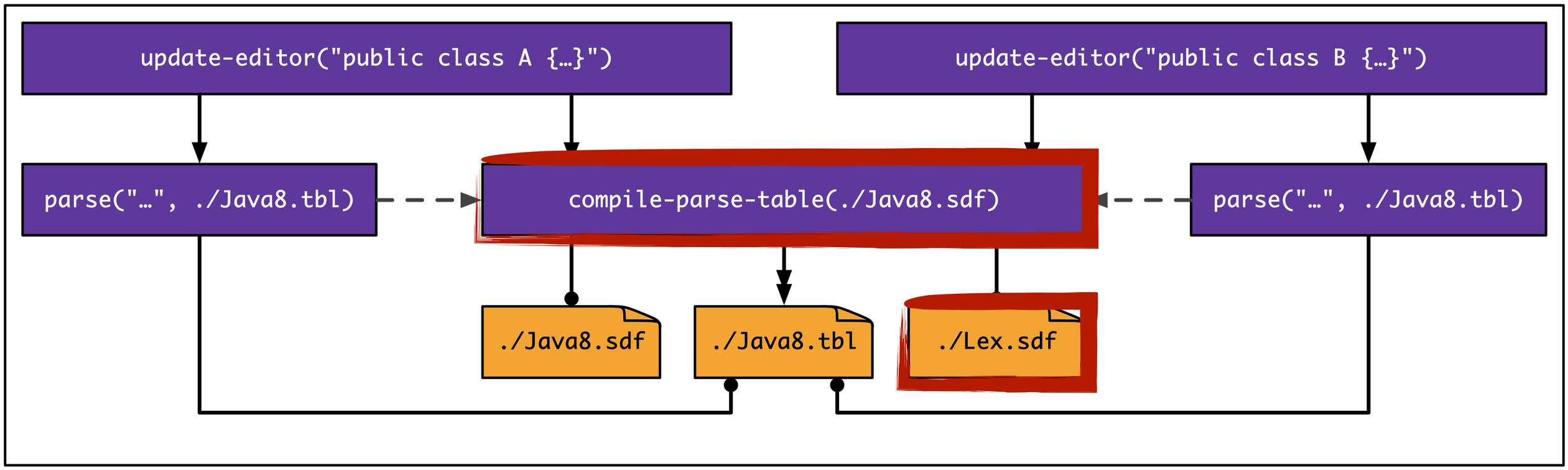



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

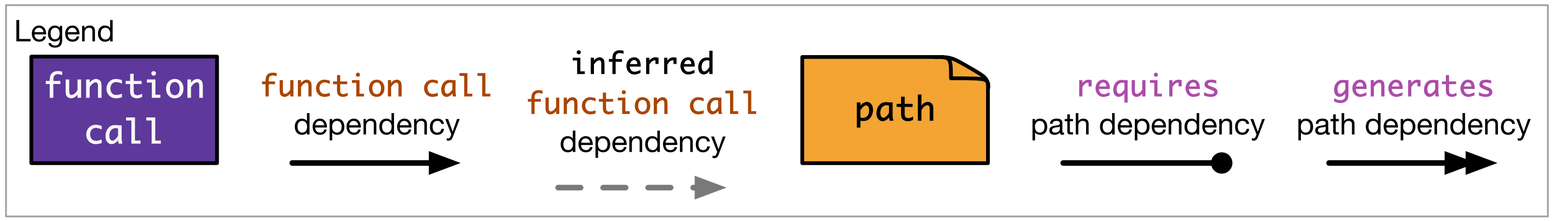


