Four facets of good open source libraries

Bay Scala, 28 April 2017

haoyi.sg@gmail.com

Agenda

Four facets of good open source libraries

Not specific to any particular library or field

Hopefully useful if you want to build one in future

About me

Previously software engineer at Dropbox

Bright

technology services

Currently at Bright technologies (www.bright.sg)

- Data-science/Scala consulting
- Fluent Code Explorer (www.fluentcode.com)

Early contributor to Scala.js, author of Ammonite REPL, Scalatags, FastParse, ...

haoyi.sg@gmail.com, www.lihaoyi.com, @li_haoyi on Twitter, @lihaoyi on Github

About me: Libraries I’ve Written

https://github.com/lihaoyi/**Ammonite**

https://github.com/lihaoyi/**utest**

https://github.com/lihaoyi/**scalatags**

https://github.com/lihaoyi/**fastparse**

https://github.com/lihaoyi/**autowire**

https://github.com/lihaoyi/**upickle-pprint**

https://github.com/lihaoyi/**sourcecode**

Goals of an open-source library

Goals of an “open-source library”

Make a library you use

Make a library your friends & colleagues use

Make a library complete strangers use

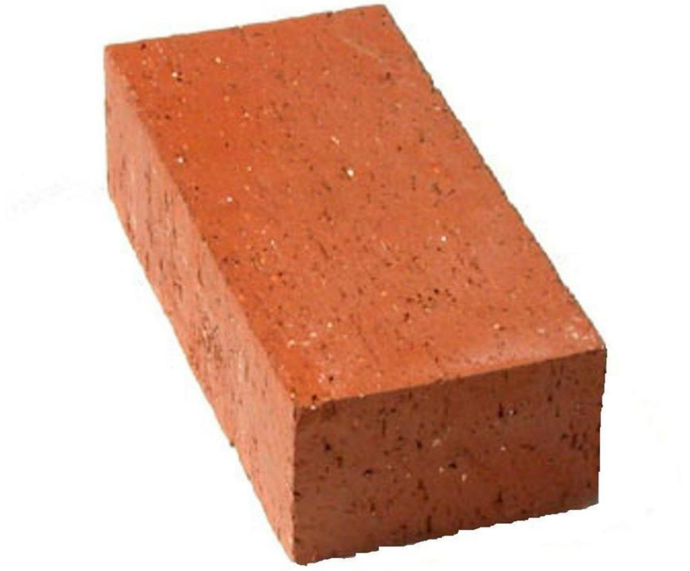
Non-goals of an “open-source library”

Answer lots of questions

Talk to lots of people

Build a community

Library vs Community



Library vs Community

What a user wants from a Library

What a user wants from a Library

Use your library without reading docs

Learn without talking to a human (i.e. you)

Have the library cater to him when he’s new

Have the library cater to him when he’s an expert

Fix a specific problem in his project you’ve never seen

Four facets of good open source libraries

Four facets of good open source libraries

Intuitiveness: use library w/o reading docs

Layering: cater to users both newbie and expert

Documentation: learn w/o talking to a human

Shape: fix a problem in a project you’ve never seen

Four facets of good open source libraries

Intuitiveness

Layering

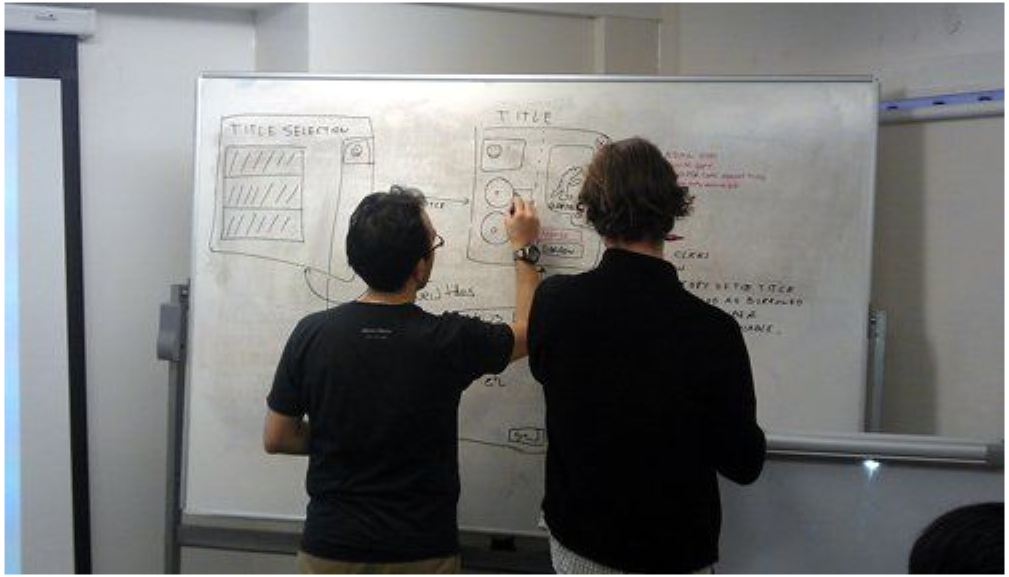
Documentation

Shape

What does it mean to be intuitive?

You can use a library without looking up docs

What does it mean to be intuitive?



What does it mean to be intuitive?

You can use a library without looking up docs

```
In [1]: import requests

In [2]: r = requests.get('https://api.github.com/events')

In [3]: r.json()
```

What does it mean to be intuitive?

You can use a library without looking up docs

```
In [1]: import requests

In [2]: r = requests.get('https://api.github.com/events')

In [3]: r.json()

[{'actor': ...,
  'created_at': '2017-04-08T09:06:34Z',
  'id': '5651890323',
  'payload': {'action': 'started'},
  'public': True,
  'repo': {'id': 87593724,
   'name': 'davydovanton/web_bouncer',
   'url': 'https://api.github.com/repos/davydovanton/web_bouncer'},
  'type': 'WatchEvent'},
```

What does it mean to be intuitive?

Matt DeBoard—

> *I'm going to get `@kennethreitz <https://twitter.com/kennethreitz>`_'s Python requests module tattooed on my body, somehow. The whole thing.*

Intuition is Consistency

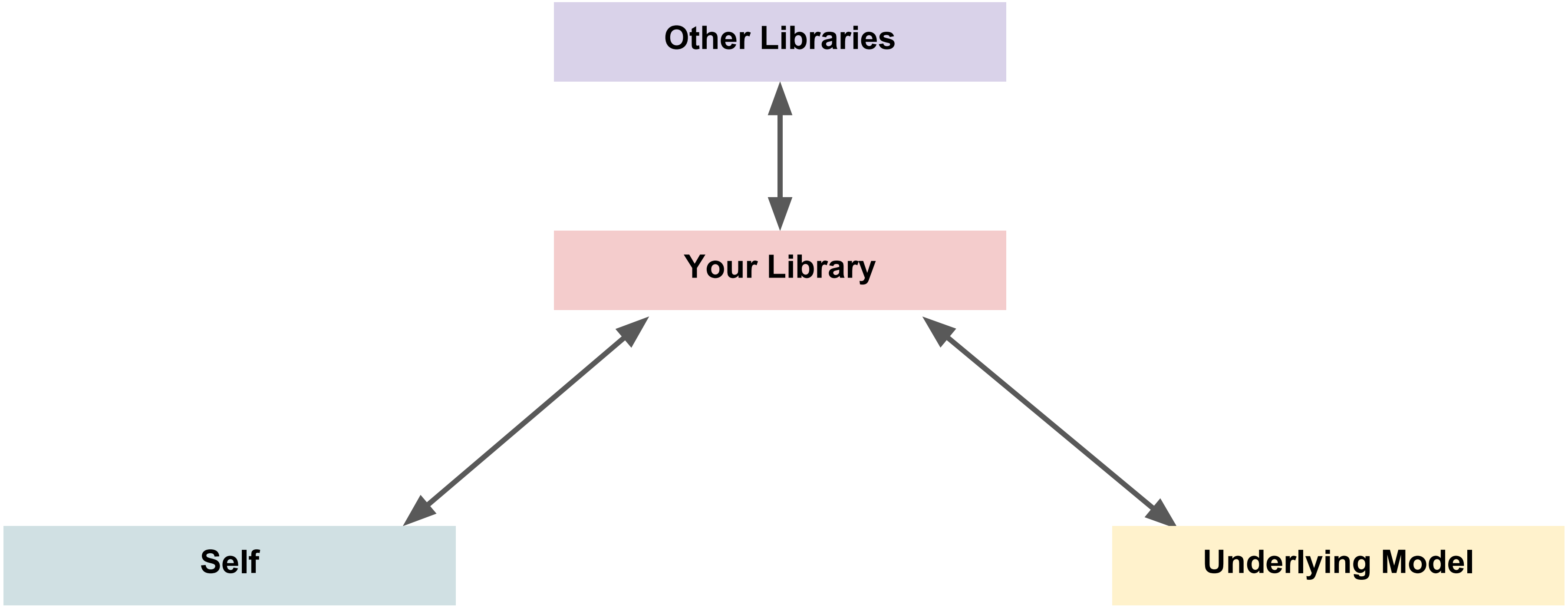
```
r = json.loads('{"hello": "world"}')
```

```
r = requests.get('https://api.github.com/events')
```

