A Deep Dive into the Mill Scala Build Tool

Li Haoyi, Scaladays Madrid 13 Sep 2023

About Myself

Been using Scala since 2012

Work on the Databricks Developer Platform

Lots of Scala OSS work: Scala.js, Mill, Ammonite, uPickle, Fastparse, …

Author of Hands-on Scala Programming

Why Care About Mill

Why Care About Mill

Gradle is such garbage. I can't count the number of times our ex-Gradler has said "you're not going to like the answer..." - Hacker News

SBT's bizarre abstraction and unreadable syntax is a huge, frustrating obstacle to adopting Scala - Reddit

I only have good things to say about Mill, except maybe that I wish it had been released 10 years earlier or so :) - Reddit

The developer experience is so delightful compared to the other build system in the JVM land. And the task model seems just right - Hacker News

A Deep Dive into the Mill Scala Build Tool

1. **Why Build Tools Are Hard**

2. Getting Started with Mill

3. Mill Fundamentals

4. Why Mill Works

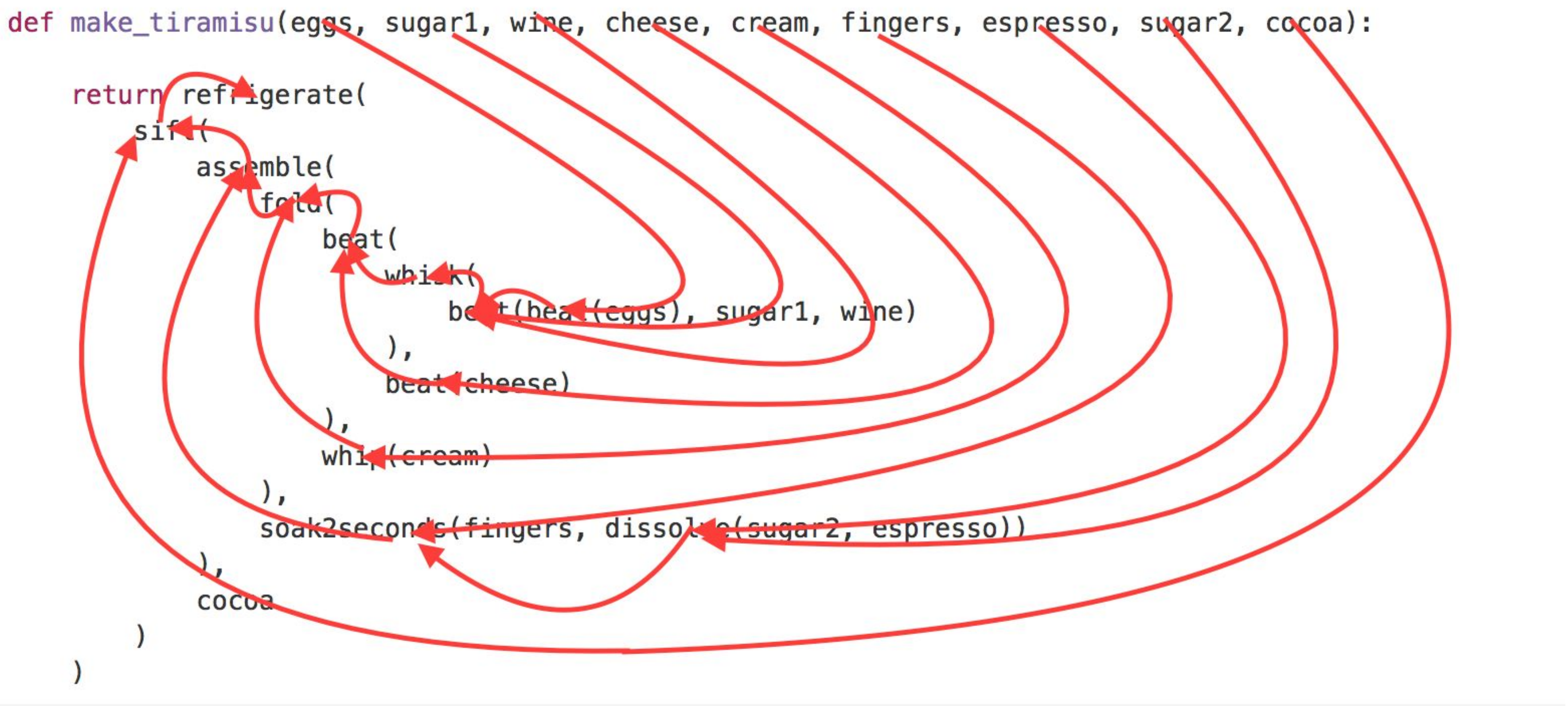
1.1 Build Tools As Pure Functional Programs

```python
def make_tiramisu(eggs, sugar1, wine, cheese, cream, fingers, espresso, sugar2, cocoa):
    beat_eggs = beat(eggs)
    mixture = beat(beat_eggs, sugar1, wine)
    whisked = whisk(mixture)
    beat_cheese = beat(cheese)
    cheese_mixture = beat(whisked, beat_cheese)
    whipped_cream = whip(cream)
    folded_mixture = fold(cheese_mixture, whipped_cream)
    sweet_espresso = dissolve(sugar2, espresso)
    wet_fingers = soak2seconds(fingers, sweet_espresso)
    assembled = assemble(folded_mixture, wet_fingers)
    complete = sift(assembled, cocoa)
    ready_tiramisu = refrigerate(complete)
    return ready_tiramisu
```

1.2 Build Tools As Pure Functional Programs

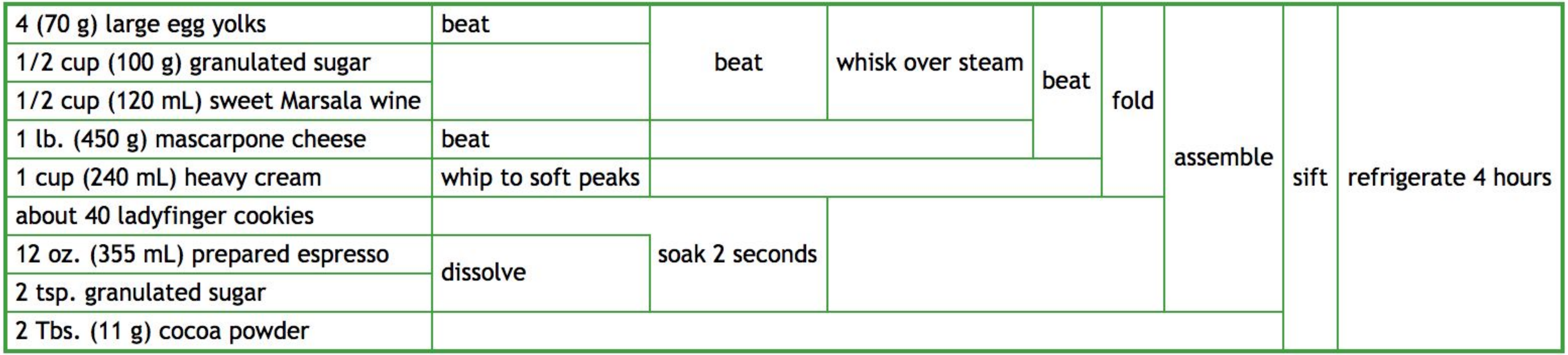
```python
def make_tiramisu(eggs, sugar1, wine, cheese, cream, fingers, espresso, sugar2, cocoa):
    return refrigerate(
        sift(
            assemble(
                fold(
                    beat(
                        whisk(
                            beat(beat(eggs), sugar1, wine)
                        ),
                        beat(cheese)
                    ),
                    whip(cream)
                ),
                soak2seconds(fingers, dissolve(sugar2, espresso))
            ),
            cocoa
        )
    )
```

1.3 Build Tools As Pure Functional Programs



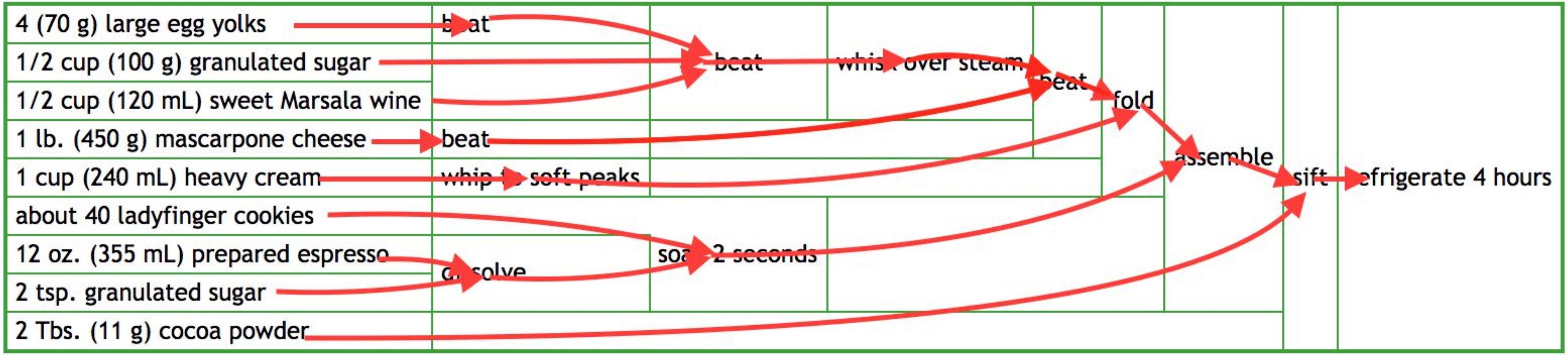
```
def make_tiramisu(eggs, sugar1, wine, cheese, cream, fingers, espresso, sugar2, cocoa):

    return refrigerate(
        sift(
            assemble(
                fold(
                    beat(
                        whisk(
                            beat(beat(eggs), sugar1, wine)
                        ),
                        beat(cheese)
                    ),
                    whip(cream)
                ),
                soak2seconds(fingers, dissolve(sugar2, espresso))
            ),
            cocoa
        )
    )
```

1.4 Build Tools As Pure Functional Programs



<table>
<tr><td>4 (70 g) large egg yolks</td><td>beat</td><td rowspan="3">beat</td><td rowspan="3">whisk over steam</td><td rowspan="4">beat</td><td rowspan="5">fold</td><td rowspan="8">assemble</td><td rowspan="9">sift</td><td rowspan="9">refrigerate 4 hours</td></tr>
<tr><td>1/2 cup (100 g) granulated sugar</td><td rowspan="2"></td></tr>
<tr><td>1/2 cup (120 mL) sweet Marsala wine</td></tr>
<tr><td>1 lb. (450 g) mascarpone cheese</td><td>beat</td><td colspan="2"></td></tr>
<tr><td>1 cup (240 mL) heavy cream</td><td>whip to soft peaks</td><td colspan="3"></td></tr>
<tr><td>about 40 ladyfinger cookies</td><td></td><td rowspan="3">soak 2 seconds</td><td rowspan="3" colspan="3"></td></tr>
<tr><td>12 oz. (355 mL) prepared espresso</td><td rowspan="2">dissolve</td></tr>
<tr><td>2 tsp. granulated sugar</td></tr>
<tr><td>2 Tbs. (11 g) cocoa powder</td><td colspan="6"></td></tr>
</table>

1.4 Build Tools As Pure Functional Programs



4 (70 g) large egg yolks

1/2 cup (100 g) granulated sugar

1/2 cup (120 mL) sweet Marsala wine

1 lb. (450 g) mascarpone cheese

1 cup (240 mL) heavy cream

about 40 ladyfinger cookies

12 oz. (355 mL) prepared espresso

2 tsp. granulated sugar

2 Tbs. (11 g) cocoa powder

beat

beat

whisk over steam

beat

fold

beat

whip to soft peaks

soak 2 seconds

dissolve

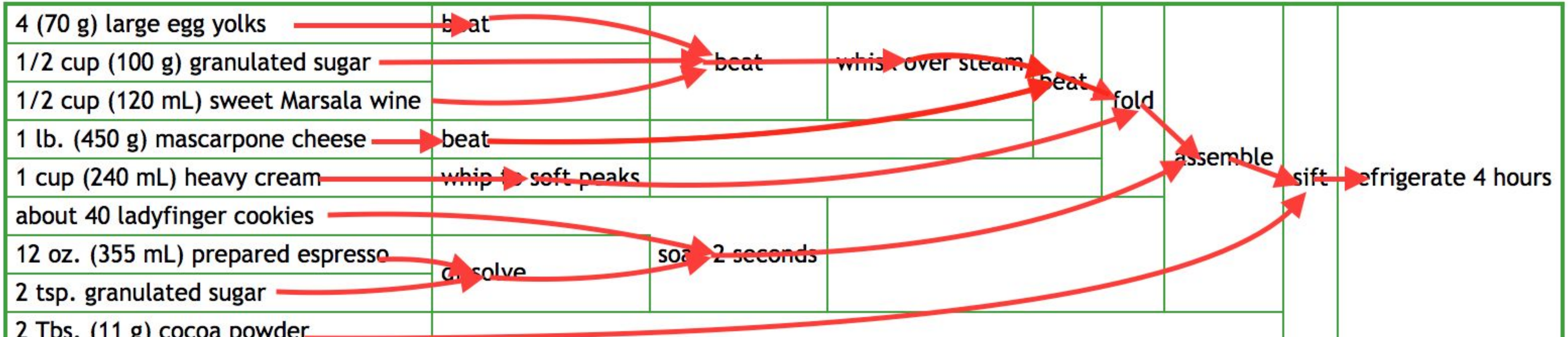
assemble

sift

refrigerate 4 hours

1.4 Build Tools As Pure Functional Programs

```
beat_eggs = beat(eggs)
mixture = beat(beat_eggs, sugar1, wine)
whisked = whisk(mixture)
beat_cheese = beat(cheese)
cheese_mixture = beat(whisked, beat_cheese)
whipped_cream = whip(cream)
folded_mixture = fold(cheese_mixture, whipped_cream)
sweet_espresso = dissolve(sugar2, espresso)
```



4 (70 g) large egg yolks

1/2 cup (100 g) granulated sugar

1/2 cup (120 mL) sweet Marsala wine

1 lb. (450 g) mascarpone cheese

1 cup (240 mL) heavy cream

about 40 ladyfinger cookies

12 oz. (355 mL) prepared espresso

2 tsp. granulated sugar

2 Tbs. (11 g) cocoa powder

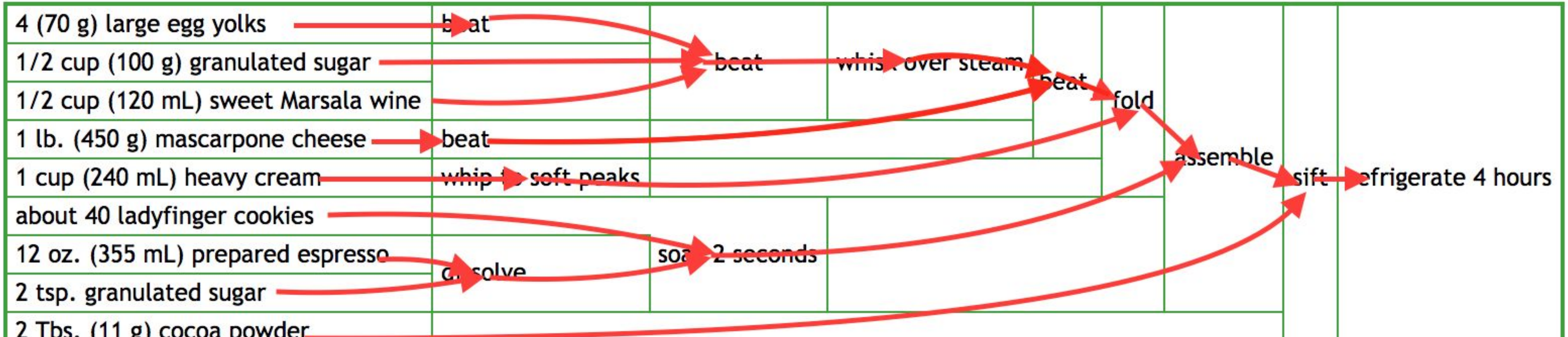
1.4 Build Tools As Pure Functional Programs

```
beat_eggs = beat(eggs)
mixture = beat(beat_eggs, sugar1, wine)
whisked = whisk(mixture)
beat_cheese = beat(cheese)
cheese_mixture = beat(whisked, beat_cheese)
whipped_cream = whip(cream)
folded_mixture = fold(cheese_mixture, whipped_cream)
sweet_espresso = dissolve(sugar2, espresso)
```

