
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE
Angular Initial Load
Performance Optimization
Hosted by Alex Thalhammer

Outline

1. 13:30 - 13:35 Intro
2. 13:35 - 14:45 ILPO Tools
3. 14:45 - 15:00 NgOptimizedImage (since NG 14.2.0)
4. 15:00 - 15:30 ☕ Break
5. 15:30 - 16:15 NG Lazy Loading done right (since NG 13 - like forever)
6. 16:15 - 16:45 NG SSR + Prerendering + Hydration (since NG 16)
7. 16:45 Q&A

About me …

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE ARCHITECT

Hi, it's me → @LX_T 

- **Alex Thalhammer** from Graz, Austria (since 1983)
  - Angular Software Tree GmbH (since 2019)
- WebDev for 22+ years (I‘ve come a long way baby)
- WordPress Dev (Web, PHP & jQuery, 2011 - 2017)
- **Angular Dev** (Web, TS, Rx since 2017 - NG 4.0.0 ☺)
- **Angular Evangelist, Coach** & **Consultant** (since 2020)
  - Member of Angular Architects https://www.angulararchitects.io/

What is Performance?

- What did you say?

- **Web performance** refers to the speed in which web pages are downloaded and displayed on the user's web browser. **Web performance optimization (WPO)**, or **website optimization** is the field of knowledge about increasing web performance.
  -- https://en.wikipedia.org/wiki/Web_performance

Why Performance Optimization?

- According to *Facebook*, when people have to wait too long for a webpage to load, they're more likely to abandon the page all together.

- According to *Google*, 40% of visitors will abandon a website if it takes longer than 3 seconds to load

- According to *Amazon*, 0.1s less loading time results in around 1% more sales

Angular Performance Optimization

We distinguish between

- Initial load performance (classical web performance)
- Runtime performance (during usage, e.g. scrolling frame rate)

Topics of my Performance Workshop

Intro, UAAO-Framework & Audit Tools

Runtime Performance

Initial Load Performance

Other topics like PWA, Web Worker Signals, RxJS & NgRx (State Mgmt.)

Outline

1. 13:30 - 13:35 Intro
2. 13:35 - 14:45 ILPO Tools
3. 14:45 - 15:00 NgOptimizedImage (since NG 14.2.0)
4. 15:00 - 15:30 ☕ Break
5. 15:30 - 16:15 NG Lazy Loading done right (since NG 13 - like forever)
6. 16:15 - 16:45 NG SSR + Prerendering + Hydration (since NG 16)
7. 16:45 Q&A

Setup for Labs I

- NodeJS version **16.14**.x (and higher 16) or **18.10**.x (and higher 18)
  - NodeJS version 14 dropped in NG 16

Actively supported versions

This table covers Angular versions under active support.

ANGULAR	NODE.JS	TYPESCRIPT	RXJS
16.0.x	^16.14.0 || ^18.10.0	>=4.9.3 <5.1.0	^6.5.3 || ^7.4.0
15.1.x || 15.2.x	^14.20.0 || ^16.13.0 || ^18.10.0	>=4.8.2 <5.0.0	^6.5.3 || ^7.4.0
15.0.x	^14.20.0 || ^16.13.0 || ^18.10.0	~4.8.2	^6.5.3 || ^7.4.0
14.2.x || 14.3.x	^14.15.0 || ^16.10.0	>=4.6.2 <4.9.0	^6.5.3 || ^7.4.0
14.0.x || 14.1.x	^14.15.0 || ^16.10.0	>=4.6.2 <4.8.0	^6.5.3 || ^7.4.0
13.3.x	^12.20.0 || ^14.15.0 || ^16.10.0	>=4.4.3 <4.7.0	^6.5.3 || ^7.4.0
13.1.x || 13.2.x	^12.20.0 || ^14.15.0 || ^16.10.0	>=4.4.3 <4.6.0	^6.5.3 || ^7.4.0
13.0.x	^12.20.0 || ^14.15.0 || ^16.10.0	~4.4.3	^6.5.3 || ^7.4.0

Try "node -v"

Windows / Linux: Use NVM ☺

Mac Users: Use Homebrew 

Setup for Labs II

- Git
  - Personally I like to use Sourcetree
- IDE: VS Code (free) or IntelliJ/WebStorm (not free, but better IMHO)
  - Personally I prefer IntelliJ/WebStorm
- Chrome (or Chromium based alternative)
  - Other browsers are okay for private matters

Ready for Takeoff!

- What are your questions?

Image from: https://bit.ly/ng-tools-img

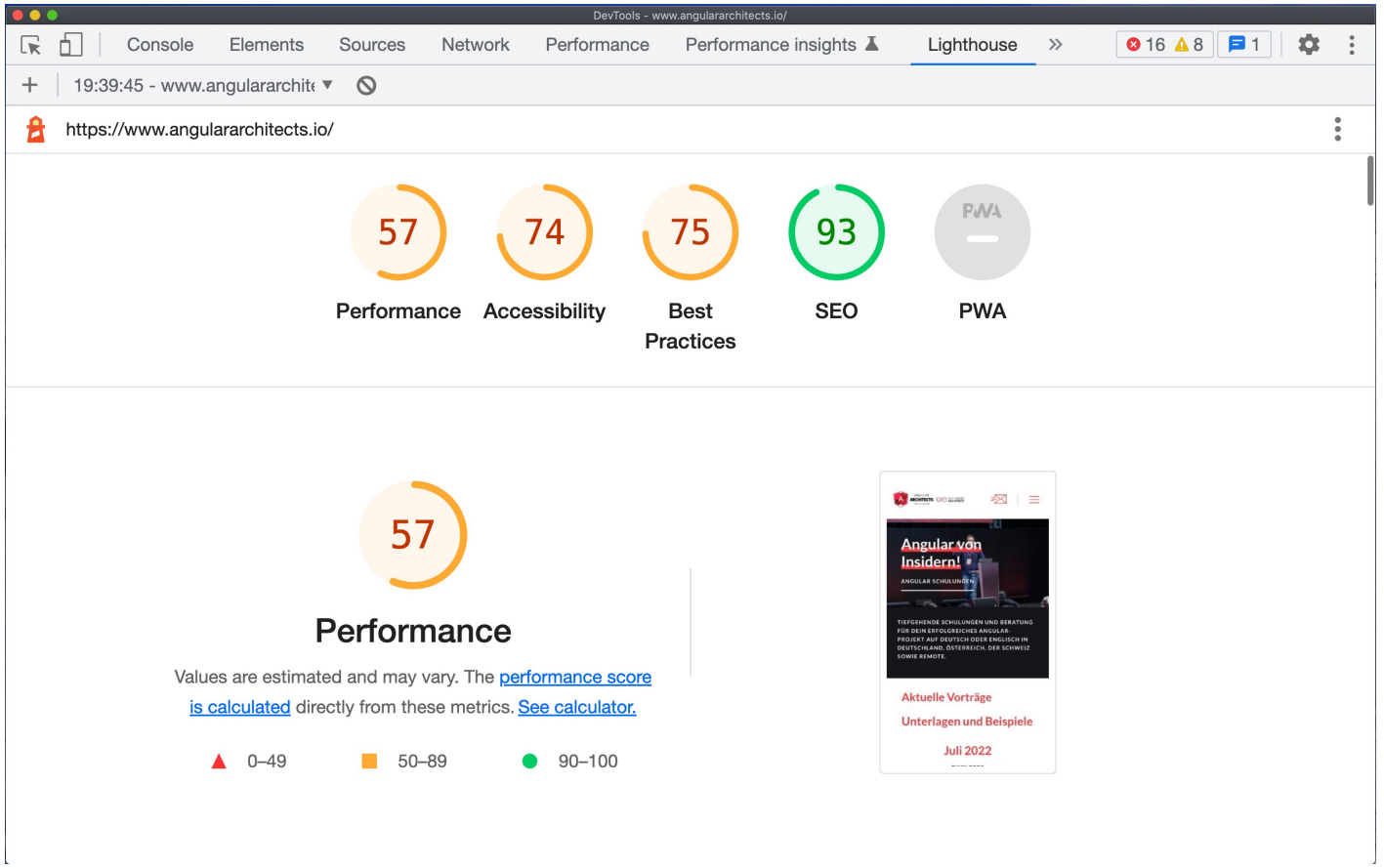
#1: Chrome Lighthouse – Prod Build Localhost

