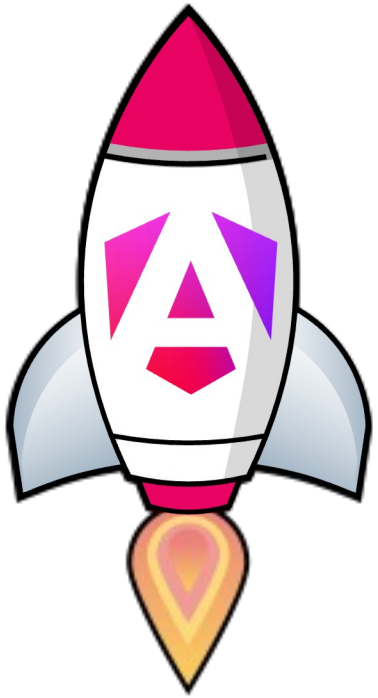
ANGULAR
ARCHITECTS

Initial Load
Lazy Loading

Alexander Thalhammer | @LX_T

Outline - Initial Load Performance



- Assets & Build
- Lazy Loading & Deferrable Views
- SSR, SSG & Incremental Hydration

Assets & Build

- Use web performance best practices
- Use **NgOptimizedImage** (since NG 14.2.0)
- Use build optimization & Tree Shaking
- Try to avoid large 3rd party deps
  - like CSS or component frameworks

Agenda

Web Performance – Issues & Solutions

- *Slow server infrastructure → HTTP/3 not HTTP/1.1, CDN*
- *Browser Caching not configured correctly → Configure it*
- *Compression not configured correctly → Brotli or Gzip*
- *Images not optimized → Use .webp, .avif or .svg*
- *Images not properly sized → Use srcsets*
- *Unused JS code or CSS styles → Clean up & lazy load assets*
- *Too large assets, many assets → Clean up & lazy load assets*
- …

NgOptimizedImage

lazyLoading / defer

NgOptimizedImage

ANGULAR
ARCHITECTS

Use NgOptimizedImage (since NG 14.2.0)

- Problem: Lighthouse or PageSpeed image errors / warnings
- Identify: Lighthouse & PageSpeed / WebPageTest or DevTools
- Solution: Use NgOptimizedImage's **ngSrc** instead of **src** attr.
  - Takes care of intelligent **lazy loading** of images outside viewport
  - **Prioritization** of critical images ("above-the-fold")
  - Together with an **image provider** it creates **.webp** format for us ☺
  - Also creates **srcset & sizes attributes** (for responsive sizes, since NG 15)
    - Also supports high-res devices ("Retina images")

