Scala eXchange *9 December 2016*

# Visions for collaboration, competition, and interop in Scala

Erik Osheim (@d6)

stripe

# who am i?

- **typelevel** member λ
- maintain **spire**, **cats**, and other scala libraries
- interested in expressiveness **and** performance ◕
- support machine learning at **stripe**

code at *http://github.com/non*

# original motivations

When proposing this talk I was thinking about:

- transitive dependency pain
- versioning pain
- build system pain
- binary compatibility pain
- standardization pain

Sensing a theme?

# original motivations

Hume's advice for dealing with *philosophical melancholy*:

```
> I dine, I play a game of backgammon,
> I converse, and am merry with my friends.
```

This also works for many problems in FL/OSS!

# bait-and-switch

Instead, I've decided to refocus around this theme:

```
> This talk will explore the challenges of
> designing and depending on Scala libraries,
> and also the trade-offs inherent in
> standardization.
```

(Ultimately this talk is biased toward future library authors.)

# what is this talk about?

Intended to be a high-level guide on library development/design:

- "real world" library development
- based on my experiences
- mildly-opinionated
- confessional style

TL;DR There isn't a lot of code on these slides.

# our agenda

1. Typelevel overview and pitch
2. Why write a library?
3. Competition in the library design space
4. Collaboration in library development
5. Some good problems to have
6. Conclusions

# typelevel overview

Some top-line numbers:

- 6 summits/events
- 6 hack days
- 46 projects
- 15 incubator projects
- hundreds of contributors

Thanks to everyone whose work makes this possible! 👏

# cats

- provides abstractions for functional programming in Scala.
- (e.g. `Traverse`, `Monad`, `Free`, and so on.)
- almost two years old
- current version: 0.8.1
- 120+ contributors
- active and helpful Gitter channel (1148 people)

http://github.com/typelevel/cats

# cats update

- Worked through some changes removals (e.g. `Xor`). ✺
- A few outstanding design issues (e.g. mtl, default impls) ☁
- Hopefully not too major. 🤞
- Still no 1.0 release (sorry!) ✂
- But currently quite stable (we use it at **stripe**!) 🔔

Take-away: Cats is already quite useful to many people!

# teleology

noun, *philosophy* (from Greek *telos*, "end," and *logos*, "reason")

1. the doctrine that final causes exist.

2. the study of the evidences of design or purpose in nature.

3. such design or purpose.

# typelevel teleology

Why does Typelevel exist?

```
for {
  c <- typelevelCompiler
  t <- pluginsAndTools(c)
  l <- foundationalLibraries(t)
  s <- specializedDataTypes(t, l)
  d <- domainLibraries(t, l, s)
} yield amazingSoftware(t, l, s, d)
```