Explicit dependencies between computations

No Side Effects, uncontrolled mutation

Build Graph = Dataflow Graph = Call Graph



4 (70 g) large egg yolks

1/2 cup (100 g) granulated sugar

1/2 cup (120 mL) sweet Marsala wine

1 lb. (450 g) mascarpone cheese

1 cup (240 mL) heavy cream

about 40 ladyfinger cookies

12 oz. (355 mL) prepared espresso

2 tsp. granulated sugar

2 Tbs. (11 g) cocoa powder

beat

beat

whisk over steam

beat

fold

beat

whip to soft peaks

assemble

sift

refrigerate 4 hours

soak 2 seconds

dissolve

1.5 So What’s Hard About Build Tools?

1.5 So What’s Hard About Build Tools?

1. What tasks depends on what?
2. Where do input files come from?
3. What needs to run in what order?
4. What can be parallelized and what can’t?
5. Where can tasks read/write to disk?
6. How are tasks cached?
7. How are tasks run from the CLI?
8. How are cross-builds (across different configurations) handled?
9. How do I define my own custom tasks?
10. How do tasks pass data to each other?
11. How to manage the repetition in a build?
12. What is a "Module"? How do they relate to "Tasks"?
13. How do you customize a task or module to do something different?
14. What APIs do tasks use to actually do things?
15. How is in-memory caching handled?

1.5 So What’s Hard About Build Tools?

1. What tasks depends on what? - SBT Scope Delegation
2. Where do input files come from? - SBT How to Track File Inputs and Outputs
3. What needs to run in what order?
4. What can be parallelized and what can’t?
5. Where can tasks read/write to disk? - Gradle Working With Files
6. How are tasks cached? - Gradle Incremental Build
7. How are tasks run from the CLI? - SBT Parsing and Tab Completion
8. How are cross-builds (across different configurations) handled?
9. How do I define my own custom tasks? - Gradle Authoring Tasks
10. How do tasks pass data to each other?
11. How to manage the repetition in a build? - SBT Plugins and Best Practices
12. What is a "Module"? How do they relate to "Tasks"?
13. How do you customize a task or module to do something different?
14. What APIs do tasks use to actually do things? - SBT Paths and Globs, Gradle Working with Files
15. How is in-memory caching handled?

A Deep Dive into the Mill Scala Build Tool

1. **Why Build Tools Are Hard**

2. Getting Started with Mill

3. Mill Fundamentals

4. Why Mill Works

A Deep Dive into the Mill Scala Build Tool

1. Why Build Tools Are Hard

2. **Getting Started with Mill**

3. Mill Fundamentals

4. Why Mill Works

2 Getting Started with Mill

1. **Single Scala Module**

2. Customizing Tasks

3. Multiple Scala Modules

2.1 Single Scala Module

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"
  def ivyDeps = Agg(
    ivy"com.lihaoyi::scalatags:0.8.2",
    ivy"com.lihaoyi::mainargs:0.4.0"
  )

  object test extends ScalaTests {
    def ivyDeps = Agg(ivy"com.lihaoyi::utest:0.7.11")
    def testFramework = "utest.runner.Framework"
  }
}
```

2.1 Single Scala Module

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"
  def ivyDeps = Agg(
    ivy"com.lihaoyi::scalatags:0.8.2",
    ivy"com.lihaoyi::mainargs:0.4.0"
  )

  object test extends ScalaTests {
    def ivyDeps = Agg(ivy"com.lihaoyi::utest:0.7.11")
    def testFramework = "utest.runner.Framework"
  }
}
```

```
// src/
```

2.1 Single Scala Module

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"
  def ivyDeps = Agg(
    ivy"com.lihaoyi::scalatags:0.8.2",
    ivy"com.lihaoyi::mainargs:0.4.0"
  )

  object test extends ScalaTests {
    def ivyDeps = Agg(ivy"com.lihaoyi::utest:0.7.11")
    def testFramework = "utest.runner.Framework"
  }
}
```

```scala
// src/Foo.scala
package foo
import scalatags.Text.all._
import mainargs.{main, ParserForMethods}

object Foo {
  def generateHtml(text: String) = h1(text).toString

  @main
  def main(text: String) =
    println(generateHtml(text))

  def main(args: Array[String]): Unit =
    ParserForMethods(this).runOrExit(args)
}
```

2.1 Single Scala Module

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"
  def ivyDeps = Agg(
    ivy"com.lihaoyi::scalatags:0.8.2",
    ivy"com.lihaoyi::mainargs:0.4.0"
  )

  object test extends ScalaTests {
    def ivyDeps = Agg(ivy"com.lihaoyi::utest:0.7.11")
    def testFramework = "utest.runner.Framework"
  }
}
```

```
> ./mill resolve _
assembly
...
clean
...
compile
...
run
...
show
...
inspect
...
```

2.1 Single Scala Module

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"
  def ivyDeps = Agg(
    ivy"com.lihaoyi::scalatags:0.8.2",
    ivy"com.lihaoyi::mainargs:0.4.0"
  )

  object test extends ScalaTests {
    def ivyDeps = Agg(ivy"com.lihaoyi::utest:0.7.11")
    def testFramework = "utest.runner.Framework"
  }
}
```

```
> ./mill inspect compile
compile(ScalaModule.scala:212)
    Compiles the current module to
    generate compiled classfiles.
Inputs:
    scalaVersion
    upstreamCompileOutput
    allSourceFiles
    compileClasspath
```

2.1 Single Scala Module

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"
  def ivyDeps = Agg(
    ivy"com.lihaoyi::scalatags:0.8.2",
    ivy"com.lihaoyi::mainargs:0.4.0"
  )

  object test extends ScalaTests {
    def ivyDeps = Agg(ivy"com.lihaoyi::utest:0.7.11")
    def testFramework = "utest.runner.Framework"
  }
}
```

```
> ./mill compile
...
compiling 1 Scala source to...
```

2.1 Single Scala Module

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"
  def ivyDeps = Agg(
    ivy"com.lihaoyi::scalatags:0.8.2",
    ivy"com.lihaoyi::mainargs:0.4.0"
  )

  object test extends ScalaTests {
    def ivyDeps = Agg(ivy"com.lihaoyi::utest:0.7.11")
    def testFramework = "utest.runner.Framework"
  }
}
```

```
> ./mill compile
...
compiling 1 Scala source to...

> ./mill run --text hello
<h1>hello</h1>
```

2.1 Single Scala Module

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"
  def ivyDeps = Agg(
    ivy"com.lihaoyi::scalatags:0.8.2",
    ivy"com.lihaoyi::mainargs:0.4.0"
  )

  object test extends ScalaTests {
    def ivyDeps = Agg(ivy"com.lihaoyi::utest:0.7.11")
    def testFramework = "utest.runner.Framework"
  }
}
```

```
> ./mill compile
...
compiling 1 Scala source to...

> ./mill run --text hello
<h1>hello</h1>

> ./mill test
...
+ foo.FooTests.simple ...  <h1>hello</h1>
+ foo.FooTests.escaping ...
<h1>&lt;hello&gt;</h1>
```

2.1 Single Scala Module

```
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"
  def ivyDeps = Agg(
    ivy"com.lihaoyi::scalatags:0.8.2",
    ivy"com.lihaoyi::mainargs:0.4.0"
  )

  object test extends ScalaTests {
    def ivyDeps = Agg(ivy"com.lihaoyi::utest:0.7.11")
    def testFramework = "utest.runner.Framework"
  }
}
```

```
> ./mill assembly
```

2.1 Single Scala Module

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"
  def ivyDeps = Agg(
    ivy"com.lihaoyi::scalatags:0.8.2",
    ivy"com.lihaoyi::mainargs:0.4.0"
  )

  object test extends ScalaTests {
    def ivyDeps = Agg(ivy"com.lihaoyi::utest:0.7.11")
    def testFramework = "utest.runner.Framework"
  }
}
```

```
> ./mill assembly


> ./mill show assembly
".../out/assembly.dest/out.jar"
```

2.1 Single Scala Module

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"
  def ivyDeps = Agg(
    ivy"com.lihaoyi::scalatags:0.8.2",
    ivy"com.lihaoyi::mainargs:0.4.0"
  )

  object test extends ScalaTests {
    def ivyDeps = Agg(ivy"com.lihaoyi::utest:0.7.11")
    def testFramework = "utest.runner.Framework"
  }
}
```

```
> ./mill assembly

> ./mill show assembly
".../out/assembly.dest/out.jar"

> java -jar out/assembly.dest/out.jar --text hello
<h1>hello</h1>

> ./out/assembly.dest/out.jar --text hello
<h1>hello</h1>
```

2 Getting Started with Mill

1. Single Scala Module

2. **Customizing Tasks**

3. Multiple Scala Modules

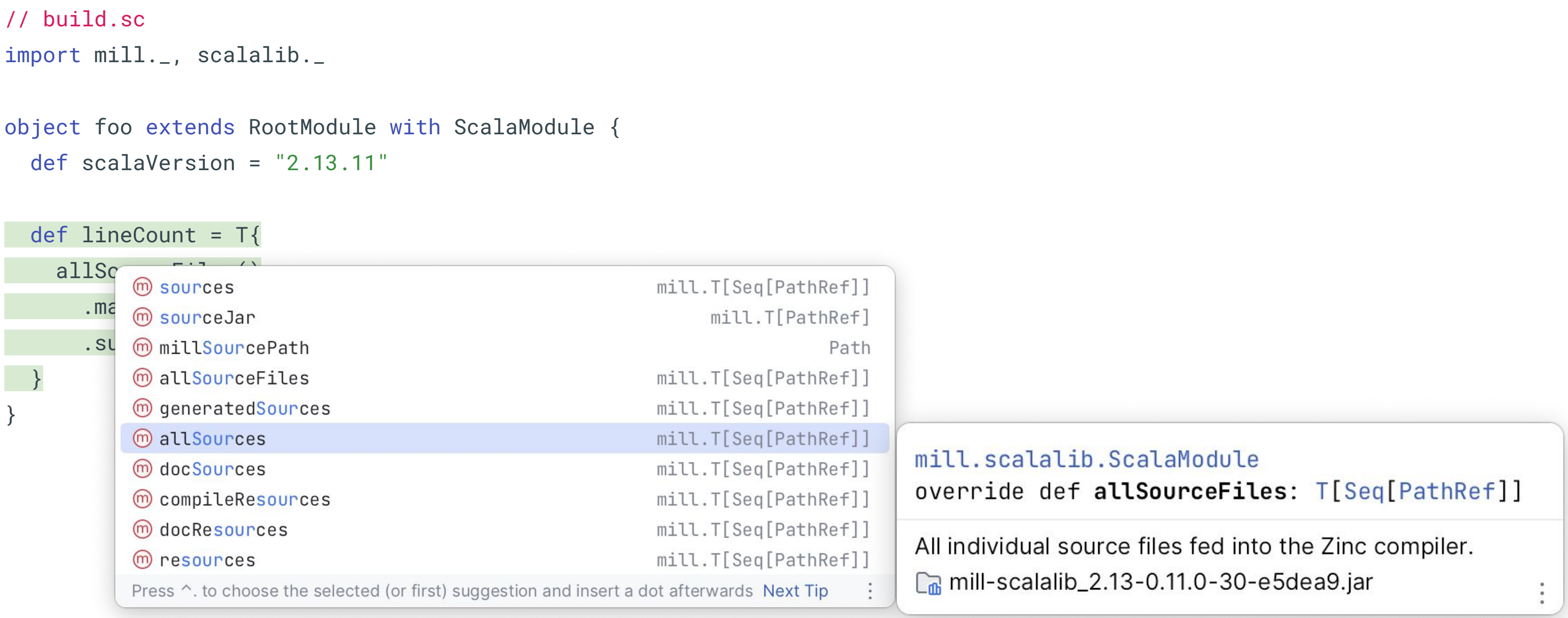
2.2 Customizing Tasks

```
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"
}
```

2.2 Customizing Tasks

```
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles()
      .map(f => os.read.lines(f.path).size)
      .sum
  }
}
```

2.2 Customizing Tasks



```
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSc
      .ma
      .su
  }
}
```

```
ⓜ sources                          mill.T[Seq[PathRef]]
ⓜ sourceJar                              mill.T[PathRef]
ⓜ millSourcePath                                    Path
ⓜ allSourceFiles                   mill.T[Seq[PathRef]]
ⓜ generatedSources                 mill.T[Seq[PathRef]]
ⓜ allSources                       mill.T[Seq[PathRef]]
ⓜ docSources                       mill.T[Seq[PathRef]]
ⓜ compileResources                 mill.T[Seq[PathRef]]
ⓜ docResources                     mill.T[Seq[PathRef]]
ⓜ resources                        mill.T[Seq[PathRef]]
```

Press ^. to choose the selected (or first) suggestion and insert a dot afterwards Next Tip

```
mill.scalalib.ScalaModule
override def allSourceFiles: T[Seq[PathRef]]
```

All individual source files fed into the Zinc compiler.

mill-scalalib_2.13-0.11.0-30-e5dea9.jar

2.2 Customizing Tasks

```
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles()
      .map(f => os.read.lines(f.path).size)
      .sum
  }
}
```

2.2 Customizing Tasks

```
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles()
      .map(f => os.read.lines(f.path).size)
      .sum
  }
}
```

```
> ./mill show lineCount
19
```

2.2 Customizing Tasks

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles()
      .map(f => os.read.lines(f.path).size)
      .sum
  }
}
```

```
> ./mill show lineCount
19


> ./mill inspect lineCount
lineCount(build.sc:7)
Inputs:
    allSourceFiles
```

2.2 Customizing Tasks

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles()
      .map(f => os.read.lines(f.path).size)
      .sum
  }

  override def resources = T{
    os.write(
      T.dest / "line-count.txt", "" + lineCount()
    )
    Seq(PathRef(T.dest))
  }
}
```

2.2 Customizing Tasks

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles()
      .map(f => os.read.lines(f.path).size)
      .sum
  }

  override def resources = T{
    os.write(
      T.dest / "line-count.txt", "" + lineCount()
    )
    Seq(PathRef(T.dest))
  }
}
```

