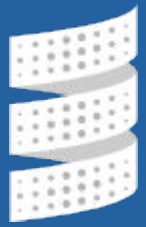
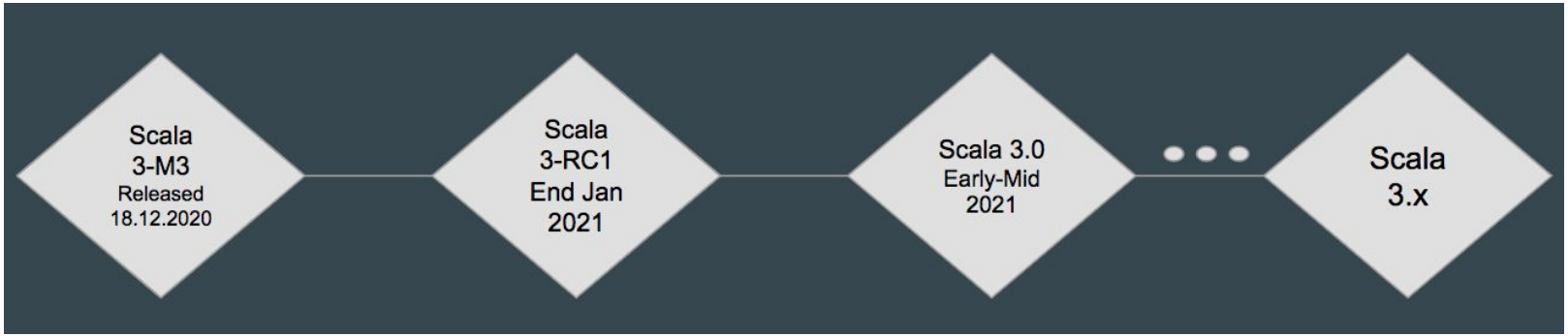
Teads

Scala 3



Early 2021 overview

Timeline



Not there yet, official release planned for mid year, BUT

- Libs are already starting to release a Scala3 version
- There shouldn't be much more changes until the release outside of fixes
- sbt 1.5.0-M1 has been released two days ago with Scala3 compatibility

Compatibility - 2.13.4+ => 3.x

```
sbt.version = 1.5.0-M1
```

```
lazy val scala2UsingScala3 = project.in(file("scala2-using-scala3"))
  .settings(
    scalaVersion := "2.13.4",
    scalacOptions += "-Ytasty-reader",
    resolvers += Resolver.JCenterRepository,
    libraryDependencies += "org.typelevel" % "cats-core_3.0.0-M3" % "2.3.1"
  )
```

2.13.4+ Scala project can use Scala 3.x libraries

Still need a custom resolver for now

Compatibility - 3.x => 2.13

```
sbt.version = 1.5.0-M1
```

```
lazy val scala3UsingScala2 = project.in(file("scala3-using-scala2"))
  .settings(
    scalaVersion := "3.0.0-M3",
    libraryDependencies += "org.typelevel" % "cats-core_2.13" % "2.3.1"
  )
```

3.x Scala project can use Scala 2.13 libraries

Compatibility - Metaprogramming

- New macro implementation with Scala 3
- Macros from Scala 2 and Scala 3 are not compatible, you cannot depend on the other kind
- Libs and projects will have to manage two macro definitions to stay compatible https://scalacenter.github.io/scala-3-migration-guide/docs/macros/migration-tutorial.html#mixing-macro-definitions

Migration

- Goal is to make it automatic in most cases
- There are some compatibilities which will need to be handled manually
- You can also switch from the Scala 2 syntax to the new Scala 3 syntax automatically https://scalacenter.github.io/scala-3-migration-guide/docs/scala-3-syntax-rewriting.html

What changed - Quiet syntax

- Goodbye parenthesis

- Indentation-based with optional braces possible

```
if x < 0 then
   "negative"
else if x == 0 then
   "zero"
else
   "positive"

if x < 0 then -x else x

while x >= 0 do x = f(x)

for x <- xs if x > 0
yield x * x

for
   x <- xs
   y <- ys
do
   println(x + y)

try body
catch case ex: IOException => handle
```

What changed - Re-scoping implicit

- Implicits were too powerful
  Used for too many different things

- Challenging for tooling

  But were becoming better in latest versions of IntelliJ plugin

```scala
// Scala 2

def i1(
  implicit
  x: T,
  y: ExecutionContext
): Future[T] = ???

implicit def i2(x: Int): String = ???

implicit class PrettyString[A](
  v: List[A]
) {
   def toPrettyString: String = ???
}
```

