Evolution Gaming

Scala.js in a modern web application.

Vladimir Pavkin

Riga Dev Days 2017

About the speaker

Evolution Gaming

10 YEARS ALIVE

Vladimir Pavkin

Scala Developer at Evolution Gaming

- JavaScript since 2012
- Scala & Scala.js since 2014

https://github.com/vpavkin

http://pavkin.ru

Agenda

I. Brief introduction into Scala.js (5 min)
II. How we use Scala.js in Evolution Gaming (5 min)
III. Examples, **pros** and **cons**, based on our real experience (30 min)
IV. Conclusion, Q&A (5 min)

I. What is Scala.js ?

What Scala.js is not:

- a framework for web apps like Angular, React, GWT, etc.
- a new language

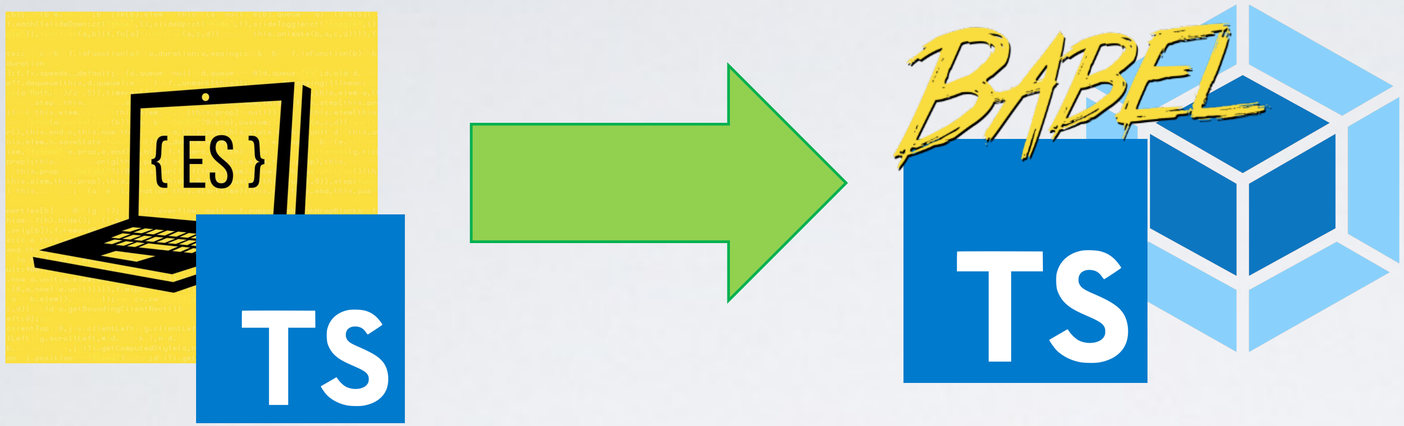
What Scala.js is not:

- a framework for web apps like Angular, React, GWT, etc.
- a new language

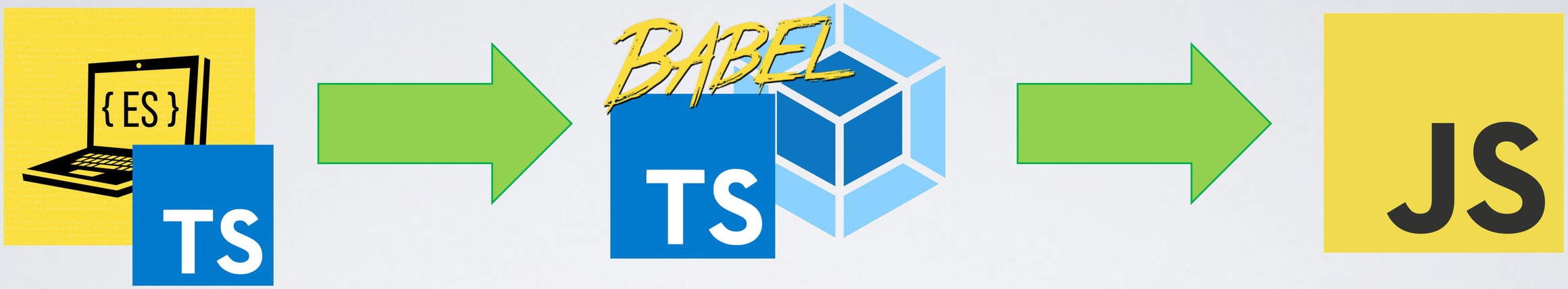
Scala.js is a transpiler from Scala to JS.

Is it really different?

Is it really different?