```scala
// src/Foo.scala
package foo

object Foo{
  def main(args: Array[String]): Unit = {
    val lineCount = scala.io.Source
      .fromResource("line-count.txt")
      .mkString

    println(s"Line Count: $lineCount")
  }
}
```

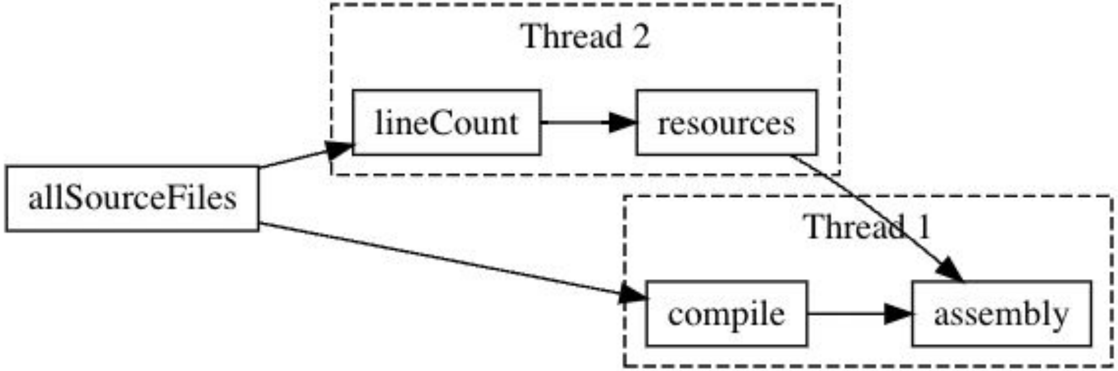
2.2 Customizing Tasks

```scala
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles()
      .map(f => os.read.lines(f.path).size)
      .sum
  }

  override def resources = T{
    os.write(
      T.dest / "line-count.txt", "" + lineCount()
    )
    Seq(PathRef(T.dest))
  }
}
```

```scala
// src/Foo.scala
package foo

object Foo{
  def main(args: Array[String]): Unit = {
    val lineCount = scala.io.Source
      .fromResource("line-count.txt")
      .mkString

    println(s"Line Count: $lineCount")
  }
}
```

```
> ./mill run
...
Line Count: 19
```

2.2 Customizing Tasks

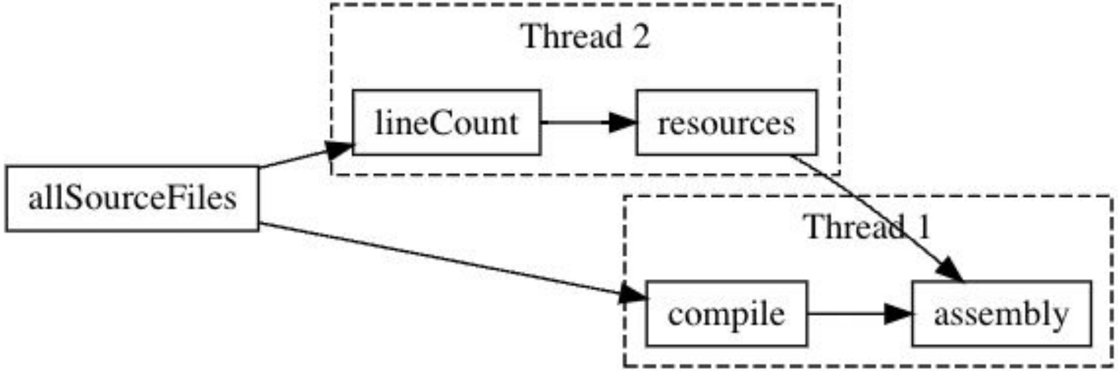


```
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles()
      .map(f => os.read.lines(f.path).size)
      .sum
  }

  override def resources = T{
    os.write(
      T.dest / "line-count.txt", "" + lineCount()
    )
    Seq(PathRef(T.dest))
  }
}
```

```
object Foo{
  def main(args: Array[String]): Unit = {
    val lineCount = scala.io.Source
      .fromResource("line-count.txt")
      .mkString

    println(s"Line Count: $lineCount")
  }
}
```

```
> ./mill run
...
Line Count: 19
```

2.2 Customizing Tasks



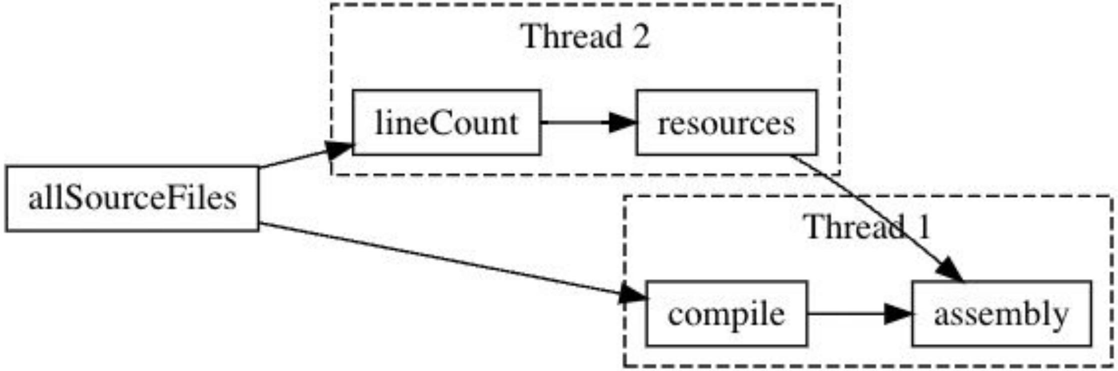
```
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles()
      .map(f => os.read.lines(f.path).size)
      .sum
  }

  override def resources = T{
    os.write(
      T.dest / "line-count.txt", "" + lineCount()
    )
    Seq(PathRef(T.dest))
  }
}
```

```
object Foo{
  def m
    val
      .
      .

    pri
  }
}
```

```
> ./mill show lineCount
19


> ./mill inspect lineCount
lineCount(build.sc:7)
Inputs:
    allSourceFiles
```

```
> ./mill run
...
Line Count: 19
```

2.2 Customizing Tasks



```
// build.sc
import mill._, scalalib._

object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles()
      .map(f => os.read.lines(f.path).size)
      .sum
  }

  override def resources = T
    os.write(
      T.dest / "line-count.txt", "" + lineCount()
    )
    Seq(PathRef(T.dest))
  }
}
```

Overrides member from JavaModule (mill.scalalib)

Press ⌘U to navigate

```
object Foo{
  def m
    val
      .
      .
```

```
> ./mill show lineCount
19


> ./mill inspect lineCount
7)
```

```
> ./mill run
...
Line Count: 19
```

2 Getting Started with Mill

1. Single Scala Module

2. Customizing Tasks

3. **Multiple Scala Modules**

2.3 Multiple Scala Modules

```
// build.sc
import mill._, scalalib._

trait MyModule extends ScalaModule {
  def scalaVersion = "2.13.11"
}

object foo extends MyModule {
  def moduleDeps = Seq(bar)
  def ivyDeps = Agg(ivy"com.lihaoyi::mainargs:0.4.0")
}

object bar extends MyModule {
  def ivyDeps = Agg(ivy"com.lihaoyi::scalatags:0.8.2")
}
```

2.3 Multiple Scala Modules

```
// build.sc
import mill._, scalalib._

trait MyModule extends ScalaModule {
  def scalaVersion = "2.13.11"
}

object foo extends MyModule {
  def moduleDeps = Seq(bar)
  def ivyDeps = Agg(ivy"com.lihaoyi::mainargs:0.4.0")
}

object bar extends MyModule {
  def ivyDeps = Agg(ivy"com.lihaoyi::scalatags:0.8.2")
}
```

```
build.sc
foo/
    src/
        Foo.scala
    resources/
        ...
bar/
    src/
        Bar.scala
    resources/
        ...
out/
    foo/
        compile.json
        compile.dest/
        ...
    bar/
        compile.json
        compile.dest/
        ...
```

2.3 Multiple Scala Modules

```
// build.sc
import mill._, scalalib._

trait MyModule extends ScalaModule {
  def scalaVersion = "2.13.11"
}

object foo extends MyModule {
  def moduleDeps = Seq(bar)
  def ivyDeps = Agg(ivy"com.lihaoyi::mainargs:0.4.0")
}

object bar extends MyModule {
  def ivyDeps = Agg(ivy"com.lihaoyi::scalatags:0.8.2")
}
```

```
> ./mill resolve __.run
foo.run
bar.run
```

2.3 Multiple Scala Modules

```scala
// build.sc
import mill._, scalalib._

trait MyModule extends ScalaModule {
  def scalaVersion = "2.13.11"
}

object foo extends MyModule {
  def moduleDeps = Seq(bar)
  def ivyDeps = Agg(ivy"com.lihaoyi::mainargs:0.4.0")
}

object bar extends MyModule {
  def ivyDeps = Agg(ivy"com.lihaoyi::scalatags:0.8.2")
}
```

```
> ./mill resolve __.run
foo.run
bar.run


> ./mill __.compile
```

2.3 Multiple Scala Modules

```scala
// build.sc
import mill._, scalalib._

trait MyModule extends ScalaModule {
  def scalaVersion = "2.13.11"
}

object foo extends MyModule {
  def moduleDeps = Seq(bar)
  def ivyDeps = Agg(ivy"com.lihaoyi::mainargs:0.4.0")
}

object bar extends MyModule {
  def ivyDeps = Agg(ivy"com.lihaoyi::scalatags:0.8.2")
}
```

```
> ./mill resolve __.run
foo.run
bar.run

> ./mill __.compile

> ./mill foo.run --foo-text hello --bar-text world
Foo.value: hello
Bar.value: <p>world</p>
```

2.3 Multiple Scala Modules

```scala
// build.sc
import mill._, scalalib._

trait MyModule extends ScalaModule {
  def scalaVersion = "2.13.11"
}

object foo extends MyModule {
  def moduleDeps = Seq(bar)
  def ivyDeps = Agg(ivy"com.lihaoyi::mainargs:0.4.0")
}

object bar extends MyModule {
  def ivyDeps = Agg(ivy"com.lihaoyi::scalatags:0.8.2")
}
```

```
> ./mill resolve __.run
foo.run
bar.run

> ./mill __.compile

> ./mill foo.run --foo-text hello --bar-text world
Foo.value: hello
Bar.value: <p>world</p>

> ./mill show {foo,bar}.assembly
{
  "foo.assembly": "out/foo/assembly.dest/out.jar",
  "bar.assembly": "out/foo/assembly.dest/out.jar"
}
```

2 Getting Started with Mill

1. Single Scala Module

2. Customizing Tasks

3. **Multiple Scala Modules**

A Deep Dive into the Mill Scala Build Tool

1. Why Build Tools Are Hard

2. Getting Started with Mill

3. **Mill Fundamentals**

4. Why Mill Works

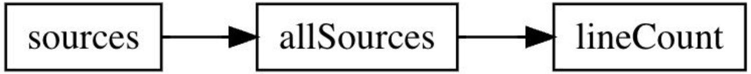
3.Mill Fundamentals

1. **Tasks**
2. Modules
3. DIY Java Modules

3.1 Tasks

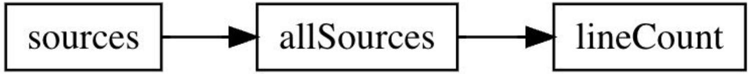
```scala
// build.sc
import mill._

def sources = T.source { millSourcePath / "src" }

def allSources = T {
  os.walk(sources().path)
    .filter(_.ext == "java")
    .map(PathRef(_))
}

def lineCount: T[Int] = T {
  println("Computing line count")
  allSources()
    .map(p => os.read.lines(p.path).size)
    .sum
}
```

3.1 Tasks



```
// build.sc
import mill._

def sources = T.source { millSourcePath / "src" }

def allSources = T {
  os.walk(sources().path)
    .filter(_.ext == "java")
    .map(PathRef(_))
}

def lineCount: T[Int] = T {
  println("Computing line count")
  allSources()
    .map(p => os.read.lines(p.path).size)
    .sum
}
```

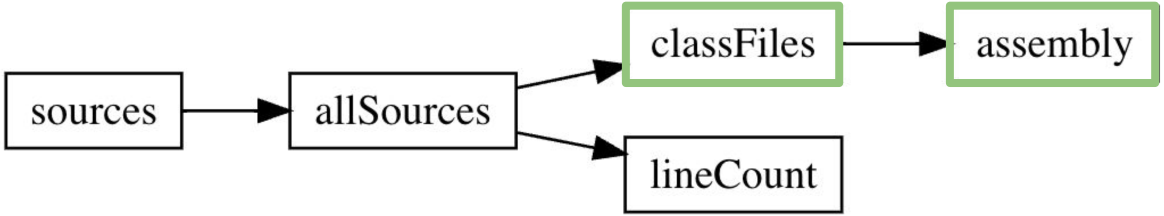
3.1 Tasks



```
// build.sc
import mill._

def sources = T.source { millSourcePath / "src" }

def allSources = T {
  os.walk(sources().path)
    .filter(_.ext == "java")
    .map(PathRef(_))
}

def lineCount: T[Int] = T {
  println("Computing line count")
  allSources()
    .map(p => os.read.lines(p.path).size)
    .sum
}
```

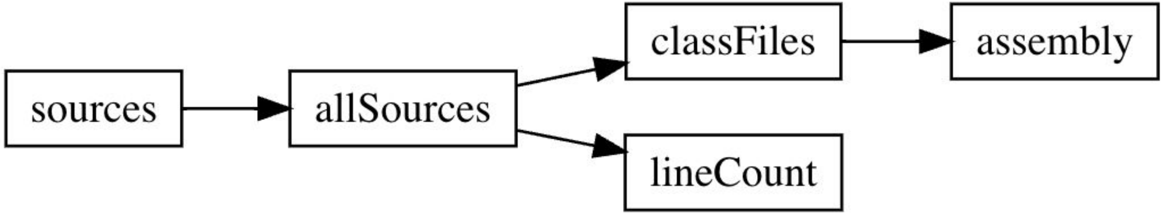
```
> ./mill show lineCount
Computing line count
16
```

3.1 Tasks



```
def classFiles = T {
  println("Generating classfiles")
  val javacArgs = allSources().map(_.path)
  os.proc("javac", javacArgs, "-d", T.dest)
    .call(cwd = T.dest)

  PathRef(T.dest)
}

def assembly = T {
  println("Generating assembly")
  val jarPath = T.dest / "out.jar"
  os.proc("jar", "-cfe", jarPath, "foo.Foo", ".")
    .call(cwd = classFiles().path)

  PathRef(T.dest / "out.jar")
}
```

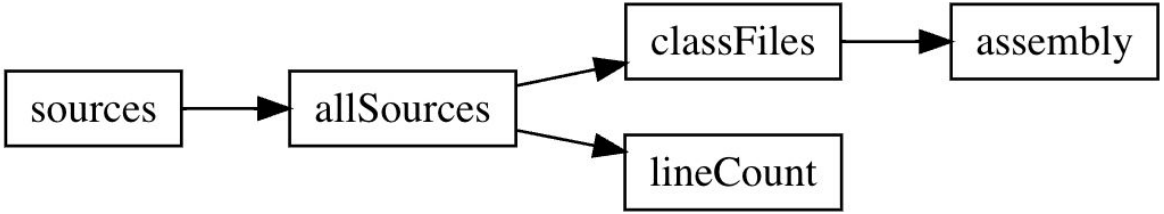
3.1 Tasks



```
def classFiles = T {
  println("Generating classfiles")
  val javacArgs = allSources().map(_.path)
  os.proc("javac", javacArgs, "-d", T.dest)
    .call(cwd = T.dest)