What changed - Re-scoping implicit - Given

- Provide context parameters

  Used in combination with `using`

- Create a `given` alias to provide a specific value to the scope

```
trait Ord[T]:
  def compare(x: T, y: T): Int

given intOrd: Ord[Int] with
  def compare(x: Int, y: Int) =
    if x < y then -1 else if x > y then +1 else 0

given Ord[Double] with
  def compare(x: Double, y: Double) =
    if (x < y) then -1 else if x > y then +1 else 0

// Alias given
given global: ExecutionContext = ForkJoinPool()
```

What changed - Re-scoping implicit - Using

```
def maximum[T](xs: List[T])(using Ord[T]): T =
  xs.reduceLeft(max)

// Replacing implicitly
summon[Ord[Int]].compare(2,3)
// Behind, written simply as
def summon[T](using x: T): x.type = x
```

https://dotty.epfl.ch/docs/reference/contextual/givens.html
https://dotty.epfl.ch/docs/reference/contextual/using-clauses.html

What changed - Re-scoping implicit - Type conversion

```
// Source code
abstract class Conversion[-T, +U] extends (T => U):
  def apply (x: T): U

// Implem
given Conversion[String, Token] with
   def apply(str: String): Token = new KeyWord(str)

// Even shorter
given Conversion[String, Token] = new KeyWord(_)
```

https://dotty.epfl.ch/docs/reference/contextual/conversions.html

What changed - Re-scoping implicit - Extension methods

```
case class Circle(x: Double, y: Double, radius: Double)
extension (c: Circle)
  def circumference: Double = c.radius * math.Pi * 2

// With generic type
extension [T](xs: List[T])
   def second = xs.tail.head
extension [T: Numeric](x: T)
   def + (y: T): T = summon[Numeric[T]].plus(x, y)
```

https://dotty.epfl.ch/docs/reference/contextual/extension-methods.html

Type system improvements - Enumerations

- Provide useful basic methods

- Enough in most cases
  So libs like `enumeratum` should be way less needed

```
enum Color(val rgb: Int):
  case Red   extends Color(0xFF0000)
  case Green extends Color(0x00FF00)
  case Blue  extends Color(0x0000FF)

Color.valueOf("Blue") // Blue
Color.values // Array(Red, Green, Blue)
Color.fromOrdinal(0) // Red
```

Type system improvements - Opaque types

- Replace current value classes
- Zero cost abstraction !

```
opaque type PlacementId = Long

extension (p: PlacementId)
  def toVast: String = s"pid=$p"
```

Type system improvements - Intersection types

```
trait A:
  def children: List[AA]

trait B:
  def children: List[BB]

class C extends A, B:
  def children: List[AA & BB] = ???

val x: A & B = new C
val y: List[A & B] = x.children
```

https://dotty.epfl.ch/docs/reference/new-types/intersection-types.html

Type system improvements - Union types

```
case class Username(name: String)
case class Password(hash: Hash)

def help(id: Username | Password) =
  id match
    case Username(name) => lookupName(name)
    case Password(hash) => lookupPassword(hash)
```

https://dotty.epfl.ch/docs/reference/new-types/union-types.html

Better OOP - Trait with parameters

```
trait Greeting(val name: String):
   def msg = s"How are you, $name"

class C extends Greeting("Bob"):
   println(msg)
```

No need to switch to abstract classes anymore

https://dotty.epfl.ch/docs/reference/other-new-features/trait-parameters.html

Better OOP - Transparent traits

```
transparent trait S
trait Kind
object Var extends Kind, S
object Val extends Kind, S

// Before x: Set[Kind with S with Product with Serializable]
// Now x: Set[Kind]
val x = Set(if condition then Val else Var)
```

Improved type inference !

https://dotty.epfl.ch/docs/reference/other-new-features/transparent-traits.html

And way more

- Context functions
- Improved compilation errors
- Improved type inference
- Polymorphic function types
- Type lambdas (was working but needed a plugin before)
- Match types
- Export clauses
- Improved metaprogramming (?)
- NPE out of Scala (but still here with Java interop…)

…

References

https://dotty.epfl.ch/
https://docs.scala-lang.org/scala3/new-in-scala3.html
https://scalacenter.github.io/scala-3-migration-guide/
https://github.com/lampepfl/dotty/issues

Teads

Questions ?