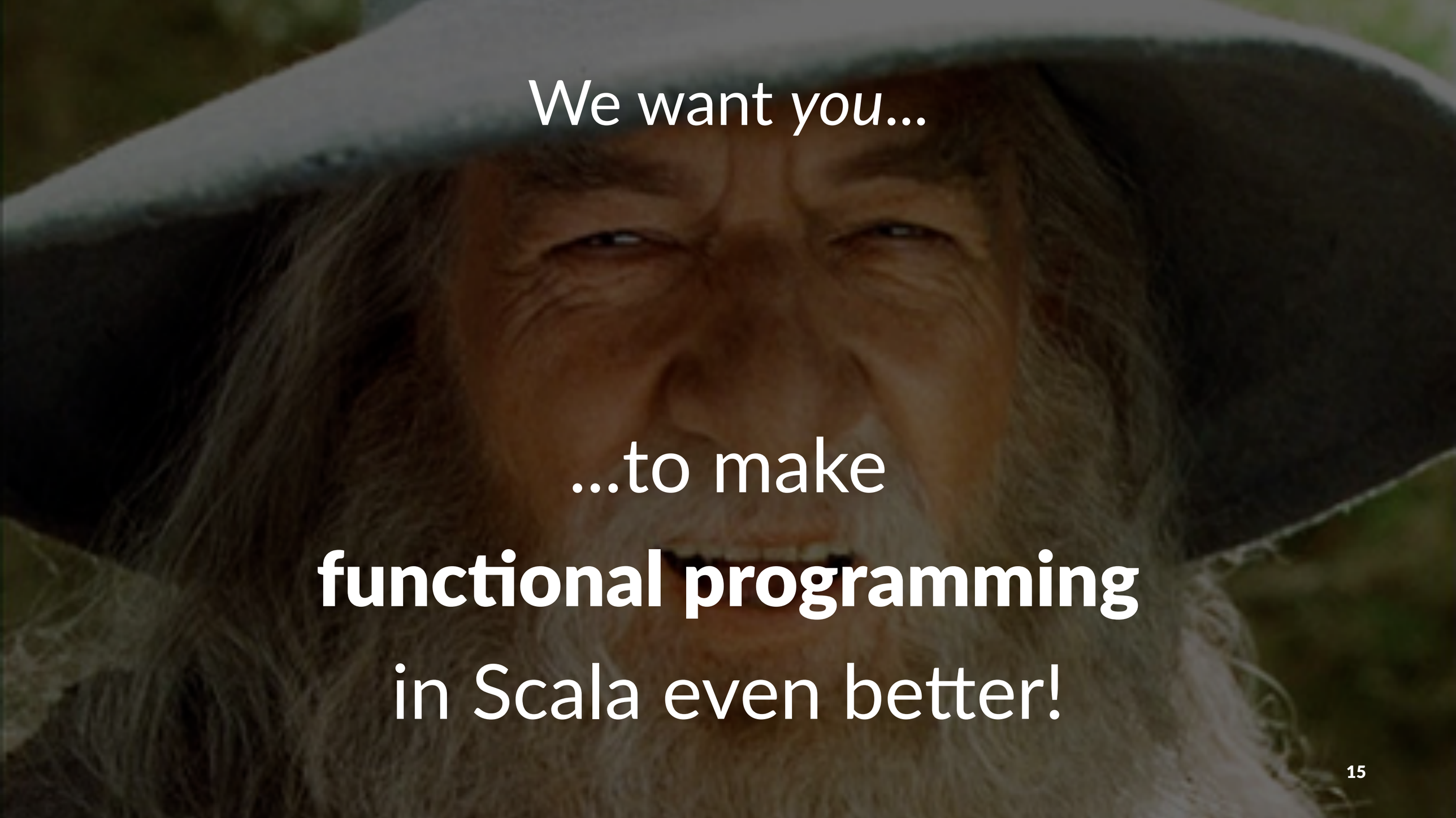
# typelevel teleology

*"Cats aims to remove barriers preventing people from doing functional programming in Scala."*

Q: Why should people do functional programming?

A: Because using it they can produce *amazing software*!

