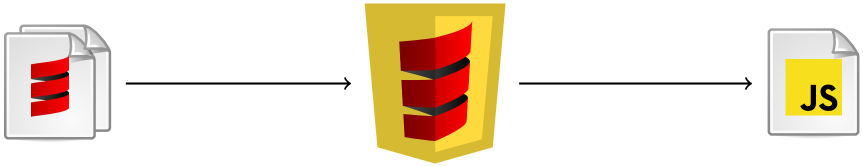
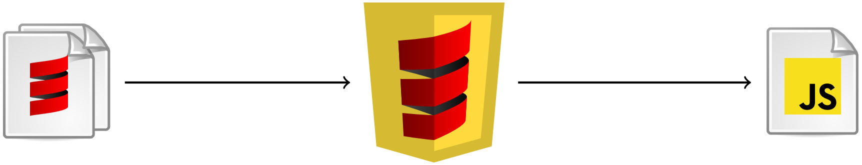
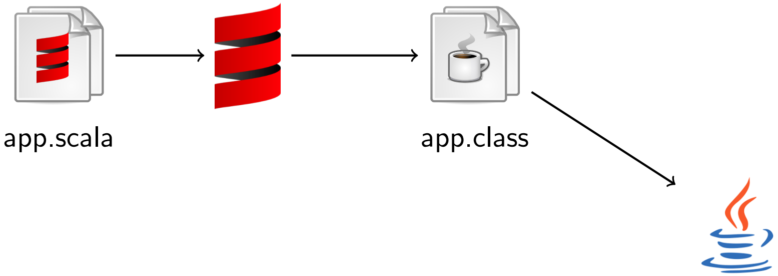

