ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

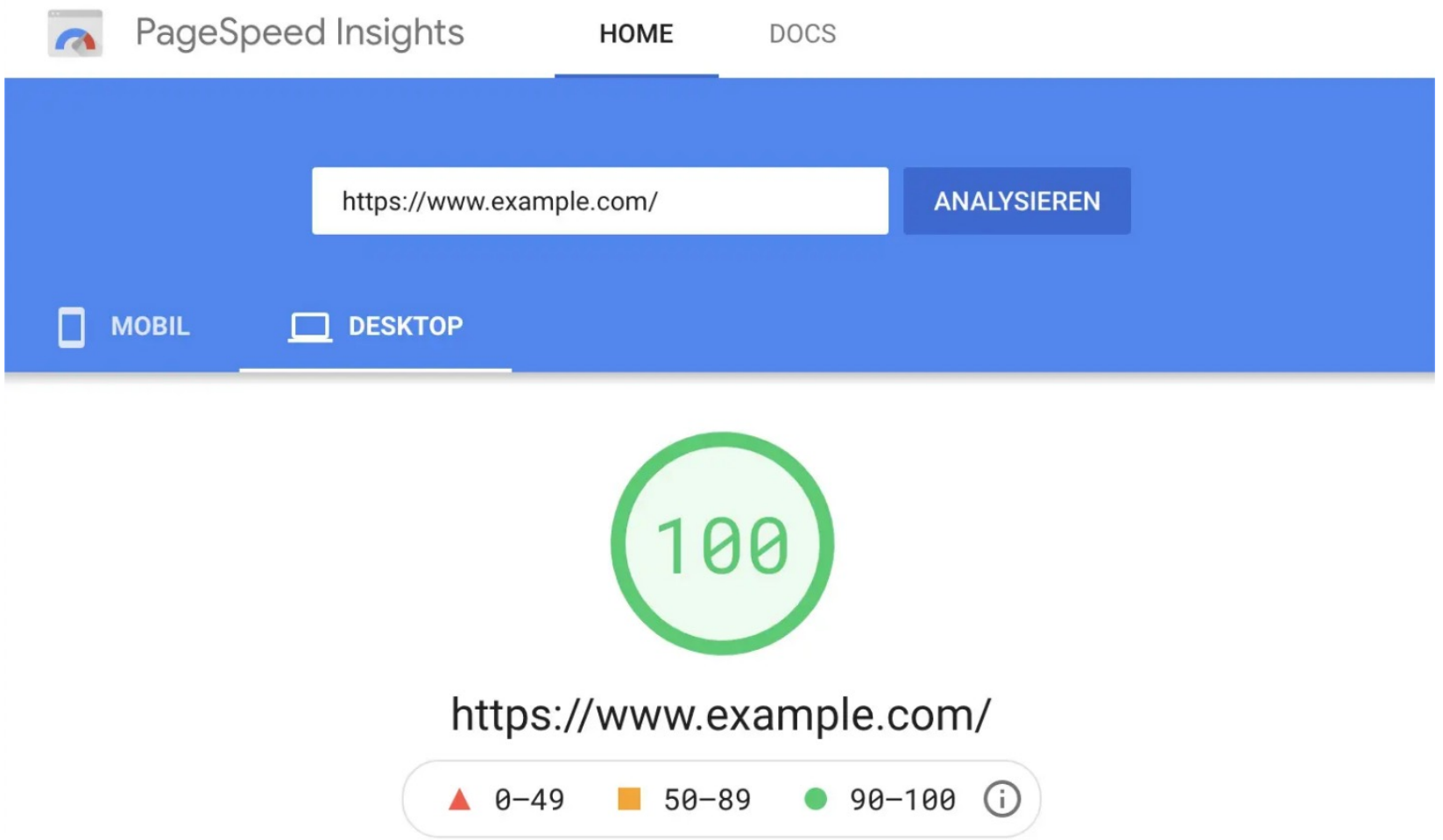
Angular Initial Load Performance Optimization