# typelevel pitch

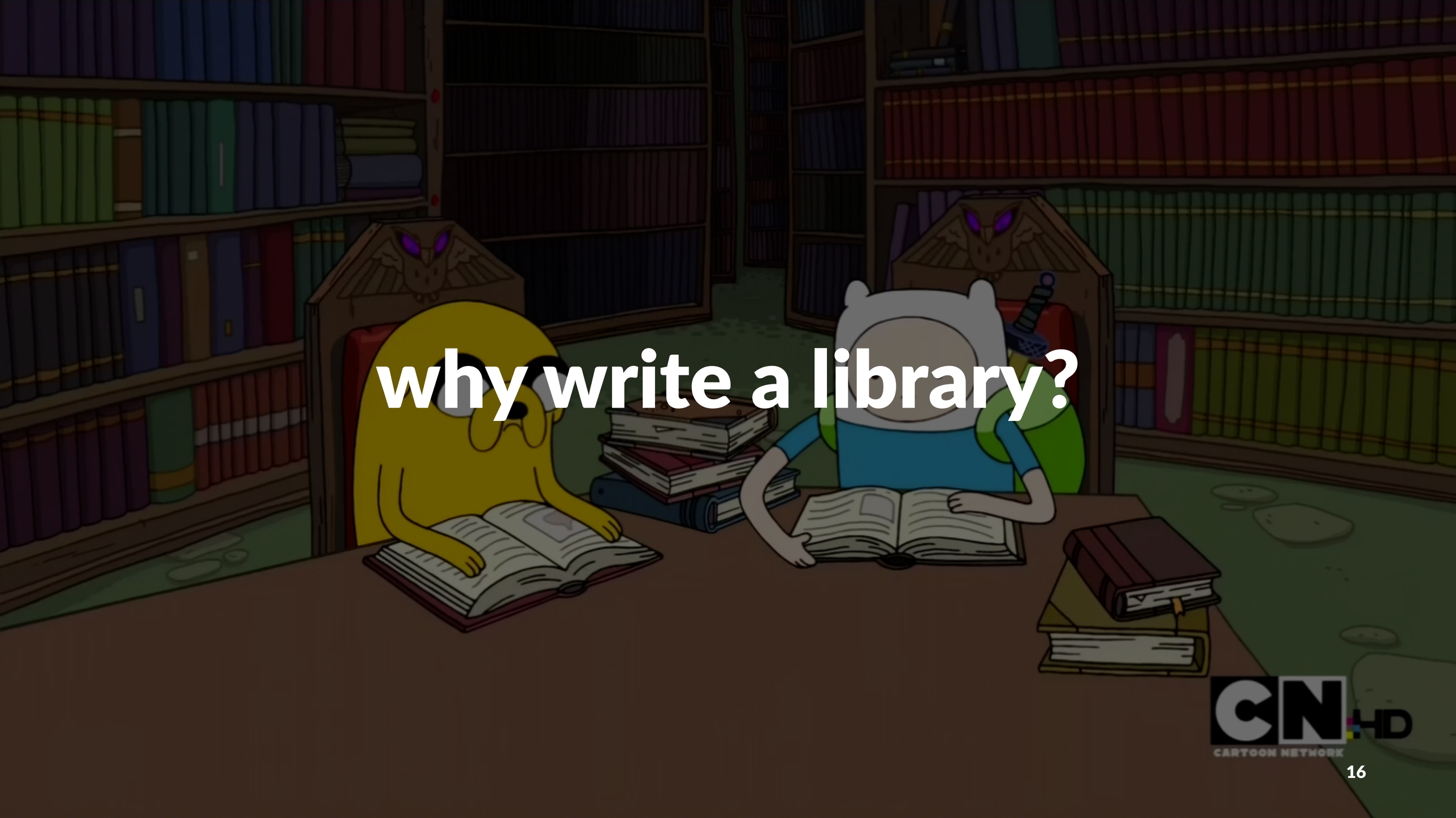


We want *you*...

...to make

**functional programming**

in Scala even better!


why write a library?
CN HD
CARTOON NETWORK

# why write a library?

1. share work you've already done
2. scratch an itch
3. explore a new idea
4. rethink an existing library
5. compulsive quest for perfection[1]
6. just for fun! ☃

---

[1] While accepting that perfection is likely unattainable.

# why write a library?

It's important to consider your own motivations:

- why are you choosing to write this code?
- how much time do you want to devote to it?
- what part of it will you find fulfilling?
- how will you know when you are successful?

# important disclaimer

You're not obligated to write a library!

(Or do any unpaid FL/OSS work for that matter!)

*My goal: be real about the pros/cons/details of making a library.*

# writing a library

Broad recommendations:

- find a design principle or ethos
- focus on what is most necessary or interesting
- let your enthusiasm run wild -- don't get too serious
- give yourself permission to stop
- create a README as soon as you push to a repo!

# anatomy of a README

For every new project I copy and update an old README:

```
NAME            | project name as you want it written
FUN             | quote, song lyric, joke, etc.
OVERVIEW        | description/summary (tl;dr usage?)
QUICK START     | sbt snippet, version info
DETAILED USAGE  | working code snippets + prose
CAVEATS         | known issues, disclaimers, etc.
LICENSE         | copyright, license link, etc.
```

(Example: https://github.com/non/sortilege)

# why start with a README?

- the longer you wait the harder it will be
- explaining your library will show you its flaws
- if you decide to release your library[2] you'll need one
- if you put the project down it's more likely someone else can pick it up
- don't obsess, just do what you can
- it can be fun!

---

[2] It's totally reasonable to work on a library you don't plan to release! (cf. early Shapeless)

# why release a library?

Semi-exhaustive list of reasons to release:

1. you'd like to use it in other code[3]

2. people have requested that you release it[4]

3. you just feel like it's ready to be released

Writing a library is for yourself -- releasing it is for others.

