How we are going to migrate to Scala 3

Lukas Rytz, Scala Team @ Lightbend

@lrytz Twitter / GitHub

Bern, CH

Talk Outline

🛤️ Perspective on Scala 3

🤝 Working Together

⏳ Timeline and Migration Path

Part 1

A Perspective on Scala 3

Principles Behind Scala 3

- Compatibility with Scala 2 – evolution, no revolution
- Simplifications: features need to carry their weight
- Embrace idioms and become more opinionated
- Consistency: enforce Scala's strengths

Paradigm Shift

- Best example: implicits. Low-level feature to express
  - Type classes
  - Extension methods
  - Contextual abstraction
  - Type level computation
- Implicit conversions are too easy to define

Extension Methods, Toplevel Definitions

```
package object p {
  implicit class StringExtension(private val s: String)
      extends AnyVal {
    def bold = s"*$s*"
  }
}
```

Simpler with problem-specific features in Scala 3:

```
package p
def (s: String) bold = s"*$s*"
```

Enumerations, Abstract Data Types

- scala.Enumeration: hacks using reflection, open bugs
- ADTs are a very common idiom, require boilerplate

```scala
sealed abstract class Option[+T] {
  def isEmpty = this eq None
}
final case class Some[+T](v: T)
    extends Option[T]
case object None
    extends Option[Nothing]
```

```scala
enum Option[+T] {
  case Some(v: T)
  case None
  def isEmpty = this eq None
}
```

Type Class Encoding

```
trait Show[-A] { def show(a: A): String }

object Show {
  implicit val IntShow: Show[Int] = a => s"int $a"

  implicit def optionShow[T](implicit s: Show[T]): Show[Option[T]] = {
    case Some(v) => s"some ${s.show(v)}"
    case None    => "none"
  }
}

def show[T](v: T)(implicit s: Show[T]) = s.show(v)

show(Some(1)) // "some int 1"
```

```
delegate IntShow for Show[Int] = ...
```

```
delegate [T] for Show[Option[T]] given (s: Show[T]) = ...
```

```
def show[T](v: T) given (s: Show[T]) = ...
```

Type Class Encoding

```
trait Show[-A] { def show(a: A): String }

delegate IntShow for Show[Int] = a => s"int $a"

delegate [T] for Show[Option[T]] given (s: Show[T]) = {
  case Some(v) => s"some ${s.show(v)}"
  case None    => "none"
}

def show[T](v: T) given (s: Show[T]) = s.show(v)

show(Some(1)) // "some int 1"
```

Type System Evolution

- Union and intersection types (not tagged)
- Type lambdas
- Function types: dependent, polymorphic, implicit
- Improved type inference

Scala 3 by Migration Impact

1. Breaking changes
2. New features
3. De-emphasized features that continue to be supported
4. Unchanged features

dotty.epfl.ch/docs/reference/features-classification.html

Breaking Changes

- Unsupported:

  `forSome` (wildcards `List[_]` are ok)

  early initializers

- Scala 2 compatibility mode: procedure syntax

  symbol literals

  auto application

  `DelayedInit` (to do)

  packages in implicit scope

  🐌 operator `_@_*`

Macros and Metaprogramming

- New API to implement macros
  - More principled (inlining, quotes, splices, TASTy-based)
  - Safer (typed trees only)
  - Talk by Nicolas Stucki (earlier today)
- Some macros no longer needed (type class derivation)

Specialization

- Still on the drawing board
- Scala 3 will deliver specialization for core types (functions, tuples) neede for performance
- By difficulty: methods, classes, superclasses / trais
- Reach out to the Scala 3 team at EPFL if you're affected

New Features

- Incomplete list: trait parameters, opaque types, toplevel definitions, enums, extension methods, enhanced type system, match types, inline matches

- New features can be introduced gradually in a codebase
  - Requirement: no cross-building with Scala 2

Scala 2 Support

- The Scala 3 compiler supports almost all of Scala 2
- Scala 2 features that continue to work:

implicits (parameters, values, conversions, classes)

package objects, package object inheritance

value classes

XML literals

compound types (`A with B`)

