ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Angular Initial Load Performance Optimization

Hosted by Alex Thalhammer

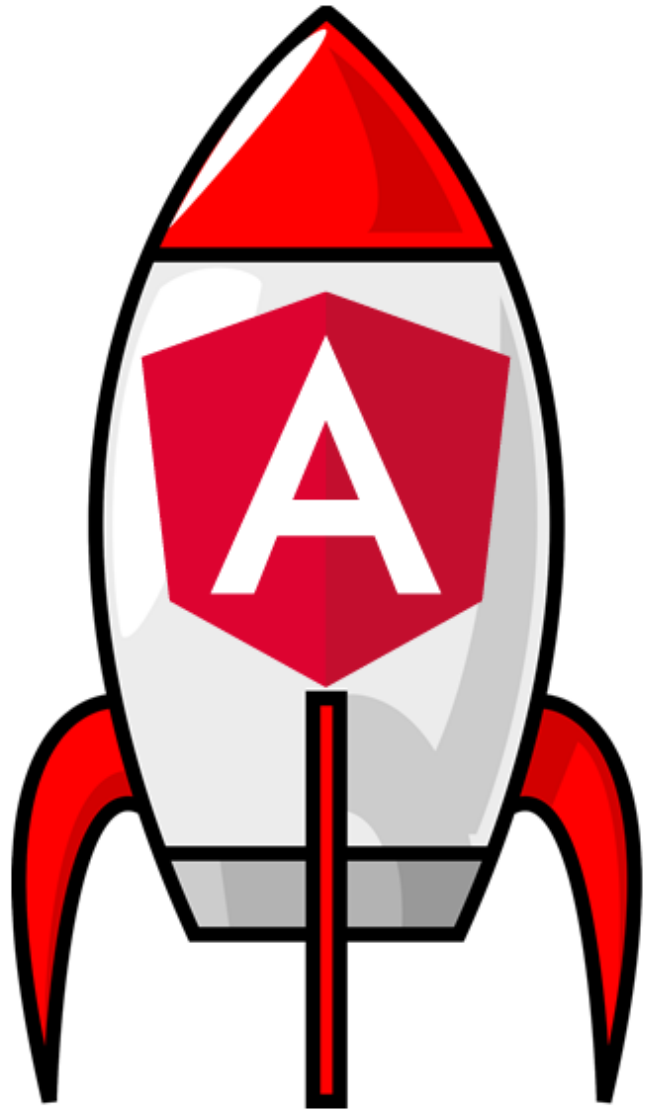
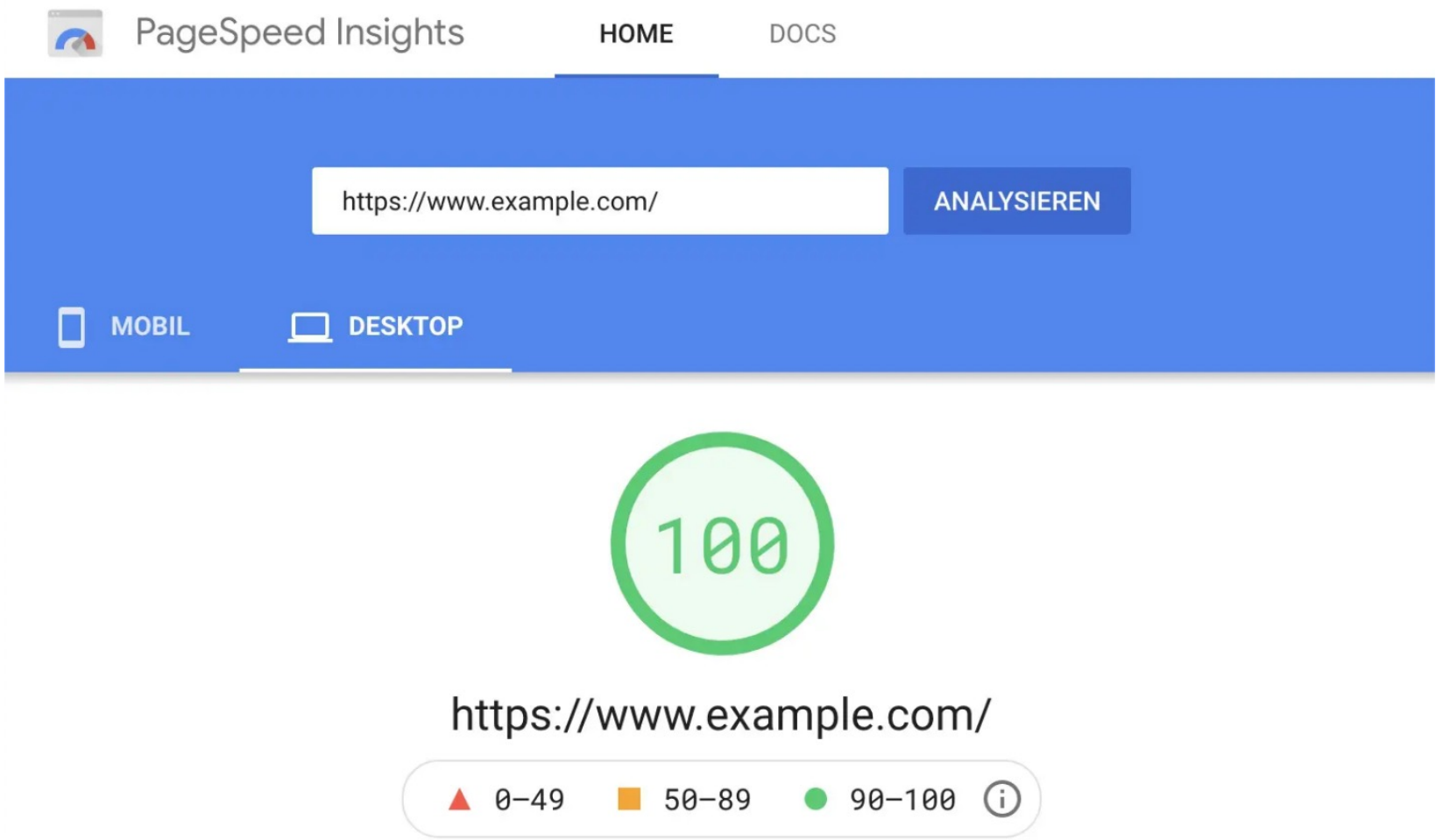