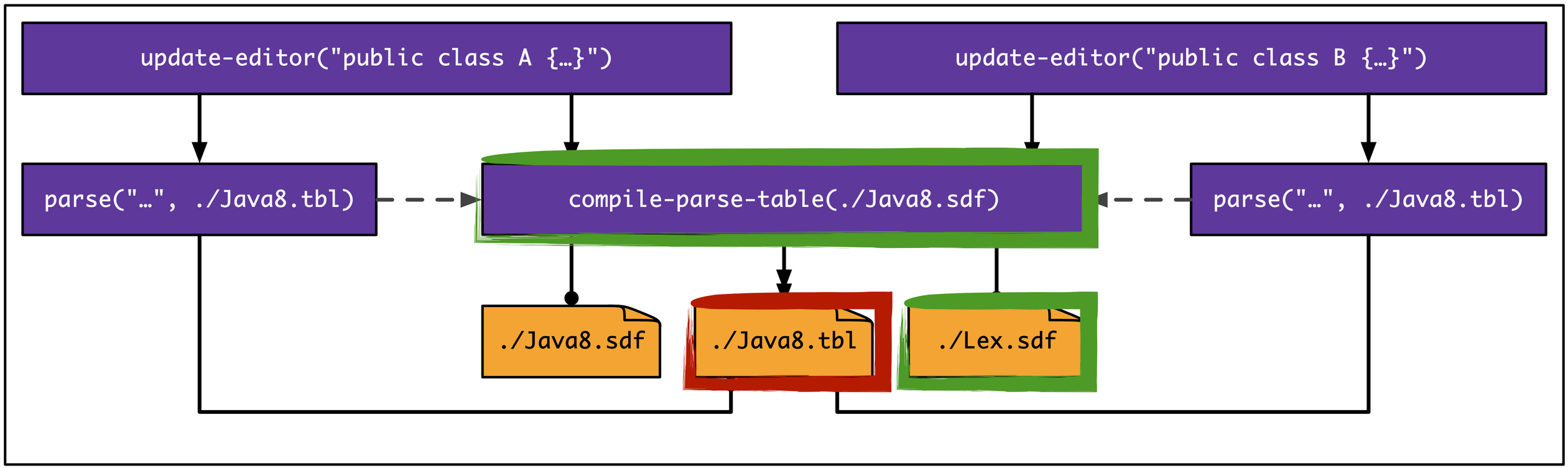



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

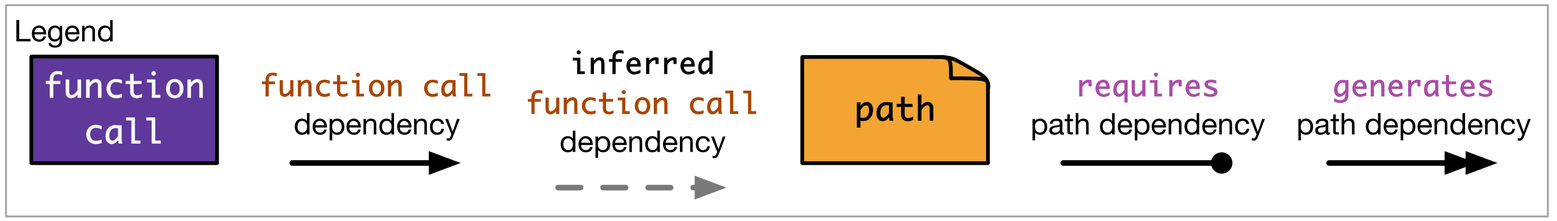


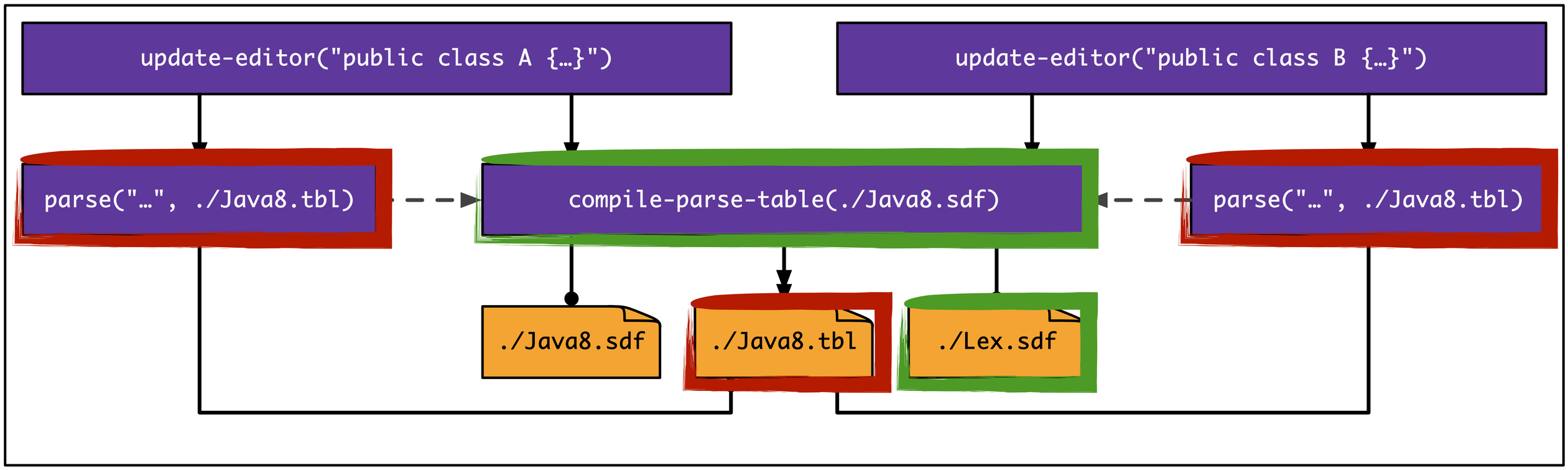



