International
JavaScript Conference

Angular 5 vs. React
When to Choose Which?

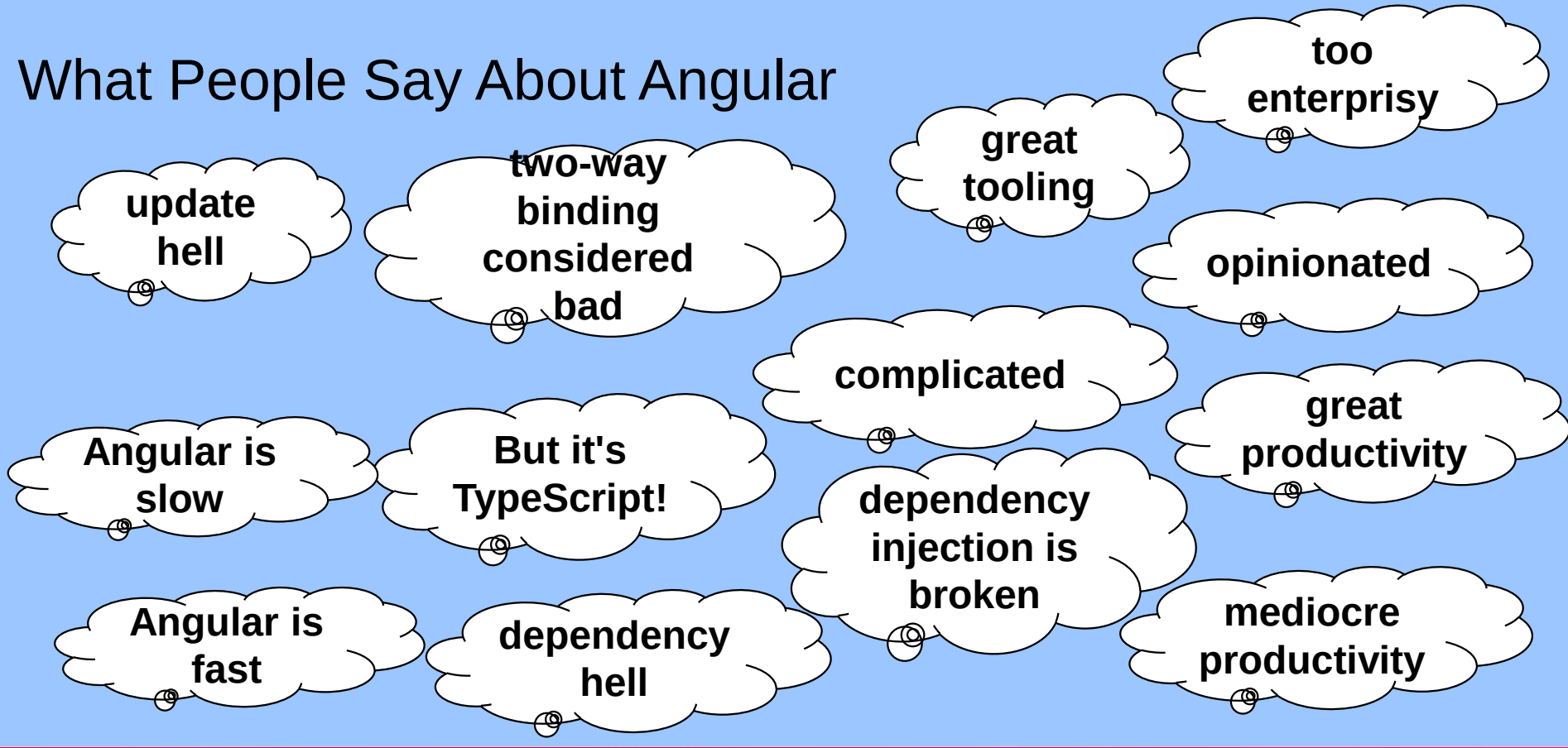
Stephan Rauh
Dr. Marius Hofmeister

OPITZ CONSULTING Deutschland GmbH