1. Prod build
    1. ng / nx b(uild)

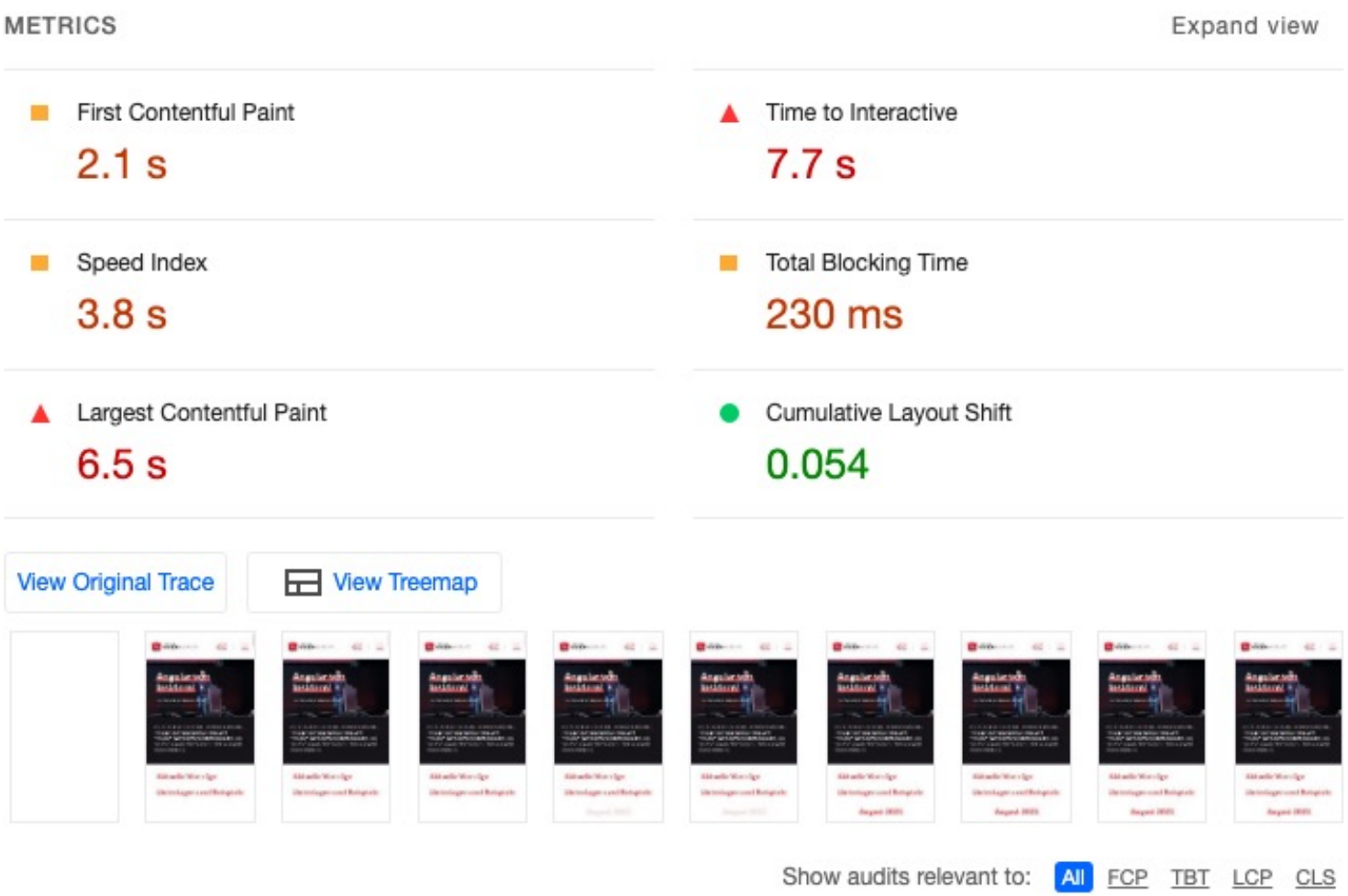
2. Serve the build on your localhost with
    - Using localhost tool like MAMP / WAMP / XAMPP
    - NPM serve
        - https://www.npmjs.com/package/serve
    - Or edit hosts file manually if you know how to do that ☺