JS

# The Scala.js Compilation Pipeline

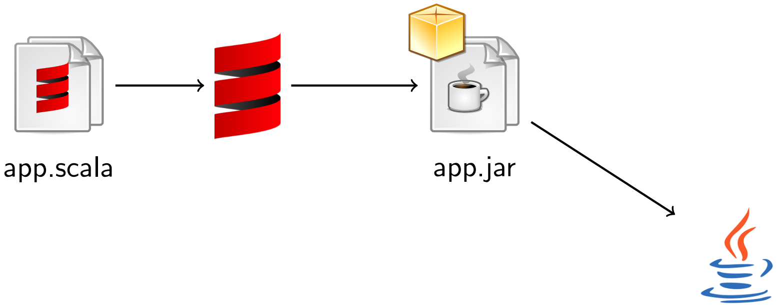


Tobias Schlatter – @gzm0

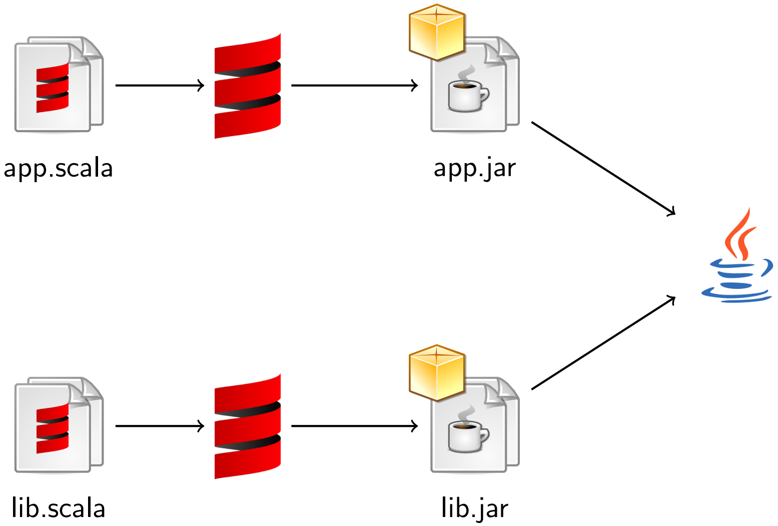
## Scala JVM Pipeline



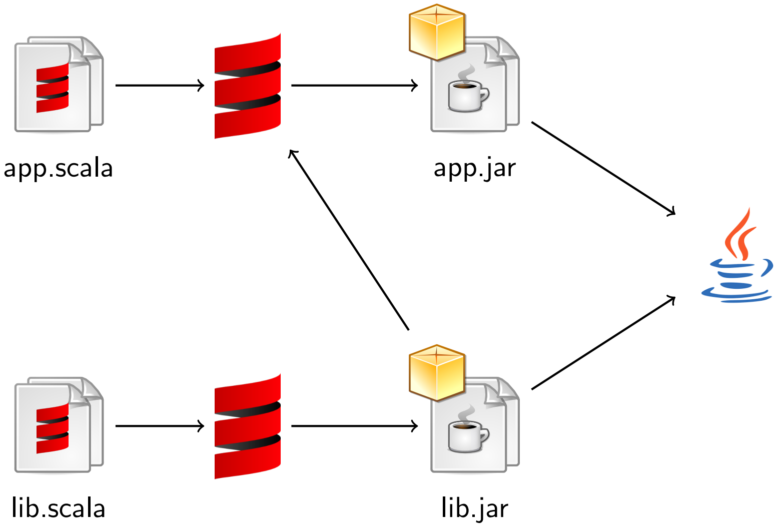
## Scala JVM Pipeline



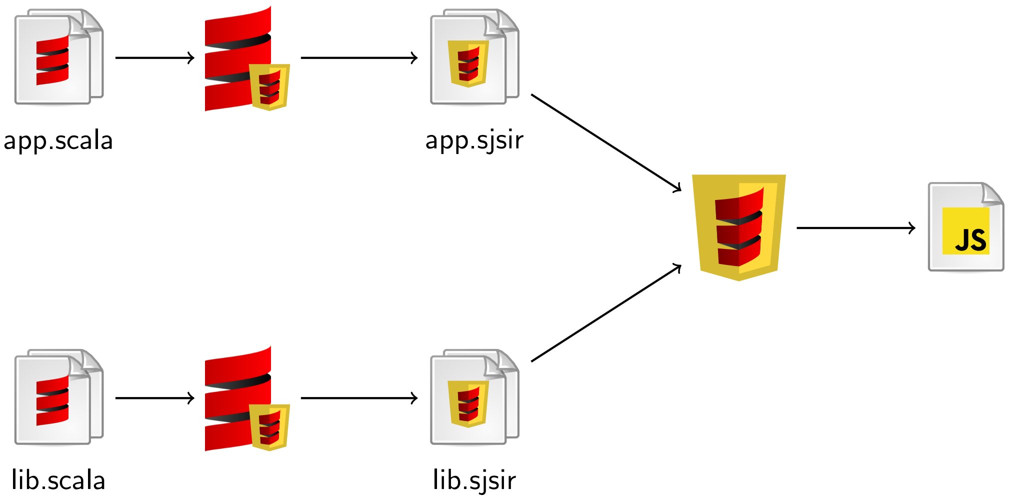
## Scala JVM Pipeline



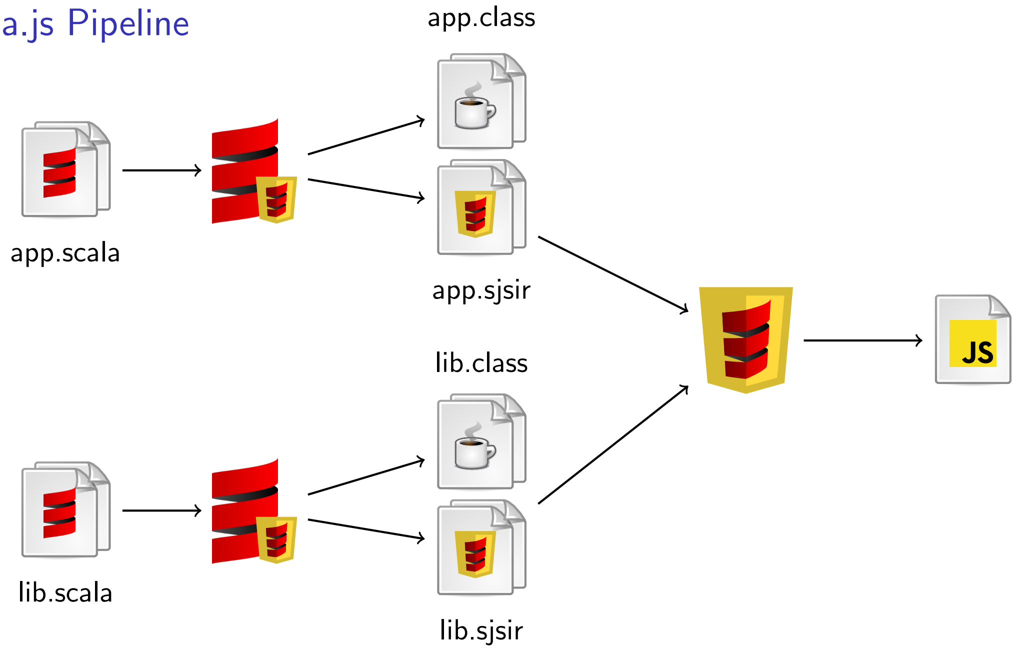
## Scala JVM Pipeline



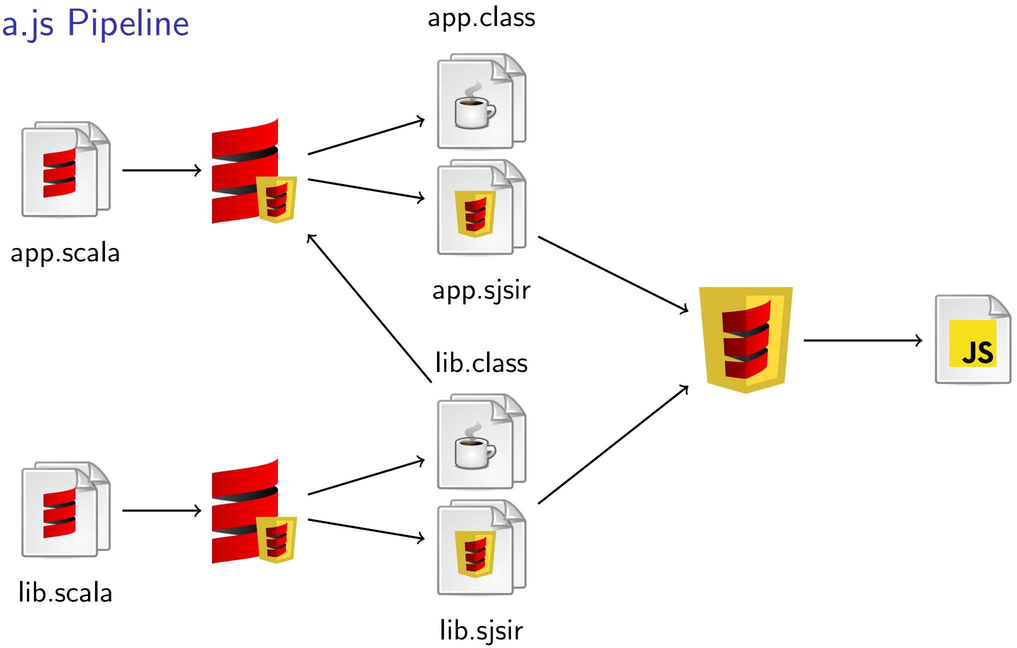
## Scala.js Pipeline



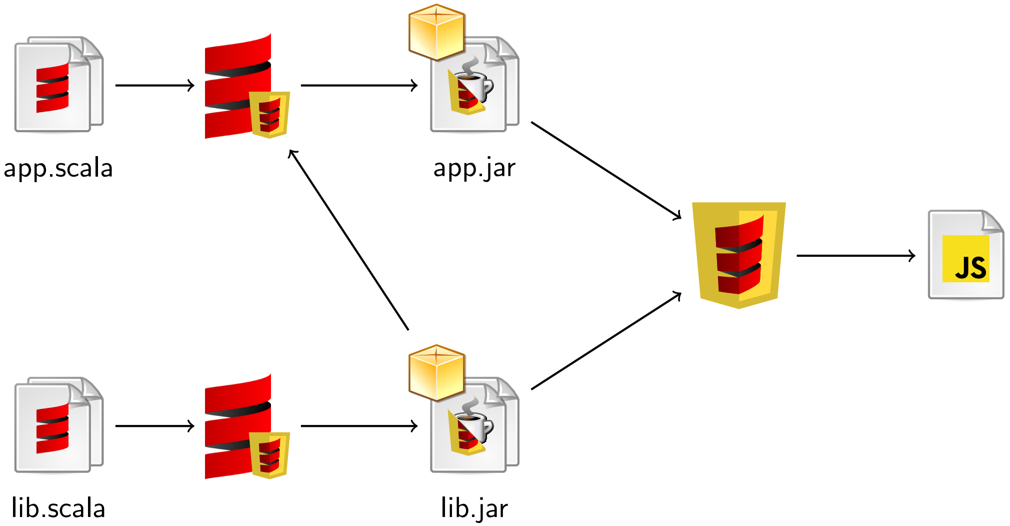
## Scala.js Pipeline



## Scala.js Pipeline



## Scala.js Pipeline



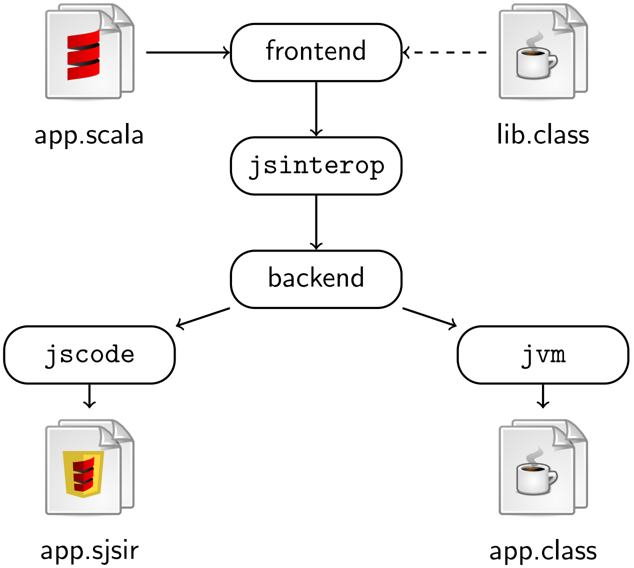
# Phases of the Scala.js Compiler

```
parser
namer
packageobjects
typer
jsinterop
patmat
superaccessors
extmethods
pickler
refchecks
uncurry
tailcalls
specialize
explicitouter
erasure
posterasure
lazyvals
lambdalift
constructors
flatten
mixin
jscode
cleanup
delambdafy
icode
jvm
terminal
```

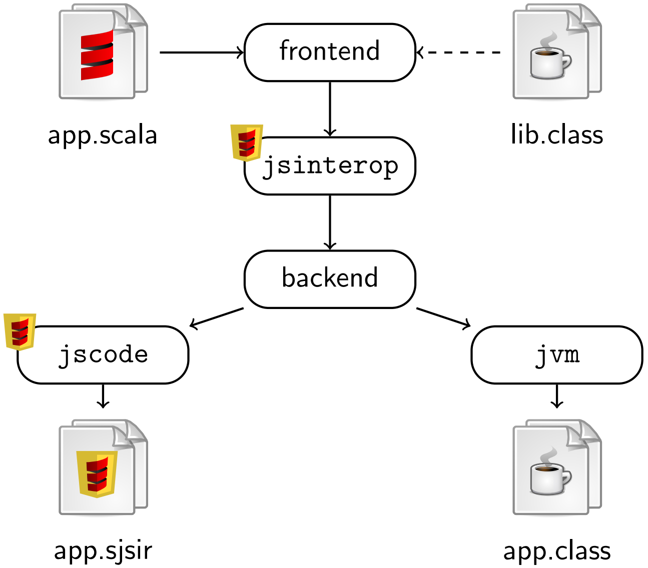
## Phases of the Scala.js Compiler

```
parser
namer
packageobjects
typer
jsinterop
patmat
superaccessors
extmethods
pickler
refchecks
uncurry
tailcalls
specialize
explicitouter
erasure
posterasure
lazyvals
lambdalift
constructors
flatten
mixin
jscode