Demo

ngSrc

ANGULAR
ARCHITECTS

Build optimization

Use Build Optimization – Solution

- Use production build
  - *ng b(uild) (--c production)*
- Set up angular.json correctly
- New builder in NG17

```
"builder": "@angular-devkit/build-angular:application",
[…]
"production": {
  "optimization": true
},
```

```
"@angular-devkit/build-angular:browser",

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

Demo

NG Build Configuration

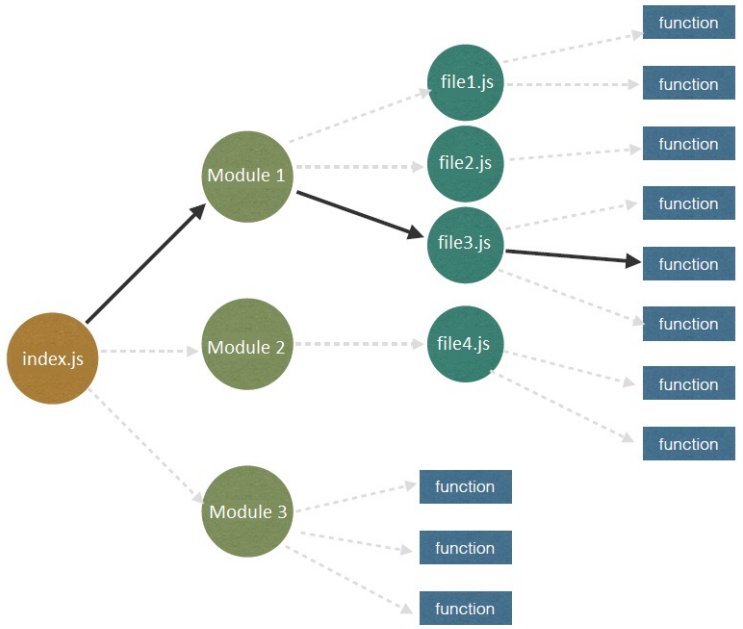
ANGULAR
ARCHITECTS

Advantages of Angular Ivy (since V9)