Image from: https://bit.ly/ng-initial-load

Outline

1. Use web performance best practices
2. Use Build Optimization
3. Avoid large 3rd party libs / CSS frameworks
4. Use Lazy Loading
5. Critical Rendering Path / Above the fold
6. Server-side rendering
7. Prerender on the server

Best Practices



#1: Use web performance best practices - I

Problem(s):

- *Images not optimized (use webp or avif)*
- *Images not properly sized (use srcsets)*
- *Slow server infrastructure*
- *Unused JS code or CSS styles*
- *Too large assets, too many assets*
- *Caching not configured correctly*
- *Compression not configured correctly*
- *…*

#1: Use web performance best practices - II

Identify:

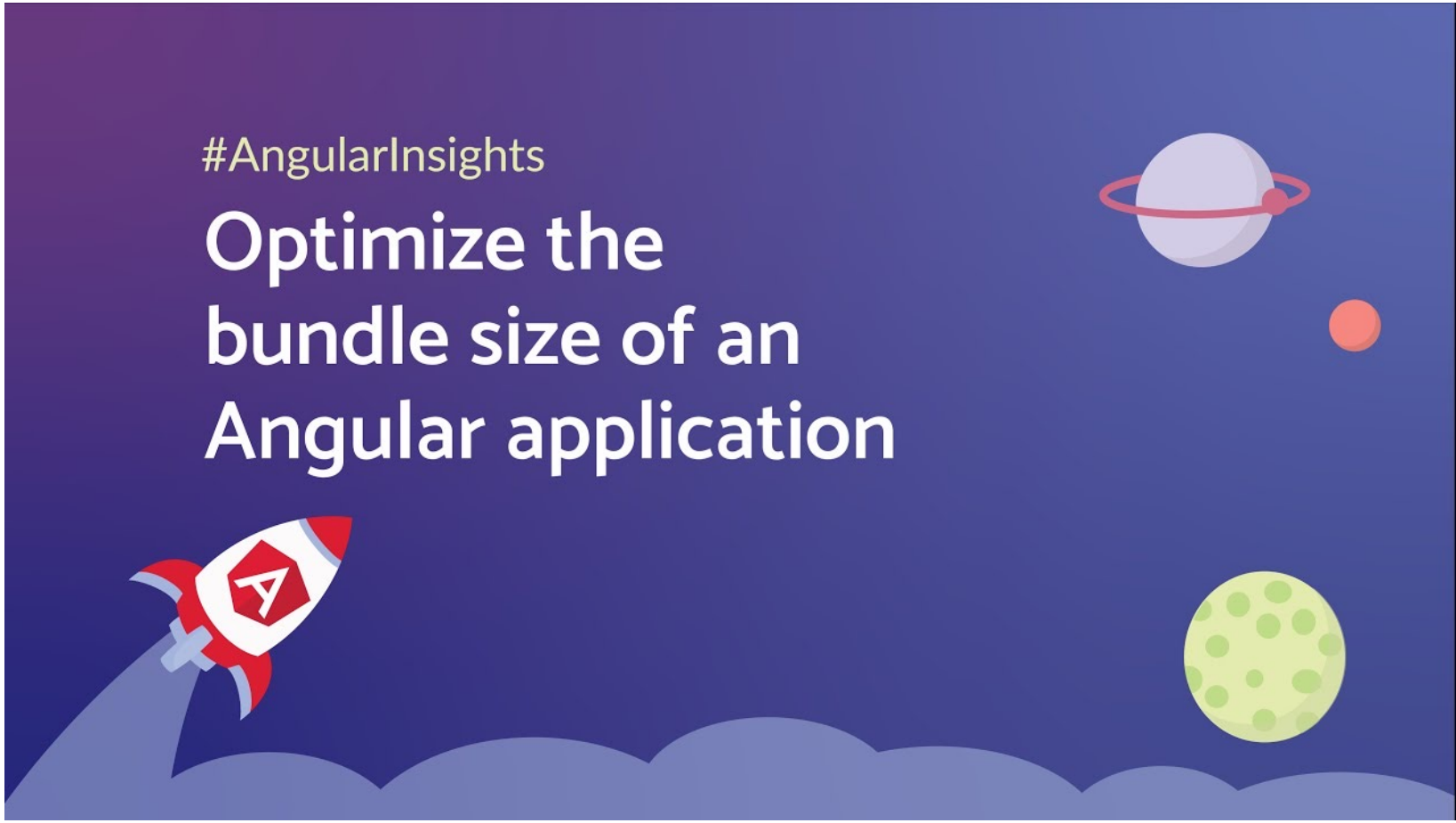
- Lighthouse & PageSpeed Insights
- WebPageTest or
- Chrome DevTools

#1: Use web performance best practices - III

Solution(s):

- *Images not optimized → Use .webp, .avif or .svg*
- *Images not properly sized → Use srcsets*
- *Slow server infrastructure → HTTP/2, CDN*
- *Unused JS code or CSS styles → Clean up & lazy load assets*
- *Too large assets, too many assets → Clean up & lazy load assets*
- *Caching not configured correctly → Configure it*
- *Compression not configured correctly → Brotli or Gzip*
- *…*