3. Open in Chrome and run Lighthouse

#1: Chrome Lighthouse – Summary



#1: Chrome Lighthouse – Details



DEMO - Chrome Lighthouse

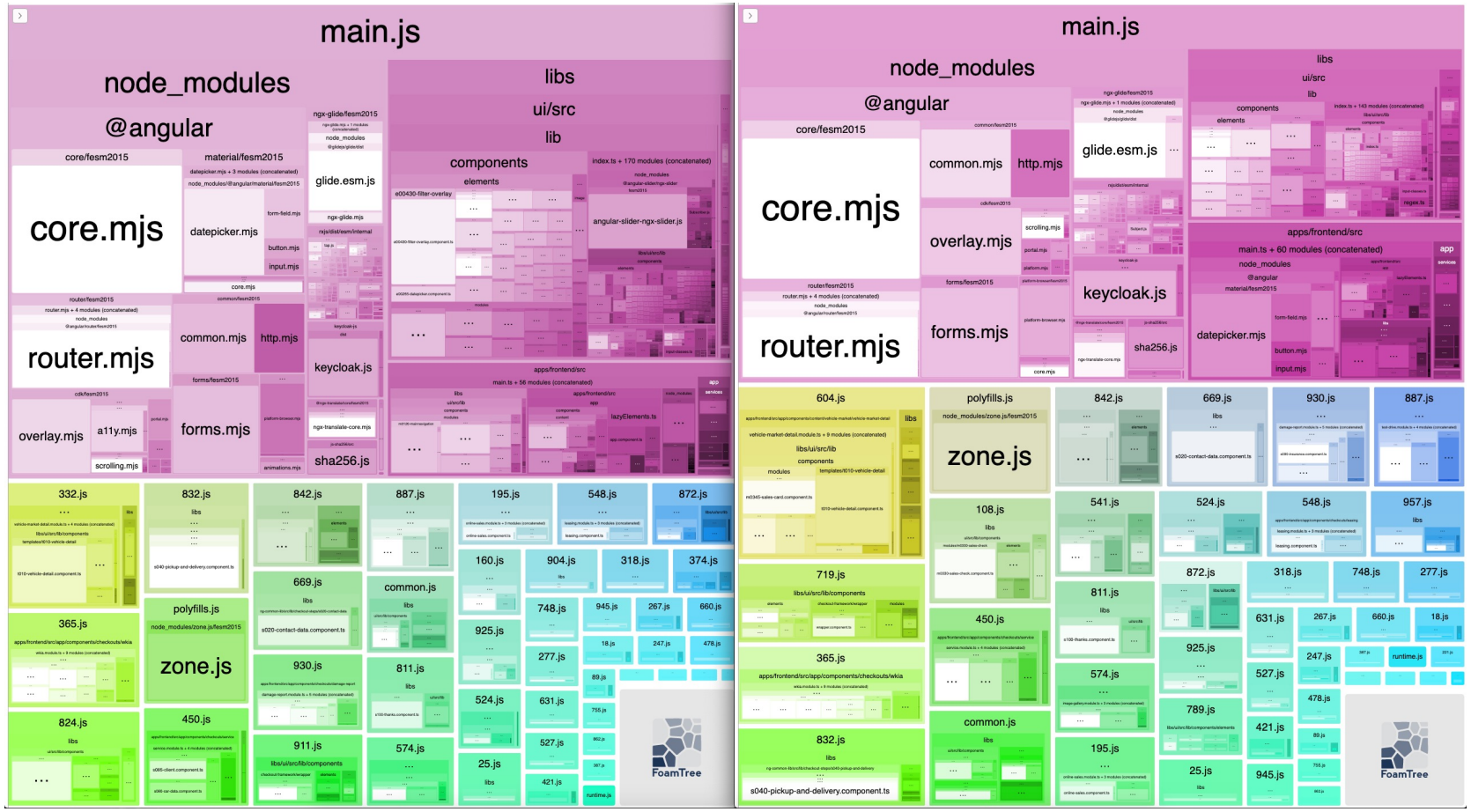
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

#1.2: Webpack Bundle Analyzer

- Needs a generated stats.json to work
  - Either set in build options (angular.json)
  - Or just use the build flag "--stats-json"
- Analyzes the whole build
- Visualize size of all webpack output js files
  - Good to analyze lazy loading
- Interactive, zoomable and colorful treemap ☺

#1.2: Webpack Bundle Analyzer (colorful)



DEMO Webpack Bundle Analyzer

https://github.com/webpack-contrib/webpack-bundle-analyzer

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

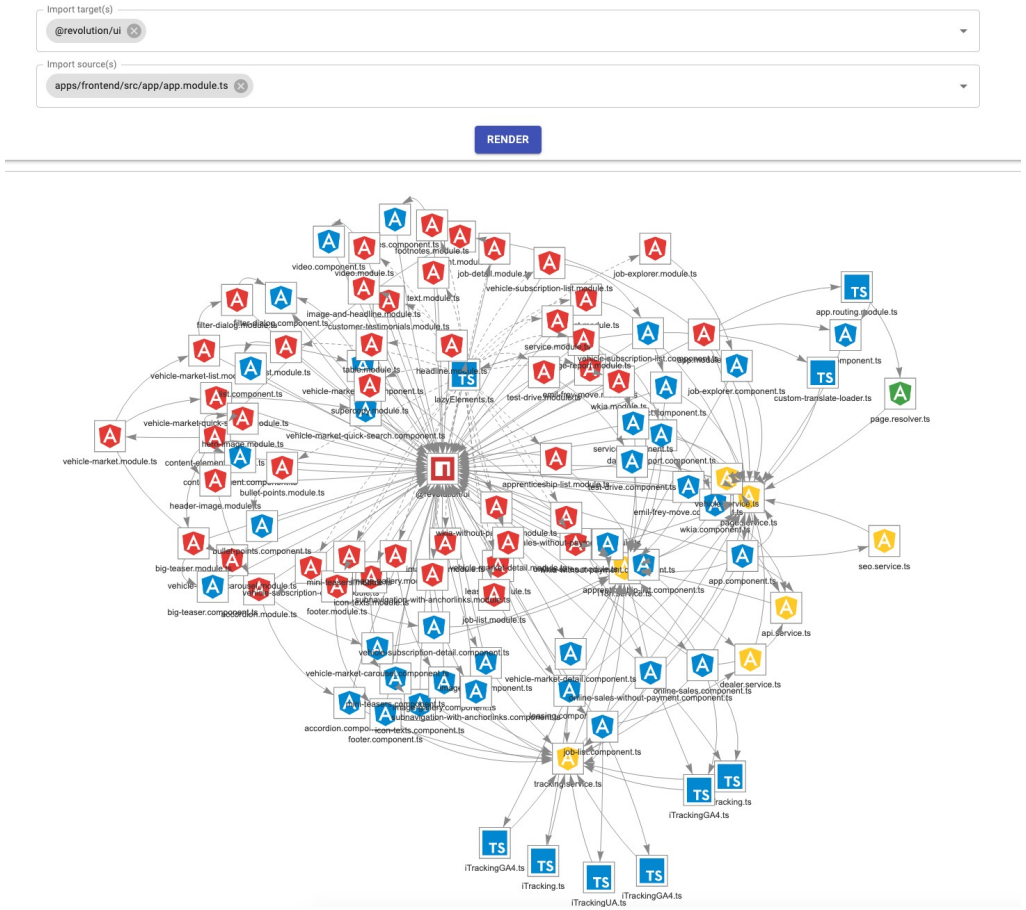
#1.3: Import Graph Visualizer

- The Import Graph Visualizer is a development tool for filtering and visualizing import paths within a JavaScript/TypeScript application

- Allows filtering import paths by source and target modules

- Allows you to zoom in to a limited subsection of your app, which will likely be easier to analyze than the entire app as a whole

#1.3: Import Graph Visualizer – Example



#1.3: Import Graph Visualizer – How To

- npm i -g @rx-angular/import-graph-visualizer

- npx import-graph-visualizer --entry-points path/to/entry/module --ts-config path/to/tsconfig