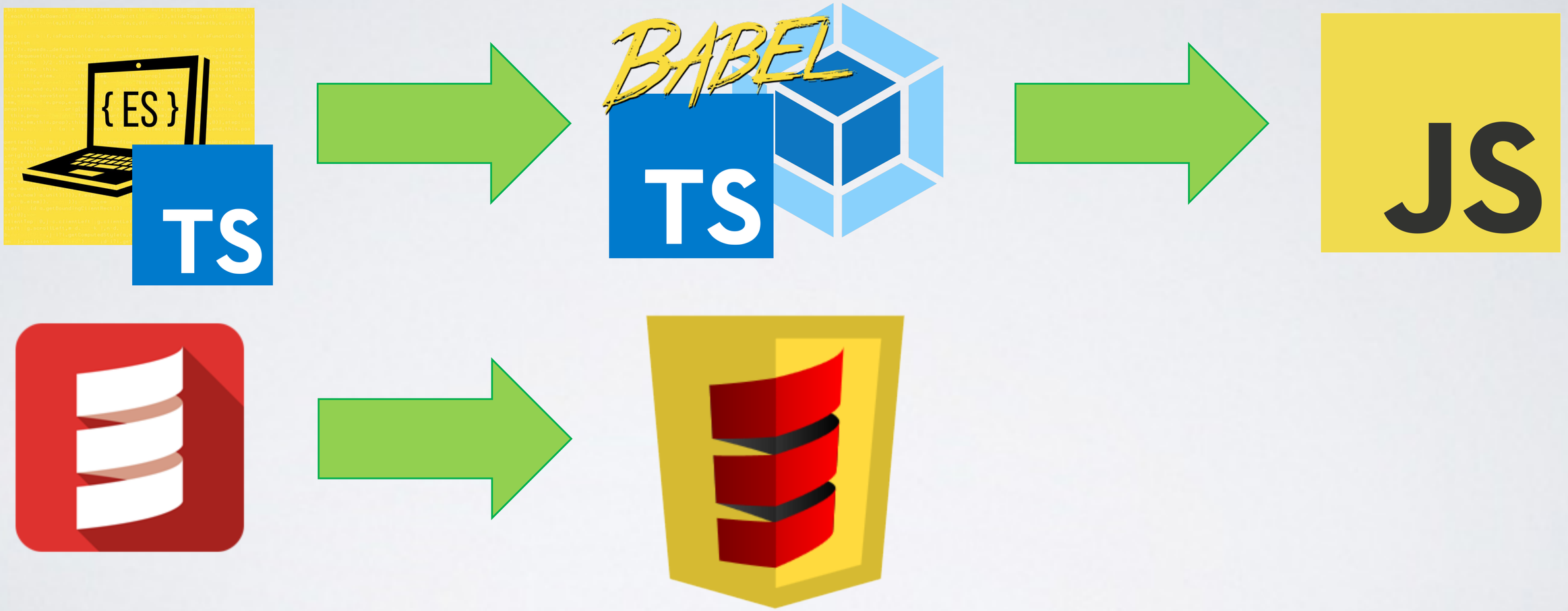


Is it really different?

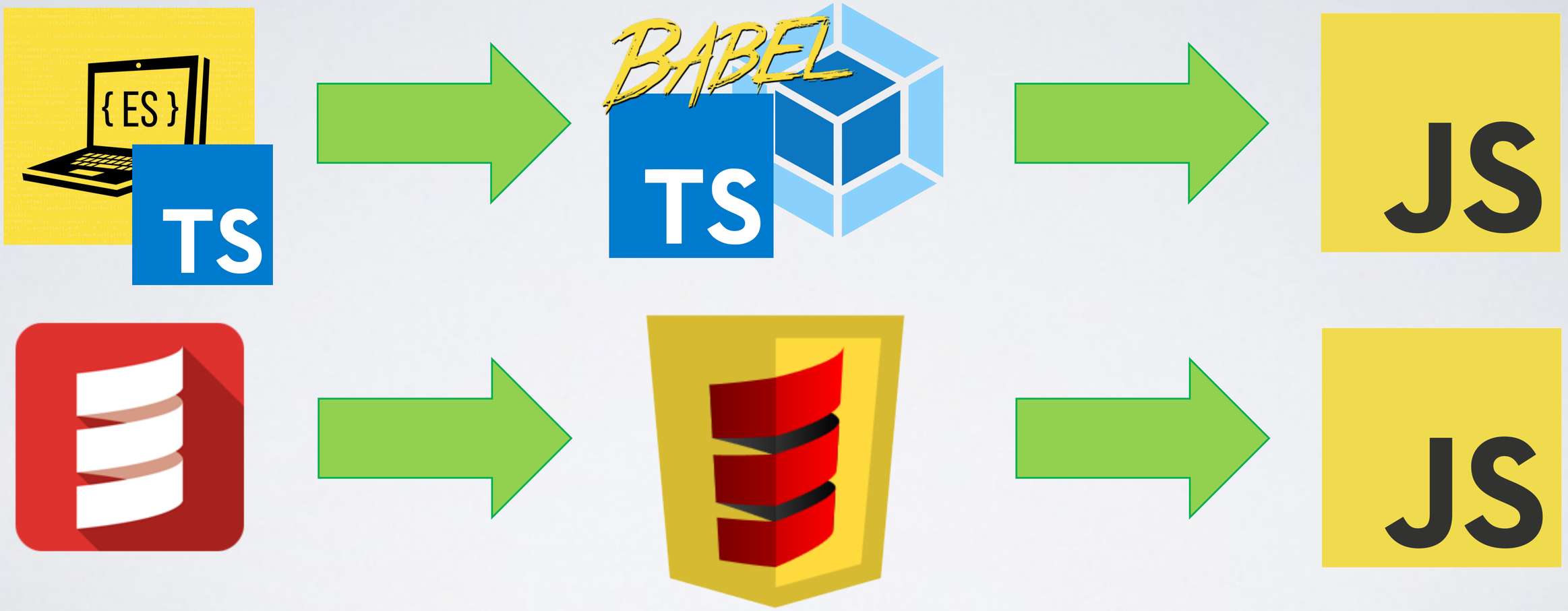


Is it really different?