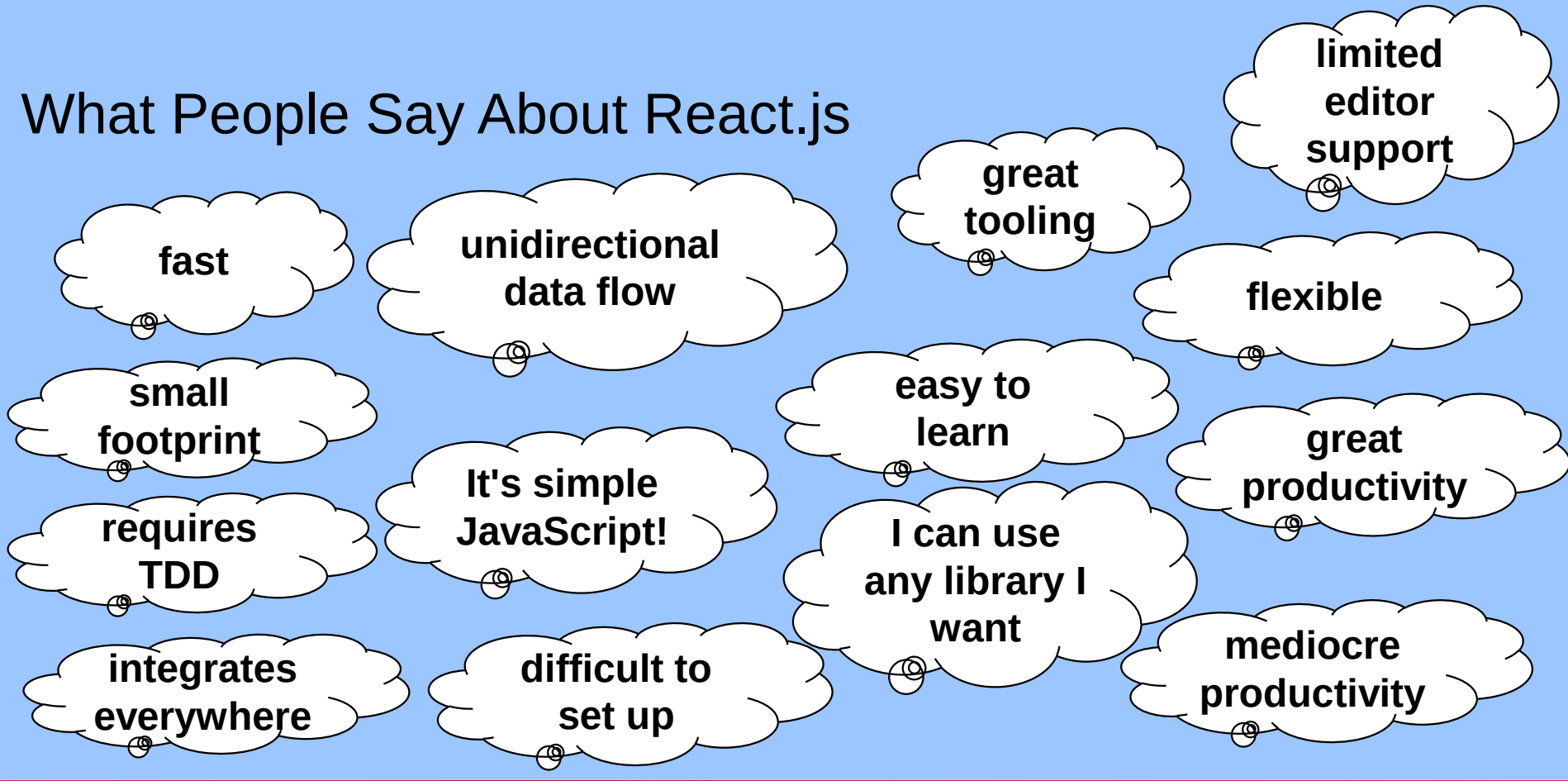
Setting the Stage

https://upload.wikimedia.org/wikipedia/commons/5/52/Summer_Solstice_Sunrise_over_Stonehenge_2005.jpg