---

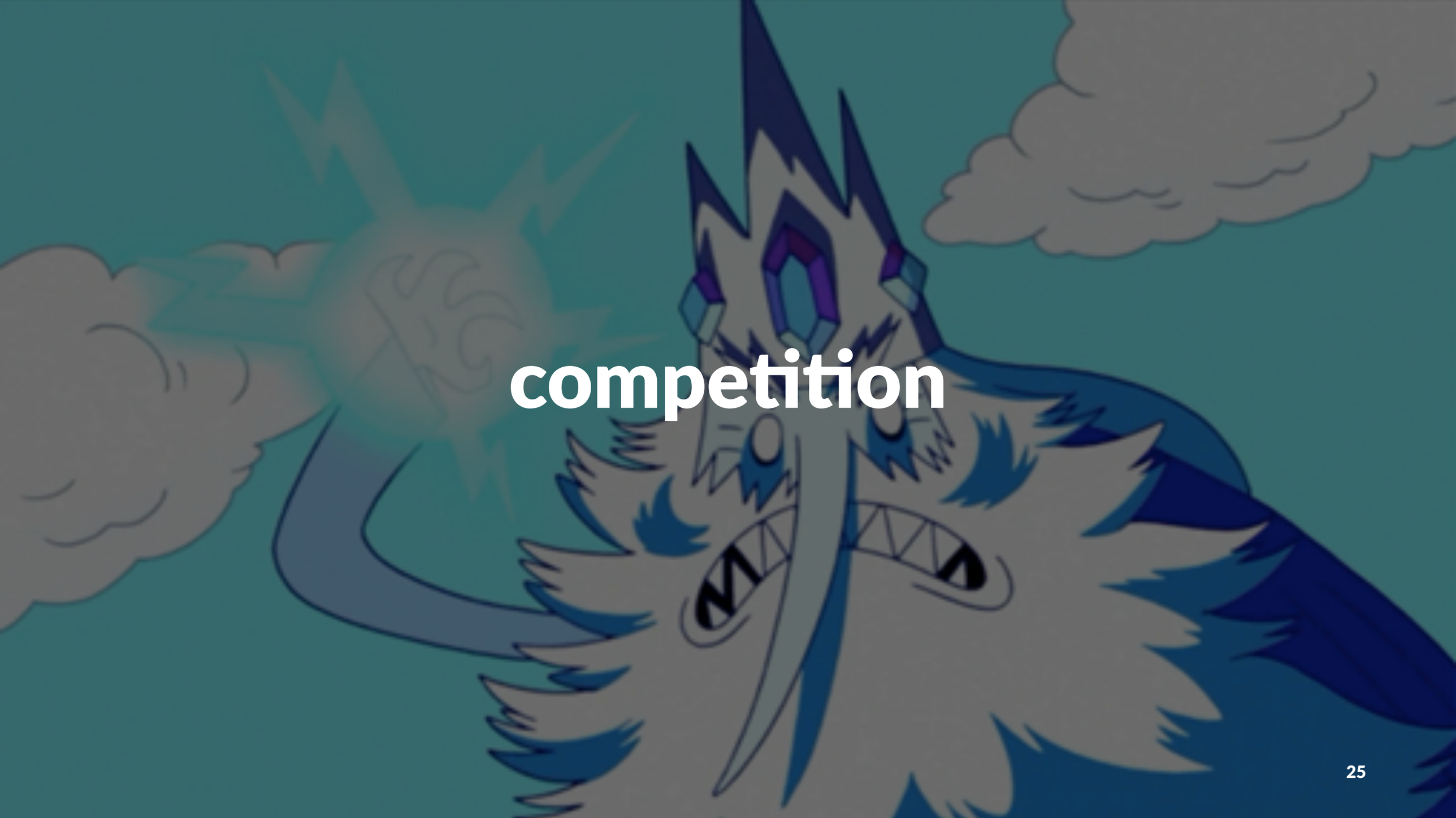
[3] If you're at work, check with your coworkers first! 😉

[4] Use your best judgement -- avoid peer pressure.

# releasing a library

- make sure you've documented your library 📖
- review, write tests, and try to measure/estimate coverage ☑
- manage people's expecations of compatibilility ⚒
- try to lay out basic roadmap/future work 🔅
- you can probably release sooner than you think[5] 💡

[5] This is still hard for me sometimes.



# competition

# competition

We should feel free to compete with (or reinvent) existing libraries.

My claim: competition is *usually* a sign of health.

- a library's existence doesn't require people to use it
- diversity now is a longterm investment in the platform
- diversity encourages specialization
- pressure on established libraries is productive
- the alternative is worse

# example #1: property-based testing

The scene: *early 2015*

The project: *Cats*

The realization: *the functions used in our law-checking were bogus!*

The problem: *most of Cats' interesting laws use function values*

 ⇒ 

# example #1: property-based testing

(Background: the problem had existed for years.)

2015-01-17: `@rickynils` had opened #136
(*Improve arbitrary function generation*)

2015-05-05: `@xuwei-k` creates scalaprops

2016-06-12: @non submits #171
(*Support non-constant arbitrary functions*)

2015-06-21: `@rickynils` merges #171

2016-02-03: `@rickynils` releases ScalaCheck 1.13.0.

# example #1: property-based testing

Not only does Kenji's new propery-based testing library not hurt ScalaCheck, but it was an active catalyst to improving ScalaCheck.

- better function values generated
- deterministic testing
- minimize generator failures
- (coming soon) displaying/re-running failing seeds

The ScalaCheck changes caught real bugs in many projects, including Cats!

# example #2: json parsing

Due to `$work` requirements, in 2012 I got interested in fast JSON parsing.

- Hand-rolled parser for performance
- Benchmarked using Caliper (now JMH) against popular parsers
  - Scala parsers (Lift, Json4s, Spray, Play, Argonaut, Rojoma, etc.)
  - Java parsers (GSON and Jackson, initially Smart-json)
- Made it easy to benchmark on arbitrary JSON files

The result?