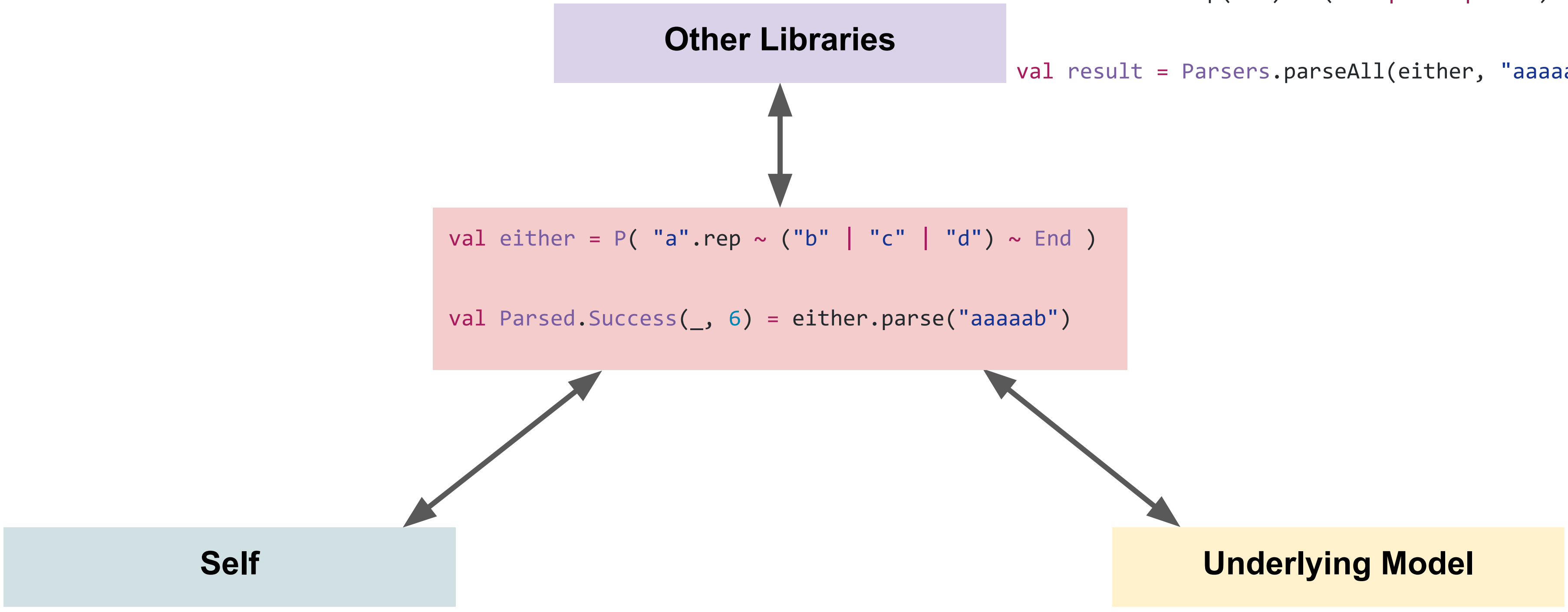
```
r = requests.post('https://api.github.com/events')
```

```
GET /events
HTTP/1.1
Host: api.github.com
```

Intuition is Consistency



FastParse Consistency



```
val either = rep("a") ~ ("b" | "c" | "d" )

val result = Parsers.parseAll(either, "aaaaab")
```

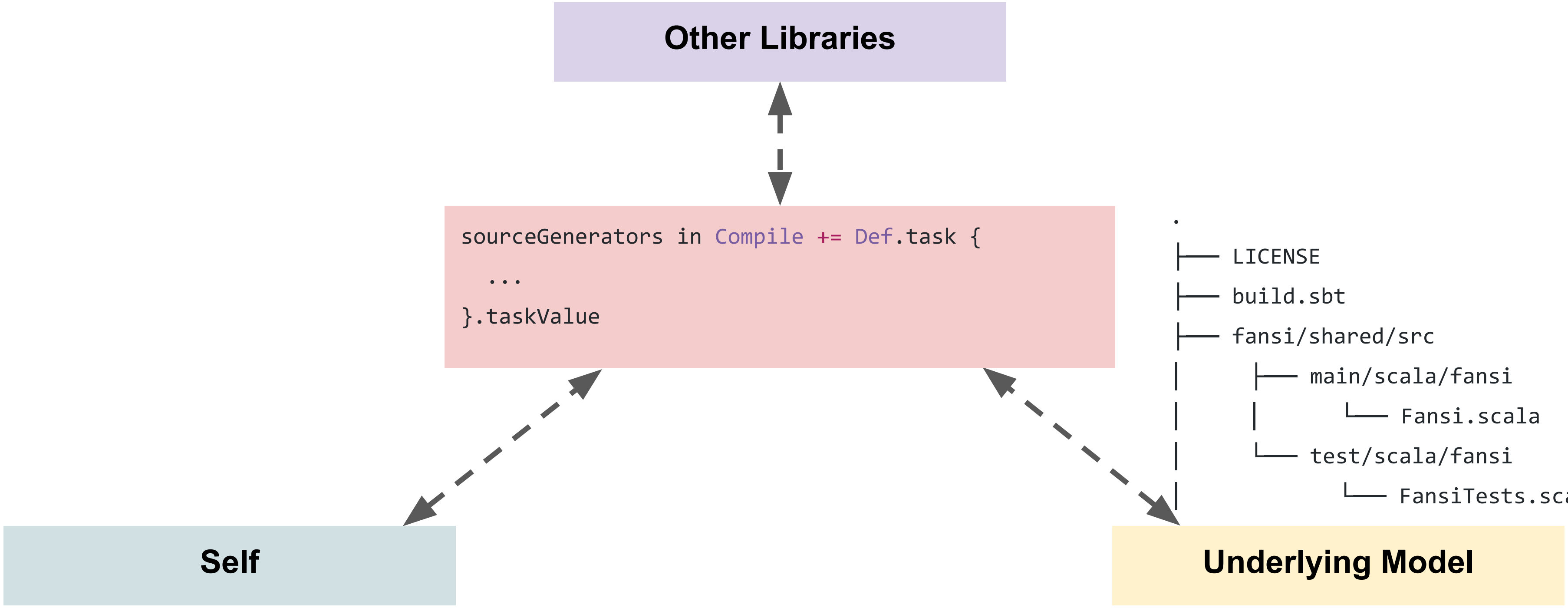
```
val either = P( "a".rep ~ ("b" | "c" | "d") ~ End )

val Parsed.Success(_, 6) = either.parse("aaaaab")
```

```
val option = P( "c".? ~ "a".rep(sep="b").! ~ End )
```

```
either = "a"* ("b" | "c" | "d") ;
```

SBT ***<u>In</u>***-consistency

```
val file = new File(canonicalFilename)
val bw = new BufferedWriter(new FileWriter(file))
bw.write(text)
bw.close()
```



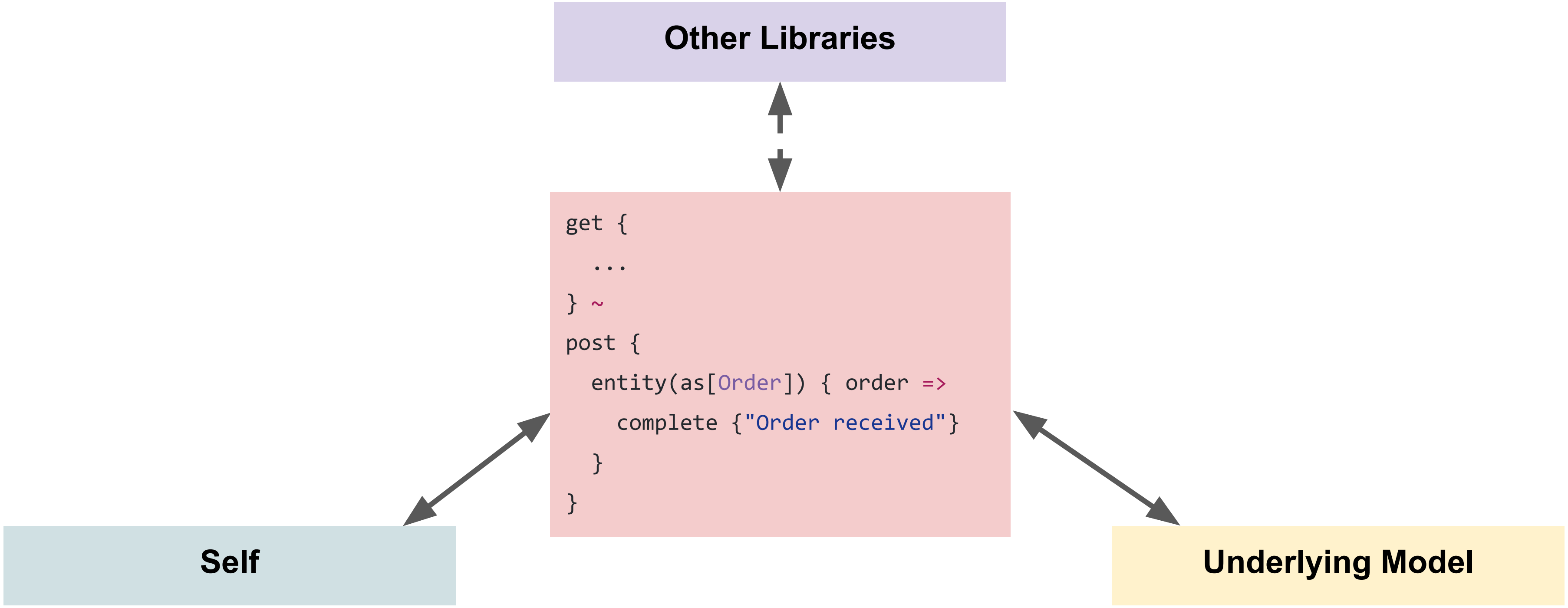
```
sourceGenerators in Compile += Def.task {
  ...
}.taskValue
```

```
.
├── LICENSE
├── build.sbt
├── fansi/shared/src
|       ├── main/scala/fansi
|       |       └── Fansi.scala
|       └── test/scala/fansi
|               └── FansiTests.scala
```

```
name := "Hello",
libraryDependencies += derby
```