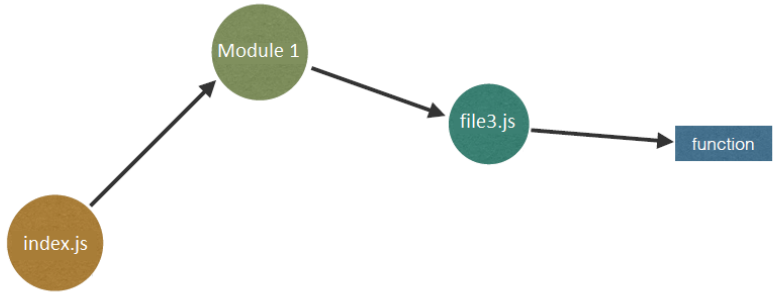
- Angular ***ViewEngine*** itself was not tree-shakable
- Default since NG 10, for libs default since NG 12
- AOT per default → You don't need to include the compiler!
- ***Ivy*** also does a lot of under the hood optimization
- Tools can easier analyse the code
  - Remove unneeded parts of frameworks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

Tree Shaking

Before Tree Shaking



After Tree Shaking



Assets & Build

- Use web performance best practices
- Use **NgOptimizedImage** (since NG 14.2.0)
- Use build optimization & Tree Shaking
- Avoid large 3rd party deps / CSS frameworks

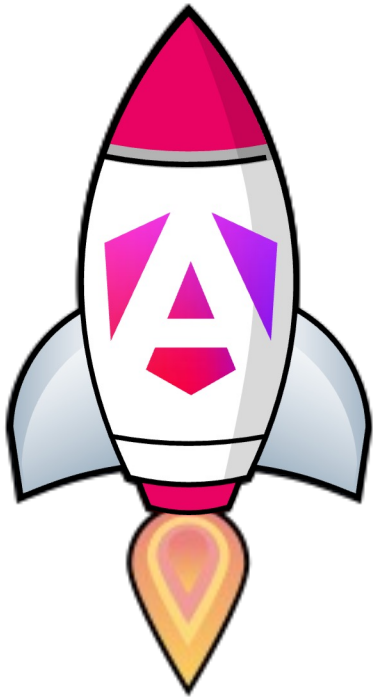
Recap

References

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

- Angular Docs
  - NgOptimizedImage
  - NG Build

Outline - Initial Load Performance



- Assets & Build
- Lazy Loading & Deferrable Views
- SSR, SSG & Incremental Hydration

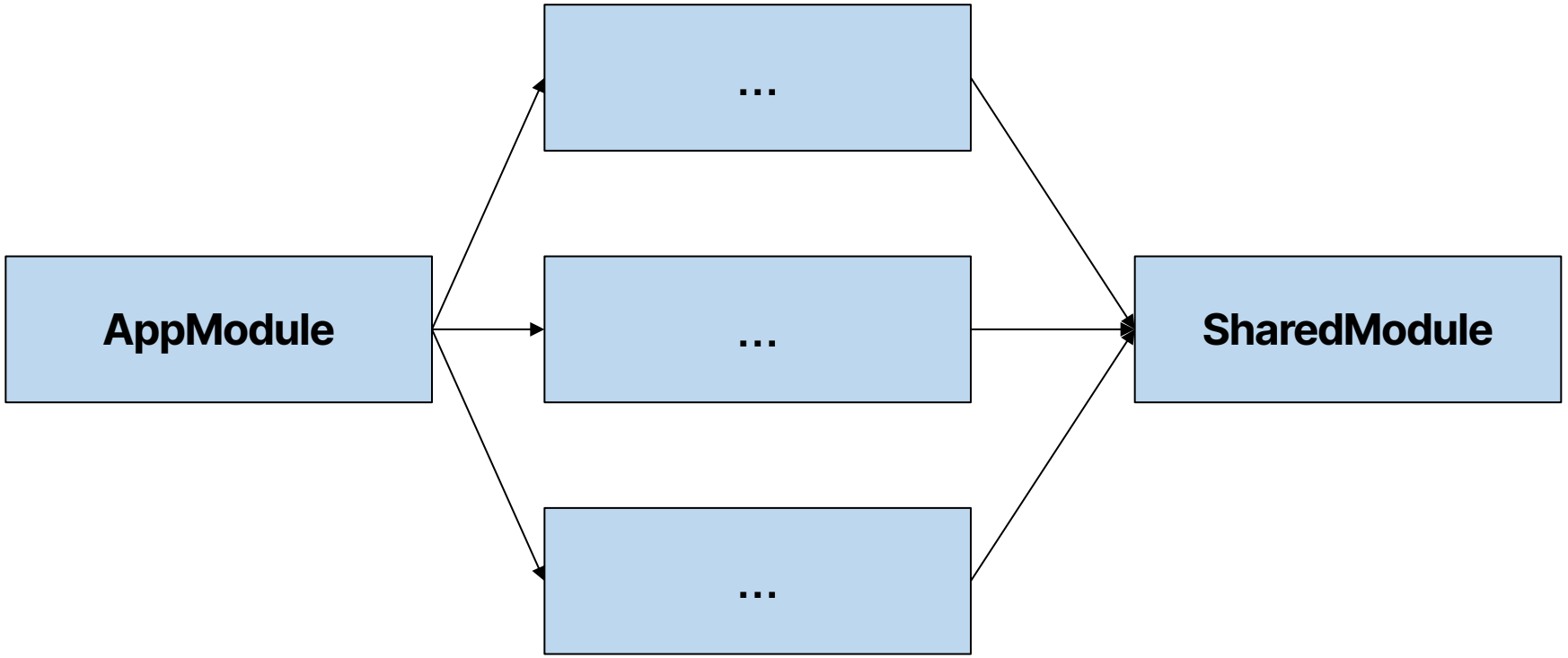
Lazy Loading

Lazy Loading & Deferring

- Lazy Loading via modules / features