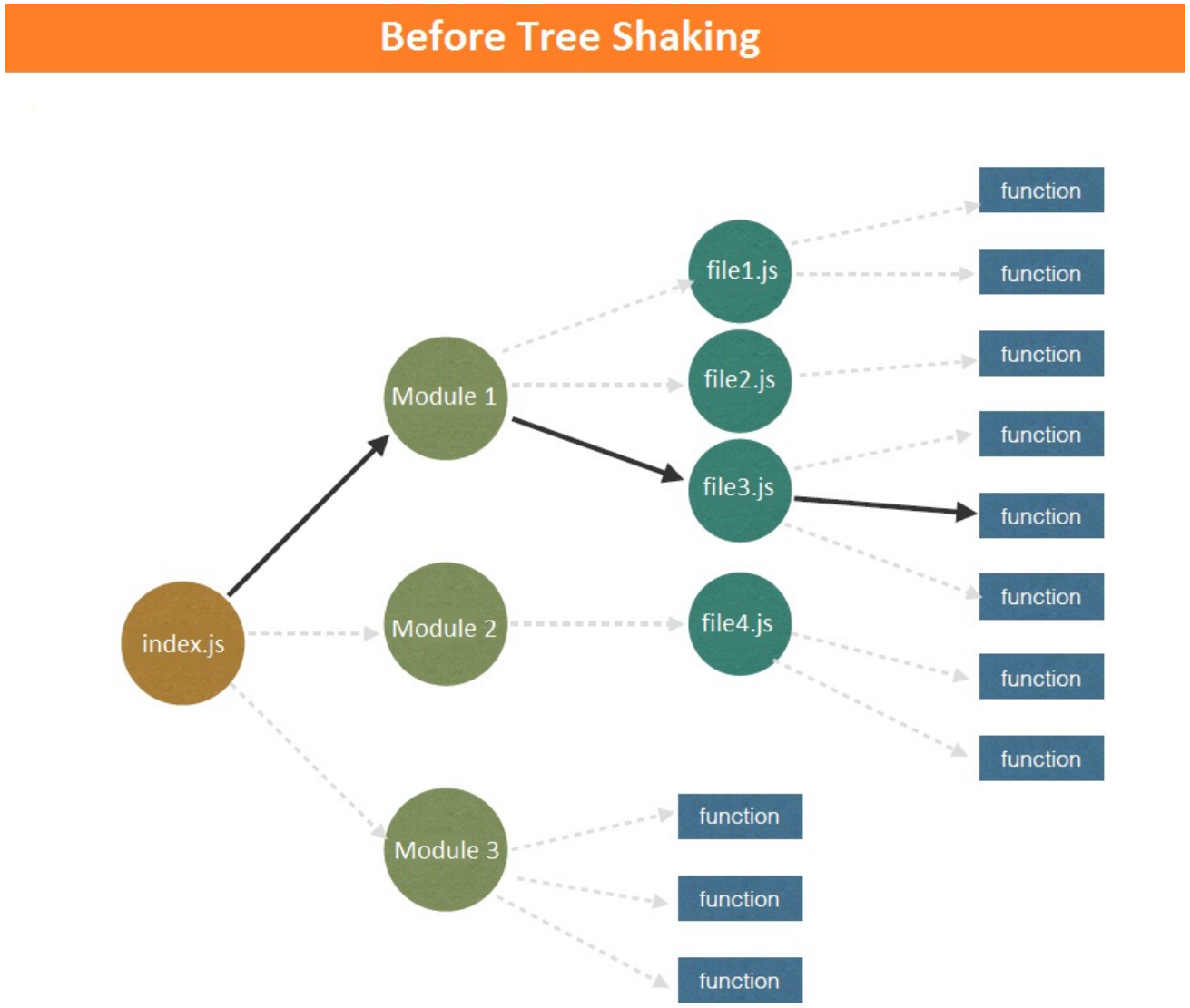
Build optimization