cleanup
delambdafy
icode
jvm
terminal
```

## Phases of the Scala.js Compiler



## Phases of the Scala.js Compiler



## Scala.js Compiler – frontend

```
@JSExport
class MultiAlerter {
  val msgs = new HelloFactory

  @JSExport
  def multiAlert(n: Int) =
    for (i <- 1 to n) dom.alert(msgs.hello(i))




}

class HelloFactory {
  def hello(x: Int) =
    s"Hello World #$x"

  def helloDebug() = "Hello World"
}
```

## Scala.js Compiler – frontend

```
@JSExport
class MultiAlerter {
  val msgs: HelloFactory = new HelloFactory

  @JSExport
  def multiAlert(n: Int): Unit =
    for (i <- 1 to n) dom.alert(msgs.hello(i))




}

class HelloFactory {
  def hello(x: Int): String =
    s"Hello World #$x"

  def helloDebug(): String = "Hello World"
}
```

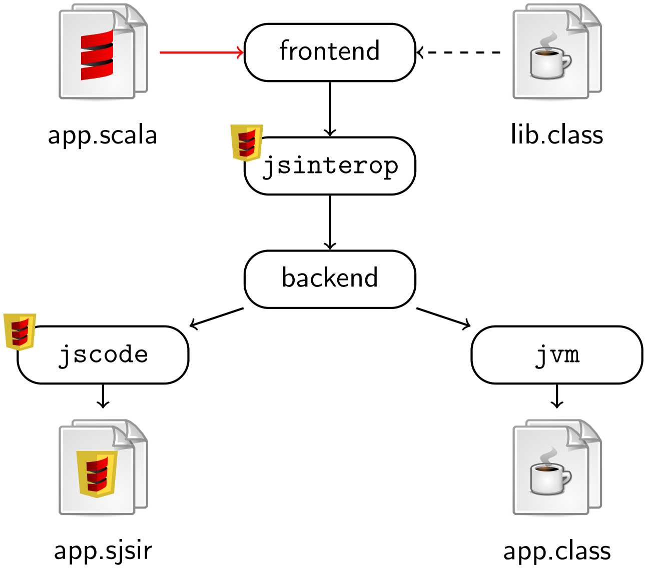
## Scala.js Compiler – frontend

```scala
@JSExport
class MultiAlerter {
  val msgs: HelloFactory = new HelloFactory

  @JSExport
  def multiAlert(n: Int): Unit = {
    intWrapper(1).to(n).foreach[Unit] { (i: Int) =>
      dom.alert(msgs.hello(i))
    }
  }
}

class HelloFactory {
  def hello(x: Int): String =
    s"Hello World #$x"

  def helloDebug(): String = "Hello World"
}
```