- e.g. npx import-graph-visualizer --entry-points src/main.ts --ts-config tsconfig.app.json

DEMO – Import Graph Visualizer

https://github.com/rx-angular/import-graph-visualizer#readme

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE ARCHITECT

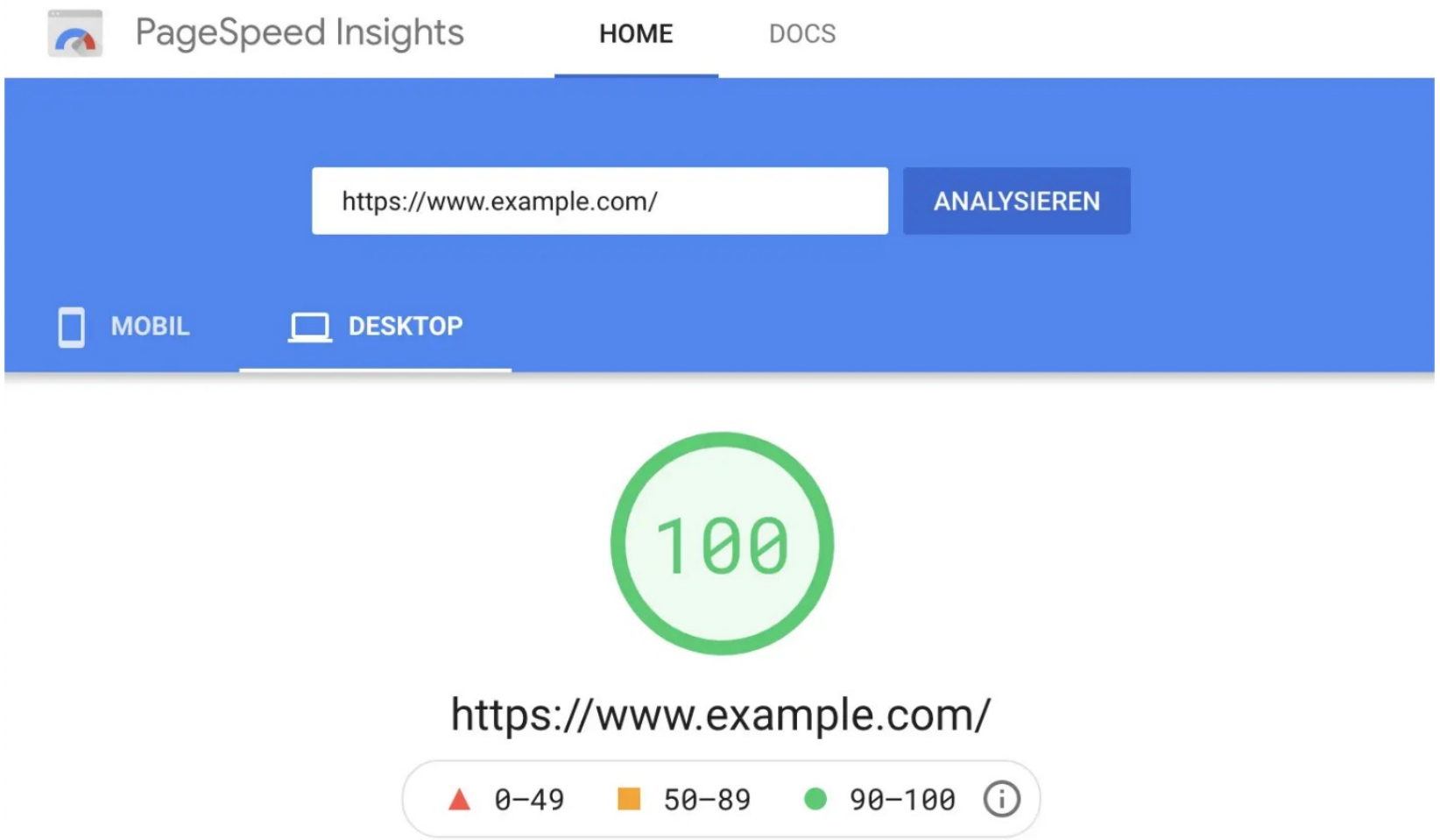
Lab

Tools

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Bonus: Use Web Performance Best Practices



#3: Use NgOptimizedImage (since NG 14.2.0)

- Problem: *Lighthouse or PageSpeed report image errors / warnings*
- Identify: Lighthouse & PageSpeed Insights / WebPageTest or DevTools
- Solution: Use NgOptimizedImage's ngSrc instead of src attribute
  - Takes care of intelligent lazy loading
  - Prioritization of critical images ("above-the-fold")
  - Also creates srcset & sizes attributes (for responsive sizes, since NG 15.0.0)
  - Also supports high-resolution devices ("Retina images")

DEMO – NgOptimizedImage

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Lab

NgOptimizedImage

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Bonus: Use @defer once NG 17 is out

```
@defer (on viewport) {
   <my-cmp />
}
```