# example #2: json parsing

- Jon Pretty's *Rapture JSON* supported multiple JSON backends
- This inspired Jawn to decouple its own parser and AST
- `jawn-parser` is now used by Circe, Rapture, and others
- It can support basically[6] any JSON AST

[6] Every AST I've encountered so far, at least.

# example #2: json parsing

Popular JSON libraries stayed popular, but in at least a few[7] cases[8] the authors were inspired to make their JSON parsers significantly faster.

🏁 🏁

The real winner was the community (IMO at least).

---

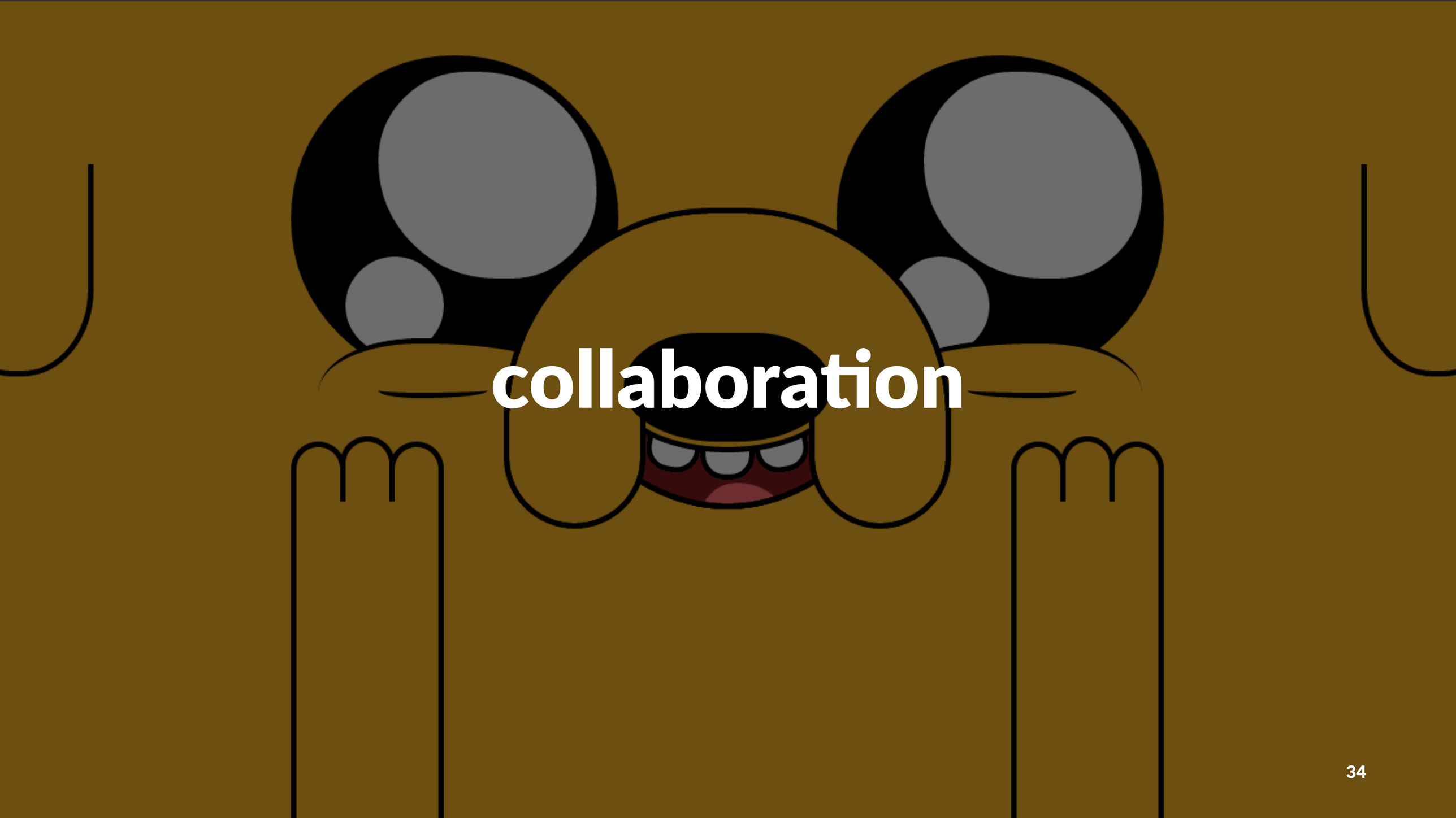
[7] Spray-json (Mathias showed me some optimization tricks I hadn't thought of!)

[8] Rojoma-json

# competition

Why bring these examples up?

