Mill: A Build Tool based on Pure Functional Programming

Li Haoyi, Scala.Love 18 April 2020
(another 3 minutes, we haven’t started yet)

Who am I?

Developer Tools at Databricks

- Heavy users of Scala, Bazel build tool
- >1MLOC of Scala, Python, Jsonnet, Javascript, Docker, Kubernetes, Cloudformation, ...

Previously at Dropbox

- Python webdev! Coffeescript webdev!
- Developer tooling, integration testing, static program analysis, ..

Open source

- Ammonite, Mill, Fastparse, Scalatags, uPickle,
- Requests-Scala, OS-Lib, PPrint, Fansi, Cask,
- uTest, sourcecode, ...

Agenda

What is the Mill Build Tool?

What is Functional Programming All About?

How are Build Tools similar to FP?

How are Build Tools *not* similar to FP?

How the Mill Build Tool uses FP

Agenda

<u>What is the Mill Build Tool?</u>

What is Functional Programming All About?

How are Build Tools similar to FP?

How are Build Tools *not* similar to FP?

How the Mill Build Tool uses FP

What is Mill

An embedded Scala library, built on top of Ammonite Scala Scripts

Heavily inspired by SBT, Bazel, Scala.Rx, other things

Just plain Scala*

Simple Build

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

object foo extends ScalaModule{
  def scalaVersion = "2.13.1"
}
```

Simple Build

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

object foo extends ScalaModule{
  def scalaVersion = "2.13.1"
}
```

Module in foo/, sources in foo/src/

Set Scala version to 2.13

Simple Build

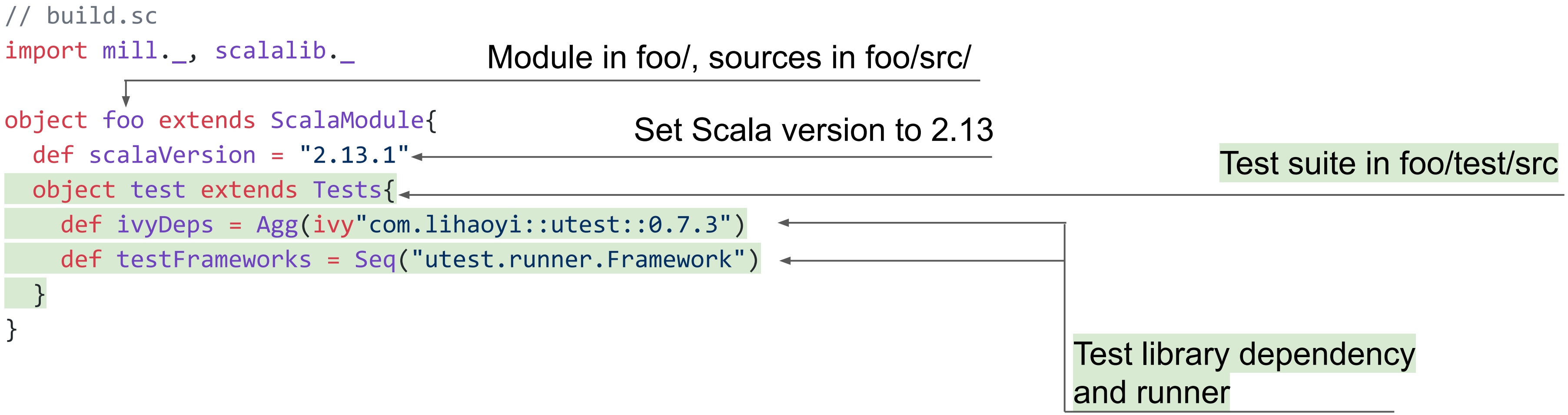
```scala
// build.sc
import mill._, scalalib._

object foo extends ScalaModule{
  def scalaVersion = "2.13.1"
}
```

Module in foo/, sources in foo/src/

Set Scala version to 2.13

```
$ mill foo.compile
$ mill foo.run
$ mill foo.assembly
```

Simple Build with test



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

object foo extends ScalaModule{
  def scalaVersion = "2.13.1"
  object test extends Tests{
    def ivyDeps = Agg(ivy"com.lihaoyi::utest:0.7.3")
    def testFrameworks = Seq("utest.runner.Framework")
  }
}
```

Module in foo/, sources in foo/src/

Set Scala version to 2.13

Test suite in foo/test/src

Test library dependency and runner

```
$ mill foo.compile
$ mill foo.run
$ mill foo.assembly
$ mill foo.test
$ mill foo.test.compile
```

Multi-module build

```
import mill._, scalalib._

trait AppModule extends ScalaModule{
  def scalaVersion = "2.13.1"
}

object shared extends AppModule

object foo extends AppModule{
  def moduleDeps = Seq(shared)
}
object bar extends AppModule{
  def moduleDeps = Seq(shared)
}
```

```
$ mill shared.compile
$ mill shared.run
$ mill shared.assembly

$ mill foo.compile
$ mill foo.run
$ mill foo.assembly

$ mill bar.compile
$ mill bar.run
$ mill bar.assembly
```

Multi-module build

```scala
import mill._, scalalib._

trait AppModule extends ScalaModule{
  def scalaVersion = "2.13.1"
}

object shared extends AppModule

object foo extends AppModule{
  def moduleDeps = Seq(shared)
  def resources = T.sources{
   os.copy(bar.assembly().path, T.dest / "bar.jar")
   Seq(PathRef(T.dest))
  }
}
object bar extends AppModule{
  def moduleDeps = Seq(shared)
}
```