Is it really different?



Is it really different?



Is it statically typed?

Is it statically typed?

Yes.

It supports 100% of Scala language and almost the entire standard library.

You get one of the most powerful type systems in the world.

Can I do everything I can do in JS?

Can I do everything I can do in JS?

Yes.

- All standard ECMAScript APIs are provided
- You can write dynamic code that will compile directly to native JS (but why?)
- You can (and probably will) use existing JS libraries

Is it production ready?

Is it production ready?

Yes.

- 1.0.0 is coming this year, though 0.6.16 is production ready
- Very good performance: https://www.scala-js.org/doc/internals/performance.html
  - Most of the times has the same performance as native JavaScript
  - Up to 2 times slower in worst cases
- Small runtime size: 100Kb minified
- Supported by all major Scala libraries

II. Scala.js in 

Scala.js in Evolution Gaming

Projects:

- Currently we have 2 internal projects, written entirely in Scala.js:
  - Big employee scheduling application (desktop only)
  - Mobile-first web app
- We plan to do more Scala.js development for new projects

Evolution Gaming 10 YEARS ALIVE

Scala.js in Evolution Gaming

Showcase:

- SPA
- Thousands of scheduled employees
- Complex and heavy UI

Position Requirements | Hide Allocation Control | ← | [redacted] | → | 05.2017 | Fluid since: 05.05.2017

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Morning 07 - 15	✓	8	8	✓	8	8	✓	✓	8	8	8	8	✓	✓	8	8	8	8	✓	✓	8	8	8	8	✓	✓	8	8	8	8	✓
Afternoon 15 - 23	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Night 23 - 07	8	8	✓	✓	8	8	8	8	✓	✓	8	8	8	8	✓	✓	8	8	8	8	✓	✓	8	8	8	8	✓	✓	8	8	8

Employee	I/O	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1. Afu Popova	277.34	07-15	07-15	07..23	07..23	15-23	15-23	07-15	07-15	07..23	07..23	15-23	15-23	07-15	07-15	07..23	07..23	15-23	15-23	07-15	07-15	07..23	07..23	15-23	15-23	07-15	07-15	07..23	07..23	15-23	15-23	07-15
2. Akp Motruk	344.66																															
3. Alb Yecoshenko	218.66		07-15	07-15	07-15																											
4. Ang Pymonenko	338.66			07-15																												
5. Bas Honchar	18.66	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15
6. Bel Gurka	346.66																															
7. Bil Babenko	26.66			07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15	
8. Bir Khomenko	18.66	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15
9. Cel Lyakh	18.66	15-23	15-23	15-23	15-23			15-23	15-23	15-23	15-23			15-23	15-23	15-23	15-23			15-23	15-23	15-23	15-23			15-23	15-23	15-23	15-23			15-23
10. Cip Kurylenko	346.66																															
11. Cir Kovalenko	18.66	15-23	15-23			15-23	15-23	15-23	15-23			15-23	15-23	15-23	15-23			15-23	15-23	15-23	15-23			15-23	15-23	15-23	15-23			15-23	15-23	15-23
12. Cos Herasymenko	26.66	07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15			07-15	07-15	07-15	07-15			07-15
13. Dog Dzubenko	18.66	15-23	15-23	15-23	15-23			15-23	15-23	15-23	15-23			15-23	1											15-23	15-23	15-23	15-23			15-23
14. Hys Romanenko	26.66	15-23	15-23			15-23	15-23	15-23	15-23			15-23	15-23	15-23	1											15-23	15-23			15-23	15-23	15-23
15. Kar Palamarchuk	26.66			15-23	15-23	15-23	15-23			15-23	15-23	15-23	15-23															15-23	15-23	15-23	15-23	
16. Kil Doroshenko	18.66	23-07	23-07	23-07	23-07			23-07	23-07	23-07	23-07			23-07	2											23-07	23-07	23-07	23-07			23-07
17. Kiz Lutsenko	15.66	23-07	23-07			23-07	23-07	23-07	23-07			23-07	23-07	23-07	2											23-07	23-07			23-07	23-07	23-07
18. Kra Miroshnychenko	26.66			23-07	23-07	23-07	23-07			23-07	23-07	23-07	23-07															23-07	23-07	23-07	23-07	
19. Kur Tsvyk	346.66																															
20. Okt Tsipkin	82.66	23-07	23-07	23-07	23-07			23-07	23-07	23-07	23-07			23-07	2											23-07	23-07	23-07	23-07			23-07
21. Ruz Fop	26.66			23-07	23-07	23-07	23-07			23-07	23-07	23-07	23-07															23-07	23-07	23-07	23-07	
22. Sag Motruk	346.66																															
23. Sen Yehorenko	340.95																															
24. She Mishchenko	346.66																															
25. Smi Pylypenko	338.66																															