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

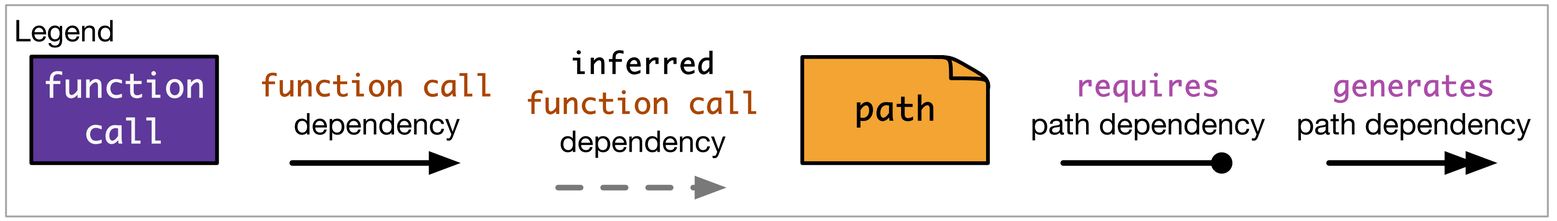


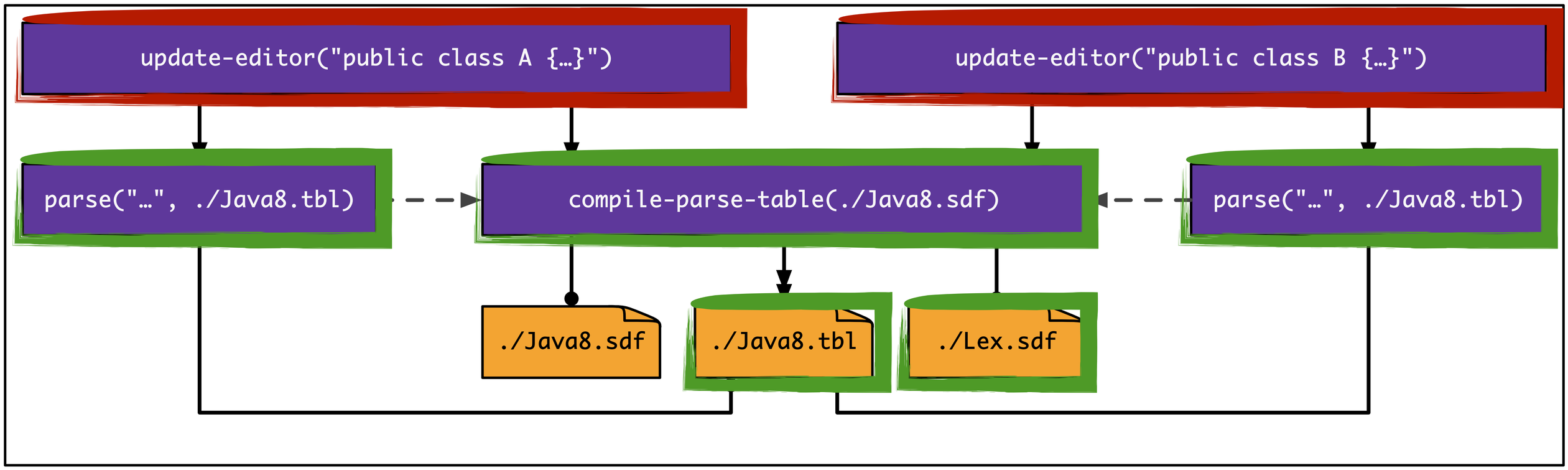