```
$ mill shared.compile
$ mill shared.run
$ mill shared.assembly

$ mill foo.compile
$ mill foo.run
$ mill foo.assembly

$ mill bar.compile
$ mill bar.run
$ mill bar.assembly
```

Make foo include bar’s assembly jar in its resources

Things Mill Does

Resolve libraryDependencies to make Dependency Jars

Compile Source files and Dependency Jars to make Class files

Run Code Generation to make Generated Source Files

Test Class files and Dependency Jars to make Test Results

Zip Class files to make Jars

Zip Class files and Dependency Jars to make Assemblies

Package Jars and Dependency Jars to make Docker Containers

...

Why Mill is interesting?

Fast: background daemon is the default, keeps JVM warm and responsive

- Response times ~200ms

Out of the box functionality: compile, test, executable assemblies, publishing, …

- No plugins required, provides everything you need!

Battle-tested in the Wild

- Used in the Ammonite build (179 submodules), Kotlin/Java builds, static site generators, web asset pipelines, constructing PDFs, ...

Intuitive and trivially customizable

- Write plain-old-Scala, get all the good stuff free: caching, file-watching, parallelism, ...
- *“ I am so happy every time I have to tweak a little build stuff and find that my project uses Mill! Build system easy mode FTW”* - Rex Kerr

Agenda

What is the Mill Build Tool?

What is Functional Programming All About?

How are Build Tools similar to FP?

How are Build Tools *not* similar to FP?

How the Mill Build Tool uses FP

What’s Functional Programming All About?

Haskell?

Ocaml?

Clojure?

Scala?

F#?

Scheme?

What’s Functional Programming Not All About?

Macros/Metaprogramming?

Parentheses?

Powerful Type Systems?

Monads?

Writing Interpreters?

Constructing Programs?

What’s Functional Programming Not All About?

~~Macros/Metaprogramming?~~ *Scala, Haskell, OCaml don’t use macros all that much*

Parentheses?

Powerful Type Systems?

Monads?

Writing Interpreters?

Constructing Programs?

What’s Functional Programming Not All About?

~~Macros/Metaprogramming?~~ *Scala, Haskell, OCaml don’t use macros all that much*

~~Parentheses?~~ *Haskell, Ocaml, F# don’t have many parentheses*

Powerful Type Systems?

Monads?

Writing Interpreters?

Constructing Programs?

What’s Functional Programming Not All About?

~~Macros/Metaprogramming?~~ *Scala, Haskell, OCaml don’t use macros all that much*

~~Parentheses?~~ *Haskell, Ocaml, F# don’t have many parentheses*

~~Powerful Type Systems?~~ *Most Lisps have no typechecker*

Monads?

Writing Interpreters?

Constructing Programs?

What’s Functional Programming Not All About?

~~Macros/Metaprogramming?~~ *Scala, Haskell, OCaml don’t use macros all that much*

~~Parentheses?~~ *Haskell, Ocaml, F# don’t have many parentheses*

~~Powerful Type Systems?~~ *Most Lisps have no typechecker*

~~Monads?~~ *Ocaml/Lisp don’t use monads much*

Writing Interpreters?

Constructing Programs?

What’s Functional Programming Not All About?

~~Macros/Metaprogramming?~~ *Scala, Haskell, OCaml don’t use macros all that much*

~~Parentheses?~~ *Haskell, Ocaml, F# don’t have many parentheses*

~~Powerful Type Systems?~~ *Most Lisps have no typechecker*

~~Monads?~~ *Ocaml/Lisp don’t use monads much*

~~Writing Interpreters?~~ *Ocaml doesn’t do this much, most are written in C*

Constructing Programs?

What’s Functional Programming Not All About?

~~Macros/Metaprogramming?~~ *Scala, Haskell, OCaml don’t use macros all that much*

~~Parentheses?~~ *Haskell, Ocaml, F# don’t have many parentheses*

~~Powerful Type Systems?~~ *Most Lisps have no typechecker*

~~Monads?~~ *Ocaml/Lisp don’t use monads much*

~~Writing Interpreters?~~ *Ocaml doesn’t do this much, most are written in C*

~~Constructing Programs?~~ *PHP templated Javascript fragments aren’t FP*

Case Study: Michael Chu’s Classic Tiramisu

http://www.cookingforengineers.com/recipe/60/The-Classic-Tiramisu-original-recipe

Tiramisu Presented Two Ways: Imperative

1. Begin by assembling four large egg yolks, 1/2 cup sweet marsala wine, ...
2. Stir two tablespoons of granulated sugar into the espresso and put it in the refrigerator to chill.
3. Whisk the egg yolks
4. Pour in the sugar and wine and whisked briefly until it was well blended.
5. Pour some water into a saucepan and set it over high heat until it began to boil.
6. Lowering the heat to medium, place the heatproof bowl over the water and stirred as the mixture began to thicken and smooth out.
7. Whip the heavy cream until soft peaks.

...

Tiramisu Presented Two Ways: Imperative

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

    return eggs # it's now a tiramisu
```

Refactoring Imperative Recipes

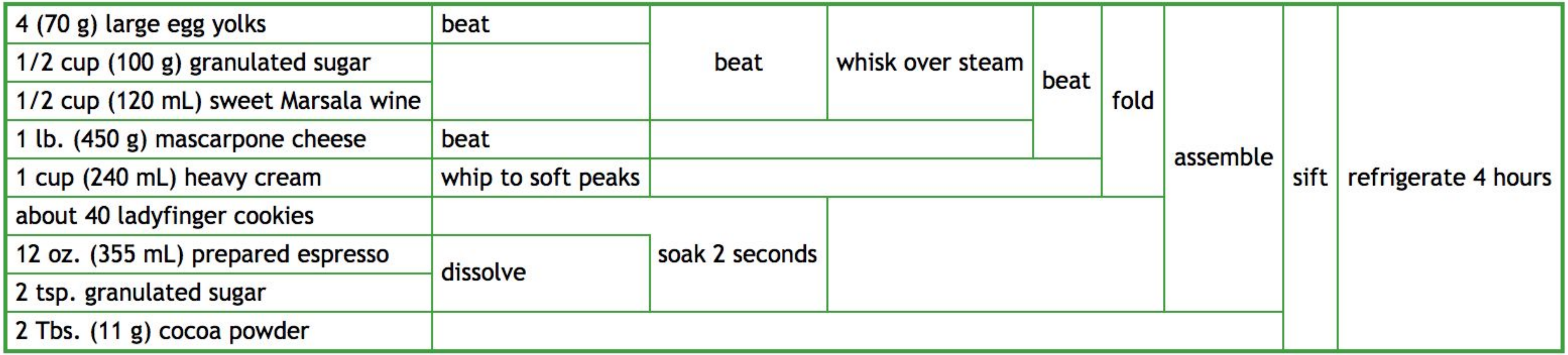
```
dissolve(sugar2, espresso)
whisk(eggs)
beat(eggs, sugar1, wine)
whisk(eggs) # over steam
whip(cream)
beat(cheese)
beat(eggs, cheese)
fold(eggs, cream)
assemble(eggs, fingers)
sift(eggs, cocoa)
refrigerate(eggs)

return eggs # it's now a tiramisu
```