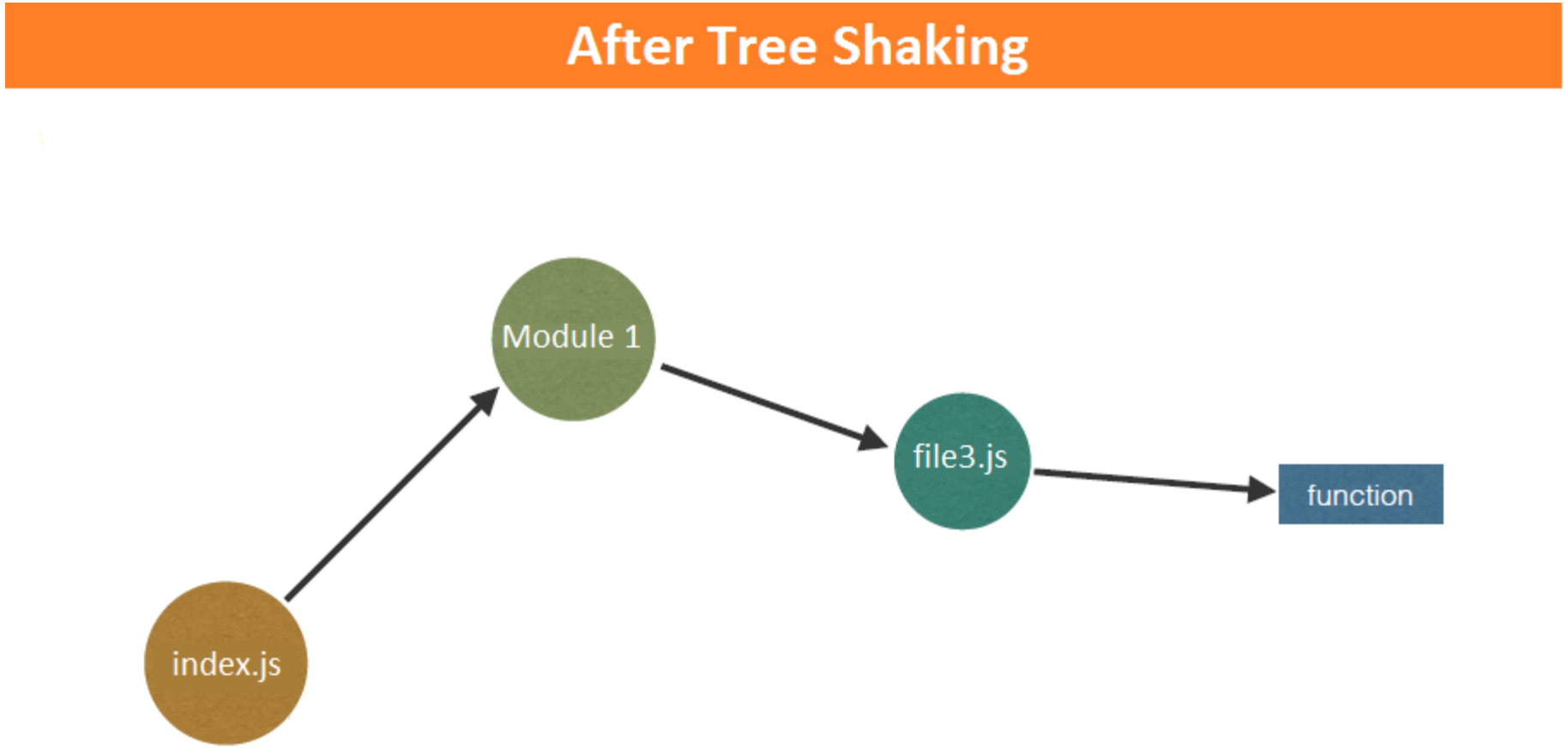
Advantages of Angular Ivy (since V9)