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

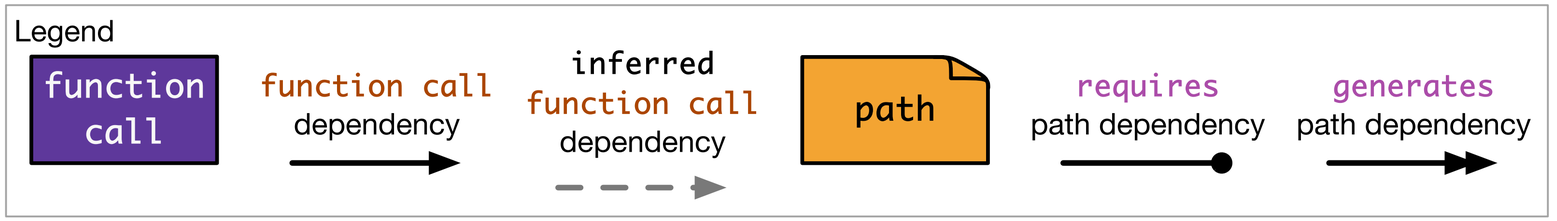


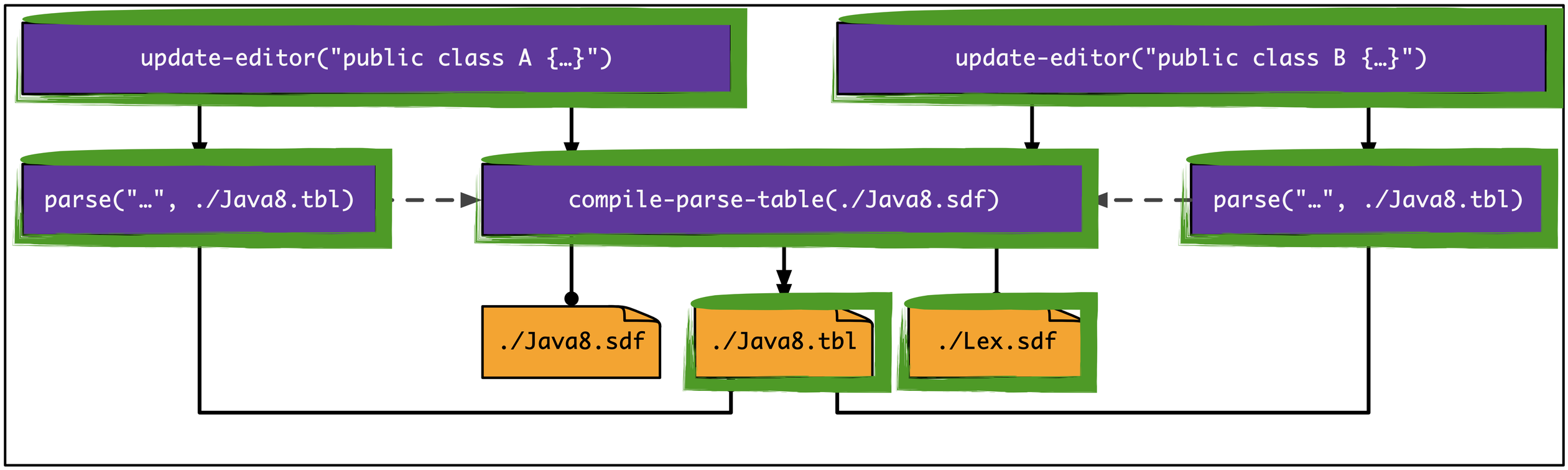



```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