If I have two people to make this tiramisu, which parts can I start working on in parallel?

After beating the eggs and cheese, I realize I bought the wrong kind of cream. What work do I need to re-do, and with what ingredients?

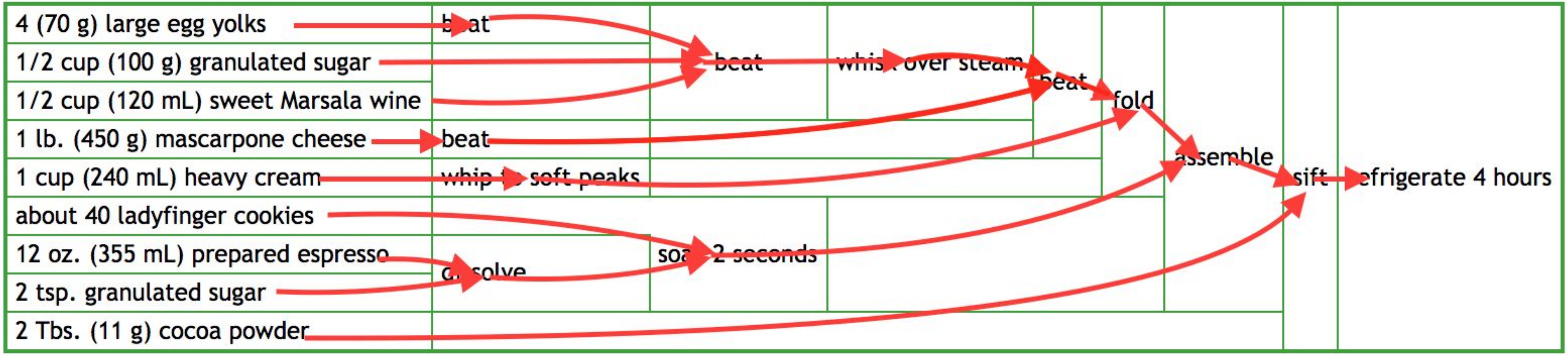
I spilled the bowl after beating in the mascapone cheese into the egg mixture; what ingredients do I need to recover?

Tiramisu Presented Two Ways: Functional



<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 colspan="3" rowspan="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>