  PathRef(T.dest)
}

def assembly = T {
  println("Generating assembly")
  val jarPath = T.dest / "out.jar"
  os.proc("jar", "-cfe", jarPath, "foo.Foo", ".")
    .call(cwd = classFiles().path)

  PathRef(T.dest / "out.jar")
}
```

```
> ./mill show assembly
Generating classfiles
Generating assembly
".../out/assembly.dest/out.jar"
```

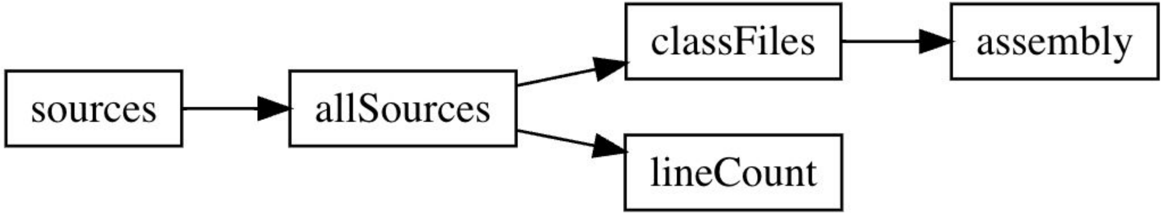
3.1 Tasks



```scala
def classFiles = T {
  println("Generating classfiles")
  val javacArgs = allSources().map(_.path)
  os.proc("javac", javacArgs, "-d", T.dest)
    .call(cwd = T.dest)

  PathRef(T.dest)
}

def assembly = T {
  println("Generating assembly")
  val jarPath = T.dest / "out.jar"
  os.proc("jar", "-cfe", jarPath, "foo.Foo", ".")
    .call(cwd = classFiles().path)

  PathRef(T.dest / "out.jar")
}
```

```
> ./mill show assembly
Generating classfiles
Generating assembly
".../out/assembly.dest/out.jar"

> ./mill show assembly
".../out/assembly.dest/out.jar"
```

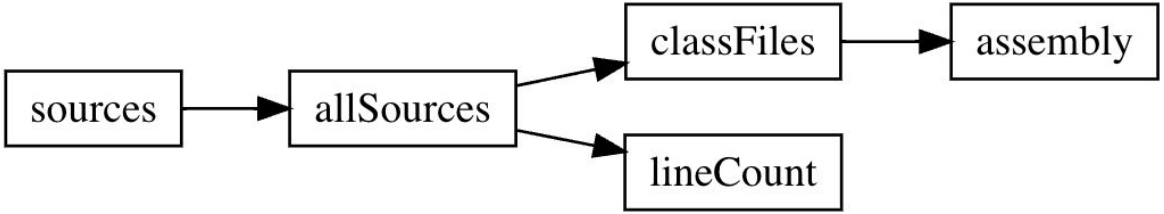
3.1 Tasks



```
def classFiles = T {
  println("Generating classfiles")
  val javacArgs = allSources().map(_.path)
  os.proc("javac", javacArgs, "-d", T.dest)
    .call(cwd = T.dest)

  PathRef(T.dest)
}

def assembly = T {
  println("Generating assembly")
  val jarPath = T.dest / "out.jar"
  os.proc("jar", "-cfe", jarPath, "foo.Foo", ".")
    .call(cwd = classFiles().path)

  PathRef(T.dest / "out.jar")
}
```

```
> ./mill show assembly
Generating classfiles
Generating assembly
".../out/assembly.dest/out.jar"

> ./mill show assembly
".../out/assembly.dest/out.jar"

> echo "package hello2" > src/hello2.java
```

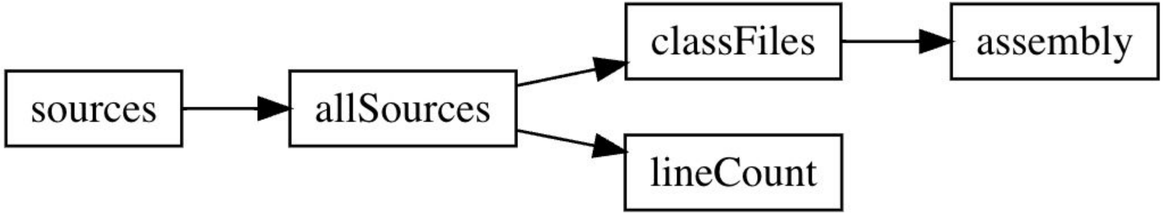
3.1 Tasks



```scala
def classFiles = T {
  println("Generating classfiles")
  val javacArgs = allSources().map(_.path)
  os.proc("javac", javacArgs, "-d", T.dest)
    .call(cwd = T.dest)

  PathRef(T.dest)
}

def assembly = T {
  println("Generating assembly")
  val jarPath = T.dest / "out.jar"
  os.proc("jar", "-cfe", jarPath, "foo.Foo", ".")
    .call(cwd = classFiles().path)

  PathRef(T.dest / "out.jar")
}
```

```
> ./mill show assembly
Generating classfiles
Generating assembly
".../out/assembly.dest/out.jar"

> ./mill show assembly
".../out/assembly.dest/out.jar"

> echo "package hello2" > src/hello2.java

> ./mill show assembly
Generating classfiles
Generating assembly
".../out/assembly.dest/out.jar"
```

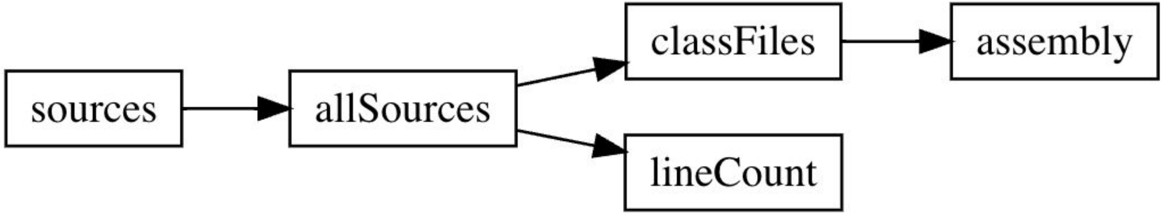
3.1 Tasks



```scala
def classFiles = T {
  println("Generating classfiles")
  val javacArgs = allSources().map(_.path)
  os.proc("javac", javacArgs, "-d", T.dest)
    .call(cwd = T.dest)

  PathRef(T.dest)
}

def assembly = T {
  println("Generating assembly")
  val jarPath = T.dest / "out.jar"
  os.proc("jar", "-cfe", jarPath, "foo.Foo", ".")
    .call(cwd = classFiles().path)

  PathRef(T.dest / "out.jar")
}
```

```
> ./mill show assembly
Generating classfiles
Generating assembly
".../out/assembly.dest/out.jar"

> ./mill show assembly
".../out/assembly.dest/out.jar"

> echo "package hello2" > src/hello2.java

> ./mill show assembly
Generating classfiles
Generating assembly
".../out/assembly.dest/out.jar"

> java -jar out/assembly.dest/out.jar  i am cow
Foo.value: 31337
args: i am cow
```

3.1 Tasks



```
// build.sc
import mill._

def sources = T.source { millSourcePath / "src" }

def allSources = T {
  os.walk(sources().path)
    .filter(_.ext == "java")
    .map(PathRef(_))
}

def lineCount: T[Int] = T {
  println("Computing line count")
  allSources()
    .map(p => os.read.lines(p.path).size)
    .sum
}
```

```
def classFiles = T {
  println("Generating classfiles")
  val javacArgs = allSources().map(_.path.toString
  os.proc("javac", javacArgs, "-d", T.dest)
    .call(cwd = T.dest)

  PathRef(T.dest)
}

def assembly = T {
  println("Generating jar")
  val jarPath = T.dest / "out.jar"
  os.proc("jar", "-cfe", jarPath, "foo.Foo", ".")
    .call(cwd = classFiles().path)

  PathRef(T.dest / "out.jar")
}
```

3.Mill Fundamentals

1. Tasks

2. **Modules**

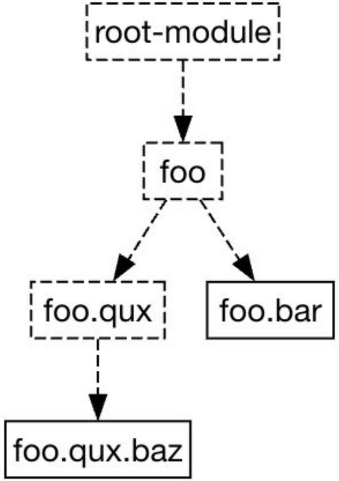
3. DIY Java Modules

3.2 Modules

```
// build.sc
import mill._

object foo extends Module {
  def bar = T { "hello" }
  object qux extends Module {
    def baz = T { "world" }
  }
}
```

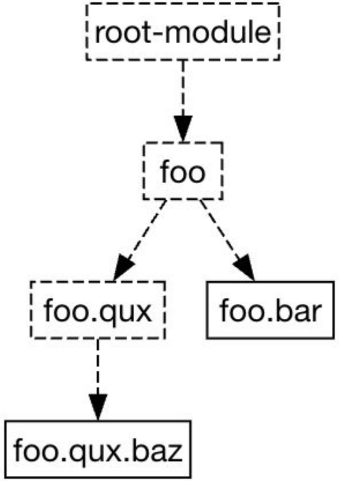
3.2 Modules

```
// build.sc
import mill._

object foo extends Module {
  def bar = T { "hello" }
  object qux extends Module {
    def baz = T { "world" }
  }
}
```



3.2 Modules

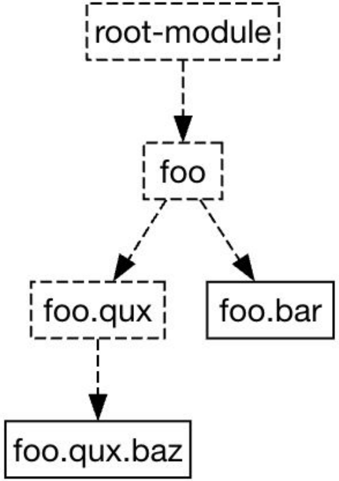
```
// build.sc
import mill._

object foo extends Module {
  def bar = T { "hello" }
  object qux extends Module {
    def baz = T { "world" }
  }
}
```



```
> ./mill show foo.bar
"hello"

> ./mill show foo.qux.baz
"world"
```

3.2 Modules

```
// build.sc
import mill._

object foo extends Module {
  def bar = T { "hello" }
  object qux extends Module {
    def baz = T { "world" }
  }
}
```



```
> ./mill show foo.bar
"hello"

> ./mill show foo.qux.baz
"world"

> cat out/foo/bar.json
..."value": "hello"...

> cat out/foo/qux/baz.json
..."value": "world"...
```

3.2 Modules

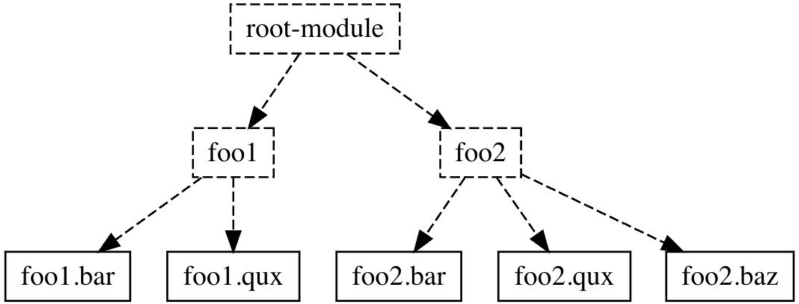
```
// build.sc
import mill._

trait FooModule extends Module {
  def bar: T[String] // required override
  def qux = T { bar() + " world" }
}
```

3.2 Modules

```
// build.sc
import mill._

trait FooModule extends Module {
  def bar: T[String] // required override
  def qux = T { bar() + " world" }
}

object foo1 extends FooModule{
  def bar = "hello"
  def qux = super.qux().toUpperCase
}
object foo2 extends FooModule {
  def bar = "hi"
  def baz = T { qux() + " I am Cow" }
}
```

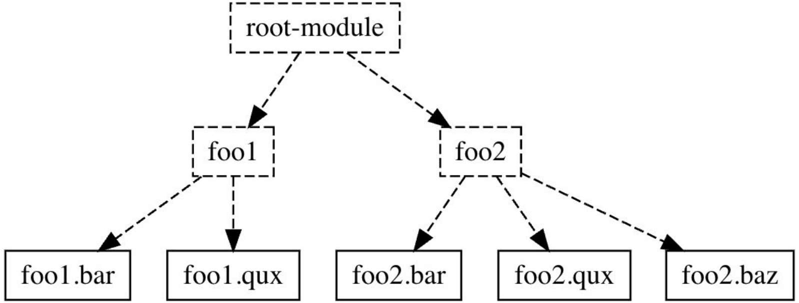
3.2 Modules

```
// build.sc
import mill._

trait FooModule extends Module {
  def bar: T[String] // required override
  def qux = T { bar() + " world" }
}

object foo1 extends FooModule{
  def bar = "hello"
  def qux = super.qux().toUpperCase
}
object foo2 extends FooModule {
  def bar = "hi"
  def baz = T { qux() + " I am Cow" }
}
```



3.2 Modules

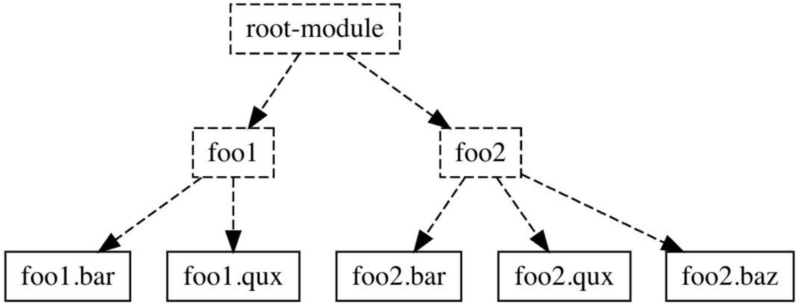
```
// build.sc
import mill._

trait FooModule extends Module {
  def bar: T[String] // required override
  def qux = T { bar() + " world" }
}

object foo1 extends FooModule{
  def bar = "hello"
  def qux = super.qux().toUpperCase
}
object foo2 extends FooModule {
  def bar = "hi"
  def baz = T { qux() + " I am Cow" }
}
```



```
> ./mill show foo1.bar
"hello"
```

3.2 Modules

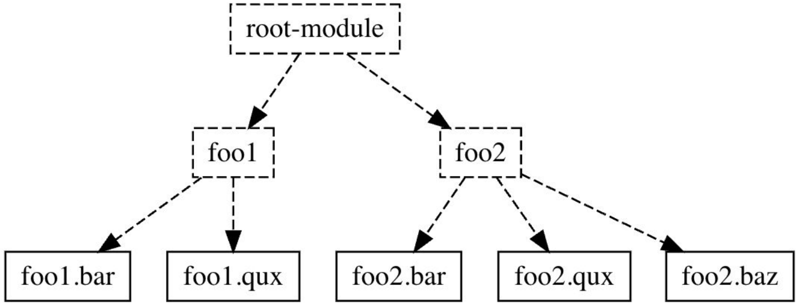
```
// build.sc
import mill._

trait FooModule extends Module {
  def bar: T[String] // required override
  def qux = T { bar() + " world" }
}

object foo1 extends FooModule{
  def bar = "hello"
  def qux = super.qux().toUpperCase
}
object foo2 extends FooModule {
  def bar = "hi"
  def baz = T { qux() + " I am Cow" }
}
```