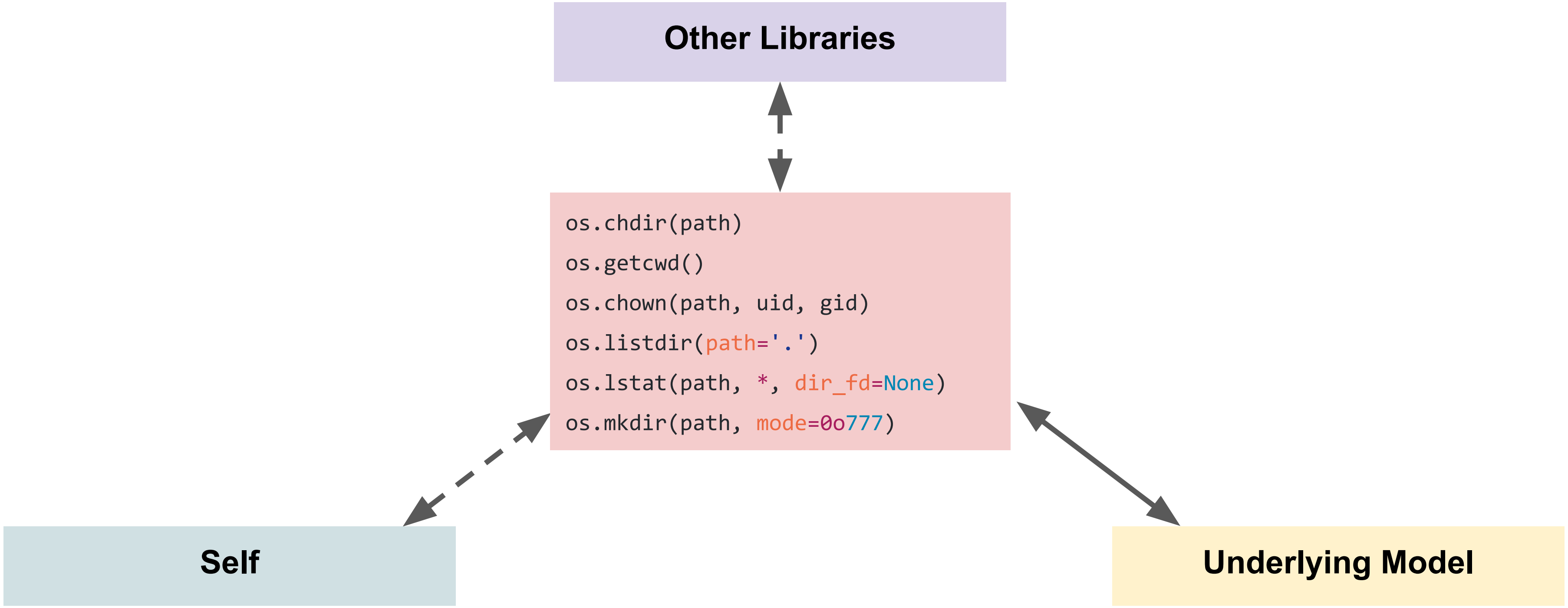
Partial Consistency

```
GET    /about      redirect(to = "https://test.com/")
GET    /orders     notFound
GET    /clients    error
GET    /posts      todo
```



```
get {
  ...
} ~
post {
  entity(as[Order]) { order =>
    complete {"Order received"}
  }
}
```

Partial Consistency



```
os.chdir(path)
os.getcwd()
os.chown(path, uid, gid)
os.listdir(path='.')
os.lstat(path, *, dir_fd=None)
os.mkdir(path, mode=0o777)
```

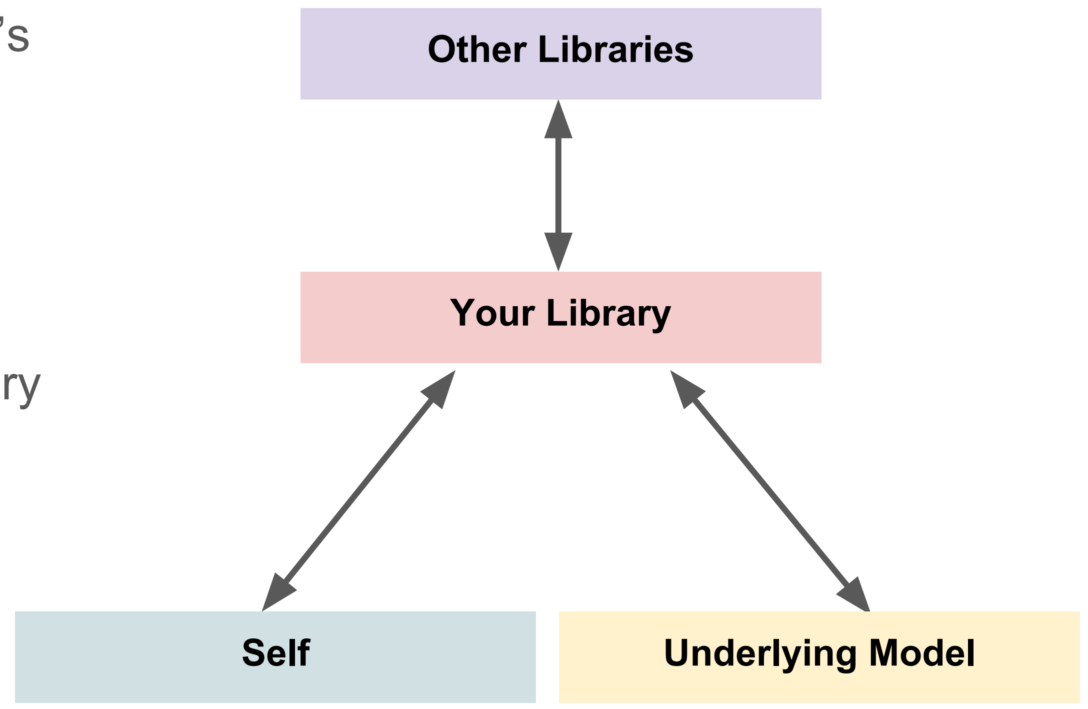
```
int chown(const char *pathname, uid_t owner, ...);
int lstat(const char *restrict path, ...);
```

Intuition is Consistency

Consistency is relative to your user's existing experiences

User's expectations come from ***multiple sources*** often contradictory

Make trade-offs consciously



Four facets of good open source libraries

Intuitiveness

Layering

Documentation

Shape

Layering your Library

Layering your Library

Do you provide a simple API for people to get started with?

Do you provide a powerful, complex API for power users to make use of?

Why not both?

Layered APIs

Newbie API

- Simple to get started with, discoverability is paramount
- Requires no configuration

Intermediate API

- Doesn’t need to be quite as simple, user already knows basics
- Probably need *some* configuration for their project

Expert API

- Configurability and “power” matters the most here
- Discoverability no longer matters so much

Layered APIs

```
# Beginner API
In [1]: import requests
In [2]: r = requests.get('https://api.github.com/events')
```

```
# Intermediate API
In [3]: r = requests.post("http://httpbin.org/get",
  headers={'user-agent': 'my-app/0.0.1'},
  data={'key1': 'value1', 'key2': 'value2'}
)
```

```
# Advanced API
In [4]: s = requests.Session()
In [5]: s.auth = ('user', 'pass')
In [6]: s.headers.update({'x-test': 'true'})
In [7]: r = s.get('http://httpbin.org/headers', headers={'x-test2': 'true'})
```

```
# Streaming API
In [8]: r = requests.get('http://httpbin.org/stream/20', stream=True)
```

Insufficiently Layered APIs

```
# Request-Level API
import akka.actor.ActorSystem
import akka.http.scaladsl.Http
import akka.http.scaladsl.model._
import akka.stream.ActorMaterializer

import scala.concurrent.Future

implicit val system = ActorSystem()
implicit val materializer = ActorMaterializer()

val responseFuture: Future[HttpResponse] =
  Http().singleRequest(HttpRequest(uri = "http://akka.io"))
```

Messy Imports; *part of your public API*

Mysterious incantations a newbie doesn't care about

What a newbie actually wants

```
# Host-Level API
...
```

Layered APIs

```
# Beginner API
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

```
# Intermediate API
...
```

Insufficiently Layered APIs

```
import akka.actor.ActorSystem
import akka.http.scaladsl.Http
import akka.http.scaladsl.model._
import akka.http.scaladsl.server.Directives._
import akka.stream.ActorMaterializer
import scala.io.StdIn
```

Messy Imports; *part of your public API*

```
object WebServer {
  def main(args: Array[String]) {

    implicit val system = ActorSystem("my-system")
    implicit val materializer = ActorMaterializer()
    // needed for the future flatMap/onComplete in the end
    implicit val executionContext = system.dispatcher
```

Mysterious incantations a newbie doesn't care about

Insufficiently Layered APIs

```
    val route =
      path("hello") {
        get {
          complete(HttpEntity(ContentTypes.`text/html(UTF-8)`,
            "<h1>Say hello to akka-http</h1>")
          )
        }
      }
    val bindingFuture = Http().bindAndHandle(route, "localhost", 8080)
    println(s"Server online at http://localhost:8080/\nPress RETURN to stop...")
    StdIn.readLine() // let it run until user presses return
    bindingFuture.flatMap(_.unbind()).onComplete(_ => system.terminate())
  }
}
```

```
# Intermediate API
```

Layered APIs

```
# Beginner API
import akka.http.scaladsl.model.{ContentTypes, HttpEntity}
import akka.http.scaladsl.server.Directives._
import akka.http.scaladsl.server.{HttpApp, Route}
import akka.http.scaladsl.settings.ServerSettings
import com.typesafe.config.ConfigFactory
object WebServer extends HttpApp {
  def route: Route = path("hello") {
    get {
      complete(HttpEntity(ContentTypes.`text/html(UTF-8)`, "<h1>Say hello to akka-http</h1>"))
    }
  }
}
WebServer.startServer("localhost", 8080, ServerSettings(ConfigFactory.load))
```