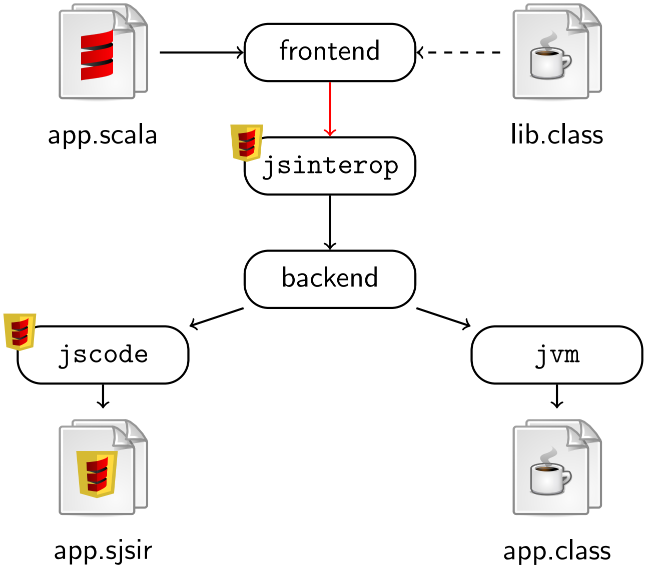
## Scala.js Compiler – frontend

```scala
@JSExport
class MultiAlerter {
  val msgs: HelloFactory = new HelloFactory

  @JSExport
  def multiAlert(n: Int): Unit = {
    intWrapper(1).to(n).foreach[Unit] { (i: Int) =>
      dom.alert(msgs.hello(i))
    }
  }
}

class HelloFactory {
  def hello(x: Int): String =
    StringContext.apply("Hello World #", "").s(x)

  def helloDebug(): String = "Hello World"
}
```

## Phases of the Scala.js Compiler



## Phases of the Scala.js Compiler



## Scala.js Compiler – `jsinterop`

### Responsibilities

- JavaScript Interoperability Errors
- Exports / JavaScript Methods

```scala
@JSExport
class MultiAlerter {
  val msgs: HelloFactory = new HelloFactory

  @JSExport
  def multiAlert(n: Int): Unit = // snip
  def $js$exported$meth$multiAlert(n: Int): Any = multiAlert(n)
}

class HelloFactory // Unchanged
```

## Scala.js Compiler – `jsinterop`

### Responsibilities

- JavaScript Interoperability Errors
- Exports / JavaScript Methods

```
@JSExport
class MultiAlerter {
  val msgs: HelloFactory = new HelloFactory

  @JSExport
  def multiAlert(n: Int): Unit = // snip
  def $js$exported$meth$multiAlert(n: Int): Any = multiAlert(n)
}

class HelloFactory // Unchanged
```

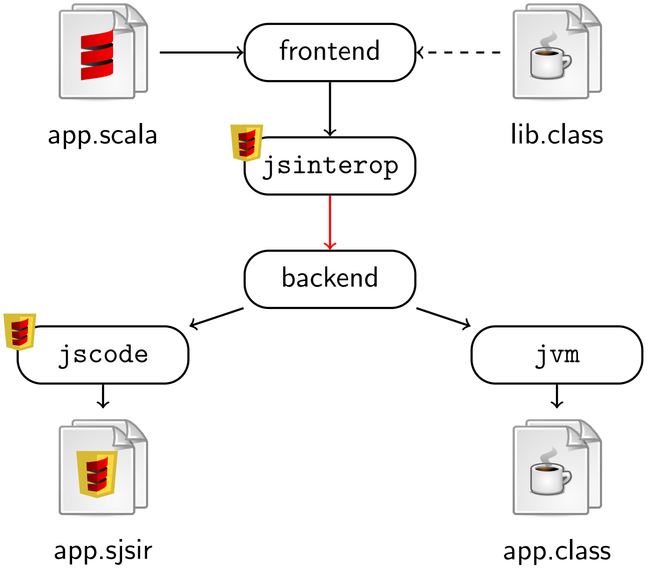
## Scala.js Compiler – `jsinterop`

### Responsibilities

- JavaScript Interoperability Errors
- Exports / JavaScript Methods

```scala
@JSExport
class MultiAlerter {
  val msgs: HelloFactory = new HelloFactory

  @JSExport
  def multiAlert(n: Int): Unit = // snip
  def $js$exported$meth$multiAlert(n: Int): Any = multiAlert(n)
}

class HelloFactory // Unchanged
```

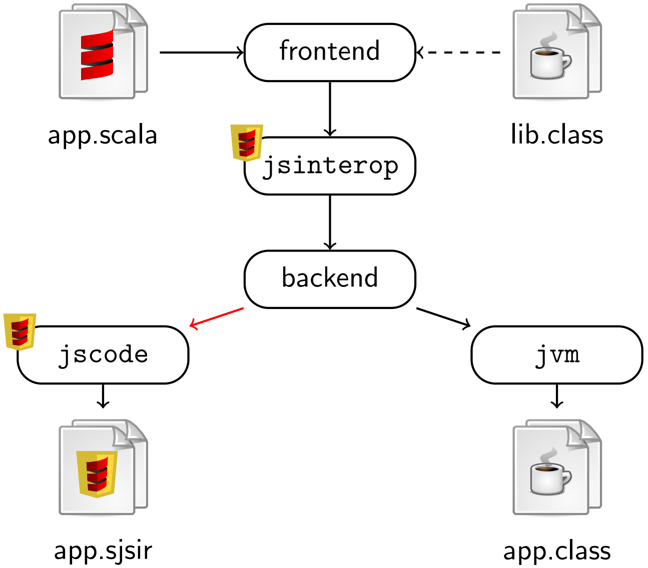
# Phases of the Scala.js Compiler



## Scala.js Compiler – After backend

```
class MultiAlerter {
  val msgs: HelloFactory = _
  def <init>(): MultiAlerter = { msgs = new HelloFactory }

  def multiAlert(n: Int): Unit = {
    RichInt.to$extension0(intWrapper(1), n).foreach[Unit](
      (new <$anon: Function1>(MultiAlerter.this): Function1)));
  }
  def $js$exported$meth$multiAlert(n: Int): Object = // snip
}

class HelloFactory {
  def hello(x: Int): String = {
    new StringContext(wrapRefArray(Array{"Hello World #", ""}))
      .s(genericWrapArray(Array{Int.box(x)}));
  }
  def helloDebug(): String = "Hello World"
}
```

## Phases of the Scala.js Compiler



## Scala.js Compiler Output: The IR

### General

- AST form (typed)
- Complex expressions
- JavaScript operations

### Types

- No generics (erasure)
- Primitive types (`int`)
- Class types (`foo.Bar`)

### Classes / Interfaces

- Single class inheritance
- Multi interface inheritance
- No Overloading
  (instead: name mangling)
- JavaScript methods
  (aka Exports)

## Scala.js Compiler Output: The IR

### General

- AST form (typed)
- Complex expressions
- JavaScript operations

### Classes / Interfaces

- Single class inheritance
- Multi interface inheritance
- No Overloading
  (instead: name mangling)
- JavaScript methods
  (aka Exports)