What People Say About Angular



What People Say About React.js



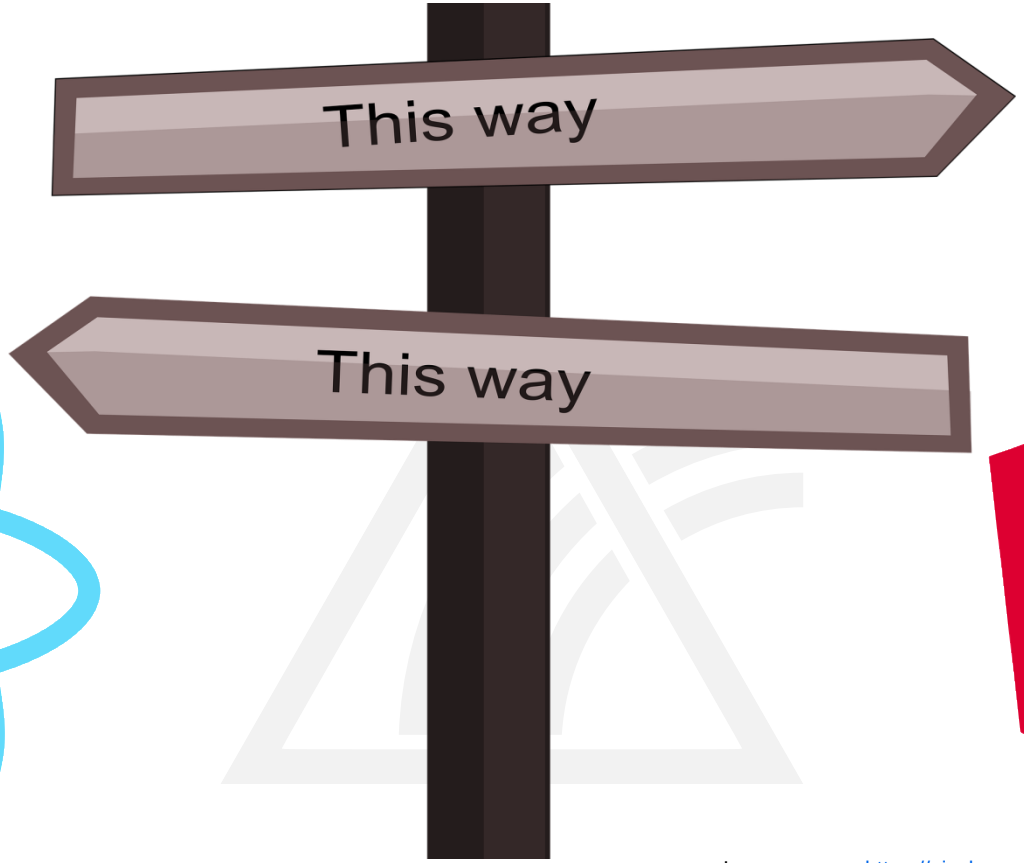


Image source: https://pixabay.com/de/hier-entlang-verwirren-718660/, https://angular.io/presskit, https://en.wikipedia.org/wiki/File:React-icon.svg

What's Going On?

- Different target audiences
  - Web developers vs. enterprise developers
- Different use cases
  - Interactive web pages
  - Web shops
  - Back-office processing
  - Internet vs intranet

Image source: https://pixabay.com/de/online-speicher-gesch%C3%A4ft-kaufen-1905889/

History 101: AngularJS 1.x

- Combined ideas from Backbone, Knockout, Web Components spec
- Made MVVM popular in the browser
- Declarative programming style
- Got rid of DOM manipulations
- Added dependency injection to JavaScript
- Brought custom components

History 101: Migration to Angular 2+

- AngularJS had a couple of issues
- Painful migration to Angular 2+
- Many breaking changes of ng2
  - In the beta versions!

→ Frustrated developers left

Angular 5?

- Angular 2+ comes with strong improvements
- Angular 3 has been skipped because there's already version 3 of the router
- Angular 4 contains two minor **breaking** changes
- Angular 5 (23.10.) contains changes concerning performance and usability

History 101: React.js

- Features
  - Virtual DOM
  - Unidirectional data flow
  - Transactional state (Redux)
  - JSX
  - Powerful toolchain

"React was transformational because it singlehandedly made MVC seem like obsolete tech & unleashed unidirectional flow on the masses.“ – Eric Elliot

Contemporary History

- React has been re-written from Scratch (Fiber)
  - Suitability for animation, layout, and gestures
  - Main feature incremental rendering
  - Less resources required
  - Fully downward compatible

http://isfiberreadyyet.com/

- New contender: Vue.js
  - “less opinionated”

Image source: https://pixabay.com/de/%C3%A4pfel-kiwi-orangen-obst-vitamine-428075/

What About Your Needs?