```
# Intermediate API
```

Layering

Simple code for newbies

Advanced features for experts

Newbie API

- Simple to get started with, discoverability is paramount
- Requires no configuration

Intermediate API

- Doesn’t need to be quite as simple, user already knows basics
- Probably need *some* configuration for their project

Expert API

- Configurability and “power” matters the most here
- Discoverability no longer matters so much

Four facets of good open source libraries

Intuitiveness

Layering

Documentation

Shape

Documentation is a Feature

Documentation is a Feature

Mediocre library w/ good docs vs. Amazing library w/ poor docs

- Looks the same from the outside

Most of your users do not *want* to talk to you

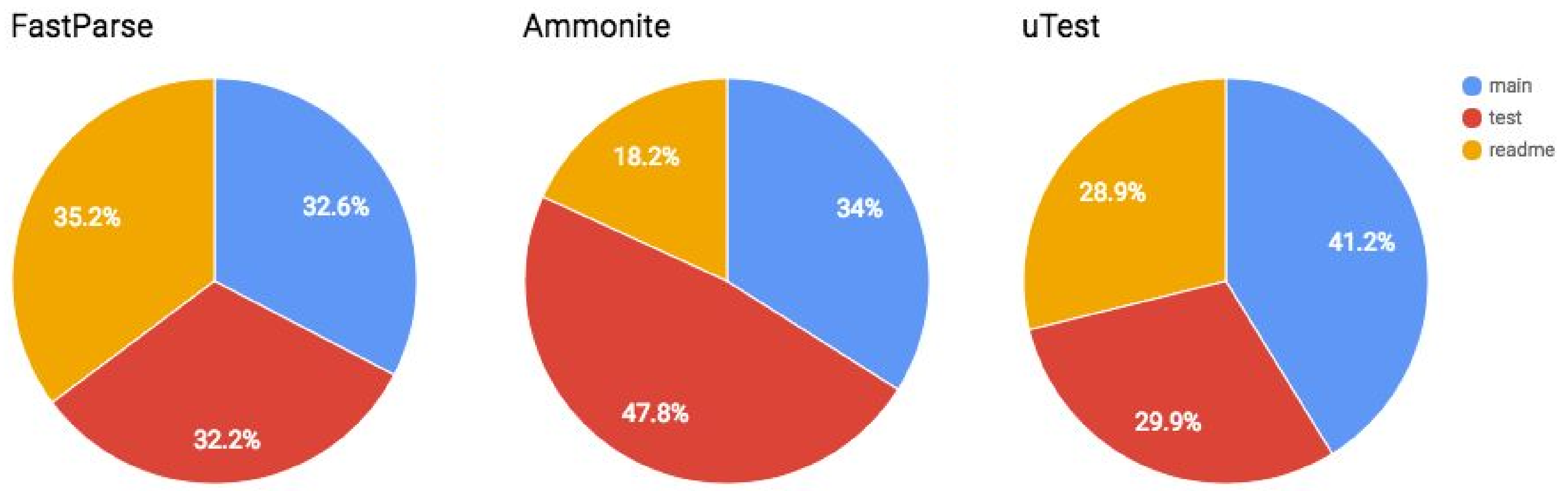
- You probably do not want to talk to most of your users either

Proportional Documentation

Proportional Documentation: FastParse

```
find fastparse -name "*.scala" | grep main | xargs wc -l
    1987 total

find fastparse -name "*.scala" | grep test | xargs wc -l
    1957 total

find . -name "*.scalatex" | xargs wc -l
    2143 total
```

Proportional Documentation



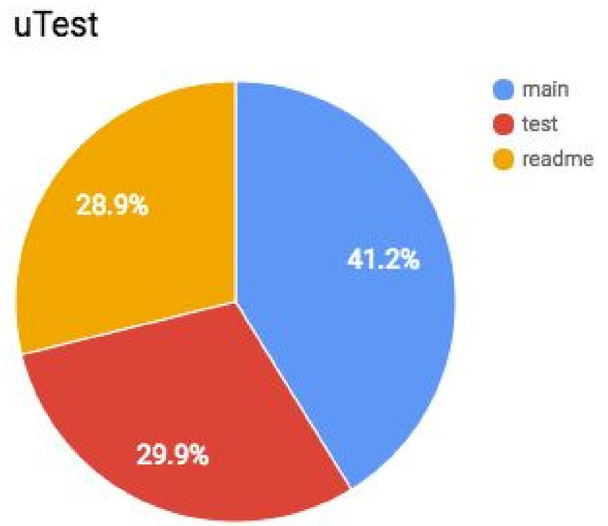
Proportional Documentation

Main code is the stuff that runs

Test code makes sure Main code does what it should

Docs make sure people can learn how to use it

All are important to the goal of “Make a library complete strangers use”



Layered Documentation

Intro Topics

- What is this library?
- Why should I care?

Newbie Topics

- I want to use this library. How?

Intermediate Topics

- I have been using this library for a while.
- What are the problems I will face?

Advanced Topics

- I am an expert in the library.
- How does its internals work?
- Why was it built in this way?

Intro Topics

FastParse is a parser-combinator library for Scala that lets you quickly and easily write recursive descent text- and binary data parsers in Scala

Newbie Topics

The simplest parser matches a single string:

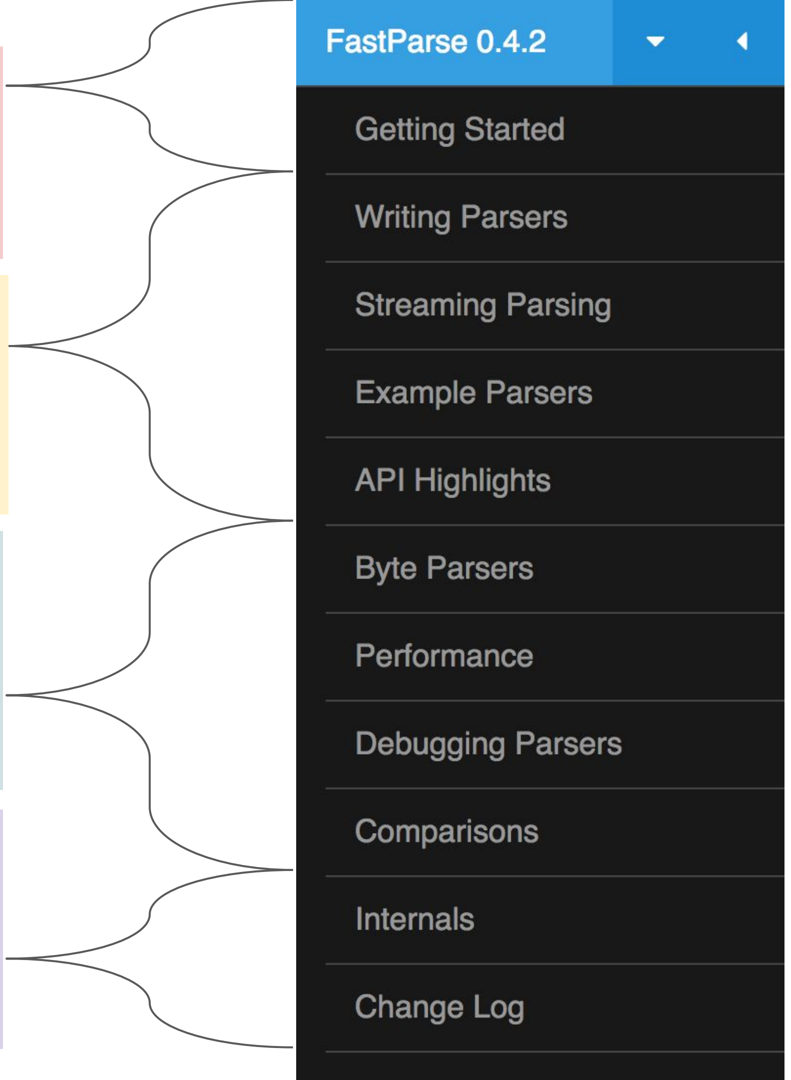
```scala
val parseA = P( "a" )
val Parsed.Success(value, successIndex) = parseA.parse("a")
```

Intermediate Topics

While for super-high-performance use cases you may still want a hand-rolled parser, for many ad-hoc situations a FastParse parser would do just fine.

Advanced Topics

FastParse is designed as a fast, immutable interpreter. That means It does not do significant transformations of the grammar. The structure of the parser you define is the structure that will run.



Intro Topics

ScalaTags is a small, fast XML/HTML/CSS construction library for Scala that takes fragments in plain Scala code that look like...

Newbie Topics

This is a bunch of simple examples to get you started using Scalatags.

```
body(h1("This is my title"), ...)
```

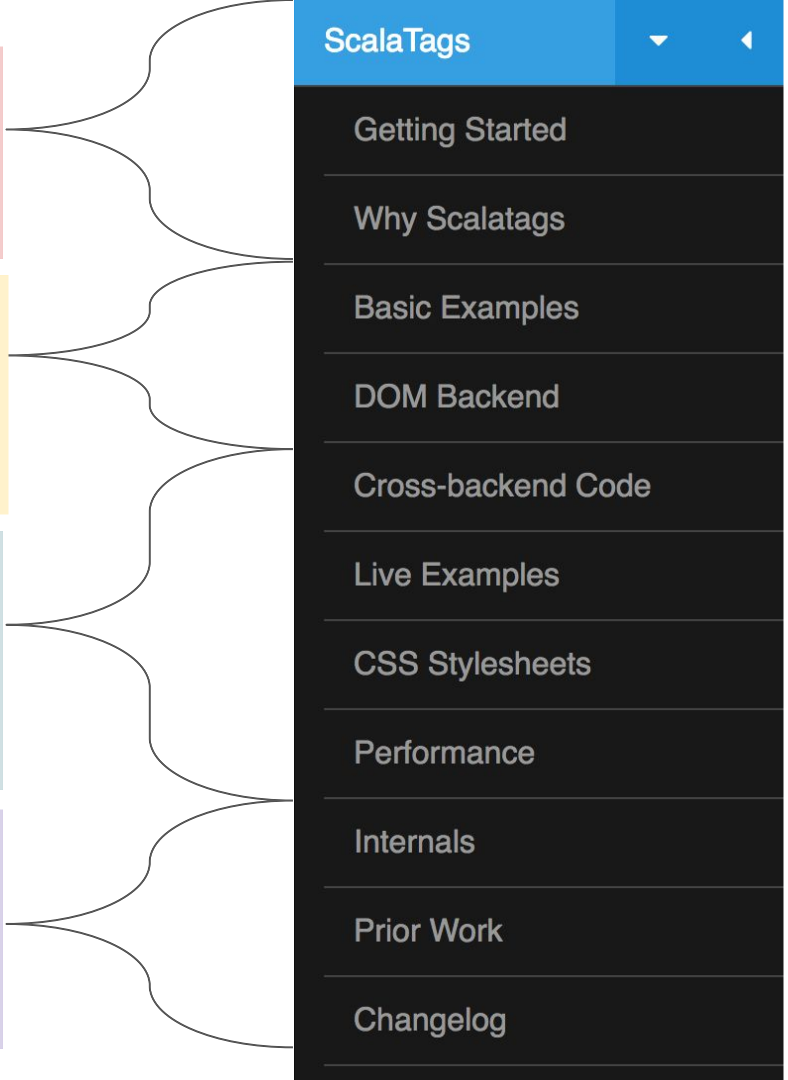
Intermediate Topics

If you wish to, it is possible to write code that is generic against the Scalatags backend used, and can be compiled and run on both Text and JsDom backends at the same time! This is done by...

Advanced Topics

Scalatags has pretty odd internals in order to support code re-use. Essentially, each Scalatags package is an instance of