### Types

- No generics (erasure)
- Primitive types (`int`)
- Class types (`foo.Bar`)

```
val x = "Foo"; x.charAt(1)

// Abstract Syntax Tree
Block(
  ValDef("x", Literal("Foo")),
  Apply(
    Select(Ident("x"), "charAt"),
    List(Literal(1))
  )
)
```

## Scala.js Compiler Output: The IR

### General

- AST form (typed)
- Complex expressions
- JavaScript operations

### Types

- No generics (erasure)
- Primitive types (`int`)
- Class types (`foo.Bar`)

### Classes / Interfaces

- Single class inheritance
- Multi interface inheritance
- No Overloading
  (instead: name mangling)
- JavaScript methods
  (aka Exports)

## Scala.js Compiler Output: The IR

### General

- AST form (typed)
- Complex expressions
- JavaScript operations

### Classes / Interfaces

- Single class inheritance
- Multi interface inheritance
- No Overloading
  (instead: name mangling)
- JavaScript methods
  (aka Exports)

### Types

- No generics (erasure)
- Primitive types (`int`)
- Class types (`foo.Bar`)

```
val result = {
  val helper = 1 + 2
  helper * 2
}

// VS

val helper = 1 + 2
val result = helper * 2
```

## Scala.js Compiler Output: The IR

### General

- AST form (typed)
- Complex expressions
- JavaScript operations

### Types

- No generics (erasure)
- Primitive types (`int`)
- Class types (`foo.Bar`)

### Classes / Interfaces

- Single class inheritance
- Multi interface inheritance
- No Overloading
  (instead: name mangling)
- JavaScript methods
  (aka Exports)

## Scala.js Compiler Output: The IR

### General

- AST form (typed)
- Complex expressions
- JavaScript operations

### Classes / Interfaces

- Single class inheritance
- Multi interface inheritance
- No Overloading
  (instead: name mangling)
- JavaScript methods
  (aka Exports)

### Types

- No generics (erasure)
- Primitive types (`int`)
- Class types (`foo.Bar`)

## Scala.js Compiler Output: The IR

### General

- AST form (typed)
- Complex expressions
- JavaScript operations

### Types

- No generics (erasure)
- Primitive types (`int`)
- Class types (`foo.Bar`)

### Classes / Interfaces

- Single class inheritance
- Multi interface inheritance
- No Overloading
  (instead: name mangling)
- JavaScript methods
  (aka Exports)

## Scala.js Compiler Output: The IR

### General

- AST form (typed)
- Complex expressions
- JavaScript operations

### Classes / Interfaces

- Single class inheritance
- Multi interface inheritance
- No Overloading
  (instead: name mangling)
- JavaScript methods
  (aka Exports)

### Types

- No generics (erasure)
- Primitive types (`int`)
- Class types (`foo.Bar`)

# Scala.js Compiler Output: The IR

## General

- AST form (typed)
- Complex expressions
- JavaScript operations

## Classes / Interfaces

- Single class inheritance
- Multi interface inheritance
- No Overloading (instead: name mangling)
- JavaScript methods (aka Exports)

## Types

- No generics (erasure)
- Primitive types (`int`)
- Class types (`foo.Bar`)

## Scala.js Compiler Output: The IR

### General

- AST form (typed)
- Complex expressions
- JavaScript operations

### Classes / Interfaces

- Single class inheritance
- Multi interface inheritance
- No Overloading
  (instead: name mangling)
- JavaScript methods
  (aka Exports)

### Types

- No generics (erasure)
- Primitive types (`int`)
- Class types (`foo.Bar`)

## Scala.js Compiler – `jscode`

### Calling JavaScript

#### Scala Source Code

```
def multiAlert(n: Int) =
  for (i <- 1 to n) dom.alert(msgs.hello(i))

object dom extends js.GlobalScope {
  def alert(message: String): Unit = js.native
}
```

#### Scala.js IR

```
def multiAlert__I__V(n: int) {
  // for (i <- 1 to n) {
    <global>["alert"](
      arg$outer.msgs__LHelloFactory().hello__I__T(i));
  // }
}
```

## Scala.js Compiler – jscode

Calling JavaScript

### Scala Source Code

```scala
def multiAlert(n: Int) =
  for (i <- 1 to n) dom.alert(msgs.hello(i))

object dom extends js.GlobalScope {
  def alert(message: String): Unit = js.native
}
```

### Scala.js IR

```
def multiAlert__I__V(n: int) {
  // for (i <- 1 to n) {
    <global>["alert"](
      arg$outer.msgs__LHelloFactory().hello__I__T(i));
  // }
}
```

## Scala.js Compiler – jscode

Calling JavaScript

### Scala Source Code

```scala
def multiAlert(n: Int) =
  for (i <- 1 to n) dom.alert(msgs.hello(i))

object dom extends js.GlobalScope {
  def alert(message: String): Unit = js.native
}
```

### Scala.js IR

```
def multiAlert__I__V(n: int) {
  // for (i <- 1 to n) {
    <global>["alert"](
      arg$outer.msgs__LHelloFactory().hello__I__T(i));
  // }
}
```

# Scala.js Compiler – jscode

## Method Exports

### Scala Code after jsinterop

```
def $js$exported$meth$multiAlert(n: Int): Any =
  multiAlert(n)
```

### Scala.js IR

```
def $$js$exported$meth$multiAlert__I__O(n: int): any = {
  this.multiAlert__I__V(n);
}

def "multiAlert"(arg0: any): any = {
  val prep0: int = arg0.asInstanceOf[I];
  this.$$js$exported$meth$multiAlert__I__O(prep0)
}
```

## Scala.js Compiler – `jscode`

Method Exports

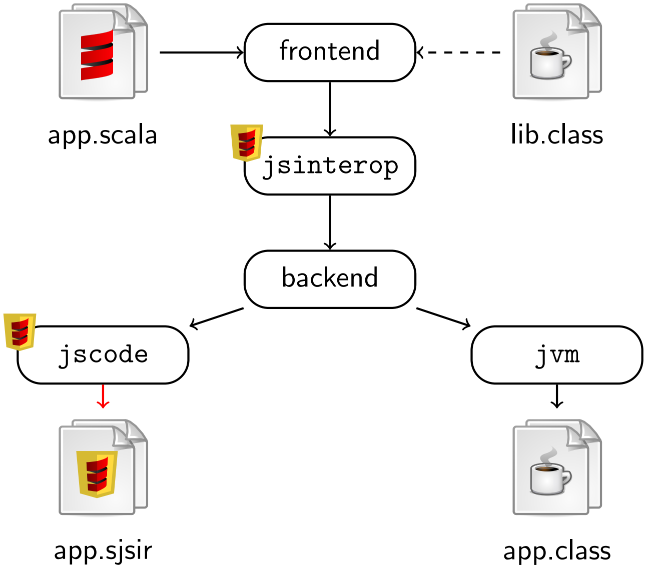
### Scala Code after `jsinterop`

```
def $js$exported$meth$multiAlert(n: Int): Any =
  multiAlert(n)
```

