ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Initial Load Performance Lazy Loading & Deferrable Views

Alex Thalhammer

Outline 02 - Initial Load Performance

- Assets & Build
- Lazy Loading & Deferrable Views
- SSR & SSG

Outline 02b - Lazy Loading & Defering

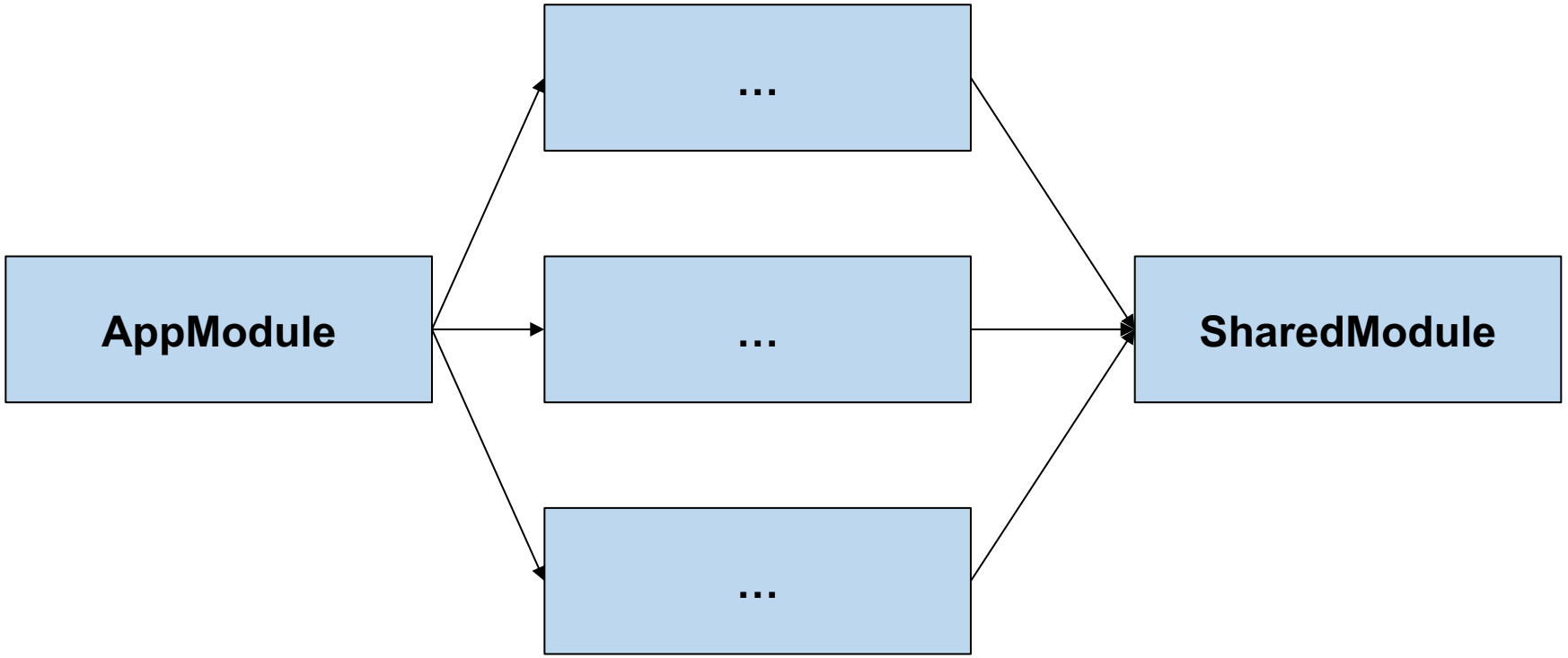
- Lazy Loading via modules / features
  - 2 most common pitfalls and their solutions
- Lazy Loading via standalone components
- Preloading
  - PreloadAllModules
  - Other strategies
- Lazy Loading without the router (a bit complicated)
- Lazy Loading below the fold (very, very complicated)
- Deferring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