Tiramisu Presented Two Ways: Functional

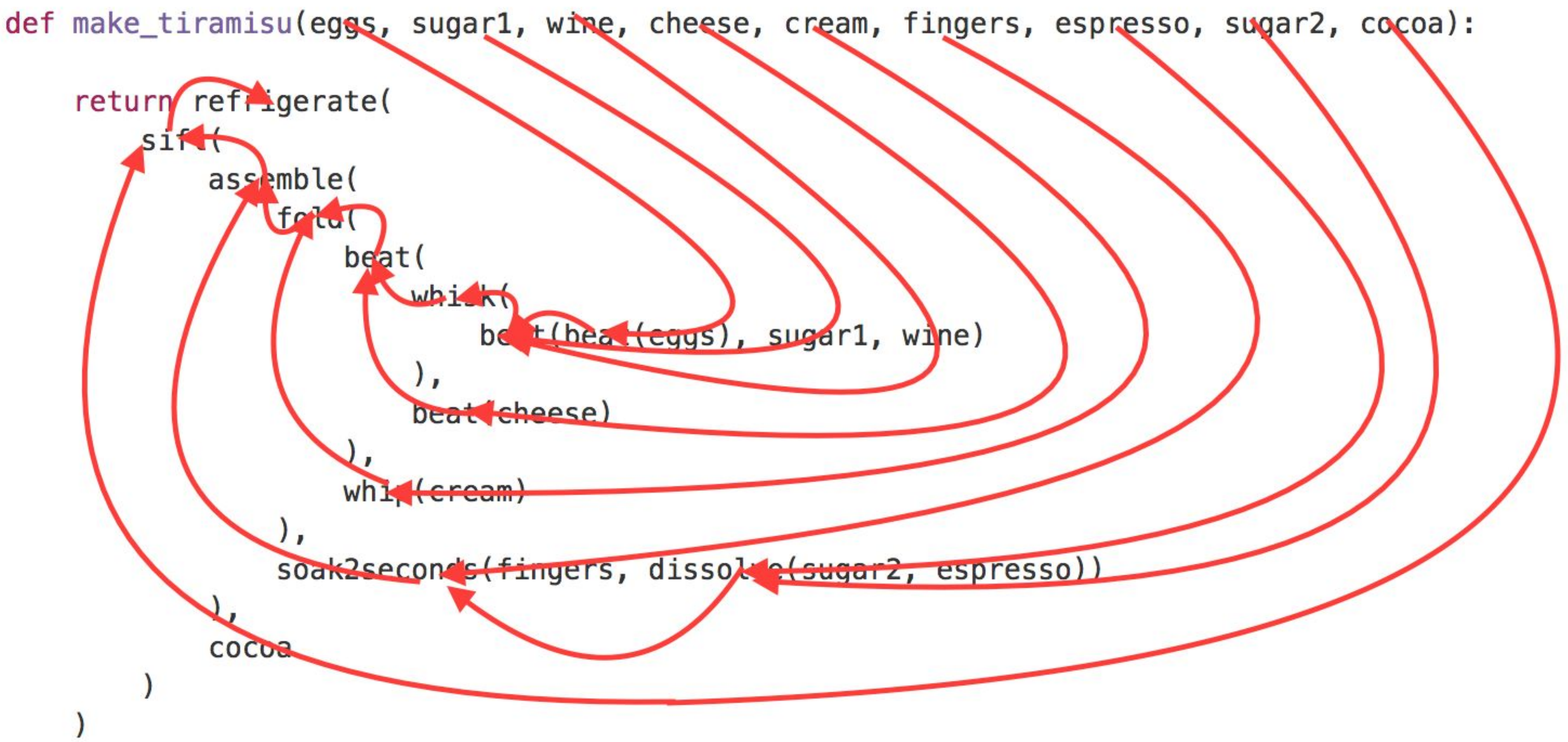


<table>
<tr><td>4 (70 g) large egg yolks</td><td>beat</td><td rowspan="3">beat</td><td rowspan="3">whisk over steam</td><td rowspan="4">beat</td><td rowspan="5">fold</td><td rowspan="8">assemble</td><td rowspan="9">sift</td><td rowspan="9">refrigerate 4 hours</td></tr>
<tr><td>1/2 cup (100 g) granulated 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>

Tiramisu Presented Two Ways: Functional

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

Tiramisu Presented Two Ways: Functional



```
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
        )
    )
```

Tiramisu Presented Two Ways: Functional

```
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
```

Tiramisu Presented Two Ways

Imperative

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

return eggs # it's now a tiramisu
```

Functional

```
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
```

FP is about Dataflow vs Control Flow

Imperative

- Sequence of Steps

- Unclear where the ordering matters, and where it doesn’t

- Dependencies between steps implicit in the ordering of the steps

Functional

- Dependency Graph of Steps

- Trivially obvious which steps must happen before each other steps

- Dependencies betweens steps explicit in the graph structure

Refactoring Functional Recipes