Hys Romanenko, 13.05.2017

15:00-23:00 **Work** *Dealer* Edit Remove

One-click add shift:

Morning | Afternoon | Night | Morning 12 | Night 12 | Custom

Role: Dealer

Add absence

Position Requirements | Monthly Schedule | ← | [redacted] | → | Wednesday, 03.05.2017 | Morning

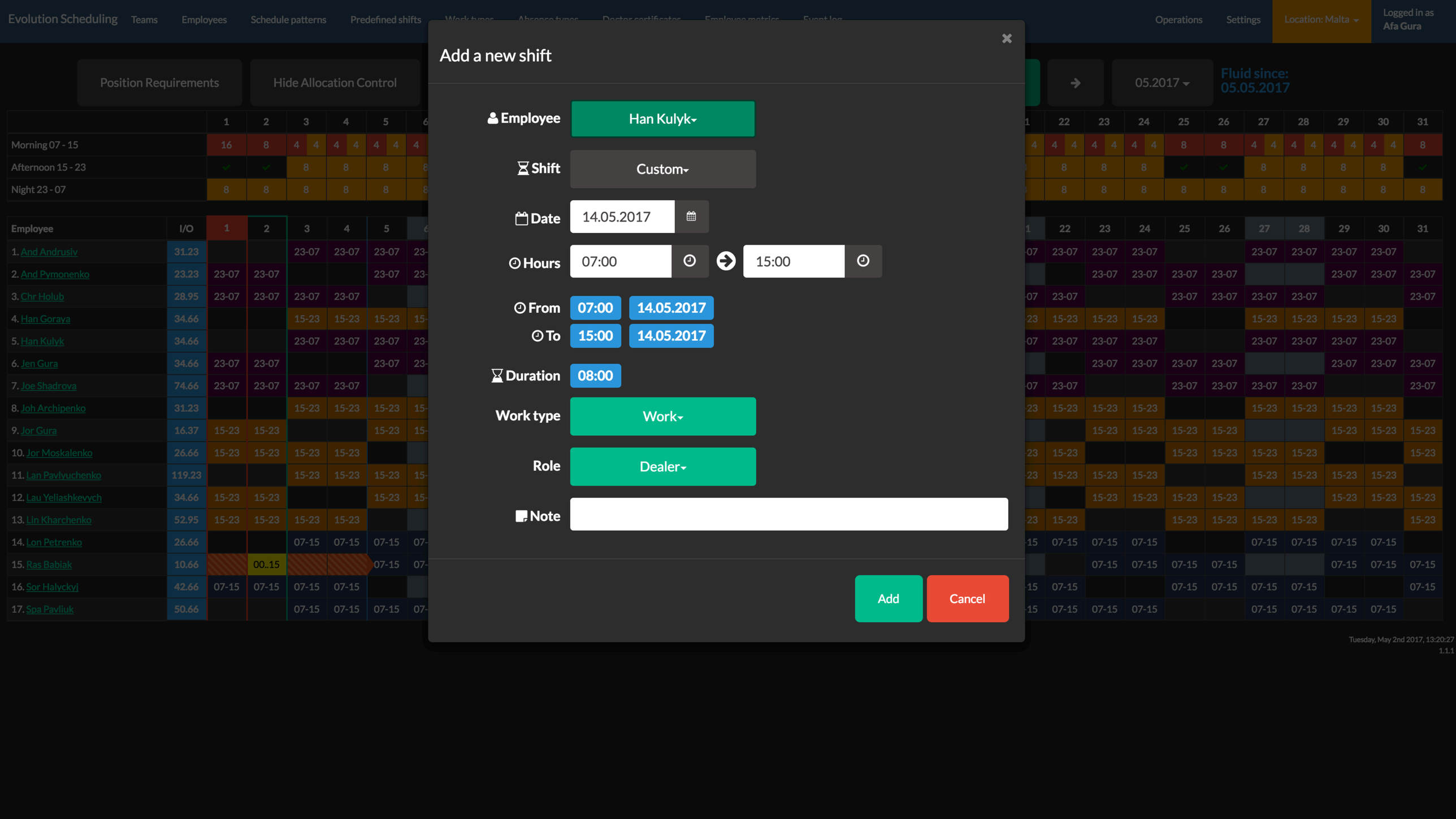
Position	Uncovered	23:00	00:00	01:00	02:00	03:00	04:00	05:00	06:00	07:00	08:00	09:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00	21:00	22:00
Overallocation		✓	✓	✓	✓	2	2	2	2	1	1	1	1	✓	✓	✓	✓	1	1	1	1	1	1	1	1
Dealer 1	✓	23 - 07, Chr Holub								07 - 15, Lon Petrenko								15 - 23, Joh Archipenko							
Dealer 2	✓	23 - 07, And Pymonenko								07 - 15, Sor Halyckyj								15 - 23, Lan Pavlyuchenko							
Dealer 3	✓	23 - 03, Joe Shadrova												11 - 15, Spa Pavliuk				15 - 23, Han Goraya							
Dealer 4	4	23 - 03, Jen Gura												11 - 15				15 - 23, Lin Kharchenko							

Overallocated work

-	8																	15 - 23, Jor Moskalenko							
-	4					03 - 07, Joe Shadrova																			
-	4					03 - 07, Jen Gura																			
-	4									07 - 11, Spa Pavliuk															