BREAK

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

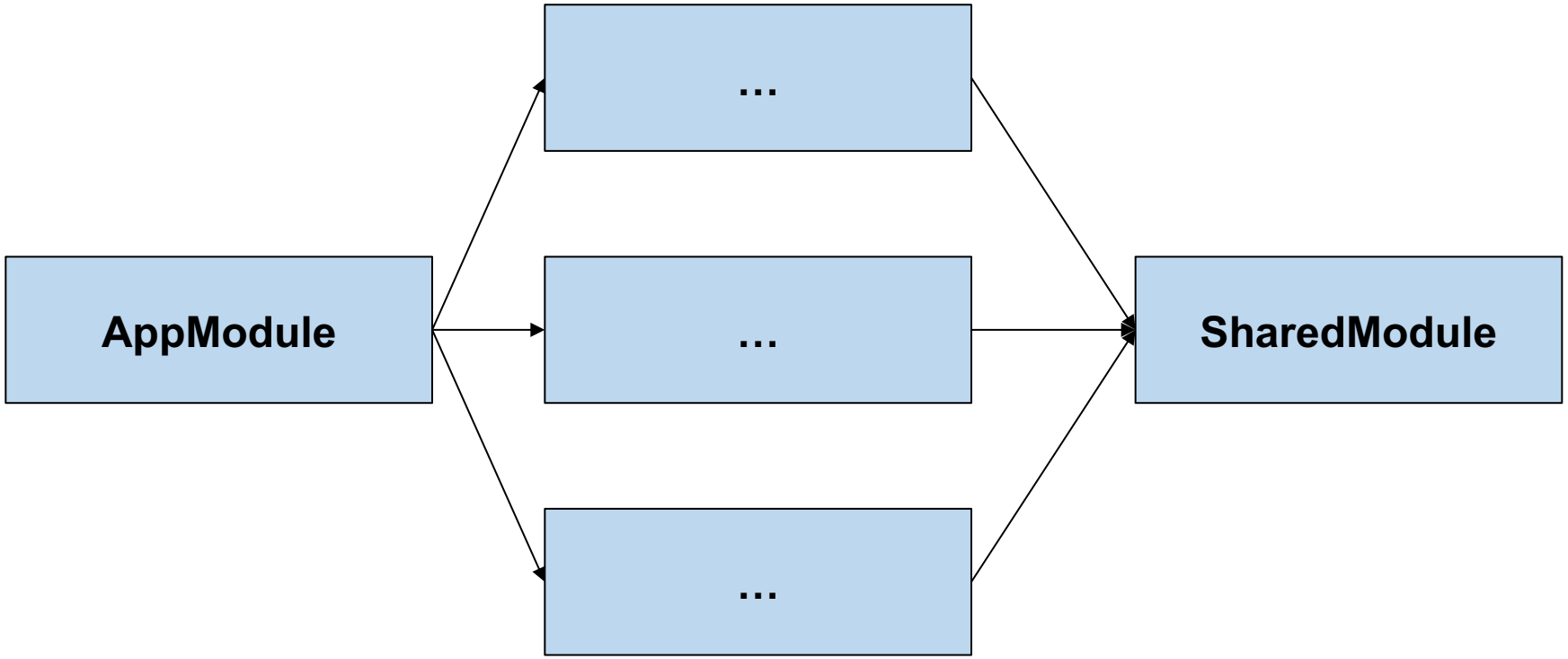
SOFTWARE
ARCHITECT

Lazy Loading

#5: Angular Lazy Loading

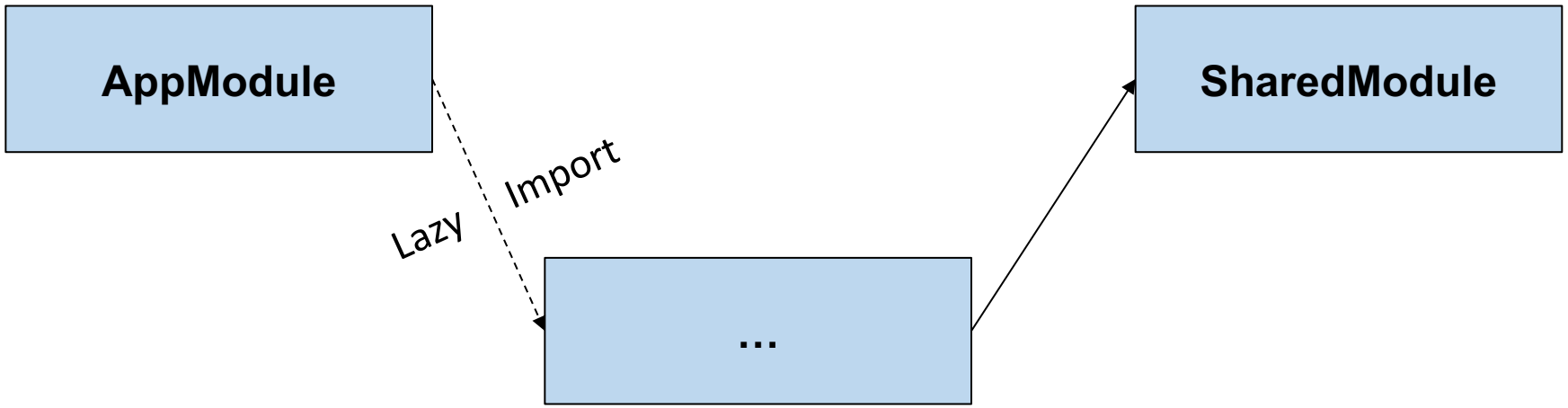
- Lazy Loading means: Load it later, after startup
- Better initial load performance

#5: Angular Module | Feature Structure



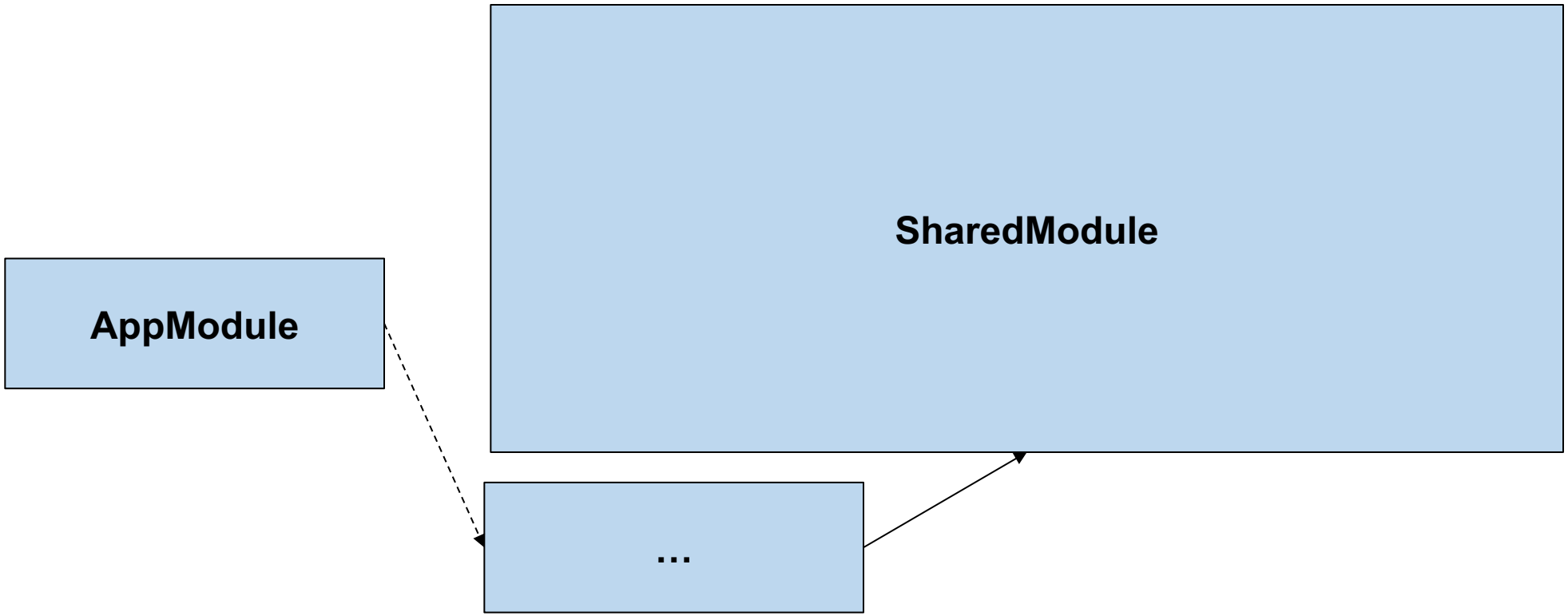
Root Module **Feature Modules** **Shared Module**