```
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
```

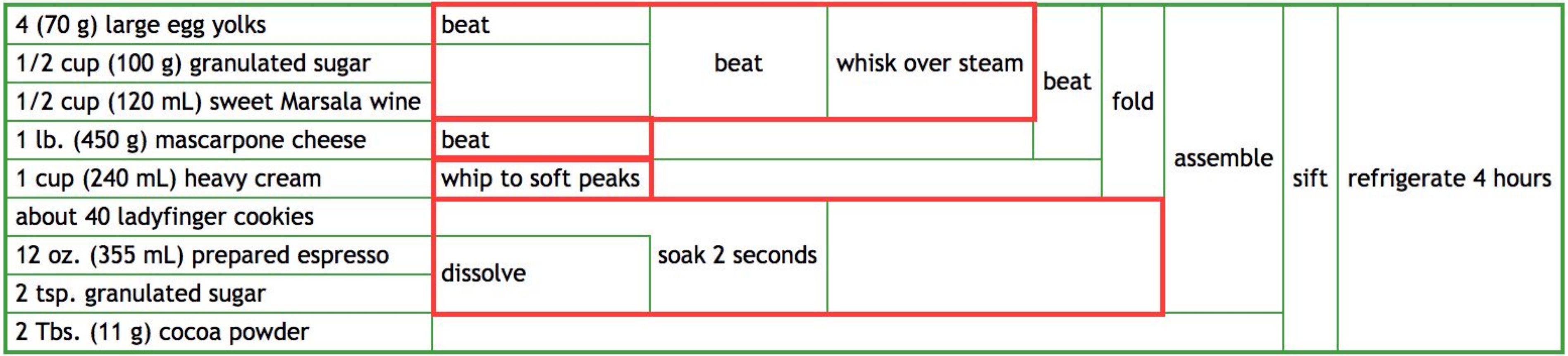
If I have two people to make this tiramisu, which parts can I start working on in parallel?

My expresso hasn't arrived yet; what can I start cooking first?

I spilled the bowl after beating in the mascapone cheese into the egg mixture; what ingredients do I need to recover?

Kitchen Refactoring

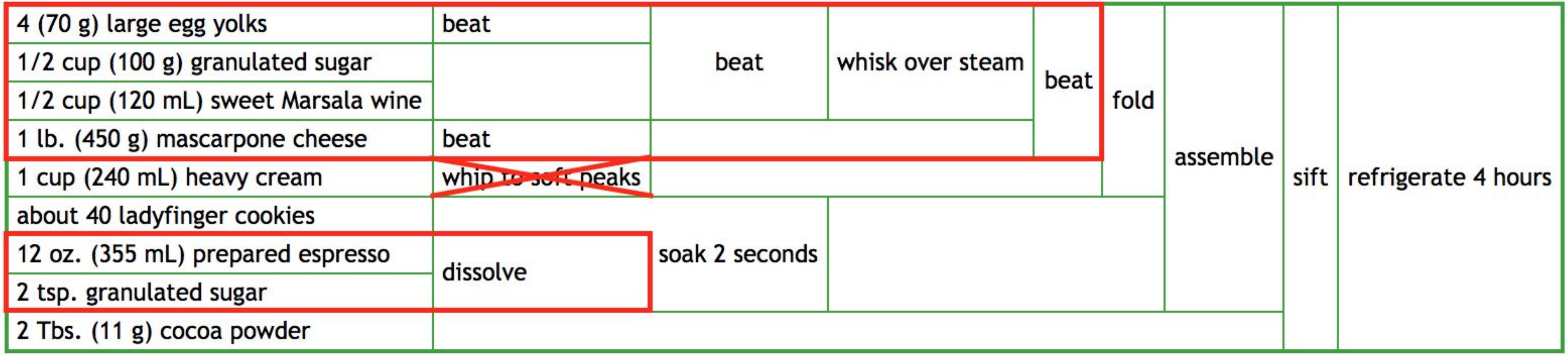
If I have two people to make this tiramisu, which parts can I start working on in parallel?



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

Kitchen Refactoring

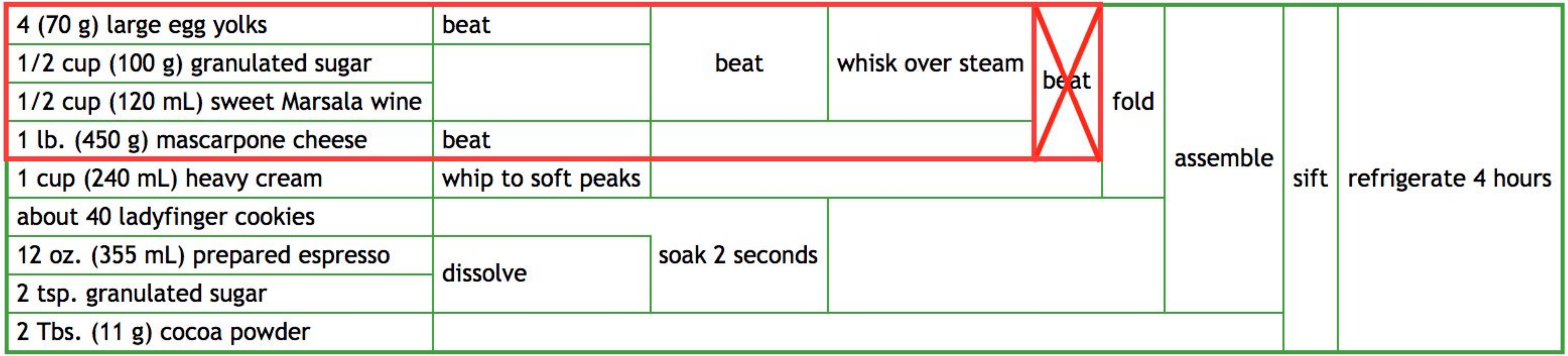
After beating the eggs and cheese, I realize I bought the wrong kind of cream. What work do I need to re-do, and with what ingredients?



4 (70 g) large egg yolks	beat	beat	whisk over steam	beat	fold	assemble	sift	refrigerate 4 hours
1/2 cup (100 g) granulated sugar								
1/2 cup (120 mL) sweet Marsala wine								
1 lb. (450 g) mascarpone cheese	beat							
1 cup (240 mL) heavy cream	~~whip to soft peaks~~							
about 40 ladyfinger cookies		soak 2 seconds						
12 oz. (355 mL) prepared espresso	dissolve							
2 tsp. granulated sugar								
2 Tbs. (11 g) cocoa powder								

Kitchen Refactoring

I spilled the bowl after beating in the mascapone cheese into the egg mixture; what ingredients do I need to recover?



4 (70 g) large egg yolks	beat	beat	whisk over steam	~~beat~~	fold	assemble	sift	refrigerate 4 hours
1/2 cup (100 g) granulated sugar								
1/2 cup (120 mL) sweet Marsala wine								
1 lb. (450 g) mascarpone cheese	beat							
1 cup (240 mL) heavy cream	whip to soft peaks							
about 40 ladyfinger cookies		soak 2 seconds						
12 oz. (355 mL) prepared espresso	dissolve							
2 tsp. granulated sugar								
2 Tbs. (11 g) cocoa powder								

Functional Programming is about Dataflow

We are thinking about the dependencies between parts of your program, rather than lists of steps that have to be done

Makes parallelization, re-ordering, and incremental re-computation trivial

Often serialized to a linear sequence of lines of code, but the core structure is the dataflow graph

Agenda

What is Mill?

What is Functional Programming All About?

How are Build Tools similar to FP?

How are Build Tools *not* similar to FP?

How the Mill Build Tool uses FP

What Are Build Tools About

Ant

Maven

Groovy

SBT

Bazel

Pants

Mill

...

Build Tools Do Things

Resolve libraryDependencies to make Dependency Jars

Compile Source files and Dependency Jars to make Class files

Run Code Generation to make Generated Source Files

Test Class files and Dependency Jars to make Test Results

Zip Class files to make Jars

Zip Class files and Dependency Jars to make Assemblies

Package Jars and Dependency Jars to make Docker Containers

...

Build Tools Do Things

Resolve libraryDependencies to make Dependency Jars

Compile Source files and Dependency Jars to make Class files

Run Code Generation to make Generated Source Files

Test Class files and Dependency Jars to make Test Results

Zip Class files to make Jars

Zip Class files and Dependency Jars to make Assemblies

Package Jars and Dependency Jars to make Docker Containers

...

Build Tools Do Things

```
$DO_THING to $FOO and $BAR to make $OUTPUT
```

Build Tools Do Things

```
OUTPUT = DO_THING(FOO, BAR)
```

Build Tool vs Tiramisu Requirements

Parallelize different build steps so the overall build completes faster

If I have two people to make this tiramisu, which parts can I start working on in parallel?

Someone changed a source file; what steps do I need to do to re-generate all artifacts that are affected by that change?

After beating the eggs and cheese, I realize I bought the wrong kind of cream. What work do I need to re-do, and with what ingredients?

Someone wants to run tests and re-run them every time their input files change; which files should we watch?

I spilled the bowl after beating in the mascapone cheese into the egg mixture; what ingredients do I need to recover?

How are Build Tools similar to FP?

Builds are largely* made up of of pure functions: `OUTPUT = DO_THING(FOO, BAR)`

The ease that FP allows re-factoring, parallelizing and analyzing your computation is exactly what a build tool needs to do!

While FP often helps humans do these things, a build tool needs to do them automatically, but they’re really the same things

Agenda

What is the Mill Build Tool?

What is Functional Programming All About?

How are Build Tools similar to FP?

How are Build Tools *not* similar to FP?

How the Mill Build Tool uses FP

How are Build Tools *not* similar to FP?

Introspectability

Persistence

Long-lived Workers

Ad-hoc Overrides

How are Build Tools *not* similar to FP?

Introspectability

Persistence

Long-lived Workers

Ad-hoc Overrides

Introspectability