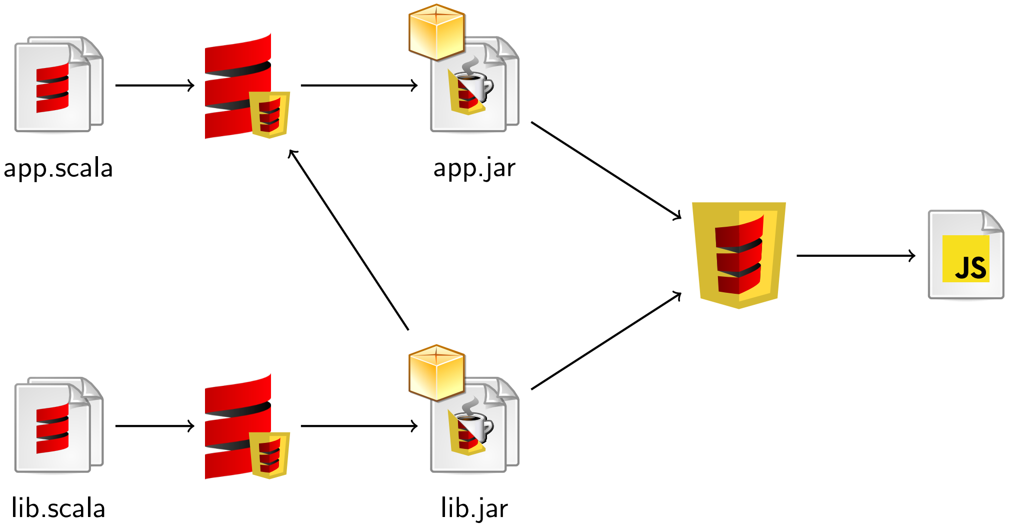
### Scala.js IR

```
def $$js$exported$meth$multiAlert__I__O(n: int): any = {
  this.multiAlert__I__V(n);
}

def "multiAlert"(arg0: any): any = {
  val prep0: int = arg0.asInstanceOf[I];
  this.$$js$exported$meth$multiAlert__I__O(prep0)
}
```

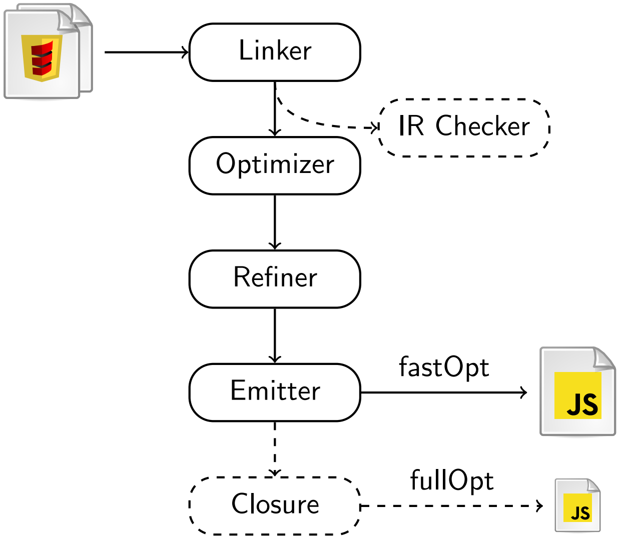
## Phases of the Scala.js Compiler



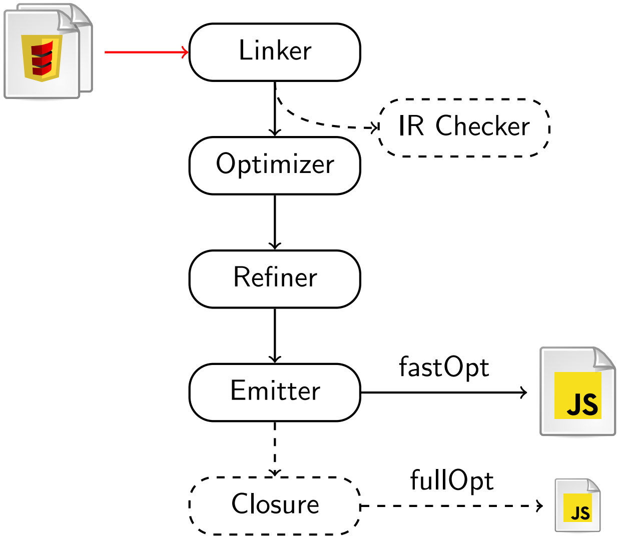
## Scala.js Pipeline



## Phases of the Scala.js Linker



## Phases of the Scala.js Linker



# Scala.js Linker – Linker Phase

## Alive Code Inclusion

```
@JSExport
class MultiAlerter {
  val msgs = new HelloFactory

  @JSExport
  def multiAlert(n: Int) =
    for (i <- 1 to n) dom.alert(msgs.hello(i))
}

class HelloFactory {
  def hello(x: Int) = s"Hello World #$x"
  def helloDebug() = "Hello World"
}
```

# Scala.js Linker – Linker Phase

## Alive Code Inclusion

```
@JSExport
class MultiAlerter {
  val msgs = new HelloFactory

  @JSExport
  def multiAlert(n: Int) =
    for (i <- 1 to n) dom.alert(msgs.hello(i))
}

class HelloFactory {
  def hello(x: Int) = s"Hello World #$x"
  def helloDebug() = "Hello World"
}
```

## Scala.js Linker – Linker Phase

### Alive Code Inclusion

```
@JSExport
class MultiAlerter {
  val msgs = new HelloFactory

  @JSExport
  def multiAlert(n: Int) =
    for (i <- 1 to n) dom.alert(msgs.hello(i))
}

class HelloFactory {
  def hello(x: Int) = s"Hello World #$x"
  def helloDebug() = "Hello World"
}
```

## Scala.js Linker – Linker Phase

### Alive Code Inclusion

```
@JSExport
class MultiAlerter {
  val msgs = new HelloFactory

  @JSExport
  def multiAlert(n: Int) =
    for (i <- 1 to n) dom.alert(msgs.hello(i))
}

class HelloFactory {
  def hello(x: Int) = s"Hello World #$x"
  def helloDebug() = "Hello World"
}
```