```
> ./mill show foo1.bar
"hello"

> ./mill show foo1.qux
"HELLO WORLD"
```

3.2 Modules

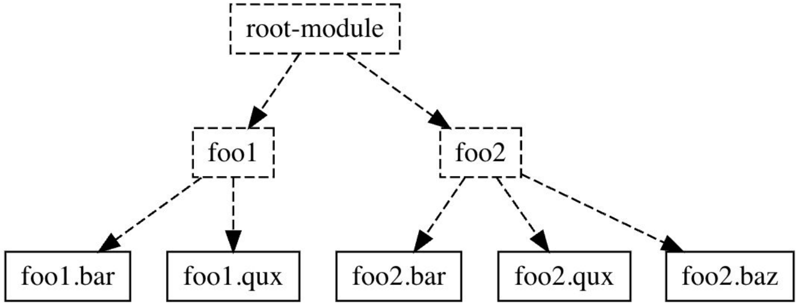
```
// build.sc
import mill._

trait FooModule extends Module {
  def bar: T[String] // required override
  def qux = T { bar() + " world" }
}

object foo1 extends FooModule{
  def bar = "hello"
  def qux = super.qux().toUpperCase
}
object foo2 extends FooModule {
  def bar = "hi"
  def baz = T { qux() + " I am Cow" }
}
```



```
> ./mill show foo1.bar
"hello"

> ./mill show foo1.qux
"HELLO WORLD"

> ./mill show foo2.qux
"hi world"
```

3.2 Modules

```
// build.sc
import mill._

trait FooModule extends Module {
  def bar: T[String] // required override
  def qux = T { bar() + " world" }
}

object foo1 extends FooModule{
  def bar = "hello"
  def qux = super.qux().toUpperCase
}
object foo2 extends FooModule {
  def bar = "hi"
  def baz = T { qux() + " I am Cow" }
}
```



```
> ./mill show foo1.bar
"hello"

> ./mill show foo1.qux
"HELLO WORLD"

> ./mill show foo2.qux
"hi world"

> ./mill show foo2.baz
"hi world I am Cow"
```

3.Mill Fundamentals

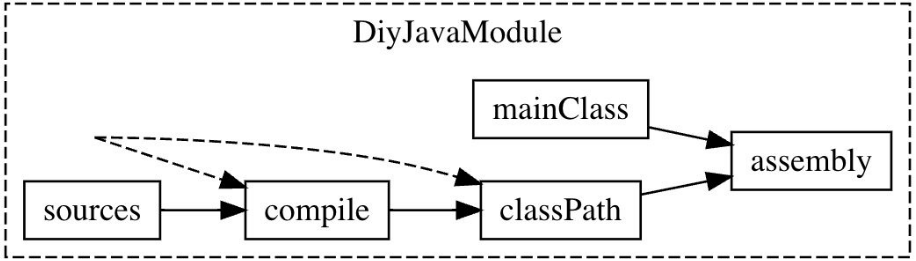
1. Tasks

2. Modules

3. **DIY Java Modules**

3.3 DIY Java Modules

```scala
import mill._
trait DiyJavaModule extends Module{
  def moduleDeps: Seq[DiyJavaModule] = Nil
  def mainClass: T[Option[String]] = None
  def upstream: T[Seq[PathRef]] = T{
    T.traverse(moduleDeps)(_.classPath)().flatten
  }
  def sources = T.source(millSourcePath / "src")
  def compile = T {
    val allSources = os.walk(sources().path)
    val cpFlag = upstream().map(_.path).mkString(":")
    os.proc("javac", "-cp", cpFlag, allSources, "-d", T.dest).call()
    PathRef(T.dest)
  }
  def classPath = T{ Seq(compile()) ++ upstream() }
  def assembly = T {
    for(cp <- classPath()) os.copy(cp.path, T.dest, mergeFolders = true)

    val mainFlags = mainClass().toSeq.flatMap(Seq("-e", _))
    os.proc("jar", "-c", mainFlags, "-f", T.dest / s"out.jar", ".")
      .call(cwd = T.dest)

    PathRef(T.dest / s"out.jar")
  }
}
```



3.3 DIY Java Modules

```scala
import mill._
trait DiyJavaModule extends Module{
  def moduleDeps: Seq[DiyJavaModule] = Nil
  def mainClass: T[Option[String]] = None
  def upstream: T[Seq[PathRef]] = T{
    T.traverse(moduleDeps)(_.classPath)().flatten
  }
  def sources = T.source(millSourcePath / "src")
  def compile = T {
    val allSources = os.walk(sources().path)
    val cpFlag = upstream().map(_.path).mkString(":")
    os.proc("javac", "-cp", cpFlag, allSources, "-d", T.dest).call()
    PathRef(T.dest)
  }
  def classPath = T{ Seq(compile()) ++ upstream() }
  def assembly = T {
    for(cp <- classPath()) os.copy(cp.path, T.dest, mergeFolders = true)

    val mainFlags = mainClass().toSeq.flatMap(Seq("-e", _))
    os.proc("jar", "-c", mainFlags, "-f", T.dest / s"out.jar", ".")
      .call(cwd = T.dest)

    PathRef(T.dest / s"out.jar")
  }
}
```

```scala
object foo extends DiyJavaModule {
  def moduleDeps = Seq(bar)
  def mainClass = Some("foo.Foo")

  object bar extends DiyJavaModule
}

object qux extends DiyJavaModule {
  def moduleDeps = Seq(foo)
  def mainClass = Some("qux.Qux")
}
```

3.3 DIY Java Modules

```scala
import mill._
trait DiyJavaModule extends Module{
  def moduleDeps: Seq[DiyJavaModule] = Nil
  def mainClass: T[Option[String]] = None
  def upstream: T[Seq[PathRef]] = T{
    T.traverse(moduleDeps)(_.classPath)().flatten
  }
  def sources = T.source(millSourcePath / "src")
  def compile = T {
    val allSources = os.walk(sources().path)
    val cpFlag = upstream().map(_.path).mkString(":")
    os.proc("javac", "-cp", cpFlag, allSources, "-d", T.dest).call()
    PathRef(T.dest)
  }
  def classPath = T{ Seq(compile()) ++ upstream() }
  def assembly = T {
    for(cp <- classPath()) os.copy(cp.path, T.dest, mergeFolders = true)

    val mainFlags = mainClass().toSeq.flatMap(Seq("-e", _))
    os.proc("jar", "-c", mainFlags, "-f", T.dest / s"out.jar", ".")
      .call(cwd = T.dest)

    PathRef(T.dest / s"out.jar")
  }
}
```

```scala
object foo extends DiyJavaModule {
  def moduleDeps = Seq(bar)
  def mainClass = Some("foo.Foo")

  object bar extends DiyJavaModule
}

object qux extends DiyJavaModule {
  def moduleDeps = Seq(foo)
  def mainClass = Some("qux.Qux")
}
```

```
> ./mill show qux.assembly
".../out/qux/assembly.dest/qux.jar"
```

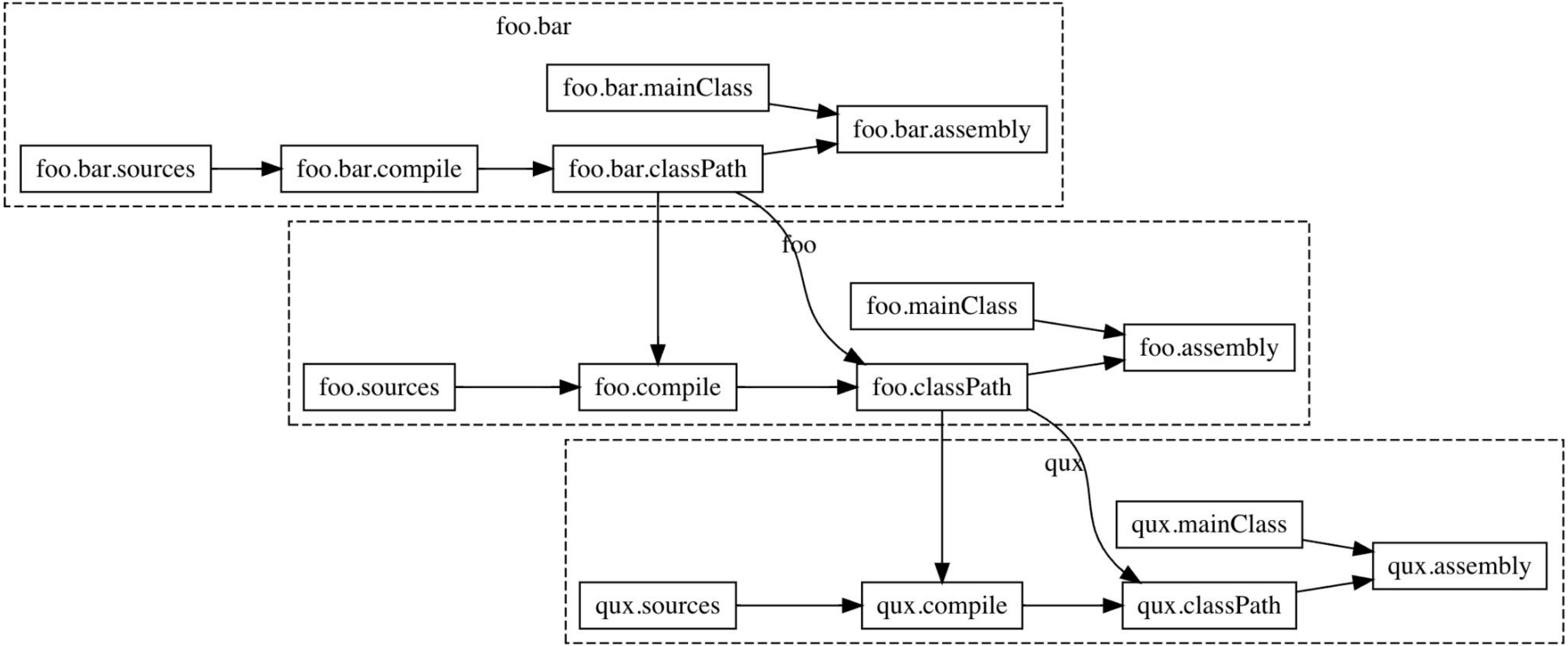
3.3 DIY Java Modules

```scala
import mill._
trait DiyJavaModule extends Module{
  def moduleDeps: Seq[DiyJavaModule] = Nil
  def mainClass: T[Option[String]] = None
  def upstream: T[Seq[PathRef]] = T{
    T.traverse(moduleDeps)(_.classPath)().flatten
  }
  def sources = T.source(millSourcePath / "src")
  def compile = T {
    val allSources = os.walk(sources().path)
    val cpFlag = upstream().map(_.path).mkString(":")
    os.proc("javac", "-cp", cpFlag, allSources, "-d", T.dest).call()
    PathRef(T.dest)
  }
  def classPath = T{ Seq(compile()) ++ upstream() }
  def assembly = T {
    for(cp <- classPath()) os.copy(cp.path, T.dest, mergeFolders = true)

    val mainFlags = mainClass().toSeq.flatMap(Seq("-e", _))
    os.proc("jar", "-c", mainFlags, "-f", T.dest / s"out.jar", ".")
      .call(cwd = T.dest)

    PathRef(T.dest / s"out.jar")
  }
}
```

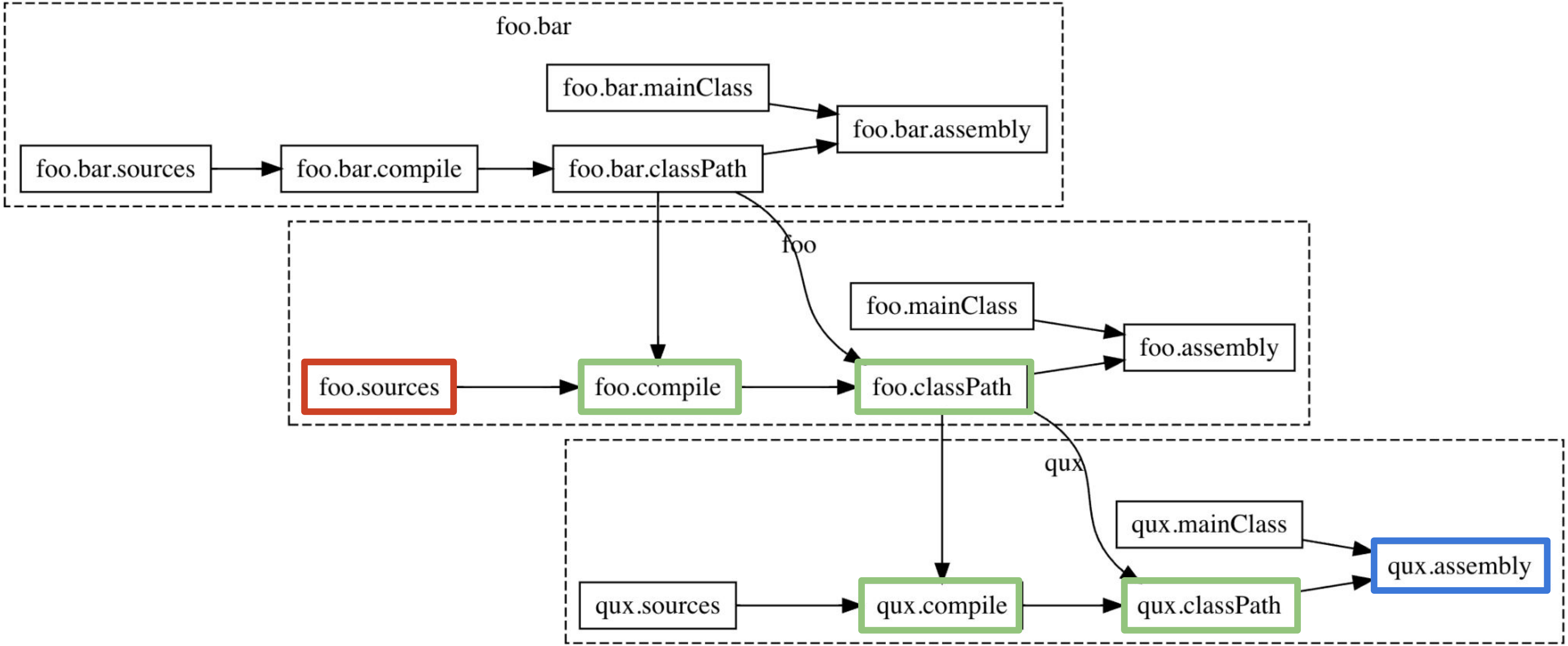
```scala
object foo extends DiyJavaModule {
  def moduleDeps = Seq(bar)
  def mainClass = Some("foo.Foo")

  object bar extends DiyJavaModule
}

object qux extends DiyJavaModule {
  def moduleDeps = Seq(foo)
  def mainClass = Some("qux.Qux")
}
```

```
> ./mill show qux.assembly
".../out/qux/assembly.dest/qux.jar"