Hosted by Alex Thalhammer

Outline

1. Use web performance best practices
2. Use NgOptimizedImage (since NG 14.2.0)
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 & prerender or cache on the server
8. Use a URL cache

Web Performance Best Practices



#1: Use web performance best practices- I

Problems:

- *Images not optimized*
- *Images not properly sized*
- *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.org or
- Chrome DevTools

#1: Use web performance best practices - III

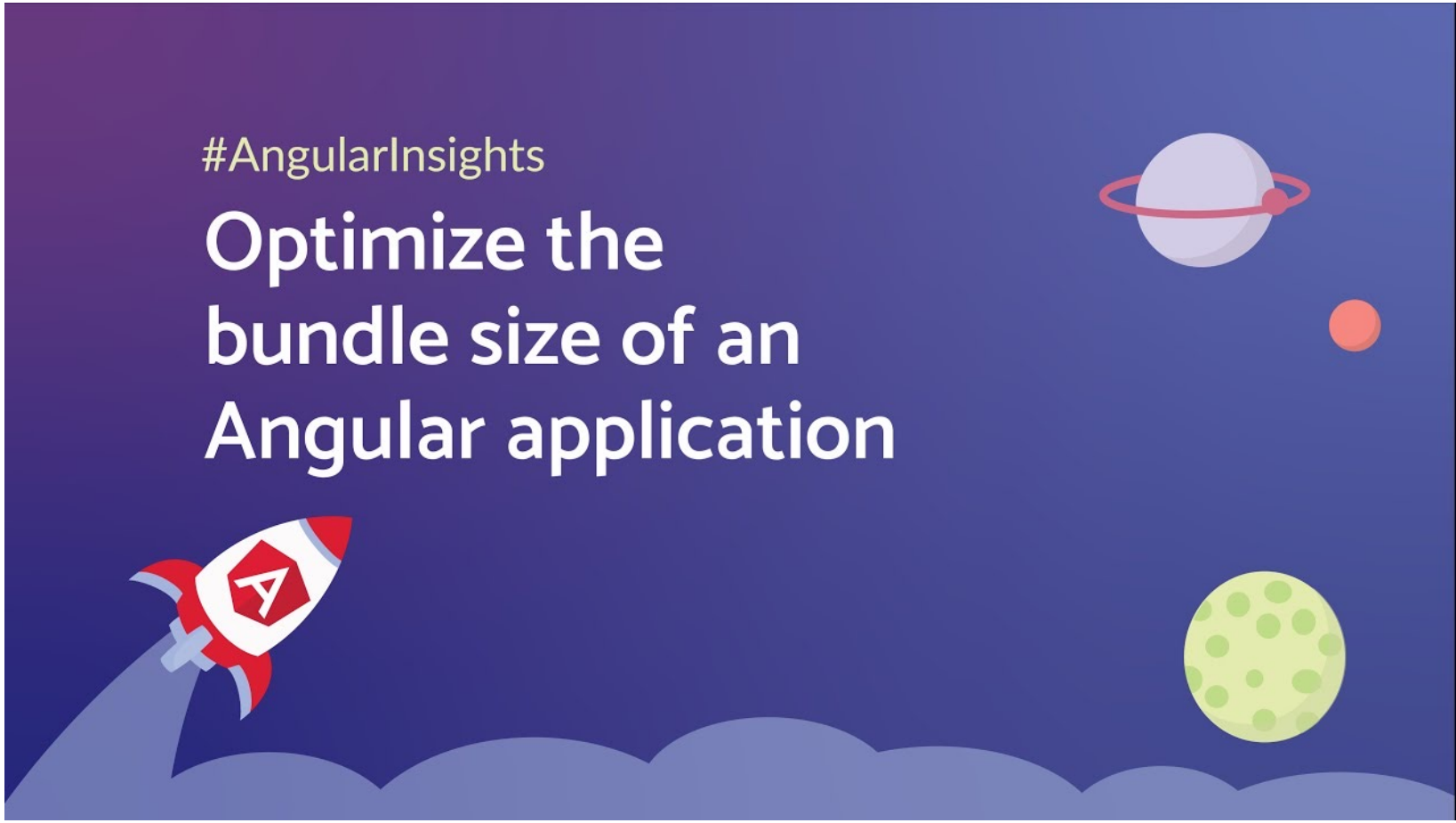
Solutions:

- *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*
- *…*

#2: 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")

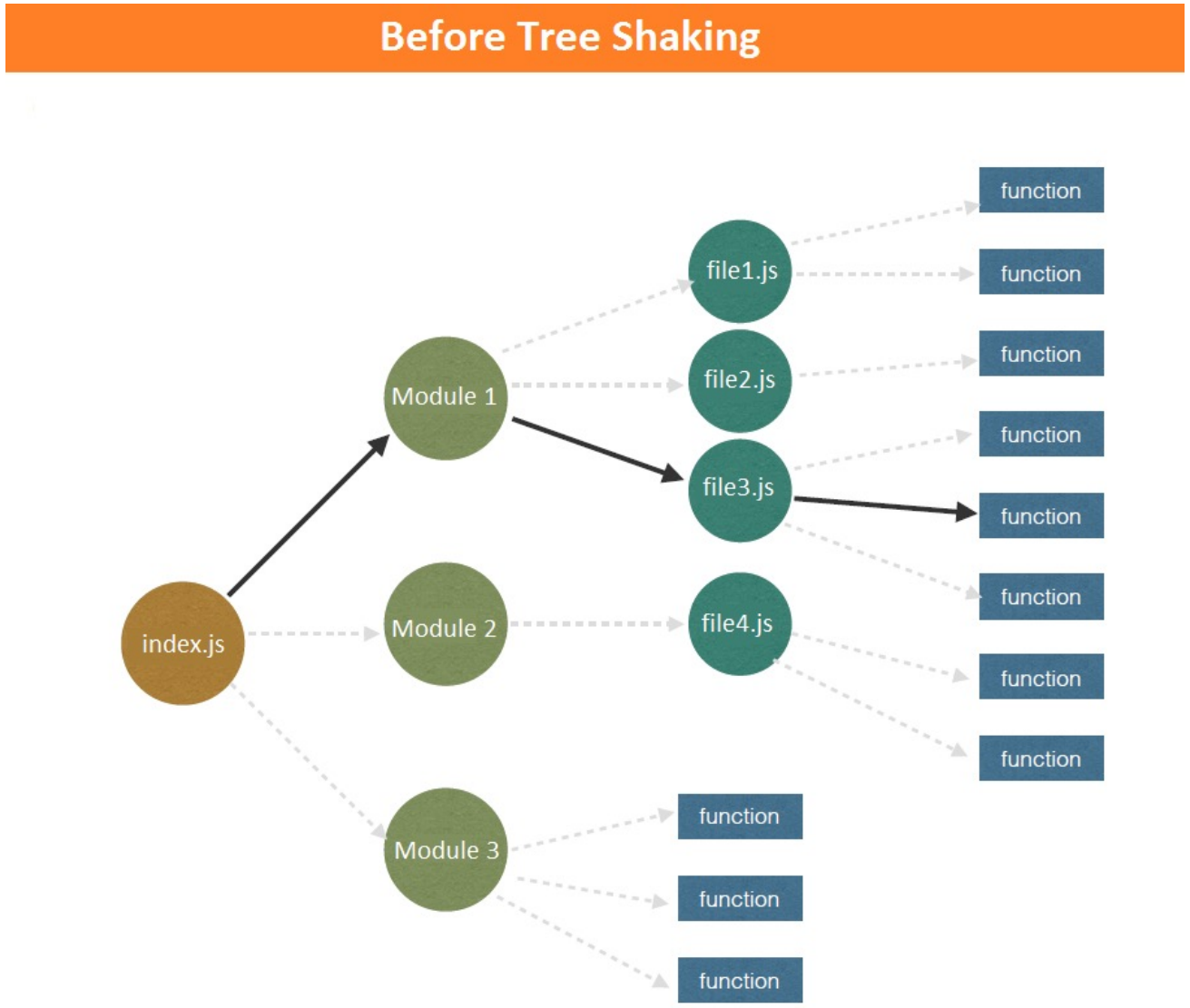
Build optimization



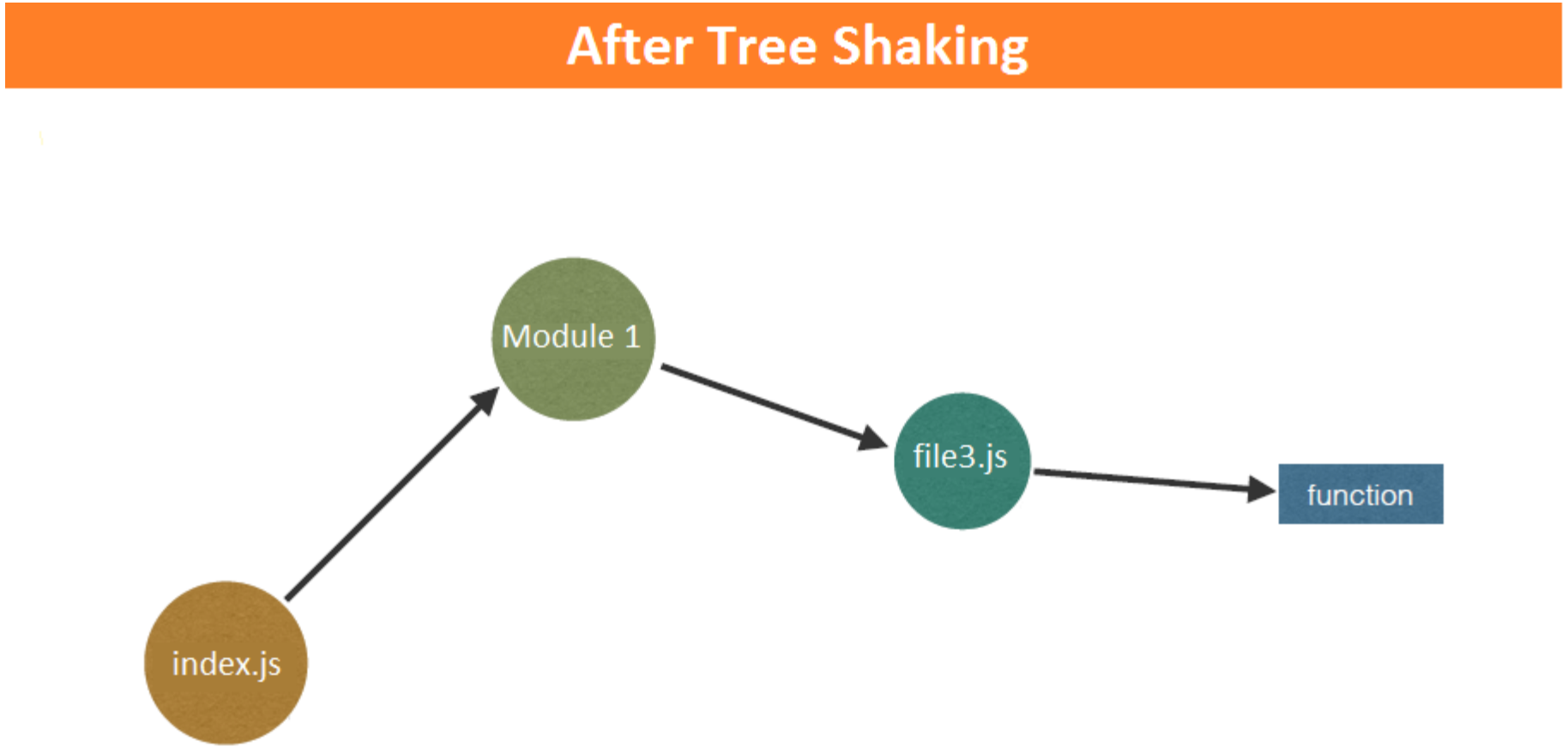
#3: Advantages of Angular Ivy (since V9)