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



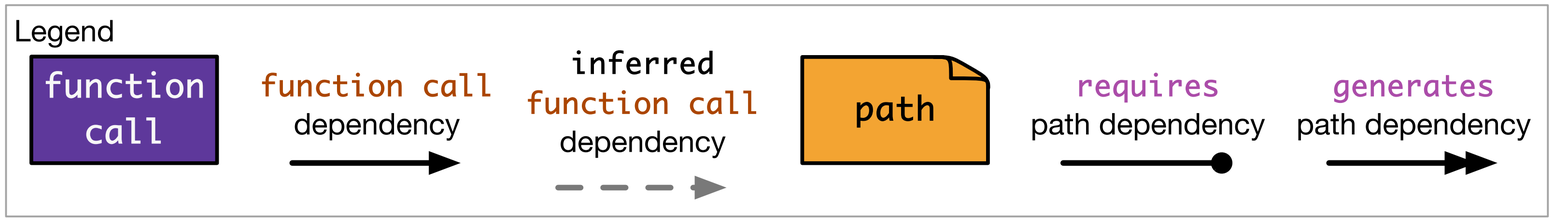


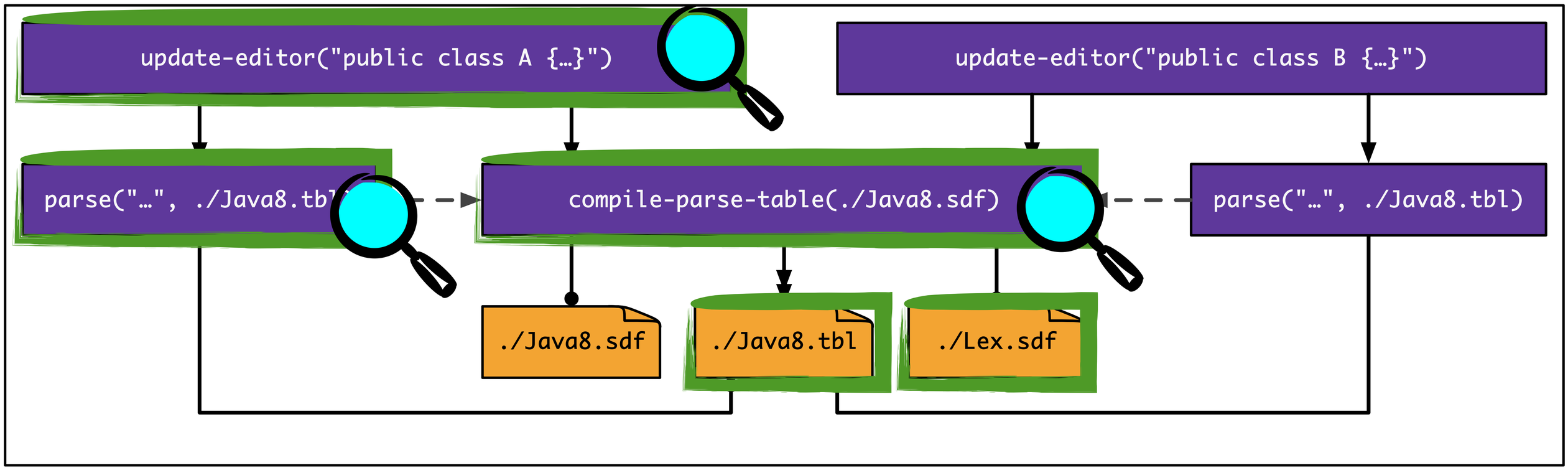