Unchanged in Scala 3

- Standard library, including collections
- Tooling: sbt, IntelliJ, VS Code
- Ecosystem: we will invest in helping maintainers to cross-build their libraries
- Everything else, for example: classes and objects

  functions pattern matching JVM & JS

  Java interop regression tests

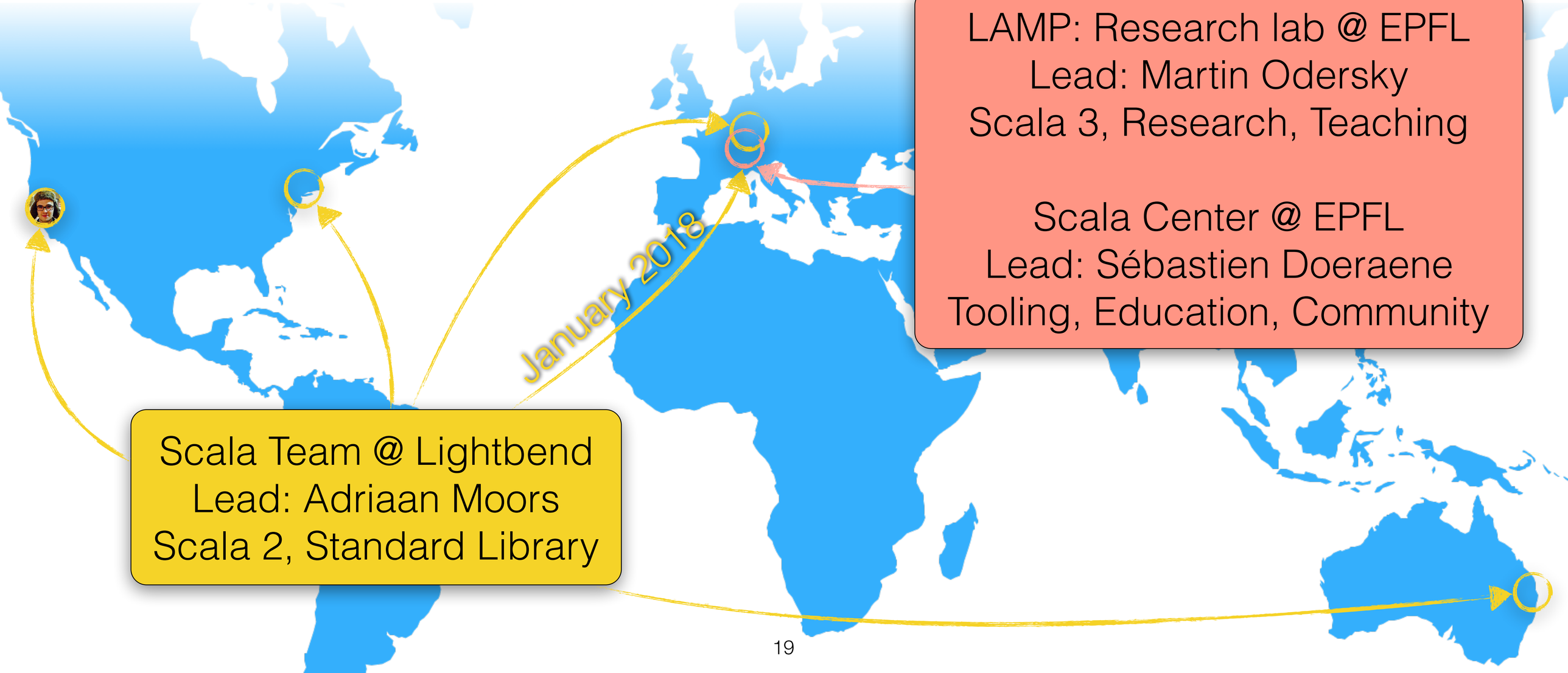
Talk Outline

🛤️ Perspective on Scala 3

🤝 Working Together

⏳ Timeline and Migration Path

Who's Behind Scala



LAMP: Research lab @ EPFL
Lead: Martin Odersky
Scala 3, Research, Teaching

Scala Center @ EPFL
Lead: Sébastien Doeraene
Tooling, Education, Community

Scala Team @ Lightbend
Lead: Adriaan Moors
Scala 2, Standard Library

Scala 3 Design Discussions

- Issues or PRs at github.com/lampepfl/dotty
- Discourse contributors.scala-lang.org/c/language-design
- SIP Committee: EPFL, Scala Center, Lightbend, Community
- Offline, over ☕, 🍲 or 🍺
  - 3x per year at Lightbend meetups (Scala team + Martin)
  - Weekly at EPFL meetings (EPFL team + Adriaan)

Scala 2.14: Prepare for 3

- Backport features: type lambdas, opaque types, trait parameters, toplevel definitions
- Deprecations: `forSome` existentials, auto-application, package object inheritance, early initializers
- Removals: procedure syntax, symbol literals

2.14 and 3: Developed Together

- Same standard library
- Invest in sharing code: test suite, compiler components
- Enable maintainers to cross-build on 2.14 and 3

2.14 and 3: Binary Interop

- Scala 3 code can use libraries compiled by Scala 2.14
  - Allows migrating the ecosystem gradually
  - The compilers generate binary compatible bytecode
  - Caveat: Scala 2 macros
- Scala 2.14 will emit TASTy, enables common tooling