```
trait Bundle[Builder, Output <: FragT, FragT]{...}
```



Incorrectly Layered Docs

Bad Newbie Topics (Old SBT Getting Started)

After examining a project and processing any build definition files, sbt will end up with an immutable map (set of key-value pairs) describing the build.

Build definition files do not affect sbt's map directly.

Instead, the build definition creates a huge list of objects with type Setting[T] where T is the type of the value in the map. (Scala's Setting[T] is like Setting<T> in Java.) A Setting describes a transformation to the map, such as adding a new key-value pair or appending to an existing value. (In the spirit of functional programming, a transformation returns a new map, it does not update the old map in-place.)

In build.sbt, you might create a Setting[String] for the name of your project like this:

```
name := "hello"
```

This Setting[String] transforms the map by adding (or replacing) the name key, giving it the value "hello". The transformed map becomes sbt's new map.

Good Newbie Topics (New SBT Getting Started)

A *build definition* is defined in `build.sbt`, and it consists of a set of projects (of type `Project`). Because the term *project* can be ambiguous, we often call it a *subproject* in this guide.

For instance, in `build.sbt` you define the subproject located in the current directory like this:

```
lazy val root = (project in file("."))
  .settings(
    name := "Hello",
    scalaVersion := "2.12.1"
  )
```

Each subproject is configured by key-value pairs.

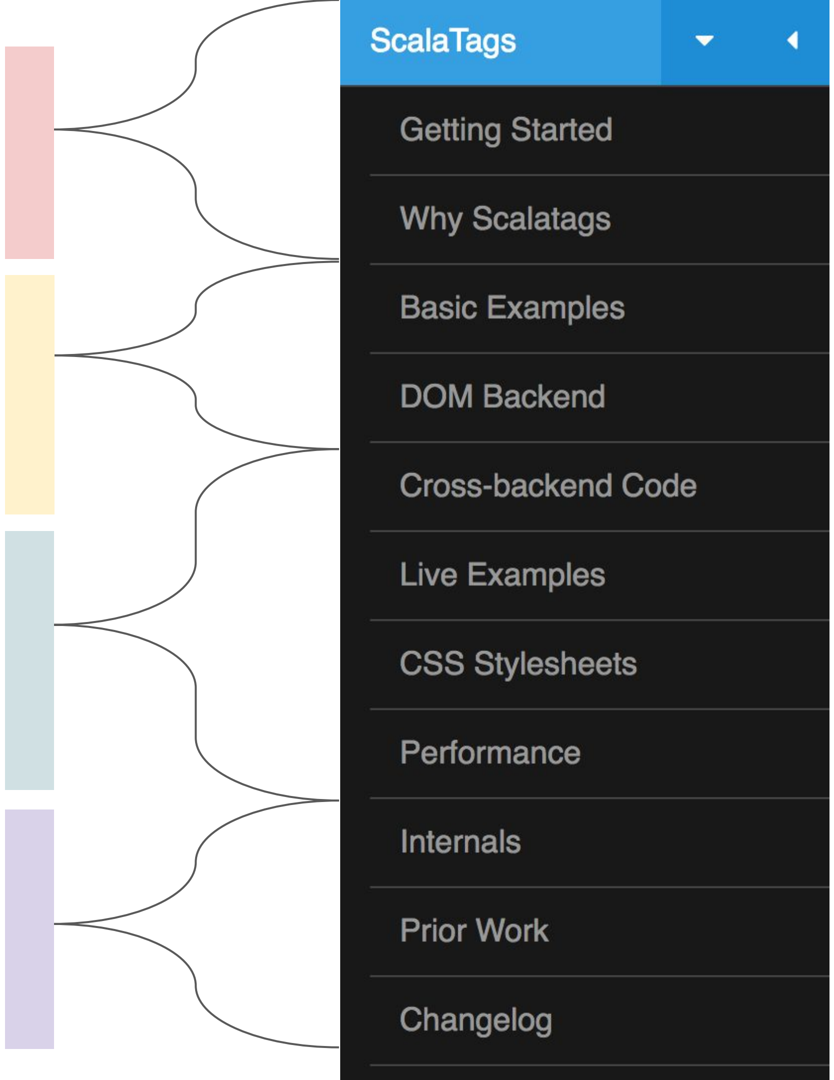
For example, one key is name and it maps to a string value, the name of your subproject.

Layered Documentation

Quantity of documentation is not all that matters

Structure is equally important

Documentation in wrong place, e.g. internal implementation details in newbie area, is *actively harmful*



Four facets of good open source libraries

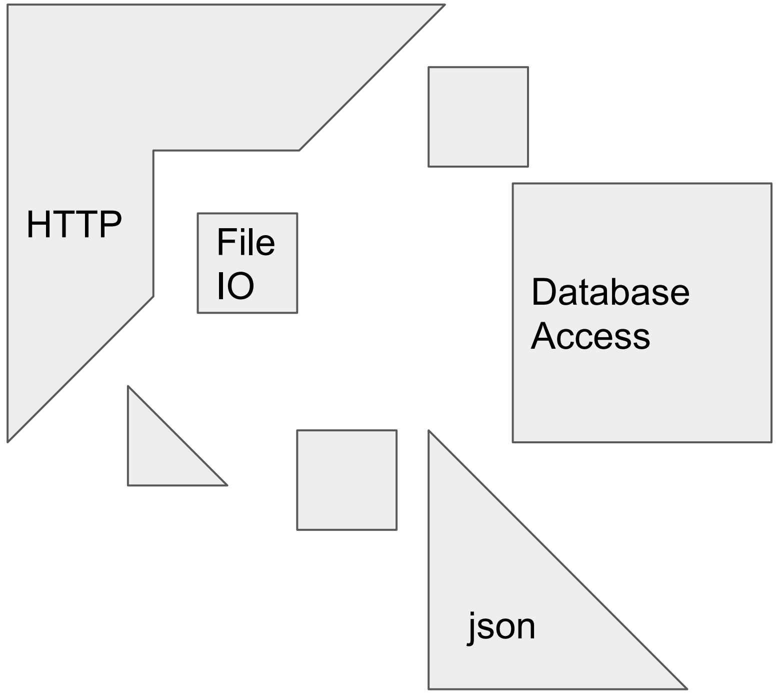
Intuitiveness

Layering