  - 2 most common pitfalls and their solutions
- Lazy Loading via standalone components
- Preloading
  - PreloadAllModules
  - Other strategies
- ~~Lazy Loading without the router (a bit complicated)~~
- ~~Lazy Loading below the fold (very, very complicated)~~
- Deferring (brand new in NG 17, very lean, replaces last 2)

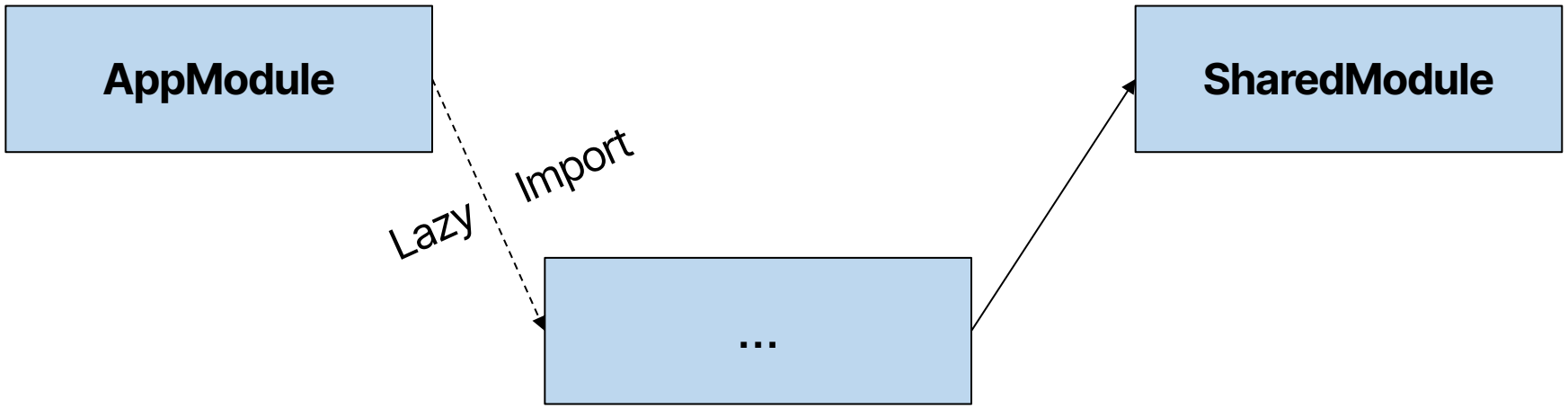
Agenda

Angular Module | Feature Structure



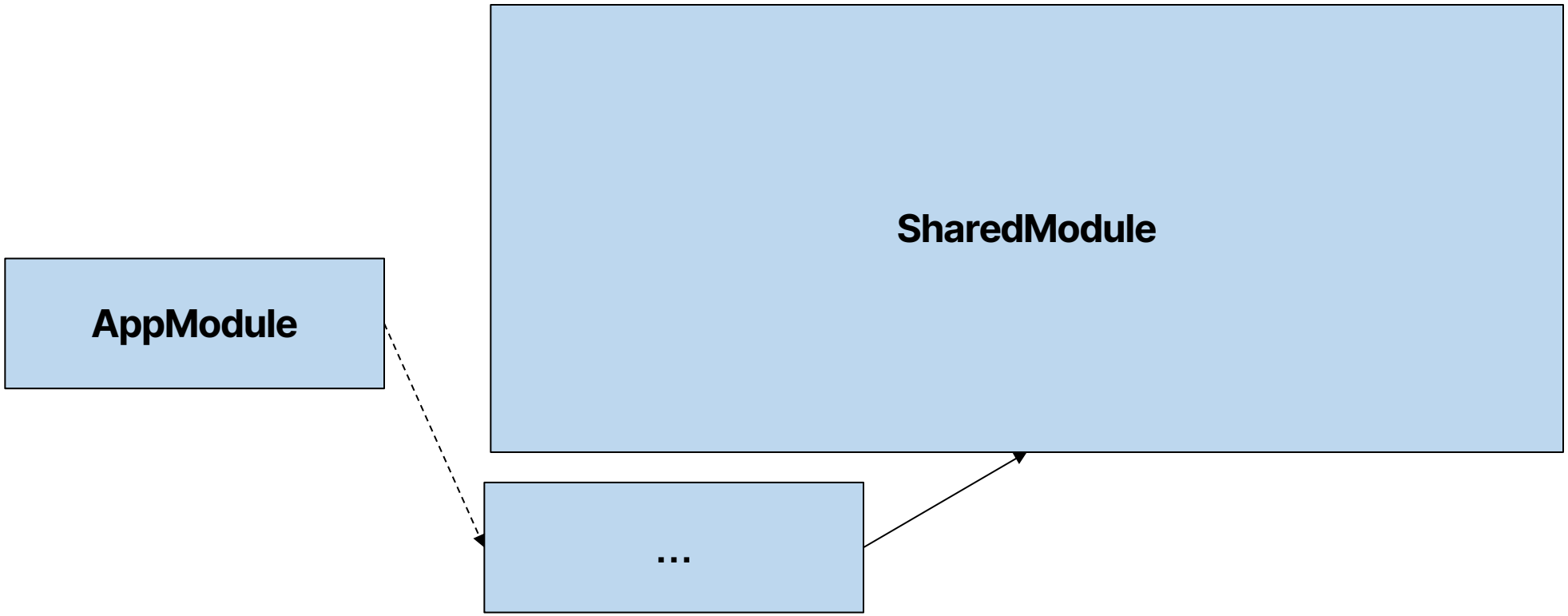
Root Module **Feature Modules** **Shared Module**

Angular Lazy Loading – Theory