- not because I want to brag about my own work
- these examples illustrate healthy diversity and competition
- there are plenty of other examples (e.g. Specs2 and ScalaTest)
- *<reference to running sub-four minute miles>*


collaboration

# collaboration

Collaboration is:

- asking questions
- reporting issues
- fixing bugs or adding features
- helping set up builds, CI, releases, etc.
- adding documentation or tutorials
- talking through future plans

# collaboration

Why are collaborators especially important?

- scratching different itches
- often have different backgrounds and knowledge than you
- can provide valuable encouragement and support
- can pick up slack when you are busy (or low energy)
- (...or just helpfully give you a poke now and then)

# example #1: spire

2011-07-17: Olivier Chafik[9] helps me[10] write a compiler plugin
2011-09-08: Tom Switzer argues against a design I proposed
2011-12-17: Tom Switzer and I propose a series of SIPs
2013-02-26: Spire 0.3.0 is released (powered by macros[11])
2013-11-17: Lars Hupel blogs about Discipline[12]
2014-03-07: `spire-ops` 0.1.0 release (later `machinist`)

---

[9] Author of loop-optimizing ScalaCLPlugin

[10] Having written Scala for all of five months

[11] Thanks to Eugene Burmako, Jason Zaugg, and Olivier Chafik

[12] Created to support efficient law-checking in Spire

# example #1: spire

Take-aways:

- My original library design was overly specific and limited
- Finding a design supporting several real use-cases was fruitful
- We were lucky to have help from so many people
- Early principles[13] helped keep Spire on track over many years.

---

[13] *"Intended to be fast, generic, and precise."*

# example #2: circe

https://github.com/circe/circe

- not a project I'm directly involved in (hi `@travisbrown`!)
- fork of argonaut-io/argonaut
- notable for its quality and dedication to interoperabilty
- 50 contributors
- 31 downstream projects (using or integrating with circe)

# example #2: circe

Circe supports (or uses):

- Jawn and Jackson (for parsing)
- Shapeless (for `auto` and `semiauto` derivation)
- Refined (uses refined type's predicates during decoding)
- Scodec (able to decode bit and byte vectors)
- Monocole (experimental optics support)
- Spray (marshaling and unmarshaling data)

# examples #2: circe

Still providing patches upstream to argonaut:

- `JsonPath` (thanks `@julien-truffaut`!)
- bug fixes and patches e.g. #257 (numeric failures)
- two-way communication between projects

Extra-project collaboration is still collaboration.

# collaboration

Collaboration take-aways from Spire:

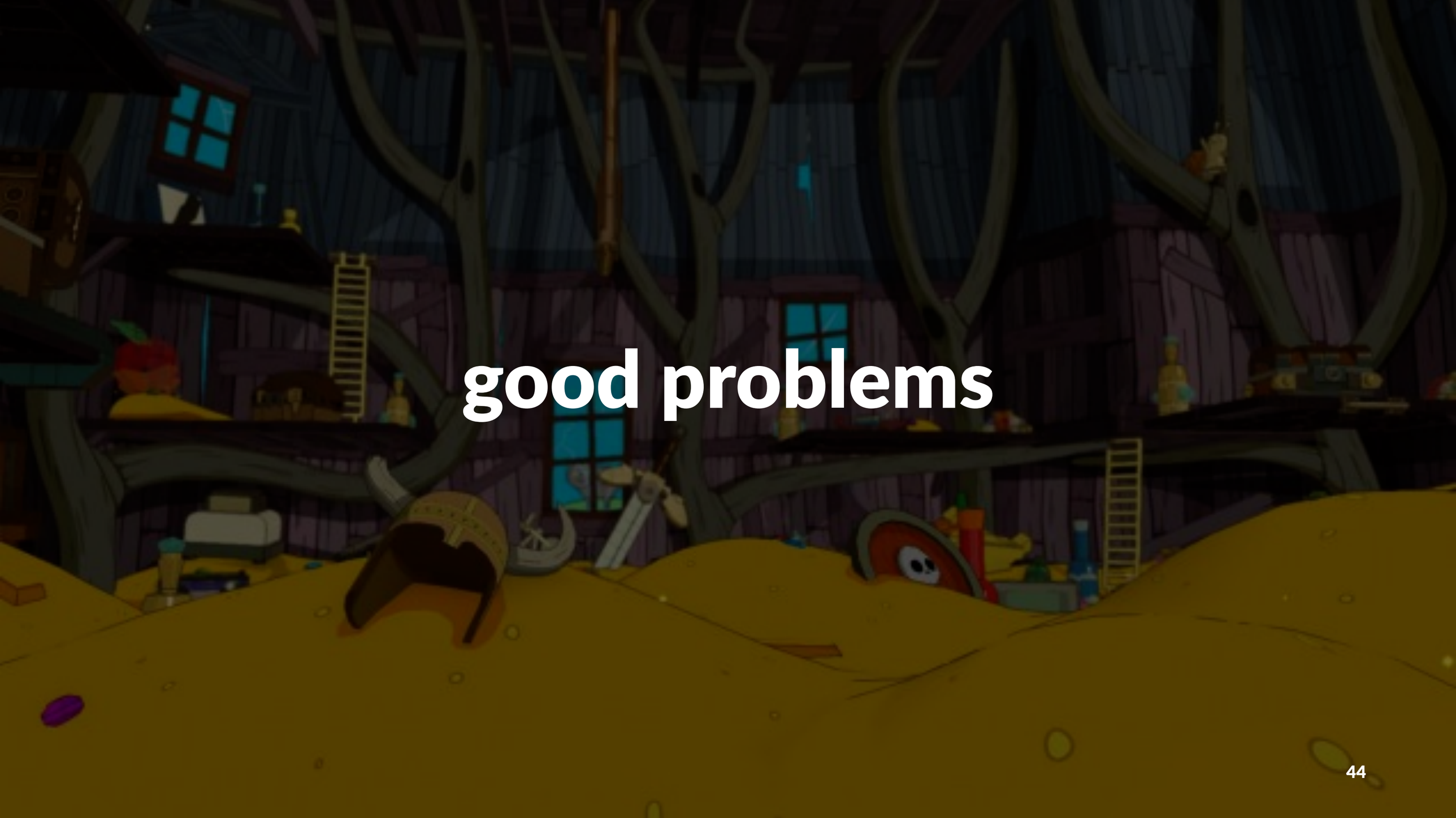
- is often initially awkward
- can be the difference between good and great
- timing outside one's control -- try to be ready[14]
- hard to unvervalue the time you take to help someone