> java -jar out/qux/assembly.dest/qux.jar
Foo.value: 31337
Bar.value: 271828
Qux.value: 9000
```

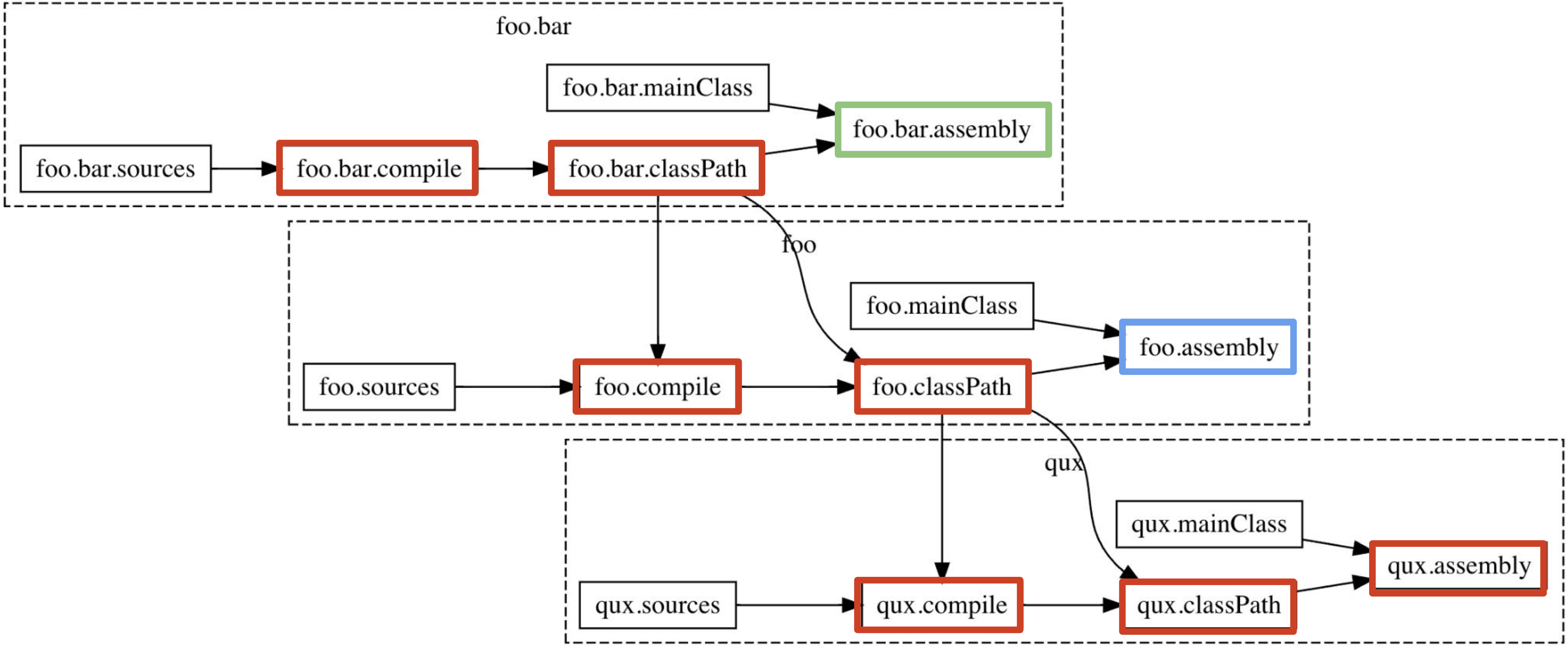
3.3 DIY Java Modules



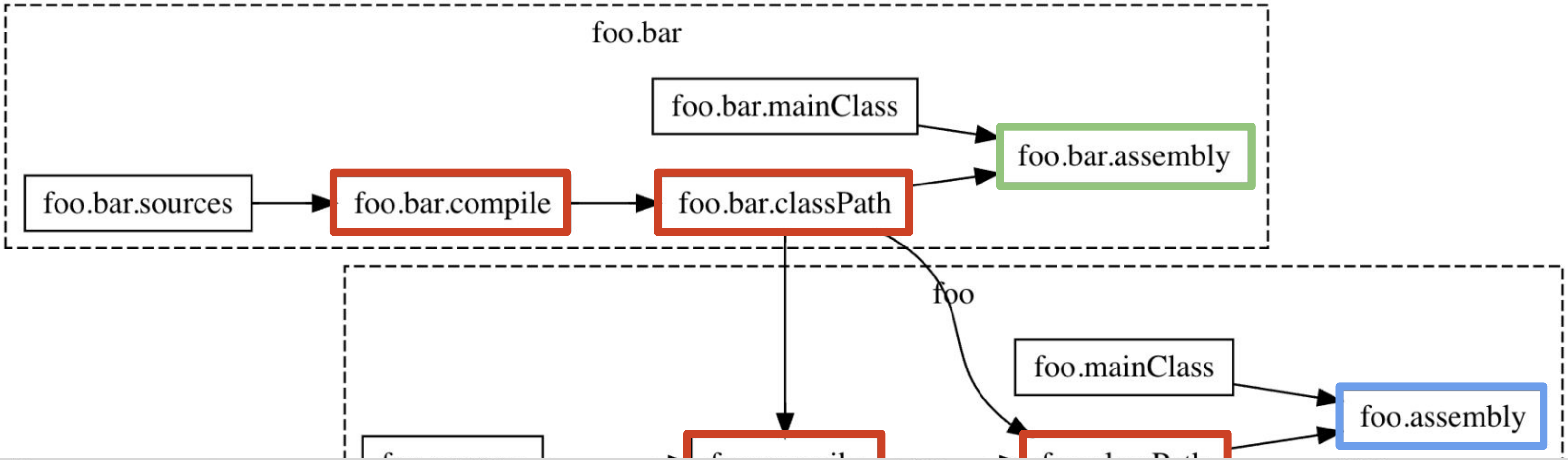
3.3 DIY Java Modules

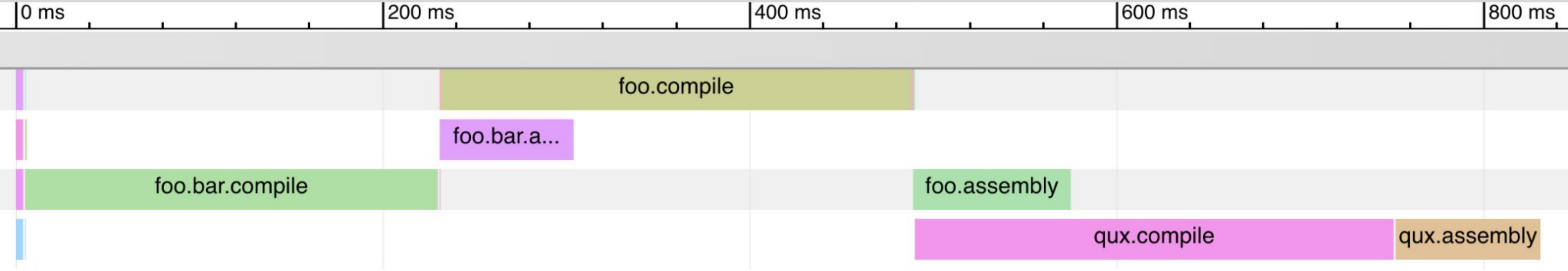


3.3 DIY Java Modules

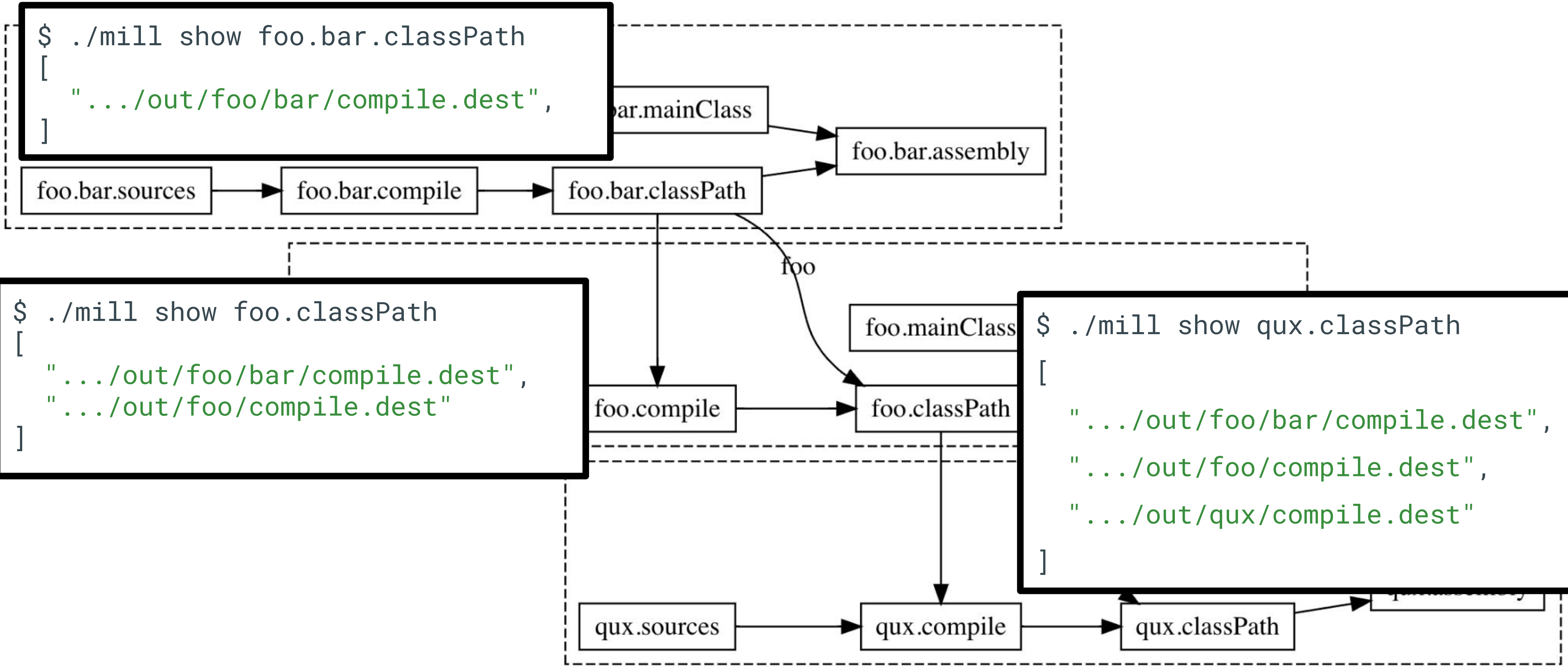


3.3 DIY Java Modules





3.3 DIY Java Modules

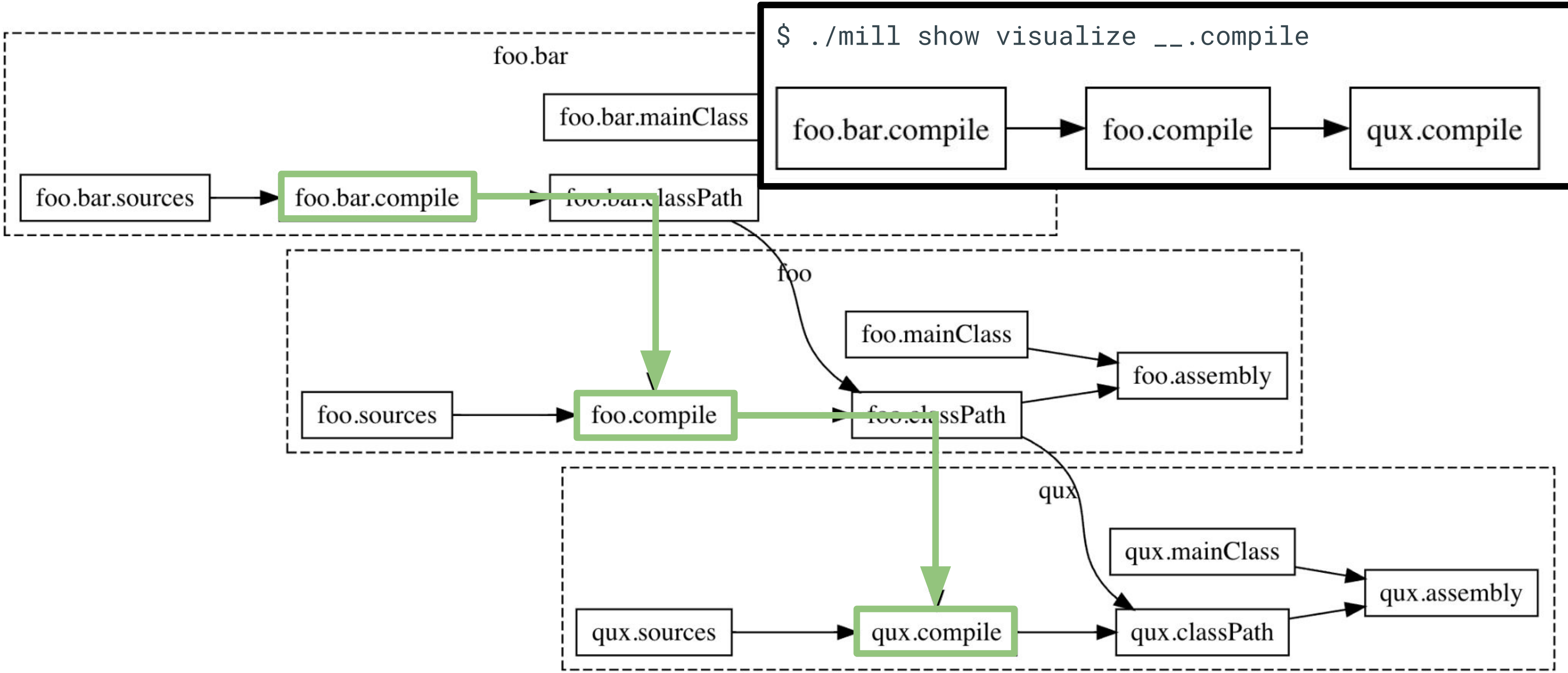