- 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

#3: Tree Shaking



#3: Tree Shaking



#3: Use Build Optimization – I

Problem:

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

Identify:

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

#3: Use Build Optimization – II

Solution:

- Use production build
  - ng b(uild) (--c production)

- Set up angular.json correctly

```
"production": {
  "buildOptimizer": true,
  "optimization": true,
  "vendorChunk": true
}
```

DEMO – Build Configuration

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

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

- Problem: *Importing large 3rd party libraries that are not treeshakable*
  - *moment*
  - *lodash*
  - *charts*
  - …
- Identify: Source Map Analyzer or Webpack Bundle Analyzer
- Solution: Remove or replace that lib / framework
  - *moment → date-fns*
  - *lodash → lodash-es*
  - …

DEMO – Large Libs

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Lab

Initial Load Performance

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

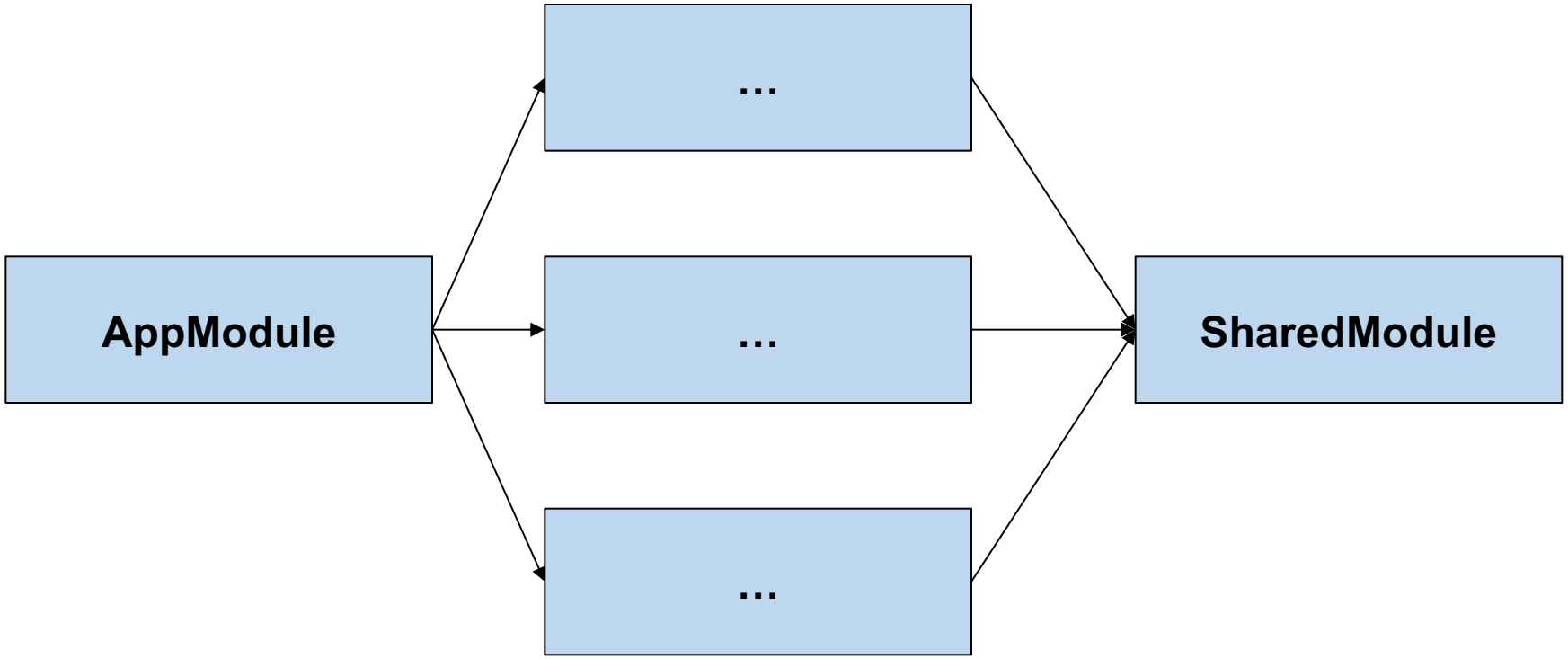
SOFTWARE ARCHITECT

Lazy Loading

#5: Angular Lazy Loading

- Lazy Loading means: Load it later, after startup
- Better initial load performance
- But: Delay during execution for loading on demand

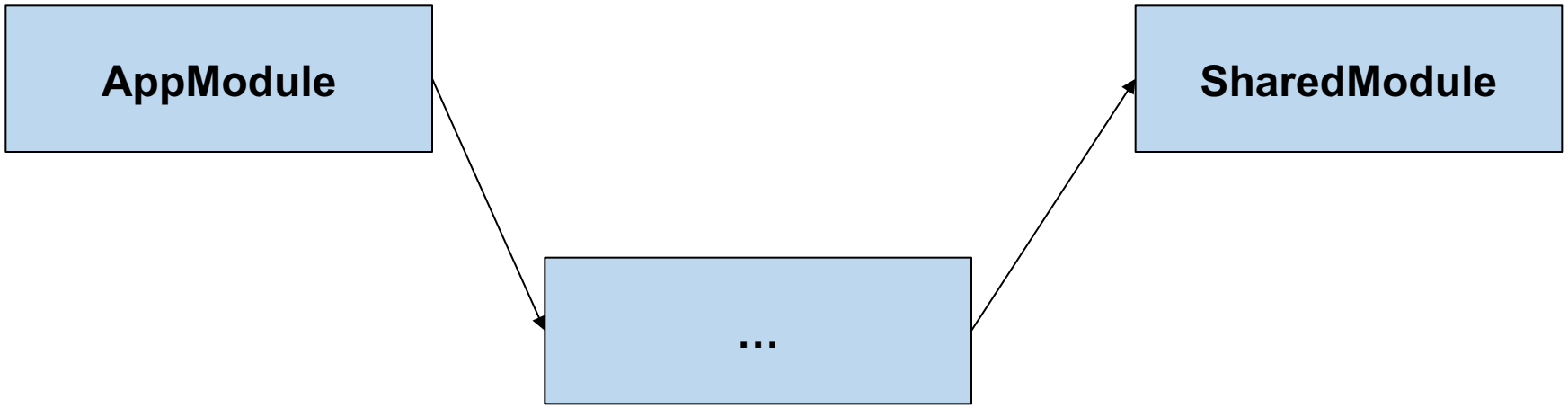
#5: Angular Module Structure



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

ANGULAR ARCHITECTS INSIDE KNOWLEDGE
SOFTWARE ARCHITECT

#5: Angular Lazy Loading



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

ANGULAR ARCHITECTS INSIDE KNOWLEDGE
SOFTWARE ARCHITECT

#5: AppModule Routes 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)
    }
];
```

#5: 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

#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(
        ROUTE_CONFIG,
        { preloadingStrategy: 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
- Import Graph Visualizer

#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

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

#5.2 Lazy Loading without Standalone Comps

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

DEMO –Lazy Loading Standalone

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

DEMO – Dynamic Lazy Loading

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

#5.5 Lazy Loading without the router

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

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

Lab

Lazy Loading

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

#6: Critical Rendering Path / Above the fold

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

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

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

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE ARCHITECT

Server-side rendering

#7: 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

#7: Server-side rendering (Angular 16)

- New feature called *“non destructive hydration”*

#7.5: 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

#8: Use a (URL) cache

Alternative Solution:

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

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 & prerender or cache on the server**
8. Use a URL cache

References

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

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