```python
def make_tiramisu(eggs, sugar1, wine):
    beat_eggs = beat(eggs)
    mixture = beat(beat_eggs, sugar1, wine)
    whisked = whisk(mixture)
```

Pure functions are *opaque*, the only thing we can do is run them on their input

Introspectability

```python
def make_tiramisu(eggs, sugar1, wine):
    beat_eggs = beat(eggs)
    mixture = beat(beat_eggs, sugar1, wine)
    whisked = whisk(mixture)
```

Pure functions are *opaque*, the only thing we can do is run them on their input

Build tools need to inspect the structure of your computation: to parallelize steps, incrementally re-compute things, decide what inputs to watch

How are Build Tools *not* similar to FP?

Introspectability

Persistence

Long-lived Workers

Ad-hoc Overrides

Persistence

```python
def make_tiramisu(eggs, sugar1, wine):
    beat_eggs = beat(eggs)
    mixture = beat(beat_eggs, sugar1, wine)
    whisked = whisk(mixture)
```

Pure functions have their intermediate values live in memory

Persistence

```
def make_tiramisu(eggs, sugar1, wine):
    beat_eggs = beat(eggs)
    mixture = beat(beat_eggs, sugar1, wine)
    whisked = whisk(mixture)
```

Pure functions have their intermediate values live in memory

Build tools output for each step needs to be serialized (how?), and saved somewhere (where?) so the next time someone runs your build tool we can re-use the results that had been computed before

How are Build Tools *not* similar to FP?

Introspectability

Persistence

Long-lived Workers

Ad-hoc Overrides

Long-lived Workers

```python
def make_tiramisu(eggs, sugar1, wine):
    beat_eggs = beat(eggs)
    mixture = beat(beat_eggs, sugar1, wine)
    whisked = whisk(mixture)
```

Pure functions have no side effects and can be run any way we like

Long-lived Workers

```
def make_tiramisu(eggs, sugar1, wine):
    beat_eggs = beat(eggs)
    mixture = beat(beat_eggs, sugar1, wine)
    whisked = whisk(mixture)
```

Pure functions have no side effects and can be run any way we like

Build steps often require some amount of setup: your JVM needs to be warmed up, your incremental compilation caches populated, your wkhtmltopdf worker process spawned. Setup/teardown is often expensive, so we want to re-use things. Sometimes concurrency is limited/disallowed (e.g. due to resource limits)

How are Build Tools *not* similar to FP?

Introspectability

Persistence

Long-lived Workers

Ad-hoc Overrides

Ad-hoc Overrides

```
def make_tiramisu(eggs, sugar1, wine):
    beat_eggs = beat(eggs)
    mixture = beat(beat_eggs, sugar1, wine)
    whisked = whisk(mixture)
```

A pure function’s parameters contains of everything you can do to customize it

Ad-hoc Overrides

```python
def make_tiramisu(eggs, sugar1, wine):
    beat_eggs = beat(eggs)
    mixture = beat(beat_eggs, sugar1, wine)
    whisked = whisk(mixture)
```

A pure function’s parameters contains of everything you can do to customize it

Templated sequences of build steps often need ad-hoc customization: a custom artifactName here, special javacOptions there, some weird code-generation step over there, and for this *one specific module* we need to turn off incremental compilation due to bugs in the incremental compiler.

How are Build Tools *not* similar to FP?

Introspectability

Persistence

Long-lived Workers

Ad-hoc Overrides

Agenda

What is the Mill Build Tool?

What is Functional Programming All About?

How are Build Tools similar to FP?

How are Build Tools *not* similar to FP?

How the Mill Build Tool uses FP

Naive Pure FP vs Mill Build Logic

Naive Pure Function

```scala
def makeTiramisu(eggs, sugar1, wine) = {
  val beatEggs = beat(eggs)
  val mixture =
    beat(beatEggs, sugar1, wine)
  val whisked = whisk(mixture)
}
```

Mill Module

```scala
trait TiramisuModule extends Module{
  def eggs = T.input{...}
  def sugar1 = T.input{...}
  def wine = T.input{...}
  def beatEggs = T{ beat(eggs()) }
  def mixture = T{
    beat(beatEggs(), sugar1(), wine())
  }
  def whisked = T{ whisk(mixture()) }
}
```

How the Mill build tool uses FP, with...

Introspectability

Persistence

Long-lived Workers

Ad-hoc Overrides

How the Mill build tool uses FP, with...

Introspectability

Persistence

Long-lived Workers

Ad-hoc Overrides

Introspectability via Free Applicative

Mill turns build steps (“targets”) into a *Free Applicative* structure, using the T{...} macro

```
def makeTiramisu(eggs, sugar1, wine) = {
  def beatEggs = T{ beat(eggs()) }
  def mixture = T{ beat(beatEggs(), sugar1(), wine()) }
  def whisked = T{ whisk(mixture()) }
}
```

Introspectability via Free Applicative

Mill turns build steps (“targets”) into a *Free Applicative* structure, using the T{...} macro

```
def makeTiramisu(eggs, sugar1, wine) = {
  def beatEggs = T{ beat(eggs()) }
  def mixture = T{ beat(beatEggs(), sugar1(), wine()) }
  def whisked = T{ whisk(mixture()) }
}
```