```
$ ./mill show foo.bar.classPath
[
  ".../out/foo/bar/compile.dest",
]
```

```
$ ./mill show foo.classPath
[
  ".../out/foo/bar/compile.dest",
  ".../out/foo/compile.dest"
]
```

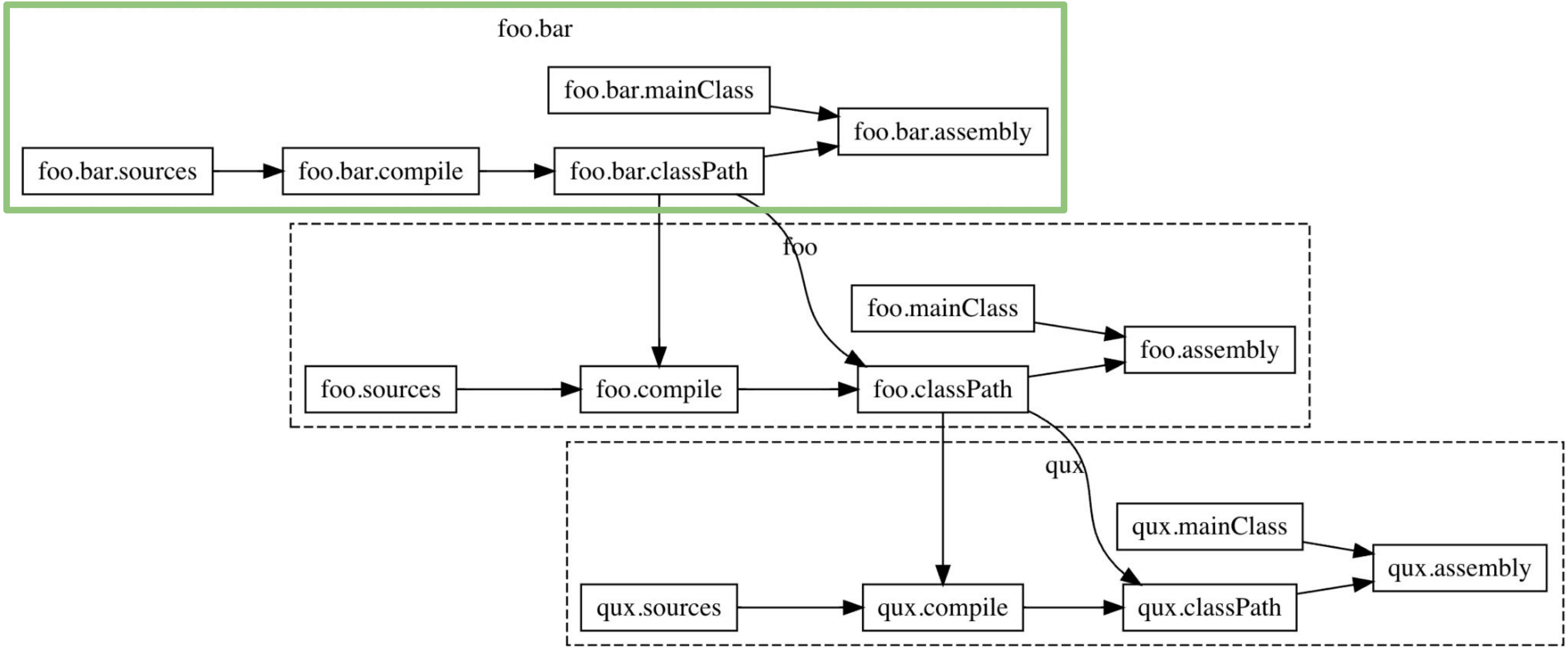
```
$ ./mill show qux.classPath
[
  ".../out/foo/bar/compile.dest",
  ".../out/foo/compile.dest",
  ".../out/qux/compile.dest"
]
```

3.3 DIY Java Modules

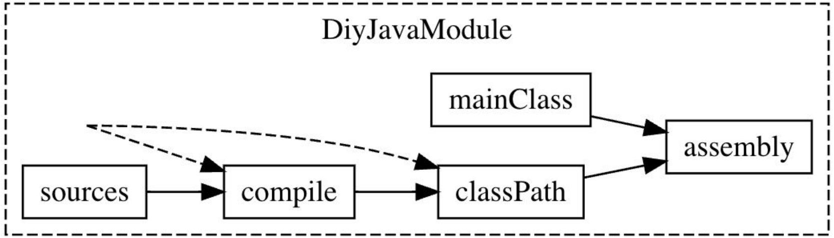


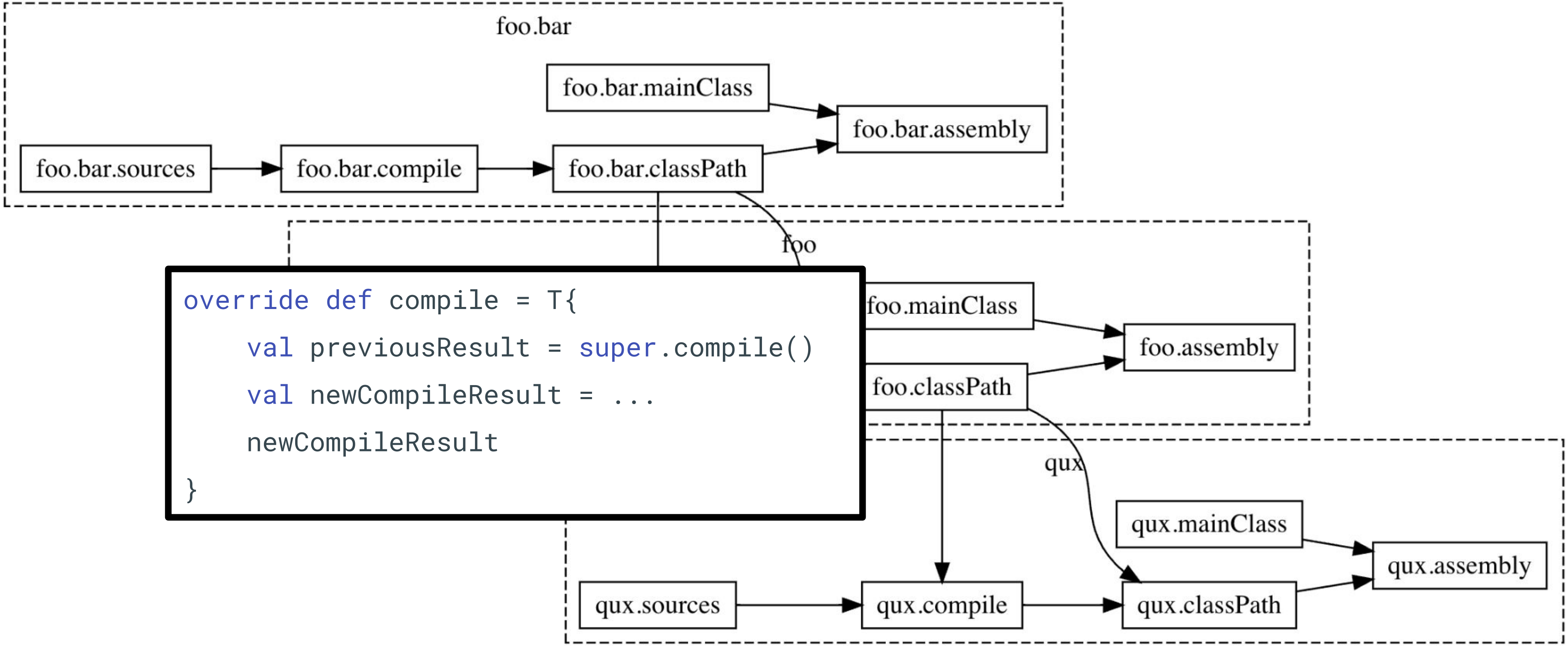
```
$ ./mill show visualize __.compile
```

3.3 DIY Java Modules



3.3 DIY Java Modules

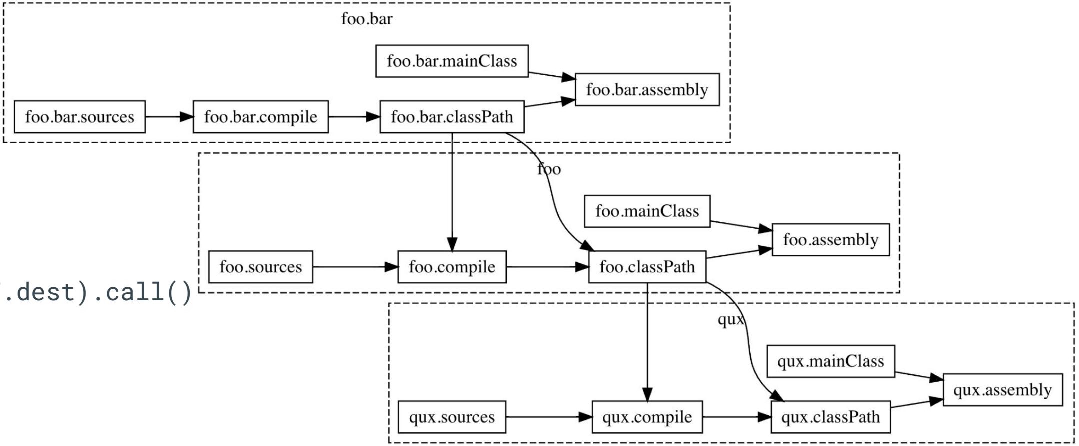




```
override def compile = T{
    val previousResult = super.compile()
    val newCompileResult = ...
    newCompileResult
}
```

3.3 DIY Java Modules

```scala
import mill._
trait DiyJavaModule extends Module{
  def moduleDeps: Seq[DiyJavaModule] = Nil
  def mainClass: T[Option[String]] = None
  def upstream: T[Seq[PathRef]] = T{
    T.traverse(moduleDeps)(_.classPath)().flatten
  }
  def sources = T.source(millSourcePath / "src")
  def compile = T {
    val allSources = os.walk(sources().path)
    val cpFlag = upstream().map(_.path).mkString(":")
    os.proc("javac", "-cp", cpFlag, allSources, "-d", T.dest).call()
    PathRef(T.dest)
  }
  def classPath = T{ Seq(compile()) ++ upstream() }
  def assembly = T {
    for(cp <- classPath()) os.copy(cp.path, T.dest, mergeFolders = true)

    val mainFlags = mainClass().toSeq.flatMap(Seq("-e", _))
    os.proc("jar", "-c", mainFlags, "-f", T.dest / s"out.jar", ".")
      .call(cwd = T.dest)

    PathRef(T.dest / s"out.jar")
  }
}
```



3.3 DIY Java Modules

```scala
// build.sc
import mill._, scalalib._

trait MyModule extends ScalaModule {
  def scalaVersion = "2.13.11"
}

object foo extends MyModule {
  def moduleDeps = Seq(bar)
  def ivyDeps = Agg(ivy"com.lihaoyi::mainargs:0.4.0")
}

object bar extends MyModule {
  def ivyDeps = Agg(ivy"com.lihaoyi::scalatags:0.8.2")
}
```

```scala
object foo extends DiyJavaModule {
  def moduleDeps = Seq(bar)
  def mainClass = Some("foo.Foo")

  object bar extends DiyJavaModule
}

object qux extends DiyJavaModule {
  def moduleDeps = Seq(foo)
  def mainClass = Some("qux.Qux")
}
```

A Deep Dive into the Mill Scala Build Tool

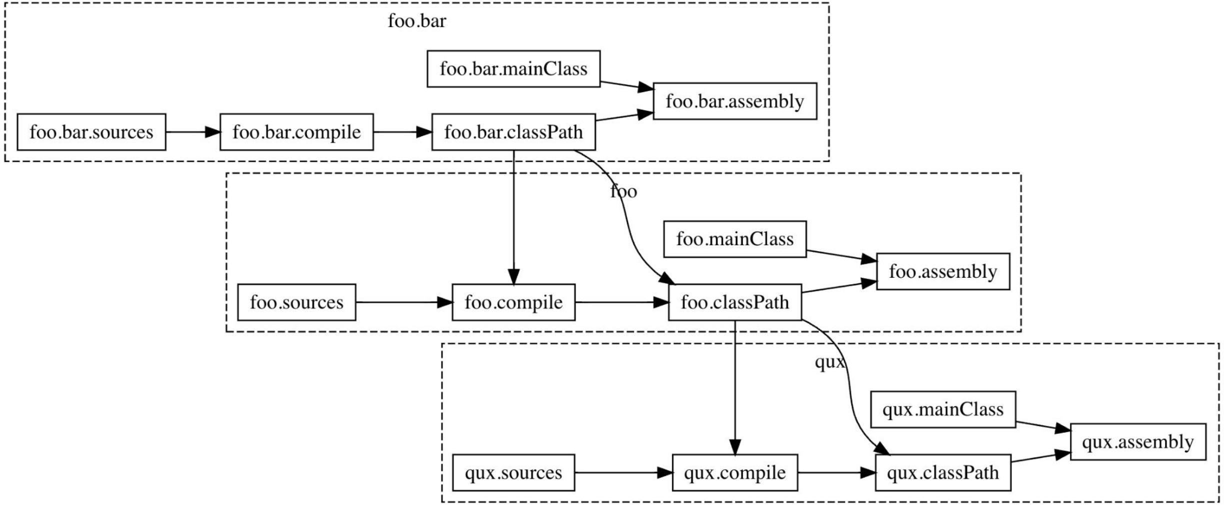
1. Why Build Tools Are Hard

2. Getting Started with Mill

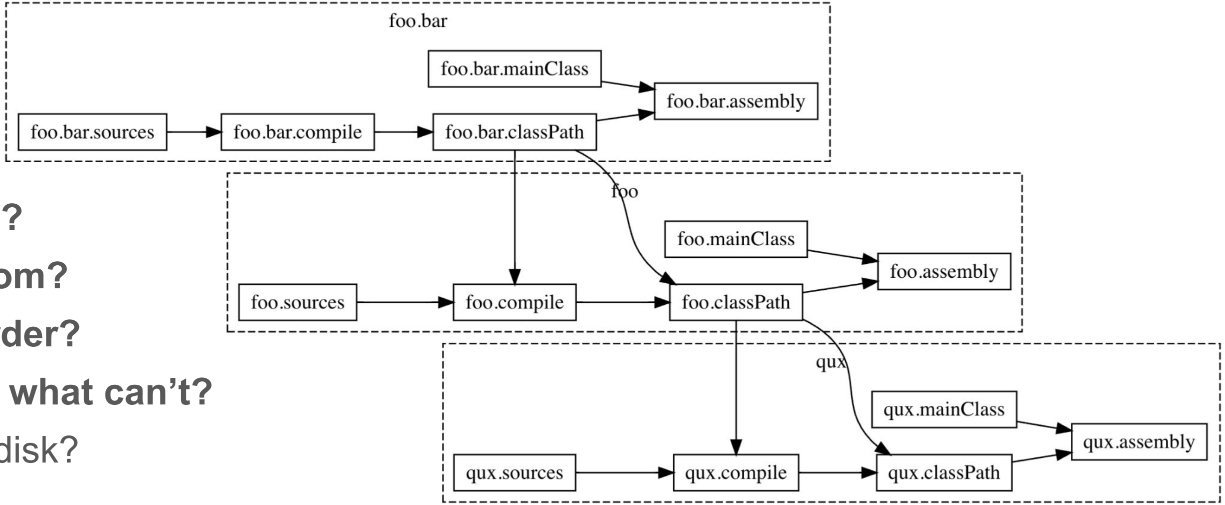
3. Mill Fundamentals

4. **Why Mill Works**

4.1 The Task Graph

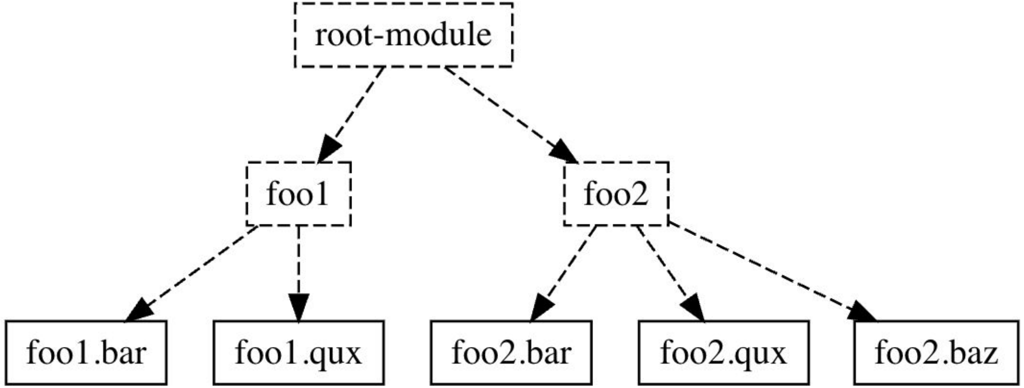


4.1 The Task Graph



1. **What tasks depends on what?**
2. **Where do input files come from?**
3. **What needs to run in what order?**
4. **What can be parallelized and what can't?**
5. Where can tasks read/write to disk?
6. How are tasks cached?
7. How are tasks run from the CLI?
8. How are cross-builds (across different configurations) handled?
9. How do I define my own custom tasks?
10. How do tasks pass data to each other?
11. How to manage the repetition in a build?
12. What is a "Module"? How do they relate to "Tasks"?
13. How do you customize a task or module to do something different?
14. What APIs do tasks use to actually do things?
15. How is in-memory caching handled?

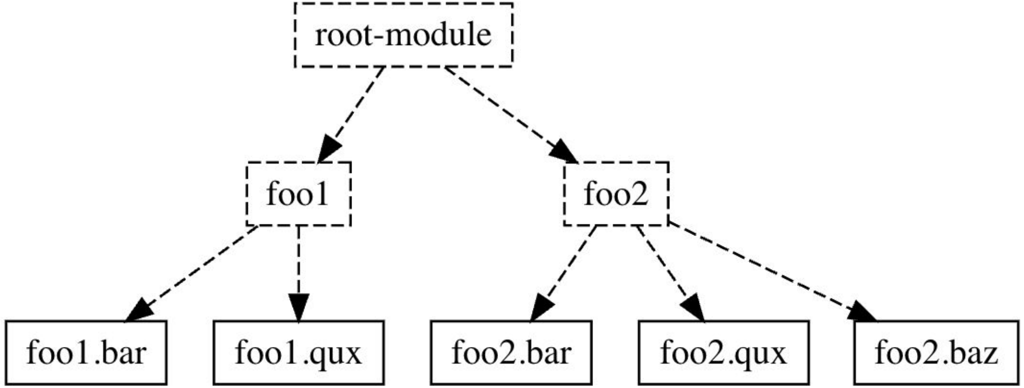
4.2 The Module Tree



1. ~~What tasks depends on what?~~
2. ~~Where do input files come from?~~
3. ~~What needs to run in what order?~~
4. ~~What can be parallelized and what can't?~~
5. Where can tasks read/write to disk?
6. How are tasks cached?
7. How are tasks run from the CLI?
8. How are cross-builds (across different configurations) handled?
9. How do I define my own custom tasks?
10. How do tasks pass data to each other?
11. How to manage the repetition in a build?
12. What is a "Module"? How do they relate to "Tasks"?
13. How do you customize a task or module to do something different?
14. What APIs do tasks use to actually do things?
15. How is in-memory caching handled?

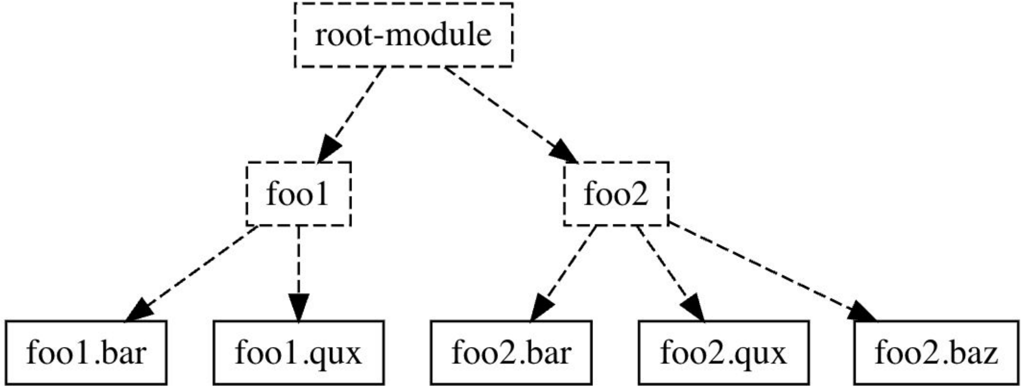
4.2 The Module Tree

1. ~~What tasks depends on what?~~
2. ~~Where do input files come from?~~
3. ~~What needs to run in what order?~~
4. ~~What can be parallelized and what can't?~~
5. Where can tasks read/write to disk?
6. How are tasks cached?
7. How are tasks run from the CLI?
8. How are cross-builds (across different configurations) handled?
9. How do I define my own custom tasks?
10. How do tasks pass data to each other?
11. How to manage the repetition in a build?
12. What is a "Module"? How do they relate to "Tasks"?
13. How do you customize a task or module to do something different?
14. What APIs do tasks use to actually do things?
15. How is in-memory caching handled?

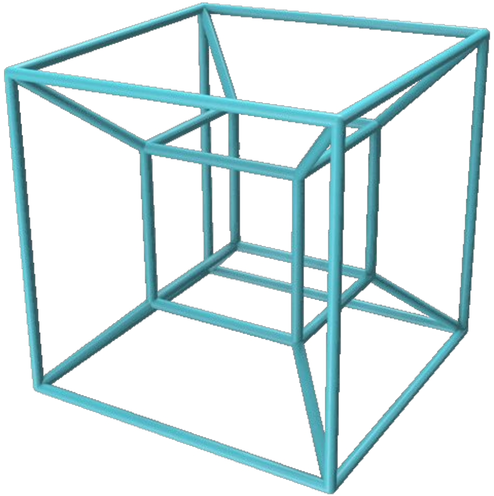