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE ARCHITECT

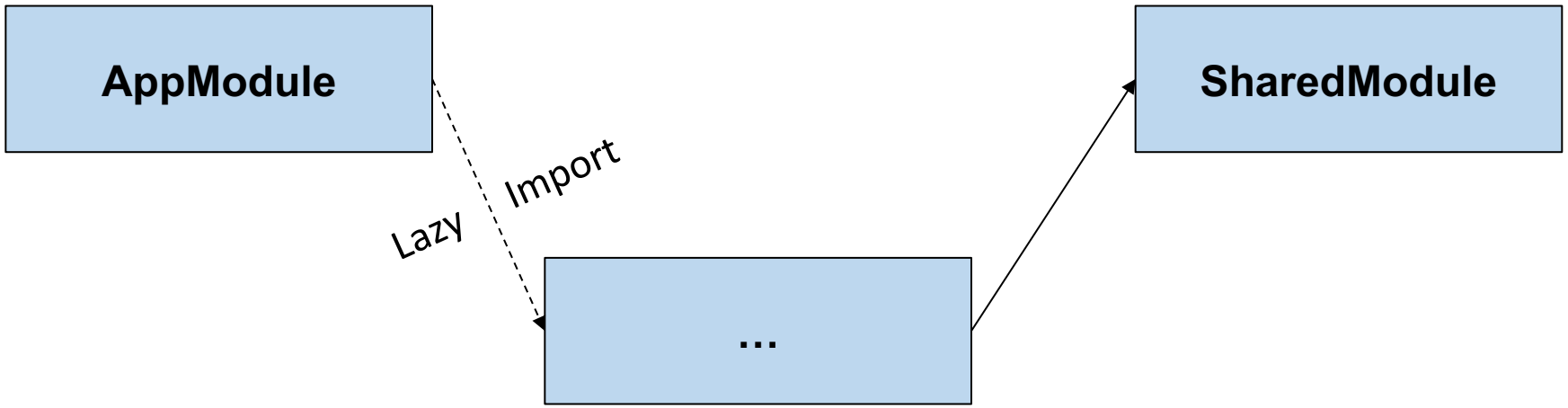
Lazy Loading

Angular Module | Feature Structure



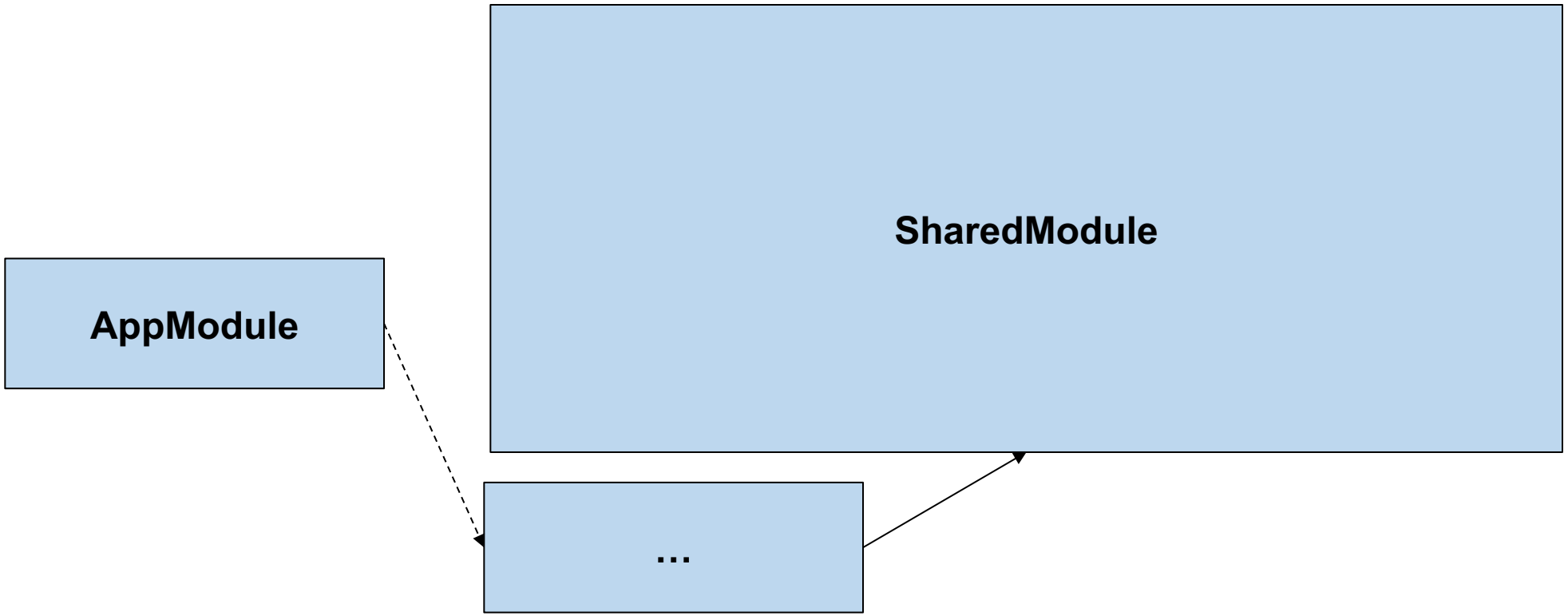
Root Module **Feature Modules** **Shared Module**