Absent

Employee	Absence type	Period	Note
Ras Babiak	Sick	14.04.2017 — +∞	
Joe Shadrova	MARRIAGE	13.04.2017 — 16.04.2017	
Jen Gura	INJURY	14.04.2017 — 16.04.2017	
Lon Petrenko	UNJUSTIFIED	14.04.2017 — 14.04.2017	
Joh Archipenko	NOTPAYED	14.04.2017 — 30.04.2017	
Jor Moskalenko	Paid Vacation	14.04.2017 — 28.04.2017	
And Pymonenko	ANNUAL_PAID	14.04.2017 — 30.04.2017	
Chr Holub	ANNUAL_UNPAID	14.04.2017 — 29.04.2017	
And Andrusiv	ACADEMIC	14.04.2017 — 30.04.2017	
Han Kulyk	BIRTH	14.04.2017 — 14.04.2017	
Sor Halyckyj	Sick	14.04.2017 — 20.04.2017	
Jor Gura	OTHER_PAID	14.04.2017 — 27.04.2017	
Lin Kharchenko	SICK_UNPAID	14.04.2017 — 22.04.2017	
Lau Yeliashkevych	Sick	14.04.2017 — 18.04.2017	
Han Goraya	BEREAVMENT	14.04.2017 — 15.04.2017	
Lan Pavlyuchenko	PREGNANCY	14.04.2017 — 30.04.2017	


Evolution Scheduling
Teams
Employees
Schedule patterns
Predefined shifts
Operations
Settings
Location: Malta
Logged in as
Afa Gura
Position Requirements
Hide Allocation Control
05.2017
Fluid since:
05.05.2017
Add a new shift
Employee
Han Kulyk
Shift
Custom
Date
14.05.2017
Hours
07:00
15:00
From
07:00
14.05.2017
To
15:00
14.05.2017
Duration
08:00
Work type
Work
Role
Dealer
Note
Add
Cancel
Tuesday, May 2nd 2017, 13:20:27
1.1.1

Scala.js in Evolution Gaming

Front-end Stack:

- **Scala.js React**. Scala bindings for ReactJS
- **Diode**. Redux-like state management (close to Elm)
- **ScalaCSS**. Type safe embedded DSL for CSS
- Native JS: **Bootstrap** + **jQuery**, **moment-js**

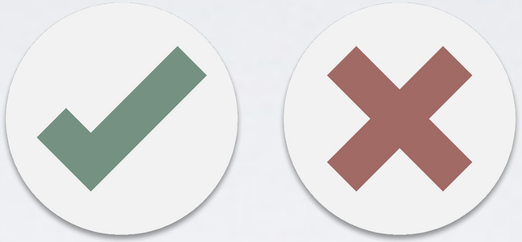
Other, not front-end-specific libraries:

- Cats, shapeless, circe, monix, spire, scalatest, scalacheck, ...

Scala.js in Evolution Gaming

Random facts:

- ~1000 Scala files, < 0.1% JS code (our own code in LOC)
- Production minified JS file size: **3.2 Mb**, **+850Kb** JS dependencies (before gzip)
- Compile time: **4.5 minutes**
- JS optimization + emission time (CI build only): **2.5 minutes**
- "Hot reload" time: **10-15 seconds**