#5: Angular Lazy Loading – Theory



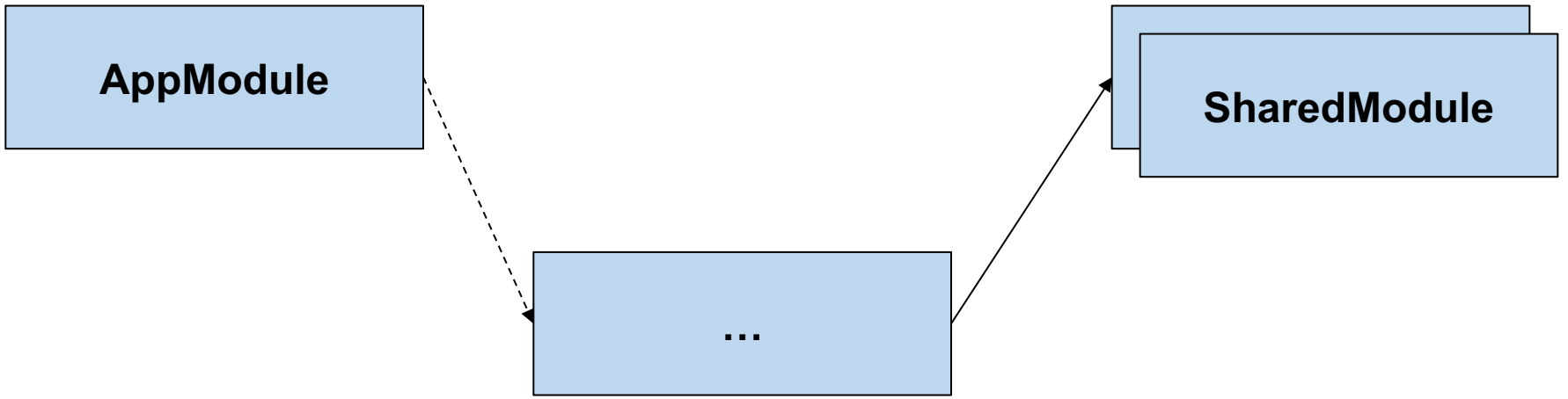
Root Module **Feature Modules** **Shared Module**

#5: Angular Lazy Loading – Common Pitfall



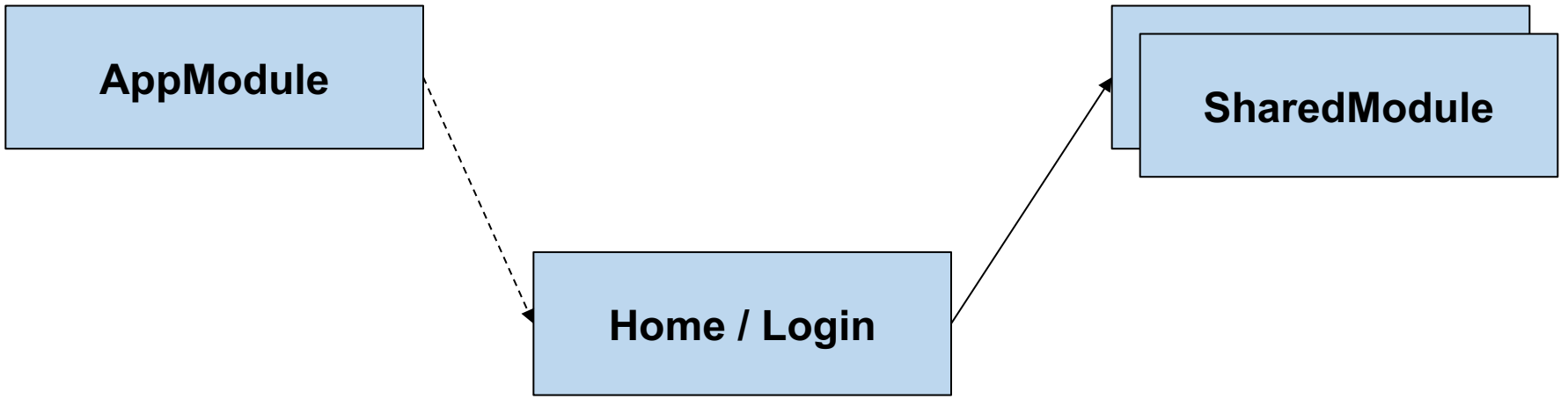
Root Module **Feature Modules** **Huge Shared Module**

#5: Angular Lazy Loading - Solution



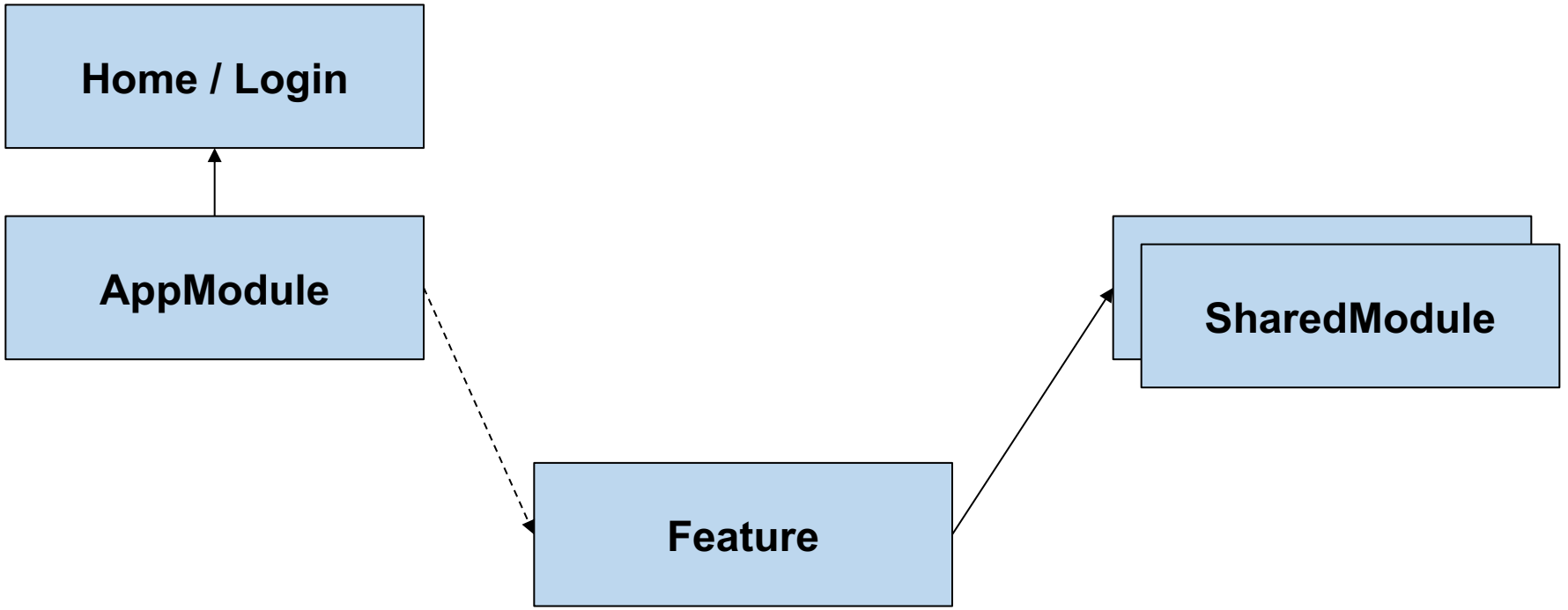
Root Module **Feature Modules** **Small Shared Modules**