Angular Lazy Loading – Theory



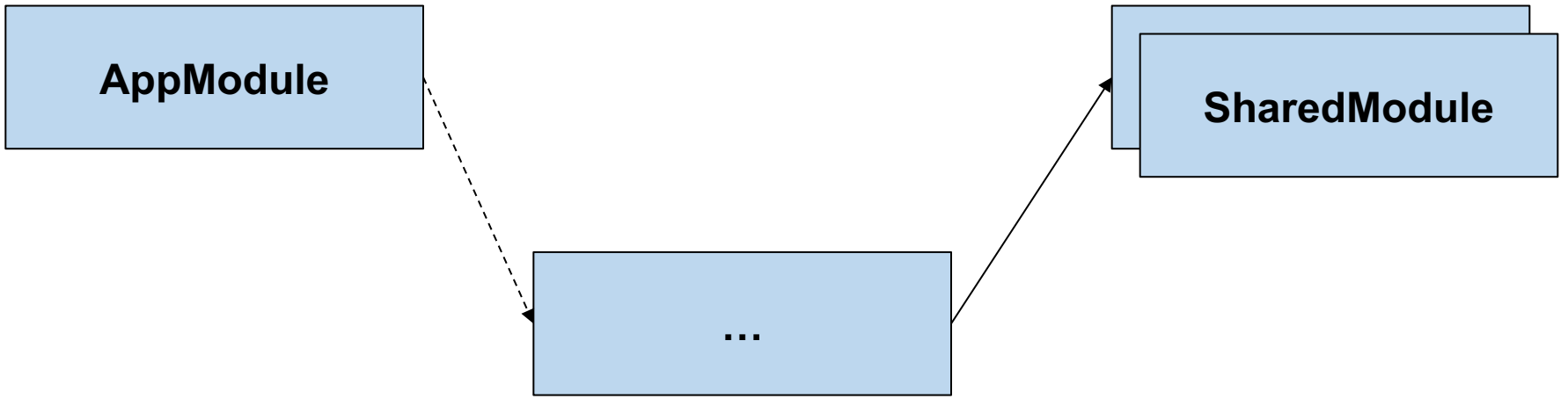
Root Module **Feature Modules** **Shared Module**

Angular Lazy Loading – Common Pitfall



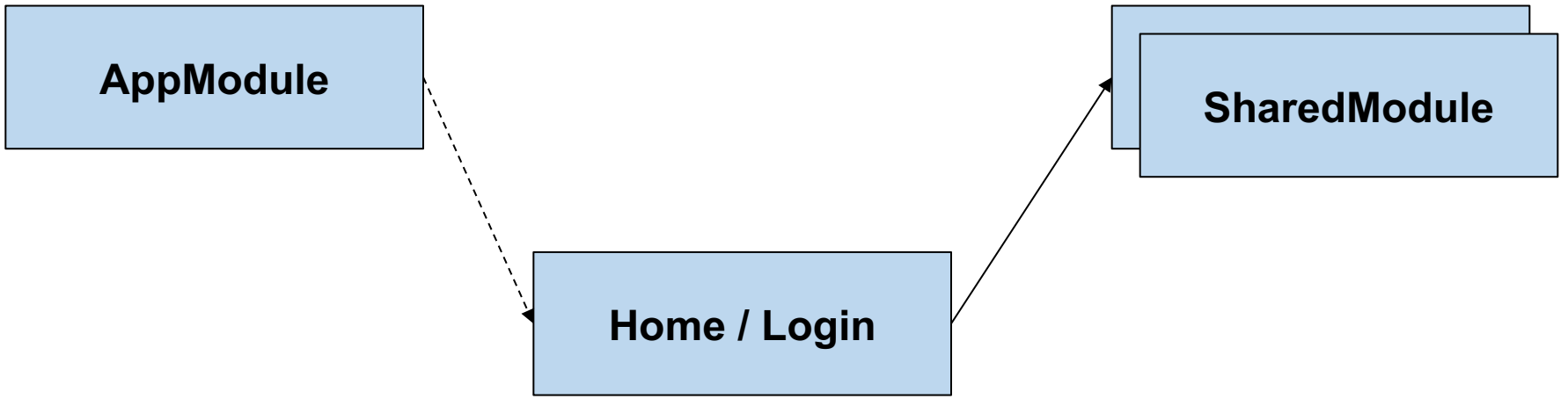
Root Module **Feature Modules** **Huge Shared Module**