## Scala.js Linker – Linker Phase

### Alive Code Inclusion

```scala
@JSExport
class MultiAlerter {
  val msgs = new HelloFactory

  @JSExport
  def multiAlert(n: Int) =
    for (i <- 1 to n) dom.alert(msgs.hello(i))
}

class HelloFactory {
  def hello(x: Int) = s"Hello World #$x"
  def helloDebug() = "Hello World"
}
```

## Scala.js Linker – Linker Phase

### Alive Code Inclusion

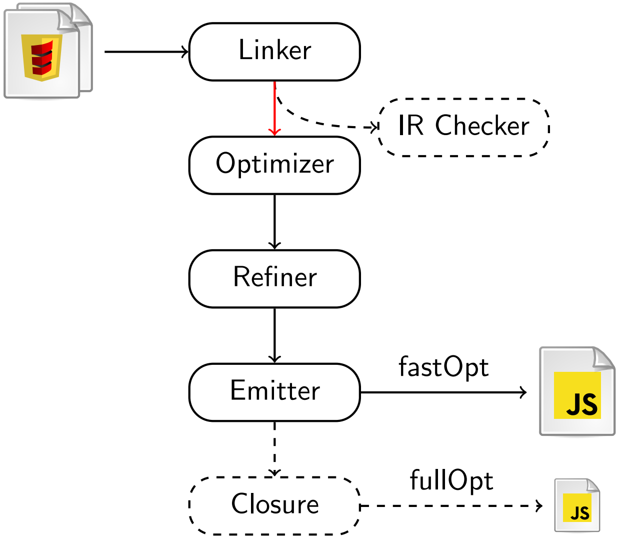
```scala
@JSExport
class MultiAlerter {
  val msgs = new HelloFactory

  @JSExport
  def multiAlert(n: Int) =
    for (i <- 1 to n) dom.alert(msgs.hello(i))
}

class HelloFactory {
  def hello(x: Int) = s"Hello World #$x"
  def helloDebug() = "Hello World"
}
```

## Scala.js Linker – Linker Phase

### Alive Code Inclusion

```scala
@JSExport
class MultiAlerter {
  val msgs = new HelloFactory

  @JSExport
  def multiAlert(n: Int) =
    for (i <- 1 to n) dom.alert(msgs.hello(i))
}

class HelloFactory {
  def hello(x: Int) = s"Hello World #$x"
  def helloDebug() = "Hello World"
}
```

## Phases of the Scala.js Linker



## Scala.js Linker – Optimizer Phase

```
def multiAlert__I__V(n: int) {
  // for (i <- 1 to n) {
    <global>["alert"](
      arg$outer.msgs__LHelloFactory().hello__I__T(i));
  // }
}
```

```
def multiAlert__I__V(n: int) {
  var i: int = 0
  while (i <=[int] n) {
    <global>["alert"](this.msgs$1.hello__I__T(i));
    i = i +[int] 1;
  }
}
```

## Scala.js Linker – Optimizer Phase

```
def multiAlert__I__V(n: int) {
  mod:sr_RichInt$.to$extension0__I__I__sci_Range$Inclusive(
    mod:s_Predef$.intWrapper__I__I(1), n).foreach$mVc$sp__F1__V(
      new sjsr_AnonFunction1().init___sjs_js_Function1(
      (lambda<this>(arg$outer: LMultiAlerter, i$2: any) = {
        val i: int = i$2.asInstanceOf[I];
        <global>["alert"](
          arg$outer.msgs__LHelloFactory().hello__I__T(i)
        );
        (void 0)
      }
  )))
}
```

## Scala.js Linker – Optimizer Phase

```
def multiAlert__I__V(n: int) {
  // for (i <- 1 to n) {
    <global>["alert"](
      arg$outer.msgs__LHelloFactory().hello__I__T(i));
  // }
}
```

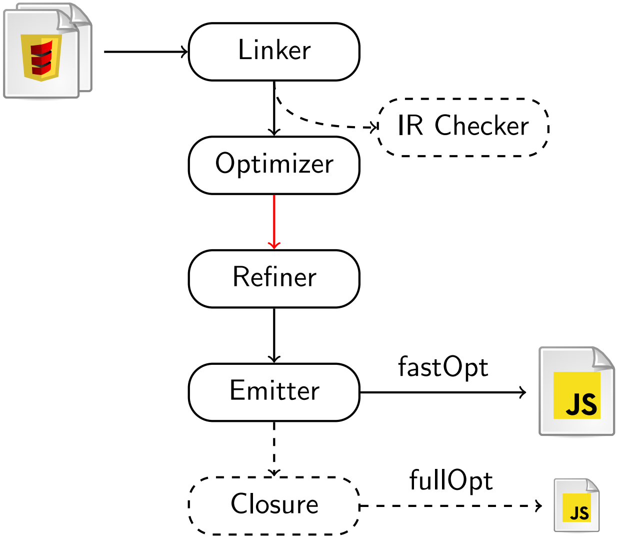
```
def multiAlert__I__V(n: int) {
  var i: int = 0
  while (i <=[int] n) {
    <global>["alert"](this.msgs$1.hello__I__T(i));
    i = i +[int] 1;
  }
}
```

## Scala.js Linker – Optimizer Phase

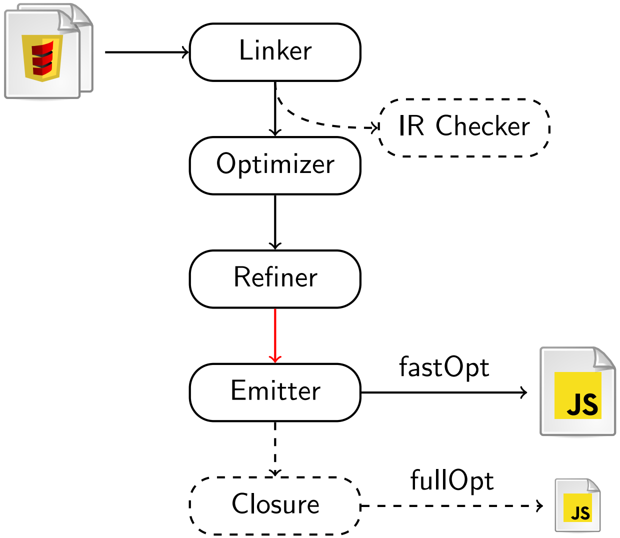
```
def multiAlert__I__V(n: int) {
  // for (i <- 1 to n) {
    <global>["alert"](
      arg$outer.msgs__LHelloFactory().hello__I__T(i));
  // }
}
```

```
def multiAlert__I__V(n: int) {
  var i: int = 0
  while (i <=[int] n) {
    <global>["alert"](this.msgs$1.hello__I__T(i));
    i = i +[int] 1;
  }
}
```