- Angular ViewEngine itself was not tree-shakable
- Default since NG 10, for libs default since NG 12
- AOT per default → You don't need to include the compiler!
- Ivy also does a lot of under the hood optimization
- Tools can easier analyse the code
  - Remove unneeded parts of frameworks
  - Called Tree Shaking
    - Also 3rd party and
    - Even our own libs

Tree Shaking



Tree Shaking



#2: Use Build Optimization – I

Problem:

- *Too large build*
- *Downloading the Angular App takes too much time / ressources*

#2: Use Build Optimization – II

Identify:

- CSS / JS Files not minimized
- Unused JS code included in the build

#2: Use Build Optimization – III

Solution:

- Use production build
  - ng build --c production (--stats-json)

- Set up angular.json correctly

```
"optimization": true,
"outputHashing": "all",
"sourceMap": false,
"namedChunks": false,
"extractLicenses": true,
"vendorChunk": false,
"buildOptimizer": true,
```

DEMO – Build Configuration

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

#3: Avoid large 3rd party libs / CSS frameworks

Problem:

- *Importing large 3rd party libraries that are not treeshakable*
  - *moment*
  - *lodash*
  - *charts*
  - *…*

#3: Avoid large 3rd party libs / CSS frameworks

Identify:

- Source Map Analyzer or
- Webpack Bundle Analyzer

#3: Avoid large 3rd party libs / CSS frameworks

Solution:

- Remove or replace that lib / framework
  - *moment → date-fns*
  - *lodash → lodash-es*
  - *…*