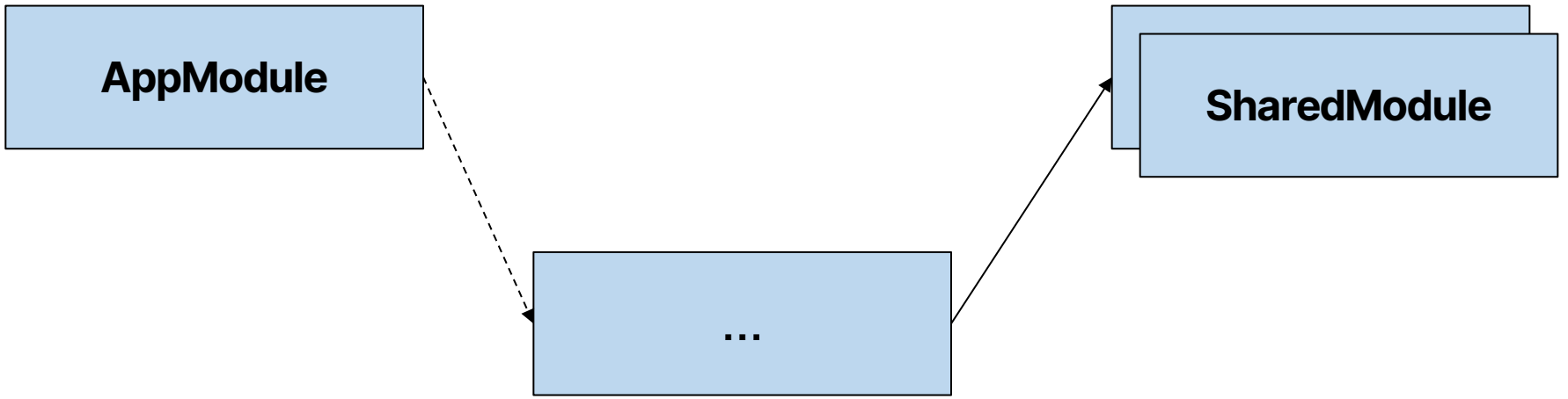
Root Module **Feature Modules** **Shared Module**

Angular Lazy Loading – Common Pitfall



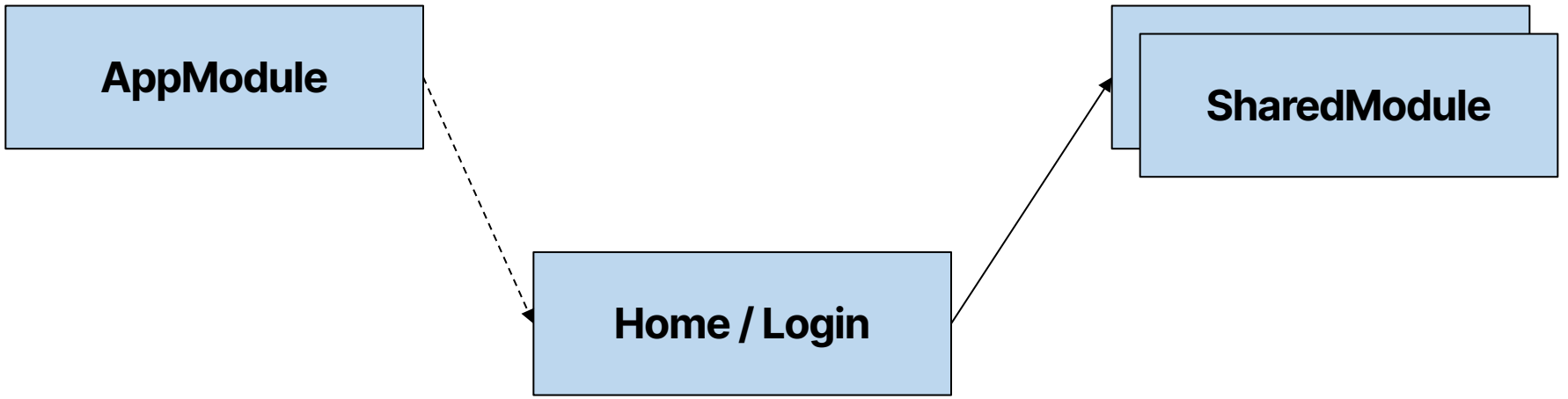
Root Module **Feature Modules** **Huge Shared Module**

Angular Lazy Loading - Solution

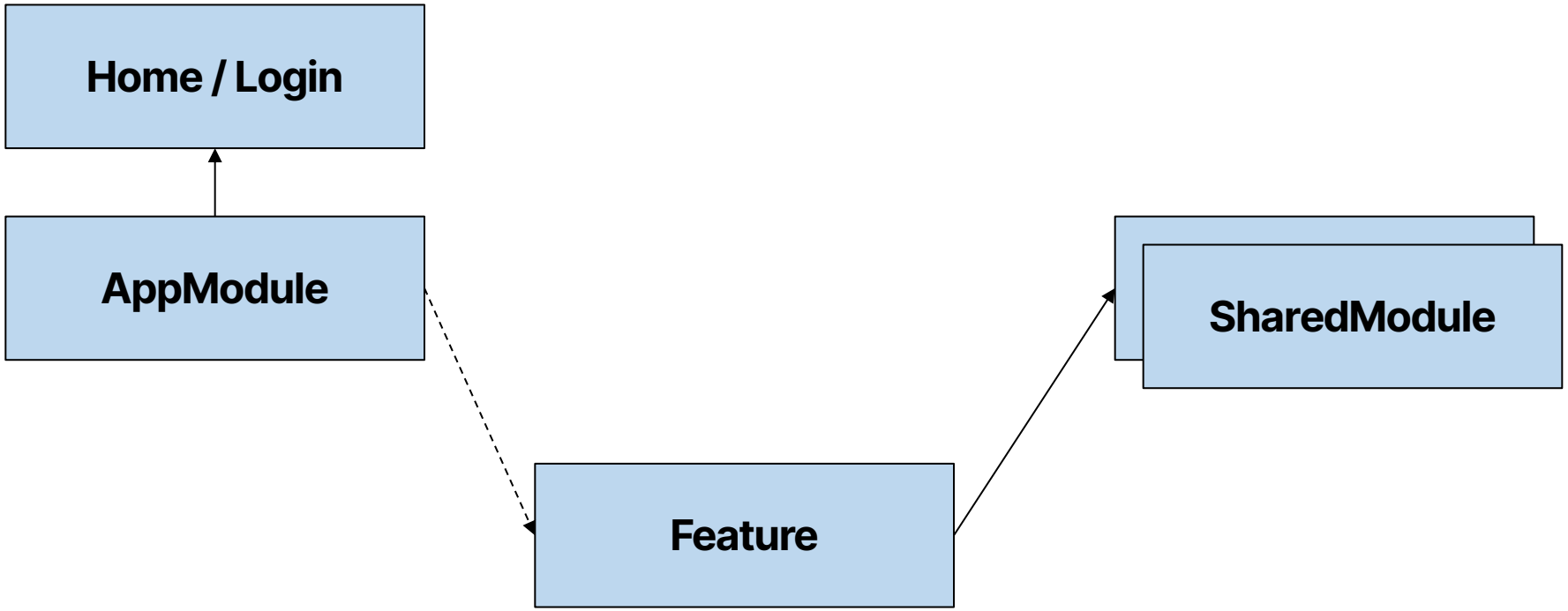


Root Module **Feature Modules** **Small Shared Modules**

Angular Lazy Loading – Another Pitfall



Angular Lazy Loading – Solution



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; // for standalone
```