```
my-project/
    foo1/
        src/
            ...
        resources/
            ...
    foo2/
        src/
            ...
        resources/
            ...
```

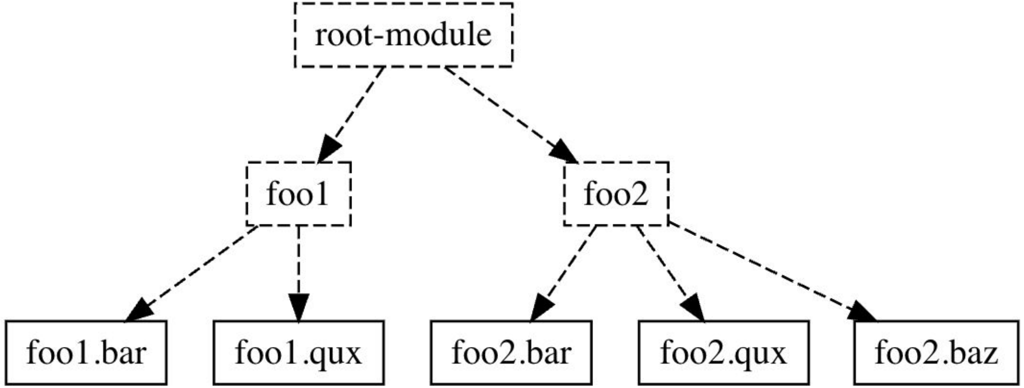
4.2 The Module Tree



1. ~~What tasks depends on what?~~
2. ~~Where do input files come from?~~
3. ~~What needs to run in what order?~~
4. ~~What can be parallelized and what can't?~~
5. Where can tasks read/write to disk?
6. How are tasks cached?
7. How are tasks run from the CLI?
8. How are cross-builds (across different configurations) handled?
9. How do I define my own custom tasks?
10. How do tasks pass data to each other?
11. How to manage the repetition in a build?
12. What is a "Module"? How do they relate to "Tasks"?
13. How do you customize a task or module to do something different?
14. What APIs do tasks use to actually do things?
15. How is in-memory caching handled?



4.2 The Module Tree



1. ~~What tasks depends on what?~~
2. ~~Where do input files come from?~~
3. ~~What needs to run in what order?~~
4. ~~What can be parallelized and what can't?~~
5. **Where can tasks read/write to disk?**
6. **How are tasks cached?**
7. **How are tasks run from the CLI?**
8. **How are cross-builds (across different configurations) handled?**
9. How do I define my own custom tasks?
10. How do tasks pass data to each other?
11. How to manage the repetition in a build?
12. What is a "Module"? How do they relate to "Tasks"?
13. How do you customize a task or module to do something different?
14. What APIs do tasks use to actually do things?
15. How is in-memory caching handled?

4.2 Embedding into Scala

```scala
import mill._, scalalib._
object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles().map(f => os.read.lines(f.path).size).sum
  }

  override def resources = T{
    os.write(T.dest / "line-count.txt", "" + lineCount())
    Seq(PathRef(T.dest))
  }
}
```

1. ~~What tasks depends on what?~~
2. ~~Where do input files come from?~~
3. ~~What needs to run in what order?~~
4. ~~What can be parallelized and what can't?~~
5. ~~Where can tasks read/write to disk?~~
6. ~~How are tasks cached?~~
7. ~~How are tasks run from the CLI?~~
8. ~~How are cross-builds (across different configurations) handled?~~
9. How do I define my own custom tasks?
10. How do tasks pass data to each other?
11. How to manage the repetition in a build?
12. What is a "Module"? How do they relate to "Tasks"?
13. How do you customize a task or module to do something different?
14. What APIs do tasks use to actually do things?
15. How is in-memory caching handled?

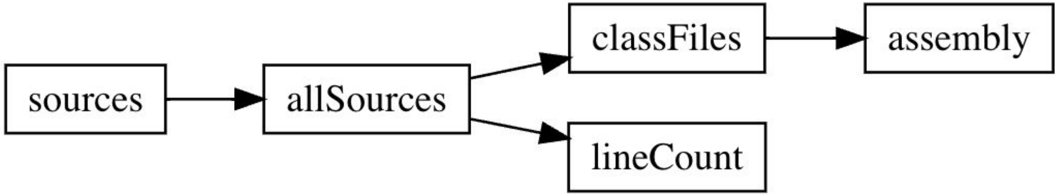
4.2 Embedding into Scala

```scala
import mill._, scalalib._
object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles().map(f => os.read.lines(f.path).size).sum
  }

  override def resources = T{
    os.write(T.dest / "line-count.txt", "" + lineCount())
    Seq(PathRef(T.dest))
  }
}
```

1. ~~What tasks depends on what?~~
2. ~~Where do input files come from?~~
3. ~~What needs to run in what order?~~
4. ~~What can be parallelized and what can't?~~
5. ~~Where can tasks read/write to disk?~~
6. ~~How are tasks cached?~~
7. ~~How are tasks run from the CLI?~~
8. ~~How are cross-builds (across different configurations) handled?~~
9. **How do I define my own custom tasks?**
10. **How do tasks pass data to each other?**
11. **How to manage the repetition in a build?**
12. **What is a "Module"? How do they relate to "Tasks"?**
13. **How do you customize a task or module to do something different?**
14. **What APIs do tasks use to actually do things?**
15. **How is in-memory caching handled?**

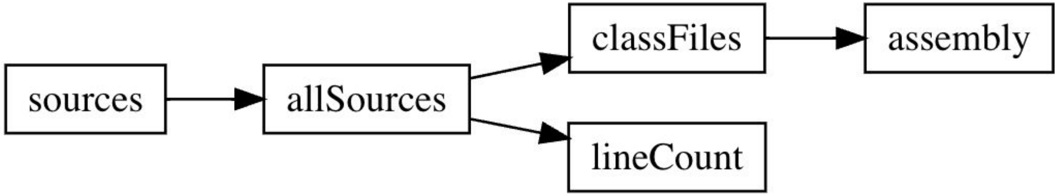
4.2 Embedding into Scala

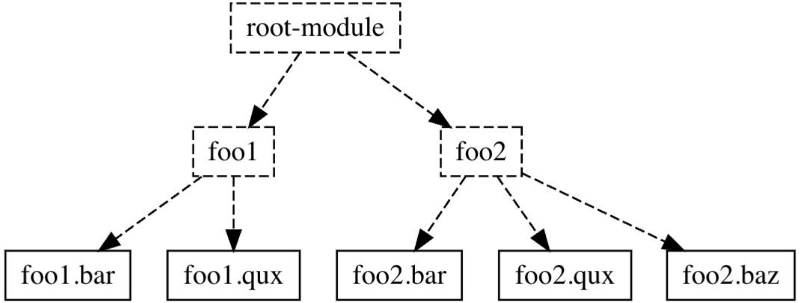
```
def sources = T.source { ... }

def allSources = T { ...sources()... }

def lineCount= T { ...allSources()... }

def classFiles = T {  ...allSources()...}

def assembly = T { ...classFiles()...}
```



4.2 Embedding into Scala

```
def sources = T.source { ... }


def allSources = T { ...sources()... }


def lineCount= T { ...allSources()... }


def classFiles = T {  ...allSources()...}


def assembly = T { ...classFiles()...}
```





```
trait FooModule extends Module {
  def qux = ...
}


object foo1 extends FooModule{
  def bar = ...
  def qux = ...
}
object foo2 extends FooModule {
  def bar = ...
  def baz = ...
  // def qux = ...
}
```

4.2 Embedding into Scala

1. What tasks depends on what?
2. Where do input files come from?
3. What needs to run in what order?
4. What can be parallelized and what can’t?
5. Where can tasks read/write to disk?
6. How are tasks cached?
7. How are tasks run from the CLI?
8. How are cross-builds (across different configurations) handled?
9. How do I define my own custom tasks?
10. How do tasks pass data to each other?
11. How to manage the repetition in a build?
12. What is a "Module"? How do they relate to "Tasks"?
13. How do you customize a task or module to do something different?
14. What APIs do tasks use to actually do things?
15. How is in-memory caching handled?

```scala
object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles().map(f => os.read.lines(f.path).size).sum
  }

  override def resources = T{
    os.write(T.dest / "line-count.txt", "" + lineCount())
    Seq(PathRef(T.dest))
  }
}
```

A Deep Dive into the Mill Scala Build Tool

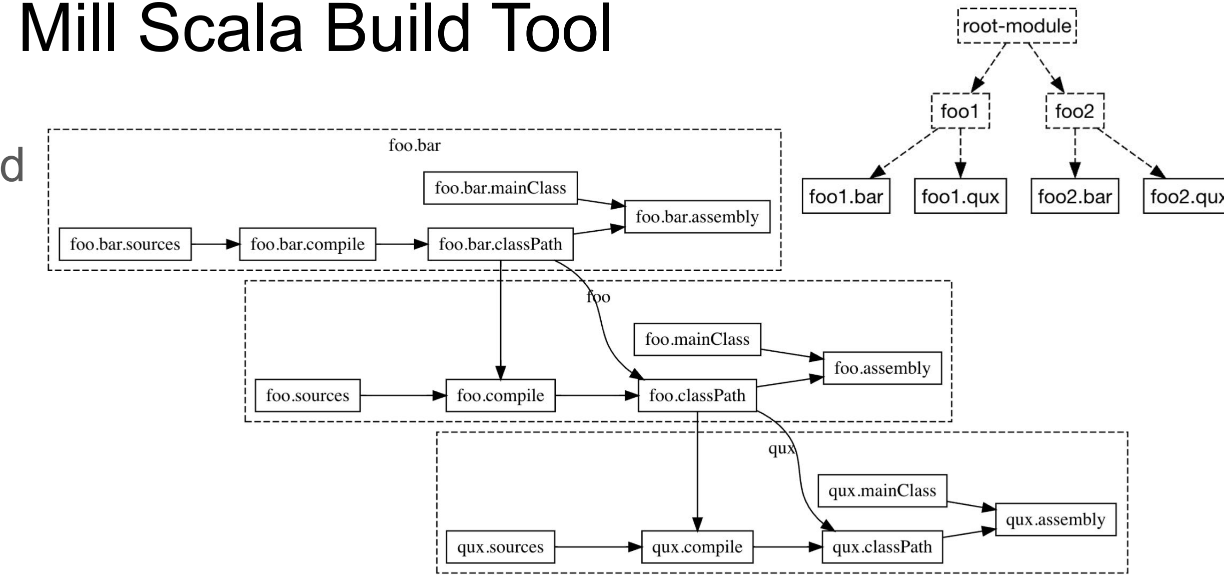
1. Why Build Tools Are Hard

2. Getting Started with Mill

3. Mill Fundamentals

4. Why Mill Works

https://mill-build.com/



```
import mill._, scalalib._
object foo extends RootModule with ScalaModule {
  def scalaVersion = "2.13.11"

  def lineCount = T{
    allSourceFiles().map(f => os.read.lines(f.path).size).sum
  }
```