[14] with a good README, type-checked docs, or even a manual.

# collaboration

Collaboration take-aways from Circe:

- the more connections you make the more compelling your library can be
- gives people many different ways to get involved
- forking is a form of collaboration (at least potentially)



# good problems

# good problems

List of "problems" which are a sign you're doing something *right*:

1. People are reporting bugs and opening PRs ♙
2. People are asking for releases ♘
3. Feature requests & scope creep ♗
4. People need you to upgrade versions (or not) ♖
5. Other libraries are competing in the same space ♕

# problem #1: bugs and PRs

# problem #1: bugs and PRs

Often come at unexpected times (e.g. when you're at a conference 😉)

Best case:
- Fix obvious bug (or add obviously-missing feature)
- Fits with existing style, naming conventions, etc.
- Easy to sign-off on

Usual case:
- Takes some time to understand the issue and/or solution
- May require modification[15] of some sort
- Can be hard to formulate a good response quickly

---

[15] Either of the PR or of your own vision/ideas.

# problem #1: bugs and PRs

Despite all of this, trying to respond quickly is really important!

- **especially** if they are on the wrong track!
- explaining your library is a good way to vet it
- getting even one extra person on the same page is *huge*
- you aren't the only person reading issue/PR conversations
- issue and PR conversations are historically important

(Confession: this is something at which I'm still improving.)

# problem #2: releases

So now let's say other people are hacking on your library.

It's likely that eventually they'll want it released so they can use it.

⚠ don't panic! ⚠

# problem #2: releases

Setting up your project to release is a one-time pain:

- Set up Sonatype (or Bintray) credentials
- Set up GPG key
- Configure plugins like `sbt-release`, `sbt-sonatype`, etc.
- Just requires patience and persistance
- Sort of like visiting the DMV[16] (or localized equivalent)

---

[16] *The Department of Motor Vehicles*

# problem #2: releases

- it's embarassing to mess up releases
- but not really a big deal (just release a new version)
- people will use whatever number is in your README

**You'll never botch as many releases as I have!**

**(seriously)**

# problem #2: releases

Binary compatibility:
- don't worry about this at first!
- often not expected for 0.x library versions
- can be a real pain (although less so in Scala 2.12!)

This burden is likely to grow over time:
- people asking for binary compatibility is a sign of success[17]
- be transparent with potential users about compatibility roadmap

---

[17] Simon Peyton-Jones says: *"avoid success at all costs."*

# problem #3: feature requests & scope creep

Other people will use your library in ways you never imagined.

*What happens when they propose changes you didn't foresee?*

(Similar to the problem of unexpected PRs.)