React.js:

- "Just a library"
- Concentrates on the "V" of MVC
- React is all about components and composition
- Ideal for complex user interactions

Angular:

- Full-blown framework
- Covers all of the MVVM paradigm (or MVC, or MVW)
- Angular is all about structuring your application
- Allows for large-scale business applications

Framework vs. Library

Angular:

- Forms and validation
- Modules
- Shadow DOM / local CSS
- Built-in support for AJAX, HTTP, and Observables
- Routing
- (Optional) two-way binding

React.js:

- Uses 3rd party libraries
- Can be combined straightforwardly with libraries
- No two-way binding
  - deliberate decision

Image source: https://upload.wikimedia.org/wikipedia/en/0/02/Telehealth_-_Blood_Pressure_Monitor.jpg

Market Share

- No unbiased figures available
- Educated guess
- React.js and Angular dominate the browser market

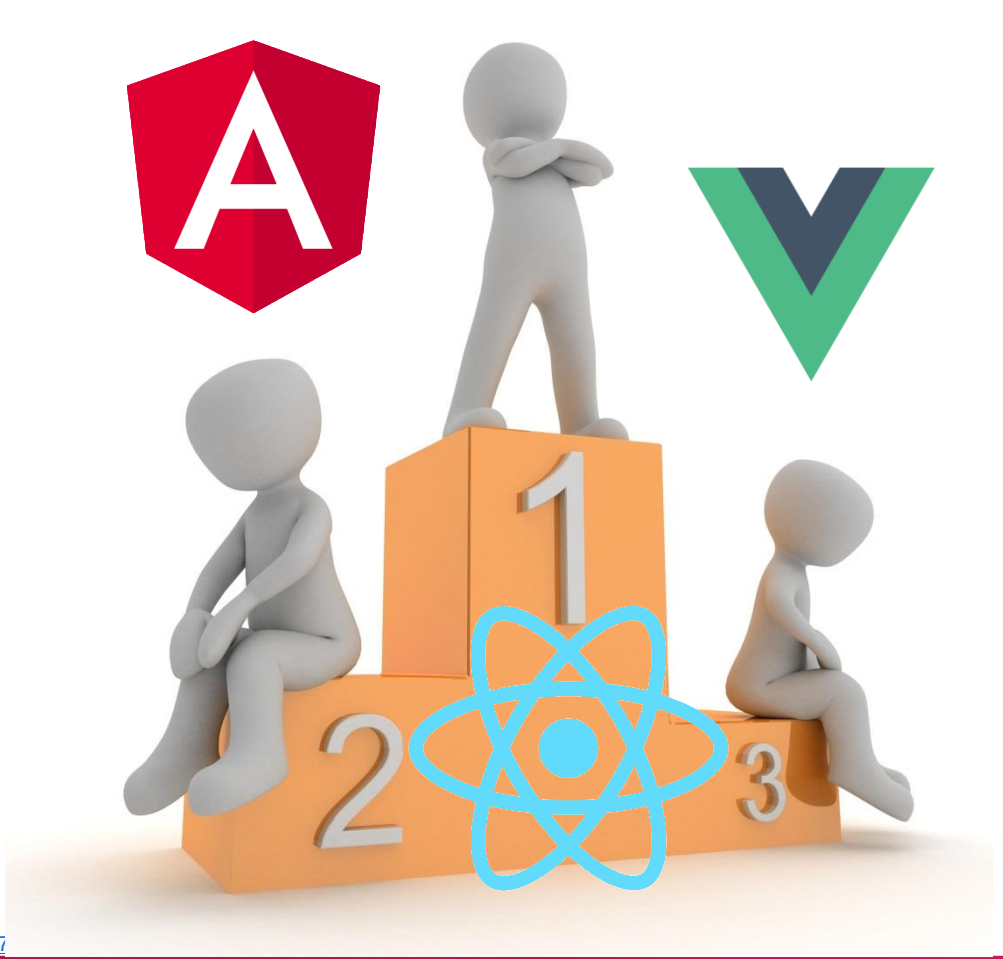


Image source: https://pixabay.com/de/sieger-siegertreppe-101397

License

React:

- Since version 16.0.0 (09/2017): MIT License
- Before: Adapted from 3-Clause BSD,
- With patent grant voided if licensor engages in any patent litigation with Facebook or its subsidiaries.

Angular:

- MIT License

But it's TypeScript!

Image source: https://www.pexels.com/photo/adult-black-and-white-body-dark-271418/

TypeScript Myths

- Strong types reduce my productivity
- TDD detects more errors than strong types
- Finding type definitions for everything is painful
- Before long, everybody uses any or starts ignoring error messages

TypeScript Mythbusting

- Strong types reduce my productivity
  - True. But only for small programs.
  - Some teams report tremendous productivity boost
- TDD detects more errors than strong types
  - Strong types allow you to omit trivial tests
- Finding type definitions for everything is painful
  - True. But things have improved a lot.
- Before long, everybody uses any or starts ignoring error messages
  - The Angular compiler has become very strict
  - Enterprise teams have learned not to ignore error messages

Another TypeScript Myth

- React uses JavaScript.
  - There's also TSX!