```
def makeTiramisu(eggs, sugar1, wine) = {
  val beatEggs = beat(eggs)
  val mixture = beat(beatEggs, sugar1, wine)
  val whisked = whisk(mixture)
}
```

Introspectability via Free Applicative

Mill turns build steps (“targets”) into a *Free Applicative* structure, using the T{...} macro

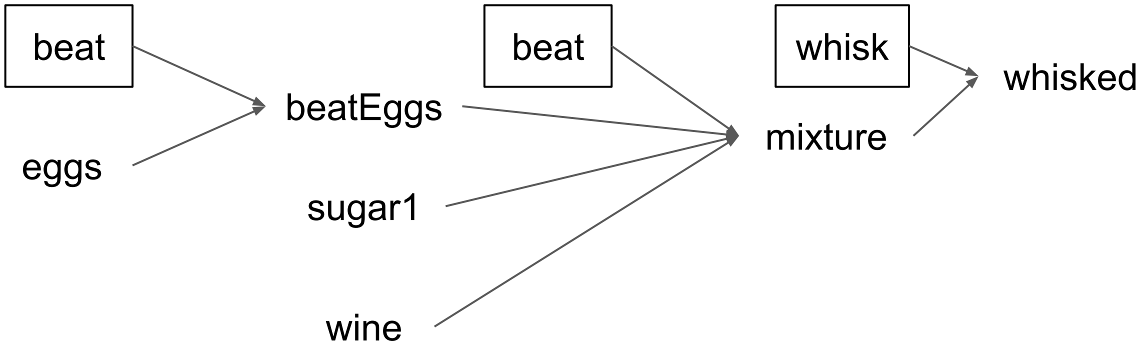
```
def makeTiramisu(eggs, sugar1, wine) = {
  def beatEggs = T{ beat(eggs()) }
  def mixture = T{ beat(beatEggs(), sugar1(), wine()) }
  def whisked = T{ whisk(mixture()) }
}
```

```
def makeTiramisu(eggs, sugar1, wine) = { // “idiom bracket” transformation
  def beatEggs = T.zipMap(eggs){ v1 => beat(v1) }
  def mixture = T.zipMap(beatEggs, sugar1, wine){ (v1, v2, v3) => beat(v1, v2, v3) }
  def whisked = T.zipMap(mixture){ v1 => whisk(v1) }
}
```

Introspectability via Free Applicative

Mill turns build steps (“targets”) into a *Free Applicative* structure, using the T{...} macro

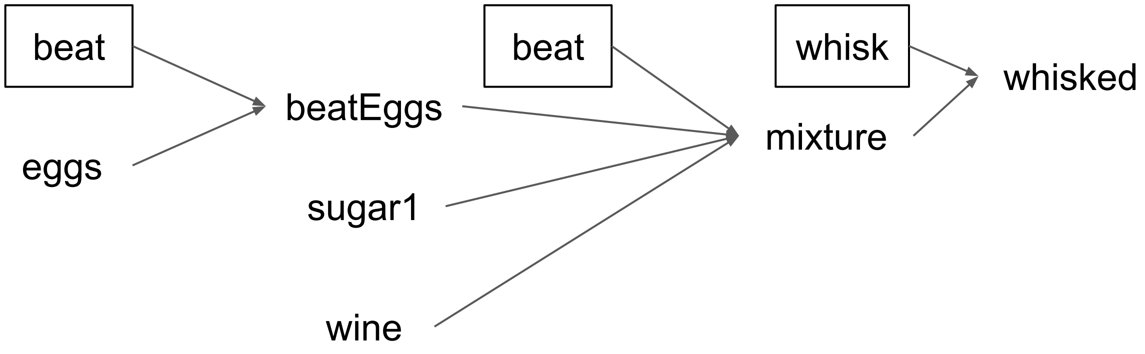
```
def makeTiramisu(eggs, sugar1, wine) = {
  def beatEggs = T.zipMap(eggs){ v1 => beat(v1) }
  def mixture = T.zipMap(beatEggs, sugar1, wine){ (v1, v2, v3) => beat(v1, v2, v3) }
  def whisked = T.zipMap(mixture){ v1 => whisk(v1) }
}
```



Introspectability via Free Applicative

Every node in the graph knows what its inputs are, and has an opaque function to compute its output value

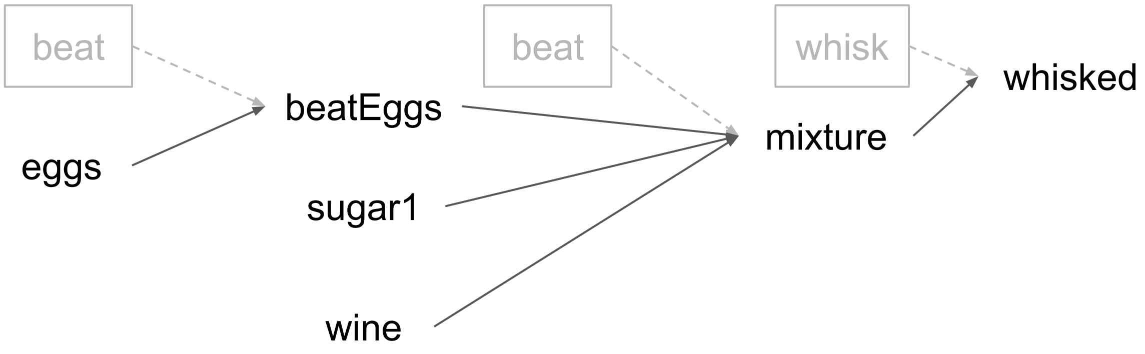
Less flexible than the *Free Monad*: the structure of the graph cannot depend on the computed value of any node



Introspectability via Free Applicative

Every node in the graph knows what its inputs are, and has an opaque function to compute its output value

Less flexible than the *Free Monad*: the structure of the graph cannot depend on the computed value of any node

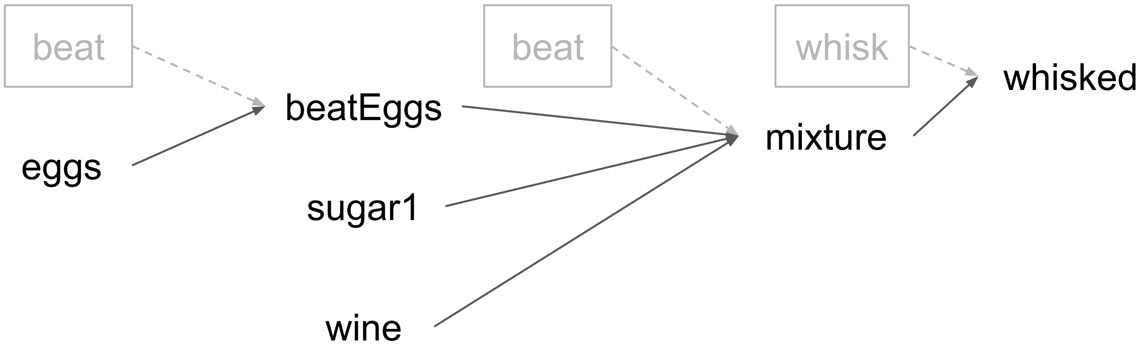


Introspectability via Free Applicative

Every node in the graph knows what its inputs are, and has an opaque function to compute its output value

Less flexible than the *Free Monad*: the structure of the graph cannot depend on the computed value of any node

Allows introspectability, parallelization, incremental computation, etc.



How the Mill build tool uses FP, with...

Introspectability

Persistence

Long-lived Workers

Ad-hoc Overrides

Persistence: How?

Every Mill target defined via **T{...}** must return a type that is serializable