DEMO – Large Libs

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

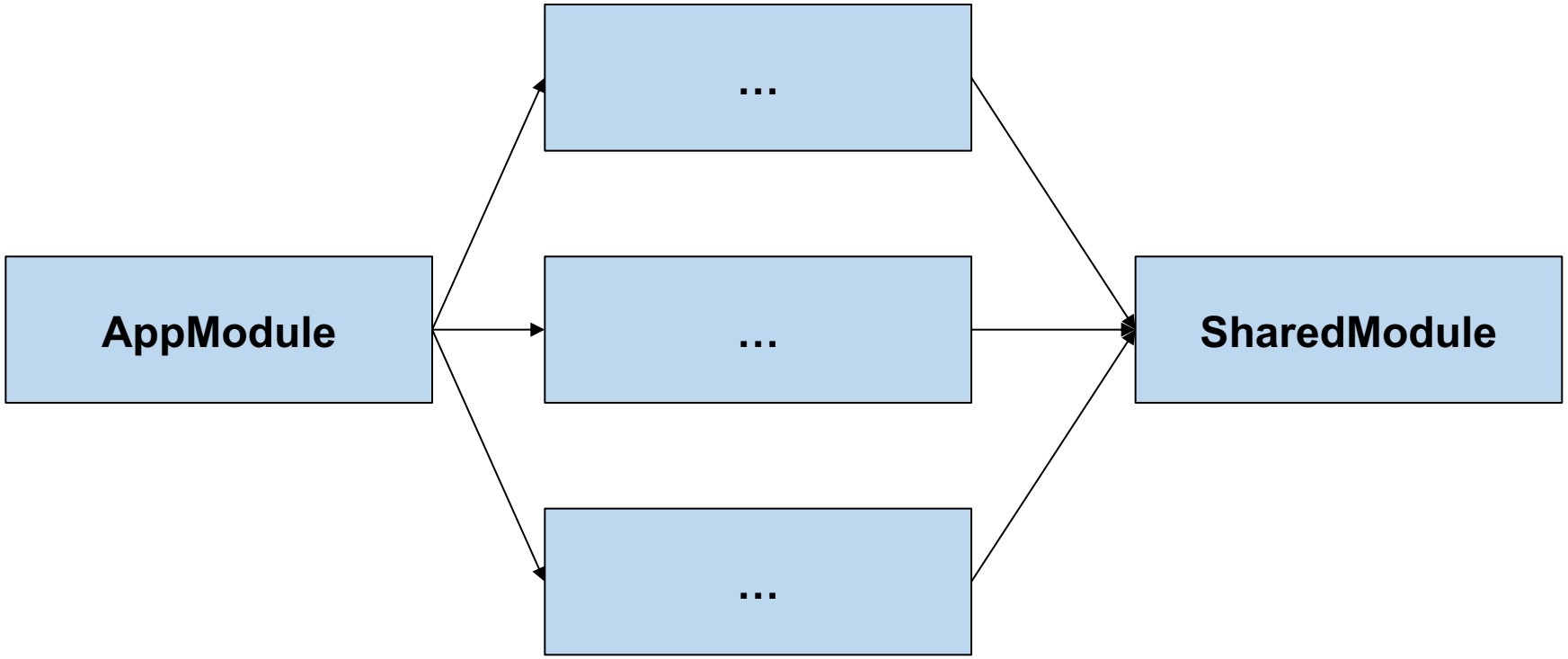
SOFTWARE
ARCHITECT

Lazy Loading

Lazy Loading