flights/flight-search

Triggers Lazy Loading w/ loadChildren

Demo

Lazy Loading NG Module

ANGULAR
ARCHITECTS

Lazy Loading with standalone components

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

Demo

Lazy Loading Standalone Component

ANGULAR
ARCHITECTS

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 / more lazy loaded modules/comps, it will be put in a common chunk
- When used by an eagerly loaded part, it will be put into main bundle

Demo

Lazy Loading Services

Lazy Loading

– Lazy Loading means: Load it later, after startup

– Better initial load performance

– But: Delay during runtime for loading on demand

Preloading

Preloading

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

– When module / component is needed it is available immediately

Activate Preloading (in AppModule)

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

Activate Preloading (in app.config.ts)

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

Demo

Lazy Loading
Preloading

ANGULAR
ARCHITECTS

Intelligent Preloading with ngx-quicklink

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

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

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

Demo

Lazy Loading Quicklink

ANGULAR
ARCHITECTS

Or CustomPreloadingStrategy

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

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
- But don't lazyload the initial feature, because it will be delayed ;-)
- Don't destroy lazy loading by (eagerly) loading huge shared module

Deferrable Views

ANGULAR
ARCHITECTS

Deferrable Views

– used via view templates control flow syntax "**@defer**"

– when router-based is not possible

– or to defer heavy components (3rd party)

Deferrable Views - syntax

```
…
@defer (on viewport) {
  <aa-lazy-component />
} @placeholder {
  <p>Component is loading on viewport.</p>
}
…
```

Deferrable Views - on

- on immediate (default)
- on idle
- on viewport
- on hover
- on interaction
- on timer(4200ms)

Deferrable Views - when

– specifies an imperative condition as an expression that returns a bool
  – best used: boolean flag

– if the condition returns to false, the swap is not reverted
  – it is a one-time operation

```
…
@defer (when condition) {
  <aa-lazy-component />
}
…
```

Deferrable Views - prefetch

– allows to specify conditions **when prefetching** of the dependencies should be triggered

– similar to the router-based **PreloadingStrategy**

```
...
@defer (on viewport; prefetch on idle) {
 <aa-lazy-component />
}
 ...
```

Deferrable Views - extras

- @placeholder
- @loading
- @error

```
…
@defer (on viewport; prefetch on idle) {
  <aa-lazy-component />
} @placeholder (minimum 500ms) {
  <img width="420" height="420" alt="lazy component placeholder"
src="placeholder.avif" />
} @loading (after 500ms; minimum 1s) {
  <img width="420" height="420" alt="lazy is loading spinner"
src="spinner.avif" />
} @error {
  <p>Why do I exist?</p>
}
…
```

Demo

Lazy Loading Deferrable Views

ANGULAR
ARCHITECTS

Lazy Loading & Deferring

- **Lazy Loading via modules / features**
  - 2 most common pitfalls and their solutions
- **Lazy Loading via standalone components**
- **Preloading**
  - PreloadAllModules
  - or QuicklinkStrategy with ngx-quicklink
- **Deferring** (brand new in NG 17, very lean)

Recap

References

- Angular Docs
  - Lazy-loading feature modules
- Angular Architects Blog
  - Deferrable Views (Blog post)

ANGULAR
ARCHITECTS

Lab 02 Initial Load – Lazy Loading

NgOptimizedImage / Lazy Loading / Deferrable Views