#5: Angular Lazy Loading – Another Pitfall



Root Module **Feature Modules** **Small Shared Modules**

#5: Angular Lazy Loading – Solution



Root Module **Feature Modules** **Small Shared Modules**

App Routes with Lazy Loading

```
export const appRoutes: Routes = [
    {
        path: 'home',
        component: HomeComponent
    },
    {
        path: 'flights_module',
        loadChildren: () => import('./flights/flights.module')
                              .then((m) => m.FlightsModule)
    },
    {
        path: 'flights_standalone',
        loadChildren: () => import('./flights/flights.routes')
    }
];
```

Routes for "lazy" Feature

```
export const flightsRoutes: Routes = [
    {
        path: 'flight-search',
        component: FlightSearchComponent,
        […]
    },
    […]
}

export default flightsRoutes;
```

flights/ flight-search

Triggers Lazy Loading w/ loadChildren

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE ARCHITECT

DEMO – Lazy Loading

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

#5.2 Lazy Loading with Standalone Comps

```
export const appRoutes: Routes = [
    {
        path: 'home',
        component: HomeComponent
    },
    {
        path: 'charts',
        loadComponent: () => import('./charts/charts.component')
                             .then((c) => c.ChartsComponent)
    }
];
```

DEMO –Lazy Loading Standalone

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

#5 Lazy Loading without the router

```
…
    @ViewChild('cnt', { read: ViewContainerRef }) vCR!: ViewContainerRef;
…
    async ngOnInit() {
        const esm = await import('./lazy/lazy.component');
        const lazyComponentRef = this.vCR.createComponent(esm.LazyComponent);
    }
…
```

```
…
    <ng-container #cnt></ng-container>
…
```

DEMO – Dynamic Lazy Loading

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Lazy Loading

- Better initial load performance

- But: Delay during execution for loading on demand

Preloading

Idea

- Once the initial load (the important one) is complete load the lazy loaded modules (before they are even used)

- Once the module will come into use it's immediately accessable

#5: Preloading Modules

- Module that might be needed later are loaded after the application started
- When module is needed it is available immediately

#5: Activate Preloading (in AppModule)

```
…
imports: [
    […]
    RouterModule.forRoot(
        appRoutes, { preloadingStrategy: PreloadAllModules }
    );
]
…
```

#5: Activate Preloading (in app.config.ts)

```
…
providers: [
    […]
    provideRouter(
        appRoutes, withPreloading(PreloadAllModules),
    ),
]
…
```

#5: Use Lazy Loading a lot

Problem:

- *Loading to much source (libs / components) at startup*
- *Resulting in a big main bundle (and vendor if used)*

Identify:

- Not using lazy loading throughout the App (source code)
- Webpack Bundle Analyzer or
- Source Map Explorer

#5: Use Lazy Loading a lot - but carefully ;-)

Solution:

- Implement lazy loading whereever you can
  - Use lazy loading with the router
    - Modules
    - Components (new since NG15!)
    - Maybe use a CustomPreloadingStrategy if App is very big
  - Use dynamic components

- Use Import Graph Visualizer to detect why things land in main bundle

- **But** don't lazyload the initial feature, because it will be delayed ;-)

- **And** don't destroy lazy loading by (eagerly) loading a huge shared module

#5 What about services?

```
…
@Injectable({
  providedIn: 'root'
})
…
```

- When used by 1 lazy loaded module/comp exlusively it will be put into that chunk
- When used by 2 or more lazy loaded modules/comps it will be put into a common chunk
- When used by an eagerly loaded part it will be put into main bundle

DEMO – Lazy Loading Services

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

#5.1 Intelligent Preloading with ngx-quicklink

```
…
imports: [
    […]
    QuicklinkModule,
    RouterModule.forRoot(
        ROUTE_CONFIG,
        { preloadingStrategy: QuicklinkStrategy }
    );
]
…
```