# problem #3: feature requests & scope creep

Steps to take:

1. try to look past your own assumptions
2. consider the proposal on its merits
3. think about your project's principles/goals[18]
4. is your library too big? too small? just right?
5. try to respond promptly, even to ask for more time

---

[18] Which are in your README, right? 😉

# problem #3: feature requests & scope creep

Fallacies:

1. *"I could do that same thing but better."* [19]
2. *"I'll do anything for my users."*
3. *"If I just leave this alone, maybe it will go away."*
4. *"I can't accept this for <vague reasons>"* [20]
5. *"Just one more thing..."* [21]

---

[19] If you want all code to be in your style, use ScalaFmt.

[20] When possible, document your principles in advance.

[21] Try to be up-front about effort required.

# problem #4: versioning chaos

Potential versioning problems:

- Supporting old (or new) Scala versions
- People complaining about your current dependencies
- People asking you to add dependencies
- Need to support several incompatible versions

# problem #4: versioning chaos

Solutions:

- Multi-project builds (e.g. `circe-core`, `circe-optics`, etc.)
- Have the courage to add dependencies you want.
- (And reject those you don't!)
- Don't support more than you want to initially.
- *(People who ask for new stuff are more likely to help.)*

# problem #5: other libraries

This is the big one. [22]

[22] Originally more of the talk was devoted to this one point.

# problem #5: other libraries

Let's start with the *ecosystem* analogy.

From Darwin's *On the Origin of Species* we know that:

- species evolve via natural selection
- individuals are competing for finite resources
- specialization and diversity arises from common ancestors

Indeed, our libraries are competing for time, attention, usage, etc.

# problem #5: other libraries

Observations:

- particular ethos or principles help drive specialization
- the bigger your footprint the more libraries you compete with
- if extrinsically-motivated: look for promising niches
- if intrinsically-motivated: don't sweat this stuff! [23]

[23] Libraries don't have to be used to be important!

# problem #5: other libraries

Things to consider with competing libraries:

- How much overlap is there? (A∧B)
- How much non-overlap is there? (A⊻B)
- Are the libraries converging or diverging?
- Is other other library's ethos similar or distinct?
- How active are the libraries?

# example #1: spire and algebird

twitter/algebird

*Abstract Algebra for Scala.*

(Started life as an internal library at Twitter.)

non/spire

*Powerful new number types and numeric abstractions for Scala*

(Started life as a series of proposals/hobby project.)

# example #1: spire and algebird

`Algebird ∧ Spire` (overlap)

*algebraic type classes, intervals*

`Algebird \ Spire` (Algebird-only)

*sketches, approximate data structures, "Scalding"*

`Spire \ Algebird` (Spire-only)

*numeric types/methods, lattices, sort/selection, PRNGs*

# example #1: spire and algebird

We pulled out a common subset into typelevel/algebra.

- took longer than it should have
- required trust and open dialogue
- found a design we could both live with

Algebird merged #523 (*Use standard algebra types*)
Spire merged #610 (*Migration to algebra, redux*)

Both projects will be releasing in time for Christmas 2016. 🎄

# example #1.5: algebra and cats

It turned out that Cats shared a common subset with Algebra as well.

- `Semigroup` through `Group`
- `Eq` through `Order`
- `Band`, `Semilattice`, etc.

We split these out into `cats-kernel`.

Algebird and Spire will be Cats-compatible very soon! 🦪 ♜ 🐈

# competition or collaboration?

Darwin's theory was followed by a wave of *social darwinism*.

- applied biological concepts to human society
- emphasized individual struggle
- de-emphasized cooperation

But there's an alternative...

# competition or collaboration?

*"Sociability is as much a law of nature as mutual struggle."*

-- Peter Kropotkin, *Mutual Aid: A Factor of Evolution*

(Corresponds to modern biological ideas of *mutualism* or *altruism*.)

# standardization

Why are standards useful?
- (potentially) fewer things to learn
- slow work at one level to permit faster work at another
- reify (or create) usage norms

When are standards potentially harmful?
- when the standard isn't what you really want
- when you think there's interesting work left to be done[24]

[24] Or when the design isn't sufficiently vetted.

# standardization

*Rugged individualist*: probably never supports standards.

*Technocrat*: probably wants everything standardized.

Most of us are somewhere in-between.

# standardization

*typelevel/algebra* can be thought of as an informal standard.

(Same with *typelevel/cats-kernel*.)

This is my preferred way to standardize.

Ultimately you'll have to judge the situation yourself!

# conclusions

# conclusions

- Writing libaries can be lots of fun.
- Follow your motivation
- Be honest with yourself (and others)
- Meditate on your principles
- Find opportunities for collaboration
- Embrace healthy competition
- Let a thousand flowers bloom! 🌼🌻🌸🌷

Scala eXchange *9 December 2016*

# *the end*

Erik Osheim (@d6)

stripe