Documentation

Shape

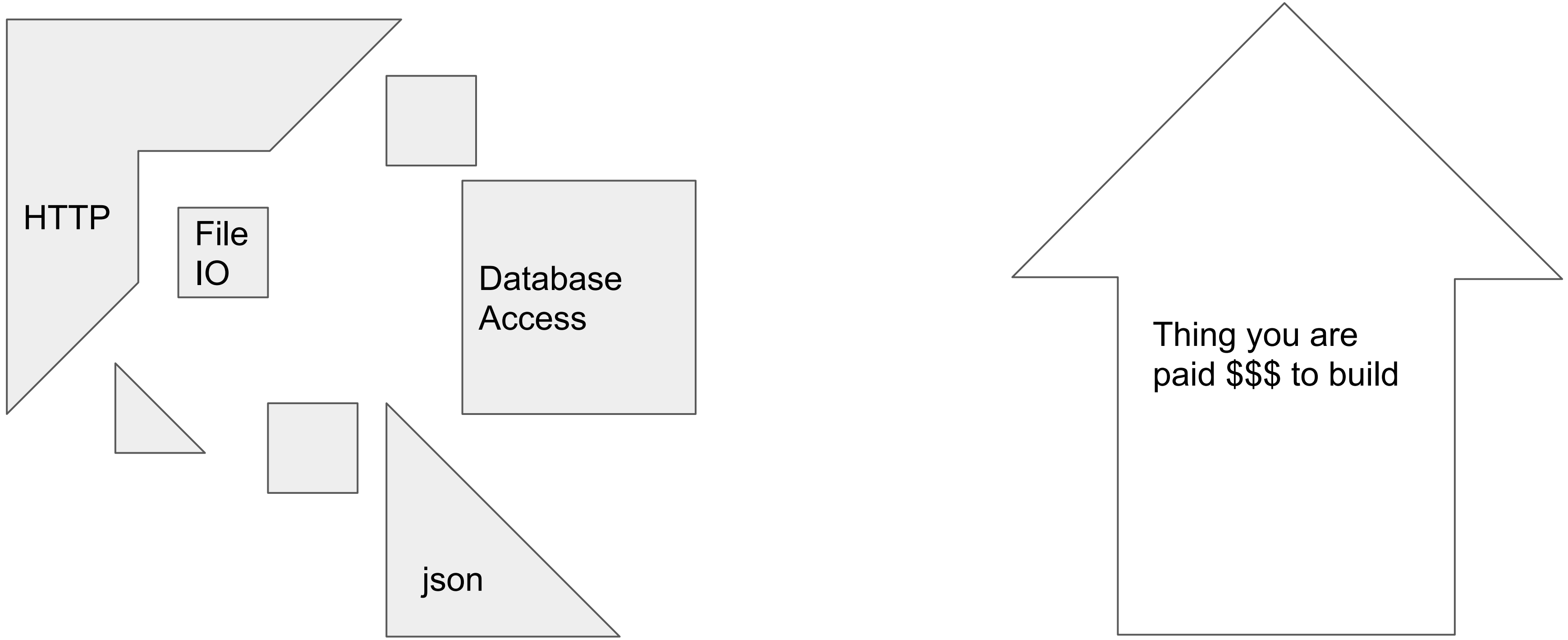
Library Shape



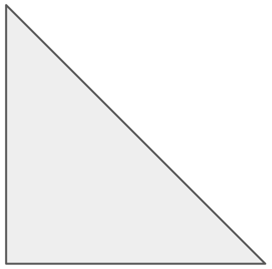
A library's API and functionality can be thought of as a "shape"

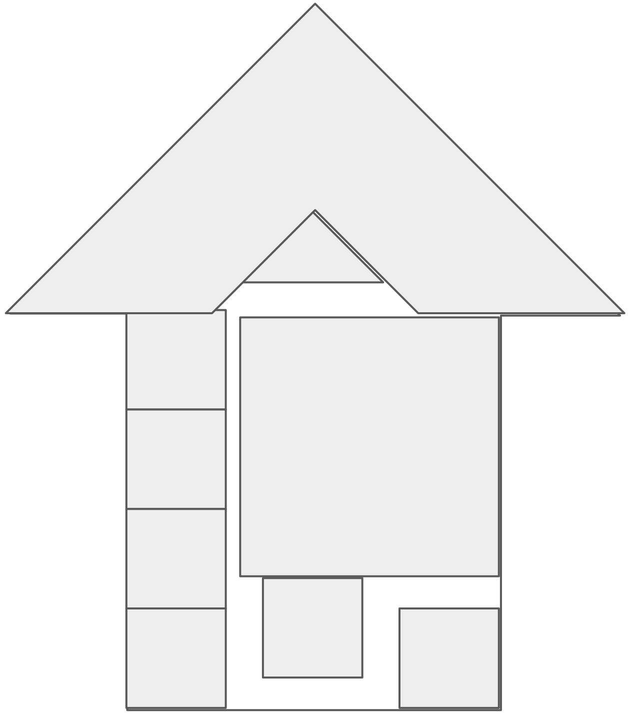
Each library covers a different portion of the space of possible problems

Library Shape

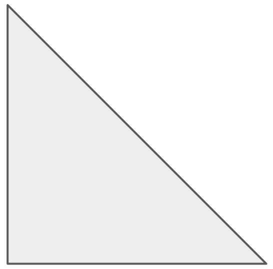


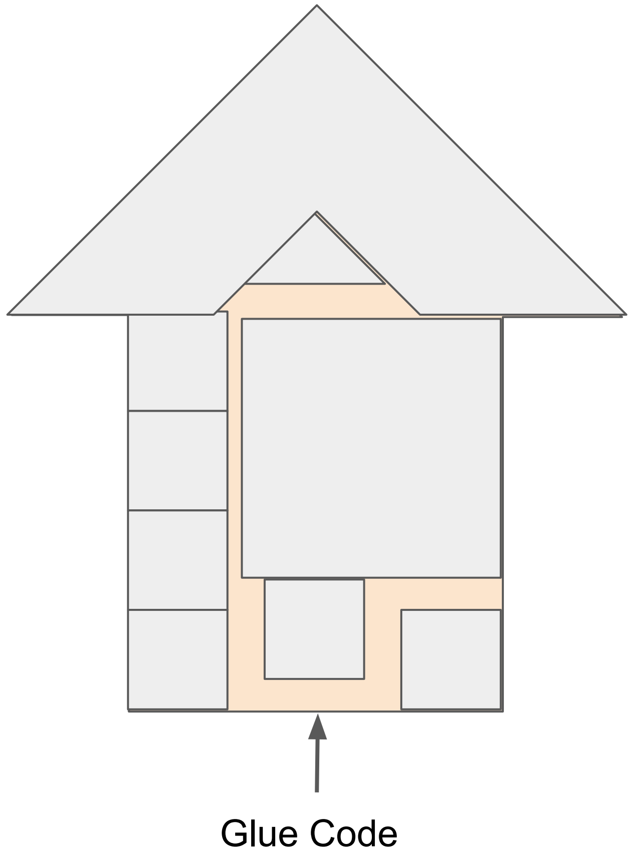
Library Shape



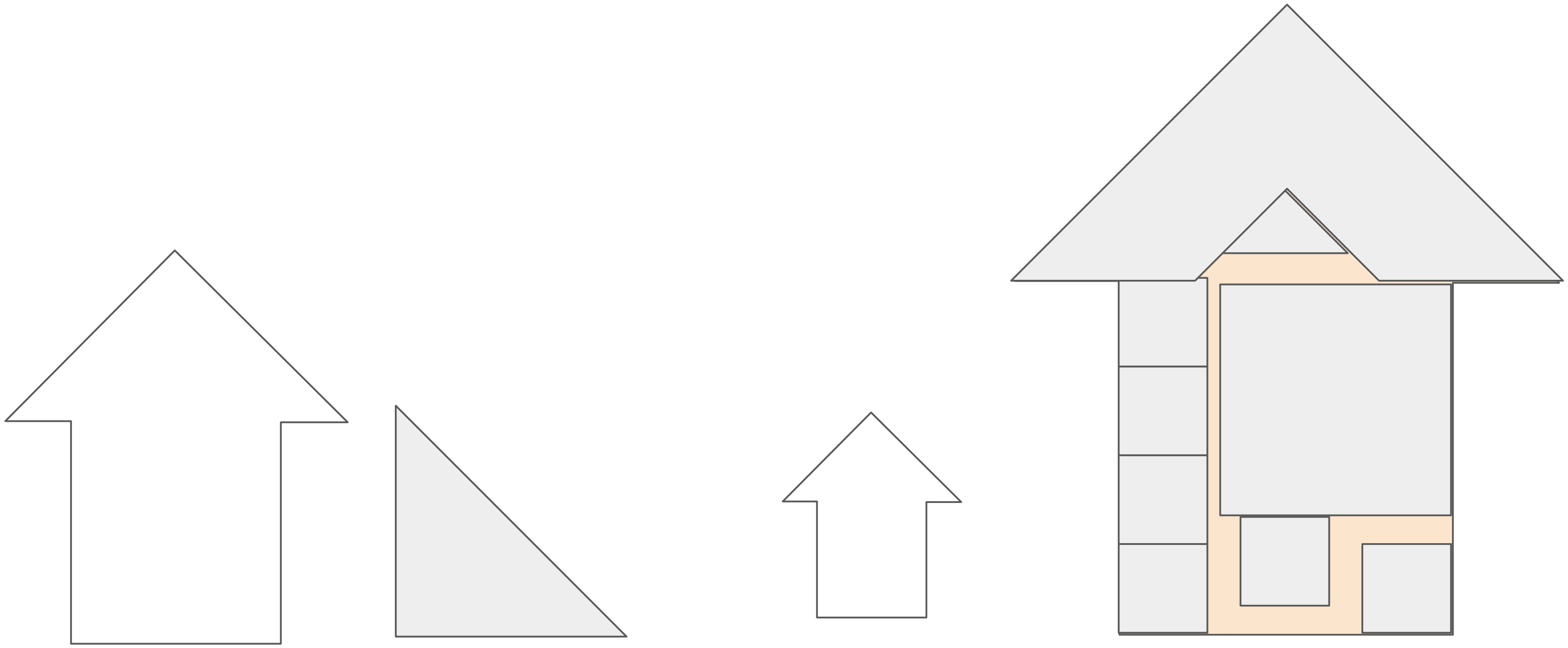


Library Shape

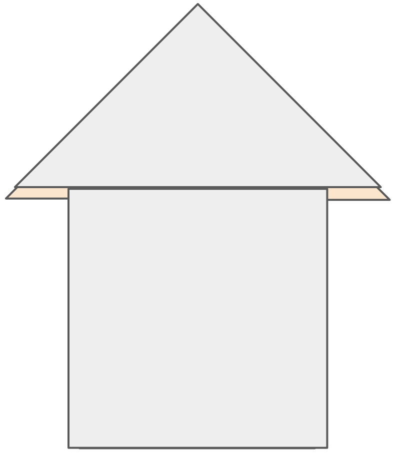


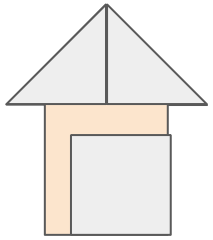


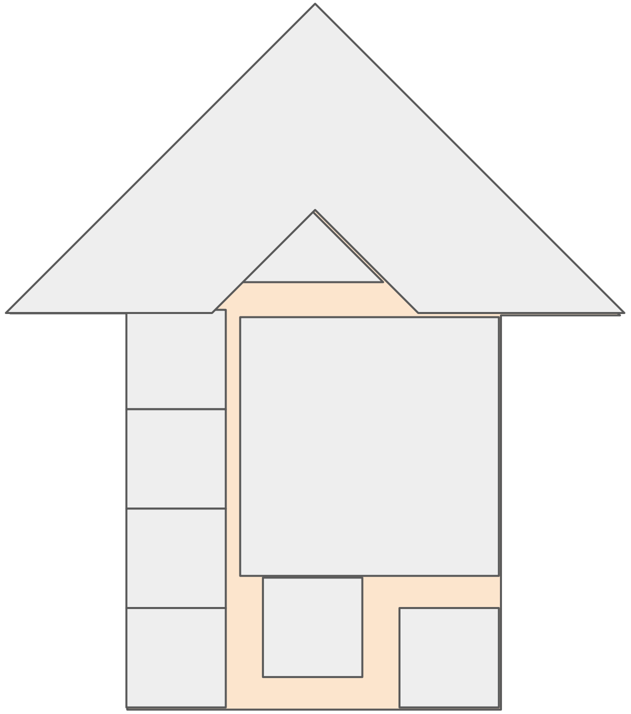
Library Shape



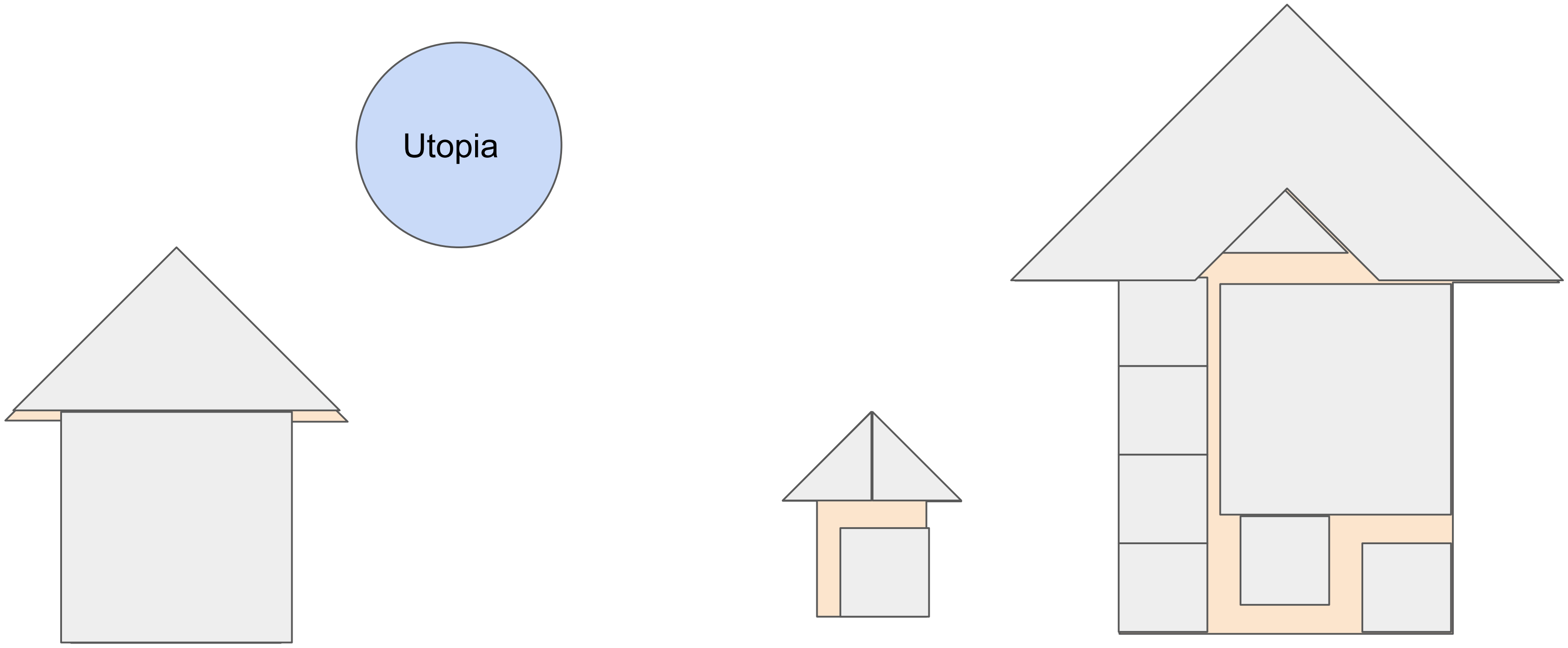
Library Shape



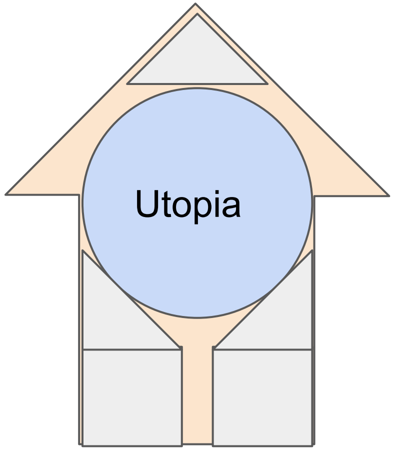


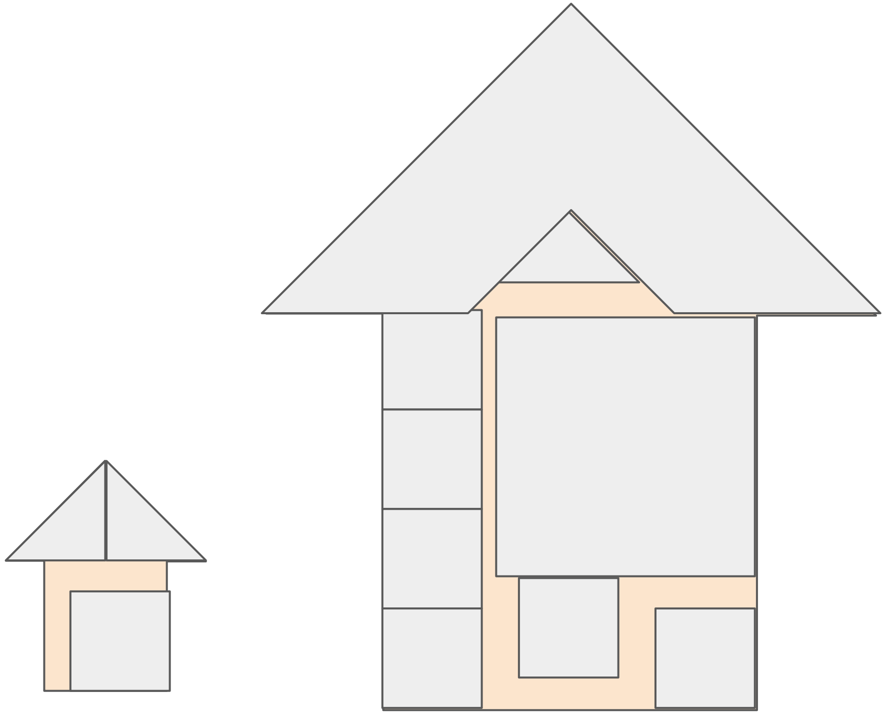


Failure Mode: Utopia Library

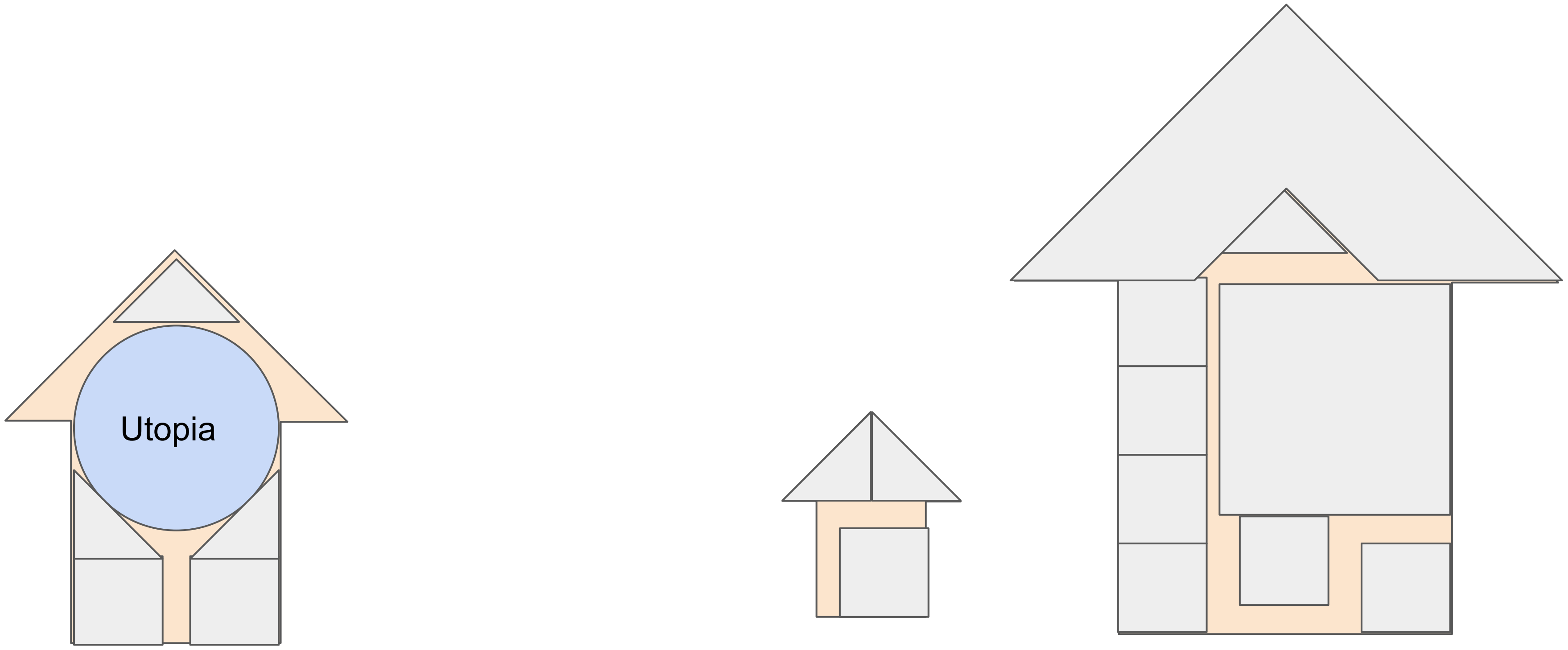


Failure Mode: Utopia Library

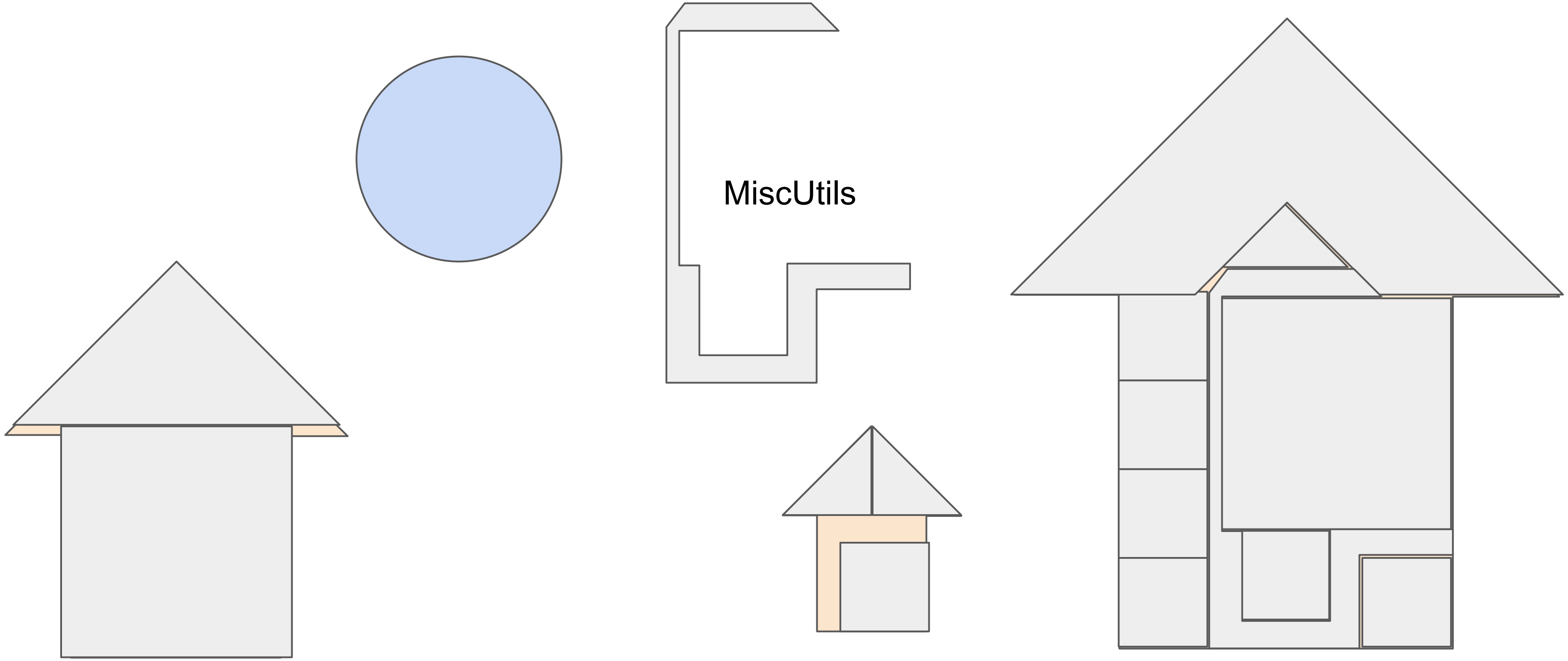




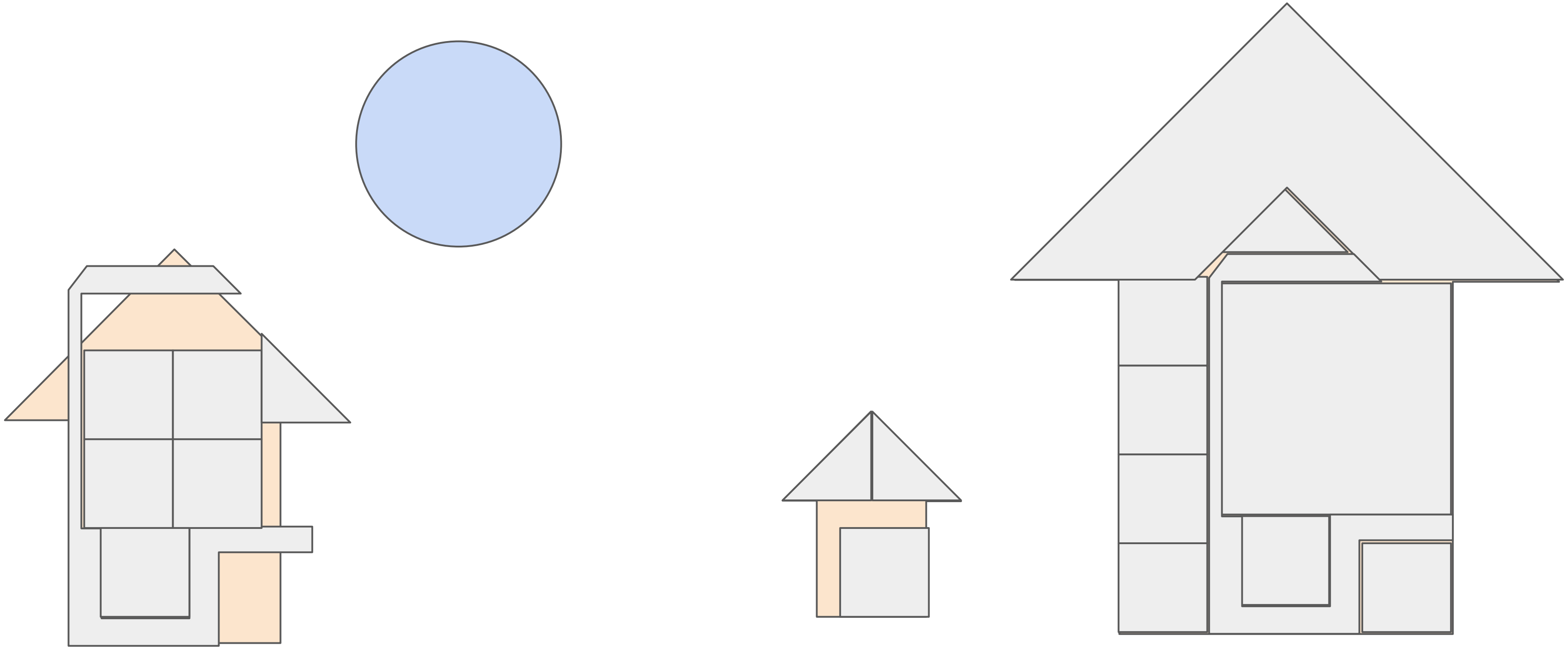
Failure Mode: Utopia Library