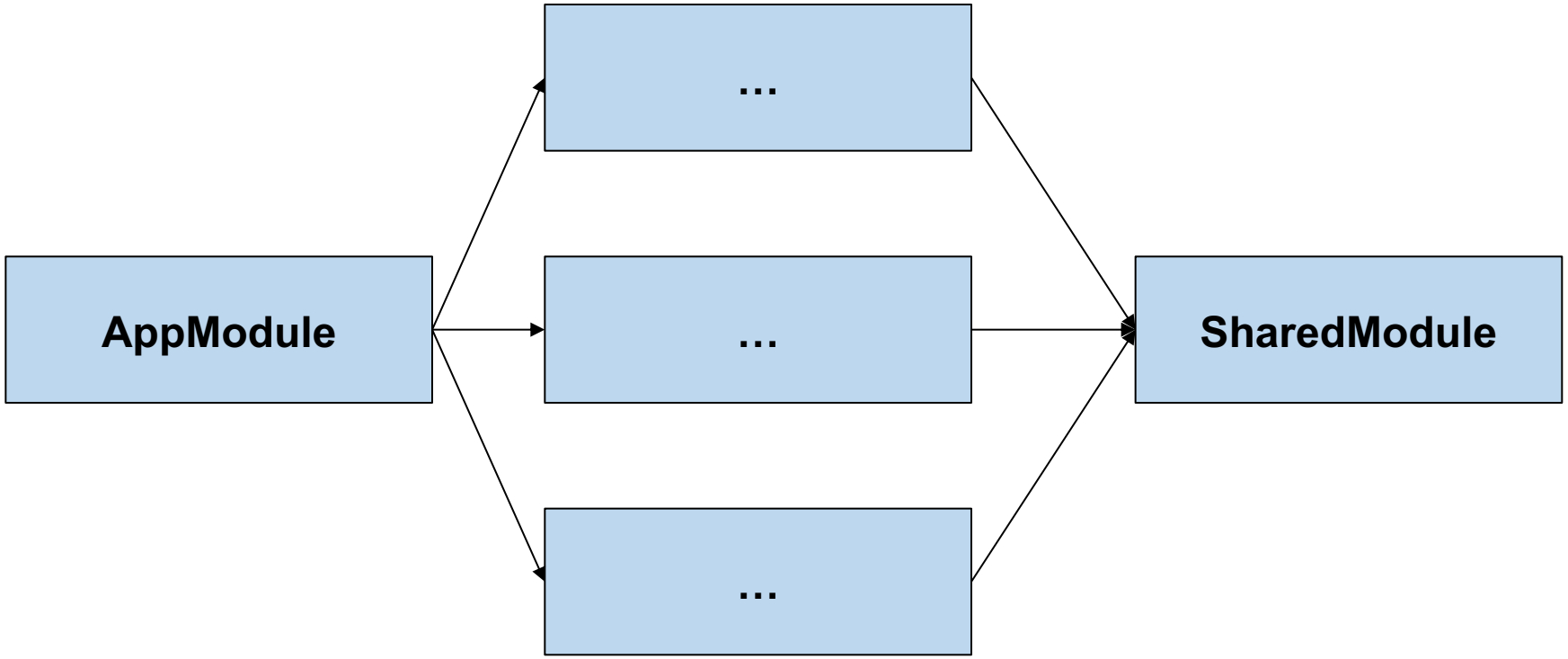
- Lazy Loading means: Loading it later
- Better startup performance
- Delay during execution for loading on demand