III. Our experience

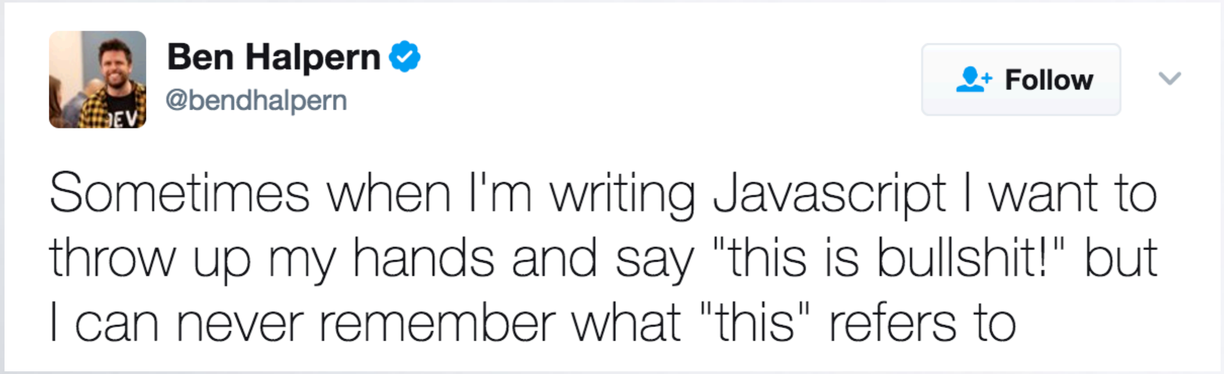
Hard things

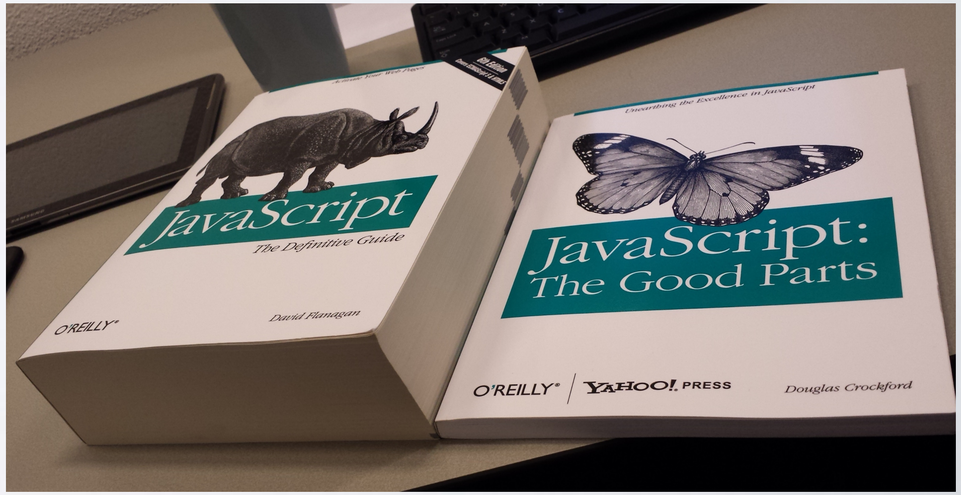
You have to know both

- Scala.js is just a transpiler, all design choices are up to you
- You'll have to:
  - Know how frontend works in general
  - Be aware of current best practices and trends (react, redux, SPA, etc.)
  - Have a basic understanding of how HTML, CSS and JS work
  - Know how ReactJS works if you're going to use scalajs-react.
  - Know Scala

You have to know both

- Scala.js is just a transpiler, all design choices are up to you
- You'll have to:
  - Know how frontend works in general
  - Be aware of current best practices and trends (react, redux, SPA, etc.)
  - Have a basic understanding of how HTML, CSS and JS work
  - Know how ReactJS works if you're going to use scalajs-react.
  - Know Scala
- Luckily, you're guarded from all the tricky JS stuff





Output file size

- Scala.js runtime itself is quite small, but…
- Really joyful Scala development requires libraries, they add up a bit
- Optimizer aggressive inlining does not help
- Not a big problem for desktop apps, mobile is another question

Optimizer slowdowns

- In general, optimizer does an awesome job
- As usual, you can't make everyone happy: specific rare cases can lead to exponential output file growth and optimization slowdowns
- Our case was fixed in 2 hours by just adding one `@noinline` annotation in Diode 1.1.0
- General advice: keep inlining in mind in case of drastic optimization/emission slowdown

More on our case:
https://gitter.im/scala-js/scala-js?at=5820353445c9e3eb4308d2e3

Reload is not so hot

- React hot reload is awesome
- Unfortunately, incremental compilation + optimization + emission take time
- Lots of natural inherent restrictions to implement hot reload, comparable to ReactJS one
- Play Framework live reload gives some fresh air
- There's a limited experimental hot reload at lihaoyi/workbench

Unexpected semantics

Unexpected semantics

```
val a: Float = 5.71F

println(a.toString)
```

Unexpected semantics


```
val a: Float = 5.71F

println(a.toString) // 5.710000038146973
```

Evolution Gaming 10 YEARS ALIVE

Unexpected semantics

```
val a: Float = 5.71F

println(a.toString) // 5.710000038146973
```

The more you know. Semantics of Scala.js:
https://www.scala-js.org/doc/semantics.html

Coverage support

- Scoverage claims to support Scala.js
- In reality, any code, involving dynamic JS calls, doesn't even compile
- There are several other issues, like incorrect environment detection

See #176 and #183 on scoverage/scalac-scoverage-plugin.

Sometimes IDEA just gives up

- Overall IDE support is wonderful, esp. comparing to JS
- When project becomes big, some big files can kill typechecker
- In some cases even highlighting refuses to work
- Minor issues with cross-project support

Good things

Code sharing

- Just **insanely useful**
- Shared core domain business logic
- Shared API protocols and Data Transfer Objects (DTO)
- Rich client and various “optimistic” scenarios **for free**
- Automatic 100% safe API synchronization **for free**
- No communication overhead on API development/synchronization
- You can even write and reuse abstract cross-platform code for things, that don’t have unified cross-platform APIs (e.g. with type classes)

Code sharing (example)

Shared DTO and codecs

```
case class Shift(
  start: LocalTime,
  end: LocalTime)

case class Schedule(
  employeeId: UUID,
  date: LocalDate,
  shifts: List[Shift])
```

```
object Codecs {

  implicit lazy val decoderLocalDate: Decoder[LocalDate] = ???
  implicit lazy val encoderLocalDate: Encoder[LocalDate] = ???

  implicit lazy val decoderLocalTime: Decoder[LocalTime] = ???
  implicit lazy val encoderLocalTime: Encoder[LocalTime] = ???

  implicit lazy val shiftEncoder = Encoder[Shift]
  implicit lazy val shiftDecoder = Decoder[Shift]

  implicit lazy val scheduleEncoder = Encoder[Schedule]
  implicit lazy val scheduleDecoder = Decoder[Schedule]
}
```