Testing

- Binary compatibility: Build with both compilers, compare classfiles
- Integration test for TASTy: "frankenstein" compiler
  - Scala 2.14: Parser, Typer → TASTy
  - Scala 3: TASTy → bytecode

Community Build

- Build the Scala ecosystem (compatible versions) from source for any Scala version
- Roughly 3M lines of code (2.12)
- Scala 3 community build getting started
- Testing, quantifying the impact of breaking changes

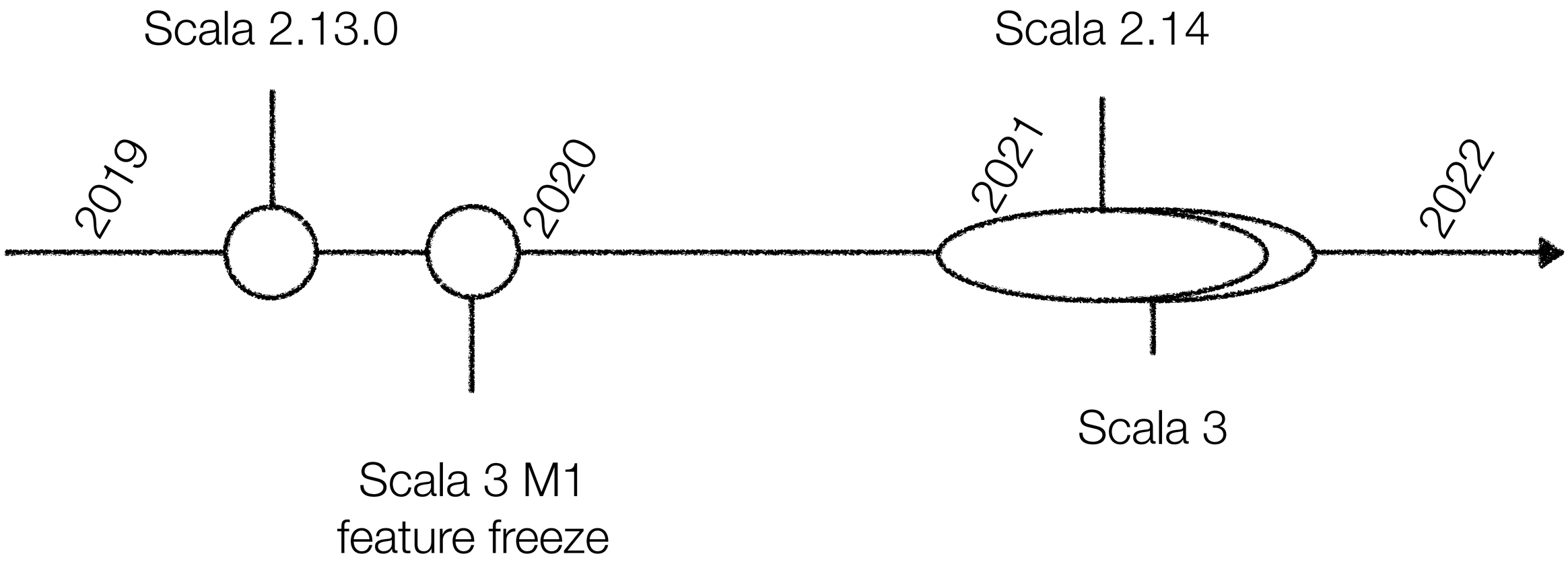
Talk Outline

🛤️ Perspective on Scala 3

🤝 Working Together

⏳ Timeline and Migration Path

Timeline


Scala 2.13.0
Scala 2.14
2019
2020
2021
2022
Scala 3
Scala 3 M1
feature freeze

Migration

- Move to 2.14 first
- Use scalafix (github.com/scala/scala-rewrites) for syntax changes (procedure syntax, symbol literals)
- On 2.14: migrate off deprecated features (`forSome`, early initializers → trait parameters)
- Rewrite macros when migrating to Scala 3

Cross Building

- Goal: one cross-building ecosystem
  - Upgrade dependencies separately from Scala 3
- Scala version dependent source directories
  - Needed for projects defining macros
  - Maybe: `//# if scala.version =~ "3.*"`

Scala Maintenance

- Lightbend Scala Team
  - Develop 2.14
  - Maintain 2.14 for a long time
  - After 2.14, support and maintain Scala 3
- LAMP Team at EPFL: develop Scala 3

Summary

🛤️ Scala 3 is Scala 2 + 1

🤝 We are all working together to ensure migration will be smooth