```
func update-editor(text: string) -> Msg* = {
  val parseTable = compile-parse-table(syntaxDef);
  parse(text, parseTable);
}
func compile-parse-table(defFile: path) -> path = {
  requires defFile;
  [requires dep | dep <- extract-deps(depFile)];
  generates tblFile;
}
func parse(text: string, pt: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

Developing interactive pipelines is

Complicated

Error Prone

High Development Cost

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

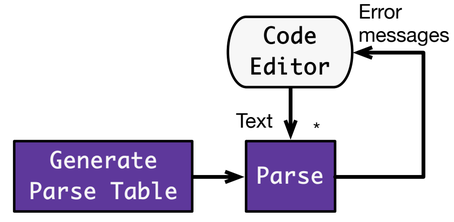
PIE DSL code

```
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

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
```

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(depFile)];
  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
```

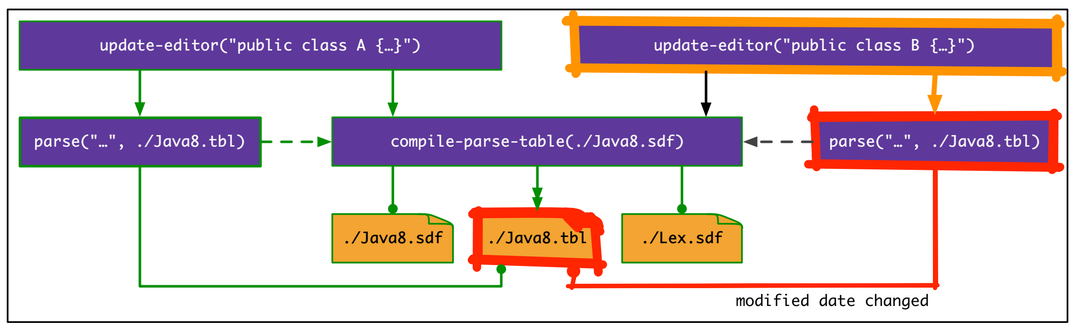
Foreign JVM code

```
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 { ... }
```


27

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(depFile)];
  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)
  }
}
```

52

Developing interactive pipelines is

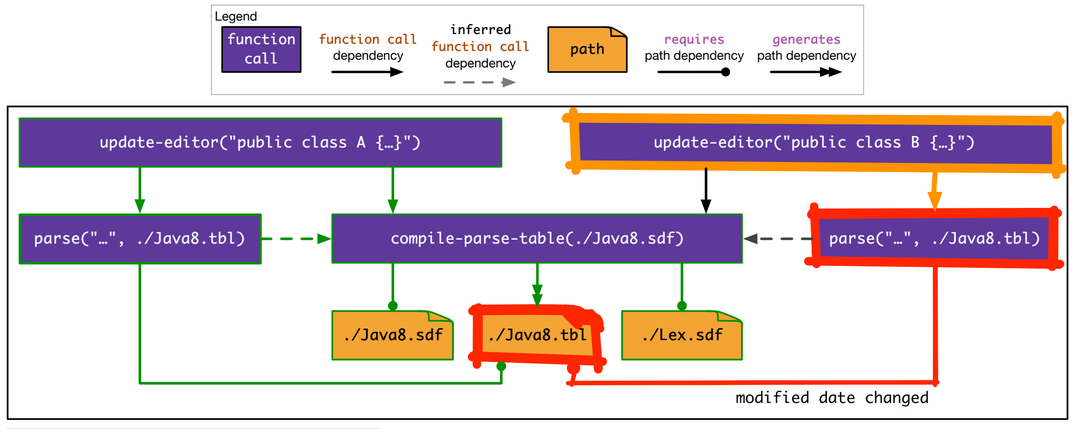
Complicated

Error Prone

High Development Cost

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

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(depFile)];
  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)
  }
}
```


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

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
```

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 { ... }
```