Evolution Gaming 10 YEARS ALIVE

Code sharing (example)

Back-end controller

```
def schedule = Action { req =>
  val schedule: Schedule = ???
  Ok(schedule.asJson)
}
```

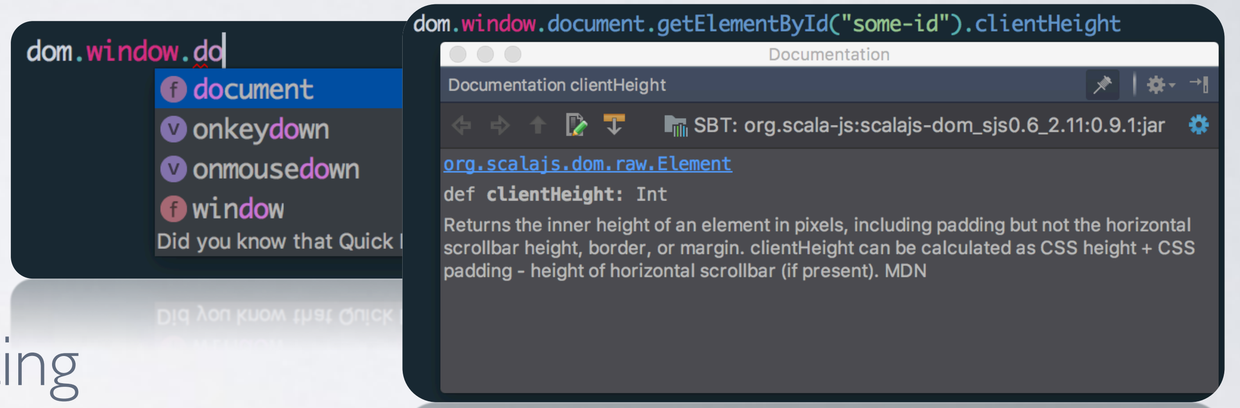
Front-end

```
def getSchedule: Future[Either[Error, Schedule]] =
  Ajax.get("http://my.service.com/schedule")
    .map(response =>
      decode[Schedule](response.responseText)
    )
```

Evolution Gaming 10 YEARS ALIVE

IDE and tooling

- All that works for Scala, works for Scala.js **out of the box**:
  - Autocomplete
  - Code navigation
  - Rich refactoring and code-generation
  - Syntax, error and code-smells highlighting



- Including various browser APIs (like DOM) and native JS library facades
- Much more **reliable** than WebStorm and alike, due to static type system
- No special plugin or IDE is required

Type safe markup

```
val items: List[String] = ???
div(
  h1(items.length, " items found:"),
  ol(
    items.map(li(_)).toTagMod
  )
)
```

Rich type system

- **Imagine the possibilities!**
- **All** Scala features are available: implicits, macros & scala-meta, type classes, higher-kinded types, etc...
- Maximum power of expression and design thought

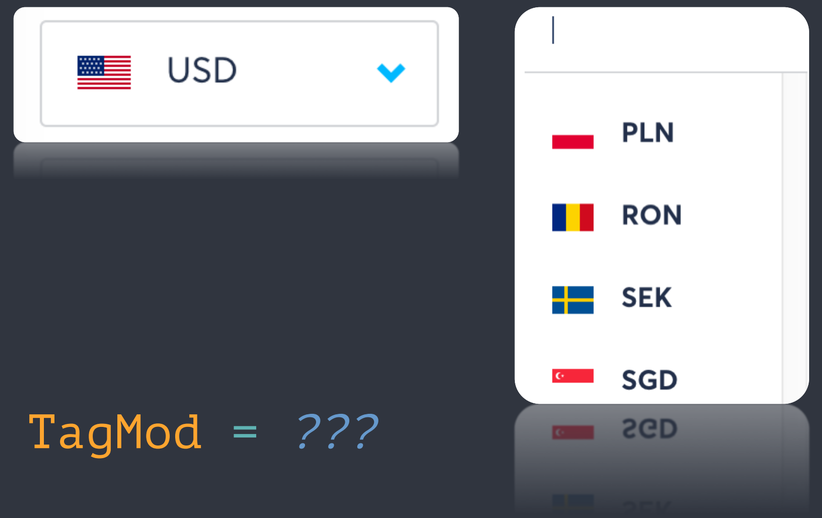
Rich type system (example)

```
class GenericSelector {

  case class Props(
    selected: String,
    onChange: String => Callback,
    all: List[String])

  def renderElement(props: Props)(element: String): TagMod = ???

  def render(props: Props) = div(props.all.map(renderElement(props)).toTagMod)
}
```



Rich type system (example)

```
class GenericSelector {

  case class Props(
    selected: String,
    onChange: String => Callback,
    all: List[String])

  def renderElement(props: Props)(element: String): TagMod = ???

  def render(props: Props) = div(props.all.map(renderElement(props)).toTagMod)
}
```

Rich type system (example)

```
class GenericSelector[T] {

  case class Props(
    selected: T,
    onChange: T => Callback,
    all: List[T])

  // how to get id for the element?
  // how to render element to html? Don't say .toString
  def renderElement(props: Props)(element: T): TagMod = ???

  def render(props: Props) = div(props.all.map(renderElement(props)).toTagMod)
}
```