https://web.dev/route-preloading-in-angular/

https://www.npmjs.com/package/ngx-quicklink

DEMO – Ngx Quicklink

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

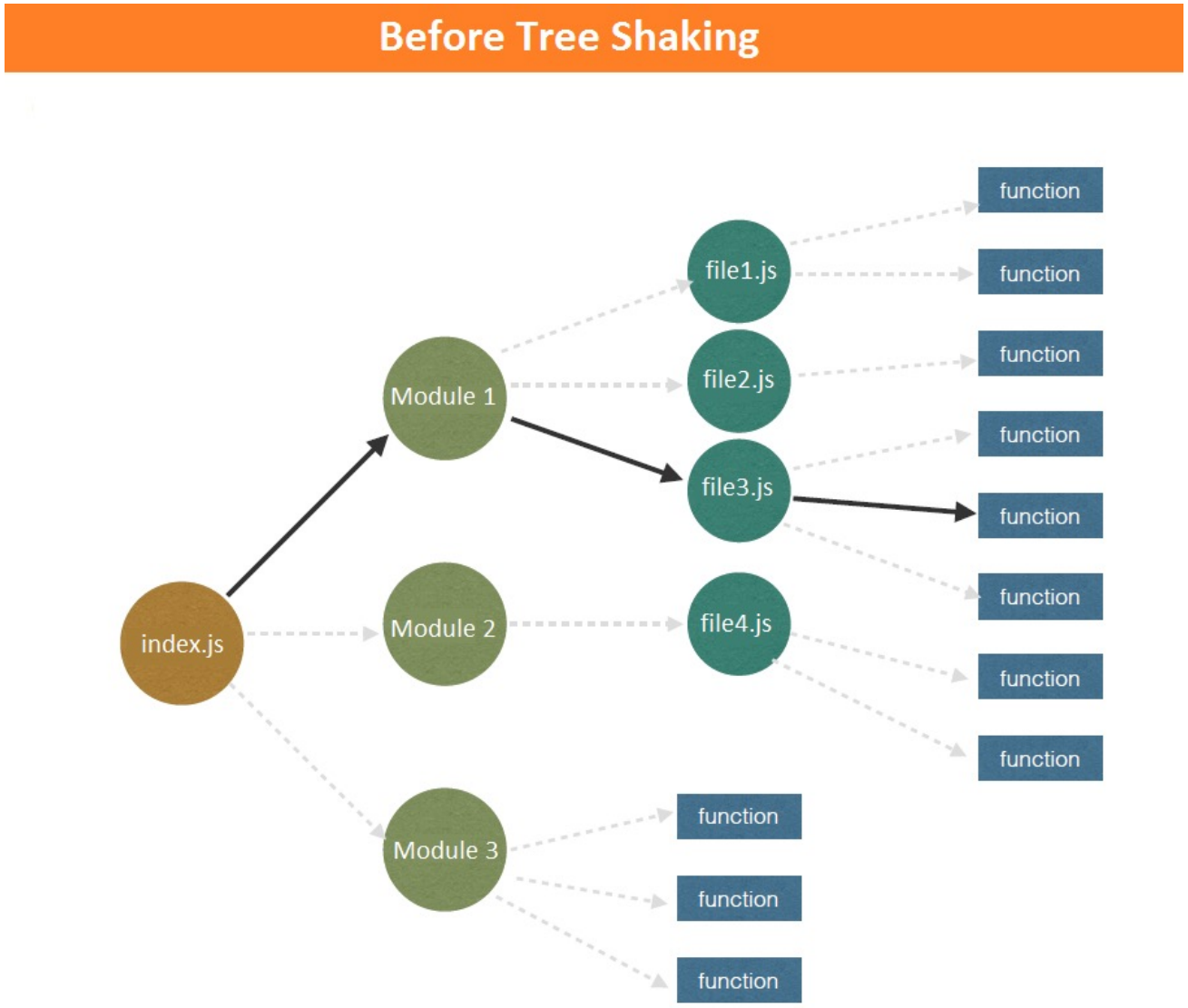
Lab

Lazy Loading

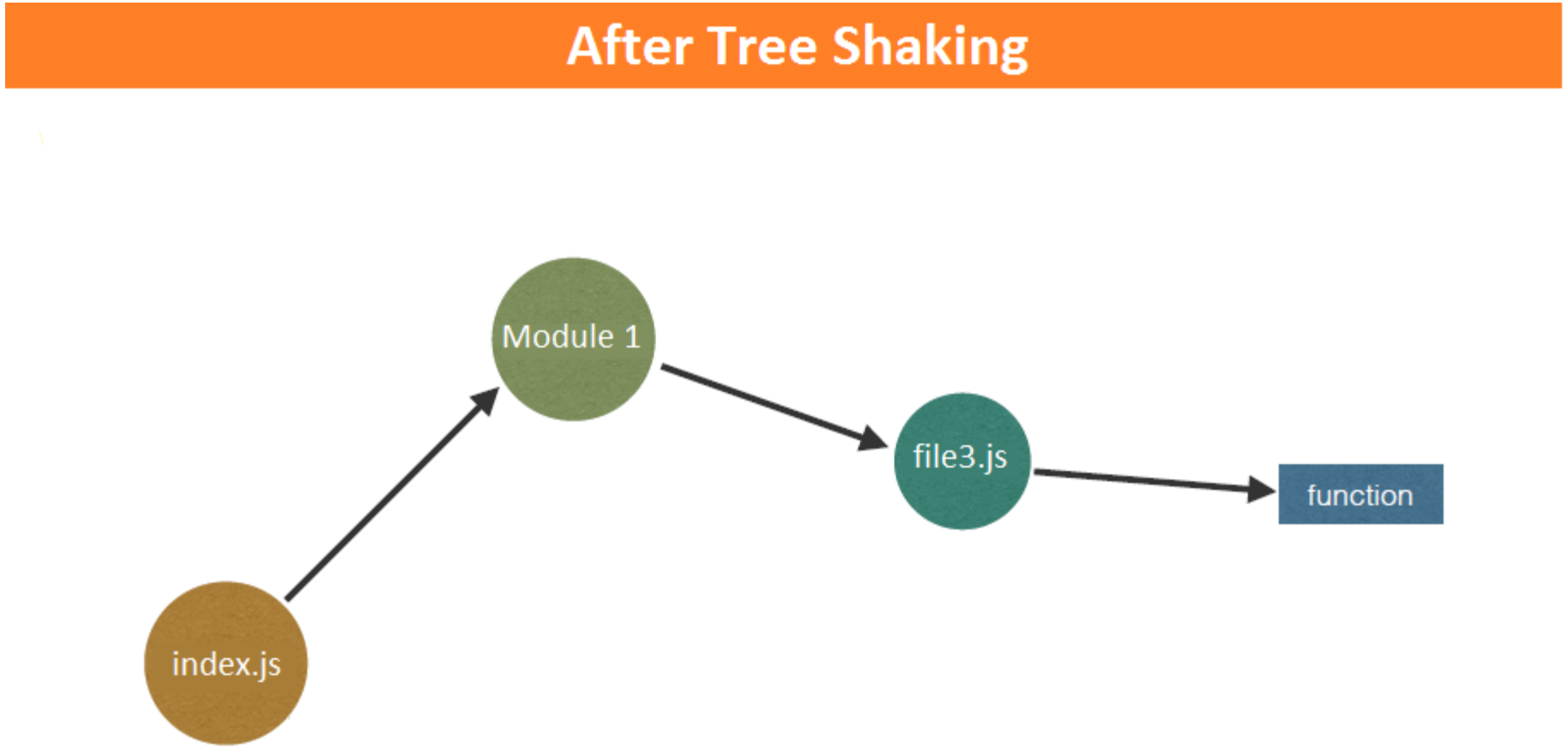
ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Bonus: Make sure to use Tree Shaking



Bonus: Make sure to use Tree Shaking



Server-side rendering

#6: Server-side rendering (Angular Universal)

- Problem: *After download rendering on the client takes too much time*
  - *Search Engines may not be abled to index the App correctly*
- Identify: After .js files have been loaded js main thread takes too long
  - Search Engines don’t index correctly
- Solution: Use Angular Universal
  - Page is rendered on the server and then served to the client
  - **But** only useful for public pages (no user login)

#6: Server-Side Rendering (Angular 16)

- New feature called *“non destructive hydration”*

DEMO – Universal

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

#6: Prerender important routes (Universal)

- Problem: *Server response takes to long cos page has to be rendered*
- Identify : Long server response time when using Universal SSR
- Solution: Prerender the important pages on the server
  - Built-in Angular Universal since V11
  - Then serve them rendered to the user

DEMO – Prerender

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE ARCHITECT

Lab

Server-Side Rendering

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Recap

1. Use web performance best practices
2. **Use NgOptimizedImage (since v15)**
3. **Use Build Optimization**
4. Avoid large 3rd party libs / CSS frameworks
5. **Use Lazy Loading done right**
6. Critical Rendering Path / Above the fold
7. **Server-Side Rendering + Prerendering + Hydration**
8. Use a URL cache

References

- Optimize the bundle size of an Angular application
  - https://www.youtube.com/watch?v=19T3O7XWJkA

- Angular Docs
  - NgOptimizedImage
  - Lazy-loading feature modules
  - Server-Side Rendering (Blog series)

Questions?

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE ARCHITECT