Module Structure



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

Lazy Loading



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

Root Module with Lazy Loading

```
const APP_ROUTE_CONFIG: Routes = [
    {
        path: 'home',
        component: HomeComponent
    },
    {
        path: 'flights',
        loadChildren: () => import('./flight-booking/flight-booking.module')
                              .then(m => m.FlightBookingModule)
    }
];
```

Routes for "lazy" Module

```
const FLIGHT_ROUTES =    [
    {
        path: '',
        component: FlightBookingComponent,
        […]
    },
    […]
}
```

Routes for "lazy" Module

```
const FLIGHT_ROUTES =    [
    {
        path: 'subroute',
        component: FlightBookingComponent,
        […]
    },
    […]
}
```

flight-booking/ subroute

Triggers Lazy Loading w/ loadChildren

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE ARCHITECT

Preloading

Preloading

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

Activate Preloading

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

#4: Use Lazy Loading

Problem:

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

#4: Use Lazy Loading a lot

Identify:

- Not using lazy loading throughout the App (source code)
- Webpack Bundle Analyzer or
- Import Graph Visualizer

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

Solution:

- Implement lazy loading whereever you can
  - Use lazy loading with the router
    - Modules
    - Components (new in NG14!)
    - Maybe use a CustomPreloadStrategy 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 ;-)

DEMO – Lazy Loading

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

#5: Critical Rendering Path / Above the fold - I

Problem:

- *Bad PageSpeed Score that cannot be fixed with #1*

#5: Critical Rendering Path / Above the fold - II

Identify:

- Initial load is too slow
  - Using Lighthouse / PageSpeed Insights or
  - WebPageTest

#5: Critical Rendering Path / Above the fold - III

Solution:

- Use custom lazy loading of content below the fold
  - Not trivial
  - Has to be implemented manually

Server-side rendering

#6: Server-side rendering (Angular Universal)

Problem:

- *After download rendering on the client takes too much time*

- *Search Engines not able to index*

#6: Server-side rendering (Angular Universal)

Identify:

- After .js files have been loaded js main thread takes too long

- Search Engines don’t index correctly

#6: Server-side rendering (Angular Universal)

Solution:

- Use Angular Universal

- Page is rendered on the server and then served to the client

#7: Prerender important routes (Universal)

Problem:

- *Server response takes to long cos page has to be rendered*

#7: Prerender important routes (Universal)

Identify:

- Long server response time when using Universal SSR

#7: Prerender important routes (Universal)

Solution:

- Prerender the important pages on the server
  - Built-in Angular Universal since V11

- Then serve them rendered to the user

#7: Use a (URL) cache

Alternative Solution:

- Of course you could also use an alternative caching solution
  - E.g. Cloudflare or any other CDN

Lab

Initial Load Performance

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Recap

1. **Use web performance best practices**
2. **Use Build Optimization**
3. **Avoid large 3rd party libs / CSS frameworks**
4. **Use Lazy Loading**
5. Critical Rendering Path / Above the fold
6. **Server-side rendering**
7. Prerender or cache on the server

References

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

- Angular Docs
  - NG Build
  - Lazy-loading feature modules
  - Server-side rendering (SSR) with Angular Universal