## Phases of the Scala.js Linker



## Phases of the Scala.js Linker



## Scala.js Compiler Output: The IR

### General

- AST form (typed)
- Complex expressions
- JavaScript operations

### Classes / Interfaces

- Single class inheritance
- Multi interface inheritance
- No Overloading
  (instead: name mangling)
- JavaScript methods
  (aka Exports)

### Types

- No generics (erasure)
- Primitive types (`int`)
- Class types (`foo.Bar`)

## Scala.js Compiler Output: The IR

### General

- AST form (typed)
- Complex expressions
- JavaScript operations

### Classes / Interfaces

- Single class inheritance
- Multi interface inheritance
- No Overloading
  (instead: name mangling)
- JavaScript methods
  (aka Exports)

### Types

- No generics (erasure)
- Primitive types (`int`)
- Class types (`foo.Bar`)

## Scala.js Linker – Emitter Phase

Desugaring

### Scala Code

```
def norm(a: Int, b: String) = {
  val a2 = a * a
  val b2 = {
    val b0 = b.toInt
    b0 * b0
  }
  math.sqrt(a2 + b2)
}
```

### Pseudo JavaScript Code

```
function norm(a, b) {
  return {
    var a2 = a * a;
    var b2 = {
      var b0 = parseInt(b);
      b0 * b0;
    };
    Math.sqrt(a2 + b2);
  };
}
```

## Scala.js Linker – Emitter Phase

Desugaring

### Scala Code

```scala
def norm(a: Int, b: String) = {
  val a2 = a * a
  val b2 = {
    val b0 = b.toInt
    b0 * b0
  }
  math.sqrt(a2 + b2)
}
```

### Pseudo JavaScript Code

```javascript
function norm(a, b) {
  return {
    var a2 = a * a;
    var b2 = {
      var b0 = parseInt(b);
      b0 * b0;
    };
    Math.sqrt(a2 + b2);
  };
}
```

## Scala.js Linker – Emitter Phase

Desugaring

### Scala Code

```scala
def norm(a: Int, b: String) = {
  val a2 = a * a
  val b2 = {
    val b0 = b.toInt
    b0 * b0
  }
  math.sqrt(a2 + b2)
}
```

### Pseudo JavaScript Code

```javascript
function norm(a, b) {
  {
    var a2 = a * a;
    var b2 = {
      var b0 = parseInt(b);
      b0 * b0;
    };
    return Math.sqrt(a2 + b2);
  };
}
```

## Scala.js Linker – Emitter Phase

Desugaring

### Scala Code

```scala
def norm(a: Int, b: String) = {
  val a2 = a * a
  val b2 = {
    val b0 = b.toInt
    b0 * b0
  }
  math.sqrt(a2 + b2)
}
```

### Pseudo JavaScript Code

```javascript
function norm(a, b) {
  {
    var a2 = a * a;
    {
      var b0 = parseInt(b);
      var b2 = b0 * b0;
    };
    return Math.sqrt(a2 + b2);
  };
}
```

## Scala.js Linker – Emitter Phase

Desugaring

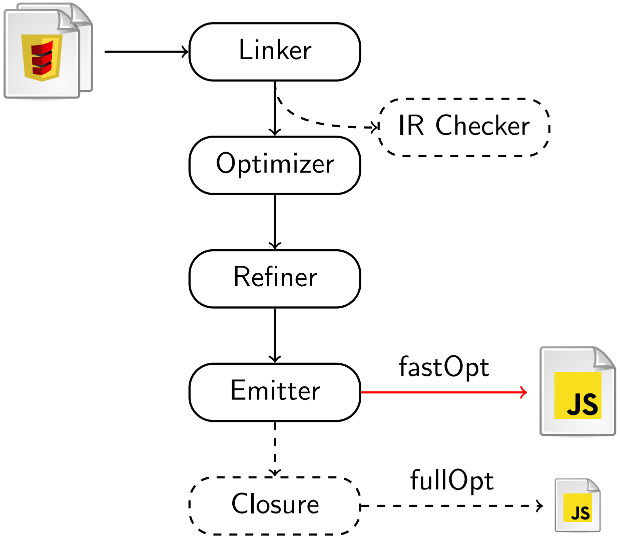
### Scala Code

```scala
def norm(a: Int, b: String) = {
  val a2 = a * a
  val b2 = {
    val b0 = b.toInt
    b0 * b0
  }
  math.sqrt(a2 + b2)
}
```

### JavaScript Code

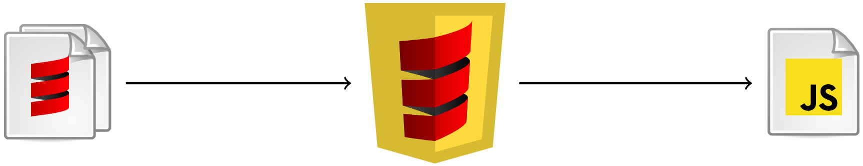
```javascript
function norm(a, b) {
  {
    var a2 = a * a;
    {
      var b0 = parseInt(b);
      var b2 = b0 * b0;
    };
    return Math.sqrt(a2 + b2);
  };
}
```

## Phases of the Scala.js Linker



## Final JavaScript (simplified)

```
/** @constructor */
var MultiAlerter = function() {
  this.msgs$1 = new HelloFactory()
};
MultiAlerter.prototype.multiAlert__I__V = function(n) {
  var i = 0;
  while (i <= n) {
    alert(this.msgs$1.hello__I__T(i));
    i = i + 1;
  }
};

/** @constructor */
var HelloFactory = function() {};
HelloFactory.prototype.hello__I__T = function(x) {
  return new StringContext(/* snip */).s(/* snip */)
};
```


JS

## Things I Shamelessly Omitted

### Scala.js IR

- Hijacked Classes
- Additional Types
  - `String`
  - `Undefined`, `Null`, `Nothing`, `NoType`
  - Array types (`int[]`, `A[]`)
  - Record types
- Labeled Blocks
  - Pattern Matches
  - Tailrec Methods
- Modules (objects)

### Compiler

- `scala.Enumeration`
- Reflective Calls
- Function literals
- Export overloading

### Linker

- Instance tests
- Longs
- Inheritance in JavaScript
- Semantics / Output modes

## Questions

¿Questions?

Tobias Schlatter – @gzm0 on GitHub / Gitter
`https://gitter.im/scala-js/scala-js`