```
def T[V: upickle.default.ReadWriter](t: V) = ...
```

Persistence: How?

Every Mill target defined via **T{...}** must return a type that is serializable

```
def T[V: upickle.default.ReadWriter](t: V) = ...
```

Currently serialization is done via uPickle JSON, but could easily be done via other formats (uPickle msgpack, Circe, whatever)

Persistence: Where?

Every target must take an implicit **mill.define.Ctx** containing a **ctx.dest** filesystem path

```
def T[V: upickle.default.ReadWriter](t: V)(implicit ctx: mill.define.Ctx) = ...
```

Persistence: Where?

Every target must take an implicit **mill.define.Ctx** containing a **ctx.dest** filesystem path

```
def T[V: upickle.default.ReadWriter](t: V)(implicit ctx: mill.define.Ctx) = ...
```

The **mill.define.Ctx** is provided automatically when your target lives inside a **Module**:

```
object myTiramisu extends Module{
  def eggs = T.input{...}
  def sugar1 = T.input{...}
  def wine = T.input{...}
  def beatEggs = T{ beat(eggs()) } // "out/myTiramisu/beatEggs/"
  def mixture = T{ beat(beatEggs(), sugar1(), wine()) } // "out/myTiramisu/mixture/"
  def whisked = T{ whisk(mixture()) } // "out/myTiramisu/whisked/"
}
```

ctx.dest also provides a place to put files on disk without risking collision!

How the Mill build tool uses FP, with...

Introspectability

Persistence

Long-lived Workers

Ad-hoc Overrides

Long Lived Workers

Mill supports the **T.worker** syntax to define a long lived worker:

```
object myTiramisu extends Module{
  def eggBeater = T.worker{setupMyEggBeater()}
  def eggs = T.input{...}
  def sugar1 = T.input{...}
  def wine = T.input{...}
  def beatEggs = T{ eggBeater.beat(eggs()) }
  def mixture = T{ eggBeater.beat(beatEggs(), sugar1(), wine()) }
  def whisked = T{ whisk(mixture()) }
}
```

Workers last as long as the Mill process, and can be re-used over and over. They can also take inputs, like **T{...}** targets, and are invalidated when their inputs change. Workers are kind of like objects, kind of like first-class functions!

How the Mill build tool uses FP, with...

Introspectability

Persistence

Long-lived Workers

Ad-hoc Overrides

Ad-hoc Overrides

Sets of similar build steps are constructed using **traits**

```
trait TiramisuModule extends Module{
  def eggs = T.input{...}
  def sugar1 = T.input{...}
  def wine = T.input{...}
  def beatEggs = T{ beat(eggs()) }
  def mixture = T{ beat(beatEggs(), sugar1(), wine()) }
  def whisked = T{ whisk(mixture()) }
}
object myTiramisu extends TiramisuModule{}
object yourTiramisu extends TiramisuModule{
  override def beatEggs = T{ preBeatEggsFromCarton() }
}
```

How the Mill build tool uses FP, with...

Introspectability

Persistence

Long-lived Workers

Ad-hoc Overrides

Conclusion

What is the Mill Build Tool?

What is Functional Programming All About?

How are Build Tools similar to FP?

How are Build Tools *not* similar to FP?

How the Mill Build Tool uses FP

Conclusion

Functional programming is thinking about dataflow, rather than control flow

- This makes refactoring, parallelism, analysis, etc. much easier

Build tools are also all about dataflow

- They benefit from all the same things that humans enjoy when using FP!

Build tools have additional concerns outside naive FP

- Introspectability, persistence, long-lived workers, ad-hoc overrides

Mill uses a mix of FP and OO features to get the best of both worlds

- Code as easy to read and intuitive as pure-FP programs, but with the speed, efficiency, and featureset that people expect from a modern build tool

Mill: A Build Tool based on Pure Functional Programming

Li Haoyi, Scala.Love 18 April 2020

New Book: Hands-on Scala Programming

A practical, project-based intro to Scala

Covers Mill, along with a ton of other things

If you liked what you saw, this will have more of it!

www.handsonscala.com

Coming soon, Summer 2020!