Angular Lazy Loading - Solution



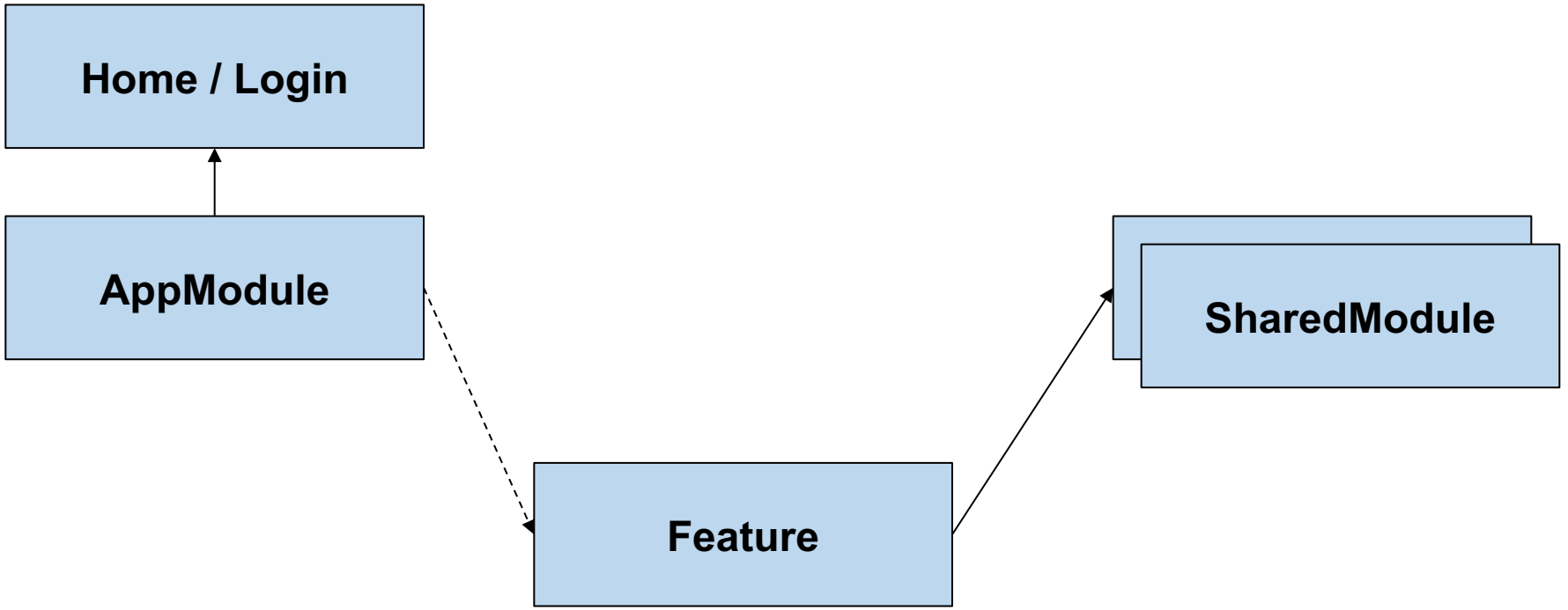
Root Module **Feature Modules** **Small Shared Modules**

Angular Lazy Loading – Another Pitfall



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

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')
                            .then((m) => m.flightsRoutes)
    }
];
```

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

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

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Lazy Loading

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

Preloading

Preloading

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

- When module 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 – Preloading

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

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

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

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

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 – Ngx Quicklink

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Or CustomPreloadingStrategy

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

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

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

Lab

Lazy Loading

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Deferrable Views

- Problem: Lazy Loading without the router and especially Lazy Loading below the fold is rather complicated and inconvenient
- NG17 has the solution in the new template syntax control flow:
- It's called: **Deferrable Views**

Deferrable Views - syntax

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

Deferrable Views - on

- on immediate (default)
- 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**

```
…
@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>
}
…
```

Recap

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

References

- Angular Docs
  - Lazy-loading feature modules

- Angular Architects Blog
  - Deferrable Views (Blog post)