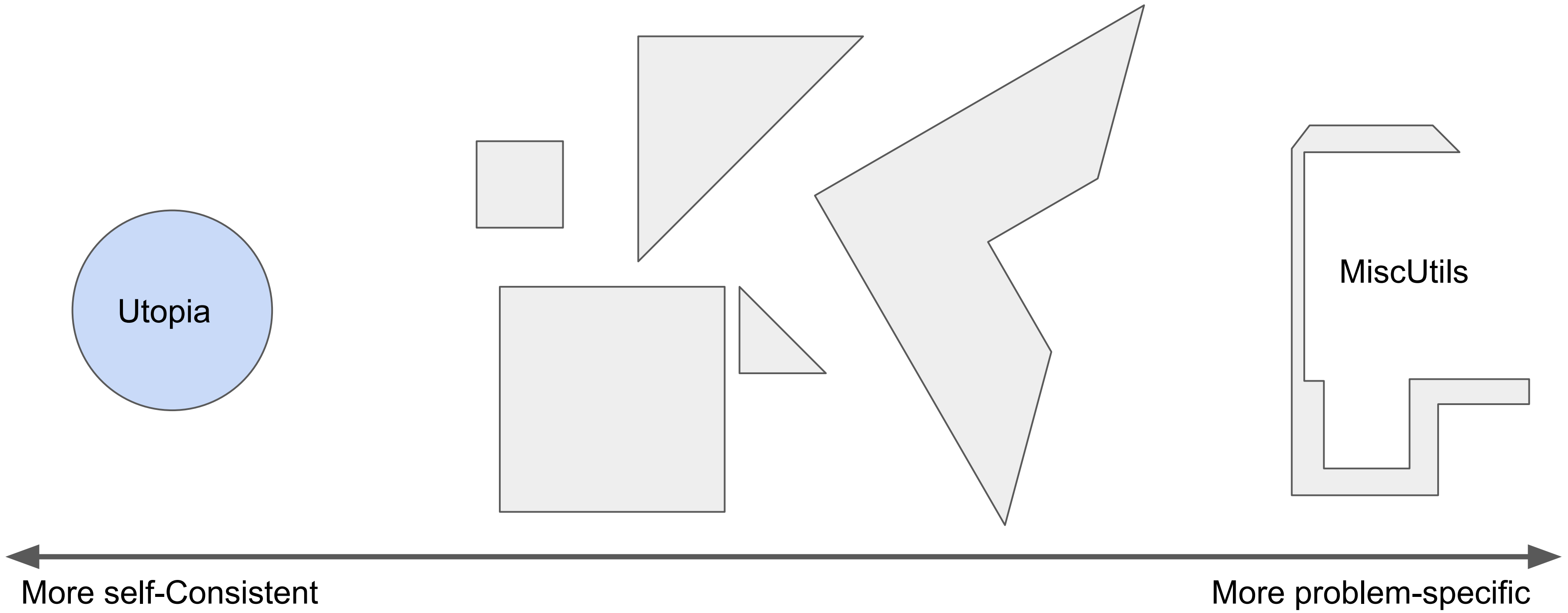
Failure Mode: Glue Library



Failure Mode: Glue Library

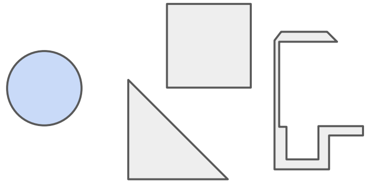


Library Shape

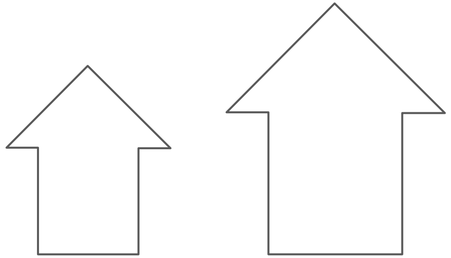


Library Shape

Balance being generic/elegant with being problem-specific



Think about how your library fits into a larger project



Conclusion

What a user wants from a Library

Use your library without reading docs

Learn without talking to a human (i.e. you)

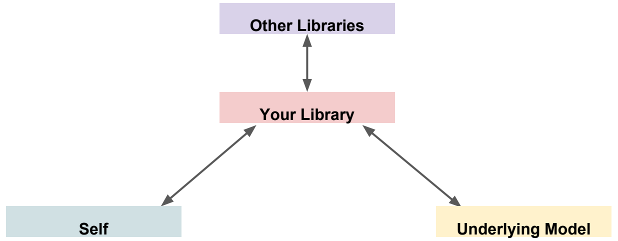
Have the library cater to him when he’s new

Have the library cater to him when he’s an expert

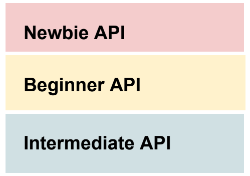
Fix a specific problem in his project you’ve never seen

Four facets of good open source libraries

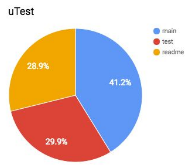
Intuitiveness

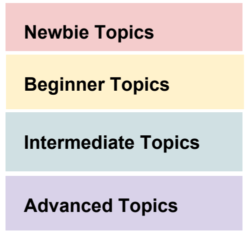


Layering

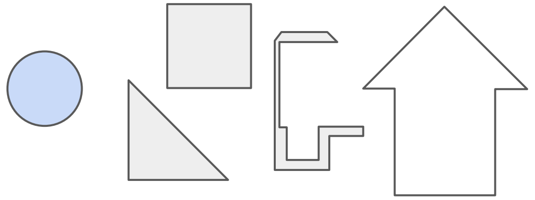


Documentation





Shape



Four facets of good open source libraries

Bay Scala, 28 April 2017

haoyi.sg@gmail.com