Rich type system (example)

```
// proves that instances of type T have a human readable name
@typeclass trait Named[T] {
  def name(t: T): String
}

// Proves that instance of type T can be uniquely identified among other T's by a string.
@typeclass trait Identified[T] {
  def uid(t: T): String
}
```

Evolution Gaming 10 YEARS ALIVE

Rich type system (example)

```
class GenericSelector[T: Named : Identified] {

  case class Props(
    selected: T,
    onChange: T => Callback,
    all: List[T])

  def renderElement(props: Props)(element: T): TagMod = ???

  def render(props: Props) = div(props.all.map(renderElement(props)).toTagMod)
}
```

Evolution Gaming 10 YEARS ALIVE

Rich type system (example)

```
class GenericSelector[F[_]: Applicative : Foldable, T: Named : Identified] {

  case class Props(
    selected: F[T],
    onChange: F[T] => Callback,
    all: List[T])

  def renderElement(props: Props)(element: T): TagMod = ???

  def render(props: Props) = div(props.all.map(renderElement(props)).toTagMod)
}
```

Rich type system (example)

```
val strictCurrencySelector = new GenericSelector[cats.Id, Currency] { /* ... */ }

val optionalCurrencySelector = new GenericSelector[Option, Currency] { /* ... */ }

val currencyMultiSelect = new GenericSelector[List, Currency] { /* ... */ }
```

Rich type system (example)

```
val strictCurrencySelector = new GenericSelector[cats.Id, Currency] { /* ... */ }

val optionalCurrencySelector = new GenericSelector[Option, Currency] { /* ... */ }

val currencyMultiSelect = new GenericSelector[List, Currency] { /* ... */ }
```

- Yes, this kind of design power is available today in the browser
- 100% type safe, checked by compiler
- No monkey-patching, dynamic casting or other nightmare

You don’t need JS developers

You don’t need JS developers

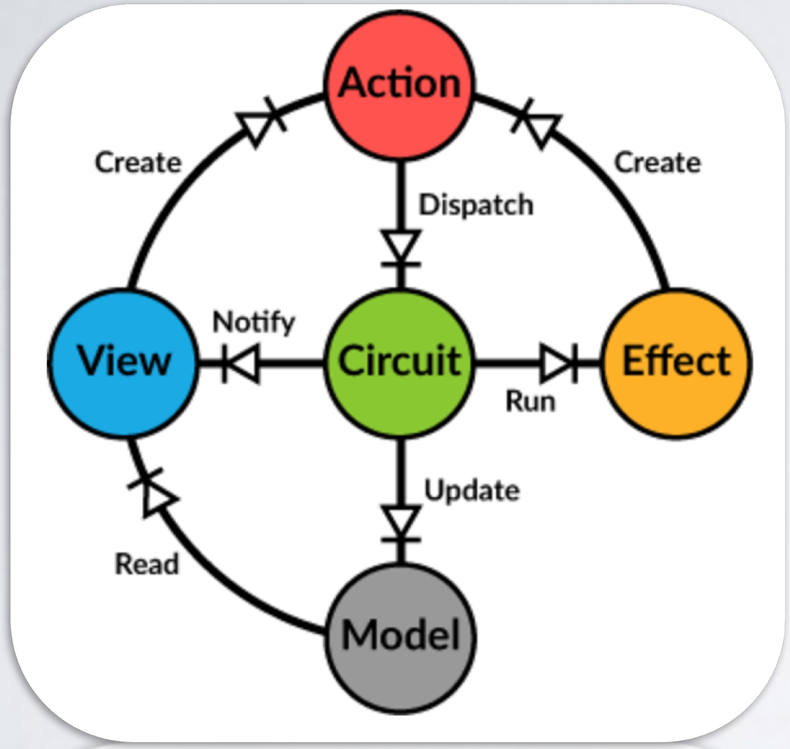
You don’t need JS developers

- Can be a great advantage for internal systems, where it’s too expensive to have a dedicated JS developer
- Scala developers own the whole codebase, eliminating cross-team (or cross-language) synchronization overhead

Seamless JS interop

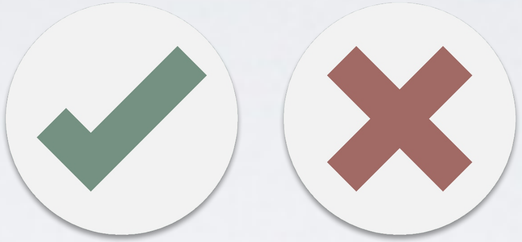
- **Very important!**
- Scala.js interop is very simple and natural in both directions
- You can easily use libraries from JS ecosystem (with additional type safety)
- You can write JS libraries in Scala and publish them for JS devs to use
- In you application code you can freely mix pure Scala and JS façade code
- Dealing with JS APIs from Scala.js is more efficient (IDE FTW)

Diode state management



- Scala type system provides a more pleasant & powerful redux experience than JS ecosystem
- **Pot** monad holds a piece of application state and supports all frontend-specific effects in a very natural way
- Really shines in combination with scalajs-react

Evolution Gaming 10 YEARS ALIVE



Wrap-up

IV. Conclusion

Thank you!
Q&A