TypeScript Highlights

- Angular classes look clean and tidy
- Autocompletion
- Errors detected by compiler
- Optional null-safety
- Type inference
- Powerful refactorings
- Hides prototypes from you
- Easy-to-grasp semantics

TypeScript	JavaScript (ES5)
class	function
decorator	function
private scope	function
closure	function
interface	n/a
constructor	function
method	function

Core Concepts

React:

- Language: JSX or TSX
- HTML and code are deliberately put in the same file
- Unconventional programming paradigm

Angular:

- Language: TypeScript + HTML + LESS (or CSS)
- HTML, CSS and code may or may not be in the same file
- Supports both traditional and reactive programming paradigms
- Complex lifecycle (doesn’t match with jQuery)

Hot Technical Topics

- Virtual DOM vs. change detection
- Unidirectional data flow
- Components
- Local CSS (shadow DOM) (Angular)
- JSX vs. TypeScript
- Lifecycle (Angular)
- Dependency injection (Angular)

Components

React:

- All about components
- Each component has
  - Mutable state
  - Immutable properties

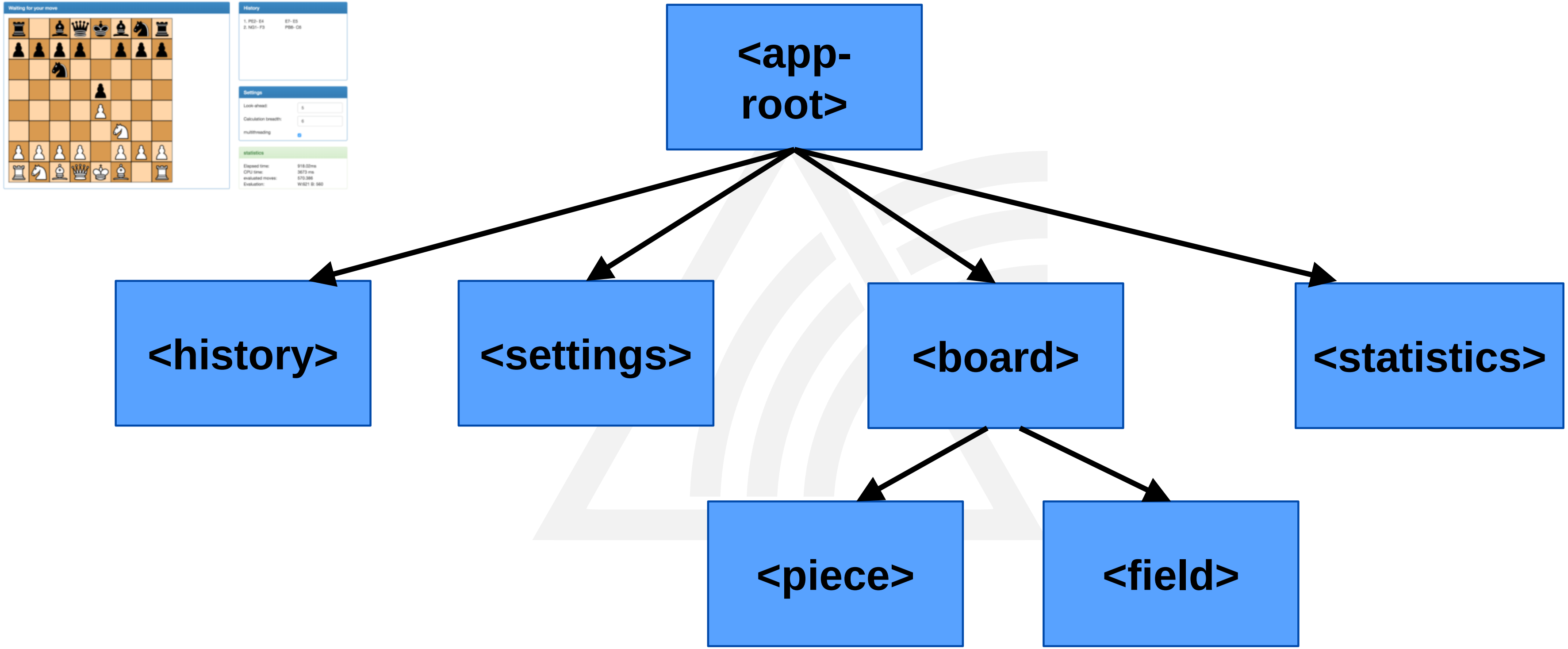
Angular:

- All about structuring your application
- Behavior is implemented in components
- State is managed in services
- Modules allow for lazy loading
  - Plus clean architecture

Components


Waiting for your move
History
1. PE2- E4
E7- E5
2. NG1- F3
PB8- C6
Settings
Look-ahead:
5
Calculation breadth:
6
multithreading
statistics
Elapsed time:
918.02ms
CPU time:
3673 ms
evaluated moves:
570.386
Evaluation:
W:621 B: 560

Components



Example Component - React

```
class ChessPiece extends React.Component {

  public state : IPieceState;

 defaultProps = { piece: "pawn", color: "black" };

  constructor() {
    this.state = {
      pos: {
        x: 5,
        y: 6
      }
    };
  }

  public render() {
    return (
        <img src="{this.props.piece} {this.props.color}.png"/>
    );
  }
}
```

- (The source code doesn't compile, but it shows the general idea)

Example Component - React

```
class ChessPiece extends React.Component {

defaultProps = { piece: "pawn" };

  constructor() {
    this.state = { x:5, y:6 }
  }

  public render() {
    return (
        <img src="{this.props.piece}.png"/>
    );
  }
}
```

- (The source code doesn't compile, but it shows the general idea)

Example Component - Angular

```
@Component({
  selector: 'app-chess-piece',
  template: '<img src="{state.piece}.png"/>',
  styleUrls: ['./chess-piece.component.css']
})
export class ChessPieceComponent
             implements OnInit {

  private state: IPieceState;

  constructor(private engine: Engine) { }

  ngOnInit() { }
}
```

Components – Data Flow (both React and Angular)

Legend:

"Pawn at E6 has moved"

Components – Data Flow (Angular only)

Components – Redux (simplified picture)
Optional with Angular and React

Rendering

React:

- Virtual DOM
- Components initially are rendered virtually, reified later and only when necessary
- Separation between immutable properties and mutable state

Angular:

- Efficient change detection strategy
- Each and every change is triggered by a change of a component's state
- Shared state should be moved to services

Local CSS (Shadow DOM)

- Angular encapsulates components in the shadow DOM
  - → CSS rules defined in a component only apply for this component
  - There's also a global CSS file
- By default, React does not use the shadow DOM
  - Here‘s a project for that: https://github.com/Wildhoney/ReactShadow

Dependency Injection (Angular Only)

- Central building block of Angular
- Allows both
  - Decoupling and
  - Exchanging services for testing purposes
- Example usecases
  - FakeHttpService instead of the standard http service
    - (since Angular 4 you can also use interceptors)
  - Testing and mocking

Tests (Angular Only)

- Tests are first class citizens of Angular
- Angular has been built with automated tests in mind
- The Angular CLI automatically generates test stubs for each entity
- Note that the strict type checks of TypeScript make many tests superfluous

Going Native

- React Native
- NativeScript (Angular)
- Ionic (Angular)

Image source: https://pixabay.com/get/eb31b70f2af7063ed1584d05fb0938c9bd22ffd41cb1144595f9c770a7/sign-2460237_1280.png

When to Use Which?

Image source: https://pixabay.com/de/richtung-weg-entscheidung-ziel-2320124/

When to Use React.js?

- Web applications with complex user interactions
- Multiple events triggering multiple view updates
- You're worried about application state
- You love functional programming
- Teams familiar with JavaScript and/or action based frameworks
- React can be added to existing apps
- Code size matters more than developer productivity
- Learning curve is reported to be high

When to Use Angular?

- Enterprise applications (large-scale)
- Easy for teams familiar with Java and component-based frameworks
- Angular is a framework to build your application with
- Unit and e2e tests are first class citizens of Angular applications
- You agree with the technical choices of the Angular team
- You're ready to spend some time learning

Any questions?

- It‘s your turn!

Let‘s make the web a place to be!

Stephan Rauh

Leiter CC „moderne Clients und agile Architekturen“

Stephan.Rauh@opitz-consulting.com

@beyondjava
http://www.beyondjava.net

Dr. Marius Hofmeister

Senior Consultant

marius.hofmeister@opitz-consulting.com

WWW.OPITZ-CONSULTING.COM @OC_WIRE OPITZCONSULTING opitzconsulting